Using the Serial Peripheral Interface (SPI) eTPU Function

by: Jenifer Scott & Geoff Emerson
      Freescale 32-Bit Embedded Controller Division

This eTPU Serial Peripheral Interface (SPI) application note is intended to provide simple C interface functions to the eTPU SPI function. The functions are useable on any product which has an eTPU module. Example code is available for the MPC5554 and MCF5235 devices. This application note should be read in conjunction with application note AN2864, "General C Functions for the eTPU."

Table of Contents

1 Function Overview

The SPI function uses three eTPU channels to form a bi-directional, synchronous serial port that can be used to communicate with a wide variety of devices. It can be used to add serial capabilities to a device without a serial port, or to add further serial port(s) to a device that already has a hardware-synchronous port. The function requires no host CPU's intervention once the function has been initialized and a request for data transmission has been made. One eTPU channel is configured to function as the clock and the other two are configured to function as serial transmitter (TxD, or 'data out') and serial receiver (RxD, or 'data in'). The eTPU SPI

function provides similar functionality to the TPU3 SIOP function. Peripheral Chip Select (PCS) functionality is not provided by the eTPU SPI function.

The main features of the function are as follows:

- Programmable baud rate period over a 23-bit range of TCR1 or TCR2 counts
- Selection of msb or lsb first shift direction
- Variable transfer size from 1 to 24 bits
- Programmable clock polarity
- Master mode only

2 SPI Overview

An SPI is primarily used to allow a microcontroller to communicate with peripheral devices. An SPI is also capable of interprocessor communications. Peripheral devices may be as simple as an ordinary transistor-transistor logic (TTL) shift register, or as complex as a complete subsystem, such as a liquid crystal diode (LCD) display driver or an analogue-to-digital (A/D) converter subsystem. The SPI system is flexible enough to interface with numerous standard product peripherals from several manufacturers.

3 Basic Timing

The basic timing arrangement for the eTPU SPI function is shown in Figure 1and Figure 2. With positive clock polarity, the master (eTPU SPI) changes the transmitted data on the falling edge of the clock. The received data from the slave is sampled on the rising edge of the clock. With negative clock polarity, the master (eTPU SPI) changes the transmitted data on the rising edge of the clock. The received data from the slave is sampled on the falling edge of the clock.
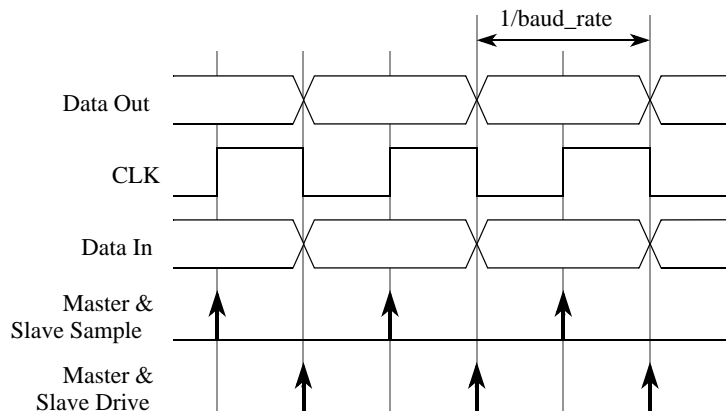

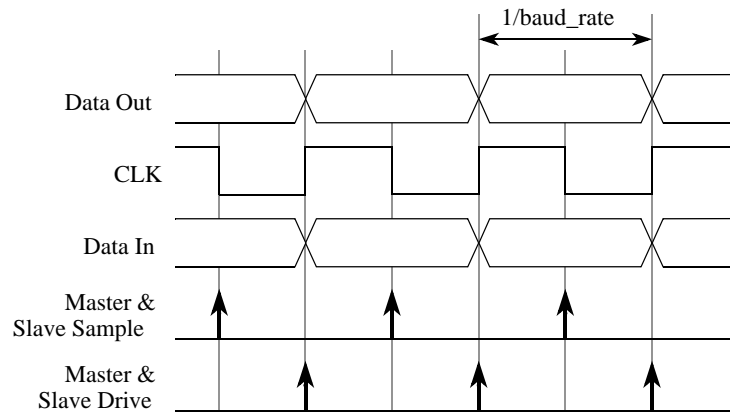**Figure 1. SPI Timing when CLK has Positive Polarity**

Figure 2. SPI Timing when CLK has Negative Polarity

4 Using the Function

A simple C API is provided to allow customers to use the SPI function quickly and easily. The API is defined in Section 5, "C Level API for eTPU SPI Function."

