Interrupts IV

ECE 3710

I remember the time I was kidnapped and they sent back a piece of my finger to my father. He said he wanted more proof.

- Rodney Dangerfield

ex: alarm system monitor

if sensors trip they output 0, do:

1. trigger alarm
2. lock doors
3. execute notification routine

connected to `PD0`

(many sensors share same pin: open drain)

someone else writes these

`(intruder(unsigned char intPresent))`

1: intruder present
(do 1--3)
0: no intruder
(reverse 1--3)

ex: alarm system monitor

implies level-based trigger

if sensors trip they output `0`, do:

1. trigger alarm
2. lock doors
3. execute notification routine

connected to `PD0`

(many sensors share same pin: open drain)

someone else writes these

`(intruder(unsigned char intPresent))`

1: intruder present
(do 1--3)
0: no intruder
(reverse 1--3)

ex: alarm system monitor

```
// 2. level triggering
PD[0x404] = 1;

// 3. low level
PD[0x408] = 0; //don't want both edges
PD[0x40C] = 0; //low level
```

```
             void GPIOPortD_Handler(void)
             {
                // no need to acknowledge interrupt

                intruder(1);

               while(PD0_DATA_B==0);

                intruder(0);
             }
```

$\nwarrow$ best be `volatile`

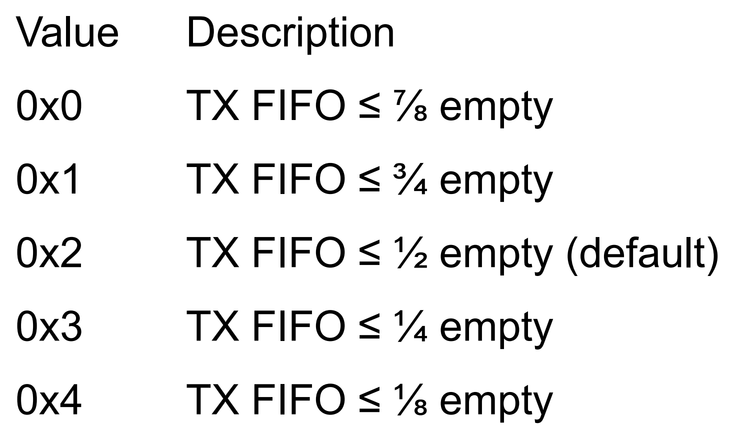
interrupts: UART

must take into account queue

(even when disabled, have one-depth queue)

`TXIFLSEL` field of `UARTIFLS`

0x32,p453



Value	Description
0x0	TX FIFO ≤ ⅞ empty
0x1	TX FIFO ≤ ¾ empty
0x2	TX FIFO ≤ ½ empty (default)
0x3	TX FIFO ≤ ¼ empty
0x4	TX FIFO ≤ ⅛ empty

interrupt when queue goes from:

0x0: 15->14 entries

0x1: 13->12 entries

0x2: 9->8 entries

0x3: 5->4 entries

0x4: 3->2 entries

note: opposite principle holds for RX

interrupts: UART

upon write to TX buffer or end of TX,
queue status checked

$\downarrow$

interrupt thrown if above conditions
not met

$\downarrow$

e.g. if `TXIFLSEL` = 0x2 and there
are fewer than five entries
(IRQ made)

note: if queue disabled interrupt
must be triggered manually

(end of TX doesn't trigger interrupt)

this could be simulator
bug

just enable queue
and don't worry about it

ex: transmit 'A'--'Z' continuously

```
; 6. level of fifo tx buffer that triggers interrupt
mov R0,#0x2 ;1/2 full transition: 9 to 8 entries
str R0,[R1,#0x34]

; 7. enable tx interrupt
mov R0,#0x20
str R0,[R1,#0x38]

; 8. enable tx and uart
mov R0,#0x101 ;0b0100000001
str R0,[R1,#0x30]

; load initial value into tx buffer to get things going
;  (if queue less than half full at tx buffer write,
;   ISR will be called)
mov R4,#0x41
str R4,[R1] ;TX first char

; 9. enable interrupts for UART0
;    UART0 is vector num 21, interrupt num five (bit)
ldr R1,=M3CP
mov R0,#0x20
str R0,[R1,#0x100]
```

store value to TX in R4

(treat R4 as global var)

