Freescale Semiconductor
Application Note

AN2867
Rev. 0.1, 10/2004

Using the DSPI Module on the MPC5500 Family

by: Steven McQuade
TECD Applications

The MPC55xx family of microcontrollers provides several options for serial communication. The range of on-chip communication modules include the eSCI (Enhanced Serial Communication Interface), FlexCAN (Flexible Controller Area Network), and DSPI (Deserial/Serial Peripheral Interface).

This application note describes how the MPC55xx family implements SPI communication, as well as how to use the serialization/deserialization functions of the DSPI module.

The DSPI module supports serialization/deserialization of internal signals, such as the eTPU, eMIOS, and SIU interrupt signals. The purpose of serialization/deserialization is to allow functional pins to be used as GPIO.

Example code that configures the DSPI module for the modes mentioned above is included for reference.

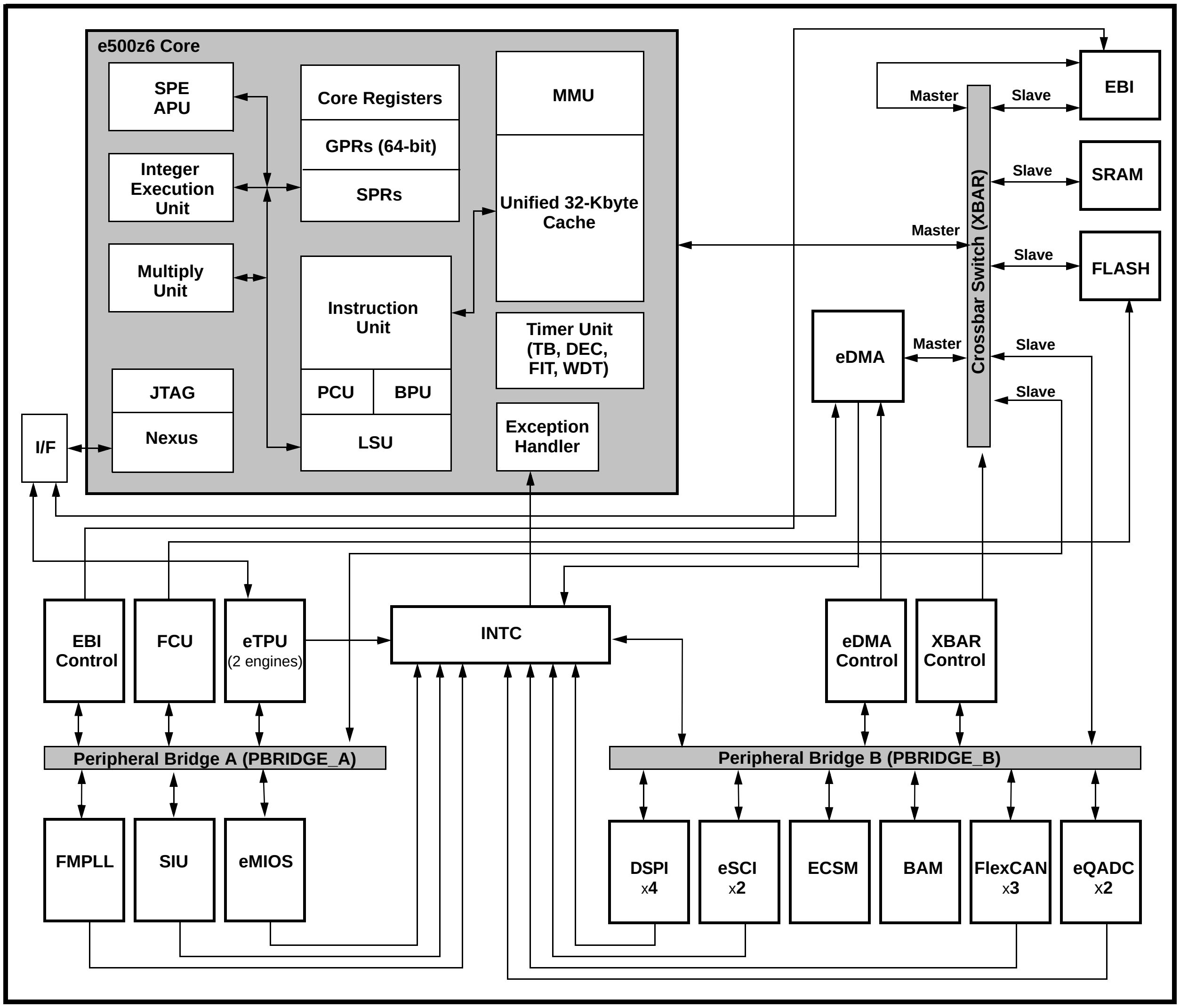
Table of Contents

1 Introduction

Figure 1 shows the block diagram of the MPC5554 device. The DSPI modules are located on Peripheral Bridge B.

All DSPI modules provide SPI (Serial Peripheral Interface), DSI (Deserial / Serial Interface), and CSI (Combined Serial Interface) functionality.

The SPI is a high speed (up to system frequency/4), full-duplex, four-wire synchronous interface. It is used to communicate to external devices such as sensors, voltage regulators, or A/D converters. To accommodate SPI transmit and receive queues, an efficient solution is to use the on-chip eDMA and SRAM modules. Alternatively, external RAM can be used.



LEGEND

APU	– Auxiliary processing unit
BAM	– Boot assist module
BIU	– Bus interface unit
BPU	– Branch processing unit
CAN	– Controller area network (FlexCAN)
DEC	– Decrementer
DSPI	– Deserial/serial peripheral interface
EBI	– External bus interface
ECSM	– Error correction status module
eDMA	– Enhanced direct memory access
eMIOS	– Enhanced modular I/O system
eQADC	– Enhanced queued analog/digital converter
eSCI	– Enhanced serial communications interface
eTPU	– Enhanced time processing unit
FIT	– Fixed interval timer
FCU	– Flash control unit
FMPLL	– Frequency modulated phase-locked loop
GPR	– General-purpose register
INTC	– Interrupt controller
I/F	– Interface
LSU	– Load/store unit
MMU	– Memory management unit
PCU	– Program counter unit
SIU	– System integration unit
SPE	– Signal processing engine
SPR	– Special purpose register
SRAM	– Static RAM
TB	– Time base
WDT	– Watchdog timer
XBAR	– Crossbar switch

Figure 1. MPC5554 Block Diagram

1.1 SPI

The SPI-bus is a 4-wire, full-duplex serial communications interface used by many microprocessor peripheral chips. The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that is standard across many microcontrollers and other peripheral chips. It provides support for a low/medium bandwidth network connection amongs MCUs and other devices supporting the SPI.

A synchronous clock shifts serial data into and out of a device in a configurable bit format. On the MPC55xx family, the frame size can be selected to be from 4–16 bits. However, when serial chaining of DSPI modules is selected, the number of bits to be transferred can be up to 64.

The SPI bus is a master/slave interface. Whenever two devices communicate, one is referred to as the "master," and the other as the "slave" device. The master drives the serial clock. When using SPI, data is simultaneously transmitted and received, making it a full-duplexed protocol.

The SPI signals on the MPC55xx family are as follows:

- SCK for serial clock, which is always driven by the master
- SOUT is the data out signal
- SIN is the serial data in signal
- PCS or peripheral chip select, used to enable the slave device

Figure 2 illustrates an 8-bit SPI transfer. Detailed in this figure are the different clock polarity options, i.e. the inactive state of the clock is LOW when DSPIx_CTAR[CPOL] = 0, or HIGH when DSPIx_CTAR[CPOL] = 1.

Also shown in Figure 2 is the sample point of the master and slave. The sample point is defined by the DSPIx_CTAR[CPHA] bit field. When this bit is 0, data is captured on the leading edge of SCK and changed on the following edge. When DSPIx_CTAR[CPHA] = 1, data is changed on the leading edge of SCK and captured on the following edge. The configurable delays are also illustrated in this figure.

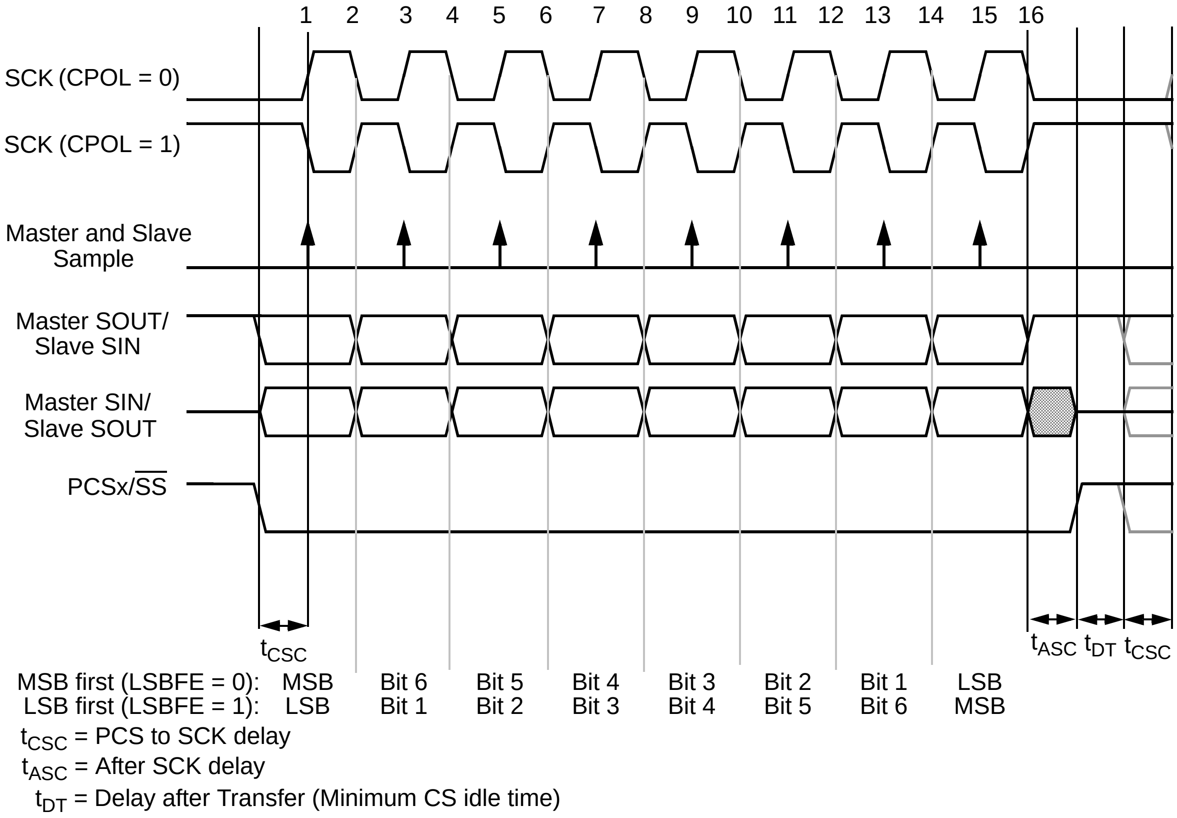

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
SCK (CPOL = 0)
SCK (CPOL = 1)
Master and Slave Sample
Master SOUT/ Slave SIN
Master SIN/ Slave SOUT
PCSx/$\overline{SS}$
t_{CSC}
t_{ASC} t_{DT} t_{CSC}
MSB first (LSBFE = 0): MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB first (LSBFE = 1): LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB
t_{CSC} = PCS to SCK delay
t_{ASC} = After SCK delay
t_{DT} = Delay after Transfer (Minimum CS idle time)

Figure 2. 8-Bit SPI Frame (CPHA = 0)

From Figure 3, it can be seen that when in SPI mode, the DSPI Tx FIFO (4-entry transmit FIFO containing commands and data) transmits the data out onto the SOUT pin. When data is received on the SIN pin, it is transferred to the Rx FIFO via the shift register. Refer to Section 2 for SPI configuration and initialization information.

