Serial Peripheral Interface

Some registers parameters are only for 55800
SPI Features

- Master/Slave
  - Supports up to 15 external devices
- Supports SPI modes 0, 1, 2 & 3
  - All combinations of clock phase and polarity
- Programmable:
  - 8 to 16 bit Data Length
  - Delays between chip selects
  - Delays between consecutive transfers
  - Delays between clock and data per chip select
- Selectable Mode Fault Detection
- Fixed or Variable peripheral selection
- Peripheral Data Controller (PDC)
  - Chained Buffer support
- Local Loop back in Master mode
Dependencies

- PMC has to be programmed 1st for SPI to work
- PIO Controller has to be programmed for the pins to behave as intended
- SPI Peripheral Inputs “see” the state of the pad.
- For example:
  - Use the SPI as a transmitter only.
  - Program the PIO controller pins for SPCK and MOSI to be outputs.
  - Program the PIO controller pin for MISO to be a GPIO.
    - The SPI peripheral’s internal MISO input will see the state of the GPIO.
- Be careful of NPCS0/NSS/GPIO pin
  - If you only have 1 external SPI devices then technically you can don’t need an external chip select.
  - However if the SPI sees a 0 on NSS PIO line, a Mode Fault can be generated.
Master Mode Clock Generation

- SCBR is 0 on reset. 0 leads to unpredictable results.
  - Set to something other than 0 before 1st transfer
  - Each Chip Select can have its own baud rate
    - FDIV is the same for all chip selects

\[
\text{SPCK} = \frac{\text{MCK}}{(N \times \text{SCBR})}
\]

<table>
<thead>
<tr>
<th>FDIV</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>
SPI Control Register SPI_CR

<table>
<thead>
<tr>
<th>LASTXFER 24</th>
<th>SWRST  7</th>
<th>SPIDIS 1</th>
<th>SPIEN 0</th>
</tr>
</thead>
</table>

- **SPIEN = 1 = SPI ENABLE**
- **SPIDIS = 1 = SPI DISABLE**
  - Current transfer completes
  - All pins are inputs

- **LASTXFER**
  - 1 = NPCS rises as soon as last bit transferred out of Shift register occurs

- **SWRST = 1 = RESET = software controlled hardware reset**
  - Writing a zero to this register has no effect

- SWRST & LASTXFER cleared by hardware
Master Mode Shift Register

- **TXEMPTY**
  - Set after any programmable delay
  - MCK can be turned off in PMC at this time

- **OVRES**
  - No data loaded into RD when = 1
  - Read SPI_SR to clear

- **TDRE**
  - Cleared when SPI_TDR written
  - Used to trigger PDC transfer

- **RDRF**
  - Cleared when SPI_RDR read
Mode Fault

- Mode Fault occurs when the SPI is a master and another master has asserted NPCS0/NSS low.
  - NPCS0/NSS is normally configured as an open drain
  - Add an external pull-up to NPCS0/NSS to prevent spurious mode faults
- Enabled by default
- SPI peripheral gets disabled when fault occurs
  1. Read SPI_SR to clear MODF bit
  2. Re-enable SPI peripheral through SPI_MR SPIEN bit
Data Transfer Delays

- Three delays can be programmed
  - Delay Between Chip Selects (DLYBCS)
    • Delays assertion from one chip select to another
    • Same delay for all chip selects
  - Delay Before SPCK (DLYBS)
    • SPCK is delayed after the chip select assertion
    • Programmable for each chip select
  - Delay Between Consecutive Transfers (DLYBCT)
    • Programmable for each chip select
Transfer Delays

- Delay Between Chip Selects (**DLYBCS**)
  - Use to accommodate SPI devices with long float times
  - Delay = # of MCK periods if FDIV = 0 or # of MCK periods * 32 if FDIV = 1
  - If DLYBCS < 6 its set to 6 to guarantee a minimum delay
    - Or 6 * 32 MCK periods if FDIV = 1
Transfer Delays

- **Delay Before SPCK (DLYBS)**
  - Defines delay from NPCS valid to 1\textsuperscript{st} valid SPCK transition
  - If DLYBS = 0, the delay = \( \frac{1}{2} \) the SPCK period
  - Delay = \# of MCK periods if FDIV = 0 or \# of MCK periods \(* 32\) if FDIV = 1

![Diagram](image-url)
**Transfer Delays**

- **Delay Between Consecutive Transfers (DLYBCT)**
  - Defines delay between 2 consecutive transfers without removing the chip select
  - Delay is always inserted after each transfer and before removing the chip select if needed
  - If DLYBCT = 0 then no delay
  - Delay = \(((32 \times N \times \text{DLYBCT}) / \text{MCK}) + ((N \times \text{SCBR}) / (2 \times \text{MCK}))\)
    - $N = 1$ if FDIV = 0, else $N = 32$
Peripheral Selection

- **Fixed: PS bit in SPI_MR = 0**
  - SPI manages data flow with only one external SPI device at a time
  - PDC uses optimal 8 or 16 bit data to transfer data – memory efficient
  - PCS field in SPI_MR used to select device on the bus