will cause IRQ as queue entries < 9

ex: transmit ‘A’--’Z’ continuously

```
UART0_Handler
  ; acknowledge IRQ
  ldr R1,=UART0
  mov R0,#0x20 ;0x10=0b100000
  str R0,[R1,#0x44]

  ; figure out character to send
  cmp R4,#0x5A ;0x5A = 'Z', 0x5B =']'
  ITE EQ
  moveq R4,#0x41 ;reset to 'A'
  addne R4,#1 ;go to next character

  ; put character in TX buffer
  str R4,[R1,#0x0]

  ; we're done until byte finishes transmitting
  ;  (uart notices that queue is less than 1/2 full
  ;   and issues interrupt)
  bx lr
```

note: not terribly efficient

(IRQ made after every TX; better to just fill queue in ISR)

your response to my inefficient code:

ex: transmit 'A'--'Z' continuously

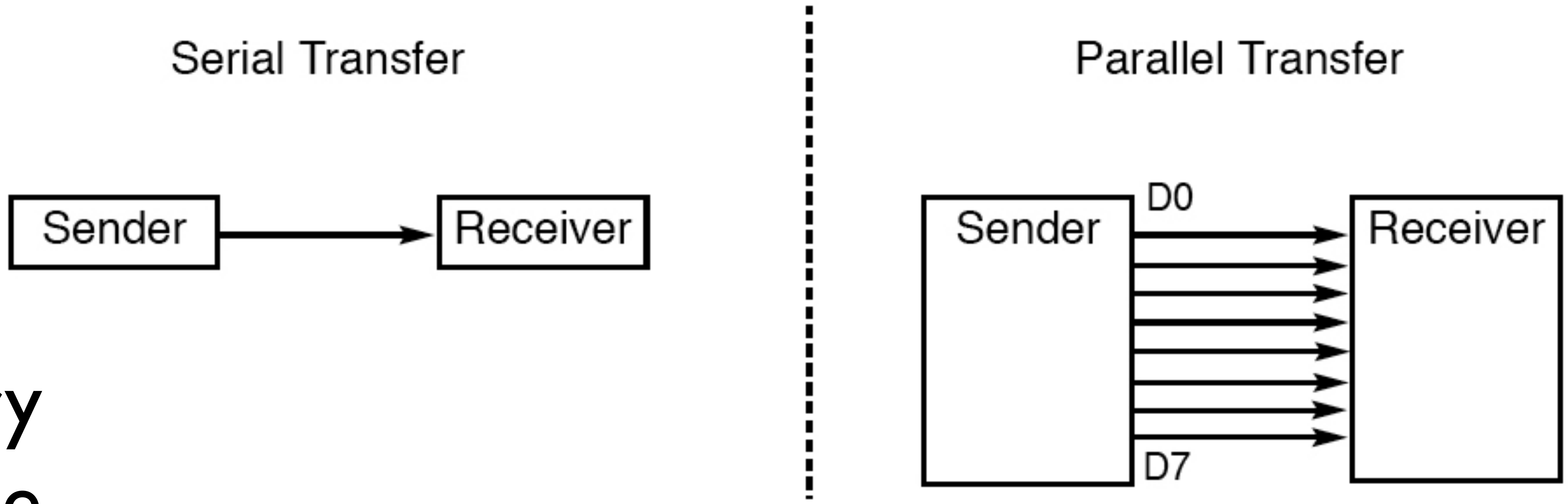
(fill queue on interrupt)

```
UART0_Handler
  ...

TX
  ; figure out character to send
  cmp R4,#0x5A ;0x5A = 'Z', 0x5B =']'
  ITE EQ
  moveq R4,#0x41 ;reset to 'A'
  addne R4,#1 ;go to next character

  ; put character in TX buffer
  str R4,[R1,#0x0]

  ; add to buffer until full
  ldr R0,[R1,#0x18]
  ands R0,#0x20 ;if result=0, z=1
  beq TX ;if z=1 branch

  ...
```

Serial Peripheral Interface (SPI)

ECE 3710

serial communication



let's try
this one

synchronous: sync'd clocks
asynchronous: clock in data stream/generated
at each end

you:

so excited to learn another serial protocol

I can tell the excitement is genuine

(more to come: I2C)

Serial Peripheral Interface (SPI)

or Synchronous Serial Interface (SSI)

