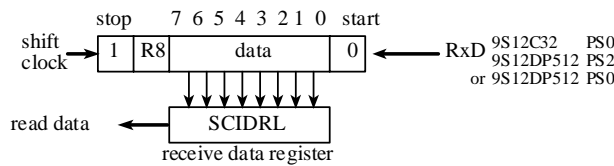
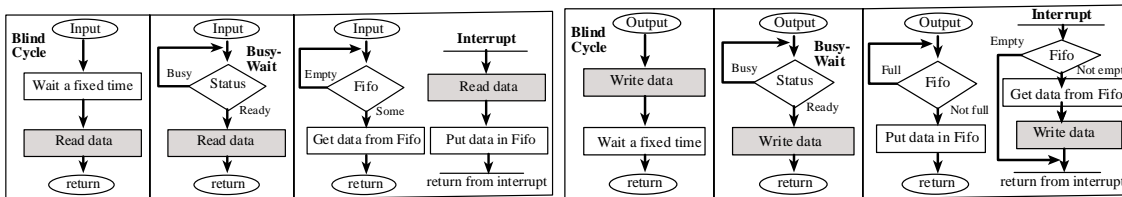


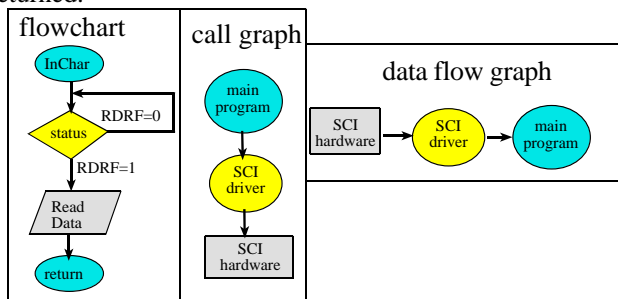
**Lecture 2 objectives**

- Review of EE319K
  - Busy-wait versus interrupt
  - SCI
  - Flow charts, Data flow, Call graphs
  - Real time systems
- Run **Lab1e** projects, real computer and simulation
- Understanding the listing, map, and S19 files
- Fixed-point (why, what, how)
  - Range, resolution, precision
  - Dropout, overflow
- Testing

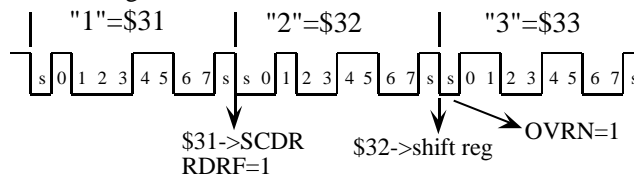


The following sequence of events occurs as one character is communicated:

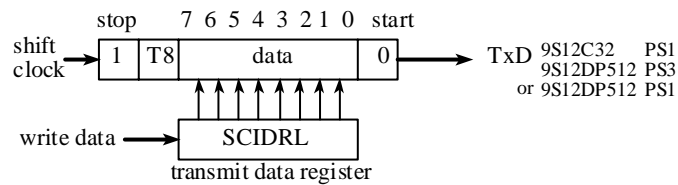
- the PC keyboard software encodes the typed key into ASCII,
- the PC terminal window transmits the 8-bit ASCII code out the COM port at 115200 bits/sec for the 9S12DP512,
- the 9S12 SCI port accepts the serial transmission,
- the **RDRF** (receive data register full) flag is set,
- the **SCI0\_InChar** function waits for **RDRF** to be set
- RDRF** is bit 5 of **SCI0SR1** on the 9S12DP512,
- the ASCII character is read from the receive serial data register **SCI0DRL** on the 9S12DP512,
- the ASCII code is returned.



If there is already data in the SCI0DRL when the shift register is finished, it will wait until the previous frame is read by the software, before it is transferred. An overrun occurs when there is one receive frame in the SCI0DRL, one receive frame in the receive shift register, and a third frame comes into RxD.

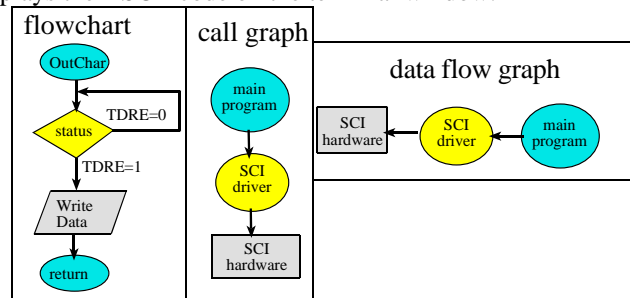


Three receive frames result in an overrun (OR) error.



To transmit a character from the 9S12 to the PC, your software calls `SCI0_OutChar`. The following sequence of events occurs:

- `SCI0_OutChar` waits for **TDRE** (transmit data register empty) to be set,
- TDRE** is bit 7 of `SCI0SR1` on the 9S12DP512,
- `SCI0_OutChar` writes the 8-bit ASCII code to the transmit data register `SCI0DRL` on the 9S12DP512,
- SCI hardware transmits the data to the PC COM serial port at 115200 bits/sec for the 9S12DP512,
- COM port on the PC accepts the serial transmission,
- HyperTerminal displays the ASCII code on the terminal window.