- **Variable: PS bit in SPI_MR = 1**
  - SPI manages data flow with more than one external SPI device.
  - Using the PDC data is transferred in 32 bit mode.
    - 32 bit SRAM locations are encoded to select the appropriate external device automatically.
    - Not as memory efficient but allows for multiple SPI device communication without processor intervention
  - PCS field in SPI_CSR0..3 select external devices on the SPI bus

- SPI still manages the programmable data length of 8 to 16 bits in either mode.
Peripheral Chip Select Decoding

- PCSDEC bit in SPI_MR = 0
  - Chip selects NPCS0 – NPCS3 are directly connected to SPI devices
  - PCS field in SPI_MR maps directly to NPCS0 to NPCS3
    - 1 of 4 encoding

- PCSDEC bit in SPI_MR = 1
  - Chip selects NPCS0 – NPCS3 are connected to a 4 to 16 decoder
  - PCS field in SPI_TDR is binary encoded.
  - SPI_CSR0 controls external SPI devices 0-3, SPI_CSR1 controls ...

- Pins NPCS0 – NPCS3 at a logic 1 indicates no device selected
  - PCS value of 1111 is reserved for no transfers when PCSDEC = 1 or 0
  - 15 external devices can be controlled when PCSDEC = 1
  - 4 external devices can be controlled when PCSDEC = 0
Peripheral Chip Select Decoding

PCSDEC = 0

SHIFT REGISTER

MOSI

SPI_MR

PCS

TD

SPI_TDR

0 1 1 1

NPCS0

NPCS1

NPCS2

NPCS3

MOSI

SPCK

MISO

SPI 0

SPI 1

SPI 3
Peripheral Chip Select Decoding

PCSDEC = 1
### Variable Peripheral Mode

#### 32 BIT SRAM MEMORY

<table>
<thead>
<tr>
<th>PCS</th>
<th>DATA</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x100</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x104</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x108</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x10C</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x110</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x114</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x118</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x11C</td>
</tr>
<tr>
<td>X</td>
<td>LASTXFER</td>
<td>0x120</td>
</tr>
</tbody>
</table>

LASTXFER = 1. Current NPCS pin de-asserts as soon as the data transfer has occurred
Fixed/Variable & Chip Select Summary

- **Fixed**
  - Communication managed with one peripheral at a time by the processor
  - Chip Selects controlled by SPI_MR
  - Memory efficient, processor in-efficient

- **Variable**
  - Highly automated communication with up to 15 devices with no processor intervention
  - Chip Selects controlled by SPI_TDR, every write to SPI_TDR can select a different SPI device
  - Processor efficient, memory in-efficient

- **Chip Select Decoding**
  - 1 of 4 Encoding
    - Bit 0 of PCS field has priority
      - PCS = 0000 = NPCS0 = 0
      - PCS = 0101 = NPCS1 = 0
    - PCS = 0101 = NPCS1 = 0
    - Binary Encoding
      - 1111 not allowed
Variable Peripheral Mode

Can you use Variable Peripheral Mode with only 4 external SPI devices?
- Yes
  - Chip Select decoding has to be done by user’s SW.
  - PCS in SPI_CSR0..3 maps directly to NPCS pins
  - Software can create bus contention.
Slave Mode

- Slave mode characteristics defined by SPI_CSR0
- RDRF in SPI_SR rises on transfer from Shift Register to Read Data Register
- If RDRF is already high, transfer is aborted, OVRES bit is set
- When a transfer starts, data shifted out is what’s present in the shift register
SPI Peripheral Data Controller

- DMA from memory to peripheral and vice versa
  - Can be external memory on EBI for those parts that have an EBI
- 1 Master Clock Cycle needed for memory to peripheral transfer
- 2 Master Clock Cycles needed for peripheral to memory transfer
- For each channel
  - 32 bit memory pointer (incremented by byte, half-word or word)
  - 16 bit transfer count (decrements)
  - 32 bit next memory pointer (incremented by byte, half-word or word)
  - 16 bit next transfer count (decrements)
- Registers
  - Receive Pointer Register (RPR) and Transmit Pointer Register (TPR)
  - Receive Counter Register (RCR) and Transmit Counter Register (TCR)
  - Receive Next Pointer Register (RNPR) and Transmit Next Pointer Register (TNPR)
  - Receive Next Counter Register (RNCR) and Transmit Next Counter Register (TNCR)
SPI PDC Chaining Buffers

Transmit Channel Example

- When SPI_TCR = 0
  - Contents of SPI_TNPR are loaded into SPI_TPR
  - Contents of SPI_TNCR are loaded into SPI_TCR
  - SPI_TNCR is set to 0
  - Flags are updated accordingly
SPI PDC Flags

- ENDRX is set when RCR = 0
- RXBUFF is set when RCR = 0 & RNCR = 0
- ENDTX is set when TCR = 0
- TXBUFE is set when TCR = 0 & TNCR = 0

