EECE 218
Microcontrollers

Serial Communication Interface
Serial Communications

- Principle: transfer a word one bit at a time
- Methods:
  - Simplex: [S] → [R]
  - Duplex: [D1] ↔ [D2]
  - Half duplex: Like duplex, using a single wire
- If data is ‘self-timed’ (no extra clock line):
  - Asynchronous communication (SCI)
    - Lower data rate, simpler connection, complex electronics
- If extra timing signal is used:
  - Synchronous communication (SPI)
    - Higher data rate, more wires, simpler electronics
Serial Communications

- SCI: Asynchronous communication
- Electrical signals:
  - 20mA current loop (industrial instruments, TTY)
  - RS-232: standard from 1969 ('PC serial port')
    - We focus here on data communication signals only (3-wire), the standard covers other control signals as well (5-wire, modem).

Voltage levels:
- $H = +3\ldots+25V$: Logic ‘0’
- $L = -3\ldots-25V$: Logic ‘1’

TTL circuits are not usable $\rightarrow$ need ‘level shifter’ (MAX232)
EIA-232 Standard

- Typical signal:

- Message elements:
  - 1 start bit
  - 5-8 data bits
  - (Optional) parity bit (odd/even/mark/space parity)
  - 1-2 stop bits

- Message is self-timed – no extra clock
Serial Communication Interfaces

- UART: Universal Asynchronous Receiver/Transmitter: single chip, full duplex device

- Typical transmission rates:
  - Bits/sec: Baud rate
  - 50 bd (TTY)... 19.2K (typical max RS-232)

- Communication errors:
  - Timing error (different rates on R/T)
  - Framing error (Start/Stop: Frame)
  - Overrun error (new word overrides old)
An HCS12 device may have one or two serial communication interface/s. These two SCI interfaces are referred to as SCI0 and SCI1.

Use the data format of one start, eight or nine data bits, and one stop bit. The collection of the start bit, eight or nine data bits, and the stop bit is called a frame.

The SCI function supports parity checking. This option requires the use of 9-bit data format.

One SCI channel uses two signal pins from Port S. The SCI0 shares the use of PS0 (RxD0) and PS1 (TxD0), whereas SCI1 shares the use of PS2 (RxD1) and PS3 (TxD1).

The SCI has the capability to send break to attract the attention of the other party of communications. A break is defined as the transmission or reception of logic 0 for a frame or longer time.

The SCI supports hardware parity for transmission and reception.

The SCI supports idling line and address mark wakeup, which is useful in multi-drop environment to reduce the software overhead.
HCS12 SCI

Figure 9.8 HCS12 SCI block diagram
Baud Rate Generation

- The HCS12 SCI module uses a 13-bit counter to generate this clock signal. This circuit is called **baud rate generator**.
- The baud rate generator divides down the E clock to derive the clock signal for reception and transmission.
- The user writes an appropriate value into the SCIxBDH and SCIxBDL (x = 0 or 1) register pair to set the baud rate.

![Figure 9.9 SCI baud rate control register](image-url)
Baud Rate Generation

- The value to be written into the baud rate generator register is the rounding of the following expression:

\[ SBR = \frac{f_E}{16 \div \text{baud rate}} \]

Table 9.2 Baud rate generation

<table>
<thead>
<tr>
<th>Desired SCI baud rate</th>
<th>Baud rate divisor for ( f_E = 16 \text{ MHz} )</th>
<th>Baud rate divisor for ( f_E = 24 \text{ MHz} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>3333</td>
<td>5000</td>
</tr>
<tr>
<td>600</td>
<td>1667</td>
<td>2500</td>
</tr>
<tr>
<td>1200</td>
<td>833</td>
<td>1250</td>
</tr>
<tr>
<td>2400</td>
<td>417</td>
<td>625</td>
</tr>
<tr>
<td>4800</td>
<td>208</td>
<td>313</td>
</tr>
<tr>
<td>9600</td>
<td>104</td>
<td>156</td>
</tr>
<tr>
<td>14,400</td>
<td>69</td>
<td>104</td>
</tr>
<tr>
<td>19,200</td>
<td>52</td>
<td>78</td>
</tr>
<tr>
<td>38,400</td>
<td>26</td>
<td>39</td>
</tr>
</tbody>
</table>
The SCI Control Registers (1 of 2)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOPS</td>
<td>SCISWAI</td>
<td>RSRC</td>
<td>M</td>
<td>WAKE</td>
<td>ILT</td>
<td>PE</td>
<td>PT</td>
</tr>
</tbody>
</table>