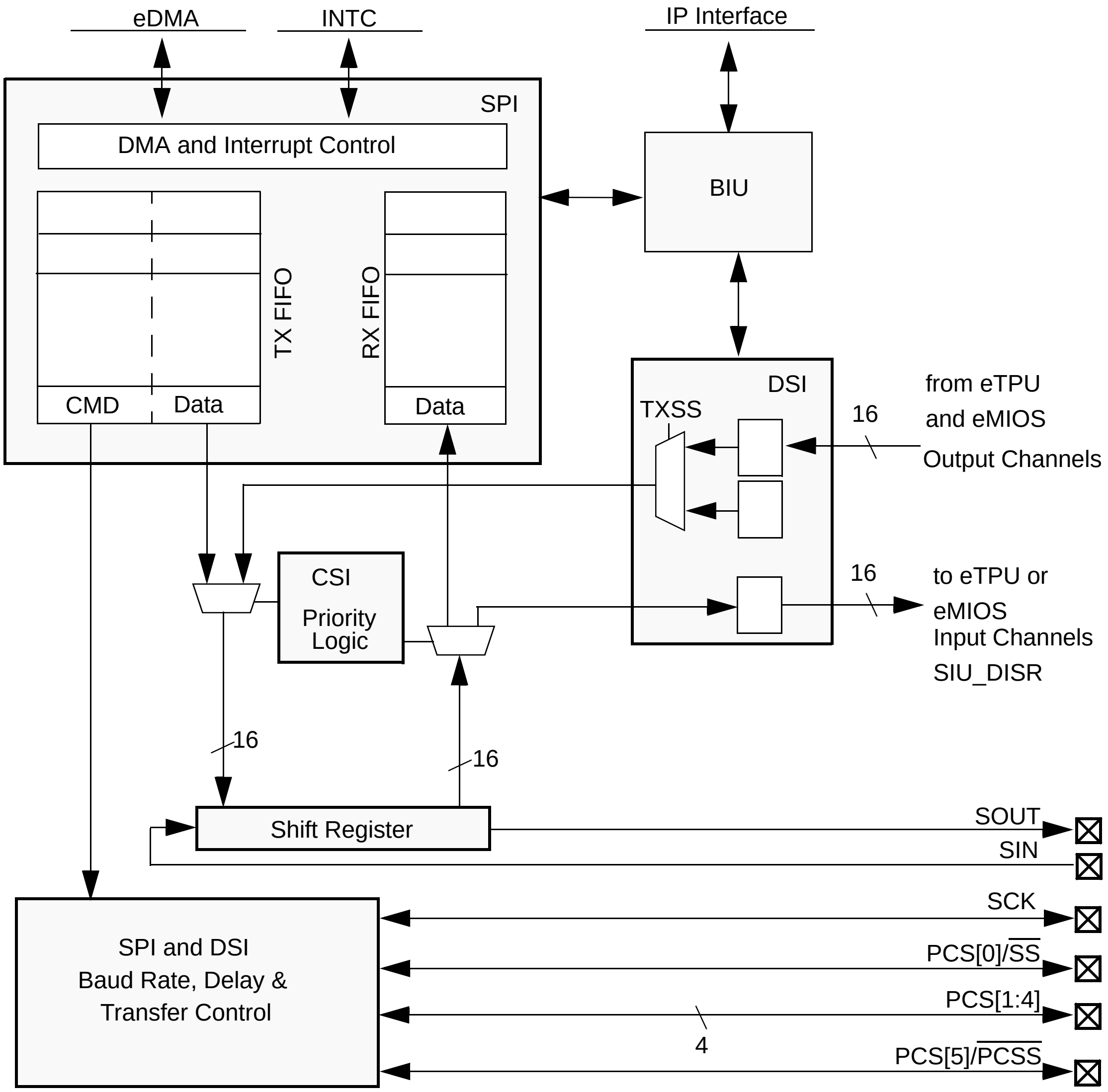


Figure 3. DSPI Block Diagram

1.2 DSI

When Deserial Serial Interface (DSI) functionality is selected, the DSPI module is configured to serialize eTPU and eMIOS output channels and deserialize received data by routing it to the eTPU, eMIOS, or internal SIU interrupt inputs.

When DSI master mode is selected, the data to be transmitted can be taken from 16 internal parallel inputs from the eTPU or eMIOS. The data can also be taken from an alternate serialization data register (DSPIx_ASDR). Selection is dependent on the value of the DSPIx_DSICR[TXSS] bit. Refer to Figure 4.

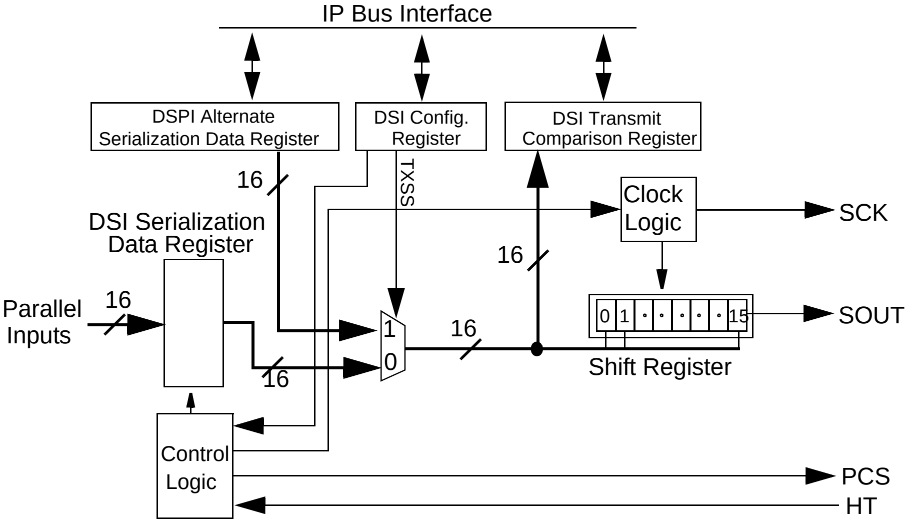


Figure 4. Serialization Block Diagram

When DSI slave mode is selected, the data received from the SPI bus is routed internally to the inputs of the eTPU, eMIOS, or SIU interrupts. The data to be routed internally is captured in the deserialization data register (DSPIx_DDR). Refer to Figure 5.

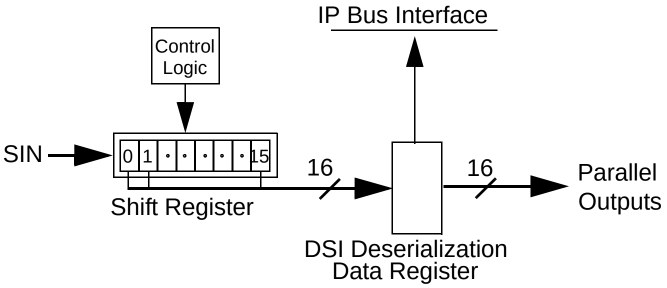


Figure 5. Deserialization Block Diagram

1.3 CSI

The Combined Serial Interface (CSI) mode allows the DSPI to operate in both SPI and DSI configuration,s interleaving DSI frames with SPI frames. When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, the DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. The user must configure the DSPI so that the two Clock and Transfer Attribute Registers (CTARs), associated with DSI data and SPI data, assert different peripheral chip select signals, denoted in Figure 5 as PCSx and PCSy. The DSI chip select is configured by writing the DSPIx_DSICR [DPCS, DSISTAS] bit fields accordingly. The SPI chip select is configured by writing the DSPIx_PUSHR[PCS,CTAS] bit fields accordingly.

NOTE

CSI mode is only supported in master mode.

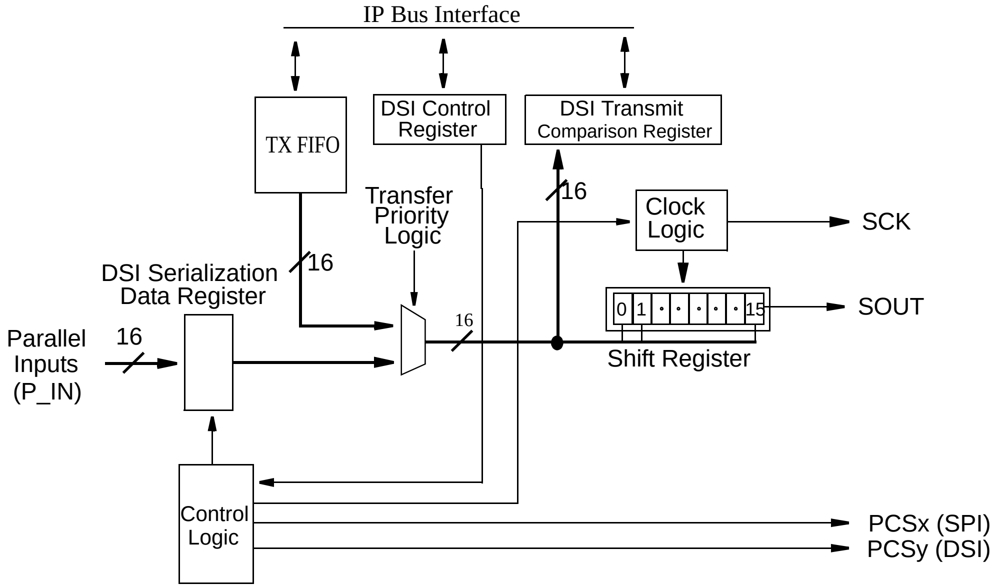


Figure 6. CSI Block Diagram

2 DSPI Module Configuration

2.1 Data Transmission

There are two ways to transmit SPI data:

1. Application code writes command and data information to the DSPIx_PUSHR register. This pushes a 32-bit word (16-bit command and 16-bit data) into the TxFIFO. The TxFIFO is 4 entries deep.
2. The other method is to configure the eDMA, such that it fetches the commands and data from system RAM (RAM is an example, any system memory including flash memory could be used) and writes these values to the DSPIx_PUSHR register; i.e. create a transmit queue. Refer to Appendix A for source code that performs this task.

Figure 7 details the more efficient case 2.

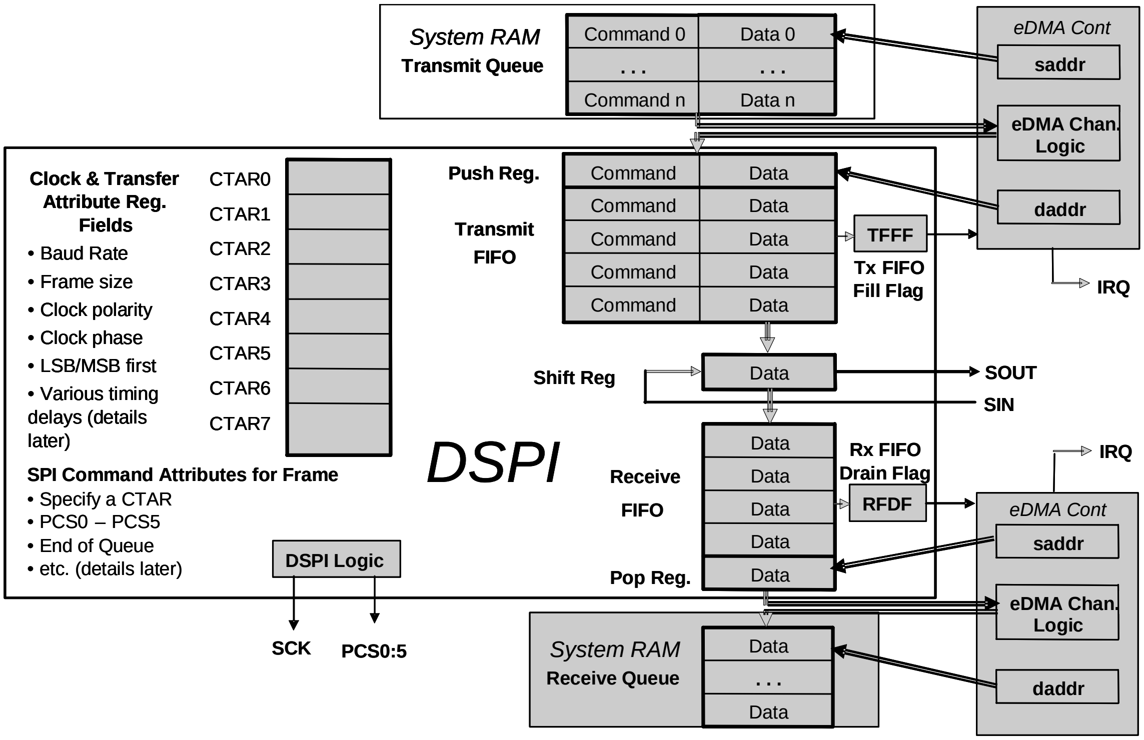


Figure 7. DSPI/eDMA Queue Creation

In this figure, DSPI*x*_SR[TFFF] Transmit FIFO Fill Flag is used to trigger an eDMA transfer or cause an interrupt (see Table 1). If a DMA transfer is selected, when TFFF = 1, the transfer is initiated. TFFF is only zeroed by the DSPI module when the TxFIFO buffer is full, i.e. when there are 4 entries in this buffer. TX FIFO entries can only be removed by shifting them out onto the SPI bus or by flushing the buffer. The TxFIFO can be flushed by writing 0b1 to DSPI*x*_MCR[CLR_TXF].

The equivalent scenario applies for the reception of SPI messages. For reception, the DSPI*x*_SR[RFDF] flag is used to trigger an eDMA transfer, moving the received data from a 4-entry RX FIFO to an area in system RAM. The draining of the RxFIFO is achieved by the associated eDMA channel reading the POP register (DSPI*x*_POPR). Refer to Appendix A for details of source code thta implements the above functionality.

Table 1. DSPI Interrupt/DMA Requests

Condition	Flag	Interrupt	DMA
End of Queue reached (EOQ)	EOQF	X	
TX FIFO is not Full	TFFF	X	X
Transfer of Current Frame Complete	TCF	X	
Attempt to Transmit with an Empty Transmit FIFO[1]	TFUF	X[1]	
RX FIFO is not Empty	RFDF	X	X
Attempt to Transmit with an Empty Transmit FIFO[1]	RFOF	X[1]	
FIFO Overrun[1]	TFUF or RFOF	X	

NOTES:
[1] The FIFO Overrun condition is created by OR-ing the TFUF and RFOF flags together.

Table 1 details all DSPI sources of interrupt requests and DMA transfers. It can be seen that there are 6 DSPI flags which can trigger an interrupt, and of those 6 flags, 2 can also cause a DMA request.

The 2 flags that can cause either an eDMA or interrupt request are DSPI*x*_SR[TFFF] and DSPI*x*_SR[RFDF]. This allows SPI communications to be handled with or without CPU intervention.

Selection of DMA over interrupt is controlled by the DSPIx_RSER [TFFF_RE,TFFF_DIRS,RFDF_RE and RFDF_DIRS] bit fields.

