Hitachi America, Ltd.

Application Note

H8/300 SCI

Carol Jacobson

Clocked Synchronous Serial Communication

INTRODUCTION

Most embedded applications require a secondary communication interface between the main controller and data collection peripherals or devices. Controllers are frequently used as slave devices for pre-processing raw data before sending it on to a higher speed master processor. Each of these design situations requires the transfer of information over long distances or a low level handshake to non-time-critical support devices. Currently available controllers provide this interface via an on-chip serial data channel(s) for local or board level communication. A simple 2-wire serial link provides an ideal data transfer path without adding significantly to board space, cost or system complexity.

Hitachi's H8/300 controller products support an on-chip Serial Communication Interface (SCI) bus, providing a multi-option channel for low to moderately high speed data transfers. The SCI channel is configurable for either asynchronous or synchronous formats and driven either by the on-chip baud rate generator or from an external clock source. This note details SCI operation in synchronous mode for maximum transfer rates. Henceforth, the term 'transfer' will refer to either data transmission or reception.

THE SYNCHRONOUS DATA FORMAT

The simplicity of the synchronous data format allows transfer operations significantly faster than asynchronous formats. The increased speed, however, results in increased complexity and a greater potential for errors. Synchronous transfers use the serial clock to define the start of a transfer and the stop point. This design eliminates the additional 20% to 30% bit time overhead consumed by start, stop and parity characters associated with an asynchronous byte frame. For example, the H8/300 family supports byte synchronous transfers up to 2.5M bits per second but only 156.5K bits per second in asynchronous mode.

Synchronous data is normally arranged and handled in large blocks of 8-bit values, with no distinction made between bytes. All bytes are assumed to be of equal priority and have no system meaning. Transfers begin on the rising edge of the first serial clock following TE enabled. On this edge, the first (LSB) bit of data in the Transmit Shift Register is placed on the TxD data line. The receiver latches the data 1/2 cycle later on the falling edge of the same clock cycle. Thereafter, one bit of data is transferred on each serial clock independently of the ready state of the receiver or the validity of the data (figure 1). If the receiver and transmitter are not synchronized to the same clock cycle, or if data is transmitted too early or too late, the remainder of the data block will be corrupted.

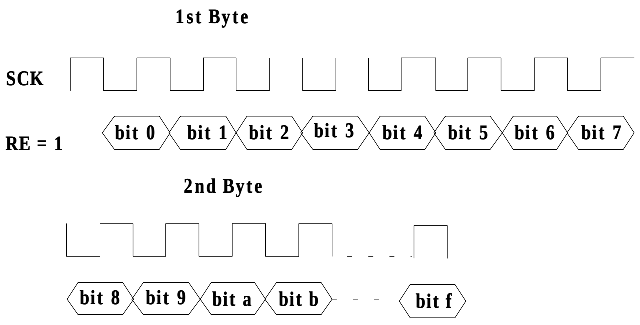

Figure 1

A data transfer with only 5% skew between the transmit and receive clocks begins to loose data after the fifth bit. The receiver latches the sixth bit, on the rising edge of the transmit clock just as the transmitter places the 7th bit on the data bus. The receiver continues to latch invalid data from this point (Figure 2). This error will only be recognized if the receiving device performs an error check on the data.

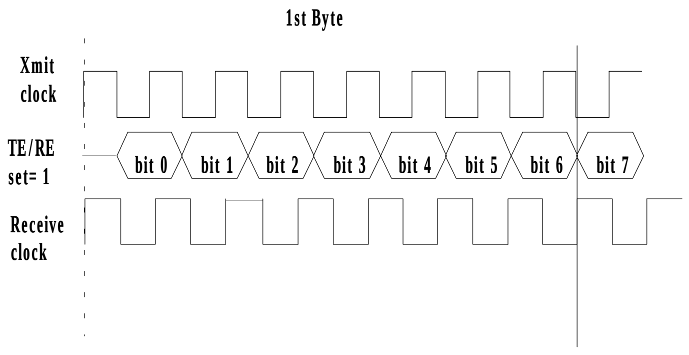

Figure 2

ERROR CHECKING

As described above, in the interest of speed there are no automatic error checks embedded in synchronous data. Intelligent systems employ a method for qualifying incoming blocks and requesting a repeat transmission when inconsistencies are detected. One such technique involves adding one or two bytes of error code to the end of each block. At the end of the transfer the receiver uses the additional information to verify the accuracy of the block. Several types of error coding are commonly employed. However, cyclic redundancy check (or CRC) is probably the most powerful and one of simplest to implement. This technique ends each block with a data byte representing the result of a mathematical interpretation of the entire block. The receiver CPU performs the same calculation on the received data and compares the result to the received error code. A mis-match of results indicates an error somewhere in the block. A second, less efficient but simpler and safer technique, is a vertical (or longitudinal) parity check. Parity checking has the advantage of not requiring additional bytes sent which will reduce transmission speed and may also be corrupted.