An initialization function call is used to set up the SPI function to match the user's requirements. Following initialization, the transmit function call should be used to send and receive data. The function works in such a way that when data is transmitted on the TxD pin, data will also be sampled on the RxD pin. The transmit function will use the SPI parameters defined during initialization.

The channel arrangement for the SPI function is fixed. The diagram and table below show this arrangement.
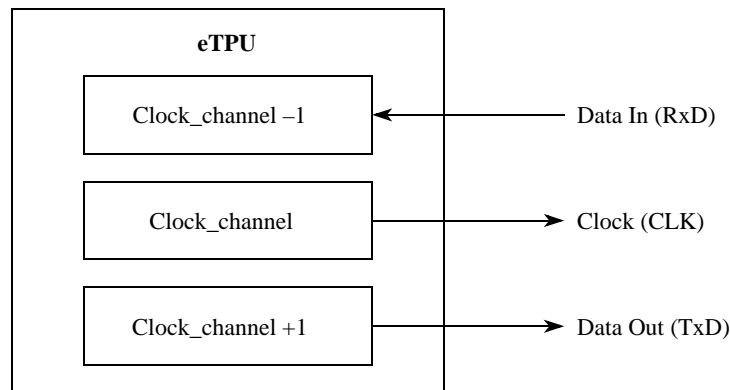
Pin function	Channel number
Data in (RxD)	Clock_channel -1
Clock	Clock_channel
Data Out (TxD)	Clock_channel +1

When a transfer of data is complete, the SPI function's clock channel sends an interrupt and DMA request. The DMA request signal for a given channel may not be connected; the connection depends on the specific integration of the eTPU. The programmer can use either the DMA request or the interrupt request, depending on the specific integration.

Typically an SPI will require a Chip Select signal to be asserted. This signal can be generated by using either the General Purpose Input/Output on the host, or by using the eTPU GPIO function.

4.1 Notes on the Performance and Use of eTPU SPI Function

Like all eTPU functions, SPI function performance in an application, to some extent, depends on the service time (latency) of other active eTPU channels. This is due to the operational nature of the eTPU scheduler.

When a single SPI function is in use and no other eTPU channels are active, the minimum time between clock edges is 37 eTPU clock cycles. This is due to the time taken for the eTPU engine to calculate and schedule the events required by the SPI function. If the time requested between clock edges is less than this minimum, the SPI function transmits and receives data correctly, but the actual time between clock edges will be longer then requested, as the baud rate will be limited by the eTPU's bandwidth. When more eTPU channels are active, performance decreases, as the eTPU engine can only service one channel at a time. However, worst-case latency in any eTPU application can be closely estimated. To analyze the performance of an application that appears to approach the limits of the eTPU, use the guidelines given in the *eTPU Reference Manual* and the information provided in the eTPU SPI software release available from Freescale.

Although the SPI function is a three-channel function, only the clock channel is scheduled for service. The TXD and RxD channel hardware is used by the clock channel. All three channels share the same parameter RAM. The use of the TxD and RxD channels does not affect he performance of the eTPU, and should therefore not be considered in latency calculations.

For the MPC5554 with a system frequency of 132 MHz, the maximum baud rate using TCR1 with a timebase_freq of 66 Mhz is 1.75 Mbits per second.

For the MCF5235 with a system frequency of 150MHz, the maximum baud rate using TCR1 with a timebase_freq of 37.5 Mhz is 0.85 Mbits per second.

Maximum baud-rate is influenced by compiler efficiency. The above numbers are given for guidance only and are subject to change. For up to date information, refer to the information provided in the eTPU SPI software release available from Freescale.

5 C Level API for eTPU SPI Function

The following functions provide easy access for the application developer into the eTPU SPI function for the application developer. Use of these functions eliminates the need to directly control the eTPU registers. The API consists of three functions. These functions can be found in the etpu_spi.h and etpu_spi.c files.

The functions are described below and are available from Freescale. In addition, the eTPU compiler generates a file called etpu_spi_auto.h. This file contains information relating to the eTPU SPI function including details of how the Parameter RAM is organized and definitions for the various API interface information.

