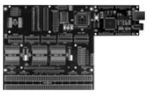


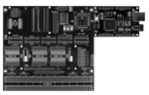
EEL 3744

Menu

- SPI Concepts
  - >Problems in serial communications
    - – Timing Synchronization: How do you line up the bit boundaries?
    - – Message Synchronization: How do you line up messages?

  - >Synchronous data solves first problem by sending clock along with message
    - – SPI performs a "physical-level" form of serial communication
  - >Section 11.4 in the S&HE Book Covers the SPI system

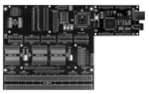
See docs/examples on web-site:
doc8331 (Sec 22), doc2595
68HC12: SPI_Master.asm, SPI_Slave.asm, SPI_Master_Tab.asm, SPI_Out.asm



EEL 3744

What is SPI?

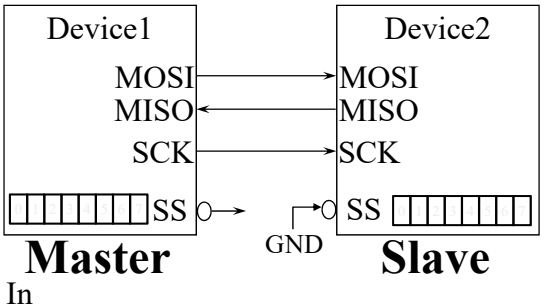
- SPI = Serial Peripheral Interface
  - >Established by Motorola
- Synchronous serial data link operating in full duplex mode
  - >Signals are carried in both directions on separate wires
- Communicate with many devices, including non-SPI devices
- May be treated as a Master or a Slave Device



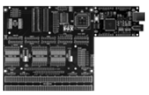
EEL 3744

Simple SPI Setup

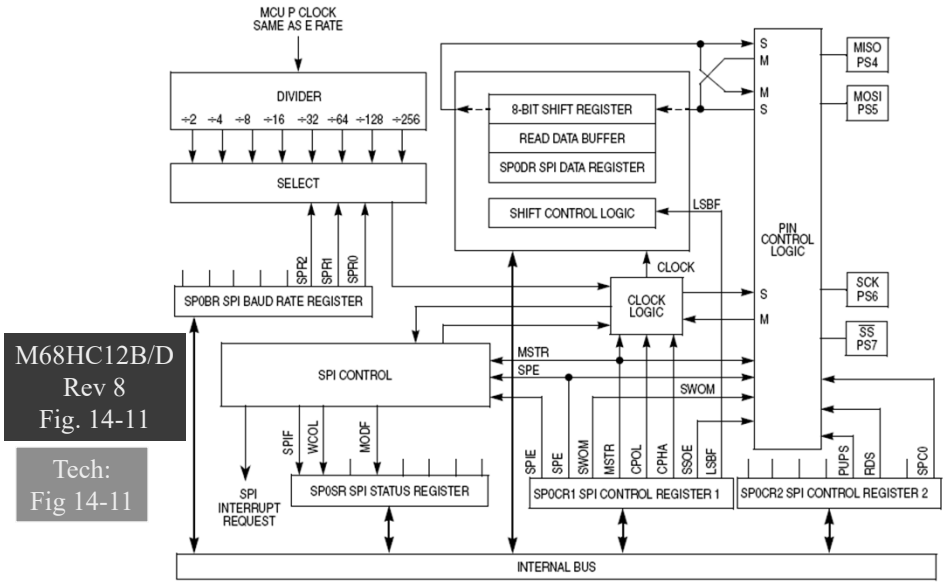
- 4 Interface Signals
  - >**SCK** - Serial Clock
    - – Output for master
    - – Input for slave
  - >**SS**(L) - Slave Select
    - – Input for slave
    - – Unrelated output for master
  - >**MOSI** - Master Out/Slave In
  - >**MISO** - Master In/Slave Out
- May use 3 or 4 pins



	Master SPI Data Register	Slave SPI Data Register
Initial	XXXX XXXX	YYYY YYYY
After 8 SCK's	YYYY YYYY	XXXX XXXX