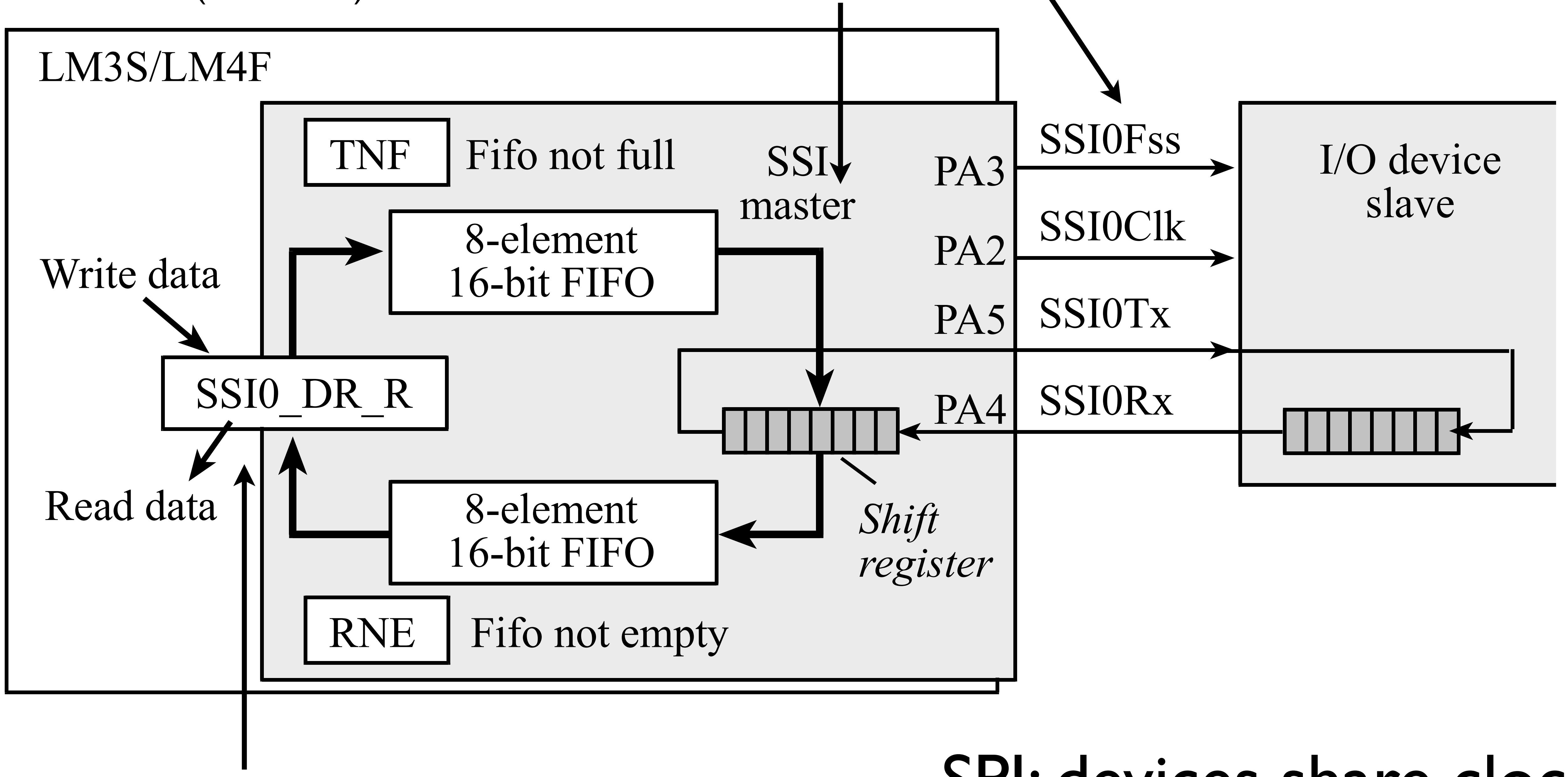
data:4--16 bits

(MSB first)

