Serial Peripheral Interface Device Emulation Routine for the MC68340

Colin Mac Donald, Freescale., East Kilbride, Scotland
Philippe Alves, ENSERG, France

INTRODUCTION

The Serial Peripheral Interface Device Emulation Routine (SPIDER) is an assembly language program that allows an MC68340 processor to emulate a master-mode Serial Peripheral Interface (SPI) without additional hardware. Besides a variety of SPI modes, the program also supports the transfer of blocks of data, minimising the amount of data handling performed by the calling software.

Designed for simple, low-throughput SPI systems, the SPIDER is a CPU32 supervisor mode program that can be interrupt driven. This optimises the SPI transfer rate to M68340 bus bandwidth ratio in slow SPI systems. As no additional hardware is required, the SPIDER is ideal for cost-sensitive and part count-sensitive applications.

The SPIDER requires only three MC68340 PORTA pins for its operation, discounting slave selects outputs and interrupt inputs. Furthermore, the program uses header definitions to allow selection of processor pins for SPIDER functions. This flexibility and low-pin count requirement means the SPIDER can be easily adapted to many MC68340 configurations. Also, although the code is non-reentrant, it is statically relocatable and therefore adaptable to a particular system's memory map.

This application note describes the SPIDER program, its test code, its configuration and its operation. Although designed and written for the MC68340, the SPIDER should be easily adaptable to other members of the M68300 family.

Features

- Requires no additional hardware
- Requires a minimum of 2 PORTA pins (uni-directional)
- Requires a minimum of 3 PORTA pins (bi-directional)
- Requires less than 1k byte of ROM
- Easily adapted to different MC68340 configurations
- Block or single word transfers
- Block transfer support minimises calling software overhead
- Eight- or sixteen-bit SPI word sizes
- Choice of clock phase
- Choice of clock polarity
- Up to two slave selects
- Can be interrupt driven to minimise bus bandwidth impact
- Choice of IRQ source
- Conditional assembly for compact code
- Statically relocatable

1. SOFTWARE OVERVIEW

SPIDER software was developed on a Freescale MC68340 evaluation system and was tested using a Freescale MC68HC05C4 evaluation module. This application note first describes the SPIDER software followed by the MC68340EVS and MC68HC05EVM test/demonstration code. A flow chart showing how the SPIDER and its MC68340 test software interrelate is shown in Figure 1.

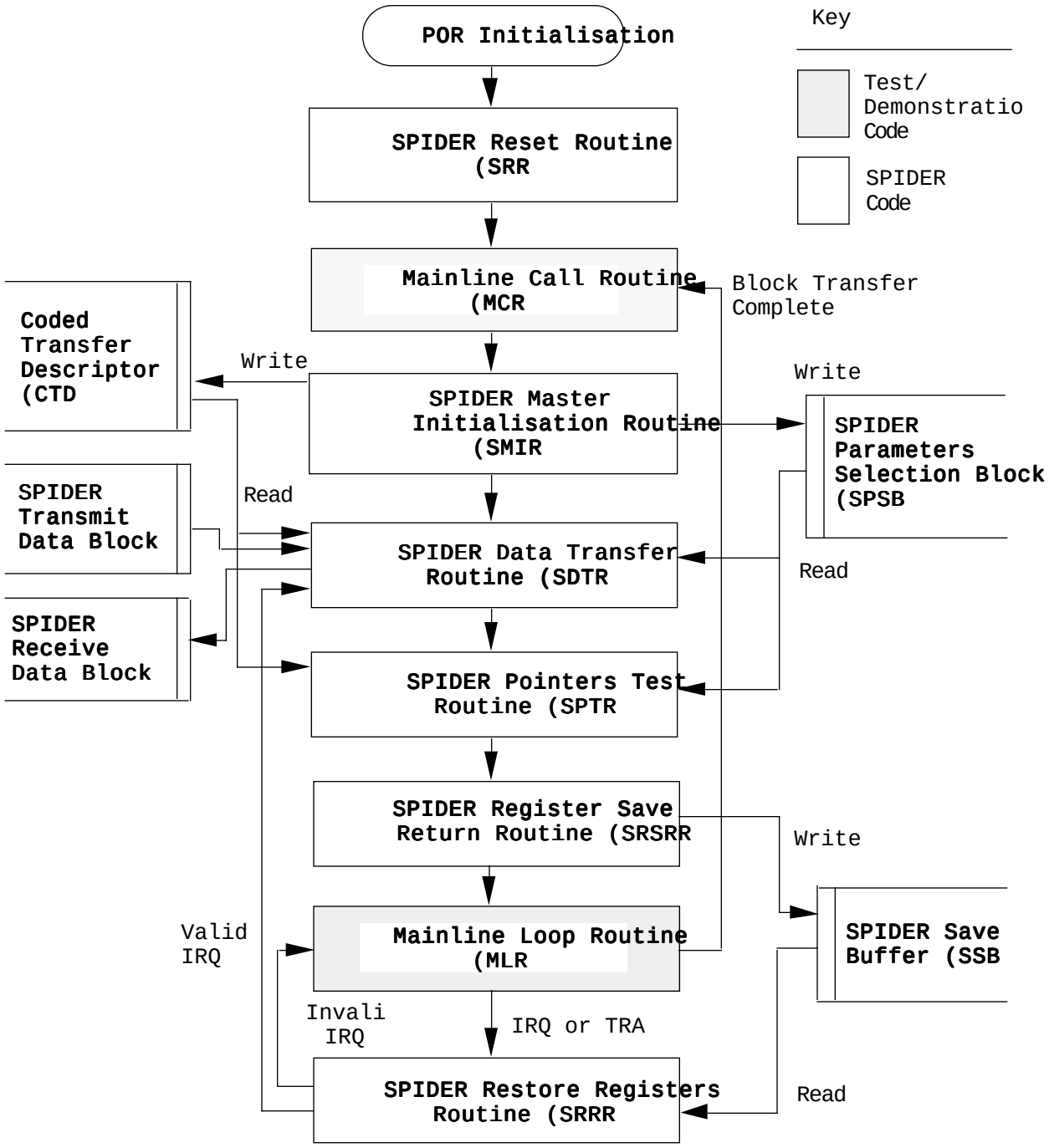

Figure 1. SPIDER and MC68340 Test Code

2. SPIDER SOFTWARE

The SPIDER software comprises six interdependent routines that can be divided into two groups (Figure 1). The first group, consisting of one routine, the SPIDER Reset Routine, is executed only once after reset. This is described in section 2.1. The second group, comprising the other five routines, is the run-time software that performs SPI transfers. This is described in section 2.2.

2.1 SPIDER Reset Routine

The SPIDER Reset Routine (SRR) shown in Figure 2 performs initialization tasks that require to be executed only once after reset. It returns control to the main line after completing these tasks:

- Configure MC68340 hardware for the SPIDER according to SPIM and DDRAM variables
- Initialize MC68340 Tgate1 as the SPIDER Interrupt Source (SIS) if TIMIRQ is set to one
- Set the CPU32 SR to SPIDER Status Register (SSR)
- Put the address of the SPIDER Register Restore Routine (SRRR) into the correct location in the exception vector table
- Clear the SPIDER running flag (SRFlag).

2.2 SPIDER Run-Time Software

Every subroutine apart from the SRR, is considered part of the SPIDER run-time code. These routines send and receive data blocks with sizes defined in the SPIDER Parameters Selection Block (SPSB). After each word is sent the SPIDER returns to the mainline code where it is called again either by a TRAP instruction or by the SPIDER IRQ SOURCE until the block transfer is complete.

2.2.1 SPIDER Parameters Selection Block

The location of the SPSB is defined by the SPSAD variable. Calling software defines the SPI transfer parameters by writing to this block before calling the SPIDER. The contents of the SPSB will be read by the SPIDER Master Init Routine (SMIR) and used to build a Coded Transfer Descriptor (CTD) using program header parameters.

The first two parameters, CPOL and CPHA, control the SPI signal relationships in the same manner as these bit fields do in the MC68HC05C4 SPCR. Additionally, the SPIDER drives the active SS to an inter-word level dependent on CPHA. If CPHA is high the SS inter-word level will be low, and vice-versa.

The SPSB is programmed as follows:

CPOL : Clock Polarity

- CPOL = $0 : Marking clock state is low
- CPOL = $1 : Marking clock state is high

CPHA : Clock Phase

- CPHA = $0 : SS is high between words, first clock edge latches
- CPHA = $1 : SS is low between words, clock pulse latches

SSX: Slave 1 or Slave 2

- SSX = $0 : Select slave 1
- SSX = $1 : Select slave 2

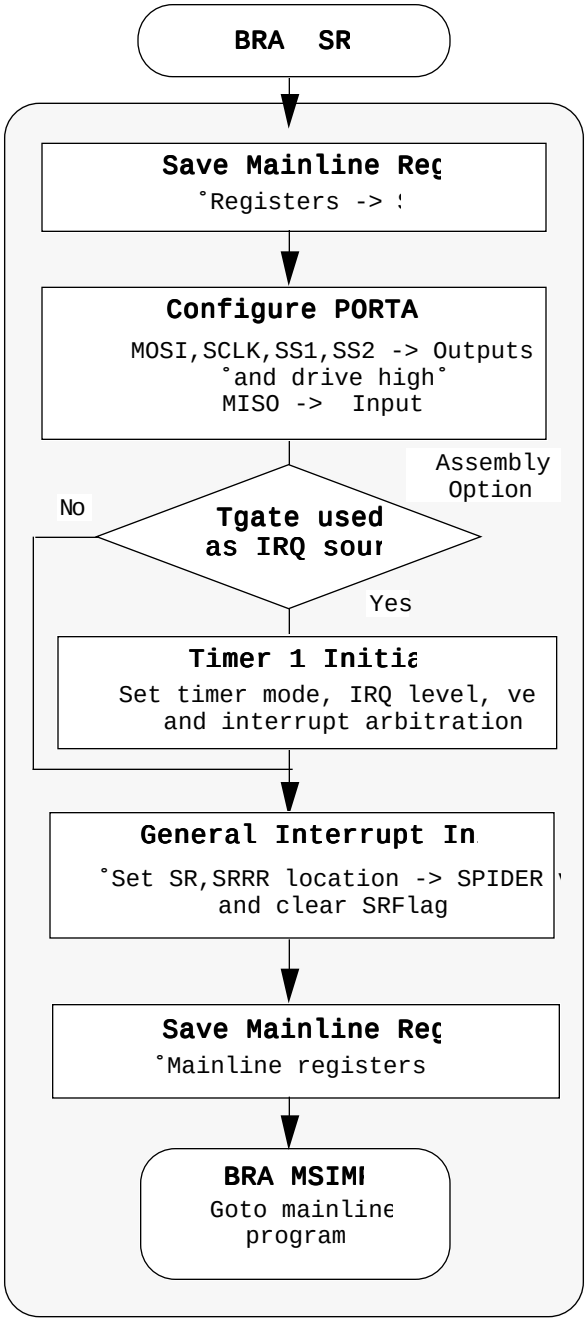

Figure 2. SPIDER Reset Routine

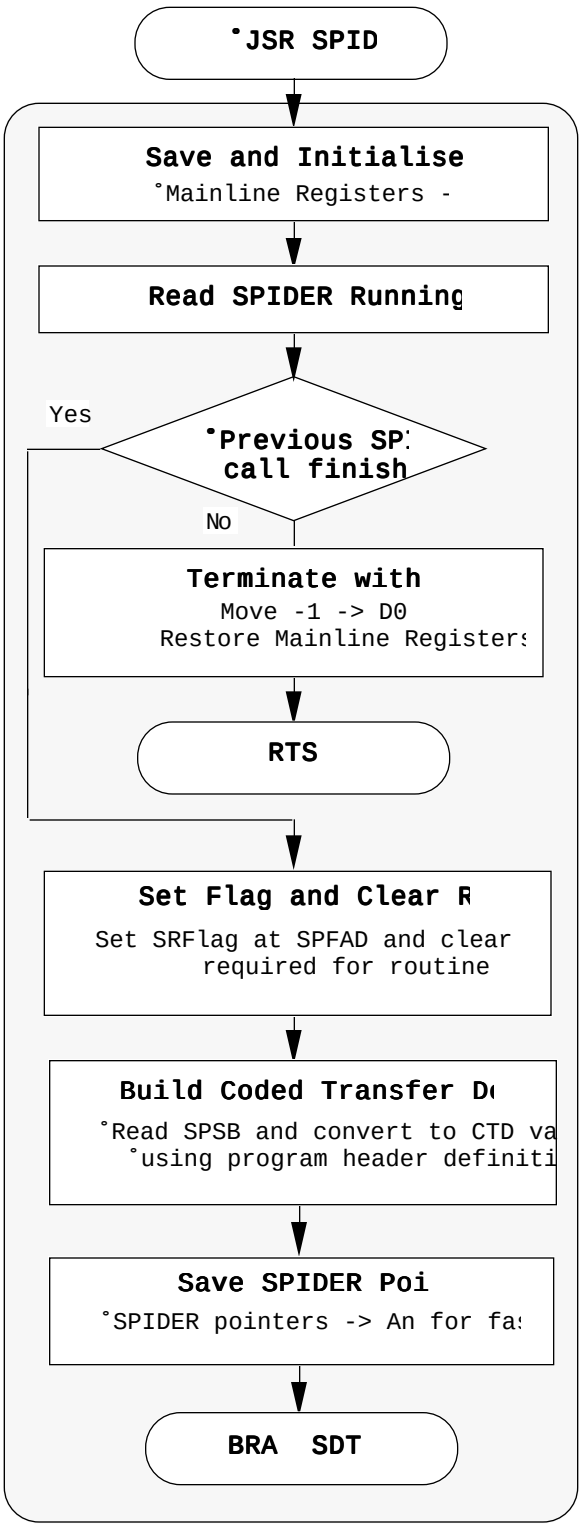

Figure 3. SPIDER Master Initialisation Routine

DSIZ: Size of SPI word

– DSIZ = $08 : 8-bit words will be transferred
– DSIZ = $10 : 16-bit words will be transferred

ADST1 : Transmit block start address (32 bits)

ADST2 : Receive block start address (32 bits)

SIZ1 : Size of transmit block (32 bits)

SIZ2 : Size of receive block (32 bits)

To improve program readability, labels are used to access individual SPSB fields. The relationship between SPSB parameters and their labels is shown in Table 1.

