CPE 390: Microprocessor Systems

Spring 2018

Lecture 13

Serial Interfaces

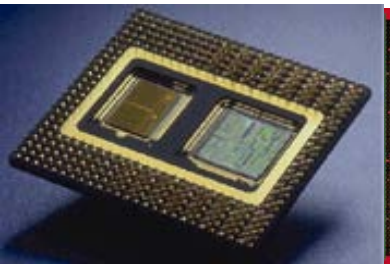
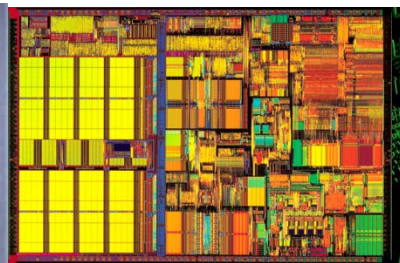
Bryan Ackland

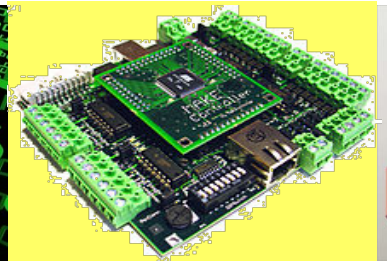
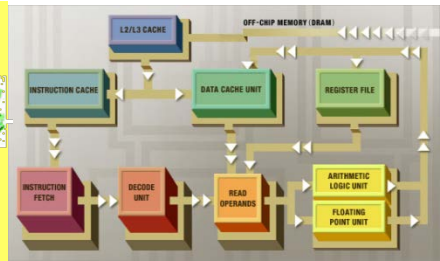
Department of Electrical and Computer Engineering

Stevens Institute of Technology

Hoboken, NJ 07030

Adapted from HCS12/9S12 An Introduction to Software and Hardware Interfacing Han-Way Huang, 2010

Why Serial Communications ?

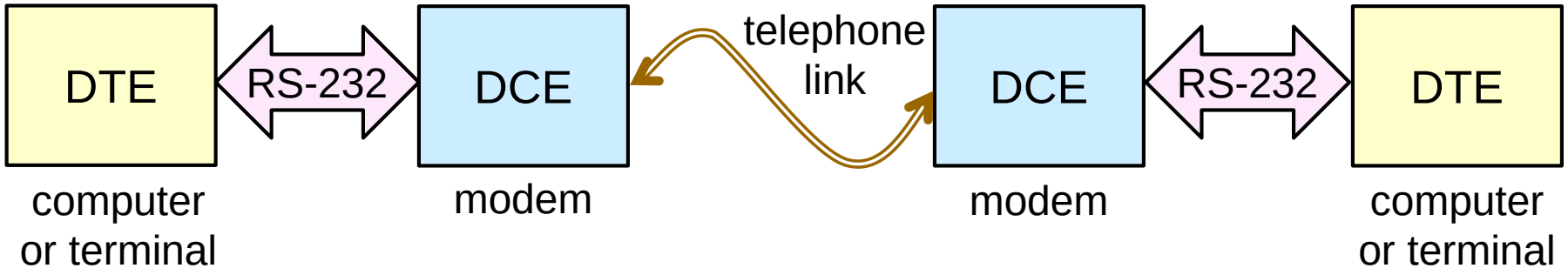
- Parallel data transfer is an efficient way to transmit high bandwidth data over short distances
  - Parallel data requires many I/O pins
  - Many I/O devices do not have high enough bandwidth to justify pin-count
- Data synchronization of parallel data is difficult to achieve over long distances
  - require differential delay between bit lines << data period
- HCS12 supports three serial communication protocols:
  - SCI Interface: Asynchronous serial transmission utilizing EIA-232 standard
  - SPI Interface: Synchronous serial link developed by Motorola for exchange between microcontrollers and peripherals
  - I^2C Interface: Synchronous serial link developed by Philips for inter-chip communications in embedded systems

Serial Communications Interface (SCI)

- Designed to full-duplex asynchronous data transmission using RS-232 (EIA-232) standard
  - full-duplex means simultaneous transmission in both directions
  - asynchronous means no clock is transmitted
- RS-232 was one of the earliest data communications protocols
  - first standard developed in 1960
  - originally used with teletypes
  - serial port on early PC's connected to terminals, printers, modems etc.
  - replaced by USB in PC's and other consumer equipment
  - still used widely in industrial products
  - connects lab EVB board to PC

