Serial Communication

A. Asynchronous communication
B. Serial communication interface (SCI)
C. SCI Registers
D. Synchronous serial I/O
E. The SPI interface
F. SPI topologies and applications

The Serial Subsystems

- 68HC11 has 2 subsystems for serial interfacing
  - the serial communication interface (SCI) that can be used to connect a terminal or personal computer to the microcontroller
  - serial peripheral interface (SPI) can provide high-speed serial communication to peripherals or other microcontroller units
Asynchronous Communication Systems

- Serial communication concepts
- Protocols and flow control
- Communication channels and modems

Serial Communication Concepts

- Asynchronous communication
  - each device uses its own clock
  - the clocks must run at the same rate but do not need to be synchronized
  - the asynchronous I/O subsystem of 68HC11 is called SCI
  - an asynchronous communication system must have at least one transmitter and one or more receivers
Start and Stop Framing. Parity

- The basic unit of information is the character or data frame.
- The transmitter can send characters at any rate, so there may be delays between the transmission of each character.
- The receiver needs to know when a character starts and when it stops => character is framed by start and stop bits.
- Parity is used to detect single bit errors:
  - type: even or odd
  - the quantity of 1 bits in the data det the parity bit
Example

- Show the framing bits when the character B is sent at 7 data bits, 2 stop bits, odd parity:

  - Solution:
    - start bit: 0
    - data bits: 0100001
    - parity bit: 1
    - stop bits: 11

Data Speed, Baud

- Serial data speed is the number of bits transmitted per second (BPS)
- Baud is the rate at which the signal changes
  - Note: bauds and BPS are not equal
  - Baud rate includes the start, parity, and stop bits
  - Often the baud is used to express the number of bits transferred per second => 9600 baud means 9600 bits per second
- BPS = (nr of data bits)/(nr of frame bits) x baud
- characters per sec = baud/(nr of frame bits)
RS-232 Interface Standard

- Equipment using asynchronous serial com. normally use the RS-232 interface
- The logic levels used for RS-232 signals are:
  - Nominal levels: +12 V for logic 0; -12 V for logic 1
- This is to allow signals to be transmitted over greater distances
- This is a bipolar form of NRZ format
- The standard defines 25 different signals
- Many signals are not used => serial ports also use a DB-9 connector
- Common signals:
  - Transmit data: TxD or TD
  - Receive data: RxD or RD
  - Request to send: TSR
  - Clear to send: CTS
  - Data set ready: DSR
  - Signal ground: SG
  - Data carrier detect: DCD
  - Data terminal ready: DTR
  - Ring indicator: RI
- From normal HCMOS and TTL levels we need to use special driver chips for ...
The use of MAX 232 chip to build an 68HC11 RS-232 port
Protocols and Flow Control

- Acknowledge/Not Acknowledge (ACK/NAK) flow control
  - transmitter waits for the receiver to send ACK or NAK (negative ACK)
- Cyclic redundancy Check (CRC).
  - a special code based on the bit pattern used to check data integrity
  - A communication protocol specifies the type of CRC used
- XON/XOFF flow control.
  - XOFF: signal to tell the transmitter to stop transmission
  - XON: signal to tell the transmitter to resume transmission

Communication Channel Operation

- Communication channel is a signal path that connects 2 stations/devices
- There are 3 basic modes to transmit data along a communication channel:
  - simplex
  - half-duplex
  - full-duplex
Modems

- Modems is a contraction of modulator-demodulator
- Modem is used to send and receive serial digital data over a telephone line
- Examples ...
- **Basics of modems**
- Modem is connected to a serial port
  - dedicated circuit
- the serial port, the RS-232 data terminal equipment (DTE) -> connected to a modem, a data communication equipment (DCE) -> to a telephone line
- Transmission ...
- Receiving ...
- The audio signal is known as the carrier signal
- Tech: PSK; DPSK; QAM
Modem Handshaking Control

- **Originate modem**: modem that starts a tr
- **Answer modem**: modem being called
- For a modem to com with another they must establish connection => they must also, maintain the connection and later terminate it
- Next slide table shows the modem control signals
- A serial com IC typically has pins for at least some of the modem control signals
- Software control of general purpose I/O pin is required for any that are missing
- Note that 68HC112 does not have any modem control signals
- Some modem chips also have a DCD signal ...