Table 1: SPIDER Parameter Selection Block

SPIDER PARAMETERS SELECTION BLOCK			
Offset (Hex)	Offset Mnemonic	Parameter	Description
0	SP_CPOL	CPOL	Clock Polarity
1	SP_CPHA	CPHA	Clock Phase
2	SP_SSX	SSX	Slave Select 1 or 2
3	SP_DSIZ	DSIZ	SPI word size 8- or 16-bits
4	SP_ADST1	ADST1	Transmit Block Start Location
8	SP_ADST2	ADST2	Receive Block Start Location
C	SP_SIZ1	SIZ1	Transmit Block Size
10	SP_SIZ2	SIZ2	Receive Block Size

2.2.2 SPIDER Master Init Routine

The SMIR has three functions. Firstly, since the SPIDER is not a reentrant routine, the SMIR checks the SPIDER running flag, SRFlag, at location SPFAD, to ensure all previous calls have finished. If it is set, the initialization is stopped, D0 is set to minus one (error status) and control is returned to the mainline program.

If the SRFlag is clear, then the SMIR builds a CTD. A CTD is a 5-byte coded version of the SPSB that the SPIDER Data Transfer Routine (SDTR) uses to perform the transfer. Figure 3 shows the flow of the SMIR and Figure 4 shows how a Coded Transfer Descriptor is built by the SMIR from data in the SPSB.

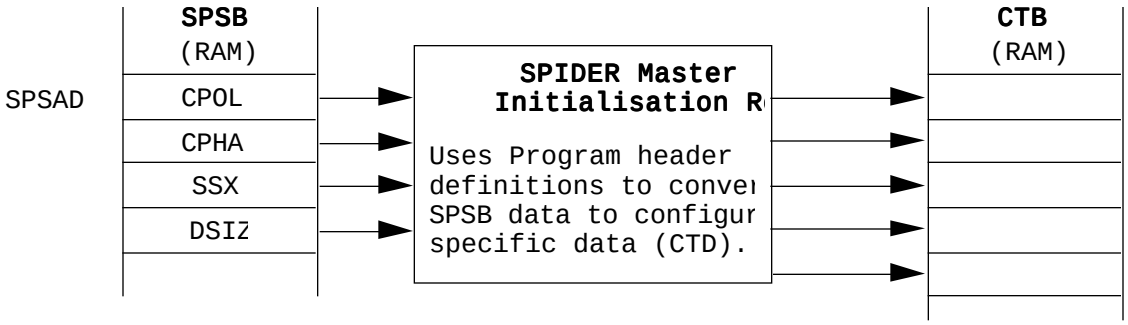

Figure 4. Coded Transfer Descriptor Build

2.2.3 SPIDER Data Transfer Routine

The SPIDER Data Transfer Routine (SDTR) follows either the SMIR for the first word transfer or the SRRR for subsequent transfers. Its main function is to send data bits on MOSI line, to read data bits on MISO line, and to generate the clock. The different steps of this routine are defined as follows:

– Initializes MC68340 registers required for the data transfer
– Drives the serial clock pin according to CPOL
– Selects SPI slave 1 or slave 2
– Reads next word from transmit block for transfer
– Increments transmit block pointer, ADST1
– Decrements transmit block size, SIZ1
– Generates the SPI clock
– Performs bi-directional transfer of one SPI word (8 or 16 bits)

If a PORTB IRQ line is selected as the SPIDER Interrupt Source then this line is redefined as a general purpose input and the interrupt mask in the CPU32 status register is set to seven. This allows detection of a SIS negation pulse by the SPIDER Pointers Test Routine (SPTR), which is necessary to prevent one SIS assertion causing multiple SPIDER transfers. The SDTR program flow is shown in Figure 6.

2.2.4 SPIDER Pointers Test Routine

If a PORTB IRQ line is used as the SIS, then the first task performed by the SPTR is to detect a high level on SIS. Only when a high level is found will flow proceed. The system designer must therefore ensure the following two conditions. Firstly, the slave device should negate SIS for a suitable period and secondly a software watchdog should be used to prevent system lockup if an SPI failure occurs. Flow proceeds by redefining the SIS line as an IRQ input, ready for the next transfer.

Next, the receive block size, SIZ2, is tested. If non-zero, the SPI word received by the SDTR is written to the receive block, the receive block pointer is incremented and SIZ2 decremented. The routine then tests the state of both SIZ1 and SIZ2. When both are equal to zero the block transfer is finished and the SRFlag is cleared. If the receive block size is larger than the transmit block size, the SPIDER will continue to send null data, when SIZ1 reaches zero, in order to keep receiving. Finally, the active SS is driven to an inter-word level depending on CPHA. Figure 5 shows the data flow of a typical SPIDER exchange and Figure 7 shows the SPTR program flow.

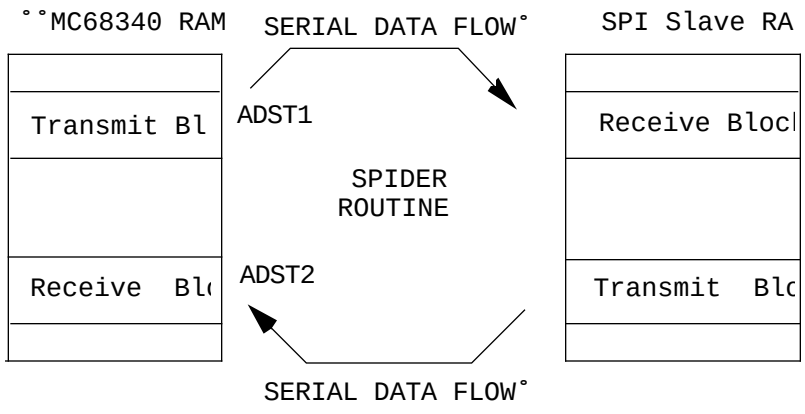

Figure 5. Typical SPIDER Data Block Exchange

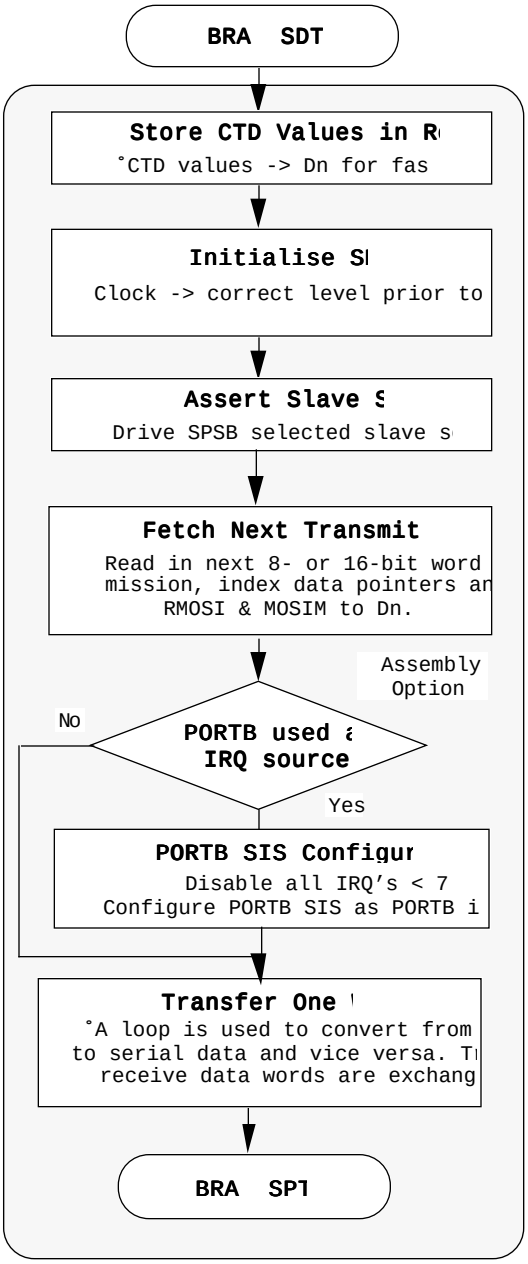

Figure 6. SPIDER Data Transfer Routine

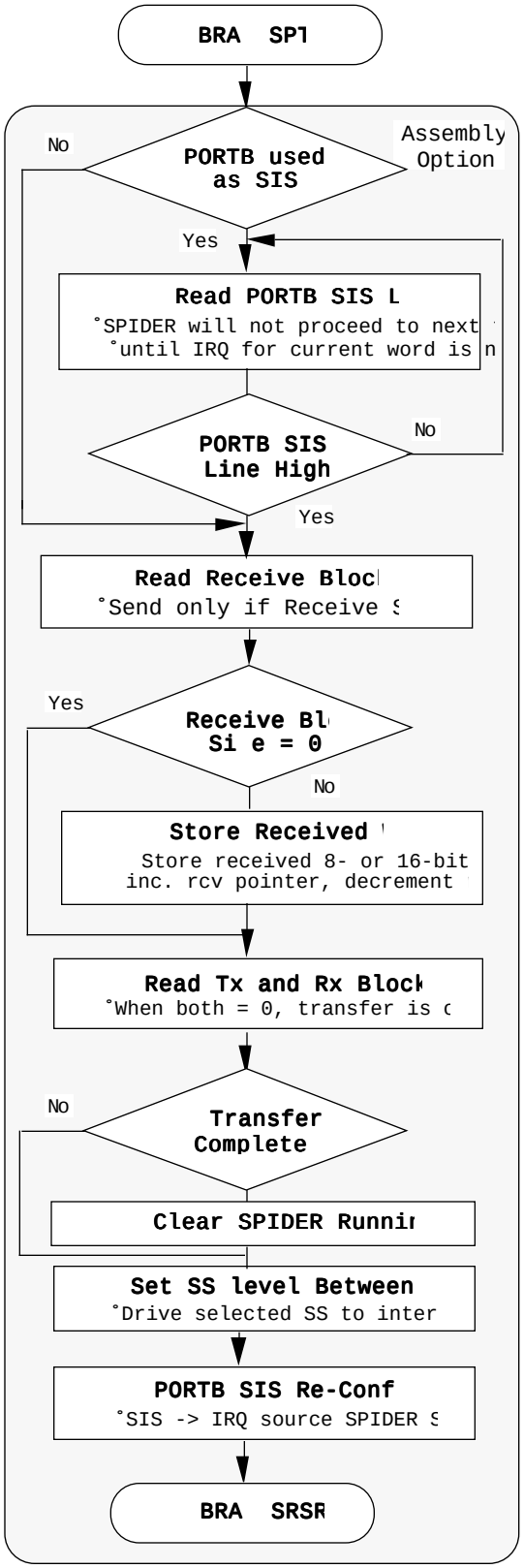

Figure 7. SPIDER Pointers Test Routine

2.2.5 SPIDER Register Save and Return Routine

The SPIDER Register Save and Return Routine (SRSRR) comprises the following tasks:

– Clears the SRFlag on block transfer completion
– Stores SPIDER CPU register values in the SPIDER Save Buffer (SSB)
– Resets Timer 1 IRQ bit if TIMIRQ is set
– Restores mainline register values from the stack
– Places the return error status in D0 (first transfer only)
– Executes the correct return code (either RTS or RTE)

Figure 8 shows the SRSRR program flow.