Unless otherwise stated the function parameter macros are defined in etpu_spi_auto.h

SPI Initialization

```
int32_t_fs_etpu_spi_init          (uint8_t clock_channel, uint8_t priority,

                                    uint32_t baud_rate, uint8_t shift_dir,

                                    uint8_t polarity, uint32_t transfer_size,

                                    uint8_t timebase, uint32_t timebase_freq);
```

SPI Transmit Data

```
void fs_etpu_spi_transmit_data (uint8_t clock_channel, uint32_t tx_data);
```

SPI Retrieve Received data

```
uint32_t fs_etpu_spi_get_data (uint8_t clock_channel);
```

5.1   SPI Initialization Function

Function fs_etpu_spi_init is used to initialize three eTPU channels for the eTPU SPI function. For further transfers of data, the transmit data function call should be used.

This function dynamically allocates parameter RAM. If dynamic allocation is not required, then the clock channel's Channel Parameter Base Address field should be written with a non-zero value before calling the fs_etpu_spi_init function.

Dynamic allocation of parameter RAM occurs if the channel has a zero in its Channel Parameter Base Address field. The Channel Parameter Base Address field is updated by the API with a non-zero value to point to the parameter RAM allocated to the channel. The fs_etpu_spi_init API will not allocate new parameter RAM if the channel has a non-zero value in its Channel Parameter Base Address field; this means that channel has already been assigned. The channel can be reconfigured to transmit/receive data with a different set of transmit/receive parameters by calling the fs_etpu_spi_init function again.

This function has the following parameters:
- clock_channel: The SPI clock channel number. For devices with two eTPUs, this parameter should be assigned a value of 1-30 for eTPU_A and 65-94 for eTPU_B. For products with a single eTPU, this parameter should be assigned a value of 1-30. Operation is not guaranteed if channels 0, 31, 64, or 95 are used as clock_channel.
- priority: The priority to assign to the eTPU SPI clock channel. The following eTPU priority definitions are found in the utilities file etpu_util.h. This parameter should be assigned one of the following values:
  — FS_ETPU_PRIORITY_HIGH
  — FS_ETPU_PRIORITY_MIDDLE

— FS_ETPU_PRIORITY_LOW

— FS_ETPU_PRIORITY_DISABLED

- baud_rate: This is the required rate of data transmission. Maximum baud rate is dependant upon system frequency, timebase frequency and eTPU loading; see Section 4.1, "Notes on the Performance and Use of eTPU SPI Function."

  The eTPU microcode schedules an event every 'half period' of the programmed eTPU SPI baud_rate. The half period is calculated by the initialization API according to the following formula:

  half_period = timebase_freq/(2* baud_rate).

  If this formula resolves to a non-integer value, then the API will effectively round down half_period. This will result in an error in the baud_rate. The accuracy of data transfers is unaffected by this error.

- shift_dir: Selection of msb (most significant bit) or lsb (least significant bit) first shift direction. This parameter should be assigned one of the following values:

  — FS_ETPU_SPI_SHIFT_DIR_MSB

  — FS_ETPU_SPI_SHIFT_DIR_LSB

- Polarity: This is the polarity of the channel. This parameter should be assigned one of the following values:

  — FS_ETPU_SPI_CLK_POL_POS

  — FS_ETPU_SPI_CLK_POL_NEG

  When polarity is FS_ETPU_SPI_CLK_POL_POS, the eTPU SPI function samples the RxD pin on the rising edge of the clock channel and changes the TxD pin on the falling edge of the clock channel.

  When polarity is FS_ETPU_SPI_CLK_POL_NEG, the eTPU SPI function samples the RxD pin on the falling edge of the clock channel and changes the TxD pin on the rising edge of the clock channel.

- transfer_size: This is the size of the data to be transferred. This parameter should be assigned a value of between 1 and 24 decimal.

- timebase: This is the timebase to use as a reference for the SPI clock signal. This parameter should be assigned one of the following values (definitions are found in the utilities file etpu_util.h):

  — FS_ETPU_TCR1

  — FS_ETPU_TCR2

- timebase_freq: This is the pre-scaled frequency of the timebase (either TCR1/TCR2), supplied by the eTPU to the function

5.2 SPI Transmit Data Function

Function fs_etpu_spi_transmit_data is used only after the SPI function has been initialized using the fs_etpu_spi_init function. The configuration selected at initialization will be used for data transfers using this function call.

