CS/ECE 6780/5780

Al Davis

Today's topics:

- **Last lecture**
  - **general serial I/O concepts**
  - **more specifics on asynchronous SCI protocol**
- **Today**
  - **specifics of synchronous SPI**
  - **details of the SCI programming ritual**

Synchronous = SPI (3 options)

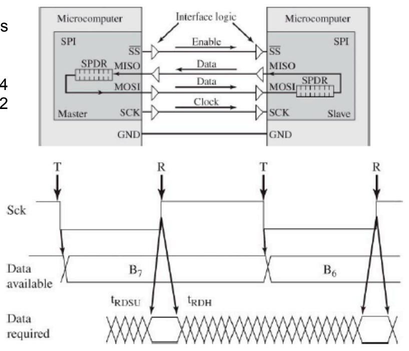
Two devices communicating with SCI operate at same frequency but have 2 separate (not synchronized) clocks.

Two devices communicating with SPI operate using the same (synchronized) clock.

Master device creates the clock while slave device(s) use the clock to latch data in or out.

SPI Master/Slave Example

9S12C32 pins
SS' on PM3
SCK on PM5
MOSI on PM4
MISO on PM2



SPI Fundamentals

Motorola SPI includes four I/O lines:

$\overline{SS}$ - slave select, used by master to indicate the channel is active.
SCK - 50% duty cycle clock generated by the master.
MOSI (master-out slave-in) - data line driven by master.
MISO (master-in slave-out) - data line driven by slave.

Transmitting device uses one edge of clock to change data, and receiving device uses other edge to accept data.

When data transfer occurs combined 16-bit register is serially shifted eight bit positions (data exchanged).

SPDR: 8-bit register but linked to form a distributed 16-bit register

More SPI Fundamentals

Common control features of the SPI module include:

- A baud rate control register
- A mode bit in the control register to select master versus slave, clock polarity, clock phase.
- Interrupt arm bit
- Ability to make outputs open-drain.

Common status bits for the SPI module include:

- SPIF, transmission complete
- WCOL, write collision
- MODF, mode fault

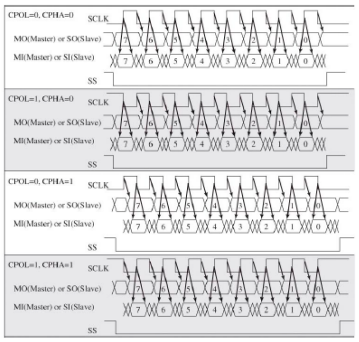
Mode fault occurs when master and slave synchronization is wrong – e.g. 2 masters

SPI Pseudo Code

```
TRANSMIT  Set n=7                     Bit counter
TLOOP     On fall of Sck, set Data=bn Output bit
          Set n=n-1
          Goto TLOOP if n>=0
          Set Data=1                  Idle output

RECEIVE   Set n=7                     Bit counter
RLOOP     On rise of Sck, read data
          Set bn=Data                 Input bit
          Set n=n-1
          Goto RLOOP if n>=0
```

SPI Modes



CPOL sets SCLK polarity – e.g. what is IDLE

CPHA sets even or odd clock edges for the receiver shift register

Note data transfer is simultaneous exchange of data:

Master SPDR ➔ Slave SPDR
||
Slave SPDR ➔ Master SPDR

9S12C32 SPI Details (Port M)

Uses four pins, $PM3 = \overline{SS}$, $PM5 = SCLK$, $PM4 = MOSI$, and $PM2 = MISO$.

If 6812 is master, set **DDRM** to make PM5, PM4, and PM3 outputs.

Can be in **run**, **wait**, or **stop** mode.

