Halting and Re-Starting the Queued Serial Peripheral Interface on Modular Microcontrollers

By Sharon Darley

Austin, Texas

Introduction

Halting the queued serial peripheral interface (QSPI) on modular microcontrollers (MCUs) before the end queue pointer is reached requires a special sequence to ensure that the current serial transfer completes and the QSPI halts in a known state on a boundary between two queue entries. Although the CPU can immediately disable the QSPI by clearing the SPE bit in the SPCR1 register, this is not a good practice since the QSPI could shut off in the middle of the current serial transfer. A loss of data from the current serial transfer could result, causing confusion for an external SPI device.

One example in which it would be necessary to halt the QSPI before the current queue pointer reaches the end queue pointer would be if a noise pulse caused one of the slave devices to lose synchronization with the QSPI.

Another example would be if an emergency condition needed to override the transfer and abort it. If the transfer halts on a known boundary condition (for instance, the current queue pointer points to the next queue entry to be transmitted), the master and slave devices have a
much better chance of successfully resynchronizing with each other than they do if a serial transfer is suddenly aborted.

**Disabling the QSPI**

The QSPI disables itself when it finishes its entire transmission (for example when the current queue pointer reaches the end queue pointer) and wraparound mode is disabled. However, to halt and disable the QSPI before it reaches the end of the transmission queue requires action by the CPU.

Safely disabling the QSPI involves three bits in three different registers:

- HALT bit in the SPCR3 register
- HALTA bit in the SPSR register
- SPE bit in the SPCR1 register

**HALT** — The HALT bit is located in the SPCR3 register. When the CPU sets this bit to a 1, the QSPI finishes executing the current serial transfer and then halts. While halted, if the command control bit (CONT of the QSPI RAM) for the last command was asserted, the QSPI continues driving the peripheral chip select pins with the value designated by the last command before the halt. If CONT was clear, the QSPI drives the peripheral chip-select pins to the value in register PORTQS.

If HALT is asserted during the last command in the queue, the QSPI completes the last command, asserts both HALTA and SPIF, and clears SPE. If the last queue command has not been executed, asserting HALT does not set SPIF nor clear SPE. QSPI execution continues when the CPU clears HALT.

**HALTA** — The QSPI asserts the HALTA flag in the SPSR register after it has come to an orderly halt. If HMIE in SPCR3 is set, the QSPI sends an interrupt request to the CPU when HALTA is asserted. The CPU can clear HALTA by reading SPSR with HALTA set and then writing a 0 to HALTA.
SPE — Setting the SPE bit in the SPCR1 register enables the QSPI, while clearing the SPE bit disables the QSPI. The CPU can disable the QSPI at any time by clearing SPE. The QSPI clears SPE when it reaches the end queue pointer and is not in wraparound mode, thus disabling itself. When the SPE bit is clear, the QSPI pins are controlled by the PORTQS and DDRQS registers.

Executing this sequence of events halts and disables the QSPI:

1. Assert the HALT bit in SPCR3.
2. Poll the HALTA bit in SPSR until the QSPI sets it.
3. Clear the SPE bit in SPCR1 (if this bit is not cleared, the QSPI will still halt, but it will not return control of its pins to the CPU).

To restart the QSPI:

1. Read HALTA in its asserted state and then clear it to a 0.
2. Set the SPE bit, if it was cleared in the halting sequence.

Example

This example illustrates how to halt and re-start the QSPI. To observe this example working, connect either the MOSI or SCK pin to an oscilloscope or logic analyzer. Then, run the program and observe the bursts of activity and inactivity of the QSPI.

The example first initializes the QSPI in the wrap-around mode. Then, the program waits in a short delay loop to make observation on an oscilloscope easier. Next, the program halts and disables the QSPI as described above and waits in another delay loop. Finally, the program re-enables the QSPI as described above. The program continuously repeats the cycle of halting and then re-enabling the QSPI.

The CPU16 code was assembled with P&E Microcomputer System’s IASM16 assembler, and the CPU32 code was assembled with P&E Microcomputer System’s IASM32 assembler.
CPU32 Code

SPCR1 EQU $FFFC1A
PORTQS EQU $FFFC15
PQSPAR EQU $FFFC16
DDRQS EQU $FFFC17
SPSR EQU $FFFC1F
SPCR0 EQU $FFFC18
SPCR2 EQU $FFFC1C
SPCR3 EQU $FFFC1E
SYNCR EQU $FFFA04
SYPCR EQU $FFFA21
TXDRAM EQU $FFFD20
CMDRAM EQU $FFFD40

ORG $400 ;begin program at $400

INIT_SIM Move.B #$7F,(SYNCR).L ;increase clock speed
CLR.B (SYPCR).L ;disable software watchdog

INIT_QSPI ANDI.W #$7F,(SPCR1).L ;Clear the SPE bit to disable QSPI.
ANDI.B #$00,(SPSR).L ;read and clear flags in SPSR
MOVE.B #$7B,(PORTQS).L ;Assign all pins to the QSPI.
MOVE.B #$7B,(PQSPAR).L ;Signal lines except for MISO are
MOVE.W #$7E,(DDRQS).L ;outputs.
MOVE.W #$8002,(SPCR0).L ;Configure the QSPI as master, select
MOVE.W #$8000,(SPCR1).L ;8 data bits per transfer, set the
MOVE.B #$4F00,(SPCR2).L ;inactivestateofSCKaslow, capture
MOVE.B #$00,(SPCR3).L ;data on the leading edge of SCK, baud
MOVEA.L #DATA,A0 ;rate is 4.19 MHz
MOVEA.L #TXDRAM,A1 ;Point A1 to the transmit data RAM.
MOVEA.L #CMDRAM,A2 ;Point A2 to the command RAM
MOVE.W #$10,D0 ;Set a counter to count down from 16
;($10), since
;there are 16 queue entries to fill.