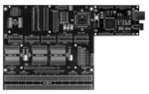
EEL 3744 **68HC12**: SPI Block Diagram



M68HC12B/D
Rev 8
Fig. 14-11

Tech:
Fig 14-11

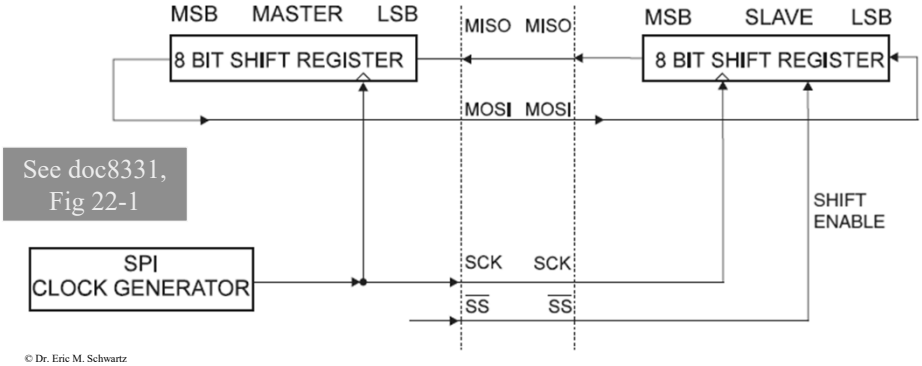
Figure 14-11. Serial Peripheral Interface Block Diagram

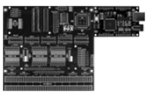


EEL 3744

XMEGA SPI

- System consists of two shift registers and a master clock generator
- Data is **shifted out** the master's MISO pin
- Data is **shifted in** the master's MOSI pin

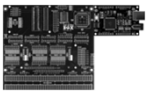




EEL 3744

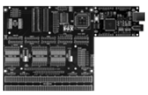
SPI Data Transfer

- SPI Master initiates and controls all data transfer
- Communication cycle is initiated by pulling the slave select low for the desired slave
  - > It is possible to ignore the SS pin and have the slave on at all times
- All data transfer is coordinated by SCK
- Data transfer is simply initiated by a Master writing data to the SPI data register
- To get input data only to a Master (i.e., no data to send to a Slave), just send "junk" data to the SPI data register


EEL 3744

SPI Details

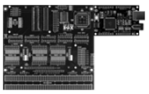
- Message Length: 8-bits
- For many SPI systems, the shift Register can be either (in XMEGA, determined by DORD in the SPI CTRL register)
  - > MSB to LSB
  - > LSB to MSB


EEL 3744

XMEGA: SPI Registers

- SPIC_CTRL
- SPIC_INTCTRL
- SPIC_STATUS
- SPIC_DATA

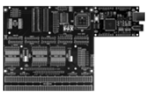
See doc8331, Sec 23



EEL 3744 **68HC12:** SPI Baud Register (SP0BR)

- SPR2-SPR0 in SP0BR: SPI bit rate select (for master)
  - > Master's bit-frequency:
    - – E-clock ÷ X, where X=2, 4, 8, …, 256 and X = $2^{(SPR+1)}$
    - – For our 2Mhz E-clock
      - ☞Bit frequencies (f_{bit}): 1MHz, 500kHz, … 7.8kHz
      - ☞Byte frequencies ($f_{byte}=f_{bit}/8$): 125kHz, 62.5kHz, … 976 Hz
      - ☞Byte periods ($T_{byte}=1/f_{byte}$): 8µs, 16µs, … 1.02ms

	7	6	5	4	3	2	1	0	
$00D2	0	0	0	0	0	SPR2	SPR1	SPR0	SP0BR
RESET	0	0	0	0	0	0	0	0	

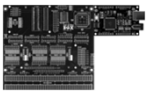


EEL 3744 **68HC12:** SPI Control Register 1 (SP0CR1)

- SPIE enables the interrupt
- SPE enables (turns on) the SPI system
- SWOM: "Port S Wired Or Mode" all or nothing
  - > 0: normal; 1: Open drain
- MSTR: 1-Master; 0-slave
- CPOL (clock polarity); CPHA (clock phase)
- SSOE enables the SS.L pin as an output on the master if SSOE = 1 and DDRS7=1
- LSBF: Least significant bit first
  - > 1: LSB first; 0: MSB first

For a master, MSTR must be set at same time or before SPE.