				4 MHz		24 MHz	
SPR2	SPR1	SPR0	Div	Freq	Bit Time	Freq	Bit Time
0	0	0	2	2 MHz	500 ns	12 MHz	83.3 ns
0	0	1	4	1 MHz	1 μs	6 MHz	166.7 ns
0	1	0	8	500 kHz	2 μs	3 MHz	333.3 ns
0	1	1	16	250 kHz	4 μs	1.5 MHz	666.7 ns
1	0	0	32	125 MHz	8 μs	750 kHz	1.33 μs
1	0	1	64	62.5 kHz	16 μs	375 kHz	2.67 μs
1	1	0	128	31.25 kHz	32 μs	187.5 kHz	5.33 μs
1	1	1	256	15.625kHz	64 μs	93.75 kHz	10.67 μs

SPIBR register::= SPI Bit Rate register
note weird divisor: $2^{SPR+1} = 2^{SPR}*2$

Run/Wait/Stop Modes

- **Run – normal operation**
- **Wait – low power mode**
  - **if SPISWAI (SPICR2[1]) is clear – same as run mode**
  - **if SPISWAI is set – wait and clock generator is turned off**
  - **if master**
    - » **any transmission stops if SPISWAI is set**
    - » **resumes when SPISWAI is cleared**
  - **if slave**
    - » **transmission continues to remain in synch with master**
- **Stop**
  - **annoying – have yet to find how to set this mode**
    - » **bonus points if someone can point me to the right answer**
  - **→ SPI inactive – consumes even less power**

SPI Control Registers

SPIDR is 8-bit register used for both input and output.

SPICR1 register species SPI mode of operation.

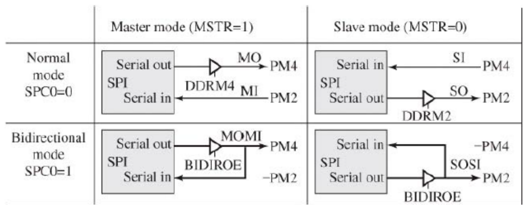
- **SPE**: enables the SPI system. bit 7
- **SPIE**: arms interrupts on the **SPIF** flag. bit 6
- **SPTIE**: arms interrupts on the **SPTEF** flag. bit 5
- **LSBF**: if 1 transmits least significant bit first. bit 0

SPISR register contains flags for the SPI system.

- **SPIF**: indicates that new data is available to be read.
- **SPTEF**: indicates that SPI data register can accept new data.
- **MODF**: mode error interrupt status flag.

see table 7.15 in the text for full layout of controls

SPI Bidirectional Mode

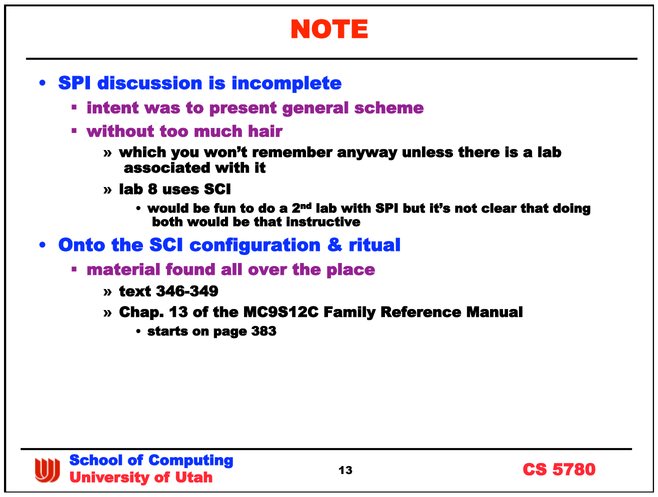


SPC0 = SPICR2[0]
BIDIROE = SPICR2[3]