RS-232 Communications Model

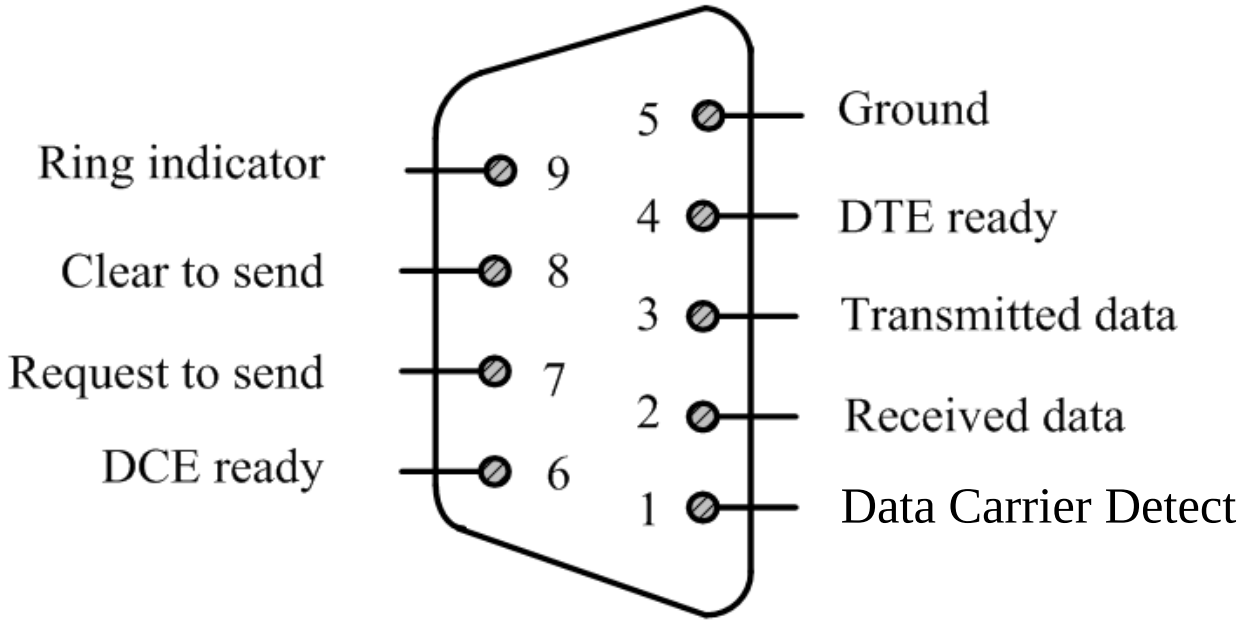
- Originally designed to transfer data over telephone lines using modems
  - Data Terminal Equipment (DTE): computers, terminals
  - Data Communications Equipment (DCE): modems, telecom equip.



- Data rates up to 20 kbps
- Link distance to 15 meters
- Full RS-232 interface specifies 22 signals
  - 2 serial lines & ground (4)
  - modem status & control signals (13)
  - backup communications and test (5)

RS-232 DB9 Connector

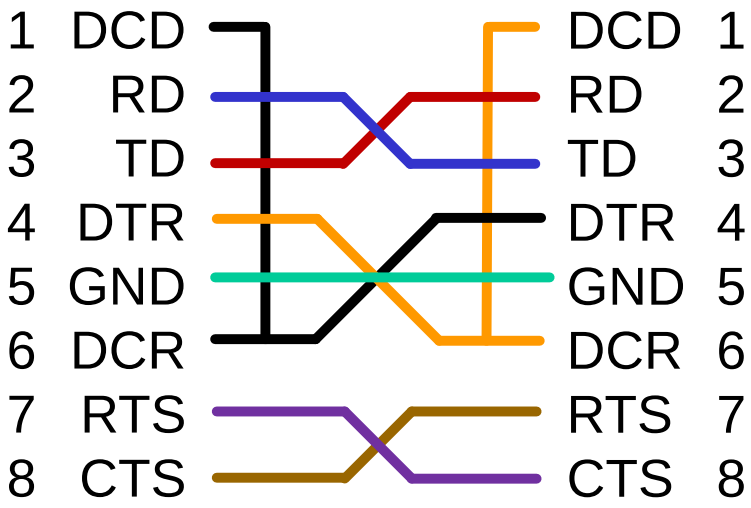
- Most applications do not require full modem control – use a simpler DB9 (9-pin) connector



  - Transmitted and Received data are the two serial data lines
  - DCE ready and DTE ready usually wired to positive voltage just to indicate presence of live equipment on line
  - RTS and CTS can be used as a hardware hand-shake to pause data when input buffer on receiving side is full

Null Modem

- Most applications of RS-232 today use direct connection of two DTE's without intervening modems
- RS-232 standard does not support DTE to DTE connection
- Use a cable that acts as a "null modem"
  - cross-connects signals to "fool" both DTE's into thinking they are communicating with a DCE (modem)