2.2 DSPIx_PUSHR

As mentioned previously, this register is used to transmit data out of the DSPI module. The following is an explanation of each of the bit fields:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CONT	CTAS			EOQ	CTCNT	0	0	0	0	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W							▨	▨								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	Base + 0x34															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	Base + 0x34															

- CONT—Continuous Peripheral Chip Select Enable. This bit field specifies if PCS signals remain asserted between transfers.
- CTAS—Clock and Transfer Attributes Select. This is used to select one of CTAR 0:7 registers to be used for timings needed in communicating to external device. (Slave mode defaults to CTAR0)
- EOQ—End of Queue
- CTCNT—Clear SPI Transfer Counter. This provides a means to clear the SPI transfer counter (used for queue management).
- PCSO:5—Peripheral Chip Select 0-5
- TxData—Data to be transmitted

NOTE

None of the control bits, including (EOQ and CTNT), are valid in slave mode.

2.3 DSPIx_CTAR[0:7]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DBR	FMSZ				CPOL	CPHA	LSBFE	PCSSCK		PASC		PDT		PBR	
W																
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	Base + 0xC (DSPIx_CTAR0); 0x10 (DSPIx_CTAR1); 0x14 (DSPIx_CTAR2); 0x18 (DSPIx_CTAR3); 0x1C (DSPIx_CTAR4); 0x20 (DSPIx_CTAR5); 0x24 (DSPIx_CTAR6); 0x28 (DSPIx_CTAR7)															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSSCK				ASC				DT				BR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	Base + 0xC (DSPIx_CTAR0); 0x10 (DSPIx_CTAR1); 0x14 (DSPIx_CTAR2); 0x18 (DSPIx_CTAR3); 0x1C (DSPIx_CTAR4); 0x20 (DSPIx_CTAR5); 0x24 (DSPIx_CTAR6); 0x28 (DSPIx_CTAR7)															

The DSPIx_CTAR[0:7] registers have the following bit fields for configuring the timing of the SPI frames:

- FMSZ Frame Size: 4 to 16 bits
- CPOL Clock Polarity: Inactive state of SCK is low (0) or high (1)
- CPHA Clock Phase:
  - — 0: SCK Leading edge: data is captured; Following edge: data is changed
  - — 1: SCK Leading edge: data is changed; Following edge: data is captured
- LSBFE: LSB First Enable – selects if LSB or MSB is transferred first.
- PCSSCK, CSSCK: PCS to SCK delay
- PASC, ASC: SCK to PCS delay
- PDT, DT: Delay after transfer (PCS negation to next PCS assertion)
- PBR, BR: Baud rate

These eight CTARs allow different frame types (baud rates, clock frequency, delays) to be entered into the TX FIFO queue.

2.4 Serialized / Deserialized Sources and Destinations

There are four sources for serialized data on the MPC5554. They are the following:

- eTPU_A, eTPU_B, and eMIOS output channels

- Memory-mapped register DSPIx_ASDR

There are five destinations for deserialized data. They are the following:

- eTPU_A, eTPU_B, and eMIOS input channels
- SIU External Interrupt Request inputs
- Memory-mapped register in the DSPI

Refer to figures 8-11 for details of the individual connections.

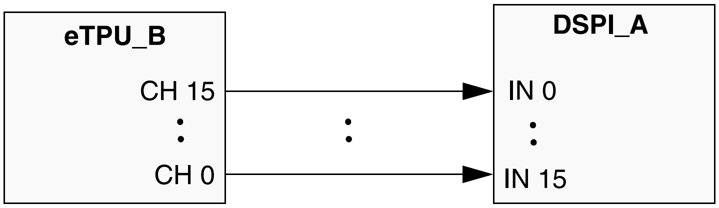


Figure 8. DSPI_A Connections

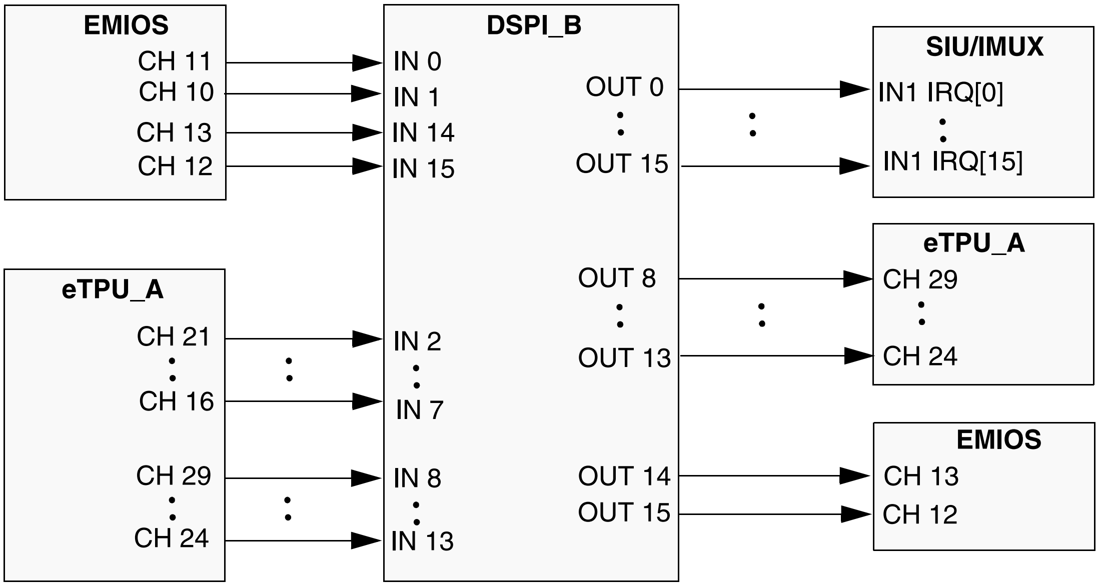


Figure 9. DSPI_B Connections

NOTE

In Figure 9, it may look as if there are 24 ouputs; however, OUT[8:13] can be routed to eTPU_A or IRQ[8:13]. Likewise, OUT[14:15] can be routed to eMIOS[13:14].

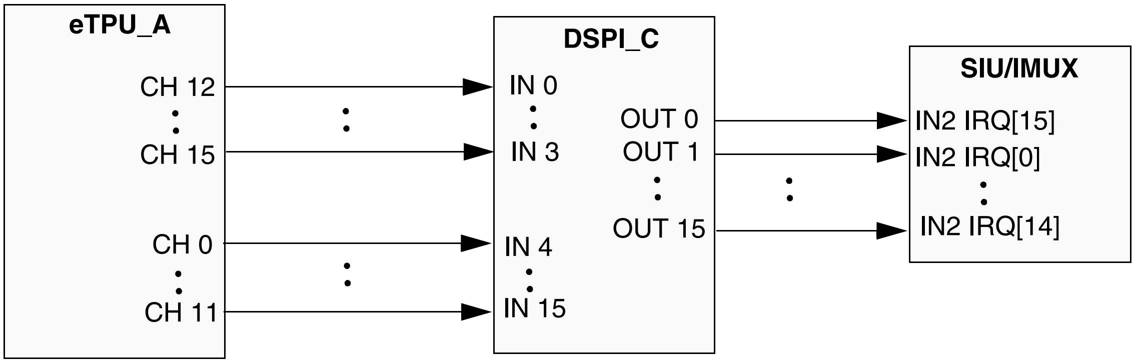


Figure 10. DSPI_C Connections

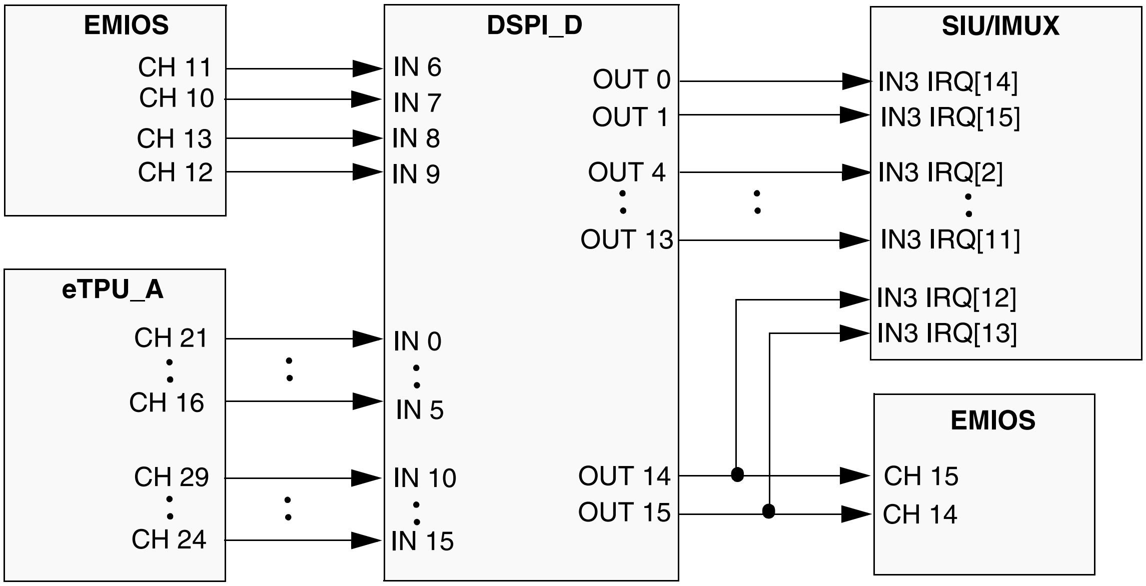


Figure 11. DSPI_D Connections

A DSI transfer can be initiated in four different ways. They are the following:

- Continuous—data transmitted continuously
- Edge sensitive hardware trigger (when chaining DSPIs)
- Change in data (any change in the data to be transmitted causes a new SPI frame to be transmitted)
- Triggered or change in data

Table 2 details how each transfer initiation is selected. The method is dependent on the DSPI_DSICR[TRRE] and DSPI_DSICR[CID] bits.

Table 2. DSI Data Transfer Initiation Control

DSPI_DSICR		Transfer Initiation Control
TRRE	CID	
0	0	Continuous
0	1	Change in Data
1	0	Triggered
1	1	Triggered or Change in Data

2.5 Serial Chaining

As the eMIOS and eTPU channels are hardwired per DSPI module, if the user's desired mix of channels to be serialized or deserialized is located on different DSPI modules, then serial chaining can be used to transfer the data in one frame. For example, if eTPU_A and eTPU_B channels are required to be serialized, then DSPI_A (for eTPU_B) and DSPI_B, _C, or _D (for eTPU_A) have to be used.

The MPC55xx supports parallel and serial chaining of DSPI modules. This can be achieved by using the hardware trigger mode. Refer to Appendix B source code that configures DSPI_A, _B, and _C for serial chaining.

Figure 10 shows a block diagram for chaining DSPI_A, _B, and _C serially to provide up to 48 (3 x 16) bits of serial data.

In Figure 12, DSPI_A is the master DSI module; both slave modules should be configured for change in data or triggered mode. When a slave has a change in data, the #MTRIG(Master Trigger) signal is asserted, propagating to the #HT (Hardware Trigger) input of the next slave or the master. Upon receiving this signal, the master then transmits a new SPI frame.

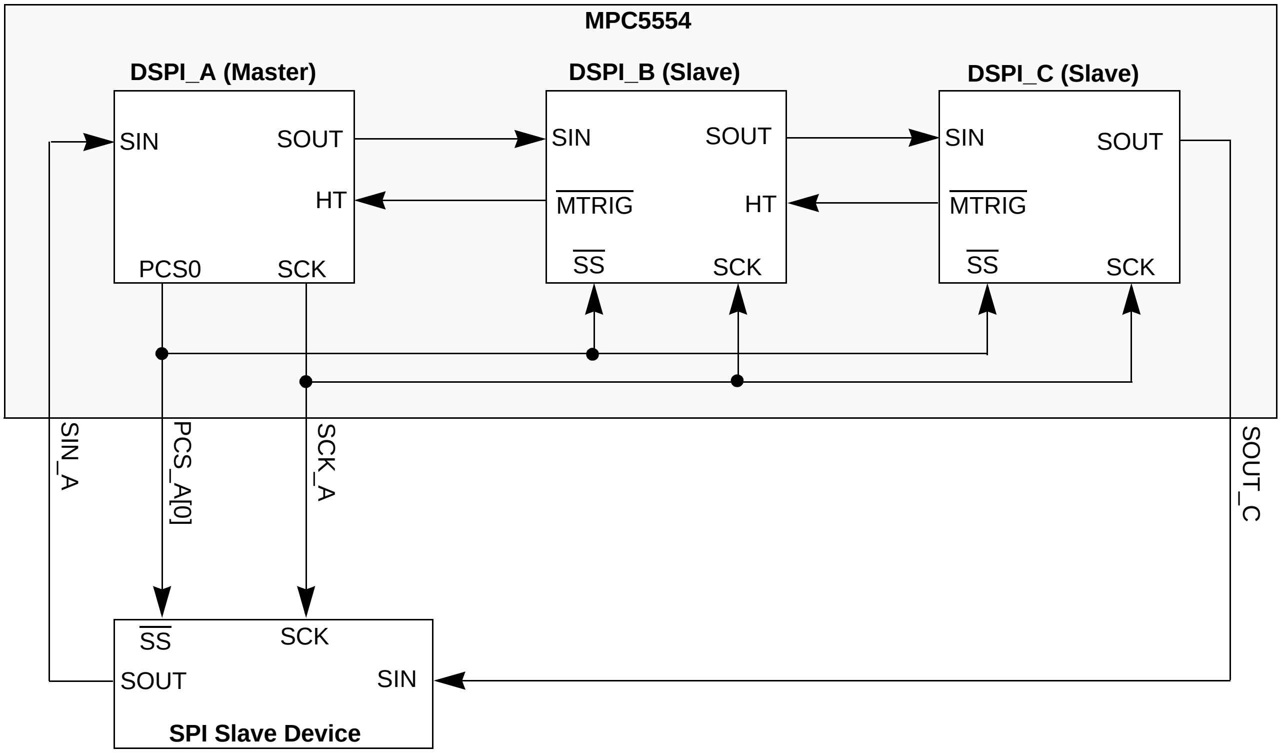


Figure 12. Example of Serial Chaining

The SIU[DISR] register controls the internal shorting of the SCK, PCS, SOUT and SIN signals between modules. Refer to Revision 1.1 or later of the *MPC5554 Reference Manual* for more detail.

