

---

# Serial data communication & serial communication interface (SCI)

---

Text Chapter 15.1, 15.2

Web Resources:

[http://en.wikipedia.org/wiki/Asynchronous\\_serial\\_communication](http://en.wikipedia.org/wiki/Asynchronous_serial_communication)

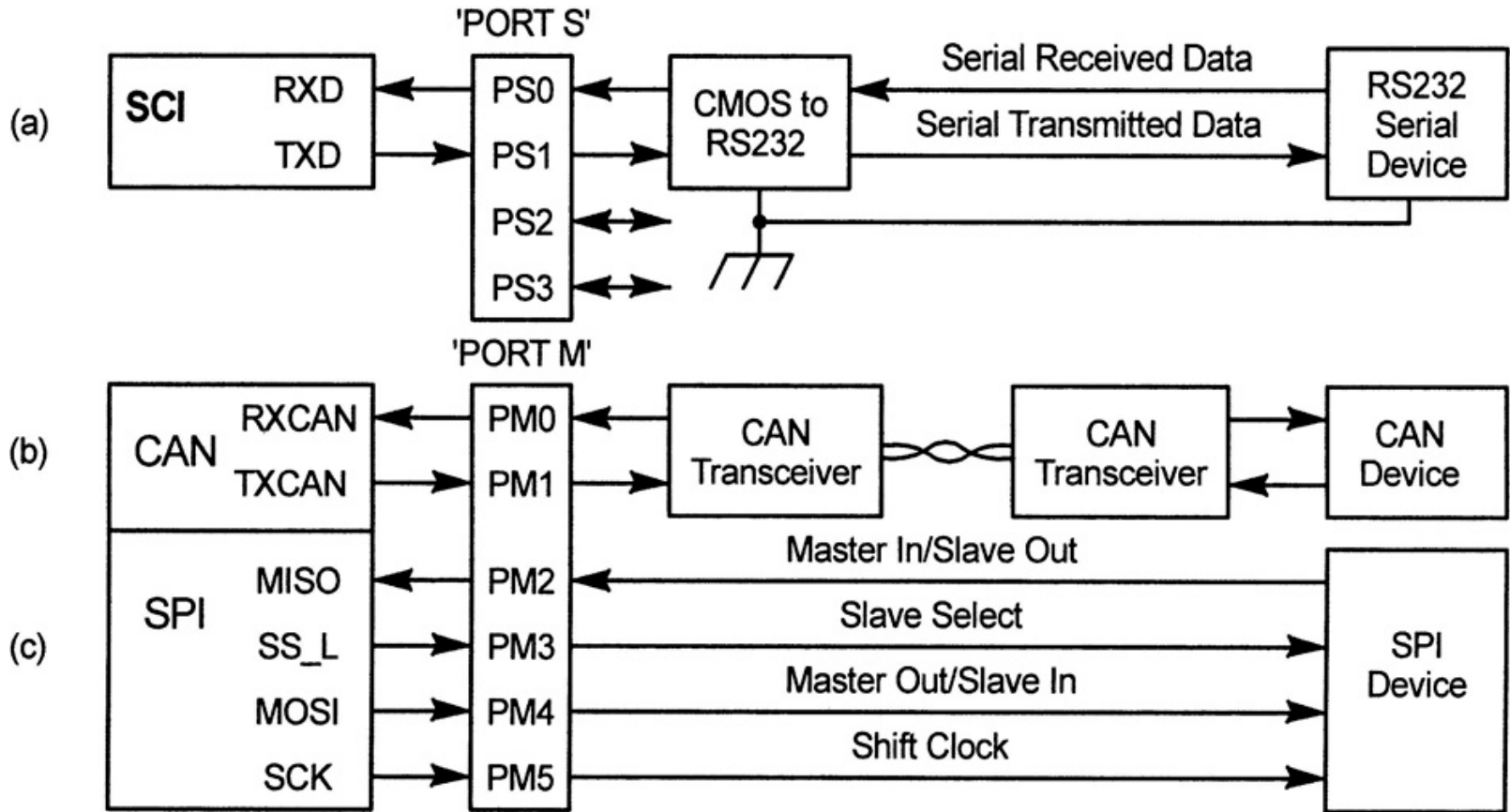
<http://www.quatech.com/support/comm-over-asyncserial.php>