### Modem Control Signal

<table>
<thead>
<tr>
<th>Signal name</th>
<th>DTE (port)</th>
<th>DCE (modem)</th>
<th>Function</th>
<th>Pin/EIA circuit label</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>Output</td>
<td>Input</td>
<td>Request to send tells the DCE that the DTE wishes to transmit.</td>
<td>4/CA</td>
</tr>
<tr>
<td>CTS</td>
<td>Input</td>
<td>Output</td>
<td>Clear to send tells the DTE that the DCE is ready to receive; DCE transmits any data sent by DTE.</td>
<td>5/CB</td>
</tr>
<tr>
<td>DTR</td>
<td>Output</td>
<td>Input</td>
<td>Data terminal ready prepares the DCE for connection to the communications channel.</td>
<td>20/CD</td>
</tr>
<tr>
<td>DSR</td>
<td>Input</td>
<td>Output</td>
<td>Data set ready indicates that the DCE is ready to operate the communications channel.</td>
<td>6/CC</td>
</tr>
<tr>
<td>RI</td>
<td>Input</td>
<td>Output</td>
<td>Ring indicator indicates that the DCE received a ring signal on the communications channel.</td>
<td>22/CE</td>
</tr>
</tbody>
</table>

1 Control signals are active low when converted to HCMOS levels.
**Handshake Control is as follows**

- To send data
  - In simplex mode
  - In duplex mode
- Establish connection

![Handshake Diagram]

---

**The Serial Communication Interface (SCI) in 68HC11**

- With additional conversion circuits the SCI can be used to communicate with remote devices
- SCI uses port D pin PD1 as TxD and PD0 as RxD
- These lines can be enabled or disabled by one of the SCI control registers (SCCR2)
- When enabled SCI subsystem has control of the respective port D lines and overrides DDRD settings
- Transmitting is a simple matter of writing bytes to a data register SCDR
  - the SCI handles the framing requirements (no parity)
- The SCI receiver automatically each framed serial character into a byte
- BAUD register is used to configure the clock
- The SCI can send break signals (SBK in SCCR2)
- Wake-up control bit (RWU)
- WAKE ...
Double Buffering in the SCI System

Transmit Operation

- System writes a byte of data to the transmit data register TDR
  - character is transferred to the output shift register
  - T8 included if enabled
- The shift register automatically outputs a start bit and the data bits and automatically adds a stop bit at the end
- BAUD register determines the shift rate
- TDRE (transmit data register empty) flag is set every time SCI transfers data from buffer to the shift register
- TDRE can be detected by polling or interrupts
  - writing the TDR buffer clears TDRE
- The system can write the second byte while the first one is still being transferred
Receive Operation

- The first low signal, a start bit, triggers the receiver to shift in the next 8 bits
  - if configured transfers 9th bit to R8 in SCCR1
- The setting of BAUD will determine the rate of sampling and shifting in
- shifting complete: content -> to RDR buffer
- Transferring this data sets the RDRF

- The receiver samples the stop bit and waits for the next start bit
- A receiver driver may read the RDR buffer any time while the next data is being shifted in
- The software should read the buffer before the next byte has shifted in completely
- RDRF is reset by reading RDR