3 DSI Examples

3.1 Serialized Mode

Up to 16 ETPU (or a combination of eTPU and eMIOS) channels can be serialized onto 3 wires (half-duplex master transmit) or 4 wires for master/slave operation. Data is taken from the eTPU, eMIOS, or ASDR and transmitted serially, as shown below in Figure 13. In the case of a half-duplex transfer, up to 13 pins can be saved as 16 outputs and can be transmitted on 3 pins. This means the 13 pins used previously for eTPU or eMIOS can be used as GPIO pins.

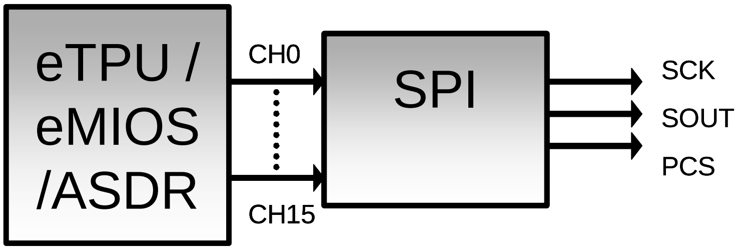


Figure 13. Serial Transmission

A practical system application for serialization would be electronic transmission control, where the eMIOS PWM outputs are fed to solenoids that control the hydraulic fluid used for gear selection. Another example would be motor control in the industrial environment.

The following serialization example configures eMIOS channels 10–13 as general purpose outputs. These outputs are fed into inputs 6–9 of DSPI_D, as detailed in Table 3. Inputs 6–9 correspond to bits 6–9 of the SPI data frame. DSPI_D is configured as a DSI master, with DSPI_C configured as an SPI slave. DSPI_D is configured to transfer when there is a change in data.

The EMIOS_CCRn[EDPOL] bit is used to toggle the value of each of the four channels. When the channel value changes, this results in a new SPI frame being transmitted. The value transmitted can be viewed in the DSPI_C_POPR register.

For example, if channel 10 is changed from 0 to 1 and all remaining bit values are 0, then the value transmitted will be 0x0080.

Refer to Appendix C for the source code of this example.

Table 3. DSPI_D Connections

DSPI_D Input	Connected to:	DSPI_D Output	Connected to:
0	eTPU_A Output Channel 21	0	Input 3 on IMUX for External IRQ[14]
1	eTPU_A Output Channel 20	1	Input 3 on IMUX for External IRQ15]
2	eTPU_A Output Channel 19	2	N/C
3	eTPU_A Output Channel 18	3	N/C
4	eTPU_A Output Channel 17	4	Input 3 on IMUX for External IRQ[2]
5	eTPU_A Output Channel 16	5	Input 3 on IMUX for External IRQ[3]
6	EMIOS Output Channel 11	6	Input 3 on IMUX for External IRQ[4]
7	EMIOS Output Channel 10	7	Input 3 on IMUX for External IRQ[5]
8	EMIOS Output Channel 13	8	Input 3 on IMUX for External IRQ[6]
9	EMIOS Output Channel 12	9	Input 3 on IMUX for External IRQ[7]
10	eTPU_A Output Channel 29	10	Input 3 on IMUX for External IRQ[8]
11	eTPU_A Output Channel 28	11	Input 3 on IMUX for External IRQ[9]
12	eTPU_A Output Channel 27	12	Input 3 on IMUX for External IRQ[10]
13	eTPU_A Output Channel 26	13	Input 3 on IMUX for External IRQ[11]
14	eTPU_A Output Channel 25	14	EMIOS Input Channel 15, Input 3 on IMUX for External IRQ[12]
15	eTPU_A Output Channel 24	15	EMIOS Input Channel 14, Input 3 on IMUX for External IRQ[13]

NOTE

Input number refers to the bit number of the SPI frame that the signal to be serialized appears on. Output refers to the bit number of the SPI frame that is connected internally to the peripheral module.

3.2 Deserialized Mode

Data from a received SPI frame is routed through to the eTPU, eMIOS channels, or to the SIU interrupts, as illustrated in Figure 14.

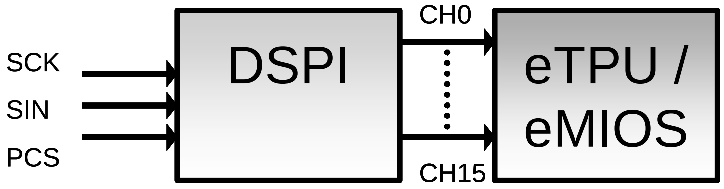


Figure 14. Data Route

The 16 SIU external interrupt inputs can be connected to either an external pin or to deserialized output signals from a DSPI module. This example is used to show how a deserialized message can cause an external interrupt on IRQ1. DSPI_A is used to transmit a SPI message to DSPI_B with bit 1 of the SPI frame controlling the interrupt. A practical example for this usage is when a SmartMOS™ device (such as the Freescale MC33394) in the system reports a fault to the MCU by asserting an interrupt.

Table 4. DSPI_D Connections

DSPI_B Input	Connected to:
0	EMIOS Output Channel 11
1	EMIOS Output Channel 10
2	eTPU_A Output Channel 21
3	eTPU_A Output Channel 20
4	eTPU_A Output Channel 19
5	eTPU_A Output Channel 18
6	eTPU_A Output Channel 17
7	eTPU_A Output Channel 16
8	eTPU_A Output Channel 29
9	eTPU_A Output Channel 28
10	eTPU_A Output Channel 27
11	eTPU_A Output Channel 26
12	eTPU_A Output Channel 25
13	eTPU_A Output Channel 24
14	EMIOS Output Channel 13
15	EMIOS Output Channel 12

DSPI_B Output	Connected to:
0	Input 1 on IMUX for External IRQ[0]
1	Input 1 on IMUX for External IRQ[1]
2	Input 1 on IMUX for External IRQ[2]
3	Input 1 on IMUX for External IRQ[3]
4	Input 1 on IMUX for External IRQ[4]
5	Input 1 on IMUX for External IRQ[5]
6	Input 1 on IMUX for External IRQ[6]
7	Input 1 on IMUX for External IRQ[7]
8	eTPU_A Input Channel 29, Input 1 on IMUX for External IRQ[8]
9	eTPU_A Input Channel 28, Input 1 on IMUX for External IRQ[9]
10	eTPU_A Input Channel 27, Input 1 on IMUX for External IRQ[10]
11	eTPU_A Input Channel 26, Input 1 on IMUX for External IRQ[11]
12	eTPU_A Input Channel 25, Input 1 on IMUX for External IRQ[12]
13	eTPU_A Input Channel 24, Input 1 on IMUX for External IRQ[13]
14	EMIOS Input Channel 13, Input 1 on IMUX for External IRQ[14]
15	EMIOS Input Channel 12, Input 1 on IMUX for External IRQ[15]

The IRQ*n* pins map to 16 independent interrupt request outputs from the SIU. An interrupt request is asserted when the corresponding IRQ Flag bit is set in in the (SIU_EISR) register, with the corresponding DMA/Interrupt Request Enable bit set in the DMA/Interrupt Request Enable Register (SIU_DIRER) and the corresponding DMA/Interrupt Select bit cleared in the DMA/Interrupt Request Select Register (SIU_DIRSR).

The IRQ Flag bit is set when an event as defined by IRQ Rising-Edge Event Enable Register (SIU_IREER) and IRQ Falling-Edge Event Enable Register (SIU_IFEER) occurs on the corresponding IRQ*n* pin. This allows for rising, falling, or both edges to cause an interrupt. Figure 16 details graphically the process mentioned above.

From Table 4, it is evident that bit 1 of the SPI frame is routed through to input 1 on the SIU_IMUX, which connects to IRQ1. When the value 0x2 is received by DSPI_B, IRQ1 will be asserted. Refer to Appendix D for source code that implements a deserialized interrupt.

The input source for each SIU external interrupt is individually specified in the external IRQ input select register (SIU_EIISR). An example of the multiplexing of an SIU external interrupt input is given in Figure 15. As shown in the figure, the IRQ0 input of the SIU can be sourced from either the IRQ0_GPIO203 pin, the DSPI_B0 serial input signal, the DSPI_C1 serialized input signal, or the DSPI_D2 serialized output signal.

All IRQ inputs are multiplexed in the same manner. The inputs to the IRQ from each DSPI module are offset by one, so that if more than one DSPI module is connected to the same external device type, a separate interrupt can be generated for each device. This also applies to DSPI modules connected to external devices of different type that have status bits in the same bit location of the deserialized information.

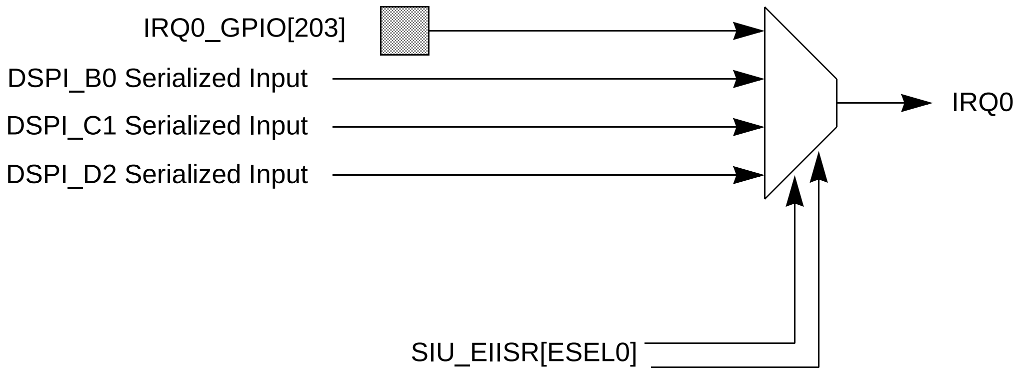


Figure 15. Multiplexing an SIU External Interrupt Input

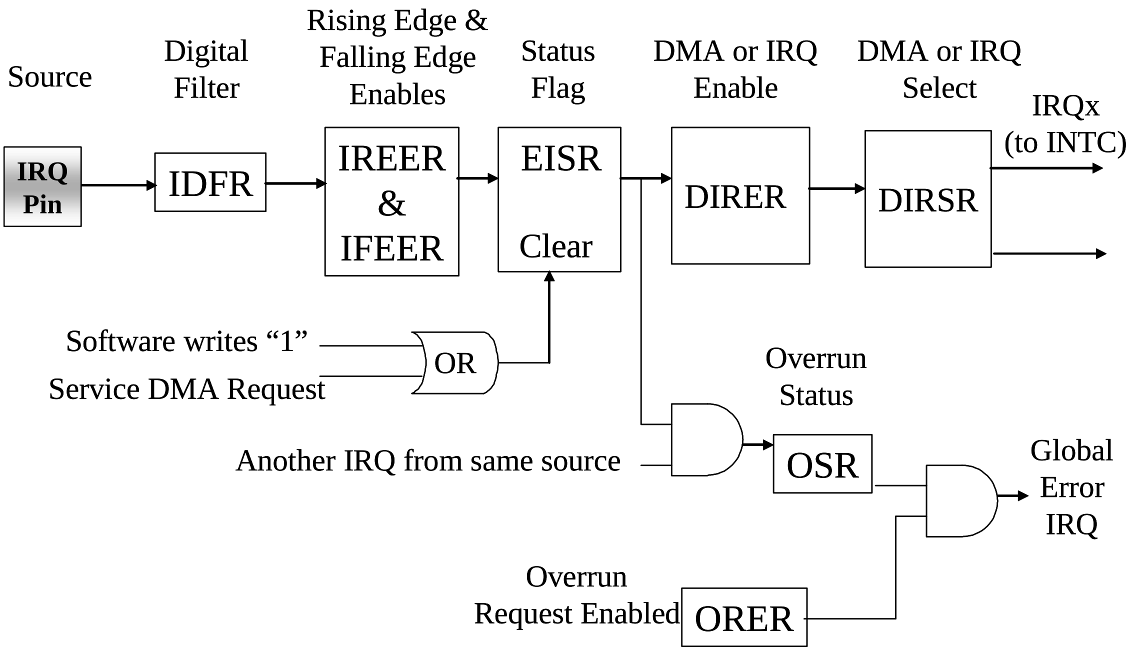


Figure 16. External Interrupt Configuration for the IRQ pins (logic gates are descriptive and not implemented in hardware)

NOTE

As every pin on the MPC55xx is configured individually, the function for the IRQ pins must be initialized to IRQ if required. However, in the example above, the pin is not monitored; it is the internal signal that causes the interrupt to be asserted. For further information on interrupts, the registers mentioned above, and those in previous sections, please refer to the *MPC5554 Reference Manual,* Revision 1.1, 05/2004 or later.

