“When working on a software development team, everyone needs to be constantly reminded what is blindingly obvious.”
Tim Fields, UT EE grad, Lead Designer for Brute Force at Microsoft

5. I/O Programming

Chapter 5 objectives are to:
• Describe I/O ports available on the 6812,
• Explain the fundamentals of I/O programming,
• Give a simple examples of I/O device drivers.

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

5.1. Basic concept of an I/O port address and direction register
Figure 5.4. A bidirectional port can be configured as a read-only input port or a readable output port.

5.2. Parallel Port I/O Programming

Figure 5.5. The input/output direction of a bidirectional port is specified by its direction register.

5.3. Serial Communications Interface, SCI

The total number of bits transmitted per second is called the **baud rate**.

$M$, selects 8-bit ($M=0$) or 9-bit ($M=1$) data frames.

A **frame** is the smallest complete unit of serial transmission.

The information rate, or **bandwidth**, is defined as the amount of data or usual information transmitted per second.

Figure 5.6. A serial data frame with $M=0$. 
5.3.1. Transmitting in asynchronous mode

The software writes to SCIDRL, then
8 bits of data are moved to the shift register
start and stop bits are added
shifts in 10 bits of data one at a time on TxD line
shift one bit per bit time (=1/baudRate)

5.3.2. Receiving in asynchronous mode

Figure 5.8. Data register shift registers implement the receive serial interface.

Jonathan W. Valvano
The receiver waits for the 1 to 0 edge signifying a start bit, then shifts in 10 bits of data one at a time from RxD line shift one bit per bit time (=1/baudRate) start and stop bits are removed checked for noise and framing errors 8 bits of data are loaded into the SCIDRL

If there is already data in the SCDR 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 SCDR, one receive frame in the receive shift register, and a third frame comes into RxD.

![Figure 5.9. Three receive data frames result in an overrun (OR) error.](image)

### 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>SCICR2</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>SCISR1</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>SCIDRL</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></td>
</tr>
</tbody>
</table>

Table 5.4. 9S12C32 SCI ports.
SCIBD

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

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

$\text{TE}$ is the Transmitter Enable bit, and
$\text{RE}$ is the Receiver Enable bit.

$\text{TDRE}$ 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

$\text{RDRF}$ 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

$\text{SCIDRL}$ 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.
* Initialize 9S12C32 SCI at 250000 bps
* Inputs: none
* Outputs: none
* Errors: none

SCI_Init
  movb #$0c, SCICR2  enable SCI TE=RE=1
  movw #1, SCIBD     250000 bps

*baud rate (bps) = 250000/BR

rts

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

![Diagram of SCI_Init and SCI_InChar](attachment://diagram.png)

**********SCI_InChar******************
* Input one character from SCI
* Inputs: none
* Outputs: RegA is ASCII character
* Registers modified: CCR

RDRF     equ $20

SCI_InChar
  brclr SCISR1,#RDRF,SCI_InChar
  * RDRF=1 when a new key is available
  ldaa SCIDRL    ASCII character
  rts

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

```assembly
***SCI_OutChar*******************
* Output one character to SCI
* Inputs: RegA is ASCII char
* Outputs: none
* Registers modified: CCR
TDRE equ $80
SCI_OutChar
    bclr SCISR1,#TDRE,SCI_OutChar
    * TDRE=1 when ready for more output
    staa SCIDRL  Start Output
    rts
    * * * ** End of SCI_OutChar * * * * *
```

Program 5.8. C functions that implement serial I/O.

```c
// 6812 initialize SCI
void SCI_Init(void)
{
    SCIBD = 0;;  // 250000 bits/sec
    SCICR2 = 0x0C;;  // enable
}
#define RDRF 0x20
// Wait for new input,
// then return ASCII code
char SCI_InChar(void)
{
    while((SCISR1&RDRF) == 0){};
    return(SCIDRL);
}
#define TDRE 0x80
// Wait for buffer to be empty,
// then output
void SCI_OutChar(char data)
{
    while((SCISR1&TDRE) == 0){};
    SCIDRL = data;
}
```

Program 5.8. C functions that implement serial I/O.