SPI Mode Selections

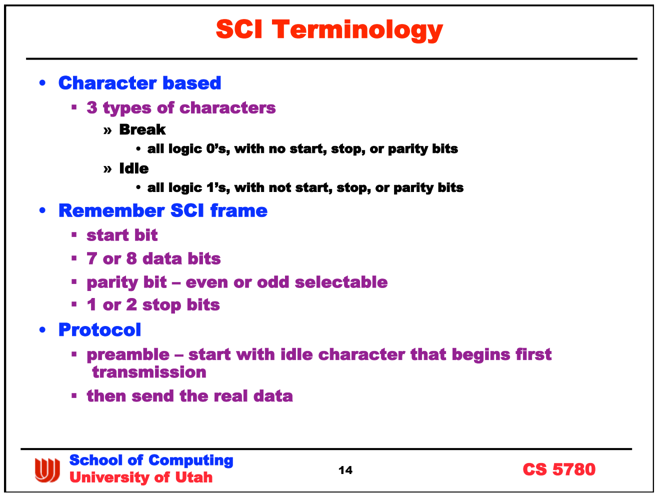
MODFEN	SSOE	Master Mode (MSTR=1)	Slave Mode (MSTR=0)
0	0	PM3 not used with SPI	PM3 is $\overline{SS}$ input
0	1	PM3 not used with SPI	PM3 is $\overline{SS}$ input
1	0	PM3 is $\overline{SS}$ input w/MODF	PM3 is $\overline{SS}$ input
1	1	PM3 is $\overline{SS}$ output	PM3 is $\overline{SS}$ input

Pin Mode	MSTR	SPC0	BIDIROE	MISO	MOSI
Normal	1	0	X	Master In	Master Out
Bidirectional	1	1	0	MISO not used	Master In
			1		Master I/O
Normal	0	0	X	Slave Out	Slave In
Bidirectional	0	1	0	Slave In	MOSI not used
			1	Slave I/O	



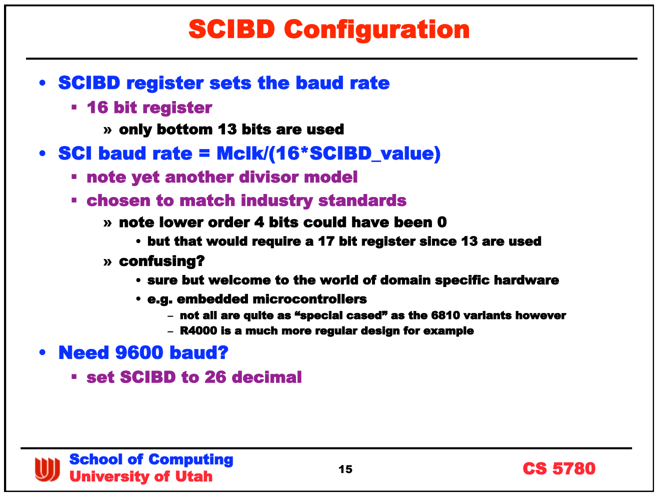
NOTE

- **SPI discussion is incomplete**
  - **intent was to present general scheme**
  - **without too much hair**
    - » **which you won't remember anyway unless there is a lab associated with it**
    - » **lab 8 uses SCI**
      - **would be fun to do a 2nd lab with SPI but it's not clear that doing both would be that instructive**
- **Onto the SCI configuration & ritual**
  - **material found all over the place**
    - » **text 346-349**
    - » **Chap. 13 of the MC9S12C Family Reference Manual**
      - **starts on page 383**