Ascii Codes

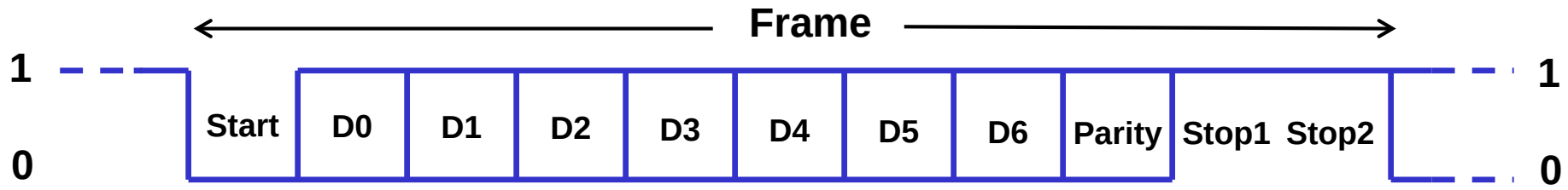
- RS-232 links are usually used to transmit ascii characters
- Each character is represented using its 7-bit ascii code
  - MSBit is sometimes used to carry a parity bit – used for error detection

LS Hex Digit

MS Hex Digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	DS	RS	US
2		!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

RS-232 Data Format

- Data is transmitted serially, one character per frame
- Each frame consists of:
  - one START bit (0)
  - 7 to 8 data bits (least sig. bit first) – frequently ASCII character
  - one optional PARITY bit
  - one or two STOP bits (1)

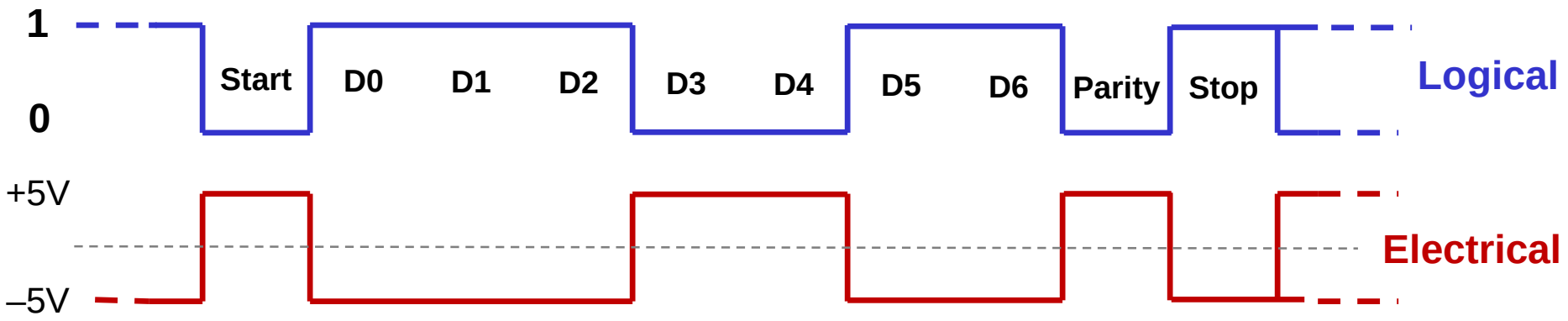


- No explicit clock signal
  - receiver uses START after line has been idle (1) for at least bit time
  - if receiver loses its place, format rapidly pulls back “into sync”
- Transmitter and Receiver must agree on:
  - baud rate (how many bit values per second)
  - number of data and stop bits and parity

RS-232 Electrical Specification

- Voltages of -3 to -25 V with respect to ground are considered logical ‘1’ (known as the *mark* condition)
- Voltages of +3 to +25 V with respect to ground are considered logical ‘0’ (known as *space* condition)
- Data rates are not specified in standard, but commonly used rates are 300, 1200, 2400, 9600 and 19200 baud

- *Example*: *Sketch the output of letter ‘g’ (ASCII $67) using a format of 1 start bit, 8 data bits (7 + odd parity) and 1 stop bit.*