Run `SCI0_DP512` project  
show

- \* `.lst`, listing file used for fundamental understanding
- \* `.map`, map file used to get addresses for debugging

### Why Fixed-Point?

express values in our software that have noninteger values  
don't have floating point hardware

### When is it appropriate to use fixed point?

range of values is known  
range of values is small

### What is a fixed-point number?

**variable integer**, called **I**. **stored**  
**fixed constant**, called **Δ**. **not stored**  
the value of a fixed-point number  $\equiv I \cdot \Delta$

### How to design a fixed-point number?

integer can be signed or unsigned  
**precision** is the number of distinguishable values that can be represented.  
determined by the number of bits used to store the variable integer.  
8 bits or 16 bits.

**resolution** is the smallest difference in value that can be represented.  
equal to the fixed constant ( $\Delta$ ).  
has units.

decimal fixed-point number =  $I \cdot 10^m$  for some fixed integer  $m$

binary fixed-point number =  $I \cdot 2^n$  for some fixed integer  $n$

### Example Temperature Measurement System (Lab 6)

#### 9S12DP512 ADC

10-bit precision (1024 alternatives, about 3 decimal digits)

analog input range is 0 to +5 V,

resolution = range/precision about 5 mV

digital output varies 0 to 1023.

#### Measurement System

range is 10 to 40 C

10-bit precision

resolution about 0.03 C

#### Internal Data (contains all information)

$\Delta=0.01$  C e.g., **23.13** C stored as 2313

#### Output Display (honest, but some data is discarded)

$\Delta=0.1$  C e.g., **23.1** C

#### Output Display (a little bit dishonest)

$\Delta=0.01$  C e.g., **23.13** C

#### Output Display (honest)

$\Delta=0.01$  C e.g., **23.13 C ( $\pm 0.03$ C)**

$$V_{in} = 5 * N/1023 = 0.0048876 * N$$

and  $x = 10 + (30C * V_{in}/5V) = 10 + 6 (C/V) * V_{in}$

thus

$$x = 10 + 30 * N/1023 = 10 + 0.0293255 * N \quad \text{where } x \text{ is in C}$$

x temp in C	$V_{in}$ (V) Analog input	N ADC output	I ( $\Delta=0.01$ C)	I calculated
10.00	0.000	0	1000	1000
10.03	0.005	1	1003	1003
16.00	1.000	205	1600	1601
25.00	2.500	512	2500	2502
40.00	5.000	1023	4000	4001

Table 1.1. Performance data of a microcomputer-based temperature measurement.

#### overflow,

$$I = 1000 + (3000 * N) / 1023; \quad \text{dropout} \quad I = 1000 + 3000 * (N / 1023);$$

#### Two solutions to overflow

reduce the size of the integers

promote to higher precision

#### promotion,

$$I = 1000 + (3000 * (\text{unsigned long})N) / 1023;$$

#### approximation

$$I = 1000 + (44 * N) / 15;$$

#### rounding

```

I = 1000+(44*N+7)/15;
I = 1000+
  (3000*(unsigned long)N+512)/1023;

```

#### Assembly optimization

```

ldd  N
ldy  #3000
emul          ;32-bit Y:D is 3000*N
ldx  #1023
ediv          ;16-bit Y is (3000*N)/1023
leay 1000,y
sty  I

```

For example, consider the following digital filter calculation.

$$y = x - 0.0532672 \cdot x_1 + x_2 + 0.0506038 \cdot y_1 - 0.9025 \cdot y_2;$$

The fixed-point implementation of this digital filter is

$$y = x + x_2 + (-14 \cdot x_1 + 13 \cdot y_1 - 231 \cdot y_2) / 256;$$

A second example comes from the integral term of a PID controller, illustrating that fixed-point need not be either decimal or binary. In this control system, we wish to calculate

$$I(n) = I(n-1) + 1.57658 \cdot (X^* - X') \cdot 0.1$$

Fixed-point is used to approximate 0.157658 as 35/222 (0.157657658). So we can implement this control equation using integer calculations.

$$I(n) = I(n-1) + 35 \cdot (X^* - X') / 222$$

How do you find two integers K,J such that K/J is 0.157658

J	K	K/J	error
222	35	0.157658	3.42E-07
444	70	0.157658	3.42E-07
666	105	0.157658	3.42E-07
888	140	0.157658	3.42E-07
907	143	0.157663	4.62E-06
869	137	0.157652	5.53E-06
685	108	0.157664	6.23E-06
647	102	0.157651	7.3E-06
463	73	0.157667	9.39E-06
926	146	0.157667	9.39E-06
425	67	0.157647	1.09E-05
850	134	0.157647	1.09E-05
704	111	0.15767	1.25E-05
945	149	0.157672	1.4E-05

[Run Lab1 project](#)

[show](#)

[modularity](#)  
[testing or software verification](#)  
[simulator](#)