Fss: data ready to be TX

(when low)

initiates comm



like UART: shared TX/RX register

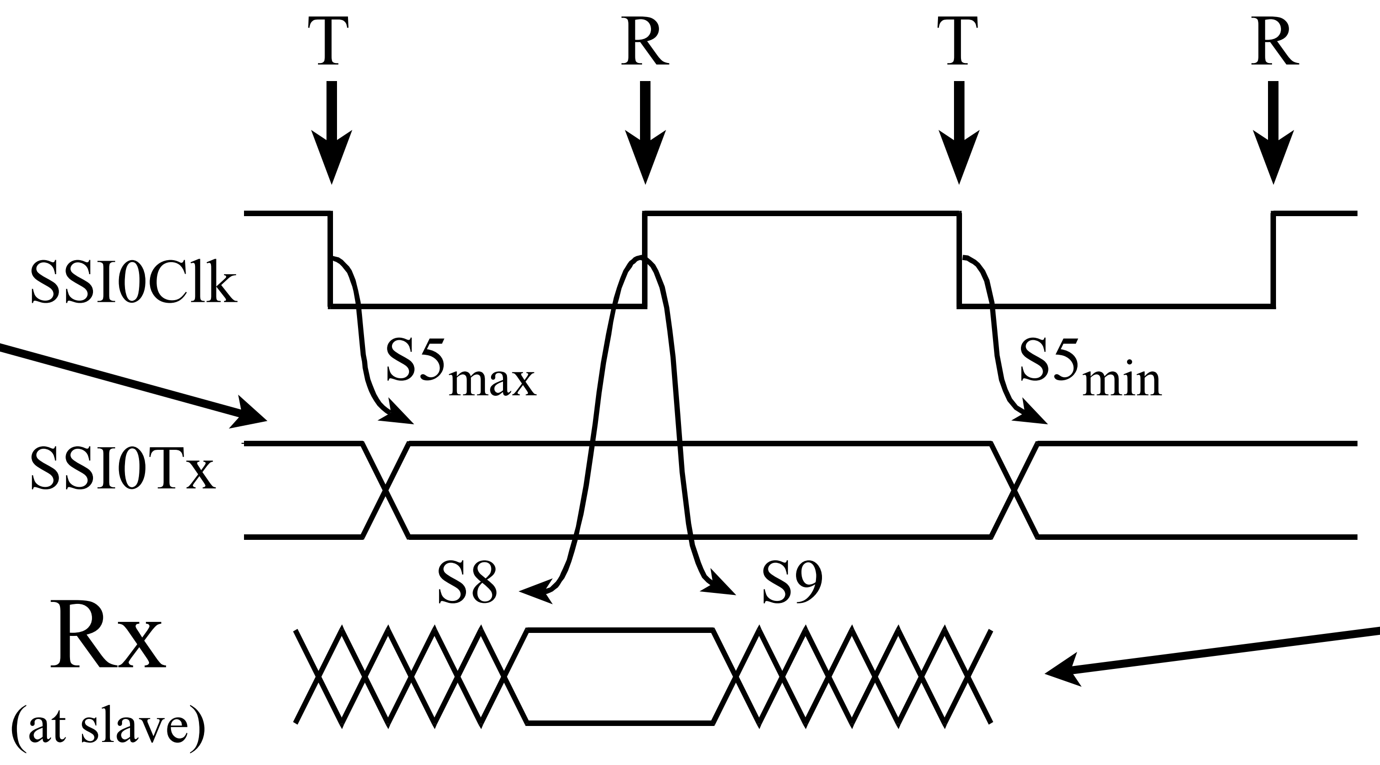
SPI: devices share clock

(master usually generates it)

SPI TX @master:

1. transmitter : change data on falling (rising)
2. receiver: latch data on rising (falling)

note: old data valid after change (0--1 clocks of *bus*)