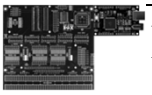
	7	6	5	4	3	2	1	0	
$00D0	SPIE	SPE	SWOM	MSTR	CPOL	CPHA	SSOE	LSBF	SP0CR1
RESET	0	0	0	0	0	0	0	0	



EEL 3744 **68HC12:** SPI Control Register 2 (SP0CR2)

- PUPS: 1-pull-up on; 0-pull-up off
- RDS: 1-reduced drive Port S; 0-reduced drive off
- SPC0:
  - > Serial Pin Control 0
  - > 1-Bidirectional mode; 0-normal two-wire mode

	7	6	5	4	3	2	1	0	
$00D1	0	0	0	0	PUPS	RDS	0	SPC0	SP0CR2
RESET	0	0	0	0	1	0	0	0	

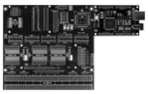


EEL 3744 **68HC12:** DDRS and SP0SR

	7	6	5	4	3	2	1	0	
$00D7	**DDRS7**	**DDRS6**	**DDRS5**	**DDRS4**	DDRS3	DDRS2	**DDRS1**	**DDRS0**	DDRS
RESET	0	0	0	0	0	0	0	0	
Used by	SS.L	SCK	MOSI	MISO	—	—	TxD0	RxD0	

- SPIF: SPI transfer complete flag
  - > Set automatically at end of SPI transfer
  - > Cleared by reading SP0SR, followed by read or write of SP0DR
- WCOL: write collision error flag
  - > Set if write to SP0DR when transfer is in progress
- MODF: mode-fault error flag
  - > Set if configured as a master and SS signal goes low

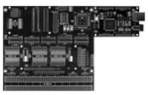
	7	6	5	4	3	2	1	0	
$00D3	**SPIF**	**WCOL**	—	**MODF**	—	—	—	—	SP0SR
RESET	0	0	0	0	0	0	0	0	


EEL 3744

68HC12: SPI Data Register (SP0DR)

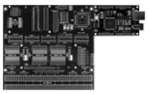
	7	6	5	4	3	2	1	0	
$00D5	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	SP0DR

- SP0DR: SPI Data Register
- A write to this register (for master only) starts the transmission and/or reception of a byte
- SP0DR is actually two registers with the same address (one for transmit and one for receive), just like SC0DR for the SCI system


EEL 3744

68HC12: SPI Concepts

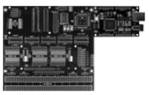
- SPI Details
  - > Message Length: 8-bits
  - > Can simultaneously transmit & receive serial data
  - > A Master/Slave mode allows a 68HC12 master to communicate with several slaves (including other 68HC12's)
  - > The SS pin allows for a particular slave to be selected
  - > Speed of transfer is maximum 1MHz for master (with E-clock=2Mhz)
    - – There are 8 speed options
  - > Clock Polarity & Clock Phase are programmable
  - > The SPI shares Port S pins (PS_7-PS_4)
    - – So use DDRS to set pin directions



EEL 3744

68HC12: SPI Concepts

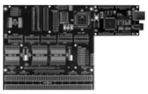
- SPI Details (continued)
  - > Even when SPI is enabled, DDRS controls direction of PS_7-PS_4
    - – Safest thing is to configure DDRS for the direction the SPI wants
  - > **Receive is double buffered; transmit is single buffered**
  - > An interrupt can be generated on completion of the transmission/reception of a byte
  - > SCK is an output when configured as a master, an input if configured as slave
  - > On both master & slave SPIs, the data is shifted on one edge of SCK and sampled on the opposite edge, where the data is stable



EEL 3744

68HC12: SPI Concepts

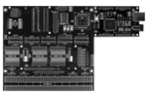
- SPI Details (continued)
  - >MISO and MOSI are the input and output for a master, and output and input for a slave, respectively
  - >If using 1 master and multiple slaves, the master drives data out its SCK and MOSI pins to the SCK and MOSI pins of the slaves. One selected slave device optionally drives data out its MISO to the MISO of the master while the other slaves have there MISO lines tri-stated (Hi-Z)
  - >On a slave, the active-low SS pin is used to enable the SPI system
    - – If SS=High (false), the device ignores SCK and makes MISO=Hi-Z.
    - – MODF←1 if a **master** device has its SS.L = Low



EEL 3744

68HC12: SPI Concepts

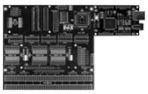
- SPI Details (continued)
  - > On a master device, the SS pin can optionally serve as an error-detection input for the SPI or a general-purpose PS_7 output not affecting the SPI
  - > If you are short on I/O ports, you can use the SPI and shift registers to expand the I/O with only 3 pins (MISO, MOSI, SCK) [see later slides for examples]
  - > For serial outputs, a write to the SPI will cause 8 bits to be shifted out of the SPI output pin (either MISO or MOSI) with a train of 8 SCK clock pulses
  - > Whether configure as master or slave, when transmission is complete, the SPI system sets the SPIF flag