One of the most common problems associated with synchronous transfers is a data overflow. A complete byte of data will be missed or overwritten if the transmitter begins shifting the third byte onto the data line before the receiver's CPU has completed reading the first byte from the receive data register. Incoming data overflows and overwrites the second byte stored in the receiver's shift register. Data overflow is generally encountered in heavily interrupt driven designs where the receiver CPU is prevented from reading data within one byte time. Unfortunately, in this situation the last and subsequent bytes of data are lost or scrambled. The H8/300 SCI port offers an overflow error detection (ORER) bit and interrupt to alert the receiver of an overflow condition. The H8 disables data reception until the ORER error bit is cleared allowing the receiver to request a repeat transmission.

INITIALIZATION

Setting bit 7, C/A in the Serial Mode Register (SMR), places the SCI in synchronous mode.

SMR

7	6	5	4	3	2	1	0
C/A	CHR	PE	O/E*	STOP	-	CKS1	CKS0
1	X	X	X	X	1	0/1	0/1

Bits 2 through 6 are ignored.

SCR

7	6	5	4	3	2	1	0
TIE	RIE	TE	RE	-	-	CKE1	CKE0
0/1	0/1	0/1	0/1	1	1	0/1	x

SETTING THE CLOCK

The source and frequency of the serial clock are controlled through:

1. The Serial Mode Register (SMR) bits CKS0 and CKS1
2. Serial Control Register (SCR) bit CKE1
3. The Bit Rate Register (BRR).

To avoid errors, most designs use a single clock source for the transmitter and receiver. When the SCI internal clock source is selected, the serial clock is also driven out on the CSCK pin regardless of the state of the CKE0 bit. This provides a common source that can be fed into the partner device.

When the SMR bit CKE1 is set to 1, the SCI looks for an external clock source input on the CSCK pin. At this time, the CSCK pin automatically switches to input mode and all other clock control bits are ignored. In synchronous mode the maximum external clock input is 4 tcyc or 2.5 MHz for a 10 MHz CPU clock.

When CKE1 is set to 0 (default value),.the on-chip baud rate generator provides the serial clock. CKS1 and CKS0 select the oscillator pre-scale value as follows:

n	CKS1	CKS0	Clock
0	0	0	Φ
1	0	1	Φ/4
2	1	0	Φ/16
3	1	1	Φ/64

The prescale value (n), the BRR value and the external oscillator frequency combine to determine the serial clock frequency. H8/300 Hardware Manuals provide tables for BRR values needed to produce common bit rates at standard oscillator frequencies. The tables show BRR as a function of baud rate and oscillator frequency (note: the frequency given in the tables refers to the speed of the external crystal (XTAL), not to the CPU clock). Due to the integer requirement for BRR, all serial clock frequencies are not available for all XTAL frequencies. In most cases, a setting is available that will produce a serial clock rate very close the desired frequency.

Example: Using a 20MHz oscillator (10MHz CPU), from equation 1, we can determine bit rates approaching but not equal to 1Mbps.

n	BRR	Bit Rate
0	3	625K
0	2	833K
0	1	1.25M

Eq. 1:

$$\text{Bit rate} = \text{osc freq.(sec)} / [8 * 2^{2n} * (N+1)]$$

where:

osc	The external crystal oscillator frequency (EXTAL)
N	The Bit Rate Register (BRR) value
n	the internal clock source

Alternatively, custom bit rates are produced by determining the BRR value for a specific rate.

Equation 1 converts to eq. 2:

$$N = \text{osc freq.(sec)} / [8 * 2^{2n} * \text{baud rate}] - 1$$

Example: (for bit rate = 125K, n = 0, osc freq. = 20 MHz = 20 x 10^6)

$$N = 20\text{x}10^6 / [8 * 1 * (125 * 10^3)] - 1 = 19$$

Decimal BRR values should be rounded to the nearest integer. This integer value can be used to calculate the exact frequency from equation 1.

TRANSMIT OPERATION

Setting the TE bit enables the transmitter. When TE is set, the I/O line multiplexed with TxD switches to output operation and the pin pulls high regardless of the contents of the port data and Data Direction Registers. Transmission begins by loading the first byte of data into the Transmit Data Register and clearing the TDRE bit. Data automatically moves from the Transmit Data Register (TDR) to the Transmit Shift Register (TSR) and the TDRE flag is set. The CPU should respond either to the resulting TxI interrupt or TDRE flag state by loading the next data byte into the TDR. One bit is shifted from the TSR onto the TxD line on the rising edge of each serial clock pulse. The data shifts out LSB first.

RECEIVE OPERATION

Setting the RE bit forces the multiplexed RxD I/O line to a high impedance input state regardless of the contents of the port data and data direction registers. Data bits, or the logic state of the RxD line, are latched and shifted into the Receive Shift Register (RSR) on the falling edge of each serial clock, LSB first. When 8 bits have been received the completed byte automatically moves from the RSR to the Receive Data Register, the RDRF flag is set and a CPU interrupt is issued if enabled. Reception of the next byte begins on the next falling edge.