SCDR

<table>
<thead>
<tr>
<th>SCDR</th>
<th>102h</th>
</tr>
</thead>
<tbody>
<tr>
<td>B7</td>
<td>R7/T7</td>
</tr>
<tr>
<td>B6</td>
<td>R6/T6</td>
</tr>
<tr>
<td>B5</td>
<td>R5/T5</td>
</tr>
<tr>
<td>B4</td>
<td>R4/T4</td>
</tr>
<tr>
<td>B3</td>
<td>R3/T3</td>
</tr>
<tr>
<td>B2</td>
<td>R2/T2</td>
</tr>
<tr>
<td>B1</td>
<td>R1/T1</td>
</tr>
<tr>
<td>B0</td>
<td>R0/T0</td>
</tr>
</tbody>
</table>

(SCI Data Register)

$102F$ (Receive and transmit double buffered)

SCCR1

<table>
<thead>
<tr>
<th>SCCR1</th>
<th>102h</th>
</tr>
</thead>
<tbody>
<tr>
<td>B7</td>
<td>R8</td>
</tr>
<tr>
<td>B6</td>
<td>T8</td>
</tr>
<tr>
<td>B5</td>
<td>M</td>
</tr>
<tr>
<td>B4</td>
<td>WAKE</td>
</tr>
<tr>
<td>B3</td>
<td>U</td>
</tr>
<tr>
<td>B2</td>
<td>U</td>
</tr>
<tr>
<td>B1</td>
<td>0</td>
</tr>
<tr>
<td>B0</td>
<td>0</td>
</tr>
</tbody>
</table>

(RESET = U U 0 0 0 0 0 0)

- R8 = Receive bit 8
- T8 = Transmit bit 8
- M = Mode (select character format)
  - 0 = 1 start, 8 data, 1 stop bit
  - 1 = 1 start, 8 data, ninth data, 1 stop bit
- WAKE = Wake up (0 = by idle line, 1 = by address mark)
Table 1: SCI Control Register 2

<table>
<thead>
<tr>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIE</td>
<td>TCE</td>
<td>RIE</td>
<td>ILIE</td>
<td>TE</td>
<td>RE</td>
<td>RWU</td>
<td>SBK</td>
</tr>
</tbody>
</table>

- **TIE**: Transmit Interrupt Enable
- **TCE**: Transmit Complete Interrupt Enable
- **RIE**: Receiver Interrupt Enable
- **ILIE**: Idle Line Interrupt Enable
- **TE**: Transmitter enable
- **RE**: Receiver enable
- **RWU**: Receiver Wake-up Control (0 = normal, 1 = asleep)
- **SBK**: Send Break

Table 2: SCI Status Register

<table>
<thead>
<tr>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDRE</td>
<td>TC</td>
<td>RDIF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>0</td>
</tr>
</tbody>
</table>

- **TDRE**: Transmit Data Register Empty Flag
- **TC**: Transmit Complete Flag
- **RDIF**: Receive Data Register Full Flag
- **IDLE**: Idle Line Detected Flag
- **OR**: Over-Run Error Flag
- **NF**: Noise Error Flag
- **FE**: Framing Error Flag

Table 3: SCI Baud Rate Control Register

<table>
<thead>
<tr>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCLR</td>
<td>SCP1</td>
<td>SCP0</td>
<td>RCKB</td>
<td>SCR2</td>
<td>SCR1</td>
<td>SCR0</td>
<td></td>
</tr>
</tbody>
</table>

- **TCLR, RCKB** are for tests mode only
- **SCP1, SCP0**: Serial Prescaler Selects

<table>
<thead>
<tr>
<th>SCP1</th>
<th>SCP0</th>
<th>Divide E by</th>
<th>Highest Baud Rate, E = 2 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>125.00K</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>41.666K</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>31.250K</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>13</td>
<td>9600 (+0.16%)</td>
</tr>
</tbody>
</table>

Table 4: SCI Rate Select

<table>
<thead>
<tr>
<th>SCR2</th>
<th>SCR1</th>
<th>SCR0</th>
<th>Prescaler Output</th>
<th>Highest Baud Rate, 9600 (SCP1 = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9600</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4800</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2400</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1200</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>600</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>32</td>
<td>300</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>64</td>
<td>150</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>128</td>
<td>75</td>
</tr>
</tbody>
</table>
SCI Software