EEL 3744

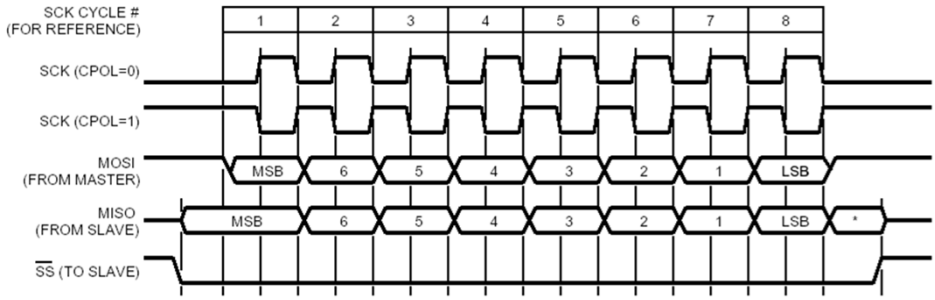
68HC12: SPI Concepts

- SPI Details (continued)
  - > In a slave device, the program should store into SP0DR immediately after the SPIF is set
  - > In a master device, the program can store into the data register any time after the flag is set
  - > In either case, if a store occurs while transmission is in progress, the SPI sets the WCOL flag in the SPSR
  - > If the SPI is operating at a high baud rate, it may not be practical to use the SPIF, especially with interrupts
    - – Delays are better in that case and have much less overhead
  - > The interrupt vector for the SPI is $FFD8-FFD9
  - > The pseudo-vector with D-Bug4744 is $080F-0811



EEL 3744 **68HC11:** SPI Clock Phase and Polarity (CPHA=0)

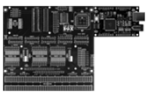
Tech: Fig 14.12



68HC11 RM: Fig 8.1

Figure 8-1. CPHA Equals Zero SPI Transfer Format

- Can not use CPHA=0 if slave's SS is grounded. [RM: page 8-3]
  - > If CPHA=0, SS must be turned off between successive serial bytes
  - > Weird, but true!



EEL 3744 **68HC11:** SPI Clock Phase and Polarity (CPHA=1)

Tech: Fig 14.13

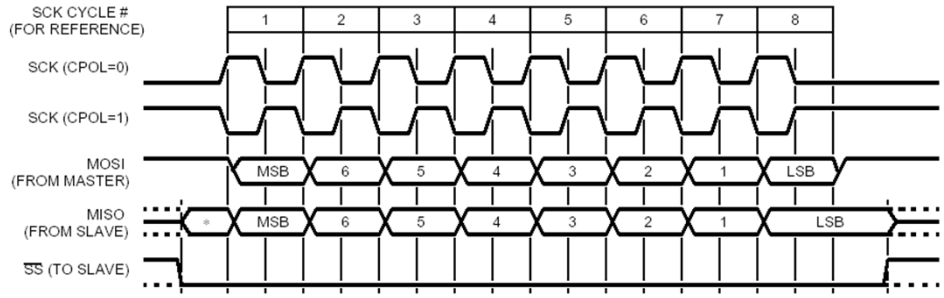
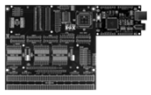
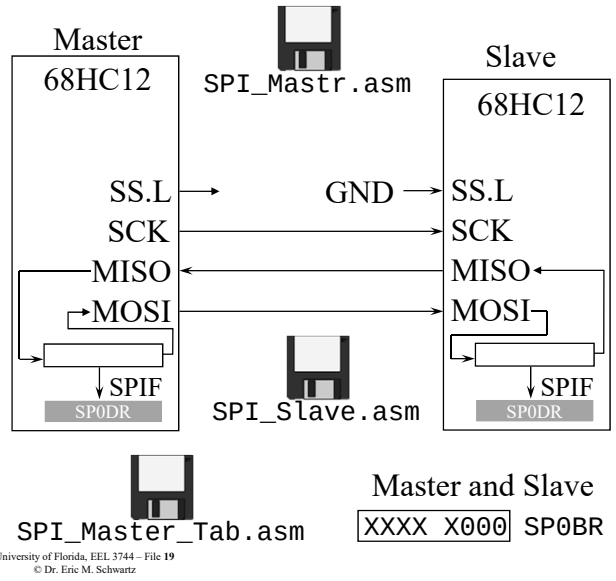