CLR.L D1

LOOP

MOVE.B (A0)+,D1 ;Begin a loop to fill the transmit RAM.
MOVE.W D1,(A1)+ ;Store the data right-justified.
MOVE.B #$00,(A2)+ ;fill command RAM: chip selects active
SUBI.W #$01,D0 ;low
BNE LOOP ;Fill next queue entry if not done
MOVE.W #$8000,(SPCR1).L ;Begin operation by setting the SPE
bit.
Disabling the QSPI

MAINLP
MOVE.W #$FFFF,D0 ;set a wait loop so that QSPI operation

WAITLOP
SUBI.W #$01,D0 ;can be observed on an oscilloscope
BNE WAITLOP

TEST
MOVE.B #$01,(SPCR3).L ;set HALT = 1
ANDI.B #$20,D0 ;wait until QSPI sets HALTA flag
BEQ TEST ;check to see if the QSPI is halted
ANDI.W #$7F,(SPCR1).L ;clear SPE -- disable QSPI
MOVE.W #$FFFF,D0 ;wait loop so that break in QSPI
WAI2
SUBI.W #$01,D0 ;transmission can be observed
BNE WAI2 ;on an oscilloscope
MOVE.B #$00,(SPCR3).L ;clear HALT
MOVE.W #$8000,(SPCR1).L ;set SPE -- re-enable QSPI
ANDI.B #$00,(SPSR).L ;clear HALTA flag -- re-start QSPI
BRA MAINLP

DATA
DB 16 ;memory used to fill transmit RAM

CPU16 Code

SPCR1 EQU $FC1A
PORTQS EQU $FC15
POQSPAR EQU $FC16
DDRQS EQU $FC17
SPSR EQU $FC1F
SPCR0 EQU $FC18
SPCR2 EQU $FC1C
SPCR3 EQU $FC1E
SYNCR EQU $FA04
SYPCR EQU $FA21
TXDRAM EQU $FD20
CMDRAM EQU $FD40

ORG $200 ;begin program at $400, immediately
;after the exception table

INIT_SIM
LDAB #$0F
TBEK
TBYK
TBZK
LDD #$7F00
STD SYNCR ;increase clock speed
CLR SYPCR ;disable software watchdog

INIT_QSPI
LDD SPCR1
ANDD #$7F
STD SPCR1 ;Clear the SPE bit to disable QSPI.
LDAB SPSR
ANDB #$00
STAB SPSR ;read and clear flags in SPSR
LDAB #$7B
STAB PORTQS ;define initial states of chip
STAB PQSPAR ;selects/SCK
LDAB #$7E
STAB DDRQS ;Assign all pins to the QSPI.
LDD #$8002
STD SPCR0 ;Signal lines except for MISO are
;outputs.

LDD #$4F00
STD SPCR2 ;Configure the QSPI as master, select
;8 data bits per transfer, set the
;inactive
;state of SCK as low, capture data on
;leading edge of SCK, baud rate is 4.19
;MHz

CLRB
TBXK
LDX #DATA ;Point X to the data to be transmitted.
LDY #TXDRAM ;Point Y to the transmit data RAM.
LDZ #CMDRAM ;Point Z to the command RAM
LDE #$10 ;Set a counter to count down from 16
;($10), since there are 16 queue
;entries to fill.

LOOP LDD 0,X
STD 0,Y ;Begin a loop to fill the transmit RAM.
AIX #$02 ;Store the data right-justified.
AIY #$02
CLR B
STAB 0,Z
INCZ ;fill command RAM: chip selects active
;low
SUBE #$01 ;Subtract one from the counter
BNE LOOP ;Fill next queue entry if not done
LDD #$8000
STD SPCR1 ;Begin operation by setting the SPE
;bit.

MAINLP LDE #$FFFF ;set a wait loop so that QSPI operation
;can be observed on an oscilloscope

WAITLOP SUBE #$01
BNE WAITLOP
LDAB #$01
STAB SPCR3 ;set HALT = 1

TEST LDAB SPSR ;wait until QSPI sets HALTA flag
ANDB #$20
BEQ TEST ; check to see if the QSPI is halted
LDAB SPCR1
ANDB #$7F
STAB SPCR1 ; clear SPE -- disable QSPI
LDE #$FFFF ; wait loop so that break in QSPI

WAI2

SUBE #$01 ; transmission can be observed on an oscilloscope
BNE WAI2
CLRB
STAB SPCR3 ; clear HALT
LDD #$8000
STD SPCR1 ; set SPE -- re-enable QSPI
CLRB
STAB SPSR ; clear HALTA flag -- re-start QSPI
BRA MAINLP

DATA
DB 16 ; memory used to fill transmit RAM