---

# HCS12 Serial communication interfaces

- SCI – Asynchronous Serial Communication Interface
    - Universal asynchronous receiver-transmitter (UART)
    - Serial comm. to terminal, PC, or other serial devices
  - SPI – Synchronous Peripheral Interface
    - High-speed synchronous serial link between HCS12 and serial peripherals or other CPU
  - MSCAN – Motorola Scalable *Controller Area Network*
    - Controller for CAN 2.0 A/B network protocol (automotive networks, etc.)
-

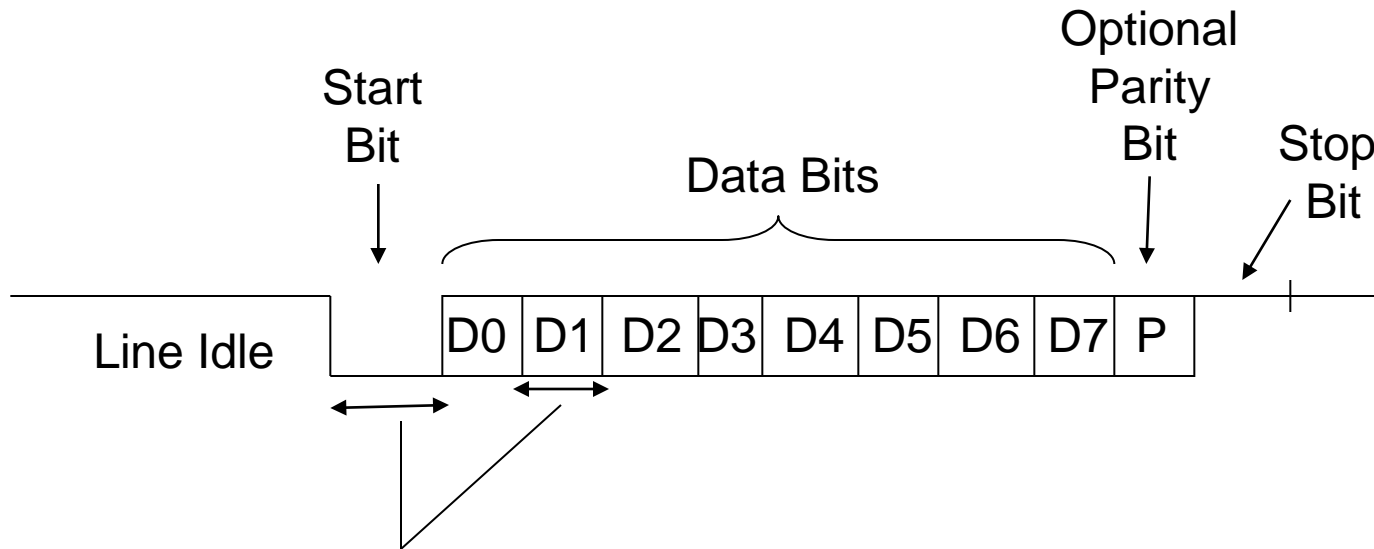
# HCS12 serial communication interfaces.



**Figure 15-1** HCS12 (a) asynchronous serial communications interface (SCI), (b) controller area network (MSCAN) interface, and (c) synchronous serial peripheral (SPI) interface.

# Asynchronous serial data format

- Transmit one byte at a time, framed by start/stop bits
- Transmitter & receiver agree on baud rate & options
- NRZ (non-return to 0) signaling