2.2.6 SPIDER Register Restore Routine

If PTRIRQ or TIMIRQ is set, an assertion of the SIS causes the SPIDER Register Restore Routine (SRRR), Figure 9, to run. Alternatively, a TRAP exception is used to jump to the SRRR. This routine stacks the values of the address and data registers and restores the SPIDER values from the SSB. After checking the SRFlag is set, control is passed to the SDTR. If the SRFlag is zero, the exception is deemed non-valid, mainline register values are restored and control is passed to the mainline program.

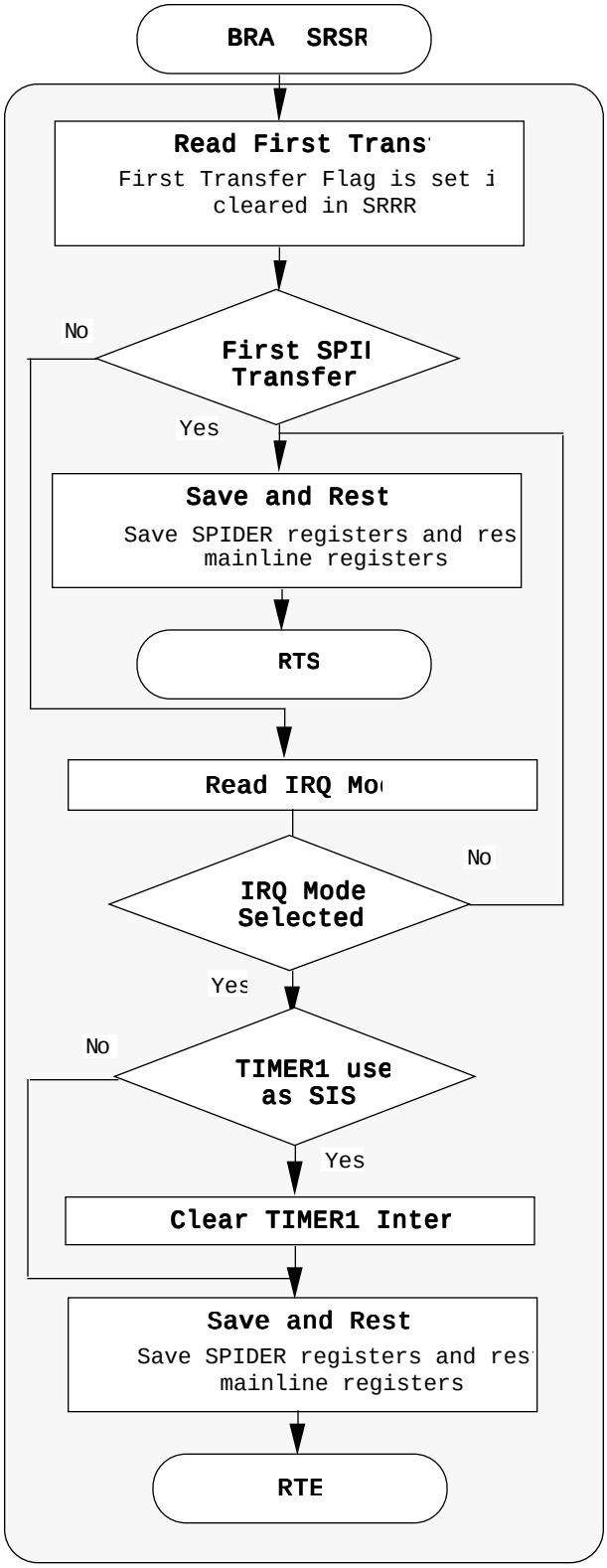

Figure 8. SPIDER Register Save and Return Routine

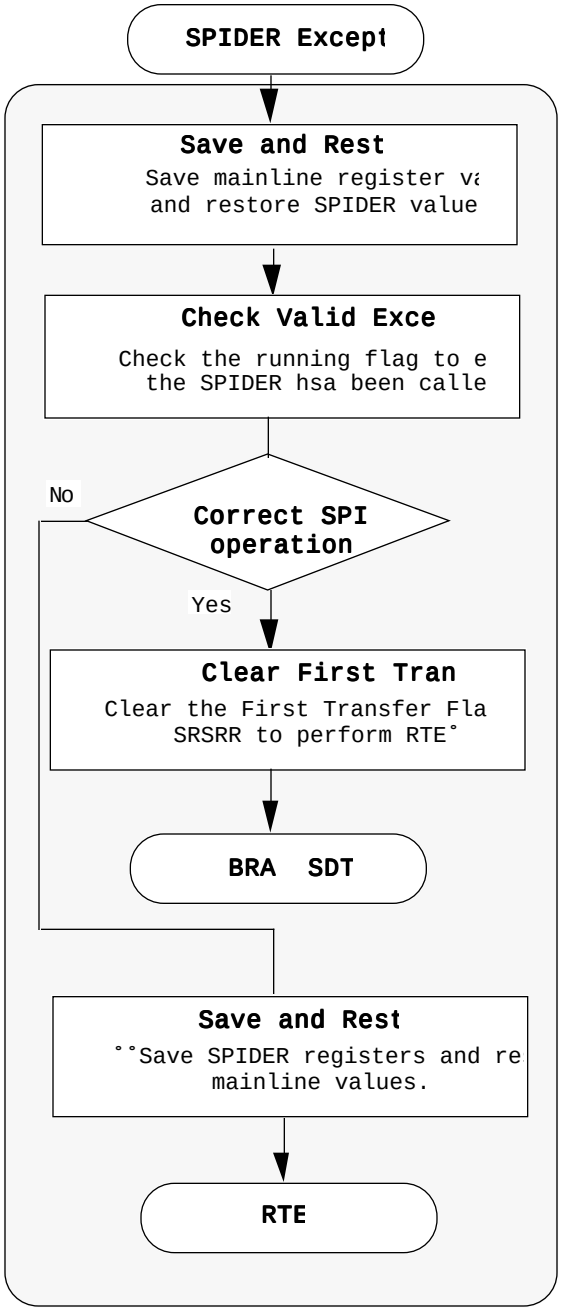

Figure 9. SPIDER Register Restore Routine

3. TEST SOFTWARE

The slave SPI device used to test (and demonstrate) the SPIDER software is the MC68HC05C4. The test code therefore consists of MC68340 based software, the Mainline Simulation Program (MSIMP) and the MC68HC05C4 SPIDER test program (STP05). These are described in sections 3.1 and 3.2.

3.1 Mainline Simulation Program

The purpose of MSIMP is to test the SPIDER by providing a software environment representative of a real system. This code consists of two sections, the Mainline Call Routine (MCR) and Mainline Loop Routine (MLR). Figure 1 shows how the MCR and MLR (indicated by shaded blocks) interface to the SPIDER software.

3.1.1 Mainline Call Routine

The MCR calls the SPIDER Master Program and checks the returned status. If an error status is returned, indicating a previous SPIDER call has not completed, it branches back to the call statement.

3.1.2 Mainline Loop Routine

The Mainline Loop Routine represents a real system's software flow following SPIDER execution and therefore depends on the method used to generate SPIDER exceptions.

If an interrupt source is used, program flow is expected to move to other tasks once the SPIDER has been started. The MLR simulates these tasks via a loop in which data and address registers are modified. Once in the MLR, a slave SPI device indicates that it is ready for the next word transfer by asserting the SIS.

If an interrupt source is not used, the MLR uses traps to call the SPIDER for every word in the transfer block and separates these calls with a long enough delay to guarantee the slave is ready.

3.2 MC68HC05C4 SPIDER Test Program

The MC68HC05C4 SPIDER Test Program (STP05) runs on an MC68HC05 EVM and tests the SPIDER's data transmission by sending received data to a storage area and to a VT100 terminal through the HC05C4's SCI. The VDU allows visual checking of the data integrity. STP05 allows the SPIDER's data reception to be checked by sending predefined blocks of data. STP05 is made up of three routines, the HC05 SPIDER Test Initialization (05STI), the HC05 SPIDER Conversion Routine (05SCR) and the HC05 SPIDER Interrupt Routine (05SIR).

As the HC05C4's SPI cannot transfer 16-bit SPI words, this test program only checks byte transfers.

3.2.1 MC68HC05 SPIDER Test initialization

This task initializes the SCI, SPI and PORTA on the MC68HC05C4. A pin from PORTA is used as the SPIDER interrupt source (SIS) on the MC68340 system. The 05STI also enables SPI interrupts and sets the HC05C4 SPI IRQ vector to point to the 05STIR.

3.2.2 MC68HC05 SPIDER Test Conversion Routine

The 05STCR routine performs the following functions.

– Sets up block transfer parameters (1*)
– Negates SIS for a PORTB SIS
– Polls SPI IRQ flag (2*)
– Negates SIS for a PORTB SIS

– Polls SCI transmitter empty bit
– Transfers last received byte to SCI (VT100 display)
– Checks send and receive block remainder sizes
– Branches to 1* or 2* depending on block sizes

3.2.3 MC68HC05 SPIDER Test IRQ Routine

The 05STIR runs each time a byte has been completely received on the SPI and the SPIF bit in the SPSR has been set. This routine performs the following functions:

– Uses the first two received bytes as sizes of receive and transmit data blocks
– Tests flags and pointers
– Stores received SPI data block in HC05 RAM
– Transmits data from a predefined data area
– Asserts the SPIDER IRQ Source

Figure 10 shows the SPIDER Test Program implemented on the MC68HC05EVM.

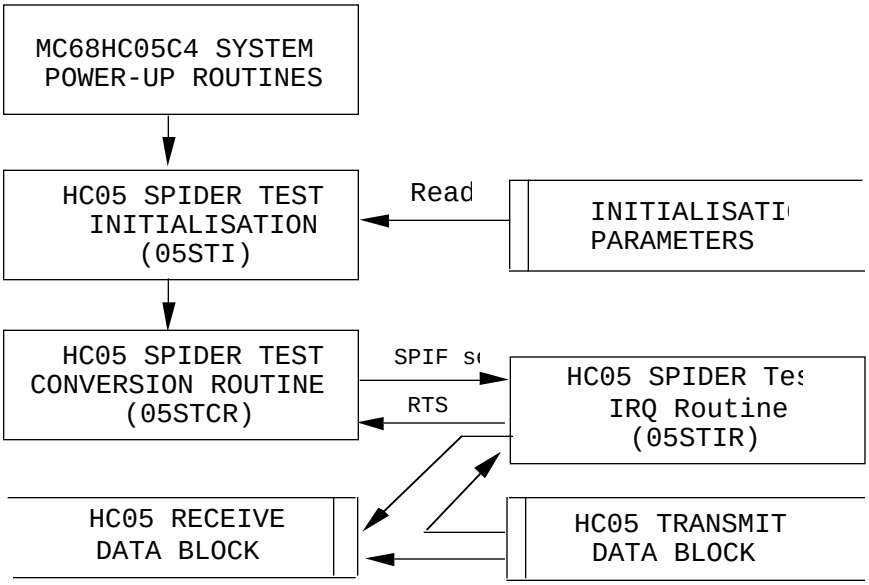

Figure 10. MC68HC05 SPIDER Test Flow

3.3 MC68HC05 SPIDER Test Configuration

The SPIDER is tested by the MCR setting ADST1 and ADST2 equal. An incoming data block therefore overwrites the transmitted data block. By using the same technique on the HC05 system two distinctly different blocks of the same size can be swapped indefinitely. The 05STCR outputs received data on the HC05 SCI and so the blocks can be visually inspected.

This test is performed using a variety of MC68340 port pins as SPI functions and for all supported SPI modes.

4. SPIDER CONFIGURATION

4.1 Configuration Parameters

The SPIDER can be easily adapted to a particular MC68340 system by assigning values to the SPIDER Configuration Parameters in the program header (Table 2).

Eight of these parameters are used to map MC68340 pins to SPI functions. SPIM, DDRAM, MOSIM, CLKPIN, SL1PIN and SL2PIN all select PORTA pins by the position of set bits in the variable's value. So, setting CLKPIN to $80 selects PORTA 7 for the SPI clock. PORTA selection also requires a further two parameters to be defined; RMOSI and RMISO. Their value can be derived using the programming guide in Table 2.