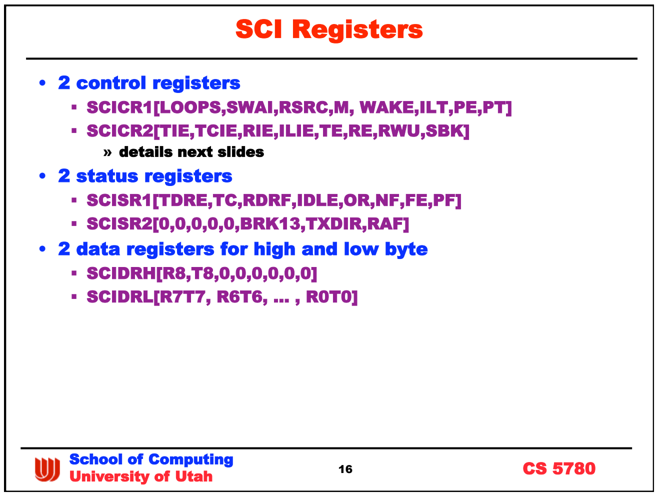
SCI Terminology

- **Character based**
  - **3 types of characters**
    - » **Break**
      - **all logic 0's, with no start, stop, or parity bits**
    - » **Idle**
      - **all logic 1's, with not start, stop, or parity bits**
- **Remember SCI frame**
  - **start bit**
  - **7 or 8 data bits**
  - **parity bit – even or odd selectable**
  - **1 or 2 stop bits**
- **Protocol**
  - **preamble – start with idle character that begins first transmission**
  - **then send the real data**



SCIBD Configuration

- **SCIBD register sets the baud rate**
  - **16 bit register**
    - » **only bottom 13 bits are used**
- **SCI baud rate = Mclk/(16*SCIBD_value)**
  - **note yet another divisor model**
  - **chosen to match industry standards**
    - » **note lower order 4 bits could have been 0**
      - **but that would require a 17 bit register since 13 are used**
    - » **confusing?**
      - **sure but welcome to the world of domain specific hardware**
      - **e.g. embedded microcontrollers**
        - – **not all are quite as "special cased" as the 6810 variants however**
        - – **R4000 is a much more regular design for example**
- **Need 9600 baud?**
  - **set SCIBD to 26 decimal**



SCI Registers

- **2 control registers**
  - **SCICR1[LOOPS,SWAI,RSRC,M, WAKE,ILT,PE,PT]**
  - **SCICR2[TIE,TCIE,RIE,ILIE,TE,RE,RWU,SBK]**
    - » **details next slides**
- **2 status registers**
  - **SCISR1[TDRE,TC,RDRF,IDLE,OR,NF,FE,PF]**
  - **SCISR2[0,0,0,0,0,BRK13,TXDIR,RAF]**
- **2 data registers for high and low byte**
  - **SCIDRH[R8,T8,0,0,0,0,0,0]**
  - **SCIDRL[R7T7, R6T6, ... , R0T0]**

SCICR1 Configuration

Bit 0 - Parity Type (PT)
- 0 - Even parity
- 1 - Odd parity

Bit 1 - Parity Enable (PE)
- 0 - Disable parity
- 1 - Enable parity

Bit 2 - Idle Line Type (ILT)
- 0 - Idle character bit count begins after start bit
- 1 - Idle character bit count begins after stop bit

Bit 3 - Wakeup Condition (WAKE)
- 0 - Idle line (idle condition on RxD) wakeup
- 1 - Address mark (1 in MSB of a received char) wakeup

More SCICR1 Configuration

Bit 4 - Data Format (M)
- 0 - 1 start bit, 8 data bits, 1 stop bit
- 1 - 1 start bit, 9 data bits, 1 stop bit

Bit 5 - Receiver Source (RSRC)
- 0 - Internal receiver to transmitter connection
- 1 - External receiver to transmitter connection (via the TxD pin)

Bit 6 - SCI Stop in Wait Mode (SCISWAI)
- 0 - SCI enabled in wait mode
- 1 - SCI disabled in wait mode

Bit 7 - Loop Select (LOOPS)
- 0 - Normal operation
- 1 - Loop operation (SCI received section is disconnected from the RxD pin allowing the RxD pin to be used for GPIO.)

SCICR2 Configuration

Bit 0 - Send Break (SBK)
- 0 - No break characters
- 1 - Transmit break characters

Bit 1 - Receiver Wakeup (RWU)
- 0 - Normal operation
- 1 - Enables wakeup and inhibits receiver interrupts.