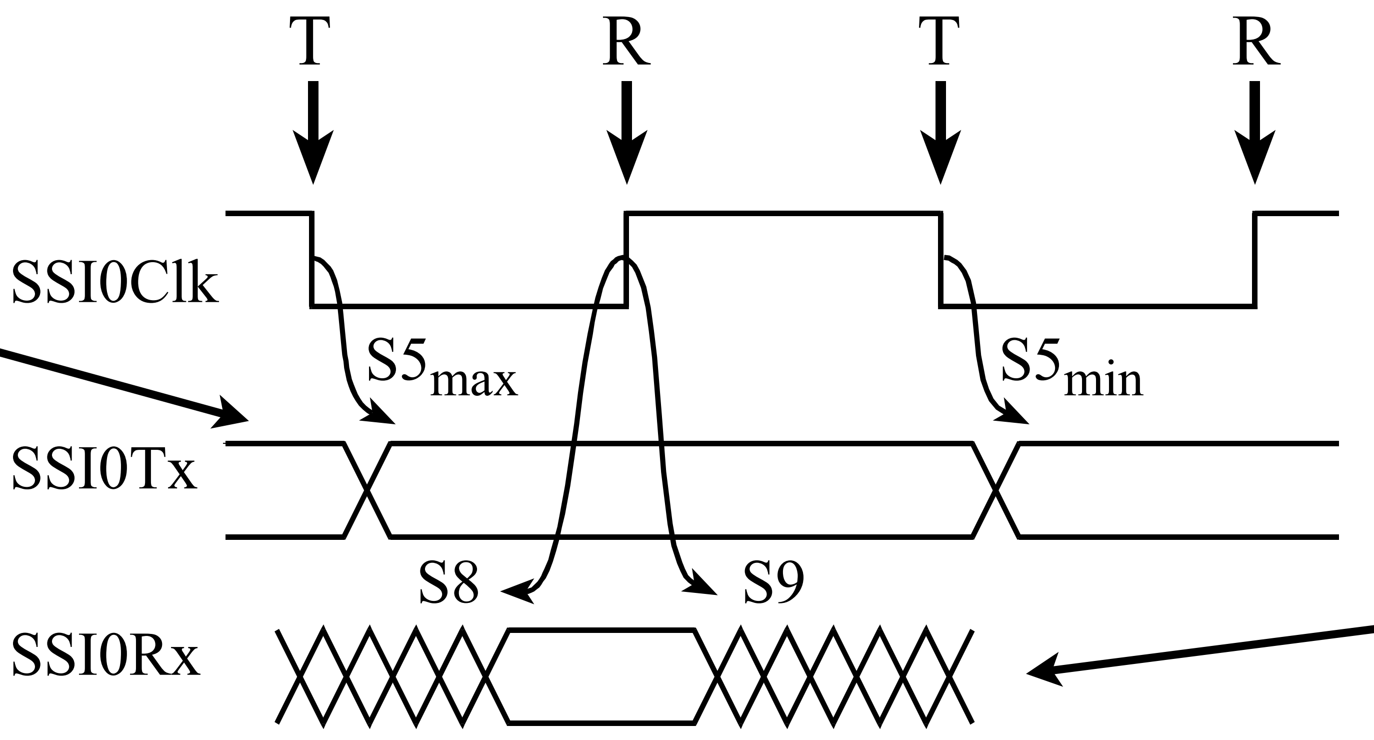
these are the same pin

SCLK < bus clock

SPI TX/RX @master:

pins

1. TX : change data on falling (rising)
2. RX: latch data on rising (falling)



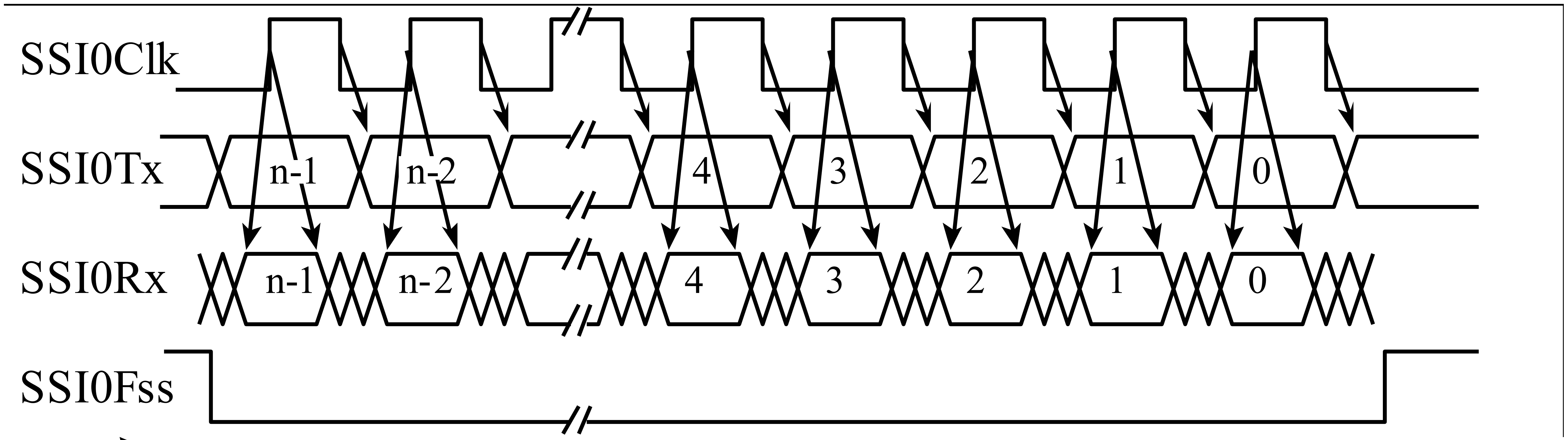
note: old data valid after change (0--1 clocks of *bus*)