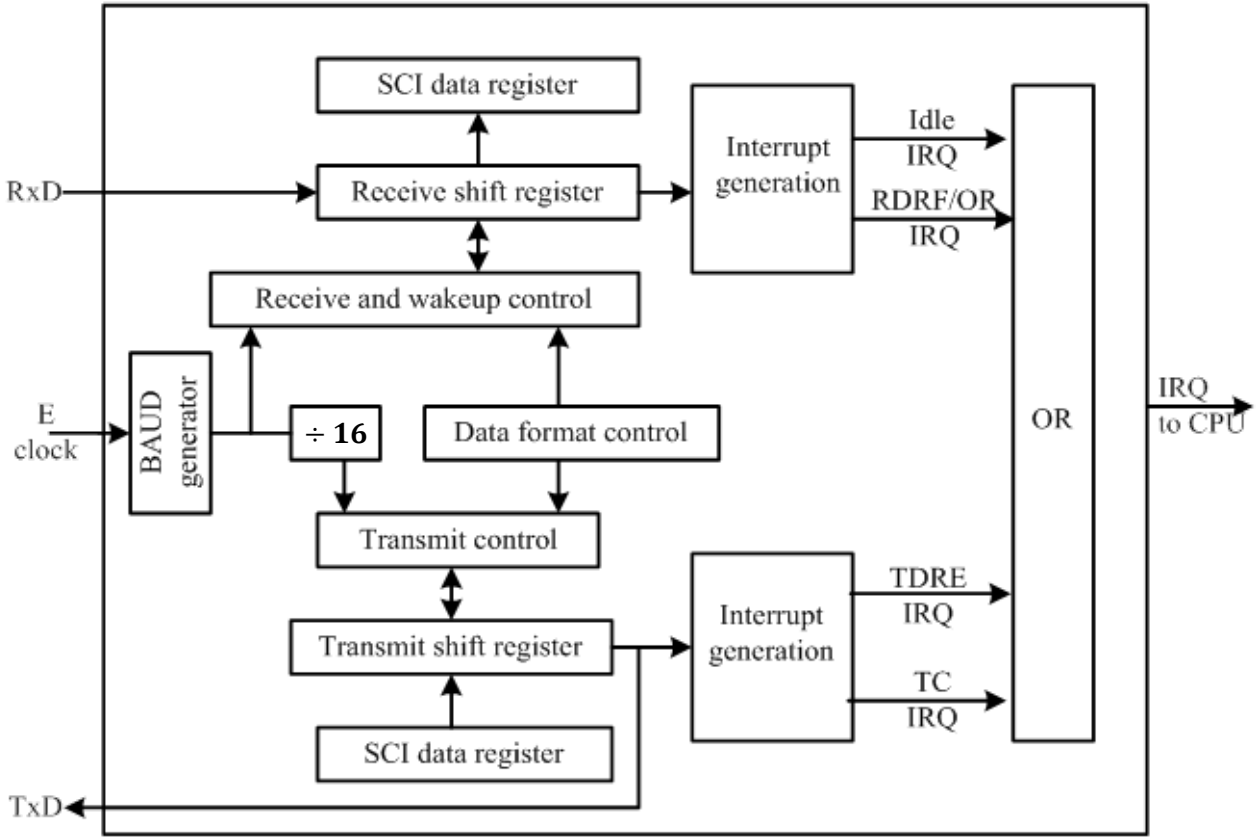


HCS12 SCI Subsystem

- HCS12 may have one or two serial communication interfaces (SCI0) and SCI1)
  - We will only describe SCI0
- SCI uses one START bit, 8 or 9 data bits* and one STOP bit
- SCI supports odd, even or no parity
  - parity is inserted in most-significant data bit position
  - option of hardware parity check on received data
- SCI shares pins with Parallel Port S
  - Receive line TxD0 uses pin PS0
  - Transmit line TxD0 uses pin PS1
- SCI supports ready flag polling and interrupts
- SCI line status can be used to wake-up processor

* *8 or 9 data bits means 7 or 8 real data bits plus parity*
if no parity is specified, transmitted parity bit is '0'

HCS12 SCI Block Diagram



- SCI has :
  - One 16-bit baud rate register (actually two 8-bit registers SCI-BDH/L)
  - Two Control registers (SCI0CR1 and SCI0CR2)
  - Two Status registers (SCI0SR1 and SCI0SR2)
  - One 16-bit data register (actually two 8-bit registers SCI0DRH/L)
    - Upper byte holds MS-bit in 9-bit (8+parity) transmissions

SCI Baud Rate Control Register

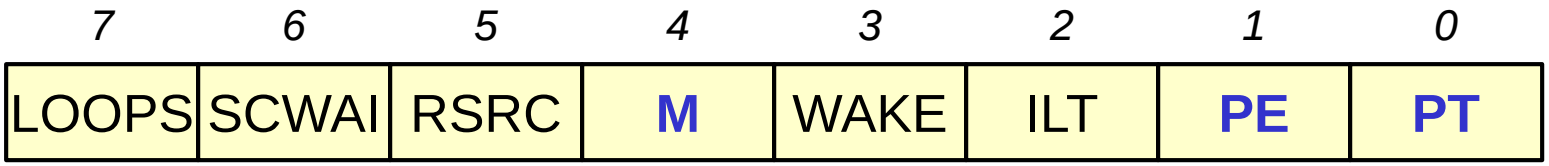
	7	6	5	4	3	2	1	0
SCI0BDH	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
	7	6	5	4	3	2	1	0
SCI0BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0

- SCI uses 16x baud rate clock to sample incoming RS-232 waveform
- SBR divides E-clock down to correct sampling frequency

$$SBR = \frac{f_{Eclk}}{16 \times baud\_rate}$$

Desired Baud Rate	Divisor for f_E = 24 MHz	Divisor for f_E = 8 MHz
300	5000	1667
1200	1250	417
2400	625	208
9600	156	52
19200	78	26