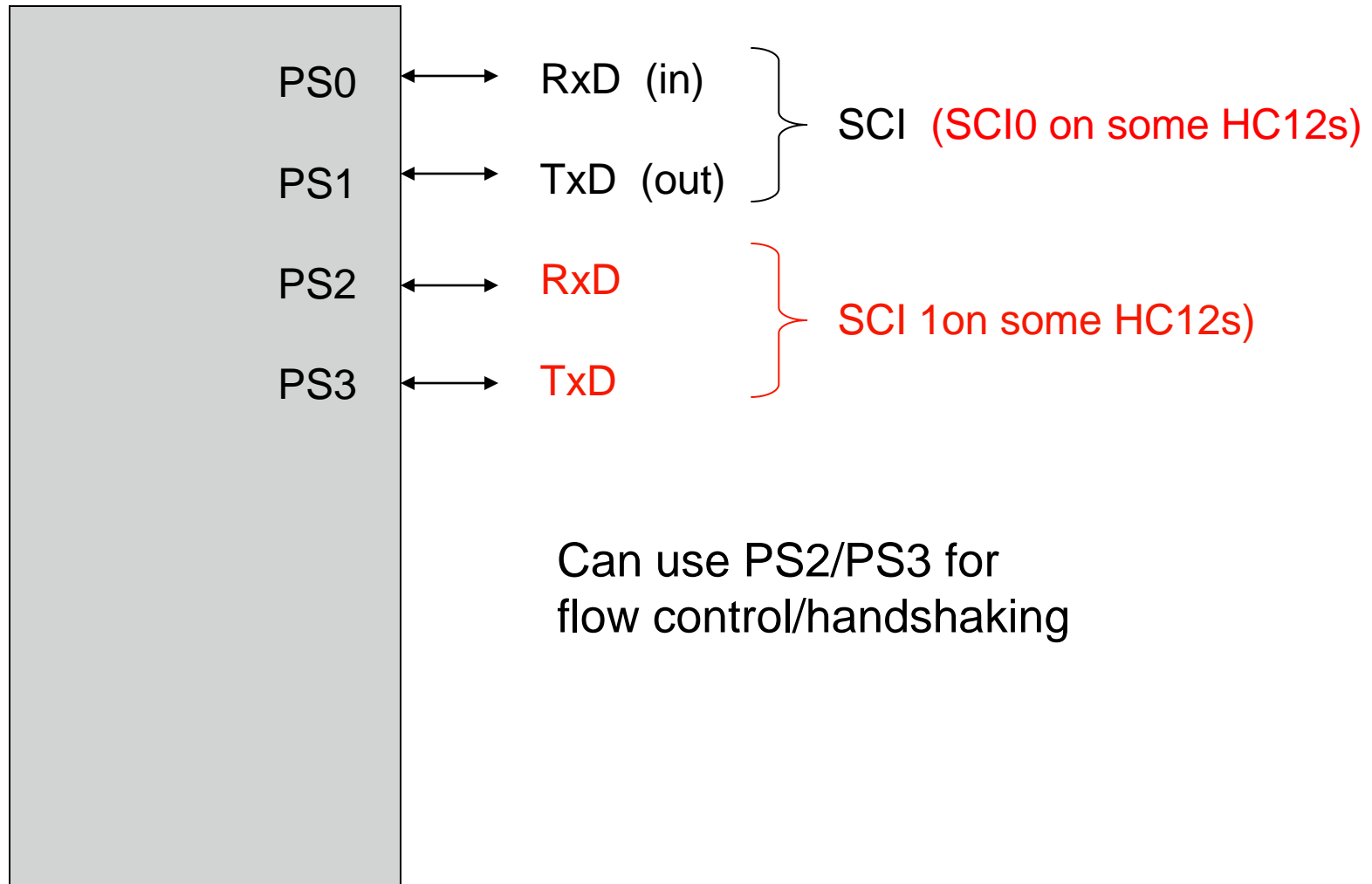


Bit time  $T = 1/(\text{bit rate})$  - determined by transmit clock  
- bit rate also referred to as "baud rate"