these are different pins

data from slave

SPI full duplex:

1. TX on falling
2. RX on rising



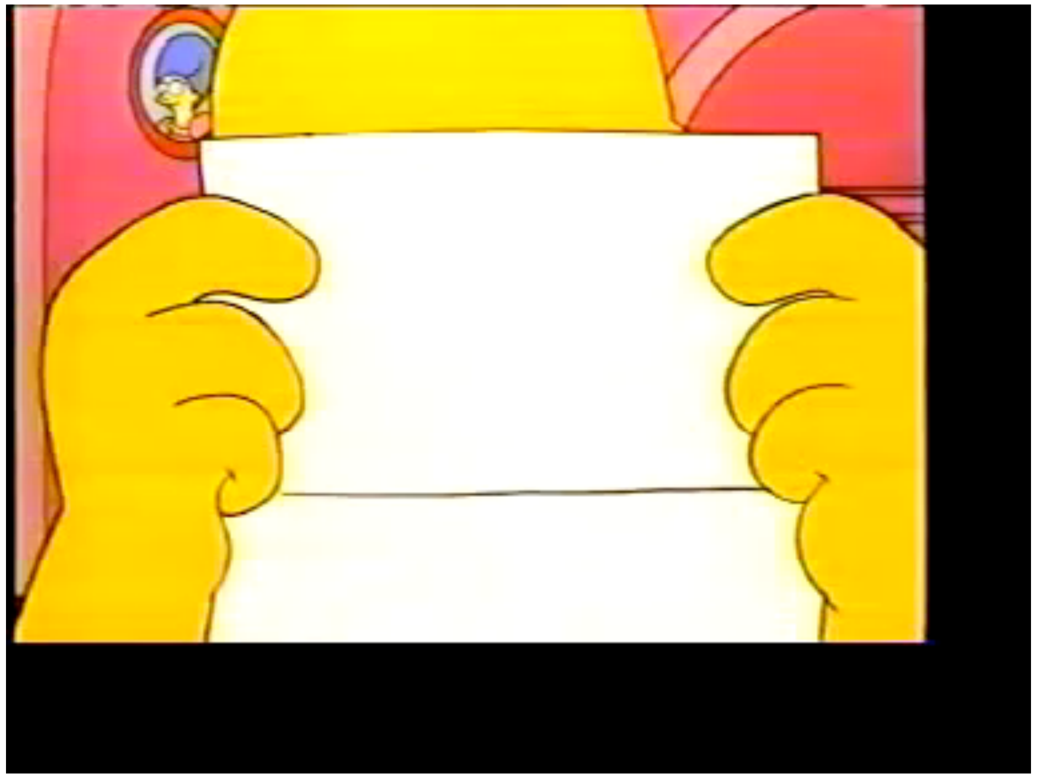
if low then we TX/RX → bring low: this is how we alert device we want to send data to it