SCI Control Register 1 (SCI0CR1)



Reset value =$00

LOOPS: Loop-back test select bit (‘0’: disabled, ‘1’: enabled)*

SCWAI: SCI stop in wait mode bit (‘0’: continues in wait, ‘1’: stops in wait)*

RSRC: Receiver source bit in loop-back mode (‘0’: internal loop-back, ‘1’: external)*

M: Data format mode bit (‘0’: 8-bit data, ‘1’: 9-bit data)

WAKE: Wake-up condition bit (‘0’: idle line wakeup, ‘1’: address-mark wakeup)*

ILT: idle line type bit (‘0’: look for idle after start, ‘1’: look for idle after stop)*

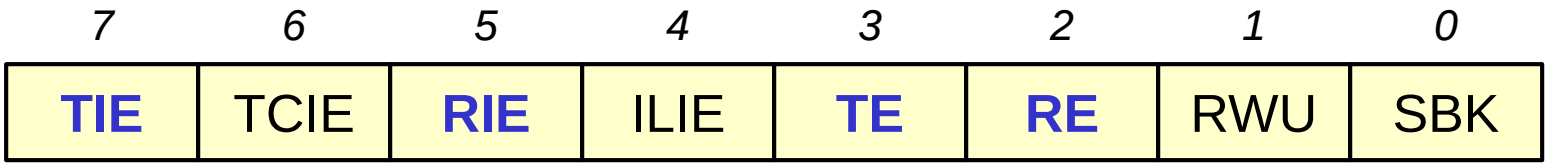
PE: Parity enable bit (‘0’: parity disabled, ‘1’: parity enabled)

PT: Parity type bit (‘0’: even parity, ‘1’: odd parity)

* *This register controls specialized features of the SCI subsystem – we will not be using these bits*

- In the EVI (lab) board, all bits in this register are set to ‘0’ during normal serial interaction between the board and the PC

SCI Control Register 2 (SCI0CR2)



Reset value =$00

TIE: Transmit interrupt enable bit (‘0’: disabled, ‘1’: enabled)

TCIE: Transmit complete interrupt enable bit (‘0’: disabled, ‘1’: enabled)*

RIE: Receiver full interrupt enable bit (‘0’: disabled, ‘1’: enabled)

ILIE: Idle line interrupt enable bit (‘0’: disabled, ‘1’: enabled)*

TE: Transmitter enable bit (‘0’: disabled, ‘1’: enabled)

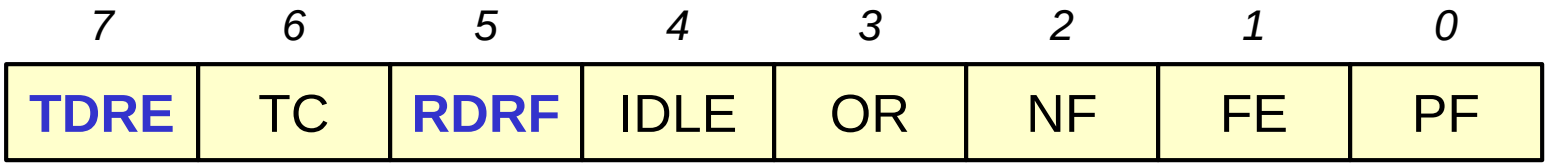
RE: Receiver enable bit (‘0’: disabled, ‘1’: enabled)

RWU: Receiver wakeup bit (‘0’: normal operation, ‘1’: specialized wakeup function)*

SBK: Send break* bit (‘0’: normal operation, ‘1’: generate a break code)*

* *We will not be using these bits*

SCI Status Register 1 (SCI0SR1)



Reset value =$00

TDRE: Transmit data register empty flag (i.e. ready to accept a new character)

TC: Transmit complete flag*

RDRF: Receiver data register full flag (i.e. have a new received character available)

IDLE**:** Idle line detected flag (i.e. receive line has become inactive)*

OR: Overrun error flag (new character received before previous character read)*

NF: Noise error flag*

FE: Framing error flag (received a ‘0’ when a stop bit was expected)*

PE: Parity error flag*

* *We will not be using these bits*

SCI Interrupts

- SCI can generate interrupt whenever any of following flags become set:

TDRE: Transmit data register empty flag (i.e. ready to accept a new character)

TC: Transmit complete flag (i.e. all bits of transmit character have been sent)

RDRF: Receiver data register full flag (i.e. have a new received character available)

IDLE: Idle line detected flag (i.e. receive line has become inactive)