reset: 0 0 0 0 0 0 0 0

LOOPS: Loop select bit
0 = loop operation disabled
1 = loop operation enabled

SCISWAI: SCI stop in wait mode
0 = SCI enabled in wait mode.
1 = SCI disabled in wait mode.

RSRC: Receiver source bit
When LOOPS = 1, the RSRC bit determines the source for the receiver shift register
0 = receiver input connected to the transmitter internally (not TxD pin).
1 = receiver input connected externally to the transmitted (TxD pin)

M: Data format mode bit
0 = one start bit, eight data bits, one stop bit
1 = one start bit, nine data bits, one stop bit

WAKE: Wakeup condition bit
0 = idle line wakeup
1 = address mark wakeup (last data bit set)

ILT: Idle line type bit
0 = idle character bit count begins after start bit
1 = idle character bit count begins after the stop bit

PE: parity enable bit
0 = parity disabled
1 = parity enabled

PT: parity type bit (for both transmit and receive)
0 = even parity selected
1 = odd parity selected

Figure 9.10 SCI control register 1 (SC0CR1/SC1CR1)
The SCI Control Registers (2 of 2)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value after reset</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TIE: Transmit interrupt enable bit</td>
<td>0</td>
<td>0 = TDRE interrupt disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = TDRE interrupt enabled</td>
</tr>
<tr>
<td>1</td>
<td>TCIE: Transmit complete interrupt enable bit</td>
<td>0</td>
<td>0 = TC interrupt disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = TC interrupt enabled</td>
</tr>
<tr>
<td>2</td>
<td>RIE: Receiver full interrupt enable bit</td>
<td>0</td>
<td>0 = RDRF and OR interrupts disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = RDRF and OR interrupt enabled</td>
</tr>
<tr>
<td>3</td>
<td>ILIE: Idle line interrupt enable bit</td>
<td>0</td>
<td>0 = IDLE interrupt disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = IDLE interrupt enabled</td>
</tr>
<tr>
<td>4</td>
<td>TE: Transmitter enable bit</td>
<td>0</td>
<td>0 = transmitter disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = transmitter enabled</td>
</tr>
<tr>
<td>5</td>
<td>RE: Receiver enable</td>
<td>0</td>
<td>0 = receiver disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = receiver enabled</td>
</tr>
<tr>
<td>6</td>
<td>RWU: Receiver wakeup bit</td>
<td>0</td>
<td>0 = normal SCI receiver</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = enables the wakeup function and inhibits further receiver interrupts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Normally, hardware wakes up the receiver by automatically clearing this bit.</td>
</tr>
<tr>
<td>7</td>
<td>SBK: Send break bit</td>
<td>0</td>
<td>0 = no break characters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1 = generate a break code, at least 10 or 11 contiguous 0s. As long as SBK remains set, the transmitter sends 0s.</td>
</tr>
</tbody>
</table>
SCI Status Registers (1 of 2)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDRE</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>PF</td>
</tr>
</tbody>
</table>

reset: 1 1 0 0 0 0 0 0

- **TDRE**: Transmit data register empty flag
  - 0 = No byte was transferred to the transmit shift register.
  - 1 = Transmit data register is empty.