- The software should initialize the SCI subsystem first – this could be a subroutine that is executed after a reset
- Other SCI I/O handler subroutines could be responsible for transmitting a byte or receiving a byte

SCI Configuration Procedure

- Config procedures
  - 1. Write BAUD register (SCP0-1, SCR0-2)
  - 2. Select the word length and wake up – write to the SCCR1 reg (M, WAKE)
  - 3. Enable interrupts, transmit, receive, and wake up as required. Write to the SCCR2 register (TIE, TCIE, RIE, ILIE, TE, RE, RWU)

- Transmit procedure/byte
  - 1. Poll SCSR or respond to the interrupt (read SCSR)

- 2. If applicable (M=1), write to T8 in the SCCR1 reg
- 3. If TDRE == 1, write to the SCDR register

- Receiver procedure
  - Poll the SCSR register or respond to the interrupt
  - If RDRF == 1, then read SCDR register
  - 3. Option: if there is an error (OR|NF|FE == 1), handle the error
  - 4. If applicable (M=1) read R8 in the SCCR1 register
Example 1 Program

- The program illustrates a polling technique to transmit and receive data. It is a loopback testing program
  - the transmit pin is connected back to the receive pin
  - its purpose is to test the serial interface, such as RS-232 port
  - the program transmits the data stored at $00 continuously to the SCI transmitter and reads it back from the SCI receiver. It stores the received byte at $01
  - the user can set configuration registers BAUD, SCCR1, and SCCR2 as desired by setting addresses $02, $03, and $04

Xtal = 8MHz, E = 2 MHz
ORG 0
TRDAT FCB $A5 ; data to tr.
RECDAT RMB 1
BAUDDAT FCB $30 ; Baud = 9600
CONF1 FCB $00 ; M=0, WAKE = 0
CONF2 FCB $0C ; TE=1, RE = 1

Initialization Block

ORG $100
LDX #REGBAS ; point to registers
LDAA BAUDDAT ; set up baud rate
STAA BAUD,X
LDAA CONF1 ; configure SCI control registers
STAA SCCR1,X
LDAA CONF2
STAA SCCR2,X
LDAA SCSR,X ; clear RDRF if set
STAA SCDR,X
LDAA TRDAT ; get data
STAA SCDR,X ; and transmit, also clears TDRE

*wait for empty transmit register
TR
BRCLR SCSR,X $80 TR

*continue when TDRE sets

*wait for full receive register
REC
BRCLR SCSR,X $20 REC

*continue when RDRF sets

LDAA SCSR,X ; read received data,
; also clears RDRF

*and store it
STAA RECDAT

*and repeat transmit loop
BRA AGN
*Listing 10.2
*This program transmits the data stored at $00 continuously
*to the SCI transmitter. User can set configuration
*registers BAUD, SCCR1, and SCCR2 as desired by, respectively,
*setting addresses $F1, $F2, and $F3.
*Warning: program set up only to handle interrupts,
*not I/O polling.
*Observe data framing with oscilloscope channel connected
*to PCl/TxD and trigger channel connected to STRB.

    ORG 0

TREAT   FCB $A5 ;data to be transmitted
BAUDDAT FCB $30 ;baud rate, initially 9600
CONF1   FCB $00 ;SCCR1 config. data
CONF2   FCB $6C ;SCCR2 config. data

*---------------------------------------------------------------

    ORG $101

*REMEMBER: Interrupt vectors should be already set up
*See 'Cros Assembly,' Appendix D.