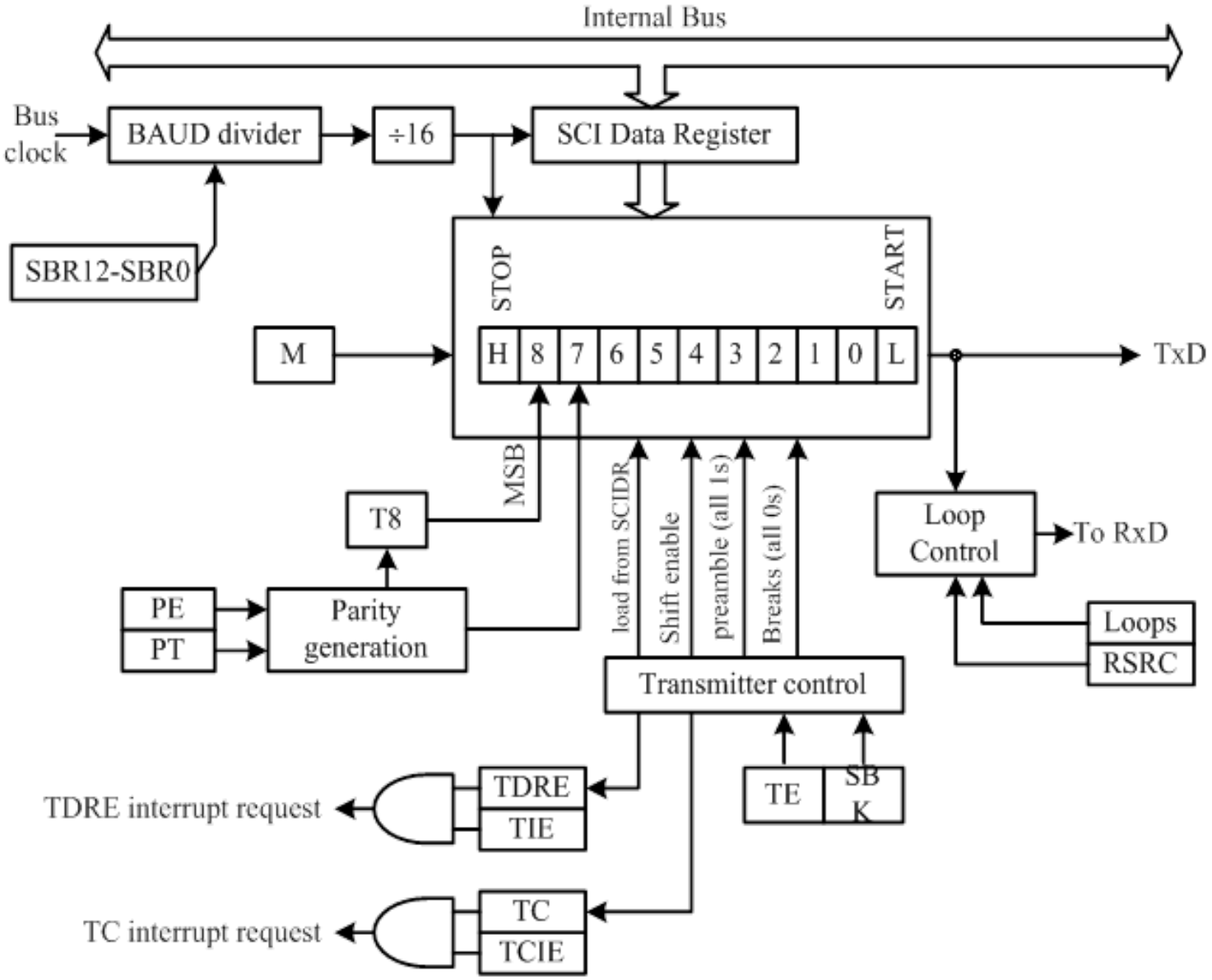
OR: Overrun error flag (new character received before previous character read)

- One or more of these interrupt sources may be enabled using various interrupt enable bits in SCI0CR2
- There is only one interrupt vector associated with each SCI
- When interrupt occurs, ISR must check flags in SCI0SR1 to determine which type of event caused interrupt

Character Transmission

- To transmit characters, first need to set up SCI
  - Select a baud rate by writing SCI0BDH/L
  - Write SCI0CR1 to configure word length, parity & any other features
  - Enable transmitter by setting TE and optional interrupt enable bits in SCI0CR2
  - A pre-amble of 10 logic ‘1’s will be transmitted
- For each character:
  - poll the TDRE flag by reading SCI0SR1 register (or wait for interrupt)
  - once flag is set, write next character to SCI data register SCIDRL
- When transmit shift register is empty and SCIDRL is full:
  - SCI transfers new data in SCIDRL to transmit shift register and sets TDRE flag
  - SCI shifts data out one bit at a time on TxD pin
  - Once shift register is empty, SCI sets TC flag in SCIOSR1 and sets output pin TxD to ‘1’ (idle)
  - Interrupts may be requested on TDRE or TC