Table 2: SPIDER Configuration Parameters

SPIDER CONFIGURATION PARAMETERS		
Parameter Mnemonic	Name of Parameter	Programming Guide
SIMBA	MC68340 SIM Base Address	Same address as stored in MBAR
SPSAD	SPIDER Parameter Selection Block	Any location in RAM
CTDAD	Coded Transfer Descriptor Address	Any location in RAM
SSBAD	SPIDER Save Buffer Address	Any location in RAM
PTAAD	MC68340 PORTA Data Address	Full address, not an offset
SPFAD	SPIDER Flag Address	Any location in RAM
T1CR	MC68340 Timer 1 Control Register	Enable TG for IRQ, Timer bypass
T1MCR	MC68340 Timer 1 MCR Value	Set IARB bits to unique level
SSR	SPIDER Status Register	Set IRQ mask < chosen SIS level
T1IRQ	MC68340 Timer 1 IRQ Level	Set SPIDER IRQ level for Timer SIS
SPIM	SPIDER Pins Mask	Bits set = SPIDER PORTA pins
DDRAM	MC68340 Data Direction Reg	Bits set = SPIDER PORTA outputs
VECNB	MC68340 SPIDER Vector	Autovector No. for chosen SIS
RMOSI	Rotation for MOSI	Value = (7 – MOSIM Bit No.)
MOSIM	MOSI Mask	Bit set = SPIDER PORTA MOSI pin
RMISO	Rotation for MISO	Value = MISO Bit No.
CLKPIN	SPI Clock Pin	Bit set = SPIDER PORTA CLK pin
SL1PIN	Slave Select No 1 Pin	Bit set = SPIDER PORTA SS1 pin
SL2PIN	Slave Select No 2 Pin	Bit set = SPIDER PORTA SS2 pin
IRQSL	SPIDER Interrupt Selection	Bit set = PORTB SIS.

In systems with a single HC05C4 compatible SPI slave, there are three ways to reduce the number of MC68340 pins required by the SPIDER. Firstly, if the slave and MC68340 are configured for CPHA=1, then the SS pin on the slave can be tied permanently low. This means an SS pin on the MC68340 can be saved if SL1PIN and SL2PIN are set to zero and SPIM and DDRAM are programmed accordingly. Secondly, if the system only requires a uni-directional SPI link then either the MOSI or MISO pin can be otherwise employed. This is achieved through appropriate programming of RMOSI, MOSI, RMISO, DDRAM and SPIM variables. Lastly, as in all SPIDER systems, an MC68340 pin can be freed if TRAPS are used to drive the transfer.

The SPSB also allows the SPSB, CTD, SRFlag and SSB to be placed anywhere in the system memory map as the parameters SPSAD, CTDAD, SPFAD and SSBAD define their start addresses. However, allowances for block size should be made to prevent overlap. The sizes are as follows: SPSB – twenty bytes; CTD – five bytes; SRFlag – one byte; SSB – fifty six bytes.

SIMBA and PTAAD, should simply be set to the addresses of the SIM Base and PORTA Data Register respectively. All remaining SPIDER Configuration Parameters relate to the exception source used to request SPI transfers. These are described in section 4.3, Exception Configuration.

4.2 Conditional Assembly

To minimise the option checking in the program code whilst using only one source file, two header parameters are used for conditional assembly of the SPIDER (Table 3). The assignment of these two variables, PRTIRQ and TIMIRQ, depends on the chosen SIS and is therefore described in section 4.3, Exception Configuration

Table 3: SPIDER Assembly Option Parameters

SPIDER ASSEMBLY OPTION PARAMETERS		
PRTIRQ	PortB IRQ Utilised	Set to one if a PORTB line is used as the SIS
TIMIRQ	Timer 1 IRQ Utilised	Set to one if TIMER1 TGATE is used as SIS

4.3 Exception Configuration

Following a SPIDER call, and the transfer of one SPI word, control returns to the mainline code. All further transfers, until a block transfer is complete, require an exception to jump to SPIDER code. There are three possible sources for the exceptions used to drive the SPIDER, namely Tgate1, a PORTB IRQ line, and TRAP instructions.

If the MC68340 Tgate1 pin is chosen as the SPI IRQ Source, then Timer 1 should be configured for Timer By-pass mode and its interrupt level and interrupt arbitration level should be appropriately set. This is achieved by defining T1CR, T1MCR and T1IRQ. Also, PRTIRQ should be set to zero and TIMIRQ set to one in order to assemble the program correctly. Furthermore, as the rising edge of Tgate generates an interrupt in Timer By-pass mode, the slave SPI device must drive the SIS high to request a transfer.

For selection of a PORTB SIS, IRQSL should identify the particular pin, PRTIRQ should be set to one and TIMIRQ set to zero. The SPIDER will automatically configure this IRQ source for autovector.

The SPIDER can operate with no interrupt source, by using TRAPs to generate exceptions. Here, the system software must be written so that the period between TRAP calls is sufficiently large for the slave device to prepare for the next transfer. To operate in this mode, both PRTIRQ and TIMIRQ should be set to zero.

Two more configuration parameters relate to exceptions. The SSR allows SPI IRQs are processed by setting the interrupt mask on the CPU32 SR the chosen SIS level. The assignment of VECNB depends on the SPIDER exception source. For Tgate, VECNB should be set in the user-defined vector range (i.e., 64–255). For a PORTB SIS, VECNB should reflect the chosen IRQ autovector number. Finally, for TRAP calls, VECNB should be the TRAP instruction vector number.

5. SPIDER CONTROL

The SPIDER has been written to minimise interaction with its controlling software. So, although it has many features and is adaptable to many systems it is still easy to initialize, call and prosecute.

An SPI transfer is initialized by creating an SPSB at the SPSAD. As the program is non-reentrant, one fixed location for SPSBs is sufficient. If used in a system with limited memory, the SPSB can have identical receive and transmit block locations – provided the transmit block is disposable after transmission. For more details on defining an SPSB, see section 2.2.1.

The program is called by BSR or JSR to the SPIDER label. The program puts the call status in D0 before returning to the mainline. Minus one in D0 indicates a previous call has not finished, while zero confirms a valid call with the first transfer completed.

If interrupts are used to request transfers, the transfer will continue at a rate determined by the system firmware. If TRAPS are used, a minimum delay between each TRAP should be used to ensure the slave SPI device can service each word transferred.

6. PERFORMANCE

The clock speed of the SPIDER SPI clock is a function of the MC68340 processor speed and memory speed. With a 16.67 MHz processor running out of synchronous memory, the SPI clock speed is 192 kHz.

The bus bandwidth consumed by the SPIDER for a given transfer block size is a function of the performance of the MC68340 and slave SPI systems. This is best measured empirically.

SPI A.C. timing characteristics for a 16.67 MHz MC68340 running the SPIDER in synchronous memory are given in Table 4, and waveforms are shown in Figures 11 and 12.

Table 4: SPIDER A.C. Timing

Serial Peripheral Interface Device Emulation Routine (SPIDER) Timing				
No.	SPI Characteristic	Symbol	Min	Max
	Operating Frequency	fop	–	195 kHz
1	Cycle Time	tcyc	5µS	–
2	Clock (SCLK) High Time	tw(SCLKH)	2µs	–
3	Clock (SCLK) Low Time	tw(SCLKL)	2µs	–
4	Enable Lead Time	tlead	9µS	–
5	Input Data Setup Time	tsin	100ns	–
6	Input Data Hold Time	thin	700ns	–
7	Output Data Setup Time	tsout	500ns	–
8	Output Data Hold Time	thout	500ns	–
9	Enable Lag Time	tlag	7µs	–

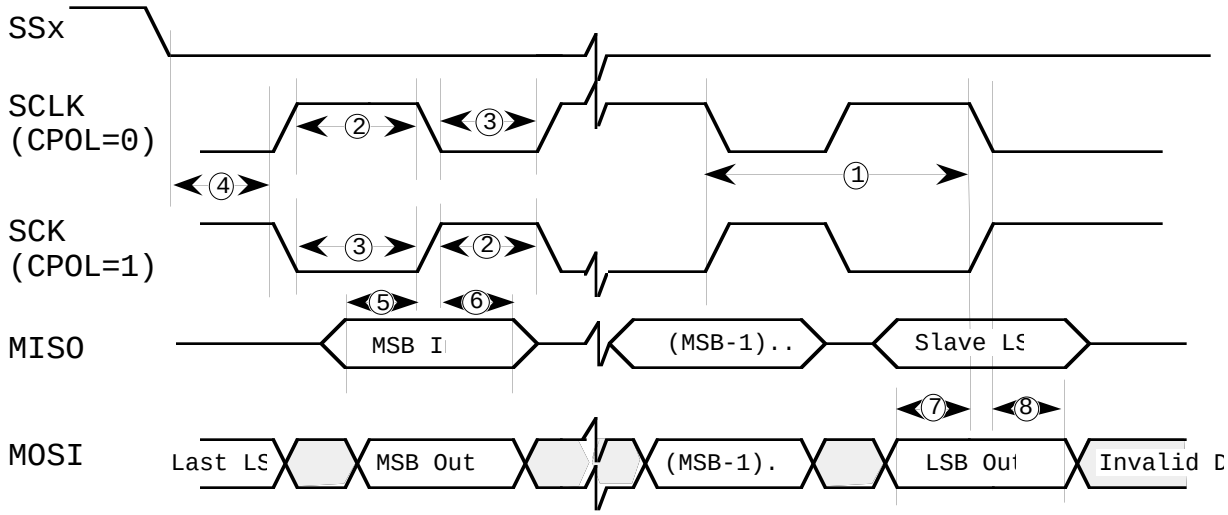

Figure 11. SPIDER A.C. Timing (CPHA = 1)

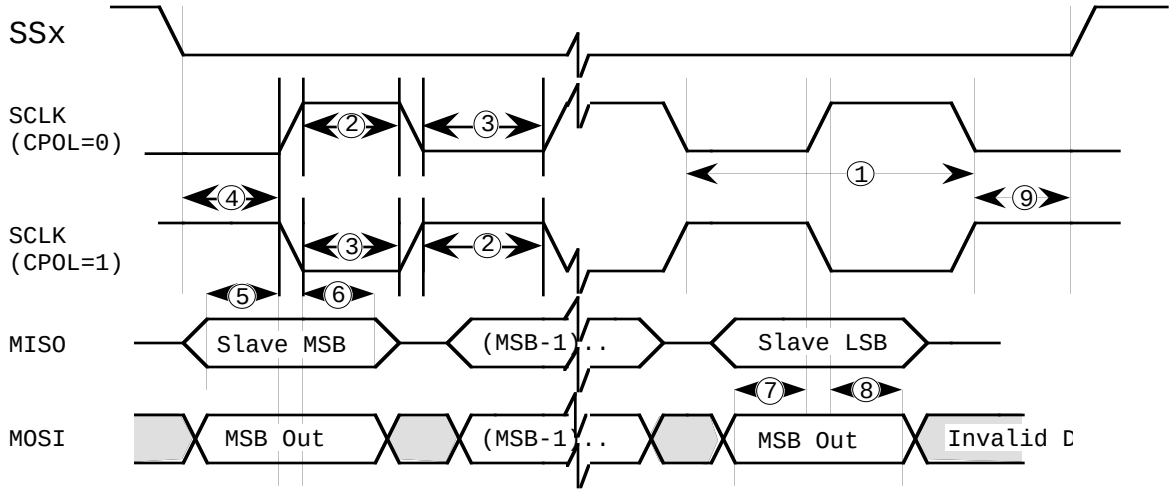

Figure 12. SPIDER A.C. Timing (CPHA = 0)

APPENDIX A. GLOSSARY OF TERMS

CTD	Coded Transfer Descriptor
MCR	Mainline Call Routine
MLR	Mainline Loop Routine
MSIMP	Mainline Simulation Program
SDTR	SPIDER Data Transfer Routine
SDReg	SPIDER Data Register
SPSB	SPIDER Parameters Selection Block
SIS	SPIDER Interrupt Source
SMIR	SPIDER Master Init Routine
SPIDER	SPI Device Emulation Routine
SPSAD	SPIDER Parameter Selection Block Address
SRSRR	SPIDER Register Save and Return Routine
SRR	SPIDER Reset Routine
SRFlag	SPIDER Running Flag
SSB	SPIDER Save Buffers
SRR	SPIDER status register
STP05	MC68HC05C4 SPIDER Test Program
05STI	MC68HC05C4 SPIDER Test Initialisation
05STCR	MC68HC05C4 SPIDER Test Conversion Routine
05STIR	MC68HC05C4 SPIDER Test IRQ Routine.