    LDX #REGRAS ;point to registers
    SEI
    LDDA BAUDDAT ;set up baud rate
    STA BAUD,X

LDAA CONFI ;configure SCI control registers
STAA SCCR1,X
LDAA CONF2
STAA SCCR2,X

LDAA #$06 ;configure PIOC
STAA PIOC,X ;to define STRB active low pulse

LDAA SCSR.X ;clear TDRE if set
LDAA TREAT ;get data
STAA SCOR,X ;and transmit

AGN
    WAI ;and wait for empty
    ;transmit register
BRA AGN

*---------------------------------------------------------------

*SCI interrupt handler to transmit

RSCT
    LOAB SCSR.X ;clear TDRE

STAA SCDR.X ;and transmit
STAA PORTB,X ;write anything to PORTB
    ;to pulse STRB
RTI ;return
SCI Registers. The SCDR (Serial Communication Data Register)

- Both the transmit data register (TDR buffer) and the receive data register (RDR) have the address of the SCI data register (SCDR)
  - if data is written to data register SCDR, data goes to the TDR
  - if data register SCDR is read, data comes from the RDR => the MCU software cannot read back the data it has already sent
The Baud Register (BAUD)

- The crystal frequency is divided by 4 to get the bus rate E
- The prescale select bits SCP1-SCP0, select an initial division factor 1, 3, 4, 13 that drive a series of divide by 2 stages
- The rate select bits (SCR2-SCR0) determine where the RT clock will be tapped off this divider chain
- The RT clock is 16x the baud-rate frequency
- This 16x baud-rate is finally divided by 16 to get the transmitter baud-rate clock

Serial Communication Control Register (SCCR1)

- The register provides the control bits that determine word length and select the method used to wake up a sleeping receiver
- Multiple receivers ...
- Wake-up. There are two ways to wake up a receiver:
  - Idle-Line wake-up
  - Address-mark wake-up
- Mode select (character format) ...
Serial Communication Control Register 2 (SCCR2)

- SCCR2 provides the control bits that enable or disable individual functions for the SCI
- Transmit interrupt enable (TIE) ...
- Transmit complete interrupt enable (TCIE) ...
- Receive interrupt enable (RIE) ...
- Idle line interrupt enable (RIE) ...
- Transmitter/Receiver enable (TE/RE)
  - puts the PD1/PD0 as a SCI transmit/receiver pins
- Receiver wake-up control (RWU)
- Send break (SBK)
  - sends zeroes as long as SBK = 1

*Transmit a block of bytes starting at address BUFF
*and ending at address ENDBUFF
*For brevity, not all source code is shown.
*Configures different framing formats using a software
*pseudo* configuration register FRAME at address $02
*as follows:

<table>
<thead>
<tr>
<th>FRAME</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>7 bits + even parity + 2 stop bits</td>
</tr>
<tr>
<td>001</td>
<td>7 bits + odd parity + 2 stop bits</td>
</tr>
<tr>
<td>010</td>
<td>7 bits + even parity + 1 stop bit</td>
</tr>
<tr>
<td>011</td>
<td>7 bits + odd parity + 1 stop bit</td>
</tr>
<tr>
<td>100</td>
<td>8 bits + 2 stop bits</td>
</tr>
<tr>
<td>101</td>
<td>8 bits + 1 stop bit</td>
</tr>
<tr>
<td>110</td>
<td>8 bits + even parity + 1 stop bit</td>
</tr>
<tr>
<td>111</td>
<td>8 bits + odd parity + 1 stop bit</td>
</tr>
</tbody>
</table>

*data definitions can be modified by user
*Stack and vectors should be already set up
*See "Cross Assembly" in Appendix D.

LDX @EGBAS ;point to registers
SEL
LDAA BAUDAT ;disable interrupts for now
STAA BAUD.X ;set up baud rate
CLR SCCRI.X ;init. T8 = 0, X = 0.
;default configuration
STAA SCCK2.X ;enable transmitter and interrupt

*Jump to transmit code as specified by FRAME data

LDY #FRTAB ;frame data
LDA# FRAME ;mask bits 3 to 7 of FRAME
AND# #607 ;multiply by 2
ASL#
AY
LDY 0,Y
JMP 0,Y ;jump address = FRTAB+2xFRAME
;jump to selected transmit code