How do you program the PDC to exceed 131,070 transfers?
- Program SPI, Including Interrupt
- Load 0xFFFF into both RCR & RNCR
- Load memory pointers RPR & RNPR
- Enable PDC channel
- When ENDRX -> 1
  - Interrupt gets generated
  - Check RNCR = 0
  - Load RNCR with another value
  - Load RNPR with the next address
SPI PDC Control & Status

- Control
  - Enable
    - Writing a 1 to the “EN” bit enables the channel if the “DIS” bit is not set
  - Disable
    - Writing a 1 to the “DIS” bit disables the channel

- Status
  - 1 = transfers for that channel are enabled

<table>
<thead>
<tr>
<th>SPI_PTCR</th>
<th>TXTDIS</th>
<th>TXTEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXTDIS</td>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPI_PTSR</th>
<th>TXTEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>RXTEN</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>RXTEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
**SPI PDC Register Locations**

- Control registers start at peripheral address offset by 0x100
  - 0x100 : SPI_RPR : SPI Receive Pointer Register, Read/Write
  - 0x104 : SPI_RCR : SPI Receive Counter Register, Read/Write
  - 0x108 : SPI_TPR : SPI Transmit Pointer Register, Read/Write
  - 0x10C : SPI_TCR : SPI Transmit Counter Register, Read/Write
  - 0x110 : SPI_RNPR : SPI Receive Next Pointer Register, Read/Write
  - 0x114 : SPI_RNCR : SPI Receive Next Counter Register, Read/Write
  - 0x118 : SPI_TNPR : SPI Transmit Next Pointer Register, Read/Write
  - 0x11C : SPI_TNCR : SPI Transmit Next Counter Register, Read/Write
  - 0x120 : SPI_PTCR : SPI PDC Transfer Control Register, Write-only
  - 0x124 : SPI_PTSR : SPI PDC Transfer Status Register, Read-only

- SPI control registers for SAM7S start at address 0x FFFE 0000
- SPI PDC control registers start at address 0x FFFE 0100
SPI Interrupts

- SPI Interrupt Enable Register SPI_IER (Write Only)
  0 = No effect
  1 = Enable

- SPI Interrupt Disable Register SPI_IDR (Write Only)
  0 = No effect
  1 = Disable

- SPI Interrupt Mask Register SPI_IMR (Read Only)
  0 = Not enabled
  1 = Enabled

SPI_IER, SPI_IDR, SPI_IMR

<table>
<thead>
<tr>
<th>TXEMPTY 9</th>
<th>NSSR 8</th>
<th>TXBUFE 7</th>
<th>RXBUFF 6</th>
<th>ENDTX 5</th>
<th>ENDRX 4</th>
<th>OVRES 3</th>
<th>MODF 2</th>
<th>TDRE 1</th>
<th>RDRF 0</th>
</tr>
</thead>
</table>

SPI Interrupts

- Receive Data Register Full
- Transmit Data Register Empty
- Mode Fault Error
- Overrun Error
- End of Receive Buffer
- End of Transmit Buffer
- Receive Buffer Full
- Transmit Buffer Empty
- NSS Rising
- Transmit Registers Empty

TDRE, RDRF 1 interrupt line goes to the AIC
Read SPI_SR to determine which interrupt occurred
## SPI Status Register SPI_SR

- Receive Data Register Full
- Transmit Data Register Empty
- Mode Fault Error
- Overrun Error
- End of Receive Buffer
- End of Transmit Buffer
- Receive Buffer Full
- Transmit Buffer Empty
- NSS Rising
- Transmit Registers Empty
- SPI Enable Status

### Bit Assignments

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>TXEMPTY</td>
<td>Transmit Registers Empty</td>
</tr>
<tr>
<td>8</td>
<td>NSSR</td>
<td>NSS Rising</td>
</tr>
<tr>
<td>7</td>
<td>TXBUFE</td>
<td>Transmit Buffer Empty</td>
</tr>
<tr>
<td>6</td>
<td>RXBUFF</td>
<td>Receive Buffer Full</td>
</tr>
<tr>
<td>5</td>
<td>ENDTX</td>
<td>End of Transmit Buffer</td>
</tr>
<tr>
<td>4</td>
<td>ENDRX</td>
<td>End of Receive Buffer</td>
</tr>
<tr>
<td>3</td>
<td>OVRES</td>
<td>Overrun Error</td>
</tr>
<tr>
<td>2</td>
<td>MODF</td>
<td>Mode Fault Error</td>
</tr>
<tr>
<td>1</td>
<td>TDRE</td>
<td>Transmit Data Register Full</td>
</tr>
<tr>
<td>0</td>
<td>RDRF</td>
<td>Receive Data Register Full</td>
</tr>
</tbody>
</table>
SPI Summary

- High Speed. 55 Mb/s for SAM7S devices
- Separate baud rate generation on each chip select
- Can control up to 15 external SPI devices
- Programmable data length of 8 to 16 bits
- Highly flexible automated DMA support
  - Put the processor to sleep while transferring data
- Programmable delays on chip selects to accommodate various bus timing from different SPI IC manufacturers
- Error checking
  - Local Loop Back
  - Mode Fault Detection