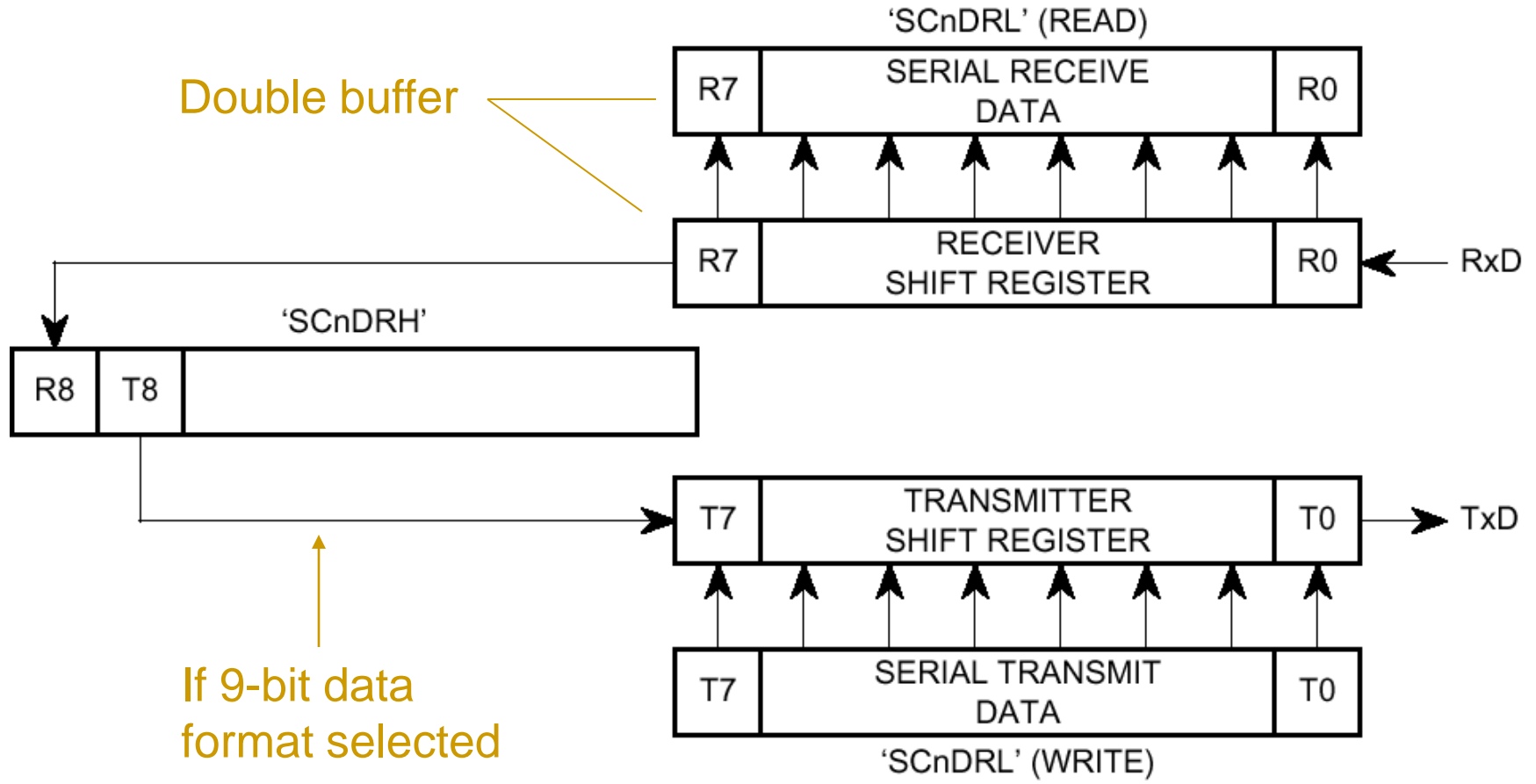
# Serial Communication Interface (SCI)

- Transmit & receive asynchronous serial data
- Full duplex operation
  - Receive data pin TxD = pin PS0 (Port S)
  - Transmit data pin RxD = pin PS1
  - Also supports half-duplex operation: TxD is bidirectional
- Programmable baud rate
- Programmable data format (8 or 9 data bits)
- Programmable parity generation/check
- Supports polling and interrupt-driven operation
- Detects several types of receive errors
- Special mode for loop-back test

# 68HC11 Ports S (PORTS \$0248, DDRS=\$024A)



# Figure 15-2 SCI data registers. (SCIDRH:SCIDRL = \$00CE:\$00CF)



# SCI Control Register 1 – Mode Control (SCICR1 = \$00CA)

7	6	5	4	3	2	1	0
LOOPS	SCIWAI	RSRC	M	WAKE	ILT	PE	PT

M : # data bits (0=8 bits, 1=9 bits)

PE : parity enable (1=enabled, 0 = disabled)

PT : parity type (If parity enabled: 0=even, 1=odd)

## Special mode bits:

LOOPS : enable SCI loop-back & single-wire mode (on TxD)

SCIWAI: stop in “wait mode” (default is enabled)

RSRC : receiver source (if LOOPS=1) – from internal xmit or TxD pin

WAKE : wakeup condition (1 => idle line (default), 0 => address line)

ILT : idle line type (RxD=1 for more than one char time)

# SCI Control Register 2

(SCICR2 = \$00CB)

7	6	5	4	3	2	1	0
TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

RE & TE : Receiver & transmitter enable

TIE : Transmitter interrupt enable

Interrupt when OK to write new byte to data register

TCIE : Transmit complete interrupt enable

Interrupt when all bits finished transmitting

RIE : Receiver interrupt enable

Interrupt when new received character in data register

ILIE = Idle line interrupt enable : line high > 1 bit time

RWU = Receiver wakeup control (sleep until wakeup code received)

