Working in teams,  
“Everything you think is blindingly obvious is wrong.”  
Tim Fields, UT EE grad, Lead Designer for Brute Force at Microsoft  

A **device driver** is a collection of software functions that allow higher level software to utilize an I/O device.

Collection of public methods (subroutines)  
SCI_Init  
SCI_InChar  
SCI_OutChar  

Collection of private objects (subroutines, globals, I/O ports)  
SCICR2  
SCIBD  
SCISR1  
SCIDRL  

**complexity abstraction**  
divide a complex problem into simple subcomponents  

**functional abstraction**  
divide a problem into modules  
grouped by function  

### 5.3.4. 6812 SCI Details

<table>
<thead>
<tr>
<th>Addr</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00C8</td>
<td>BTST</td>
<td>BSPL</td>
<td>BRLD</td>
<td>SBR12</td>
<td>SBR11</td>
<td>SBR10</td>
<td>SBR9</td>
<td>SBR8</td>
<td>SCIBD</td>
</tr>
<tr>
<td>$00C9</td>
<td>SBR7</td>
<td>SBR6</td>
<td>SBR5</td>
<td>SBR4</td>
<td>SBR3</td>
<td>SBR2</td>
<td>SBR1</td>
<td>SBR0</td>
<td></td>
</tr>
<tr>
<td>$00CB</td>
<td>TIE</td>
<td>TCIE</td>
<td>RIE</td>
<td>ILIE</td>
<td>TE</td>
<td>RE</td>
<td>RWU</td>
<td>SBK</td>
<td>SCICR2</td>
</tr>
<tr>
<td>$00CC</td>
<td>TDRE</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>PF</td>
<td>SCISR1</td>
</tr>
<tr>
<td>$00CF</td>
<td>R7T7</td>
<td>R6T6</td>
<td>R5T5</td>
<td>R4T4</td>
<td>R3T3</td>
<td>R2T2</td>
<td>R1T1</td>
<td>R0T0</td>
<td>SCIDRL</td>
</tr>
</tbody>
</table>

Table 5.4. 9S12C32 SCI ports.
SCIBD

SCI Baud Rate = \( \frac{MCLK}{(16 \cdot BR)} \)

on 9S12C32 \( MCLK = 24\text{MHz} \) (with PLL)
\( = 4\text{ MHz} \) (otherwise)

TE is the Transmitter Enable bit, and
RE is the Receiver Enable bit.

TDRE is the Transmit Data Register Empty flag.
set by the SCI hardware if transmit data register empty
if set, the software write next output to SCIDRL
cleared by two-step software sequence
first reading SCISR1 with TDRE set
then SCIDRL write

RDRF is the Receive Data Register Full flag.
set by hardware if a received character is ready to be read
if set, the software read next into from SCIDRL
cleared by two-step software sequence
first reading SCISR1 with RDRF set
then SCIDRL read

SCIDRL register contains transmit and receive data
these two registers exist at the same I/O port address
Reads access the read-only receive data register (RDR)
Writes access the write-only transmit data register (TDR)
5.3.5. SCI I/O Programming.

*Busy-waiting, gadfly, or polling* are three equivalent names software continuously checks the hardware status waiting for it to be ready.

---

**Program 5.7. Assembly functions that implement serial I/O.**

---

Jonathan W. Valvano
**************typewriter***************

problem description

input from keyboard, echo all characters on screen

data flow graph

 SCI input
 SCI output

 serial port
driver

main

serial port
driver

SCI hardware

flow chart

main

SCI_Init

data <- SCI_InChar

SCI_OutChar <- data
1) run **typewriter**
2) see SCI device driver

**************************square**************************

**Problem description**
- generate a squarewave

**Data flow graph**

```
+----------------+              +----------------+              +----------------+
| timer hardware |              | main            |              | PORTT           |
+----------------+              +----------------+              +----------------+
```

**Call graph**

```
+----------------+   +----------------+   +----------------+
| main           |   | timer driver    |   | timer hardware  |
+----------------+   +----------------+   +----------------+
```