4 MPC5xx / MPC5554 SPI Comparison

Table 5 illustrates the direct comparison of the MPC5554 DSPOI module and the MPC5xx QSPI module with regardsto SPI communication.

MPC5xx Family QSMCM QSPI	MPC5554 DSPI
Full or Half Duplex Operation	Full or Half Duplex Operation
SCK Baud Rate of up to Sysclk/4	SCK Baud Rate of up to Sysclk/4
32 Entry transmit and receive queues (16-bit)	Unlimited queue size when using DMA with internal / external RAM.
32 Serial Transfers without CPU Intervention	No CPU intervention required as the eDMA services the transmit and receive FIFOs.
	Buffered transmit and receive operation using the TX and RX FIFOs, with depths of 4 entries

	Visibility into TX and RX FIFOs for ease of debugging
	FIFO bypass mode for low-latency updates to SPI queues
Wrap Around Operation allowing continuous transmit and receive	Using the SMOD and DMOD TCD fields circular buffers can be created to do the same.
	Programmable transfer attributes on a per-frame basis:
	Programmable serial frame size of 4 to 16 bits, expandable with software control
4 Programmable Peripheral Chip Select Pins, extended to 8 on the MPC561/3 however these are multiplexed with other pins.	Six peripheral chip selects, expandable to 64 with external demultiplexer
The QSPI allows byte, half-word, and word accesses.	DSPI allows byte, half-word and word accesses.
Data stored in the receive RAM is right-justified, (i.e., the least significant bit is always in the right-most bit position within the word regardless of the serial transfer length). Unused bits in a receive queue entry are set to zero by the QSPI upon completion of the individual queue entry.	Same
Data must be written to transmit RAM in a right-justified format. The QSPI cannot modify information in the transmit RAM.	Same. DSPI must be configured correctly
The CPU writes one byte of control information for each QSPI command to be executed. The QSPI cannot modify information in command RAM.	Command queue can be written at any time provided it is allocated to volatile memory. Both the DMA and CPU may write to a volatile command queue. Each command is 16 bits.
QSPI loop mode. LOOPQ controls feedback on the data serializer for testing.	Not supported
Halt QSPI. When HALT is set, the QSPI stops on a queue boundary. It remains in a defined state from which it can later be restarted.	Same
Master/slave mode select	Same
Wired-OR mode for QSPI pins. This bit controls the QSPI pins regardless of whether they are used as general-purpose outputs or as QSPI outputs, and regardless of whether the QSPI is enabled or disabled.	Not supported directly. However wired-or functionality can be achieved by setting the PCR[ODE,WPE] bit fields correctly.
Data transfers from 8 to 16 bits are supported.	Supported with different range of values.
Clock polarity. CPOL is used to determine the inactive state of the serial clock (SCK).	Same, but on a per frame basis instead of global.
Clock phase. CPHA determines which edge of SCK causes data to change and which edge causes data to be captured.	Same, but on a per frame basis instead of global.
Serial clock baud rate. The QSPI uses a modulus counter to derive the SCK baud rate from the MCU IMB3 clock. Baud rate is selected by writing a value from 2 to 255 into SPBR.	Supported with different range of values.

Delay before SCK.	Defined differently. See document text for details.
Length of delay after transfer.	Defined differently. See document text for details.
SPI finished interrupt enable.	Not supported in DSPI. Supported in DMA controller.
HALTA and MODF interrupt enable. HMIE enables interrupt requests generated by the HALTA status flag or the MODF status flag in SPSR.	No HALTA.
QSPI finished flag. SPIF is set after execution of the command at the address in ENDQP in SPCR2. If wraparound mode is enabled (WREN = 1), the SPIF is set, after completion of the command defined by ENDQP, each time the QSPI cycles through the queue.	Not supported in DSPI. Supported in DMA controller.
Mode fault flag. The QSPI asserts MODF when the QSPI is in master mode (MSTR = 1) and the SS input pin is negated by an external driver.	Not supported
MOSI signal(and pin) changes direction dependent on master or slave mode selection.	The SOUT and SIN signals(and pins) remain fixed as output and input regardless of master/slave mode selection.

5 Conclusion

From the previous sections, it has been shown that the the MPC55xx family provides flexibility for SPI communications with the DSPI module. It has also been shown that this module can facilitate the release of module pins to be used as GPIO. This in turn provides the user with flexibility to configure the pins on the MPC55xx device to suit their application.