Figure 8-2. CPHA Equals One SPI Transfer Format



EEL 3744

Ex: 1 Master, 1 Slave



Master

1110 XXXX DDRS

1101 0100 SP0CR1
SPIE

*XXX XXXX SP0SR
SPIF

DDDD DDDD SP0DR

Slave

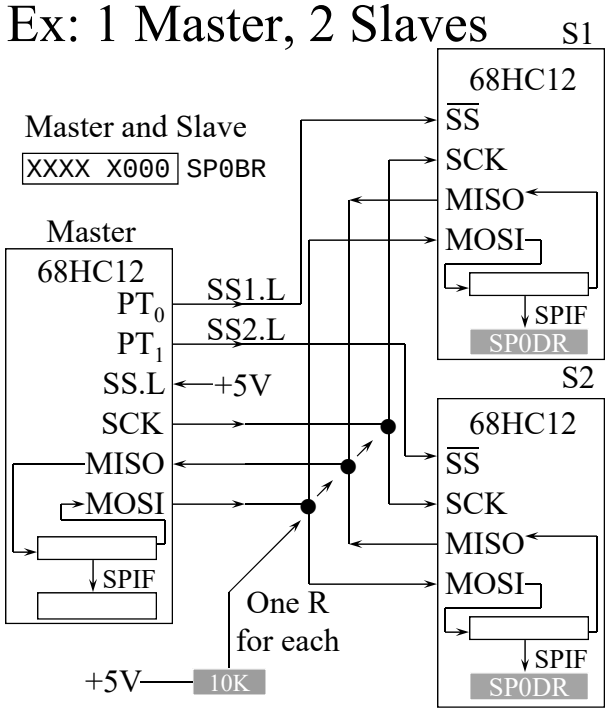
0001 XXXX DDRS

1100 0100 SP0CR
SPIE

*XXX XXXX SP0SR
SPIF

DDDD DDDD SP0DR

Ex: 1 Master, 2 Slaves



0110 XXXX DDRS

0111 1100 SP0CR1
SPIE

*XXX XXXX SP0SR
SPIF

DDDD DDDD SP0DR

Slave

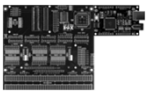
0001 XXXX DDRS

1110 1100 SP0CR
SPIE

*XXX XXXX SP0SR
SPIF

DDDD DDDD SP0DR

22

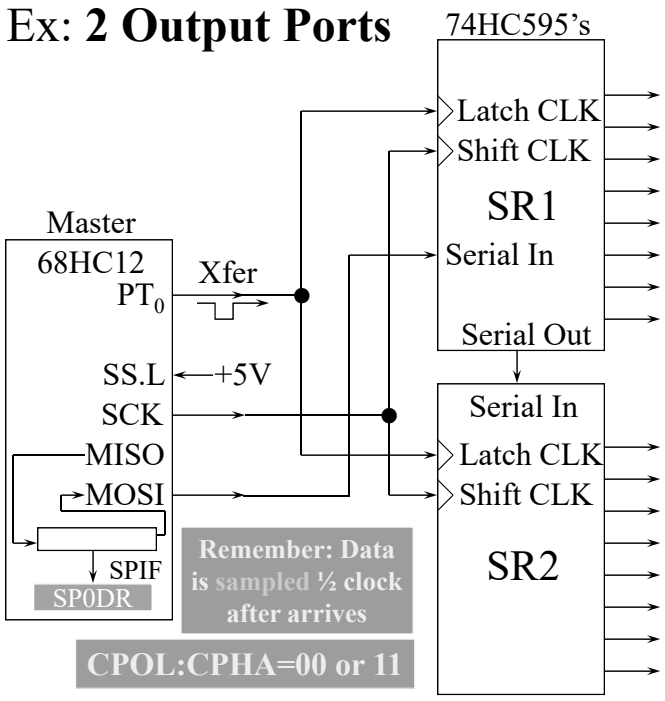


EEL 3744

SPI Concepts

- **I/O Serial Ports** - Microcontroller Expansion
- The most limited resource of a microcontroller is the number of I/O lines (pins) available
- The SPI and shift registers can expand the I/O capabilities with only 3 pins (MISO, MOSI, SCK)
- A write to the SPI transfers serially 8 bits with 8 clock pulses
- Shift registers receive the serial data and convert it to parallel for I/O processing