APPENDIX B. CODE LISTINGS

The listings for SPIDER.SA, MSIMP.SA, STP05.SA follow.

```
*****************************************************************
*******************PROGRAM HEADER********************************
* FILE NAME              : SPIDER.SA
* PROGRAM NAME           : SPIDER (SERIAL PERIPHERAL DEVICE
*                        : EMULATION ROUTINE)
* VERSION                : 1.0
* LAST MODIFIED          : 23-10-91
* LANGUAGE               : VERSAdos M68000 Assembler
* ENVIRONMENT            : MC68340 systems
* PROGRAMMER             : Philippe ALVES
* AUTHOR                 : Colin Mac Donald
* COMPANY                : FREESCALE
* FUNCTION OF PROGRAM    : Implement the SPI function on MC68340
* RELATED FILES          : SPIDER.RME, SPxxx.COM, SPxxx.DAT
*                        : MSPxx.SA
* CHANGE/HISTORY         :
*                        : First fully working version
*                        : PA 5-07-91
* Version 0.0            : Adapted from working verion 0.9.
*                        : Uses equates to define operating
*                        : conditions instead of a parameter
*                        : block CM 7-07-91
* Version 0.1            : Corrected PORTA assignments
*                        : CM/PA 8-07-91
* Version 0.2            : Took out demonstation and test code
*                        : Now the Mainline Simulation Program
*                        : (MSP) CM 9-07-91
* Version 0.3            : Minor structural mods & improved
*                        : comments CM 13-07-91
* Version 0.4            : More comments and mods CM 19-09-91
*                        : Use TRAPs for no-IRQ source
* Version 0.5            : Improve spurious call prog flow.
*                        : CM 26-09-91
* Version 0.6            : Improve PORT IRQ sensitivity to
*                        : SIS negation timing CM 27-09-91
* Version 0.7            : Add CPHA support CM 21-10-91
* Version 1.0            : Fixed D4 compare problem
*                        : CM 23-10-91
*****************************************************************
*****************************************************************
   OPT P=68010 NOPAGE

*****************************************************************
* Export SPIDER definitions - import MSP definitions
*****************************************************************

  XDEF  SIMBA            * MSIMP requires for PORTA test
  XDEF  SM_PORTA         * MSIMP requires for PORTA test
  XDEF  SM_DDRA          * MSIMP requires for PORTA test
  XDEF  SM_PPAR1         * MSIMP requires for PORTA test
  XDEF  SPIM             * MSIMP requires for PORTA test
  XDEF  SPIDER           * Entry label

  XREF  SPFAD            * SPIDER Running Flag (SRFlag) Add
  XREF  SPSAD            * SPSB Address
  XREF  MSIMP            * Mainline Simulation Program Add

*****************************************************************
*            SPIDER Assembly Option Definitions
*****************************************************************

PRTIRQ   EQU $00         * SELECT PORTB as SPIDER IRQ
TIMIRQ   EQU $01         * SELECT TIMER as SPIDER IRQ
```

```
*****************************************************************
*            MC68340 MODULE AND REGISTER OFFSETS FROM MBAR
*****************************************************************

SM_AVR   EQU $06        * MC683xx autovector register
SM_PORTA EQU $11        * PORT A Data Register
SM_DDRA  EQU $13        * PORTA Data Direction Register
SM_PPAR1 EQU $15        * Port A Pin Assignment Register
SM_PORTB EQU $19        * PORTB Data Register
SM_DDRB  EQU $1D        * PORTB Data Direction Register
SM_PPARB EQU $1F        * PORTB Pin Assignment Register

T1_MCR   EQU $600       * Timer 1 Module Configuration Register
T1_IRLB  EQU $604       * Timer 1 Interupt Register;INT Level Bits
T1_IRVB  EQU $605       * Timer 1 Interupt Register;INT Vector Bits
T1_CR    EQU $606       * Timer 1 Control Register
T1_SR    EQU $608       * Timer 1 Status Register

*****************************************************************
*            SPIDER Configuration Parameters
*****************************************************************

SIMBA     EQU $FFFFF000 * SIM base address
CTDAD     EQU $00008100 * Coded Transfer Descriptor Address
SSBAD     EQU $00008F00 * SPIDER Save Buffer ADdress
PTAAD     EQU $FFFFF011 * PortA address
T1CR      EQU $AA1C     * Timer1 CR bits
T1MCR     EQU $008E     * Timer1 MCR bits
SSR                       EQU $2000* SPIDER status reg
T1IRQ     EQU $0700     * Timer1 IRQ level bits
SPIM      EQU $1F       * SPIDER pins mask
DDRAM     EQU $1E       * DDRA mask
RMOSI     EQU $05       * MOSI rotation
MOSIM     EQU $04       * MOSI mask
RMISO     EQU $00       * MISO rotation
CLKPIN    EQU $10       * SPI clk pin selection
SL1PIN    EQU $02       * SPI slave1 selection
SL2PIN    EQU $08       * SPI slave2 selection
IRQSL     EQU $08       * SPIDER vector number

*****************************************************************
*             SPIDER Parameter Selection Block (SPSB) Offsets
*****************************************************************

SP_CPOL  EQU $00        * Start clock offset
SP_CPHA  EQU $01        * Clock phase offset
SP_SSX   EQU $02        * Slave select offset
SP_DSIZ  EQU $03        * Data size offset
SP_ADST1 EQU $04        * Transmit file location
SP_ADST2 EQU $08        * Receive file location
SP_SIZ1  EQU $0C        * Size of transmit file
SP_SIZ2  EQU $10        * Size of receive file
```

```
*****************************************************************
*****************************************************************
* Subroutine Name       : SPIDER Reset Routine
* Subroutine Mnemonic   : SRR
* Purpose               : Initialises SPIDER (see below)
* Input Parameters      : Header Variables - SIMBA,SPIM
*                       : SM_PPAR1,SM_PORTA,SM_DDRA,DDRAM
*                       : T1_CR,T1MCR,T1_MCR,VECNB,T1_IRVB,
*                       : T1CR,T1IRQ,T1_IRLB,SSR,SPFAD,VBR
* Initial Conditions    : VBR correctly assigned
* Final Conditions      : SPFlag cleared
*                       : SR set to SRR value
*                       : PORTA initialised for SPIDER
*                       : Timer1 init'd for SPIDER (conditional)
* Accessed Variables    : None
* Purpose of SRR        : This routine configures the PORTA pins
*                       : on the MC683xx according to the SPIM value
*                       : defined in the SPIDER Configuration Parameters.
*                       : It then uses conditional assembly to further
*                       : configure the MC683xx according to selected mode.
*                       : If the Timer1 Tgate Pin is chosen as the SPIDER
*                       : IRQ source then the SRR sets the timer mode,
*                       : interrupt arbitration level, IRQ level and
*                       : IRQ vector number.
*****************************************************************
*****************************************************************

SRR
* Save Mainline Registers on Stack
  MOVEM.L  D0-D7/A0-A6,-(A7)

* MC683xx PORT A INITIALISATION
  MOVE.L   #SIMBA,A0                * SIM Base Add -> A0
  ORI.B    #SPIM,SM_PPAR1(A0)       * SPI MASK -> PPRAR1
  ORI.B    #DDRAM,SM_PORTA(A0)      * Set SPI outs to 1
  ORI.B    #DDRAM,SM_DDRA(A0)       * Data Dir Mask -> DDRA

  IFGT TIMIRQ                       * Conditional ASM /  TIMIRQ =1
*****************************************************************
* TIMER1 SPIDER IRQ SOURCE INITIALISATION

* CLEAR SWR TO DISABLE TIMER
  CLR       T1_CR(A0)               * Clear Timer1 CR

* Timer1 MCR definition, including IARB bits
  MOVE.W    #T1MCR,T1_MCR(A0)       * Prog Timer1 MCR

* Put SPIDER IRQ vector into Timer1 Vector Register
  MOVE.B  #VECNB,T1_IRVB(A0)        * SPIDER Vect -> IV

* Timer1 Control Register Configuration
  MOVE.W   #T1CR,T1_CR(A0)          * T1CR BITS -> CRT+SIM_BASE

* Assign SPIDER IRQ Level to Timer1
  ORI.W   #T1IRQ,T1_IRLB(A0)        * Or only effects level bits

*****************************************************************
  ENDC                              * End conditional ASM

* Define MC683xx SR to set IRQ level
  MOVE.W   #SSR,SR                  * SPIDER SR -> SR

* Load SPIDER Register Restore Routine Locn
* at appropriate IRQ vector address
```

```
  MOVE.W   #VECNB,D0
  LSL.W    #$02,D0
  MOVEC    VBR,A1
  MOVE.L   #SRRR,(A1,D0.W)          * Vector Offset

* Clear SPIDER Running Flag during Init.
  CLR.B                    SPFAD    * Clear SPIDER Flag Address

* Restore Mainline Registers
  MOVEM.L  (A7)+,D0-D7/A0-A6

* Goto Mainline Simulation Program (MSP) at label MSIMP
  BRA      MSIMP
```

```
*****************************************************************
*****************************************************************
*  SPIDER MASTER ROUTINE - SMR
*  The SMR is the main SPIDER routine and comprises several
*  subroutines namely:-
*  SPIDER Master Initialisation Routine (SMIR)
*  SPIDER Data Transfer Routine (SDTR)
*  SPIDER Pointers Test Routine (SPTR)
*  SPIDER Registers Save and Restore Routine (SRSRR)
*  SPIDER Registers Restore Routine (SRRR)
*
*  The SMR is entered either through a BSR or JSR to SPIDER
*  or through an exception to the SDTR routine.
*
*****************************************************************
*****************************************************************


SPIDER
```

```
*****************************************************************
*****************************************************************
* Subroutine Name        : SPIDER MASTER INITIALISATION ROUTINE
* Subroutine Mnemonic    : SMIR
* Purpose                : SPI Mode Initialisation (see below)
* Input Parameters       : Header Variables - SIMBA,CTDAD
*                        : SPSAD,CLKPIN,SL1PIN & SL2PIN
* Initial Conditions     : Valid SPSB at SPSAD
*                        : Valid SRFlag at SPFAD
* Final Conditions       : This subroutine has two exit points
*                        : normal and error termination
*                        : Normal Termination
*                        : A1 = CTDAD start address
*                        : A2 = SPSAD
*                        : A3 = CTD end address
*                        : A4 = SPFAD
*                        : A7 = Supervisor Stack Pointer
*                        : D0 = SPSB byte 1
*                        : D1 = SPSB byte 2
*                        : D7 = SPIDER First Run Flag
*                        : A5-A6/D2-D6 = mainline values
*                        : Mainline A0-7/D0-7 on SPV stack
*                        : Valid CTD at CTDAD
*                        : Error Termination
*                        : A0-A7/D0-D7 = mainline values
* Accessed Variables     : SPSB,CTD
* Purpose of SMIR        : The SMIR for saving mainline register
*                        : values, building a Coded Transfer
*                        : Descriptor, and initialising pointers
*                        : for subsequent SPIDER routines
*****************************************************************
*****************************************************************

SMIR

*  Save Mainline Registers on the Stack
  MOVEM.L  D0-D7/A0-A6,-(A7)

  MOVE.L   #SPFAD,A4               * Store SPFlag value in A4
  CMP.B    #$00,(A4)               * Compare Flg to 0
  BEQ      NO_ERR                  * Not running - no error

  MOVE.L   (A7)+,D0                * Increment Stack
  MOVEQ    #$FF,D0                 * D0 = -1 ( error status)
  MOVEM.L  (A7)+,D1-D7/A0-A6       * Restore Registers
  RTS                              * Return to Mainline

NO_ERR
  MOVE.B   #$FF,SPFAD              * Set SPIDER run flag
  CLR.L    D0                      * Initialise and clear
  CLR.L    D1                      * required registers
  MOVEQ    #$01,D7                 * Set First Run Flag

*****************************************************************
* Coded Transfer Descriptor Build
* This section reads the SPI transfer parameters placed in
* the SPIDER PARAMETERS SELECTION BLOCK (SPSB). These values
* are converted into a Coded Transfer Descriptor (CTD) using
* user defined SPIDER parameters.A CTD is built as follows:-
*
* Byte1: Bit is set if corresponding PORTA Pin is used for
* clock and CPOL is high, else zero.
* BYTE2: Bit is set if corresponding PORTA pin is used for
* clock. This bit is used to toggle clk.
```

```
* BYTE3: Bit is set if corresponding PORTA pin is used for
* SS and SS between bytes is high, else zero.
* BYTE4: Bit is set if corresponding PORTA pin is defined
* as slave select but is not used
* BYTE5: Numeric value of SPI data size
*****************************************************************

  MOVE.L  #SIMBA,A0                 * Define Sim base address