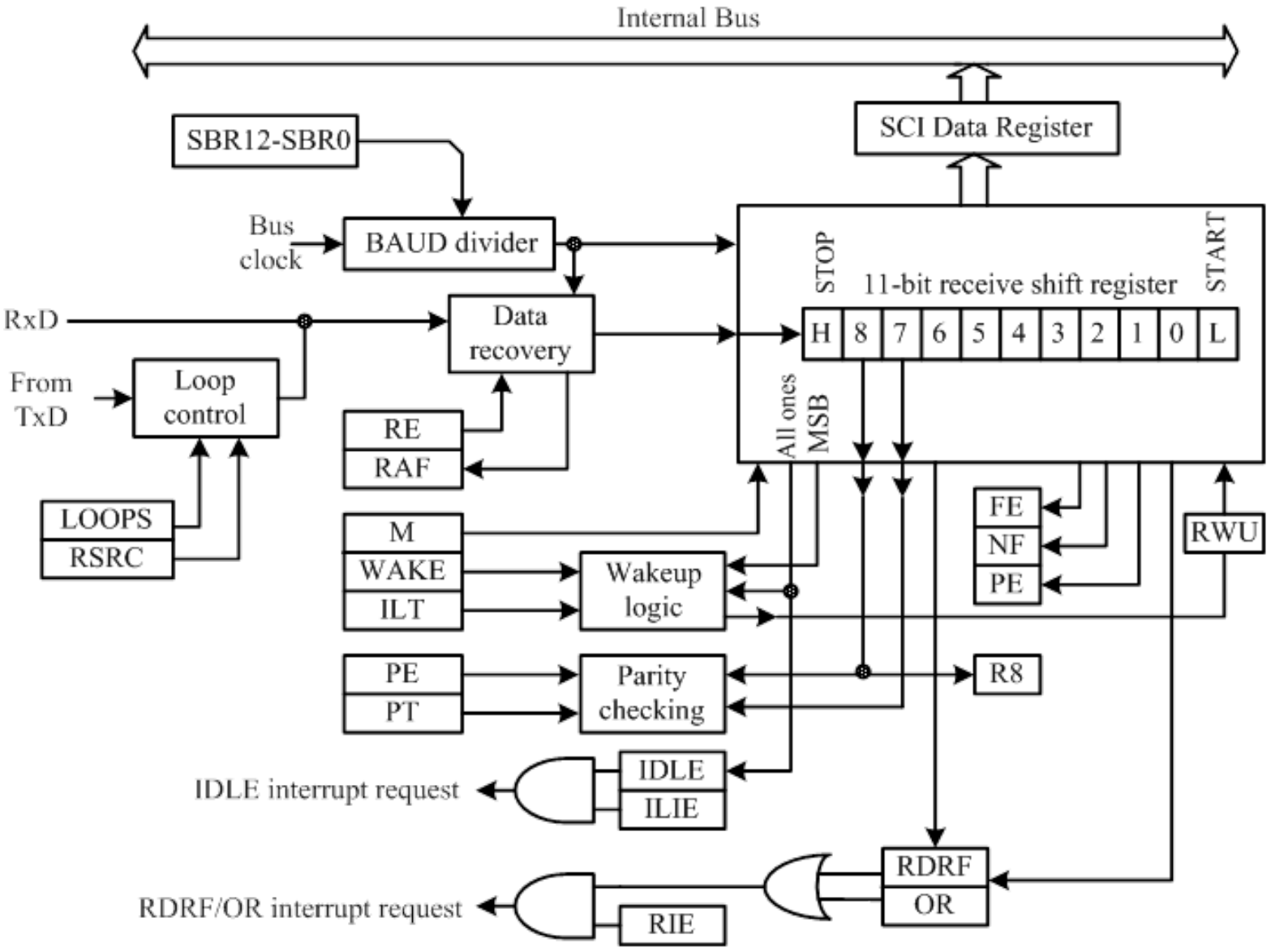
SCI Transmitter Block Diagram



Character Reception

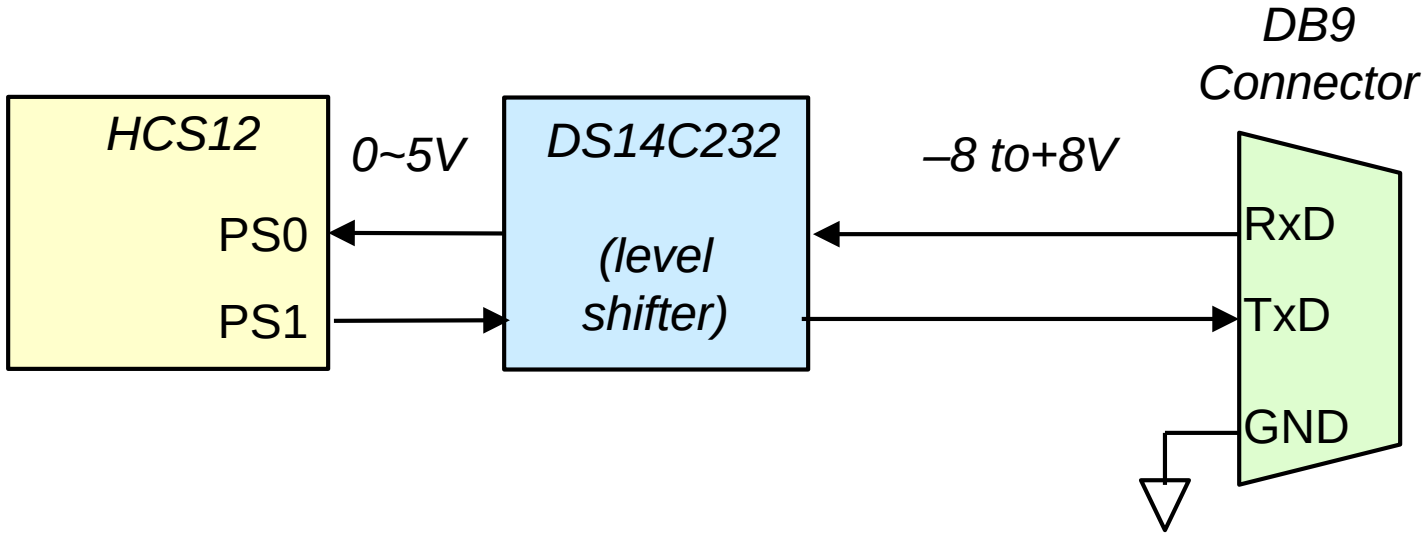
- To receive characters, in addition to transmitter setup:
  - Enable receiver by setting RE and optional interrupt enable bits in SCI0CR2
- For each character:
  - poll the RDRF flag by reading SCI0SR1 register (or wait for interrupt)
  - once flag is set, read next character from SCI data register SCIDRL
- When receive shift register is full:
  - If SCIDRL is empty, SC1 transfers data from receive shift register to SCIDRL and sets RDRF flag
  - If SCIDRL is full, SC1 does not transfer data to SCIDRL and sets overrun flag OR in SCI0SR1. New data in receiver shift register is lost. Previous data in SCIDRL is retained.
  - If software cannot keep up with receive data rate, some kind of flow control is required. Flow control can be achieved in hardware (RTS/CTS lines using extra pins) or via software protocol (XON/XOFF control characters)
  - Interrupts may be requested on RDRF or OR

SCI Receiver Block Diagram



Example: Output Character Subroutine

- Assuming SCI has already been initialized to send and receive characters at a specified baud rate, write a subroutine to output the character in accumulator A to the RS-232 port using the polling method




```
            include   "hcs12.inc"
putcSCI0:   brclr     SCI0SR1, $80, *    ; wait for TDRE to be set
            staa      SCI0DRL            ; output the character
            rts
```

Example: Input Character Subroutine

- Write a subroutine to read a character from SCI0 using the polling method. The character should be returned in accumulator A.

```
        include  "hcs12.inc"
getcSCI0: brclr  SCI0SR1, $20, *   ; wait for RDRF to be set
        ldaa     SCI0DRL           ; read the character
        rts
```

Example: Output String Subroutine