Appendix A – Simple SPI example using the eDMA

NOTE

The example source code in these Appendices does not take care of device initialization after reset. This code is intended as main routines that can be pasted into the user's initialization code.

```
//*********************************************************************************************/
/*COPYRIGHT (c) FREESCALE 2004                                                   */
/*FILE NAME: SPI_DMA.c                                                           */
/* PROJECT NAME :Copperhead(MPC5554)                                             */
/* INCLUDE FILES: MPC5554.h typedefs.h                                           */
/* VERSION: 0.1                                                                  */
/*                                                                               */
/*=========================================================                       */
/*                                                                               */
/* DESCRIPTION: This code transmits a sequence of 16 bit words                   */
/* from DSPIC to D. Transmit and receive queues are set up in system             */
/* RAM. eDMA channels 14(DSPIC TFFF) and 17(DSPID RFDF) are                      */
/*used to transfer the data.                                                     */
/*                                                                               */
/*=========================================================                       */
/*                                                                               */
/* COMPILER: Diab Data       VERSION: 5.1.2                                      */
/*                                                                               */
/* AUTHOR: Steven McQuade                   CREATION DATE: 15/03/04              */
/* LOCATION: East Kilbride, Scotland.                                            */
/*                                                                               */
/* UPDATE HISTORY                                                                */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                         */
/* ---    -----------  ---------    ----------------------                       */
/* 0.1   S.McQuade 15/03/04    Initial version of file                           */
/*                                                                               */
/************************************************************************************************/

#include "mpc5554.h"
#include "typedefs.h"




extern vuint32_t DSPI_TXQUEUE[4];     // Set up a command queue for DSPI C transmit

extern vuint32_t DSPI_RXQUEUE[4];  // Set up a result queue for DSPI D


void initDSPI(void)
```

```
 {

/******   SPI C to D using internal connections *********/

DSPI_C.MCR.R            = 0x80010000;          /* SPI mode DSPIC = Master PCS0=active lo*/

DSPI_C.CTAR[0].R        = 0x78003255;         /* FSIZE =4,CPOL=0,CPHA=0
                               LSBE=0 PCSSCK=0 CSSCK=3 PASC=0ASC=2PDT=0DT=5
                                              PBR =0 BR=4 */

DSPI_D.MCR.R            = 0x00010000;         /* SPI mode DSPID = Slave SS0=active lo*/
DSPI_D.CTAR[0].R        = 0x78003255;

SIU.DISR.R                     = 0x000000FC;/* DSPID inputs are taken from DSPIC o/ps
                                                     NO EXTERNAL CONNECTIONS
                        REQ'D*/

DSPI_C.RSER.R           = 0x03000000;         /* Setup TFFF to cause DMA transfer*/
DSPI_D.RSER.R           = 0x00030000;         /* Setup RFDF to cause DMA transfer */

}



void main (void)
{

DSPI_TXQUEUE[0] = 0x00010001;                 /* EOQ=0 PCS0=active Non continuos SCK Data = 1*/
DSPI_TXQUEUE[1] = 0x00010002;                 /* Data = 2*/
DSPI_TXQUEUE[2] = 0x00010003;                 /* Data = 3*/
DSPI_TXQUEUE[3] = 0x08010004;                 /* Data = 4, EOQ =1*/


initDMA();                                    /* Setup DMA   */
initDSPI();


  while(1)
  {
   }
}

/* ************************************************************************
*
*                  End of File
*
* ************************************************************************ */
```

```
//*****************************************************************************************/
/*                              COPYRIGHT (c) FREESCALE 2004                              */
/* FILE NAME: DMA_Init.c                                                                  */
/* PROJECT NAME :Copperhead(MPC5554)                                                      */
/* INCLUDE FILES: MPC5554.h typedefs.h                                                    */
/* VERSION: 0.1                                                                           */
/*                                                                                        */
/*=========================================================                               */
/*                                                                                        */
/* DESCRIPTION: This code initializes the eDMA and configures Channel 14                  */
/* (DSPIC TFFF) Transfer Control Descriptor DSPIC to D. Transmit and                      */
/* receive queues are set up in system RAM. eDMA channels 14(DSPIC TFFF)                  */
/* and 17(DSPID RFDF) are used to transfer the data.                                      */
/*                                                                                        */
/*                                                                                        */
/*=========================================================                               */
/*                                                                                        */
/* COMPILER: Diab Data        VERSION: 5.1.2                                              */
/*                                                                                        */
/* AUTHOR: Steven McQuade                    CREATION DATE: 15/03/04                      */
/* LOCATION: East Kilbride, Scotland.                                                     */
/*                                                                                        */
/* UPDATE HISTORY                                                                         */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                                  */
/* ---   -----------  ---------    ----------------------                                 */
/* 0.1   S.McQuade 15/03/04    Initial vesion of file                                     */
/*                                                                                        */
/******************************************************************************************/


#include "mpc5554.h"


#define DSPIA_PUSHR  0xFFF90034
#define DSPIB_PUSHR  0xFFF94034
#define DSPIC_PUSHR  0xFFF98034
#define DSPID_PUSHR  0xFFF9C034

#define DSPIA_POPR   0xFFF90038
#define DSPIB_POPR   0xFFF94038
#define DSPIC_POPR   0xFFF98038
#define DSPID_POPR   0xFFF9C038

#define DSPIA_RXFR   0xFFF9007C
```

```
// ******************* Referenced Variables *******************

vuint32_t DSPI_TXQUEUE[4];    // Set up a command queue for DSPI C transmit

vuint32_t DSPI_RXQUEUE[4];  // Set up a result queue for DSPI D

// ********************** Function Prototypes **********************
void initDMA(void);

// ******************** Initilization Function ********************
void initDMA(void)
{

// DMA Configuration Register (DMACR)
   EDMA.CR.R   = 0x0000E400;  // Group Arbitration scheme is : Fixed Priority
                              // Channel Arbitration scheme is : Fixed Priority
                              // Group 3 Priority (GRP3PRI) is : 4
                              // Group 2 Priority (GRP2PRI) is : 3
                              // Group 1 Priority (GRP1PRI) is : 2
                              // Group 0 Priority (GRP0PRI) is : 1
                              // Debug is : Disabled (0)
                              // Buffered Writes are :  Enabled (1)

// DMA Enable Request Registers (DMAERQH,DMAERQL)
   EDMA.ERQRH.R   = 0x00000000;  // DMA Enable Request Register High (Channels 63-32) :
   EDMA.ERQRL.R   = 0x00000000;  // DMA Enable Request Register Low (Channels 31-0) :
                                 // Clear these registers before initialising the channel TCD.
                                 // Enable channels after TCDs have been initialised.
                                 // Failure to do so could cause errors with a h/w triggered
                                 channel,
                                 // i.e. the dma could be triggered after the ERQR bits are set
                                 and
                                 // before the TCD is configured


// DMA Enable Error Interrupt Request Registers (DMAEEIH,DMAEEIL)
   EDMA.EEIRH.R   = 0x00000000;  // DMA Error Interrupt Enable Register High (Channels 32-63) :
   EDMA.EEIRL.R   = 0x00000000;  // DMA Error Interrupt Enable Register Low (Channels 0-31) :


// DMA Channel Priority Register Settings (DCHPRIn) Channels 0 - 63
      EDMA.CPR[0].R = 0x00;    // Channel 0 is Priority 1, Channel Preemption is Disabled
      EDMA.CPR[1].R = 0x01;    // Channel 1 is Priority 2, Channel Preemption is Disabled
      EDMA.CPR[2].R = 0x02;    // Channel 2 is Priority 3, Channel Preemption is Disabled
      EDMA.CPR[3].R = 0x03;    // Channel 3 is Priority 4, Channel Preemption is Disabled
      EDMA.CPR[4].R = 0x04;    // Channel 4 is Priority 5, Channel Preemption is Disabled
      EDMA.CPR[5].R = 0x05;    // Channel 5 is Priority 6, Channel Preemption is Disabled
      EDMA.CPR[6].R = 0x06;    // Channel 6 is Priority 7, Channel Preemption is Disabled
      EDMA.CPR[7].R = 0x07;    // Channel 7 is Priority 8, Channel Preemption is Disabled
      EDMA.CPR[8].R = 0x08;    // Channel 8 is Priority 9, Channel Preemption is Disabled
      EDMA.CPR[9].R = 0x09;    // Channel 9 is Priority 10, Channel Preemption is Disabled
      EDMA.CPR[10].R = 0x0A; // Channel 10 is Priority 11, Channel Preemption is Disabled
      EDMA.CPR[11].R = 0x0B; // Channel 11 is Priority 12, Channel Preemption is Disabled
      EDMA.CPR[12].R = 0x0C; // Channel 12 is Priority 13, Channel Preemption is Disabled
      EDMA.CPR[13].R = 0x0D; // Channel 13 is Priority 14, Channel Preemption is Disabled
      EDMA.CPR[14].R = 0x0E; // Channel 14 is Priority 15, Channel Preemption is Disabled
      EDMA.CPR[15].R = 0x0F; // Channel 15 is Priority 16, Channel Preemption is Disabled
      EDMA.CPR[16].R = 0x00; // Channel 16 is Priority 1, Channel Preemption is Disabled
      EDMA.CPR[17].R = 0x01; // Channel 17 is Priority 2, Channel Preemption is Disabled
      EDMA.CPR[18].R = 0x02; // Channel 18 is Priority 3, Channel Preemption is Disabled
      EDMA.CPR[19].R = 0x03; // Channel 19 is Priority 4, Channel Preemption is Disabled
      EDMA.CPR[20].R = 0x04; // Channel 20 is Priority 5, Channel Preemption is Disabled
      EDMA.CPR[21].R = 0x05;    // Channel 21 is Priority 6, Channel Preemption is Disabled
      EDMA.CPR[22].R = 0x06;    // Channel 22 is Priority 7, Channel Preemption is Disabled
      EDMA.CPR[23].R = 0x07;    // Channel 23 is Priority 8, Channel Preemption is Disabled
      EDMA.CPR[24].R = 0x08;    // Channel 24 is Priority 9, Channel Preemption is Disabled
      EDMA.CPR[25].R = 0x09;    // Channel 25 is Priority 10, Channel Preemption is Disabled
      EDMA.CPR[26].R = 0x0A;    // Channel 26 is Priority 11, Channel Preemption is Disabled
      EDMA.CPR[27].R = 0x0B;    // Channel 27 is Priority 12, Channel Preemption is Disabled
      EDMA.CPR[28].R = 0x0C;    // Channel 28 is Priority 13, Channel Preemption is Disabled
      EDMA.CPR[29].R = 0x0D;    // Channel 29 is Priority 13, Channel Preemption is Disabled
      EDMA.CPR[30].R = 0x0E;    // Channel 30 is Priority 15, Channel Preemption is Disabled
      EDMA.CPR[31].R = 0x0F;    // Channel 31 is Priority 16, Channel Preemption is Disabled
      EDMA.CPR[32].R = 0x00;    // Channel 32 is Priority 1, Channel Preemption is Disabled
      EDMA.CPR[33].R = 0x01;    // Channel 33 is Priority 2, Channel Preemption is Disabled
      EDMA.CPR[34].R = 0x02;    // Channel 34 is Priority 3, Channel Preemption is Disabled
      EDMA.CPR[35].R = 0x03;    // Channel 35 is Priority 4, Channel Preemption is Disabled
      EDMA.CPR[36].R = 0x04;    // Channel 36 is Priority 5, Channel Preemption is Disabled
      EDMA.CPR[37].R = 0x05;    // Channel 37 is Priority 6, Channel Preemption is Disabled
      EDMA.CPR[38].R = 0x06;    // Channel 38 is Priority 7, Channel Preemption is Disabled
      EDMA.CPR[39].R = 0x07;    // Channel 39 is Priority 8, Channel Preemption is Disabled
      EDMA.CPR[40].R = 0x08;    // Channel 40 is Priority 9, Channel Preemption is Disabled
      EDMA.CPR[41].R = 0x09;    // Channel 41 is Priority 10, Channel Preemption is Disabled
      EDMA.CPR[42].R = 0x0A;    // Channel 42 is Priority 11, Channel Preemption is Disabled
      EDMA.CPR[43].R = 0x0B;    // Channel 43 is Priority 12, Channel Preemption is Disabled
      EDMA.CPR[44].R = 0x0C;    // Channel 44 is Priority 13, Channel Preemption is Disabled
      EDMA.CPR[45].R = 0x0D;    // Channel 45 is Priority 14, Channel Preemption is Disabled
      EDMA.CPR[46].R = 0x0E;    // Channel 46 is Priority 15, Channel Preemption is Disabled
      EDMA.CPR[47].R = 0x0F;    // Channel 47 is Priority 16, Channel Preemption is Disabled
      EDMA.CPR[48].R = 0x00;    // Channel 48 is Priority 1, Channel Preemption is Disabled
      EDMA.CPR[49].R = 0x01;    // Channel 49 is Priority 2, Channel Preemption is Disabled
      EDMA.CPR[50].R = 0x02;    // Channel 50 is Priority 3, Channel Preemption is Disabled
      EDMA.CPR[51].R = 0x03;    // Channel 51 is Priority 4, Channel Preemption is Disabled
      EDMA.CPR[52].R = 0x04;    // Channel 52 is Priority 5, Channel Preemption is Disabled
      EDMA.CPR[53].R = 0x05;    // Channel 53 is Priority 6, Channel Preemption is Disabled
      EDMA.CPR[54].R = 0x06;    // Channel 54 is Priority 7, Channel Preemption is Disabled
```

```
    EDMA.CPR[55].R = 0x07;    // Channel 55 is Priority 8, Channel Preemption is Disabled
    EDMA.CPR[56].R = 0x08;    // Channel 56 is Priority 9, Channel Preemption is Disabled
    EDMA.CPR[57].R = 0x09;    // Channel 57 is Priority 10, Channel Preemption is Disabled
    EDMA.CPR[58].R = 0x0A;    // Channel 58 is Priority 11, Channel Preemption is Disabled
    EDMA.CPR[59].R = 0x0B;    // Channel 59 is Priority 12, Channel Preemption is Disabled
    EDMA.CPR[60].R = 0x0C;    // Channel 60 is Priority 13, Channel Preemption is Disabled
    EDMA.CPR[61].R = 0x0D;    // Channel 61 is Priority 14, Channel Preemption is Disabled
    EDMA.CPR[62].R = 0x0E;    // Channel 62 is Priority 15, Channel Preemption is Disabled
    EDMA.CPR[63].R = 0x0F;    // Channel 63 is Priority 16, Channel Preemption is Disabled


//  Transfer Control Descriptor for DSPI C TFFF

    EDMA.TCD[14].SADDR = (vuint32_t) &DSPI_TXQUEUE[0];        //  Start Address
    EDMA.TCD[14].DADDR = DSPIC_PUSHR;          //  Destination address
    EDMA.TCD[14].SMOD = 0x00;                  //  Source address modulo
    EDMA.TCD[14].DMOD = 0x00;                  //  Destination address modulo
    EDMA.TCD[14].DSIZE = 0x02;                 //  Destination transfer size : 32 Bits
    EDMA.TCD[14].SSIZE = 0x02;                 //  Source transfer size : 32 Bits
     EDMA.TCD[14].SOFF = 0x4;                  //  Signed source address offset
     EDMA.TCD[14].NBYTES = 0x4;                //  Inner "minor" byte count
     EDMA.TCD[14].SLAST = 0;                   //  last Signed source address adjust
     EDMA.TCD[14].DOFF = 0x0;                  //  Signed destination address offset
     EDMA.TCD[14].DLAST_SGA = 0x0;             //  Signed destination address adjust
    EDMA.TCD[14].BITERE_LINK = 0x0;     //  //Channel-to-channel Linking on Minor Loop Complete
    EDMA.TCD[14].BITER = 0x1;                 //  begining "major" iteration count
    EDMA.TCD[14].CITERE_LINK = 0x0;     //  Channel-to-channel Linking on Minor Loop Complete
    EDMA.TCD[14].CITER=0x1;                   //  Current “major” iteration count Disabled
    EDMA.TCD[14].BWC = 0x00;                  //  Bandwidth control :  No DMA Stalls
    EDMA.TCD[14].MAJORLINKCH = 0x00; //  Major Channel number
    EDMA.TCD[14].MAJORE_LINK = 0x0;     //  Major Channel Link : Disabled
    EDMA.TCD[14].DONE = 0x00;                 //  Channel Done
    EDMA.TCD[14].ACTIVE = 0x00;               //  Channel ACtive
    EDMA.TCD[14].E_SG = 0x0;                  //  Enable Scatter/Gather : Disabled
    EDMA.TCD[14].D_REQ = 0x0;                 //  Do NOT disable TCD When done
    EDMA.TCD[14].INT_HALF = 0x0;              //  Interrupt on minor loop count
    EDMA.TCD[14].INT_MAJ = 0x0;               //  Interrupt on major loop completion : Disabled
    EDMA.TCD[14].START = 0x00;                //  Explicit Channel Start bit

//  Transfer Control Descriptor for DSPI D RFDF

    EDMA.TCD[17].SADDR = DSPID_POPR;//  Start Address
    EDMA.TCD[17].DADDR = (vuint32_t) &DSPI_RXQUEUE;        //  Destination address
    EDMA.TCD[17].SMOD = 0x00;                  //  Source address modulo
    EDMA.TCD[17].DMOD = 0x00;                  //  Destination address modulo
    EDMA.TCD[17].DSIZE = 0x02;                 //  Destination transfer size : 32 Bits
    EDMA.TCD[17].SSIZE = 0x02;                 //  Source transfer size : 32 Bits
    EDMA.TCD[17].SOFF = 0;                     //  Signed source address offset
```

```
        EDMA.TCD[17].NBYTES = 0x4;              //  Inner "minor" byte count
        EDMA.TCD[17].SLAST = 0;                 //  last Signed source address adjust
        EDMA.TCD[17].DOFF = 4;                  //  Signed destination address offset
        EDMA.TCD[17].DLAST_SGA = 0;             //  Signed destination address adjust
        EDMA.TCD[17].BITERE_LINK = 0x0;         //  Channel-to-channel Linking on Minor Loop Complete
        EDMA.TCD[17].BITER = 0x1;               //  begining "major" iteration count
       EDMA.TCD[17].CITERE_LINK = 0x0;          //  Channel-to-channel Linking on Minor Loop Complete
         EDMA.TCD[17].CITER=0x1;                //  Current "major" iteration count Disabled
        EDMA.TCD[17].BWC = 0x00;                //  Bandwidth control :  No DMA Stalls
        EDMA.TCD[17].MAJORLINKCH = 0x00;        //  Major Channel number
        EDMA.TCD[17].MAJORE_LINK = 0x0;         //  Major Channel Link : Disabled
        EDMA.TCD[17].DONE = 0x00;               //  Channel Done
        EDMA.TCD[17].ACTIVE = 0x00;             //  Channel ACtive
        EDMA.TCD[17].E_SG = 0x0;                //  Enable Scatter/Gather : Disabled
        EDMA.TCD[17].D_REQ = 0x0;               //  Do NOT disable TCD When done
        EDMA.TCD[17].INT_HALF = 0x0;            //  Interrupt on minor loop count : Disabled
        EDMA.TCD[17].INT_MAJ = 0x0;             //  Interrupt on major loop completion : Disabled
        EDMA.TCD[17].START = 0x00;              //  Explicit Channel Start bit




  // DMA Enable Request Registers (DMAERQH,DMAERQL)
  // Enable Ch14=DSPIC TXFFF, Ch17=DSPID RFDF

      EDMA.ERQRH.R   = 0x00000000;  // DMA Enable Request Register High (Channels 63-32) :
      EDMA.ERQRL.R   = 0x00024000;  // DMA Enable Request Register Low (Channels 31-0) :


}
/* **************************************************************************
*
*                          End of File
*
* ************************************************************************** */
```