note: use Freescale SPI format

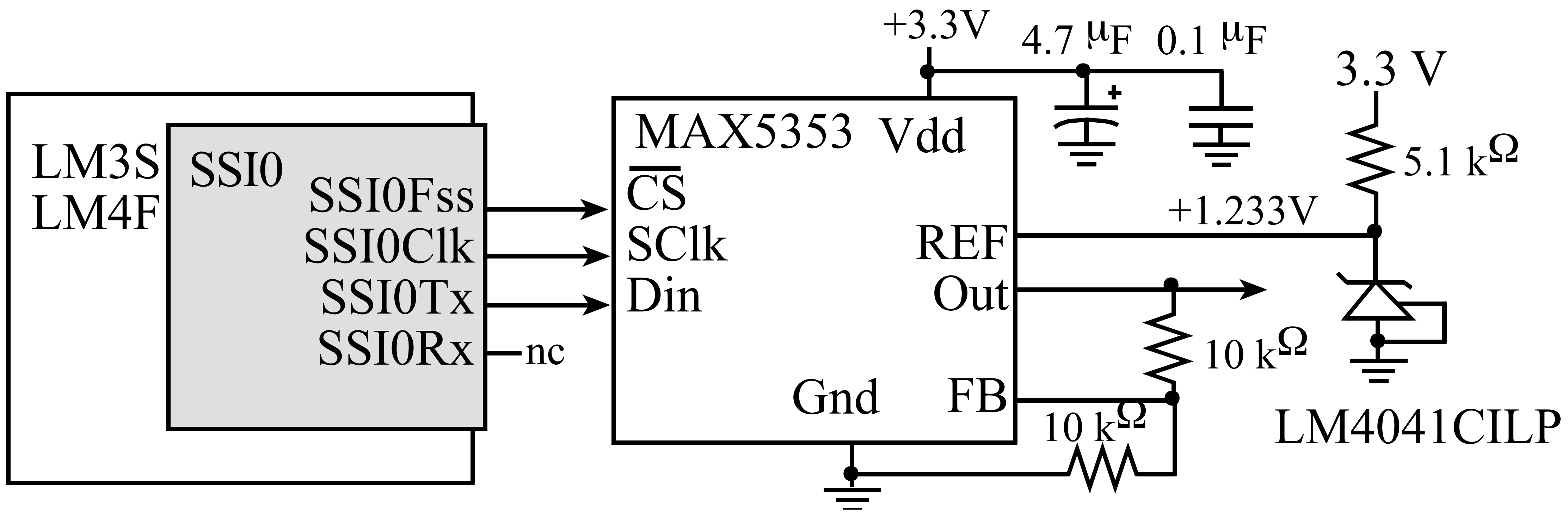
Q:

A: your touchscreen uses SPI

Student response?



ex: SPI w/DAC



Q:

1. what to send
2. how fast to send

ex: SPI w/DAC

DAC functions:

1. RX data, change output
2. RX data, don't change output
3. change output based on existing data
4. shutdown
5. NOP

C2	C1	C0	D11 : D0 MSB LSB	S0	Description
X	0	0	12 bits of data	0	Load input register; DAC register immediately updated.
X	0	1	12 bits of data	0	Load input register; DAC register unchanged.
X	1	0	XXXXXXXXXXXX	X	Update DAC register from input register.
1	1	1	XXXXXXXXXXXX	X	Shutdown
0	1	1	XXXXXXXXXXXX	X	No operation

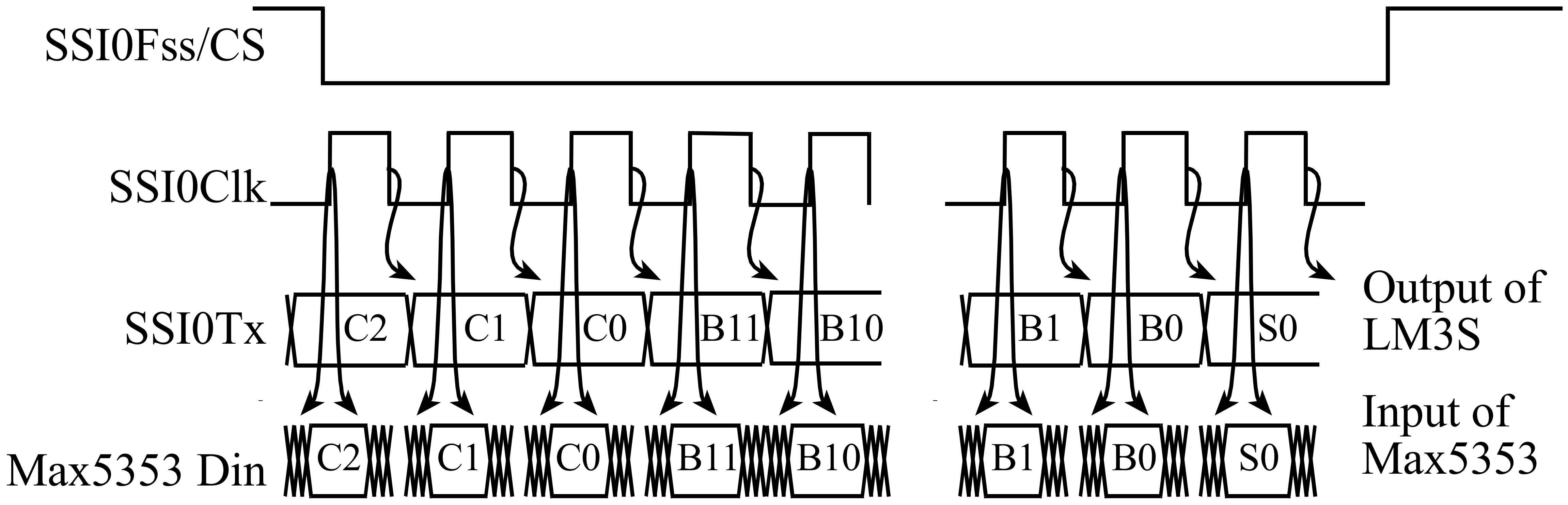
data = `[C2--C0 D11--D0 S0]` $\leftarrow$ 16-bits