  MOVE.L  #CTDAD,A1                 * Define CTD address
  MOVE.L  #SPSAD,A2                 * DEFINE SPSB POINTER


* Write strt clk & toggling eor value to CTD
* Bytes 1 and 2 of CTD.

  MOVE.B  (A2)+,D0                  * Read CPOL fm SPSB
  BEQ.S   STRCLLO                   * Go to low start
  MOVE.B  #CLKPIN,(A1)+             * Write CTD strt clk = 1
  MOVE.B  #CLKPIN,(A1)+             * Write CTD clk eor value
  BRA.S   SSALLOC                   * Goto allocate slave params
STRCLLO
  MOVE.B  #$0,(A1)+                 * Write CTD strt clk = 0
  MOVE.B  #CLKPIN,(A1)+             * Write CTD clk eor value

* Write Slave Select Parameters to CTD
* Bytes 3 and 4 of CTD
SSALLOC
  MOVE.B  (A2)+,D0                  * Read CPHA from SPSB
  MOVE.B  (A2)+,D1                  * Read slave sel no fm SPSB
  BEQ                   SS1         * If SSX=0 then slave 1

* Write Slave Select 2 Params to CTD
* Two bytes are written
  CMPI.B  #$1,D0                    * Is CPHA = 1?
  BEQ.S   SS2LO                     * If so goto SS2LO
  MOVE.B  #SL2PIN,(A1)+             * Write CTD SS2 btwn bytes = 1
  BRA.S   MSKSS1                    * goto define SS out
SS2LO
  MOVE.B  #$0,(A1)+                 * Write CTD SS2 btwn bytes = 0
MSKSS1
  MOVE.B  #SL1PIN,(A1)+             * Write CTD, mask SS1 output

* Write Slave Select 1 Params to CTD
* Two bytes are written
SS1
  CMPI.B  #$1,D0                    * Is CPHA = 1?
  BEQ.S                   SS1LO     * If so goto SS1LO
  MOVE.B  #SL1PIN,(A1)+             * Write CTD SS1 btwn bytes = 1
  BRA.S                   MSKSS2    * goto define SS output
SS1LO
  MOVE.B  #$0,(A1)+                 * Write CTD SS1 btwn bytes = 0
MSKSS2
  MOVE.B  #SL2PIN,(A1)+             * Write CTD, mask SS2 output

* Read Data Size fm SPSB and write to CTD
  MOVE.B  (A2),(A1)                 * Write Data Size

* ADD_STRT1,ADD_STRT2,SIZ1 and SIZ2 are held in SPSB
```

```
*****************************************************************
* Save SPIDER Pointers in Registers for quick access
*****************************************************************
* Save SIM base address   in A0 - already done
* Save Start SPSB address in A2
  MOVE.L   #SPSAD,A2
* Save end of CTD address in A3
  MOVE.L  A1,A3
* Save start CTD address  in A1
  MOVE.L   #CTDAD,A1
* Save SPFlag address     in A4 - already done
```

```
*****************************************************************
*****************************************************************
* Subroutine Name        : SPIDER  DATA TRANSFER ROUTINE
* Subroutine Mnemonic    : SDTR
* Purpose                : Transmits & receives one SPI word
*                        : (more details below)
* Input Parameters       : Header Variables - SPFAD, SM_PORTA
*                        : SL1PIN,SL2PIN,IRQSL,SM_PPARB,SM_AVR
*                        : SM_DDRB,SP_ADST1,SP_ADST2,RMOSI
*                        : MOSIM,PTAAD,RMISO
* Initial Conditions     : This subroutine is run after either
*                        : the SMIR or the SRRR.
*                        : Initial conditions for both are:-
*                        : Valid SPSB at SPSAD
*                        : Valid CTD at CTDAD
*                        : Valid Transmit block at SP_ADST1
*                        : Adequate RAM at SP_ADST2
*                        : A0 = SIMBA
*                        : A1 = CTD start address
*                        : A2 = SPSAD
*                        : A3 = CTD end address
*                        : A4-7/D0-7 don't care
* Final Conditions       : The exit conditions are as follows
*                        : ADST1,SIZ1 indexed
*                        : D0 holds received data word (8/16b)
*                        : D1-D6/A6 altered
*                        : A0-A5,A7/D7 as initial conditions
* Purpose of SDTR        : The purpose of the SDTR is to set up
*                        : the SPI control outputs,fetch the
*                        : correct transmit word,inc pointers
*                        : perform a single transfer and put
*                        : the received word in D0.
*                        : If a PORTB line is used for SIS then
*                        : this is also configured as an input
*                        : PORT line.
*****************************************************************
*****************************************************************

SDTR
* Also entry point for IRQ and TRAPS
  MOVE.B (A1),D5                    * CTD clk1 -> D6
  MOVE.B $01(A1),D6                 * CTD clk2 -> D6

*****************************************************************
* Drive SPI clk to CPOL level prior to SS assertion
* Set STRT_CLK data in PORTA before making output
* Slave select data already set to one in init code
* All SPIDER outs already defined as such in init code
*****************************************************************
  MOVE.B #CLKPIN,D2                 * PortA clk mask in D2
  NOT.B  D2                         * Create &-MASK
  AND.B  SM_PORTA(A0),D2            * Clk data=0, rest save
  OR.B   D5,D2                      * Add clk strt data
  MOVE.B D2,SM_PORTA(A0)            * Write mod'd PORTA


*****************************************************************
* Assert correct SS pin
*****************************************************************
  MOVE.B #SL1PIN,D2                 * Slave sel1 into D2
  ORI.B #SL2PIN,D2                  * Add slave sel2
  NOT.B  D2                         * Create mask for PORTA
  AND.B  SM_PORTA(A0),D2            * Save PORTA, both SS=0
  OR.B   $3(A1),D2                  * Negate unused SS
```

```
  MOVE.B D2,SM_PORTA(A0)            * SCKL,SS1,SS2  -> PORT A

  MOVE.L SP_ADST1(A2),A6            * Transmit block add -> A6
  CMP.B                 #$08,(A3)   * Chk SPSB data size
  BNE WORDS                         * Goto WORDS if 16-bit

  MOVE.B (A6),D0                    * read byte into D0
  ADDI.L #$01,SP_ADST1(A2)          * increment pointer by 1
  SUBI.L #$01,SP_SIZ1(A2)           * decrement block size by 1
  BRA SVSIZE                        * Goto save size

WORDS
  MOVE.W (A6),D0                    * Load transmit data word
  ADDI.L #$02,SP_ADST1(A2)          * Increment pointer by 2
  SUBI.L #$02,SP_SIZ1(A2)           * Decrement size by 2

* Send one byte
SVSIZE
  MOVE.B (A3),D4                    * No of data bits into D4

*********************************************************************
* Configuration of RMOSI for byte or word data
* If transmit data is 8-bits then  RMOSI -> RMOSI
* If transmit data is 16-bits then RMOSI -> RMOSI + 8
*********************************************************************

  CLR.L  D1                         *
  CMP.B  #$08,D4                    *
  BEQ    DATA                       *
  MOVE   D4,D1                      *

DATA
  ADD.B  #RMOSI,D1                  * RMOSI + D1 -> D1
  MOVE.B #MOSIM,D5                  * Generate inverse MOSI
  NOT.B  D5

 IFGT PRTIRQ                        * Conditional ASM
*********************************************************************
* PORTB SPIDER IRQ SOURCE HANDSHAKE INITIALISATION
*********************************************************************

* Set the CPU32 IRQ mask to level 7
  MOVE.W #$2700,SR

* Change fm IRQ to PORT line
  MOVE.B #IRQSL,D2
  EOR.B  D2,SM_PPARB(A0)

* AVR Register Configuration
  OR.B   D2,SM_AVR(A0)

* Configure PORTB SIS pin as input
  NOT.B  D2
  AND.B  D2,SM_DDRB(A0)

*********************************************************************
  ENDC                              * End conditional ASM
```

```
*********************************************************************
* Serialisation / de-serialisation routine
* D0 is used as the SPI data register.
*********************************************************************

  MOVE.L  #PTAAD,A6                 * Sim base -> A6

*********************************************************************
* If CPHA =1 , require extra clock inversion
* before data transfer occurs
*********************************************************************

  CMPI.B #$1,SP_CPHA(A2)            * Is CPHA =1?
  BNE    NXTBY                      * If not do not invert
  EOR.B  D6,(A6)                    * Invert SPI clock

NXTBY
  MOVE.W D0,D2                      * Transfer data in D2
  LSR.B  D1,D2                      * RMOSI rot'n to select MOSI
  MOVE.B #MOSIM,D3                  * MOSI mask -> D3
  AND.B  D2,D3                      * Select MOSI
  MOVE.B D5,D2                      * Inverse MOSI mask -> D2
  AND.B  (A6),D2                    * Select SCLK, SS1,SS2
  OR.B   D3,D2
  MOVE.B D2,(A6)                    * Transmit data -> MOSI
  LSL    #$01,D0                    * Get next bit ready
  EOR.B  D6,(A6)                    * Invert SPI clk
  MOVE.B #RMISO,D2                  * RMISO -> D1
  MOVE.B (A6),D3                    * Read MISO into D3
  LSR.B  D2,D3                      * Put MISO data to bit0
  AND    #$01,D3                    * Mask MISO
  OR.W   D3,D0                      * Store MISO in bit0 of SPDR
  CMP.B  #$01,D4                    * Is this the last bit?
  BEQ    LASTB                      * If not can safely invert
  EOR.B  D6,(A6)                    * Invert SPI clock
  DBF    D4,NXTBY                   * D4 is SPI data bit counter
LASTB
  CMP.B #$1,SP_CPHA(A2)             * Is CPHA = 1?
  BEQ   NOINV                       * If so nothing to do
  EOR.B D6,(A6)                     * If not invert SCLK
NOINV
```

```
*****************************************************************
*****************************************************************
* Subroutine Name       : SPIDER POINTERS TEST ROUTINE
* Subroutine Mnemonic   : SPTR
* Purpose               : Stores received word & checks
*                       : pointers, drives interword levels
*                       : (more details below)
* Input Parameters      : Header Variables - SP_ADST2,SP_SIZ2
*                       : SP_SIZ1,SP_SIZ2,SM_PORTA,IRQSL,SSR
* Initial Conditions    : This subroutine has two entry points
*                       : From SMIR or SPIDER Interrupt
*                       : Initial conditions for both are:-
*                       : Valid SPFlag at SPFAD
*                       : Valid SPSB at SPSAD
*                       : Valid CTD at CTDAD
*                       : A0 = SIMBA
*                       : A1 = CTD start address
*                       : A2 = SPSAD
*                       : A3 = CTD end address
*                       : A4 = SPFAD
*                       : A5-7,D0-7 don't care
* Final Conditions      : A0-5,A7/D0-1,D3-4,D6-7
*                       : as initial conditions
*                       : A6 = Receive store address
*                       : D2 = IRQSL (conditional)
*                       : D5 = Byte two of CTD
*                       : SR = SSR (conditional)
*                       : ADST2 incremented
*                       : SIZ2 decremented
*                       : Receive block updated (conditional)
*****************************************************************
*****************************************************************

SPTR

  IFGT PRTIRQ                           * Conditional ASM
*****************************************************************
* PORTB SPIDER IRQ SOURCE HANDSHAKE ROUTINE
*****************************************************************

WAIT
  MOVE.B  #IRQSL,D2                     * SIS pin -> D2
  AND.B   SM_PORTB(A0),D2               * Is SIS = 1 ?
  BEQ.S   WAIT                          * Wait for Slave to respond

*****************************************************************
  ENDC                                  * End Conditional ASM


* If receive size var = 0 then do not read SPI data register
* If receive size var <>0 then put received data in the SPI
* data register, in MC683xx memory

  CMP.L  #$0,SP_SIZ2(A2)                * Chk receive block size
  BLE  TST_SIZ1                         * Size <=0, do not read

  MOVE.L SP_ADST2(A2),A6                * Receive store add -> A6
  CMP.B  #$08,(A3)                      * Chk SPSB for data size
  BNE    DT_WORD                        * If not byte goto word
  MOVE.B D0,(A6)                        * Received byte -> rcv blk
  ADDI.L #$01,SP_ADST2(A2)              * Increment rcv pointer
  SUBI.L #$01,SP_SIZ2(A2)               * Decrement rcv size
  BRA    TST_SIZ1
```

```
DT_WORD
  MOVE.W D0,(A6)                    * Received word -> rcv blk
  ADDI.L #$02,SP_ADST2(A2)          * Increment rcv pointer
  SUBI.L #$02,SP_SIZ2(A2)           * Decrement rcv size

* Compare transmit and receive size values
TST_SIZ1
  MOVE.L SP_SIZ1(A2),D5             * SIZ1 -> D5
  ADD.L  SP_SIZ2(A2),D5             * SIZ2 +> D5
  BGT    SSBB                       * Non-zero? Skip flag set