Appendix B – Serial chaining example

```
//**********************************************************************************/
/*                                      COPYRIGHT (c) FREESCALE 2004           */
/* FILE NAME: dsitrigger                                                       */
/* PROJECT NAME :Copperhead(MPC5554) EVB silicon Evaluation                    */
/* INCLUDE FILES: MPC5554.h typedefs.h                                         */
/* VERSION: 0.1                                                                */
/*                                                                             */
/*=========================================================*/
/*                                                                             */
/* DESCRIPTION: This test is used to check serial chaining of DSPIA,B,C*/
/* which requires the use of the hardware trigger. The data to be              */
/* transmitted for each module is stored in the DSPIx_ASDR register.           */
/* DSPI B&C are configured to transmit 4 bits with DSPI A initialized for      */
/* 8 bit. Using a debugger change the data in either DSPIB or C. This          */
/* should result on a 16 bit frame being transmitted on SOUTC. This 16         */
/* bit value is the result of concatenating  8 bits  from DSPIA, 4 bits from   */
/* DSPI B and 4 bits from DSPIC. The slave data for DSPIs B &C can be */
/* viewed in DSPI A Deserialisation Data Register.                             */
/*                                                                             */
/*========================================================== */
/*                                                                             */
/* COMPILER: Diab Data        VERSION: 5.1.2                                   */
/*                                                                             */
/* AUTHOR: Steven McQuade                  CREATION DATE: 15/03/04    */
/* LOCATION: East Kilbride, Scotland.                                          */
/*                                                                             */
/* UPDATE HISTORY                                                              */
/* REV      AUTHOR       DATE       DESCRIPTION OF CHANGE                     */
/* ---   -----------  ---------    ----------------------                      */
/* 0.1   S.McQuade 14/04/04    Initial version of file                         */
/*                                                                             */
/**********************************************************************************/

#include "mpc5554.h"
#include "typedefs.h"


void initDSPI(void)
 {
  vuint32_t ADR =0;



/******    DSI A master, B&C DSI slaves *********/
```

```
DSPI_A.MCR.R= 0x903F0000;               /* DSI mode DSPIA = Master PCS0-5=active
                                        lo*/
DSPI_A.CTAR[0].R = 0x78003255;          /* FSIZE =16,CPOL=0,CPHA=0 LSBE=0 PCSSCK=0
                                        PASC=0ASC=2PDT=0DT=5 PBR =0 BR=4 CSSCK=3*/

DSPI_B.MCR.R= 0x103F0000;               /* DSI mode DSPIB = Slave PCS0-5=active
                                        lo*/

DSPI_B.CTAR[1].R        = 0x38003255;   /* FSIZE =16,CPOL=0,CPHA=0 LSBE=0
                                        PCSSCK=0 CSSCK=3
                                        PASC=0ASC=2PDT=0DT=5 PBR =0 BR=4
                                        */

  DSPI_C.MCR.R= 0x103F0000;             /* DSI mode DSPIC = Slave PCS0-
                                        5=active  lo*/

  DSPI_C.CTAR[1].R = 0x38003255;        /* FSIZE =16,CPOL=0,CPHA=0 LSBE=0
                                        PCSSCK=0 CSSCK=3
                                        PASC=0ASC=2PDT=0DT=5 PBR =0 BR=4
                                        */

  DSPI_A.DSICR.B.MTOE= 1;               /* Enable multiple transfers*/
  DSPI_A.DSICR.B.MTOCNT= 0xF;           /* No of SCK clocks to shift out frame. Same as the number of bits in the
                                        frame to be shifted*/
  DSPI_A.DSICR.B.TXSS= 1;               /* Source of data is ADR*/
  DSPI_A.DSICR.B.TPOL= 0;               /* Falling edge initiates a transfer*/
  DSPI_A.DSICR.B.TRRE= 1;               /* Trigger enabled*/
  DSPI_A.DSICR.B.CID = 1;               /* Change in Data disabled when 0*/
  DSPI_A.DSICR.B.DCONT= 0;              /* Non continuos DSI chip select*/
  DSPI_A.DSICR.B.DSICTAS= 0;            /* CTAR0 used*/
  DSPI_A.DSICR.B.DPCS0= 1;              /* USE DPCS0*/

  DSPI_B.DSICR.B.MTOE= 1;               /* Enable multiple transfers*/
  DSPI_B.DSICR.B.MTOCNT= 0xF;           /* No of SCK clocks to shift out frame. Same as the number of bits in the
                                        frame to be shifted*/

  DSPI_B.DSICR.B.TXSS= 1;               /* Source of data is ADR*/
  DSPI_B.DSICR.B.TPOL= 0;               /* Falling edge initiates a transfer*/
  DSPI_B.DSICR.B.TRRE= 1;               /* Trigger enabled*/
  DSPI_B.DSICR.B.CID = 1;               /* Change in Data disabled when 0*/
  DSPI_B.DSICR.B.DCONT= 0;              /* Non continuos DSI chip select*/
  DSPI_B.DSICR.B.DSICTAS= 1;            /* CTAR1 is used for slaves*/
  DSPI_B.DSICR.B.DPCS0= 1;              /* USE DPCS0*/

  DSPI_C.DSICR.B.MTOE= 1;               /* Enable multiple transfers*/
  DSPI_C.DSICR.B.MTOCNT= 0xF;           /* No of SCK clocks to shift out frame. Same as no.of bits in frame to be
                                        shifted*/
  DSPI_C.DSICR.B.TXSS= 1;             /* Source of data is ADR*/
  DSPI_C.DSICR.B.TPOL= 0      ;       /* Falling edge initiates a transfer*/
  DSPI_C.DSICR.B.TRRE= 1;             /* Trigger enabled*/
  DSPI_C.DSICR.B.CID = 1;             /* Change in Data disabled when 0*/
  DSPI_C.DSICR.B.DCONT= 0;            /* Non continuous DSI chip select*/
  DSPI_C.DSICR.B.DSICTAS= 1;          /* CTAR1 used for slaves*/
  DSPI_C.DSICR.B.DPCS0= 1;            /* USE DPCS0*/

  DSPI_A.ASDR.R       = 0x12;         /* Write alternate data*/
  DSPI_B.ASDR.R       = 0x56;         /* Write alternate data*/
  SIU.DISR.R          = 0x81569400;/* short SOUT,SIN,SCK and PCS internally
  }


void initPins(void) {

/*****************************************************************************************************
/*************************************         DSPIA (MASTER)              *******************/

/**** SCKA ****/

  SIU.PCR[93].B.PA = 0x3;             /* Initialize pin assignment to primary, SCKA */
  SIU.PCR[93].B.OBE = 1;              /* Enable output buffer (SCK OBE =1 when Master and 0 when Slave) */
  SIU.PCR[93].B.IBE = 1;              /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                      power consumption)*/
  SIU.PCR[93].B.DSC = 0x3;            /* 50pF drive strength*/

/**** SOUTC ****/

  SIU.PCR[107].B.PA = 0x2;            /* Initialize pin assignment to secondary, SOUTA */
  SIU.PCR[107].B.OBE = 1;             /* Enable output buffer as SOUTA is an output */
  SIU.PCR[107].B.IBE = 1;             /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                      power consumption)*/
  SIU.PCR[107].B.DSC = 0x3;           /* 50pF drive strength*/

/**** PCSA0 ****/

  SIU.PCR[96].B.PA = 0x3;             /* Initialize pin assignment to primary, PCSA0 */
  SIU.PCR[96].B.OBE = 1;              /* Enable output buffer as PCSA0 is an output */
  SIU.PCR[96].B.IBE = 1;              /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                      power consumption)*/
  SIU.PCR[96].B.DSC = 0x3;            /* 50pF drive strength*/


}
```

```
void main (void)
{
  initPins();                          /* Initialize pins for non-GPIO assignments - DSPI */
  initDSPI();

  DSPI_C.ASDR.R = 0x0a;                /* Write alternate data*/


   while(1)
   {
                                       /*End of code (trap)*/
   }
}


/* *****************************************************************************
*
*                        End of File
*
* ***************************************************************************** */
```

Appendix C – eMIOS Serialisation example

```
//*********************************************************************************************/
/*                                  COPYRIGHT (c) FREESCALE 2004                              */
/* FILE NAME: dsi.c                                                                           */
/* PROJECT NAME :Copperhead(MPC5554) EVB silicon Evaluation                                   */
/* INCLUDE FILES: MPC5554.h typedefs.h                                                        */
/* VERSION: 0.1                                                                               */
/*                                                                                            */
/*=========================================================                                    */
/*                                                                                            */
/* DESCRIPTION: This test case is used to verify DSPID DSI functionality.                     */
/* This source code serialises channels 10-13 of the EMIOS. DSPIC is configured*/
/* as a SPI slave in order to capture the data transmitted by DSPID. On the                   */
/* evaluation or customer’s own board connect SOUT of DSPID to SIN of DSPIC, */
/* SCKC to SCKD and PCSD0 to PCSC0(SS0). Using a debugger place a                             */
/* breakpoint at the modifyMIOSdata function call in main. Run to this point then            */
/* modify the EDPOL bit of eMIOS channels 10-13. Run the code once again.                     */
/* This time data should appear on DSPID SOUT and the transmitted value will                  */
/* be captured in DSPIC_POPR. Repeat this process to observe different values                 */
/*transmitted.                                                                                */
/*                                                                                            */
/*                                                                                            */
/*=========================================================                                    */
/*                                                                                            */
/* COMPILER: Diab Data        VERSION: 5.1.2                                                  */
/*                                                                                            */
/* AUTHOR: Steven McQuade                  CREATION DATE: 15/03/04                            */
/* LOCATION: East Kilbride, Scotland.                                                         */
/*                                                                                            */
/* UPDATE HISTORY                                                                             */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                                      */
/* ---   -----------  ---------    ----------------------                                     */
/* 0.1   S.McQuade 15/03/04    Initial version of file                                        */
/*                                                                                            */
/*********************************************************************************************/

#include "mpc5554.h"
#include "typedefs.h"

#define HI 1
#define LO 0

vuint32_t CH10 =0; vuint32_t CH11 =0; vuint32_t CH12 =0; vuint32_t CH13 =0;
vuint32_t POPR =0;
```

```
void initeMIOS();
void delay();
void modifyMIOSdata();



void modifyMIOSdata(void)
 {

 EMIOS.CH[10].CCR.B.EDPOL = CH10; /* EDPOL used to toggle pin value */
 EMIOS.CH[11].CCR.B.EDPOL = CH11; /* EDPOL used to toggle pin value */
 EMIOS.CH[12].CCR.B.EDPOL = CH12; /* EDPOL used to toggle pin value */
 EMIOS.CH[13].CCR.B.EDPOL = CH13; /* EDPOL used to toggle pin value */
  }



void delay(void)
 {
  vuint32_t a =0;

         for(a=0xFFFF;a!=0;a--)
          {}
  }



void initeMIOS(void)
 {


  EMIOS.MCR.R = 0;/* No Clocks enabled*/
  EMIOS.OUDR.R = 0;/* No update disabled*/
  EMIOS.CH[10].CADR.R = 0;/* CH10 Areg =0*/
  EMIOS.CH[10].CBDR.R = 0;/* CH10 Breg =0*/
  EMIOS.CH[11].CADR.R = 0;/* CH11 Areg =0*/
  EMIOS.CH[11].CBDR.R = 0;/* CH11 Breg =0*/
  EMIOS.CH[12].CADR.R = 0;/* CH12 Areg =0*/
  EMIOS.CH[12].CBDR.R = 0;/* CH12 Breg =0*/
  EMIOS.CH[13].CADR.R = 0;/* CH13 Areg =0*/
  EMIOS.CH[13].CBDR.R = 0;/* CH13 Breg =0*/

  EMIOS.CH[10].CCR.B.MODE = 0x1; /* Mode is General Purpose Output */
  EMIOS.CH[11].CCR.B.MODE = 0x1; /* Mode is General Purpose Output */
  EMIOS.CH[12].CCR.B.MODE = 0x1; /* Mode is General Purpose Output */
  EMIOS.CH[13].CCR.B.MODE = 0x1; /* Mode is General Purpose Output */

 EMIOS.CH[10].CCR.B.EDPOL = CH10; /* EDPOL used to toggle pin value */
 EMIOS.CH[11].CCR.B.EDPOL = CH11; /* EDPOL used to toggle pin value */
 EMIOS.CH[12].CCR.B.EDPOL = CH12; /* EDPOL used to toggle pin value */
 EMIOS.CH[13].CCR.B.EDPOL = CH13; /* EDPOL used to toggle pin value */


  }





void initDSPI(void)
 {
  uint32_t SDR =0;


/******    DSI D to C *********/

DSPI_C.MCR.R                  = 0x00010000; /* SPI mode DSPIC = SLAVE SS0=active lo*/

DSPI_C.CTAR[0].R              = 0x78003255;/*FSIZE=16,CPOL=0,CPHA=0
                                                    LSBE=0 PCSSCK=0 CSSCK=3
                                                    PASC=0ASC=2PDT=0DT=5 PBR =0 BR=4 */

DSPI_D.MCR.R                  = 0x90010000;/* DSI mode DSPID = Master PCS0=active lo*/

DSPI_D.CTAR[0].R              = 0x7800325    /*FSIZE=16,CPOL=0,CPHA=0
                                                    LSBE=0 PCSSCK=0 CSSCK=3
                                                    PASC=0ASC=2PDT=0DT=5 PBR =0 BR=4 */

DSPI_D.DSICR.B.MTOE           = 0;      /* Enable multiple transfers*/
DSPI_D.DSICR.B.MTOCNT         = 0xF;    /* No of SCK clocks to shift out frame. Same as no.of
                                        bits in frame to be shifted*/
DSPI_D.DSICR.B.TXSS           = 0;      /* Source of data is SDR*/
DSPI_D.DSICR.B.TPOL           = 0;      /* Falling edge initiates a transfer*/
DSPI_D.DSICR.B.TRRE           = 0;      /* Trigger disabled*/
DSPI_D.DSICR.B.CID            = 1;      /* Change in Data disabled when 0*/
DSPI_D.DSICR.B.DCONT          = 0;      /* Non continuous DSI chip select*/
DSPI_D.DSICR.B.DSICTAS        = 0;      /* CTAR0 used*/
DSPI_D.DSICR.B.DPCS0          = 1;      /* USE DPCS0*/


SDR = DSPI_D.SDR.R;                     /* Read value of SDR should be val on eMIOS CH10-13*/

/************************************************************************************/
/* Use this line if SCK,SIN,SOUT and PCS pins have to be used.            */
/************************************************************************************/
```

```
SIU.DISR.R                   = 0x00000000;/* No serial or parallel chaining, data
                                                  is taken from the pins */

/********************************************************************************/
/* Use this line if SCK,SIN,SOUT and PCS signals are shorted internally. */
/********************************************************************************/

//SIU.DISR.R = 0x0000FC00;/* DSPIC inputs are taken from DSPID o/ps*/


  }


void initPins(void) {

/*************************************************************************************
/************************           DSPIC (SPI Slave)                ******************/

/**** SCKC ****/

SIU.PCR[109].B.PA = 0x2;                 /* Initialize pin assignment to seconary, SCKC */
SIU.PCR[109].B.OBE = 0;                  /* Enable output buffer (SCK OBE =1 when Master and 0 when Slave) */
SIU.PCR[109].B.IBE = 1;                  /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
SIU.PCR[109].B.DSC = 0x3;/               * 50pF drive strength*/


/**** SINC ****/

SIU.PCR[108].B.PA = 0x2;       /* Initialize pin assignment to secondary, SINC */
SIU.PCR[108].B.OBE = 0;       /* Disable output buffer as SINA is an input */
SIU.PCR[108].B.IBE = 1;        /* Enable input buffer so pin can be monitored
                                         (can be set to 0 to reduce power consumption)*/
SIU.PCR[108].B.DSC = 0x3;   /* 50pF drive strength*/

/**** SOUTC ****/

SIU.PCR[107].B.PA = 0x2;                 /* Initialize pin assignment to secondary, SOUTC */
SIU.PCR[107].B.OBE = 1;                  /* Enable output buffer as SOUTA is an output */
SIU.PCR[107].B.IBE = 1;                  /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
SIU.PCR[107].B.DSC = 0x3;                /* 50pF drive strength*/

/**** PCSC0 ****/

  SIU.PCR[110].B.PA = 0x2;               /* Initialize pin assignment to secondary, SSC0*/
  SIU.PCR[110].B.OBE = 0;                /* Disable output buffer as SSC0 is an output */
  SIU.PCR[110].B.IBE = 1;                /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
  SIU.PCR[110].B.DSC = 0x3;              /* 50pF drive strength*/

/**** PCSC1 ****/

SIU.PCR[102].B.PA = 0x2;                 /* Initialize pin assignment to secondary, PCSC1 */
SIU.PCR[102].B.OBE = 1;                  /* Enable output buffer as PCSC1 is an output */
SIU.PCR[102].B.IBE = 1;                  /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
SIU.PCR[102].B.DSC = 0x3;                /* 50pF drive strength*/

/**** PCSC2 ****/

SIU.PCR[103].B.PA = 0x2;                 /* Initialize pin assignment to secondary, PCSC2 */
SIU.PCR[103].B.OBE = 1;                  /* Enable output buffer as PCSC2 is an output */
SIU.PCR[103].B.IBE = 1;                  /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
SIU.PCR[103].B.DSC = 0x3;                /* 50pF drive strength*/

/**** PCSC3 ****/

SIU.PCR[85].B.PA = 0x2;                  /* Initialize pin assignment to secondary, PCSC3 */
SIU.PCR[85].B.OBE = 1;                   /* Enable output buffer as PCSC3 is an output */
SIU.PCR[85].B.IBE = 1;                   /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
SIU.PCR[85].B.DSC = 0x3;                 /* 50pF drive strength*/

/**** PCSC4 ****/

SIU.PCR[86].B.PA = 0x2;                  /* Initialize pin assignment to secondary, PCSC4 */
SIU.PCR[86].B.OBE = 1;                   /* Enable output buffer as PCSC4 is an output */
SIU.PCR[86].B.IBE = 1;                   /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
SIU.PCR[86].B.DSC = 0x3;                 /* 50pF drive strength*/

/**** PCSC5 ****/

SIU.PCR[104].B.PA = 0x2;                 /* Initialize pin assignment to secondary, PCSA5 */
SIU.PCR[104].B.OBE = 1;                  /* Enable output buffer as PCSA0 is an output */
SIU.PCR[104].B.IBE = 1;                  /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                         power consumption)*/
  SIU.PCR[104].B.DSC = 0x3;              /* 50pF drive strength*/

/*********************************************************************************************
/****************************          DSPID (SLAVE)                  *******************/

/**** SCKD ****/
```

```
SIU.PCR[98].B.PA    = 0x2;       /* Initialize pin assignment to secondary, SCKD */
SIU.PCR[98].B.OBE = 1;           /* disable output buffer (SCK OBE =1 when Master and 0 when Slave) */
SIU.PCR[98].B.IBE   = 1;         /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                 power consumption)*/
SIU.PCR[98].B.DSC = 0x3;        /* 50pF drive strength*/

/**** SIND ****/

SIU.PCR[99].B.PA = 0x2;          /* Initialize pin assignment to secondary, SIND */
SIU.PCR[99].B.OBE = 0;           /* Disable output buffer as SINA is an input */
SIU.PCR[99].B.IBE = 1;           /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                 power consumption)*/
SIU.PCR[99].B.DSC = 0x3;         /* 50pF drive strength*/

/**** SOUTD ****/

SIU.PCR[100].B.PA = 0x2;         /* Initialize pin assignment to secondary, SOUTD */
SIU.PCR[100].B.OBE = 1;          /* Enable output buffer as SOUTD is an output */
SIU.PCR[100].B.IBE = 1;          /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                 power consumption)*/
SIU.PCR[100].B.DSC = 0x3;        /* 50pF drive strength*/

/**** PCSD0 ****/

SIU.PCR[106].B.PA = 0x2;            /* Initialize pin assignment to secondary, PCSD0 */
SIU.PCR[106].B.OBE = 1;             /* Disable output buffer as PCSB0 is an input when SS */
SIU.PCR[106].B.IBE = 1;             /* Enable input buffer so pin can be SS */
SIU.PCR[106].B.DSC = 0x3;           /* 50pF drive strength*/


/**** PCSD1 ****/

SIU.PCR[91].B.PA = 0x2;             /* Initialize pin assignment to secondary, PCSD1 */
SIU.PCR[91].B.OBE = 1;              /* Enable output buffer as PCSD1 is an output */
SIU.PCR[91].B.IBE = 1;              /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                    power consumption)*/
 SIU.PCR[91].B.DSC = 0x3;           /* 50pF drive strength*/

/**** PCSD2 ****/

SIU.PCR[96].B.PA = 0x2;             /* Initialize pin assignment to secondary, PCSD2 */
SIU.PCR[96].B.OBE = 1;              /* Enable output buffer as PCSD2 is an output */
SIU.PCR[96].B.IBE = 1;              /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                    power consumption)*/
SIU.PCR[96].B.DSC = 0x3;            /* 50pF drive strength*/

/**** PCSD3 ****/
```

```
SIU.PCR[87].B.PA = 0x2;          /* Initialize pin assignment to secondary, PCSD3 */
SIU.PCR[87].B.OBE = 1;           /* Enable output buffer as PCSA3 is an output */
SIU.PCR[87].B.IBE = 1;           /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                 power consumption)*/
SIU.PCR[87].B.DSC = 0x3;         /* 50pF drive strength*/

/**** PCSD4 ****/

SIU.PCR[88].B.PA = 0x2;          /* Initialize pin assignment to secondary, PCSD4 */
SIU.PCR[88].B.OBE = 1;           /* Enable output buffer as PCSD4 is an output */
SIU.PCR[88].B.IBE = 1;           /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                 power consumption)*/
  SIU.PCR[88].B.DSC = 0x3;       /* 50pF drive strength*/

/**** PCSD5 ****/

  SIU.PCR[92].B.PA = 0x2;        /* Initialize pin assignment to secondary, PCSD5 */
  SIU.PCR[92].B.OBE = 1;         /* Enable output buffer as PCSA0 is an output */
  SIU.PCR[92].B.IBE = 1;         /* Enable input buffer so pin can be monitored (can be set to 0 to reduce
                                 power consumption)*/
  SIU.PCR[92].B.DSC = 0x3;       /* 50pF drive strength*/



  }

void main (void)
{


  initPins();      /* Initialize pins for non-GPIO assignments - DSPI */
  initeMIOS();
  initDMA();       /* Setup DMA   */
  initDSPI();


   while(1)
   {
          modifyMIOSdata();
          POPR = DSPI_C.POPR.R;
   }
}
/* ***************************************************************************
*
*                    End of File
*
* ************************************************************************** */
```