ex: SPI w/DAC

period = 500 ns

speed:

1. if max SCLK= 2 MHz
   (can always make slower)
2. DAC requires low for at least 200 ns
3. max clock at DAC is 1/(2*200)=> 2.5 MHz
4. use SCLK =2 MHz



p483

SPI:

`SSICR1`
0x004,p488

1. select role
(master or slave)

2. set timing
(prescale and clock rate)

3. frame format/protocol
(data size andTI/FreeScale/Microwire)

`SSICPSR`
0x010,p493

`& SSICR0`
0x000,p486

`SSICR0`
0x000,p486

setting SCLK

multiple clock dividers needed

can be very slow

(SPI != high data rate)

$$SCLK = \frac{SysClk}{PRE \times (1 + CR)}$$

prescale

(even vaule [2,254])

clock rate

([0,255])

ex: 0--8191 to DAC w/immediate load

```
void SSI0Init()
{
     // 0a. activate clock for port a and ssi0
     RCGC1 = 0x10; //ssi0
     RCGC2 = 0x1; //port a

     // 0b. enable alt. function: PA[4:2]
     PA[0x420] = 0x1C;

     // 0c. direction for Fss
     PA[0x400] = 0x20;

     // 0c. digital enable: PA[5:2]
     PA[0x51C] = 0x3C;

     // 0d. disable ssi0 during config (overwrite all settings)
     SSI0[0x4] = 0;

     // 1. set role
     SSI0[0x4] = 0; //master

     // 2. set prescale and clock rate (SysClk = 12 MHz)
     // 2 MHz = 12 MHz/6 => PRE*(1+CR) = 6 => PRE = 2, CR = 2
     SSI0[0x10] = 0x2; //PRE
     SSI0[0x1] = 0x2; //CR

     // 3. frame format/protocol: 16-bit data, FreeScale, SPH=SPO=0
     SSI0[0x0] |= 0xF; // data size: 16-bits
     SSI0[0x0] &= 0xCF; //frame: freescale (bits four and five are zero)
     SSI0[0x0] &= 0x3F; //SPH=SPO=0 (bits six and zer are zero)

     // 4. enable ssi0 (use rmw cycle)
     SSI0[0x4] |= 0x2;
}
```

ex: 0--8191 to DAC w/immediate load

```
int main(void)
{
   unsigned short D;

   SSI0Init();

   //TX every combination of codes as quickly as possible
   D = 0;

   //alert slave to fact that we're sending data (Fss=0)
   PA[0x3FC] &= 0xDF; //bring pin five low

   while(1)
   {
      while(SSI0SR & 0x2) //get TNF bit; so long as TNF = 1, we can add
                          // data
      {
         SSI0DR = D<<1; //C[2:0] means immediate update; LSB must be
                       // zero for update
         D = (D+1)%8192; //2^13 = 8192 =0b10000000000000: this ensures D
                         // always less than or equal to 0b01111111111111
      }
   }
}
```