This function call is used to transmit and/or receive data. When the function transmits a data bit, a corresponding data bit is received. The SPI function can be used to transmit data only by using the transmit API function call and ignoring the received data.

The data will be justified to the left by this API call if the function has been configured to send/receive data msb first. For example, if the API is used to transmit 4 bits with msb first and tx_data= 0xFC 8431, the API would left justify the tx_data parameter to be 0x10 0000. The transmitted data would come out of the TxD channel in the following bit order: 0,0,0,1.

If the same data is transmitted lsb first, no left justification would be performed, and the transmitted data would come out of the TxD channel in the following bit order: 1,0,0,0.

This function has two parameters:

- clock_channel: The SPI clock channel number. For devices with two eTPUs, this parameter should be assigned a value of 1-30 for eTPU_A and 65-94 for eTPU_B. For products with a single eTPU, this parameter should be assigned a value of 1-30. Operation is not guaranteed if channels 0, 31, 64, or 95 are used as clock_channel.
- tx_data: This is the data to be transmitted. It will be placed in the data register to be accessed by the SPI function. Note that although tx_data is a 32-bit parameter, the eTPU SPI function is capable of transferring between 1 and 24 bits. The number of bits to be transferred is defined by parameter transfer_size in the fs_etpu_spi_init function.

It is imperative that the function is allowed to finish sending/receiving data before the channel is used to send/receive data again; otherwise, the sent/received data may be corrupted. The function sends interrupt and DMA requests when it has finished processing.

5.3 SPI Retrieve Received data

Function fs_etpu_spi_get_data is used to retrieve data which the SPI function has received.

This function has only one parameter:

- clock_channel: The SPI clock channel number. For products with two eTPUs, this parameter should be assigned a value of 1-30 for eTPU_A and 65-94 for eTPU_B. For products with a single eTPU, this parameter should be assigned a value of 1-30. Operation is not guaranteed if channels 0, 31, 64, or 95 are used as clock_channel.

This function retrieves the number of bits that the function was configured to receive by fs_etpu_spi_init from the eTPU Parameter RAM.

6   Examples of Function Use

Example code showing use of the eTPU SPI function is available from Freescale for both the MPC5500 and the MCF523*x* families of devices.

6.1 Description

This section describes a simple use of the SPI function and how to initialize the eTPU module and assign the eTPU SPI function to an eTPU channel. Examples are given for the MPC5500 series parts and the MCF523*x* series parts.

6.1.1 MPC5500 Example Code

The example consists of two files:

- spi_ppc_example.h
- spi_ppc_example.c

File spi_ppc_example.c contains the main() routine. This routine initializes the MPC5554 device for 128 MHz CPU operation and initializes the eTPU according to the information in the my_etpu_config struct (stored in file spi_ppc_example.h). The pins used in this example are configured for eTPU operation, and then the SPI function is initialized on channel 3 (ETUA2=RxD, ETPUA3=CLK ETPUA4=TxD). Eight bits of data (0xCC) are then transmitted from the TxD pin and received on the RxD pin. In order to observe the transmitted data being received, the TxD pin should be shorted to the RxD pin.

6.1.2 MCF523*x* Example Code

The example consists of four files:

- spi_mcf_example.h
- spi_mcf_example.c
- spi_mcf_example_global_etpu_gct.h
- spi_mcf_example_global_etpu_gct.c

File spi_mcf_example.c contains the main() routine. This routine initializes the MCF5235 device for 100 MHz CPU operation and initializes the eTPU according to the information in the my_etpu_config struct (stored in file spi_mcf_example_global_etpu_gct.c). The SPI function is initialized on channel 9 (ETUA8=RxD, ETPUA9=CLK ETPUA10=TxD). Eight bits of data (0xCC) are then transmitted from the TxD pin and received on the RxD pin. In order to observe the transmitted data being received, the TxD pin should be shorted to the RxD pin.

7 Summary

This applications note provides the user with a description of the Serial Peripheral Interface (SPI) eTPU function and details on how to use the function. The simple C interface functions for the eTPU SPI function that are provided in the API enable easy implementation of the SPI function in user applications. The functions are targeted for the MPC5500 and the MCF53*x* families of devices, but they can be used with any device that has an eTPU.

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

freescale™
semiconductor

AN2847
Rev. 0
10/2004