* Clear SPIDER flag, task finished
  MOVE.L #$00,(A4)                  * Clear SPIDER flag

* SS between bytes
SSBB
  MOVE.B  $2(A1),D5                 * SS between bytes -> D5
  OR.B  D5,SM_PORTA(A0)             * SS level -> PORTA

  IFGT PRTIRQ                       * Conditional ASM
*****************************************************************
* PORTB SPIDER IRQ SOURCE HANDSHAKE ROUTINE
*****************************************************************

* Change PORTB line to IRQ line SIS
  EOR.B   D2,SM_PPARB(A0)

* Restore SPIDER Status Register
  MOVE.W  #SSR,SR

*****************************************************************
  ENDC                              * End Conditional ASM
```

```
*****************************************************************
*****************************************************************
* Subroutine Name       : SPIDER REGISTER SAVE AND RESTORE ROUTINE
* Subroutine Mnemonic   : SRSRR
* Purpose               : Stores SPIDER registers and restores
*                       : mainline values from stack
* Input Parameters      : Header Variables - SSBAD
* Initial Conditions    : A0 = SIMBA
*                       : A1 = CTD start address
*                       : A2 = SPSAD
*                       : A3 = CTD end address
*                       : A4 = SPFAD
*                       : A7 = Mainline Stack Pointer
*                       : D7 = SPIDER First Run Flag
*                       : A5-6/D0-D6 don't care.
* Final Conditions      : A0-7/D0-7 = mainline values
*                       : SPIDER A0-A4/D7 values stored
*                       : at SPIDER Save Buffer ADdress (SSBAD)
*                       : Timer1 IRQ reset
*****************************************************************
*****************************************************************

SRSRR

  CMP    #$01,D7                    * If D7=1 perform rts
  BNE    RT_IRQ                     * Else rte

RT_SUB
  MOVE.L #SSBAD,A6                  * SPIDER sv buf add -> A6
  MOVEM.L D0-D7/A0-A5,-(A6)         * Save SPIDER reg values
  MOVEM.L (A7)+,D0-D7/A0-A6         * Restore Mainline reg values
  MOVEQ   #$0,D0                    * Valid call returned
  RTS

RT_IRQ

  IFGT TIMIRQ                       * Conditional ASM
*****************************************************************
* CLEAR TG BIT -> RESET IRQ -
  MOVE.W   #$2000,T1_SR(A0)
*****************************************************************
  ENDC                              * End Conditional ASM

  MOVE.L   #SSBAD,A6                * Store Save Buffer Address
  MOVEM.L  D0-D7/A0-A5,-(A6)        * Save SPIDER register values
  MOVEM.L  (A7)+,D0-D7/A0-A6        * Restore Mainline Values
  RTE                               * Return to Mainline
```

```
*****************************************************************
*****************************************************************
* Subroutine Name        : SPIDER REGISTER RESTORE ROUTINE
* Subroutine Mnemonic    : SRRR
* Purpose                : This routine is SPIDER exception
*                        : entry point for IRQ and TRAPS
*                        : Stores mainline registers and restores
*                        : SPIDER values from SPIDER Save Buffer
* Input Parameters       : Header Variables - SSBAD
* Initial Conditions     : A0-7/D0-7 = mainline values
*                        : SPIDER A0-A4/D7 values stored
*                        : at SPIDER Save Buffer ADdress (SSBAD)
* Final Conditions       : 2 exits, normal and spurious call
*                        : Spurious final conditions = initial
*                        : Normal conditions as below
*                        : A0 = SIMBA
*                        : A1 = CTD start address
*                        : A2 = SPSAD
*                        : A3 = CTD end address
*                        : A4 = SPFAD
*                        : A7 = Mainline Stack Pointer
*                        : D7 = SPIDER First Run Flag
*                        : A5-6/D0-D6 mainline values
*****************************************************************
*****************************************************************

SRRR
  MOVEM.L D0-D7/A0-A6,-(A7)         * Save Mainline registers
  MOVE.L  #SSBAD,A6
  SUBA.L  #$38,A6
  MOVEM.L (A6)+,D0-D7/A0-A5         * Restore SPIDER registers
  CMP.B   #$00,(A4)                 * Check SPIDER run flag
  BEQ     SPURET                    * If not, must be spurious

  MOVEQ   #$00,D7                   * Indicate exception generated
  BRA     SDTR                      * goto data transfer routine

* Spurious IRQ Return
SPURET
  MOVE.L  #SSBAD,A6                 * Save buffer address
  MOVEM.L D0-D7/A0-A5,-(A6)         * Save SPIDER reg values
  MOVEM.L (A7)+,D0-D7/A0-A6         * Restore mainline values
  RTE                               * Return to mainline


*****************************************************************
    END
```

```
*****************************************************************
********************PROGRAM HEADER*******************************
* FILE NAME              : MSIMP.SA
* PROGRAM NAME           : Mainline Simulation Program
* VERSION                : 1.0 (Release)
* LAST MODIFIED          : 22-10-91
* LANGUAGE               : Freescale Syntax MASM M68000 Assembler
* TARGET SYSTEM          : MC68340
* PROGRAMMER             : Philippe ALVES - ENSBERG, France
* AUTHOR                 : Colin Mac Donald - Freescale, UK
* COMPANY                : FREESCALE
* FUNCTION OF PROGRAM    : Test and demonstrate SPIDER program
*                        : configured for an SIS.
* RELATED FILES          : SPIDER.RME, SPxxx.COM, SPxxx.DAT
*                        : SPxxx.SA
* CHANGE/HISTORY         :
* Version 0.0            : Split from version 1.1 of SPIDER
*                        : Pa/CM 8-07-91
* Version 0.1            : Add further comments CM 18-09-91
* Version 1.0            : Add Register and PortA test
*                        : CM 22-10-91
*****************************************************************
*****************************************************************

  OPT P=68010
  NOPAGE

*****************************************************************
* Import SPIDER variables & export MSP variables
*****************************************************************

  XREF  SM_DDRA          * Used for PORTA Test
  XREF  SM_PPAR1         * Used for PORTA Test
  XREF  SM_PORTA         * Used for PORTA Test
  XREF  SIMBA            * Used for PORTA Test
  XREF  SPIM             * Used for PORTA Test
  XREF  SPIDER           * Branch label

  XDEF  SPFAD            * SPIDER Running Flag (SRFlag) Address
  XDEF  SPSAD            * SPSB Address
  XDEF  MSIMP            * Mainline Simulation Program Address

*****************************************************************
* Mainline Simulation Program Options
*****************************************************************

DELAY   EQU $2000       * DELAY = << 1 second

*****************************************************************
* SPIDER Param Selection Block (SPSB) Demonstration Values
*****************************************************************

CPOL    EQU $00         * Clock polarity = 0
CPHA    EQU $00         * Clock phase = 0
SSX     EQU $00         * Slave select 1
DSIZ    EQU $08         * 8-bit SPI word size
SIZ1    EQU $42         * Size of send file + 2 bytes
SIZ2    EQU $40         * Size of receive file in bytes
```

```
*****************************************************************
*****************************************************************
* Subroutine Name       : Mainline Simulation Routine
* Subroutine Mnemonic   : MSIMP
* Purpose               : Simulates calling software for SPIDER
* Input Parameters      : Header Variables
*                       : TEMPO,SPFAD,SPSAD,ADST1,ADST2,SIZ1,SIZ2
* Initial Conditions    : SPSB,TRANSMIT BLOCK defined
*                                 : A0-7/D0-7 don't care
* Accessed Variables    : SPSB,Transmit file
* Final Conditions      : Abort or Reset required to exit
*                       : Program is infinite loop
*                       : In this loop
*                       : D3-D7 -> FFFFFFFF
*                       : A0-A6 -> FFFFFFFF
*                       : D0 -> status
*                       : D1 -> count
*                       : D2 -> inverse of SPIM
*                       : A7 not modified
* Note                  : The SPIDER Application note describes
*                       : this routine as the MCR and MLR subroutines
*                       : to aid understanding.
*****************************************************************
*****************************************************************

MSIMP

* Delay loop is performed before each SPIDER call.
* HC05C4 test code sends each transfered data block to a VDU
* Delay ensures each block can be read before it changes


  MOVEA.L #SIMBA,A0               * Sim Base -> A0
  MOVE    #SPIM,D2                * Get SPIDER pins fm prg
  NOT.B   D2                      * Non-SPI pins
  OR.B    D2,SM_PPAR1(A0)         * Non-SPIDER pins -> out
  OR.B    D2,SM_PORTA(A0)         * Set outs high
  OR.B    D2,SM_DDRA(A0)          * Data dir mask to DDRA

  MOVEQ   #$FF,D3                 * Test value -> register
  MOVE.L  D3,D4                   * Test value -> register
  MOVE.L  D3,D5                   * Test value -> register
  MOVE.L  D3,D6                   * Test value -> register
  MOVE.L  D3,D7                   * Test value -> register

  MOVE.L  D3,A1                   * Test value -> register
  MOVE.L  D3,A2                   * Test value -> register
  MOVE.L  D3,A3                   * Test value -> register
  MOVE.L  D3,A4                   * Test value -> register
  MOVE.L  D3,A5                   * Test value -> register
  MOVE.L  D3,A6                   * Test value -> register

  MOVEQ   #$00,D1                 * Initialise count to zero
COUNT
  ADDQ.L  #$01,D1                 * Increment count
  CMP.L   #DELAY,D1               * Compare count against DELAY
  BNE     COUNT                   * Keep looping until match

* Call SPIDER continuously, use infinite loop
  CLR     D0                      * Set D0 to zero
RECALL
  JSR     SPIDER                  * Call SPIDER, status -> D0
  CMP.L   #$FFFFFFFF,D0           * If D0 = -1 -> SPI running
  BEQ     RECALL                  * Wait until next call
```

```
*********************************************************************
* MAINLINE LOOP ROUTINE
*********************************************************************

RUNCHK
  EOR.B   D2,SIMBA+SM_PORTA         * Invert non-SPIDER pins
  CMPI.B  #$FF,SPFAD                * Check if SPIDER is running
  BEQ     RUNCHK                    * If so, wait until finished

*********************************************************************

RELOAD
  LEA     SPSAD,A0                  * Load SPSB address into A2
  MOVE.L  #ADST1,$04(A0)            * Move Tx file loc'n to SPSB
  MOVE.L  #ADST2,$08(A0)            * Move Rx file loc'n to SPSB
  MOVE.L  #SIZ1,$0C(A0)             * Move Tx file size to SPSB
  MOVE.L  #SIZ2,$10(A0)             * Move Rx file size to SPSB
  MOVE.B  #SIZ1,ADST1               * 1st byte of Tx is Tx size
  MOVE.B  #SIZ2,ADST1+$01           * 2nd byte of Tx is Rx size
  BRA     MSIMP


*********************************************************************
* SPIDER FLAG LOCATION
*********************************************************************
SPFAD
  DS.L  1                           * FFFFFFF or 0

*********************************************************************
* SPIDER SELECTION BLOCK -
*********************************************************************
* This buffer is normally located in RAM. Its position is
* defined in the SCB by the user. This buffer is loaded before
* each SPIDER call

SPSAD
  DC.B  CPOL                        * Select clock polarity
  DC.B  CPHA                        * Select clock phase
  DC.B  SSX                         * Select slave1 or slave2
  DC.B  DSIZ                        * Select the data format
  DC.L  ADST1                       * Start address of send file
  DC.L  ADST2                       * Start address of receive file
  DC.L  SIZ1                        * Size of transmit file + 2 bytes
  DC.L  SIZ2                        * Size of receive file


*********************************************************************
* DEMONSTRATION TRANSMIT FILE
*********************************************************************

ADST1
ADST2
* START 1
* SIZ1
  DC.B   $42
* SIZ2
  DC.B   $40
* DATA TO TRANSMIT
  DC.B   $1B
  DC.B   $5B
  DC.B   $31
  DC.B   $3B
  DC.B   $31
  DC.B   $48
```

```
  DC.B   'SERIAL PERIPHERAL'
  DC.B   $0D
  DC.B   $0A
  DC.B   'INTERFACE DEVICE'