- **TC**: Transmit complete flag
  - 0 = Transmission in progress
  - 1 = No transmission in progress

- **RDRF**: Receiver data register full flag
  - 0 = SCIxDR empty
  - 1 = SCIxDR full

- **IDLE**: Idle line detected flag
  - 0 = RxD line active
  - 1 = RxD line becomes idle

- **OR**: Overrun error flag
  - 0 = no overrun
  - 1 = overrun detected

- **NF**: Noise error flag
  - Set during the same cycle as the RDRF bit but not set in the case of an overrun (OR)
  - 0 = No noise
  - 1 = Noise

- **FE**: Framing error flag
  - Set when a 0 is detected where a stop bit was expected.
  - 0 = No framing error
  - 1 = Framing error

- **PF**: Parity error flag
  - 0 = parity correct
  - 1 = incorrect parity detected

Figure 9.12 SCI status register 1 (SCI0SR1/SCI1SR1)
## SCI Status Registers (2 of 2)

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

reset: 0 0 0 0 0 0 0 0

**BK13**: Break transmit character length
- 0 = Break character is 10- or 11-bit long
- 1 = Break character is 13- or 14-bit long

**TXDIR**: transmit pin data direction in single-wire mode
- 0 = TxD pin to be used as an input in single-wire mode
- 1 = TxD pin to be used as an output in single-wire mode

**RAF**: receiver active flag
- RAF is set when the receiver detects a logic 0 during the RT1 time period of the start bit search. RAF is cleared when the receiver detects an idle character.
- 0 = no reception in progress
- 1 = reception in progress

Figure 9.14 SCI status register 2 (SCI0SR2/SCI1SR2)
Character Transmission

Figure 9.12 SCI transmitter block diagram
Send Break/Idle Characters

- **A break character** is represented by eight or nine logic 0 data bits depending on the character data length.
- Whenever one party in the data communications discovers an error, it can send break characters to discontinue the communication and start over again.
- To send break characters, the user sets the SBK bit in the SCIxCR1 register to 1.
- As long as the SBK bit is 1, the transmitter logic continuously sending out the break character.

- **An idle character** contains all 1s and has no start, stop, or parity bit.
- Depending on the character data length, an idle character can be eight or nine 1s.
- If the TE bit in the SCIxCR2 register is cleared during a transmission, the TxD signal becomes idle after the completion of the transmission in progress.
Figure 9.15 SCI receiver block diagram
Single-Wire Operation

- In this operation, the RxD pin is disconnected from the SCI module.
- The SCI module uses the TxD pin for both receiving and transmitting as illustrated below.
- Single-wire operation is enabled by setting the LOOPS and the RSRC bits in the SCIxCR1 register.
- Setting the LOOPS bit disables the path from the RxD pin to the receiver. Setting the RSRC bit connects the receiver input to the output of the TxD pin driver.
- Both transmitter and receiver must be enabled.
- The TXDIR bit determines whether the TxD pin is going to be used as an input (TXDIR = 0) or output (TXDIR = 1) in this mode of operation.

![Single-Wire Operation Diagram](image)

Figure 9.16 Single-wire operation
Flow Control of UART in Asynchronous Mode

- The SCI module will transmit data as fast as the baud rate allows.
- In some circumstances, the software may not be able to read data as fast as the data is received. There is a need for the MCU to tell the transmitting device to suspend transmission of data temporarily. Similarly, the HCS12 may need to be told to suspend transmission temporarily. This is done by flow control.
- There are two common methods of flow control: XON/XOFF and hardware.
- XON/XOFF is implemented completely in software, but requires a full-duplex communication.
- When incoming data needs to be suspended, an XOFF byte is transmitted back to the other device that is transmitting.
- To start the other device transmitting again, an XON character is transmitted.
- The XON and XOFF characters have the ASCII code of 0x11 and 0x13, respectively.
- Hardware flow control requires the use of extra signals. Generally, an input pin of the transmitter is controlled by the receiver.
- Before transmitting any character, the transmitter needs to test the flow control input pin.
Examples