Appendix D – SIU external interrupt deserialisation example

```
//*******************************************************************************************/
/*                              COPYRIGHT (c) FREESCALE 2004                                */
/* FILE NAME: DSPI_EXT_INT.c                                                                */
/* PROJECT NAME :Copperhead(MPC5554) EVB silicon Evaluation                                 */
/* INCLUDE FILES: MPC5554.h typedefs.                                                       */
/* VERSION: 0.1                                                                             */
/*                                                                                          */
/*=========================================================                                 */
/*                                                                                          */
/* DESCRIPTION: This test is to check that a deserialised message can                       */
/* cause an external interrupt on IRQ1.                                                     */
/* DSPIA is used to transmit a SPI message to DSPIB with bit 1 controlling                  */
/* the interrupt. Using a debugger write 0x00000002 to DSPIA PUSHR to transmit*/
/* data value 2 to trigger the interrupt on IRQ1.                                           */
/*                                                                                          */
/*=========================================================                                 */
/*                                                                                          */
/* COMPILER: Diab Data       VERSION: 5.1.2                                                 */
/*                                                                                          */
/* AUTHOR: Steven McQuade                  CREATION DATE: 19/04/04                          */
/* LOCATION: East Kilbride, Scotland.                                                       */
/*                                                                                          */
/* UPDATE HISTORY                                                                           */
/* REV      AUTHOR      DATE      DESCRIPTION OF CHANGE                                     */
/* ---    -----------  ---------    ----------------------                                  */
/* 0.1   S.McQuade 19/04/04    Initial version of file                                      */
/*                                                                                          */
/*******************************************************************************************/


#include "mpc5554.h"
#include "typedefs.h"
extern void initDMA();
void init_ext_int(void);
void IRQISR(void);
extern void IVOR4Handler (void);


asm void initExcep(void) {

  lis              r0, 0x4000              ; IVPR = address of start of L2SRAM
```

```
 mtIVPRr0

 lis            r0, IVOR4Handler@h    ;IVOR4 = address of handler
 ori            r0, r0, IVOR4Handler@l
 mtIVOR4r0

}

void initINTC(void) {

 INTC.MCR.B.HVEN = 0;         /* Initialize INTC for software vector mode */
 INTC.MCR.B.VTES = 0;         /* Use the default vector table entry offsets of 4 bytes */
 INTC.IACKR.R = 0x40000000;           /* Set INTC Vector table base address to start of L2SRAM */

}

void initDSPI(void) {

 /******   SPIA to DSI B *********/

DSPI_A.MCR.R           = 0x80010000;/* SPI mode DSPIA = Master, PCS0=active lo*/
DSPI_A.CTAR[0].R       = 0x78003255;/* FSIZE =16,CPOL=0,CPHA=0 LSBE=0
                                       PCSSCK=0 CSSCK=3 PASC=0ASC=2PDT=0DT=5
                                       PBR =0 BR=4 */

DSPI_B.MCR.R           = 0x10010000;/* DSI mode DSPIB = Slave, PCS0=active lo*/
DSPI_B.CTAR[1].R       = 0x78003255;/* FSIZE =16,CPOL=0,CPHA=0 LSBE=0
                                       PCSSCK=0 CSSCK=3 PASC=0ASC=2PDT=0DT=5
                                       PBR =0 BR=4 */

                               /* When config'd as a DSI slave CTAR1 must be used.*/
                               /* DSI slave means that the rx'd data is transfered to the DSPI
                               i/ps e.g. eMIOS and eTPU channels*/

 DSPI_B.DSICR.B.MTOE      = 0;    /* Enable multiple transfers*/
 DSPI_B.DSICR.B.MTOCNT    = 0xF;  /* No of SCK clocks to shift out frame.Same as no.of bits in
                                  frame to be shifted*/
 DSPI_B.DSICR.B.TXSS      = 0;    /* Source of data is SDR*/
 DSPI_B.DSICR.B.TPOL      = 0;    /* Falling edge initiates a transfer*/
 DSPI_B.DSICR.B.TRRE      = 0;    /* Trigger disabled*/
 DSPI_B.DSICR.B.CID               = 0;    /* Change in Data disabled when 0*/
 DSPI_B.DSICR.B.DCONT     = 0;    /* Non continuous DSI chip select*/
 DSPI_B.DSICR.B.DSICTAS   = 0;    /* CTAR0 used*/
 DSPI_B.DSICR.B.DPCS0     = 1;    /* USE DPCS0*/

 SIU.DISR.R = 0x00540000;         /* DSPIB inputs taken from DSPIA internally */
                                  /* No need to connect these signals on the board*/
```

```
}

void enableIrq(void) {
  INTC.CPR.B.PRI = 0;   /* Ensure INTC's current priority is 0 */
  asm(" wrteei 1");     /* Enable external interrupts */
}




void init_ext_int(void)
{
  SIU.EIISR.R = 0x00000004; /*configure mux for IRQ1 driven from DSPIB1*/
  INTC.PSR[47].R = 1;/* IRQ1 priority = 1 */
  SIU.DIRSR.R = 0x00000000; /*select interrupt request*/
  SIU.DIRER.R = 0x00000002; /*enable interrupt request*/
  SIU.IREER.R = 0x00000002; /*enable interrupt request on rising edge*/
  SIU.IFEER.R = 0x00000002; /*enable interrupt request on falling edge*/

}


void main (void) {
  int i = 0;                     /* Dummy idle counter */

  initExcep();                  /* Initialize exceptions: load IVPR only for this example */
  initINTC();                    /* Initialize INTC for hardware vector mode */
  initDSPI();                    /* Initialize DSPI*/
  init_ext_int();
  enableIrq();                   /* Ensure INTC current priority=0 & enable IRQ */
  DSPI_A.PUSHR.R = 2;            /*Trigger the interrupt*/
  while (1) { i++; }             /* Loop forever */
}

void IRQISR(void) {

  if(SIU.EISR.B.EIF1 == 1)
  {
     SIU.EISR.B.EIF1 = 0;
   }
}
```

```
/* ***************************************************************************
*
*                    End of File
*
* ************************************************************************** */
```

NOTE

All of the above source code is for example use only. Freescale does not accept liability for use of this code in the user’s application.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047, Japan
0120 191014 or +81 3 3440 3569
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.