  DC.B   $0D
  DC.B   $0A
  DC.B   'EMULATION ROUTINE'
  DC.B   '                 '
*****************************************************************
    END
```

```
*****************************************************************
********************PROGRAM HEADER*******************************
* FILE NAME              : SPT05.SA
* PROGRAM NAME           : SPIDER Test Program for HC05
* VERSION                : 1.0 Release
* LAST MODIFIED          : 26-09-91
* LANGUAGE               : MC68HC05 Assembler PASM05
* TARGET SYSTEM          : MC68HC05 Evaluation module EVM05
* PROGRAMMER             : Philippe Alves - ENSERG, France
* AUTHOR                 : Colin Mac Donald - Freescale, UK
* COMPANY                : FREESCALE
* FUNCTION OF PROGRAM    : Tests various modes of SPIDER
* CHANGE/HISTORY         : 1st pass 12-06-91
*                        : Headers,subheaders,comments
*                        : added, variable and label renaming
*                        : CM 26-09-91
*****************************************************************
*****************************************************************

*****************************************************************
* Operating Mode Selection
*****************************************************************
PRTIRQ  EQU  $00         * PORTB used as IRQ source
IRQM    EQU  $01         * IRQM = 01 SPIDER uses IRQs
                         * IRQM = 00 SPIDER uses TRAPS

*****************************************************************
* General Variables
*****************************************************************
SIZLOC  EQU  $50         * Address of Rx & Tx block sizes
FTS     EQU  $60         * Send block address
FTR     EQU  $60         * Receive block address
HSPIF   EQU  $01         * SPI IRQ routine add high
LSPIF   EQU  $89         * SPI IRQ routine add low
FLAG    EQU  $53         * 05STIR run flag address
ACCU1   EQU  $54         * Save ACCUMA address
ACCU2   EQU  $55         * Save Index Reg address
TxPTR   EQU  $57         * Send block pointer address
RxPTR   EQU  $59         * Receive block pointer
IRQFLG  EQU  $5A         * SPI IRQ flag address
SIZPTR  EQU  $5B         * Rx block size address

*****************************************************************
*  SCI Variables
*****************************************************************
SCDAT   EQU  $11         * Serial Comms Data Reg Address
SCCR2   EQU  $0F         * Serial Comms Control Reg2 Address
BDRR    EQU  $0D         * Baud Rate Register Add
SCCR1   EQU  $0E         * Serial Comms Control Reg1 Address
SCSR    EQU  $10         * Serial Comms Status Reg Address
BDRM    EQU  $F0         * Baud rate register mask

*****************************************************************
* SPI Variables
*****************************************************************
SPCR    EQU   $0A        * Serial Peripheral Control Reg Address
SPDR    EQU   $0C        * Serial Peripheral Data Reg Address
CPMK    EQU   $04        * CPOL , CPHA MASK
SPSR    EQU   $0B        * Serial Peripheral Status Reg Add
SPMK    EQU   $01        * SPR0 , SPR1 MASK
```

```
*****************************************************************
*  PORTA Variables
*****************************************************************
DDRA    EQU   $04       * Data Direction RegisterA Address
DRA     EQU   $00       * Data RegisterA Address
PORTA   EQU   $01       * PORTA I/O configuration
```

```
*****************************************************************
* Subroutine Name       : MC68HC05 SPIDER Test Initialisation
* Subroutine Mnemonic   : 05STI
* Purpose               : Initialises HC05 to test SPIDER
* Input Parameters      : Header Variables - BDRM,CPMK
* Initial Conditions    :
* Final Conditions      : PORTA,SCI & SPI initialised
*                       : SPIF cleared
*                       : SPI IRQ vector initialised
*                       : ACCUM modified
* Accessed Variables    : None
*****************************************************************
*****************************************************************

    ORG $100

05STI

  LDA SPSR              * Clear SPIF
  LDA SPDR              * Clear SPIF

* Initialise PORTA for SPIDER I/O
  LDA #PORTA            * Load PORTA definition
  STA DRA               * PORTA outdata -> 1
  STA DDRA              * PORTA outs defined
  STA DRA               * Negate SIS

* Initialise SCI for terminal display
  LDA #$08
  STA SCCR2             * Enable serial transmit
  CLR SCCR1             * 8 bits, 1 start, 1 stop
  LDA #BDRM             * xtal freq = 4Mhz ,
  STA BDRR              * 9600 baud rate

* Initialise SPI for SPIDER
  LDA #$00              * SPIE=0,SPE=0,MSTR=0
  STA SPCR              * SPI Config Reg
  ORA #CPMK             * CPOL, CPHA = 00
  STA SPCR              * SPI Config Reg

* Set SPI Exception vector to point to SPT05
  LDA #HSPIF            * SPI vector high byte
  STA $1FF4             *
  LDA #LSPIF            * SPI vector low byte
  STA $1FF5

*  Enable HC05 SPI System
  LDA #$40
  STA SPCR              * Set enable SPI bit
  LDA SPSR              * Clear SPI IRQ flag
  LDA SPDR              * Clear SPI IRQ flag
```

```
*****************************************************************
*****************************************************************
* Subroutine Name       : MC68HC05 SPIDER Test Conversion Routine
* Subroutine Mnemonic   : 05STCR
* Purpose               : Converts SPI input to SCI for VDU
*                       : display
* Input Parameters      : FTR,FTS,TxPTR,RxPTR,IRQFLG,SIZLOC,
*                       : FLAG,SIZPTR,IRQM,PORTA
* Initial Conditions    : SCI initialised
*                       : Valid data in SPI if IRQFLG set
* Final Conditions      : Infinite loop - Abort to breakout
* Accessed Variables    : IRQFLG,SIZLOC,SIZLOC+1
*****************************************************************
*****************************************************************

05STCR

  CLI                   * Clear Interrupt
  LDA #$E0              * Enable SPIF IRQs
  STA SPCR

STRT
  LDX #FTR              * Receive Add -> Index
  LDA #FTS              * Transmit Add -> ACCUM
  STA TxPTR             * ACCUM -> Tx pointer
  LDA #FTR              * Receive Add -> ACCUM
  STA RxPTR             * ACCUM -> Rx pointer
  CLR IRQFLG            * Clear 05STIR flag
  LDA #SIZLOC           * Rx block size add -> ACCUM
  STA SIZPTR            * ACCUM -> SIZE pointer
  CLR FLAG              * Clear FLAG


* Negate SPIDER IRQ Source
CHKMODE
  LDA #PRTIRQ           * Load PORTB IRQ var
  CMP #$00              * Does SPIDER use PORTB IRQ
  BNE CHKIRQ            * If yes skip negation

  LDA #PORTA            * Load PORTA def'ns
  STA DRA               * Drive outs high
  STA DDRA              * Config I/O

* Check if character has been received
CHKIRQ
  LDA IRQFLG            * Load SPI IRQ flag
  CMP #$00              * Check against 0
  BEQ CHKMODE           * If zero, loop

CHKM2
  LDA #PRTIRQ           * Load PORTB IRQ variable
  CMP #$00              * Does SPIDER use PORTB IRQ
  BNE TST2              * If yes skip negation

  LDA #PORTA            * Load PORTA def'ns
  STA DRA               * Drive outs high
  STA DDRA              * Config I/O

TST2
  LDA #$80              * TDRE mask -> ACCUM
  AND SCSR              * Check TDRE bit in SCSR
  CMP #$00              * Last byte transmitted?
  BEQ CHKM2             * If not, goto CHKM2
  LDA SCSR              * Clear Transmit complete bit
```

```
* Send SPI data to VDU
  LDA $00,X             * Index Loc data -> ACCUM
  INCX                  * Increment Index
  STA SCDAT             * Store in SCI data reg
  DEC IRQFLG            * Decrement sent size

* Check size of receive block
  LDA SIZLOC            * Receive size -> ACCUM
  CMP #$02              * Transfer size = 2?
  BNE CHKMODE           * If not, goto CHKMODE

* Check size of transfer block
  LDA SIZLOC+$01        * Transfer size -> ACCUM
  CMP #$00              * Transfer size = 0 ?
  BNE CHKMODE           * If not, goto CHKMODE

* Check
  LDA IRQFLG            * ACCU5 -> ACCUM
  CMP #$00              * Check for received byte
  BNE CHKMODE           * If not, goto CHKMODE
  BRA STRT              * Goto start
```

```
*****************************************************************
*****************************************************************
* Subroutine Name       : MC68HC05C4 SPIDER Test IRQ Routine
* Subroutine Mnemonic   : 05STIR
* Purpose               : Responsible for servicing SPIFs
* Input Parameters      : Header Variables - SIZPTR,SIZLOC
*                       : TxPTR,FLAG,RxPTR
* Initial Conditions    : Valid data in SPDR
*                       : Valid FLAG,SIZPTR,TxPTR,SIZLOC,RxPTR
*                       : ACCUM,Index not cared
* Final Conditions      : Next byte of Tx data in SPDR
*                       : Rx,Tx pointers incremented
*                       : Rx,Tx block sizes decremented
*                       : FLAG set (conditional)
*                       : ACCUM,Index restored
*                       : Assert SPIDER IRQ (conditional)
* Accessed Variables    : SIZPTR,TxPTR,RxPTR
*****************************************************************
*****************************************************************

05STIR

  STA ACCU1             * Save ACCUMA
  STX ACCU2             * Save index register

  LDA #PORTA            * Load PORTA definition
  STA DRA               * PORTA outdata -> 1
  STA DDRA              * PORTA outs defined

* Test size stored flag
  LDA FLAG              * Load flag from memory
  CMP #$ff              * If flag = 1 then Rx size
                        * & Tx size stored in SIZLOC
  BEQ STOREDT           * Branch to store data

*****************************************************************
* Store Transmit and Receive Block Sizes
*****************************************************************

* 1st and 2nd received bytes
  LDX SIZPTR            * Rx or Tx size add -> index
  LDA SPSR              * Clear SPIF part1
  CLR SPDR              * Clear SPIF part2
  LDA SPDR              * Read received byte
  STA $00,X             * store 1st or 2nd byte
  INC SIZPTR            * Increment pointer

* First pass, index=SIZLOC, 2nd pass, index=SIZLOC+1
  CPX #SIZLOC+$01       * 1st or 2nd pass?
  BNE BLCKEND           * If first, goto BLCKEND

* 2nd received byte only
  LDA #$FF              * Set size flag as both
  STA FLAG              * Rx and Tx sizes recv'd

  LDX TxPTR             * Send block ptr -> index
  LDA $00,X             * Send byte -> ACCUM
  STA SPDR              * ACCUM -> SPI Data Reg
  INC TxPTR             * Increment send ptr
  DEC SIZLOC+$01        * Decrement send size
  BRA BLCKEND           * Goto end check
```

```
*****************************************************************
* Store Data
*****************************************************************

STOREDT
* Test Send Block Size against zero
  LDA SIZLOC+$01        * Send size -> ACCUM
  CMP #$00              * Send size = 0 ?
  BEQ RCVDT             * If so, goto RCVDT

  LDX TxPTR             * Send ptr -> index
  LDA SPSR              * Clear SPIF
  LDA $00,X             * Send byte -> ACCUM
  STA SPDR              * ACCUM -> SPI Data Reg
  INC TxPTR             * Increment send ptr
  DEC SIZLOC+$01        * Decrement send size

RCVDT
* Test
  LDA SIZLOC            * Rx block size -> ACCUM
  CMP #$00              * Receive complete ?
  BEQ BLCKEND           * If so, goto end check

* Store Received Data
  LDX RxPTR             * Rx pointer -> index
  LDA SPSR              * Clear SPIF
  LDA SPDR              * Received data -> ACCUM
  STA $00,X             * Rx data -> Rx block
  INC RxPTR             * Inc Rx pointer
  DEC SIZLOC            * Decrement Rx size
  INC IRQFLG            * Set SPI IRQ flag

* Check for block transfer completion
BLCKEND
  LDA SIZLOC            * Rx block size -> ACCUM
  CMP #$02              * Rx block size = 2 ?
  BNE IRQRET            * If not, continue transfer

  LDA SIZLOC+$01        * Tx block size -> ACCUM
  CMP #$00              * Tx block size = 0 ?
  BNE IRQRET            * If not, continue transfer
  RTI                   * Return without new IRQ

* Restore state and return from IRQ
IRQRET
  LDA ACCU1             * Restore ACCUM
  LDX ACCU2             * Restore index

  LDA #IRQM             * IRQ mode -> ACCUM
  CMP #$01              * Check mode
  BEQ SIS_LOW           * If IRQ mode, set low
  RTI                   * Else, return from IRQ

SIS_LOW
  CLR DRA               * Clear Data Reg A
  RTI                   * Return from IRQ
  END
```

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

Asia/Pacific:
Freescale Semiconductor 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