SBK = Send break character

# Baud rate selection

(SCIBDH:SCIBDL = \$00C8:\$00C9)

- 13-bit # in baud rate register (BR)
- Divides M clock frequency to 16 x baud rate

$$SCIBaudRate = \frac{BusClock}{16 \times SCIBR[12:0]}$$

$$SCIBR = \frac{BusClock}{16 \times SCIBaudRate}$$

Standard baud rates (M clock 8MHz)

Baud Rate	BR value
110	4545
1200	417
9600	52
14,400	35
19,200	26
38,400	13

# SCI Status Register 1

(SCISR1 = \$00CC)

7	6	5	4	3	2	1	0
TDRE	TC	RDRF	IDLE	OR	NF	FE	PF

TDRE : transmit data register empty flag

TC : transmit complete flag (OK to write new data)

RDRF : receive data register full flag (new data received)

IDLE : idle line detected flag

## Error flags

OR = receiver overrun flag (new data overwrite unread data)

NF = noise flag (multiple bit samples not identical)

FE = framing error flag (0 when stop bit expected)

PF = parity error flag (received parity incorrect)

# SCI Status Register 2

(SCISR2 = \$00CD)

7	6	5	4	3	2	1	0
0	0	0	0	0	BK13	TXDIR	RAF

BK13: Break xmit char length (0=10/11 bits, 1=14/14 bits)

TXDIR: TxD pin direction in single-wire mode (0-in, 1-out)

RAF : receive active flag (1= reception in progress)

---

# Clearing SCI flags

- Clear SCI flags in two steps:
  - 1. Read SCI status register
  - 2. Read or write SCI data register  
(any set flags will reset)

- op's must be in this order

- may have other op's in between them

---

# HCS12 SCI I/O example (1)

; Subroutines for SCI input and output.

; Bits for control register 1

MODE: EQU %00010000 ; Mode bit  
PE: EQU %00000010 ; Parity Enable  
ODD\_P: EQU %00000001 ; Set odd parity

; Bits for control register 2

TE: EQU %00001000 ; Transmitter Enable  
RE: EQU %00000100 ; Receiver Enable

; Status register bits

TDRE: EQU %10000000 ; TX Data Register Empty  
RDRF: EQU %00100000 ; RX Data Register Full

; Baud rate register value

B9600: EQU 52 ; Baud rate = 9600

# SCI example (2)

```
.*****
```

```
,
```

```
; Subroutine init_sci
```

```
; Initialize SCI to 1 start, 8 data and 1 stop bit, odd parity and 9600 Baud.
```

```
; Inputs: None
```

```
; Outputs: None
```

```
; Reg Mod: CCR
```

```
init_sci:
```

```
    bclr    SCICR1,MODE           ; Set 1 start, 8 data and 1 stop bit
```

```
    bset    SCICR1,PE | ODD_P    ; Choose odd parity and enable it
```

```
    bset    SCICR2,TE | RE       ; Enable transmitter and receiver
```

```
    ldd     #B9600
```

```
    std     SCIBDH               ; Set baud rate
```

```
    rts
```

# SCI example (3)

```
.*****  
,
```

```
; Subroutine sci_input
```

```
; Get a character from SCI – wait if necessary
```

```
; Inputs: None
```

```
; Outputs: A = character
```

```
; Reg Mod: A, CCR
```

```
sci_input:
```

```
    brclr    SCISR1,RDRF,sci_input    ;wait for character
```

```
    ldaa    SCIDRL                    ; read data register
```

```
    rts
```

# SCI example (4)

```
.*****  
,
```

```
; Subroutine sci_out
```

```
; Send SCI data
```

```
; Inputs:  A register = data to send
```

```
; Outputs: None
```

```
; Reg Mod: CCR
```

```
sci_out:
```

```
    brclr    SCISR1,TDRE,sci_out    ; Wait for transmit data reg empty
```

```
    staa    SCIDRL                ; Output the data and reset TDRE
```

```
    rts
```



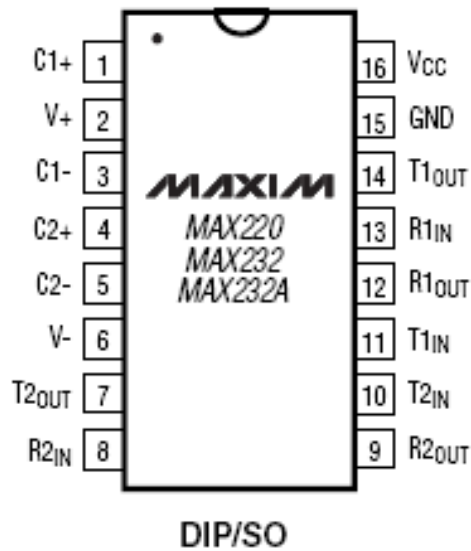
# RS-232 communication protocol

(one of many available standard protocols)