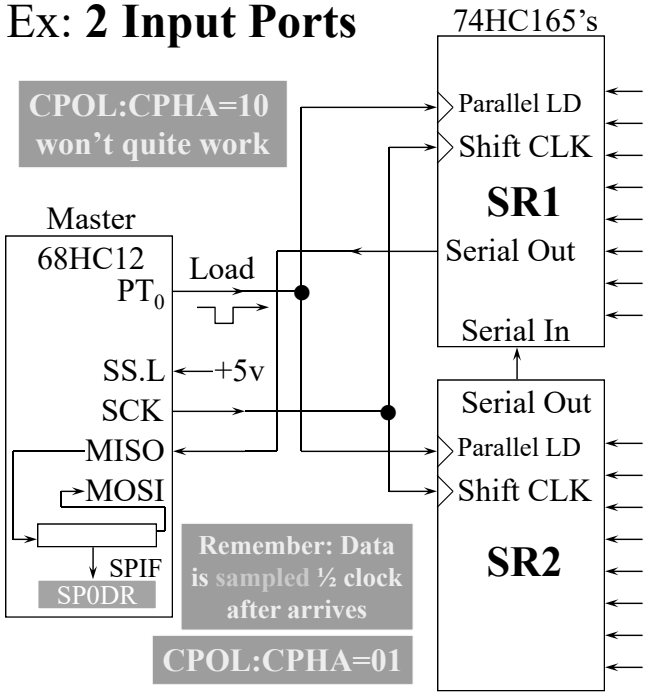
Ex: **2 Output Ports**



How to Program

- Make PT_0 high (1)
- Calculate the 2 bytes for output
- Send SR2's byte to the SPI (wr SP0DR)
  - 8-bits shift into SR2
- Send SR1's byte to the SPI (wr SP0DR)
  - 8-bits shift into SR1
- Send 0 then 1 (low pulse) to PT_0
  - Parallel pins updated on the rising edge of low pulse
- 16 output bits are now in place, but parallel I/O pins are unaffected

Ex: **2 Input Ports**

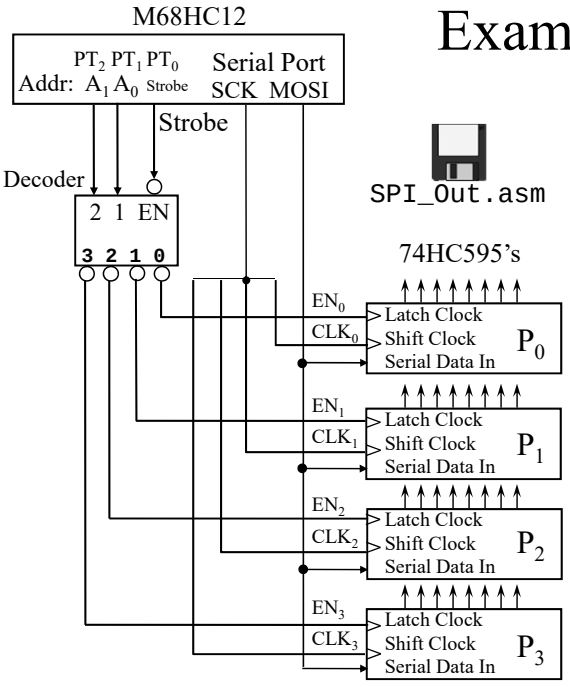


How to Program

- Make PT_0 high (1)
- Send 0 then 1 (low pulse) to PT_0
  - Shift registers are updated on rising edge of Parallel LD
- Store garbage data (XX) to SP0DR to start SPI
  - Starts SPI clock; 8 bits shifted into MISO
- Read SR1's byte into the SPI (read SP0DR)
- Save the input byte (SR1)
- Store garbage data (XX) to SP0DR to start SPI
- Read SR2's byte into the SPI (read SP0DR)
- Save the input byte (SR2)
- 16 input bits are now read, but parallel I/O pins are unaffected

25

Example

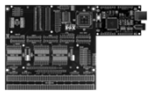


How to Program

- Write a high (1) to Strobe on the output port
  - > Turns off decoder
- Select a port, say, P_i
- Write a 2-bit address (A_1A_0) to the output port (without changing the Strobe pin)
- Write a byte to SP0DR
  - > Data shifted to P_i with SCK
- When SPIF=1, send 0 then 1 (a low pulse) to Strobe
  - > Parallel pins on P_i update on the rising edge of Strobe

CPOL:CPHA=11

27


EEL 3744

The End!