Bit 2 - Receiver Enable (RE)
- 0 - Disabled
- 1 - Enabled

Bit 3 - Transmitter Enable (TE)
- 0 - Disabled
- 1 - Enabled

More SCICR2 Configuration

Bit 4 - Idle Line Interrupt Enable (ILIE)
- 0 - IDLE interrupts disabled
- 1 - IDLE interrupts enabled

Bit 5 - Receiver Full Interrupt Enable (RIE)
- 0 - RDRF and OR interrupts disabled
- 1 - RDRF and OR interrupts enabled

Bit 6 - Transmission Complete Interrupt Enable (TCIE)
- 0 - TC interrupts disabled
- 1 - TC interrupts enabled

Bit 7 - Transmitter Interrupt Enable (TIE)
- 0 - TDRE interrupts disabled
- 1 - TDRE interrupts enabled

OR signals overrun (next byte ready to be received from the Rx shift register but the SCDR is already full – buffer overrun)
RDRF – signals receive data register full

SCISR1 Configuration

Bit 0 - Parity Error (PF)
- 0 - No parity error
- 1 - Parity error
- Clear PF by reading SCISR1 followed by SCIDRL. Doesn't get set in case of OR.

Bit 1 - Framing Error (FE) no stop bit detected
- 0 - No framing error
- 1 - Framing error
- Clear FE by reading SCISR1 with FE set followed by SCIDRL. Doesn't get set in the case of OR. When sets prohibits further data reception.

Bit 2 - Noise Flag (NF) each bit is 3x oversampled – noise if they vary
- 0 - No noise
- 1 - Noise
- Clear NF by reading SCISR1 followed by SCIDRL. Doesn't get set in the case of OR.

More SCISR1 Configuration

Bit 3 - Overrun (OR)
- 0 - No overrun
- 1 - Overrun
- Incoming data is lost, but the current data is intact. Clear OR by reading SCISR1 with OR set followed by SCIDRL.

Bit 4 - Idle Line (IDLE)
- 0 - Receiver input is active or has never become active since last IDLE flag clear
- 1 - Receiver input is idle
- Clear IDLE flag by reading SCISR1 with IDLE set followed by SCIDRL.

Bit 5 - Receive Data Register Full (RDRF)
- 0 - Data not available in SCI data register
- 1 - Received data available in SCI data register
- Clear RDRF by reading SCISR1 with RDRF set followed by SCIDRL.

Even More SCISR1 Configuration

Bit 6 - Transmit Complete (TC)
- 0 - Transmission in progress
- 1 - No transmission in progress
- Clear TC by reading SCISR1 with TC set then writing to SCIDRL. TC is set when the TDRE flag is set and no data, preamble, or break character is being transmitted.

Bit 7 - Transmit Data Register Empty (TDRE)
- 0 - No byte transferred to the transmit shift register
- 1 - Byte transferred to transmit shift register
- Clear TDRE by reading SCISR1 with TDRE set followed by writing to SCIDRL.

SCISR2 Configuration

Bit 0 - Receiver Active (RAF)
- 0 - No reception in progress
- 1 - Reception in progress

Bit 1 - Transmitter Pin Data Direction in Single-Wire Mode (TXDIR)
- 0 - TxD pin used as an input in Single-Wire mode
- 1 - TxD pin used as an output in Single-Wire mode

Bit 2 - Break Transmit Character Length (BK13)
- 0 - Break character is 10 or 11 bits long
- 1 - Break character is 13 or 14 bits long

More SCISR2 Configuration

Bit 0 - Receiver Active (RAF)
- 0 - No reception in progress
- 1 - Reception in progress

Bit 1 - Transmitter Pin Data Direction in Single-Wire Mode (TXDIR)
- 0 - TxD pin used as an input in Single-Wire mode
- 1 - TxD pin used as an output in Single-Wire mode

Bit 2 - Break Transmit Character Length (BK13)
- 0 - Break character is 10 or 11 bits long
- 1 - Break character is 13 or 14 bits long

SCI Data Register Configuration

SCIDRL is used for bits 0-7 for transmit and receive.