- Write an instruction sequence to configure the SCI0 to operate with the following parameters:
  » 9600 baud (E clock is 24 MHz)
  » One start bit, 8 data bits, one stop bit
  » No interrupt
  » Address mark wakeup
  » Disable wakeup initially
  » Long idle line mode
  » Enable transmit and receive
  » No loop back
  » Disable parity checking

- Solution: The following instruction sequence will configure the SCI0 properly:

  ```
  movb #$00,SC0BDH ; set up baud rate
  movb #156,SC0BDL ;
  movb #$4C,SC0CR1  ; select 8 data bits, address mark wakeup
  movb #$0C,SC0CR2  ; enable transmitter and receiver
  ```
Interfacing SCI with EIA-232-E

- The SCI uses 0 V and 5 V to represent 0 and 1.
- The EIA-232 signal Tx cannot be driven by the SCI TxD signal without translation.
- The EIA-232 signal Rx cannot drive the SCI RxD signal without translation.
- Voltage level translation is required for the SCI signals to drive and be driven by the EIA-232 signals.
- Examples of EIA-232 driver chips include:
  » LT1080/1081 from Linear technology
  » ST232 from SGS Thompson
  » ICL232 from Intersil
  » MAX232 from MAXIM
  » DS14C232 from National Semiconductor
  » These chips are pin-compatible.
- The DS14C232 from National Semiconductor will be used in the following illustration.
Figure 9.18 Pin assignments and connections of the DS14C232
Examples

- Write a subroutine to send a break to the communication port controlled by the SCI0 interface. The duration of the break is approximately 24,000 E clock cycles, or 1 ms at 24 MHz.

- Solution: A break character is represented by ten or eleven consecutive zeros and can be sent out by setting the bit 0 of the SCI0CR2 register.

```
#include "c:\miniide\hcs12.inc"

sendbrk bset SCI0CR2,SBK ; turn on send break
ldy #1
jsr delayby1ms
bclr SCI0CR2,SBK ; turn off send break
rts

#include “c:\miniide\delay.asm”
```
Examples

- Write a subroutine to output the character in accumulator A to the SCI0 channel using the polling method.
- Solution: The subroutine will wait until the bit 7 of SCI0SR1 register is set before sending out the character in accumulator A.

```c
#include "c:\miniide\hcs12.inc"
putcSCI0 brclr SCI0SR1,TDRE,* ; wait for TDRE to be set
staa SCI0DRL ; output the character
rts
```
Write a subroutine to read a character from SCI0 using the polling method. Return the character in accumulator A.

Solution:

```assembly
#include "c:\miniide\hcs12.inc"

getcSCI0 brclr SCI0SR1,RDRF,* ; wait until RDRF bit is set
    ldaa SCI0DRL ; read the character
    rts
```
Examples

- Write a subroutine to output a string pointed to by index register X to the SCI0 using the polling method.
- Solution: This subroutine will call putcSCI0( ) repeatedly until all characters have been sent.

```assembly
putsSCI0 ldaa 1,x+ ; get a character and move the pointer
    beq done ; is this the end of the string
    jsr putcSCI0
    bra putsSCI0

done rts
```
Examples

- Write a subroutine to input a string from SCI0. The string is terminated by the carriage return character and must be stored in a buffer pointed to by index register X.
- Solution: This subroutine will call getcSCI0( ) repeatedly until the carriage return character is sent.

```assembly
CR equ $0D
getcSCI0 jsr getcSCI0
cmpa #CR ; is the character a carriage return?
beq exit
staa 1,x+ ; save the character in the buffer pointed to by X
bra getsSCI0 ; continue
clr 0,x ; terminate the string with a NULL character
rts
```

EECE 218 – Microcontrollers
We will use SCI1 (SCI0 is used by DBUG-12)

Method for IT-driven output:

» Principle: XMIT requests an IT when TDRE or TC.
  – Transmit Data Reg Empty or Transmit Complete

» Main program: initialize SCI, do *not* enable XMIT

» When needed: set up variables, enable XMIT

» In ISR:
  – Send next character to XMIT
  – If no next character, disable XMIT