(*(This section contains sections of code
*to handle each type of FRAME configuration)

*(code for FRAME 0)
ZER0
*(code for FRAME 1)
ONE
*(code for FRAME 2)
TWO
*(code for FRAME 3)
THREE

*Code for FRAMES 4 and 5
*This is a sample of one section of code
*to show 8 data bits plus 2 stop bits or 1 stop bit
*-------------------------------------------
*no parity, 2 stop bits
POUR
SET SCCRI.X $50
*no parity, 1 stop bit
FIVE
LDY #BUFF
FIVES1
LDAA 0,Y ;get data
STAA SCCR.X ;transmit it
CLI WAI INY ;get next data byte
CPY WENDBUF ;until all bytes
Serial Communication Status Register (SCSR)

- SCSR provides input to the MCU internal interrupt logic circuits and for polling
- Transmit data register empty flag (TDRE)
  - to clear TDRE: read SCSR then write SCDR
- Transmit complete flag (TC)
  - sets when tr has reached an idle state, no pending data. To clear TC: read SCSR then write SCDR
- Receive data register full flag (RDRF)
  - To clear RDRF: read SCSR then read SCDR
- Idle line detect flag (IDLE)
  - set when an active receive line becomes idle
  - to clear: read SCSR then read SCDR
- Receive error flags (OR, NF, FE)
  - to clear: read SCSR then read SCDR
Receive Errors

- Overrun error flag (OR)
  - a newly received character has been shifted and the buffer is still full
- Noise error flag (NF)
  - the samples when receiver samples a bit are not identical
- Framing error flag (FE)
  - an invalid stop bit was detected
- Parity error:
  - SCI doesn’t have a built-in logic to detect parity error. However, software can be used if it is necessary
- Error detection: See Listing 10.4 of Chapter 10
Noise
Error

Bit boundary

High spike during zero bit

1 2 8 9 10 15 16

RT CLOCK AT 16 x BAUD RATE

Receiver samples bit at 8th, 9th, and 10th pulse.

In this case, one sample is high due to noise.

Framing
Error

Shift Register
Detects start bit

Serial bits
START

Shift Register
Shifting in bits

Serial bits
DATA & PARITY

Shift Register
Expects stop bit after data bits

Failure to detect stop bit sets flag FE

The receiver expects eight data bits.
Synchronous Serial I/O

- Synchronous means that the transmitter and receivers all use the same clock signal
  - their respective clocks may be out of phase
- The transmitter in a synchronous comm system sends a clock signal for receivers
- Categories of sync serial system: local transfers (built in); linking comp and peripheral devices over long distances (defined standard)

- Serial peripheral interface (SPI) – many IC use serial I/O for interfacing with a microcontroller
- The SPI subsystem can be used for both processor to peripheral or processor com
- The complete automobile control system uses many sensors and actuators that are controlled by ICs that use SPI

The Serial Peripheral Interface (SPI)

<table>
<thead>
<tr>
<th>Port D pin</th>
<th>SPI signal</th>
<th>Master mode</th>
<th>Slave mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD2</td>
<td>MISO (master in, slave out)</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>PD3</td>
<td>MOSI (master out, slave in)</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>PD4</td>
<td>SCK (serial clock)</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>PD5</td>
<td>SS (slave select)</td>
<td>Programmable</td>
<td>Input</td>
</tr>
</tbody>
</table>

The SPI subsystem of 68HC11 can be configured to interface directly with numerous standard product peripherals supplied by various manufacturers

The subsystem can be configured as master or slave
Bus Topology
Used in SPI Communication

Microcontroller
I/O With Bus Topology
Cascade Topology

Microcontroller I/O With Cascade Topology

**Legend:**
- OE = Output Enable
- SC = Shift Clock
- PI = Parallel Input
- LC = Latch Clock
- SP = Serial Shift
- PO = Parallel Output
- SA = Serial Input
- OH = Serial Output
- Reset
Configuration and Registers