SCIDRH bit 6 is the ninth data bit transmitted when in 9-bit mode.

SCIDRH bit 7 is the ninth data bit received when in 9-bit mode.

When running in 9-bit mode access SCIDRH before SCIDRL.

Finally Some Code

- **SCI Initialization example**

```
SCIBD = 26; //9600 baud
SCICR1 = 0x00; //no parity, 8 data bits, normal operation
SCICR2 = 0x2C; //receiver & transmitter enable,
               //RDRF interrupt enable
```

SCI Transmit Ritual

- **Note somewhat strange – TDRE is SCISR1[7]**
  - **what's missing?**

Check for TDRE by reading SCISR1.
Write data to SCIDRL.

```
if(SCISR1 & TDRE) {
  SCIDRL = data;
}
```

SCI Transmit Ritual

- **Note somewhat strange – TDRE is SCISR1[7]**
  - **what's missing?**
    - » **TDRE isn't defined – it's a bit position**
    - » **hence**
      - **#define TDRE 0x80**

Check for TDRE by reading SCISR1.
Write data to SCIDRL.

```
if(SCISR1 & TDRE) {
  SCIDRL = data;
}
```

SCISR1 Receive Ritual

- **Similar situtation**
  - **RDRF = SCISR1[5]**
    - » **similar deal**
    - » **need #define RDRF 0x20**

Check for RDRF by reading SCISR1.
Read data from SCIDRL.

```
if(SCISR1 & RDRF) {
  data = SCIDRL;
}
```

SCI Initialization

```
#define TDRE 0x80
#define RDRF 0x20
#define TXINT 0x80
void SCI_Init(void){
asm sei
  RxFifo_Init(); // empty FIFOs
  TxFifo_Init();
  SCIBD = 52;    // 9600 bits/sec
  SCICR1 = 0;    // M=0, no parity
  SCICR2 = 0x2C; // enable, arm RDRF
  asm cli        // enable interrupts
}
```

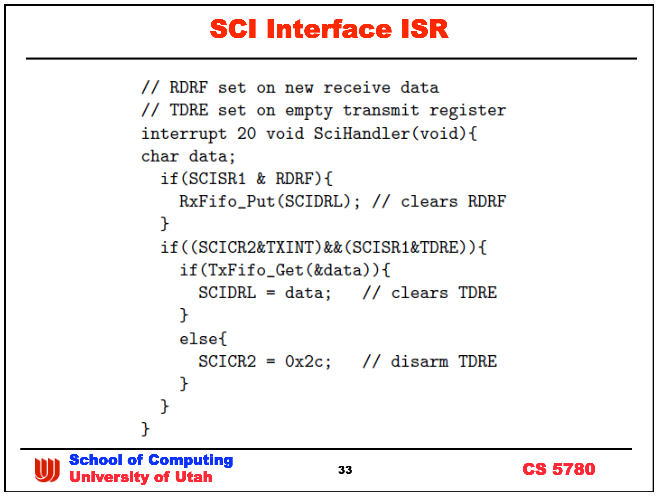
Weaknesses or Errors?

SCI Initialization

```
void SCI_Init(void){
asm sei
  RxFifo_Init(); // empty FIFOs
  TxFifo_Init();
  SCIBD = 52;    // 9600 bits/sec
  SCICR1 = 0;    // M=0, no parity
  SCICR2 = 0x2C; // enable, arm RDRF
  asm cli        // enable interrupts
}
```