**Flow chart**

```
+----------------+   +----------------+   +----------------+
| Timer_Wait     |   | EndT=TCNT+input |   | add TCNT        |
+----------------+   +----------------+   +----------------+
```

**Timer_Wait**
- *Time delay function*
- *Input: RegD is the time to wait (in 250ns cycles)*
- *Outputs: none*
- *error: input must < 32767*

```
Wloop cpd TCNT EndT<Tcnt? bpl Wloop rts
```

1) run **square**
2) see Timer device driver, show **TCNT**
3) create **ScanPoint**
4) change **PERIOD**

```plaintext
* ************square.RTF**********************
* squarewave Example
* PT0 output connected to LED

This example accompanies either of the following two books
* "Embedded Microcomputer Systems: Real Time Interfacing", Brooks-Cole,
copyright (c) 2000,
* "Introduction to Embedded Microcomputer Systems: 
  Motorola 6811 and 6812 Simulation", Brooks-Cole, copyright (c) 2002
* TExaS Copyright 2004 by Jonathan W. Valvano, valvano@mail.utexas.edu
* You may use, edit, run or distribute this file
* as long as the above copyright notices remain
* ***************************************************

org $4000

**********timer routines**********************
DDRT    equ $0242  ; Port T Data Direction Register
PTT     equ $0240  ; Port T I/O Register
TCNT    equ $0044  ; Timer Count Register
TSCR1   equ $0046  ; Timer System Control Register1

****Timer_Init**************
* turn on TCNT at 4MHz
* inputs: none
* outputs: none
* errors: assumes 4MHz E clock
Timer_Init
    movb #$80,TSCR1  ;enable TCNT
    rts

****Timer_Wait***************
* Wait a fixed amount of time
* inputs: RegD time to wait in 250ns units
* outputs: none
* errors: assumes 4MHz E clock
Timer_Wait
    addd TCNT   ; RegD is TCNT at end of wait
tloop   cpd  TCNT
    bpl  tloop
    rts

PERIOD  equ 1000       ; 1000us period squarewave
main     lds  #$4000   ; initialize stack
    bsr Timer_Init  ; enable TCNT
    bset DDRT, #$01  ; PT0 output to LED
loop     ldab PTT
    eora  #$01
    stab PTT        ; toggle LED
    ldd   #2*PERIOD
    bsr Timer_Wait
    bra  loop
    org $FFFE
    fdb  main
```
ATDCTL2 set bit 7 to enable ADC
ATDCTL5 write channel number to start ADC
  • channel number 0 to 7
ATDSTAT bit 7 SCF
  • cleared by write to ATDCTL5
  • set when ADC finished
ATDDRO first 10-bit ADC result
  • precision 10-bit, 1024 alternatives
  • range 0 to +5V
  • resolution (5-0)/1024 = 5 mV

Digital Output = 1024*\(V_{\text{in}}/5\)

<table>
<thead>
<tr>
<th>Analog Input (V)</th>
<th>Digital Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>%000000000000 $000 0</td>
</tr>
<tr>
<td>0.005</td>
<td>%00000000001 $001 1</td>
</tr>
<tr>
<td>2.500</td>
<td>%1000000000 $200 512</td>
</tr>
<tr>
<td>3.750</td>
<td>%1100000000 $300 192</td>
</tr>
<tr>
<td>5.000</td>
<td>%1111111111 $3FF 1023</td>
</tr>
</tbody>
</table>

Table 5.11. **Straight binary** format used.
1) run **tut3**
2) see ADC device driver
3) show **ADC_Init**
   simply turns it on
4) show **ADC_In**
   write channel number to **ATDCTL5**
   wait for SCF flag in **ATDSTAT**
   read 10-bit result from **ATDDR0**
5) show **SCI_OutDec**
Lab 5.2 *(same data flow as tut3)*

```
main
  \rightarrow A/D driver
  \rightarrow serial port driver
  \rightarrow timer driver

ADC hardware
SCI hardware
timer hardware
```

![Graph and diagram showing data flow](image-url)