SPCR

<table>
<thead>
<tr>
<th>$1029</th>
<th>SPIE</th>
<th>SPE</th>
<th>DWOM</th>
<th>MSTR</th>
<th>CPOL</th>
<th>CPHA</th>
<th>SPR1</th>
<th>SPR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

- SPIE = SPI Interrupt Enable
- SPE = SPI System Enable
- DWOM = Port D Wire-Or Mode (0 = normal, 1 = open-drain)
- MSTR = Master/Slave Select
  0 = Slave Mode
  1 = Master Mode
- CPOL = Clock Polarity, see Figure 10.14
  0 = active high clock, SCK idles low
  1 = active low clock, SCK idles high
- CPHA = Clock Phase Select, see Figure 10.14
- SPR1,0 = SPI Clock (SCK) Rate Select Bits

<table>
<thead>
<tr>
<th>SPR1</th>
<th>SPR0</th>
<th>E - clock Divided by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>

SPSR

| $1029 | SPIF | WCOL | MODF | | | | |
|-------|------|------|------| | | | |
| RESET | 0    | 0    | 0    | 0 | 0 | 0 | 0 |

- SPIF = SPI Interrupt Request
- WCOL = Write Collision Status Flag
- MODF = SPI Mode Error Interrupt Status Flag

DDRD

<table>
<thead>
<tr>
<th>$1009</th>
<th>B7</th>
<th>DDRD5</th>
<th>DDRD4</th>
<th>DDRD3</th>
<th>DDRD2</th>
<th>DDRD1</th>
<th>DDRD0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SS</td>
</tr>
</tbody>
</table>

- Pin = 0 SS input to detect mode fault
- Pin = 1 SS is general purpose output
- Slave = always input

- DDRD5 (Master)
  0 = SCK input enabled
  1 = SCK output enabled
- Slave = always input

- DDRD4 (Master)
  0 = MOSI input enabled
  1 = MOSI output enabled
- Slave = always input

- DDRD3 (Master)
  0 = MISO input enabled
  1 = MISO output enabled
- Slave = always input

- DDRD2 (Slave)
  0 = MISO output disabled
  1 = MISO output enabled
- Master = always input
Master SPI Operation

*Listing 10.5
*Demonstrate SPI byte output and input with
*slave always enabled, its SS line wired low.
*This program uses polling.
*-------------

DATA EQU 0

ORG $1000
LDX #$1000 ;point to registers
LDAA #$38 ;enable SPI outputs.
STAA D0RD,X ;note that D0S is general-purpose output
LDAA #$57 ;SPI as master, CPHA=1, CPOL=0
STAA SPDR,X ;and transmit it

POLL
TST SPSR,X ;wait for transfer complete
BPL POLL
LDAA SPDR,X ;get data from slave
BRA DONE ;this also clears flag, SPIF
D0NE
BRA DONE ;end of demo, now ACCA contains data
         ;received from slave.
Slave SPI Operation

*Listing 10.6
*Demonstrate SPI byte output and input with master.
*This slave is always enabled with SS line wired low.
*This program uses polling.

---

DATA EQU 0

ORG $100

LDX #$1000 ;point to registers
LDAA #$04 ;enable MISO output.

STAA DDRD,X ;note others forced as inputs
LDAA #$47 ;SPI as slave, CPHA=1, CPOL=0
STAA SPDR,X ;clock rate is really don't care
LDAA DATA ;get data from somewhere
STAA SPDR,X ;and send it to data register
* ;wait for master clock to shift it out
* ;and to shift master's data in

POLL
TST SPSR,X ;wait for transfer complete
BPL POLL ;stays here until master
* ;completes transfer
* LDAA SPDR,X ;get data from master
* ;this also clears flag, SPIF
DONE
BRA DONE ;end of demo, now ACCA contains data
;received from master.
Assignments

- Chapter 10 Exercises

THE END of the COURSE

THANK YOU !!!