- **Remember (1 error, & crappy comments)**

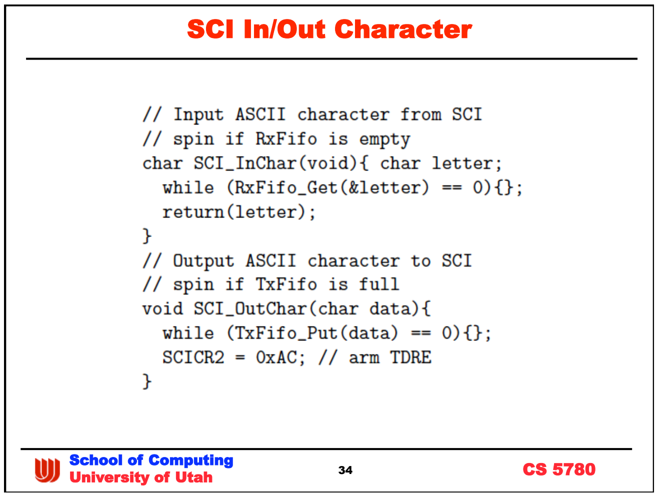
```
SCIBD = 26; //9600 baud
SCICR1 = 0x00; //no parity, 8 data bits, normal operation
SCICR2 = 0x2C; //receiver & transmitter enable,
             //RDRF interrupt enable
```



SCI Interface ISR

```
// RDRF set on new receive data
// TDRE set on empty transmit register
interrupt 20 void SciHandler(void){
char data;
  if(SCISR1 & RDRF){
    RxFifo_Put(SCIDRL); // clears RDRF
  }
  if((SCICR2&TXINT)&&(SCISR1&TDRE)){
    if(TxFifo_Get(&data)){
      SCIDRL = data;   // clears TDRE
    }
    else{
      SCICR2 = 0x2c;   // disarm TDRE
    }
  }
}
```

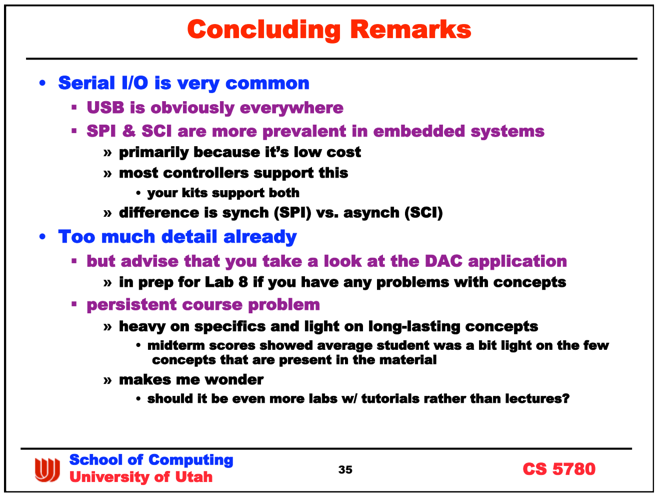
School of Computing
University of Utah 33 CS 5780



SCI In/Out Character

```
// Input ASCII character from SCI
// spin if RxFifo is empty
char SCI_InChar(void){ char letter;
  while (RxFifo_Get(&letter) == 0){};
  return(letter);
}
// Output ASCII character to SCI
// spin if TxFifo is full
void SCI_OutChar(char data){
  while (TxFifo_Put(data) == 0){};
  SCICR2 = 0xAC; // arm TDRE
}
```

School of Computing
University of Utah 34 CS 5780



Concluding Remarks

- **Serial I/O is very common**
  - **USB is obviously everywhere**
  - **SPI & SCI are more prevalent in embedded systems**
    - » **primarily because it's low cost**
    - » **most controllers support this**
      - **your kits support both**
    - » **difference is synch (SPI) vs. asynch (SCI)**
- **Too much detail already**
  - **but advise that you take a look at the DAC application**
    - » **in prep for Lab 8 if you have any problems with concepts**
  - **persistent course problem**
    - » **heavy on specifics and light on long-lasting concepts**
      - **midterm scores showed average student was a bit light on the few concepts that are present in the material**
    - » **makes me wonder**
      - **should it be even more labs w/ tutorials rather than lectures?**

School of Computing
University of Utah 35 CS 5780