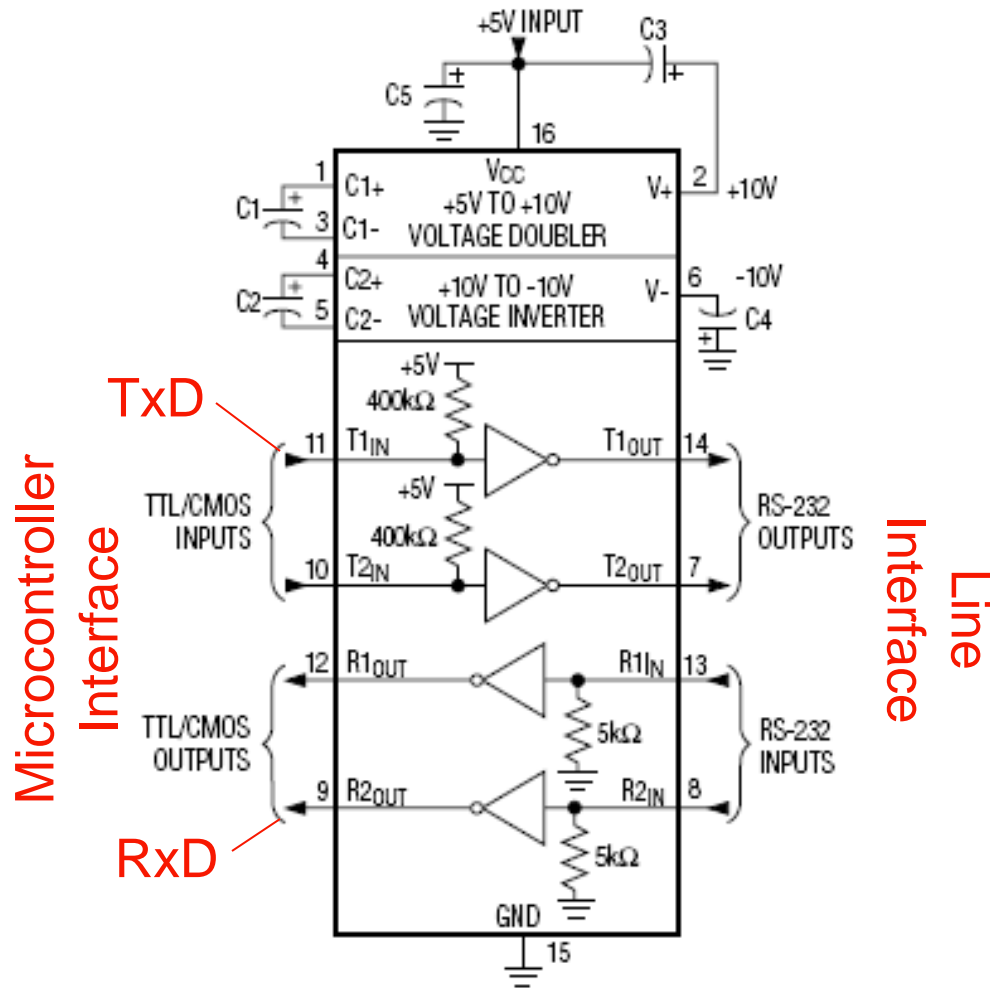
- Reliable up to about 50 feet
  - Other standards can drive longer lines
- Line voltages between -12 and +12 volts
  - Logic 1 voltage < -3 volts
  - Logic 0 voltage > +3 volts
  - Special drivers/receivers required between line and microcontroller
- Optional flow control signals
  - CTS\* = clear to send
  - RTS\* = request to send
  - DSR\* = data set ready
  - DTR\* = data terminal ready

Web resource: [http://www.lammertbies.nl/comm/info/RS-232\\_specs.html](http://www.lammertbies.nl/comm/info/RS-232_specs.html)

# Maxim RS232 drivers/receivers



CAPACITANCE ( $\mu\text{F}$ )					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1



# Multidrop serial I/O network.