- Write a subroutine to output a NULL-terminated string to the SCI0. Do not output the NULL. Register X contains a pointer to the string.

```
            include  "hcs12.inc"
putsSCI0:   ldaa     1, x+          ; get a character and increment pointer
            beq      done           ; end of string?
            jsr      putcSCIO
            bra      putsSCIO       ; go to next character
done:       rts
```

Example: Input String Subroutine

- Write a subroutine to input a carriage-return (enter) terminated string from the SCI0. Store the string in a buffer pointed to by the X register. Echo all typed characters. Echo “carriage-return” (ENTER) as “line-feed”. Allow the use of back-space to erase characters

```
CR:     EQU   $0D      ; carriage return (ENTER) code
LF:     EQU   $0A      ; line-feed code
BS:     EQU   $08      ; back-space code
WS:     EQU   $20      ; (white) space code
```

Example: Input String Subroutine (2)

```
getsSCI0:  jsr     getcSCI0      ; get a character from SCIO
           cmpa    #CR           ; is it carriage return?
           beq     stend
           staa    0,x           ; save the character
           jsr     putcSCI0      ; echo the character
           cmpa    #BS           ; is it a backspace character?
           bne     nc            ; no, continue
           dex                   ; decrement buffer pointer
           ldaa    #WS           ; output space character (to erase previous)
           jsr     putcSCI0
           ldaa    #BS           ; backspace over space character
           jsr     putcSCI0
           bra     getsSCI0      ; go get next character
nc:        inx                   ; increment pointer
           bra     getsSCI0
stend:     ldaa    #LF           ; echo LF for CR
           jsr     putcSCI0
           clr     0,x           ;terminate string with NULL
           rts
```