POLLED CONTROL

SSR

7	6	5	4	3	2	1	0
TDRE	RDRF	ORER	FER	PER	-	-	-
1/0	0/1	0/1	X	X	1	1	1

Bits 6 and 7 of the Serial Status register provide feedback on the current status of a data transfer. The Transmit Data Register (TDR) is empty and ready to receive the next byte when the TDRE is set to 1. When TDRE is set, data in the TDR has been moved to the Transmit Shift Register (TSR) and is being shifted out onto the TxD line LSB first. To ensure continuous transfers, software must move the next data byte into the TDR, clear the TDRE bit, and update data pointers before all 8 bits have been shifted out of the shift register. These operations must be completed within 8 serial clock cycles.

Serial Status Register bit 6, the Receive Data Register Full (RDRF) flag, is set when a byte of data moves from the Receive Shift Register (RSR) into the Receive Data Register (RDR). The next byte immediately begins shifting into the RSR on the falling edge of the clock. To prevent an overflow condition, the CPU must read the contents of the RDR, clear the RDRF flag, and update data pointers before the RSR has received 8 bits of new data. Therefore, as with the transmitter, these operations must be completed within 8 serial clock cycles.

Although the SCI supports byte transfers up to 2.5 MHz, continuous transmission and reception is limited by the ability of the CPU to update and service port operations within the required time frame. A minimum polling routine and associated instruction timing for continuous transmission is shown in figure 4. With a 10 MHz system clock the maximum continuous transfer rate is 2 MHz.

INTERRUPT CONTROL

TIE (bit 7 of the Serial Control Register, SCR) enables the Transmit Data Register Empty interrupt. The SCI issues an interrupt when the TRD is empty and TDRE = 1.

SCR bit 6, RIE, enables the following two interrupts:

- Receive End, RxI
- Receive Error, error flags= 1

When RIE is set, the SCI issues an interrupt when 8 bits of data have been received. If the RDRF bit is not cleared before the RSR receives 8 new bits of data, a Receive Error Interrupt is issued to signal the overflow condition. Although enabled by the same control circuit, RxI and ERI are handled by separate interrupt vectors.

Once a block transfer has started, an SCI interrupt will occur once each 8 serial clock cycles. If RxI/ERI and TxI interrupts occur simultaneously, the receiver interrupt takes priority over the transmitter. REI takes priority over RxI. Within the H8/300 architecture interrupts from the SCI take a low priority. As noted above, to guarantee continuous synchronous transfers the CPU must be able to update SCI data registers, status bits, and pointers before the next byte of data has been received or transmitted. When other interrupt sources are enabled, the designer must ensure the timing requirement can be met. Figure 5 shows an example of a transmitter interrupt service routine.

Whether operating by polled or interrupt driven control, if the TDR is not updated before all 8 bits have been shifted out of the Transmit Shift Register, the TxD line is held at the logic level of the last bit.

Likewise, if the Receive Data Register is full and the RDRF bit is not cleared before 8 bits have been received by the RSR, Serial Control register bit ORER is set to flag a receive overrun condition and data reception stops until the ORER is cleared. The last byte of data is not transferred from the RSR into the RDR.

Figure 7a, 7b & 7c gives flow charts for standard transfer operations.

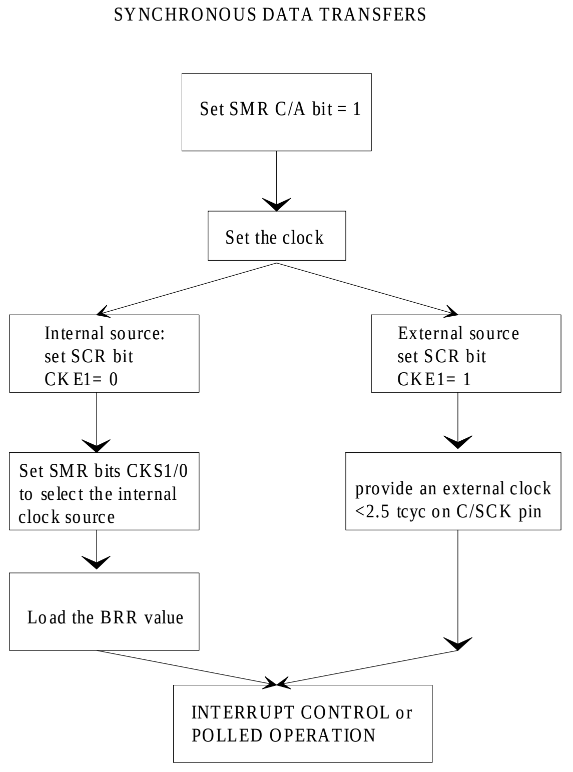
SYNCHRONOUS DATA TRANSFERS
Set SMR C/A bit = 1
Set the clock
Internal source: set SCR bit CKE1= 0
External source set SCR bit CKE1= 1
Set SMR bits CKS1/0 to select the internal clock source
provide an external clock <2.5 tcyc on C/SCK pin
Load the BRR value
INTERRUPT CONTROL or POLLED OPERATION

Figure 7a

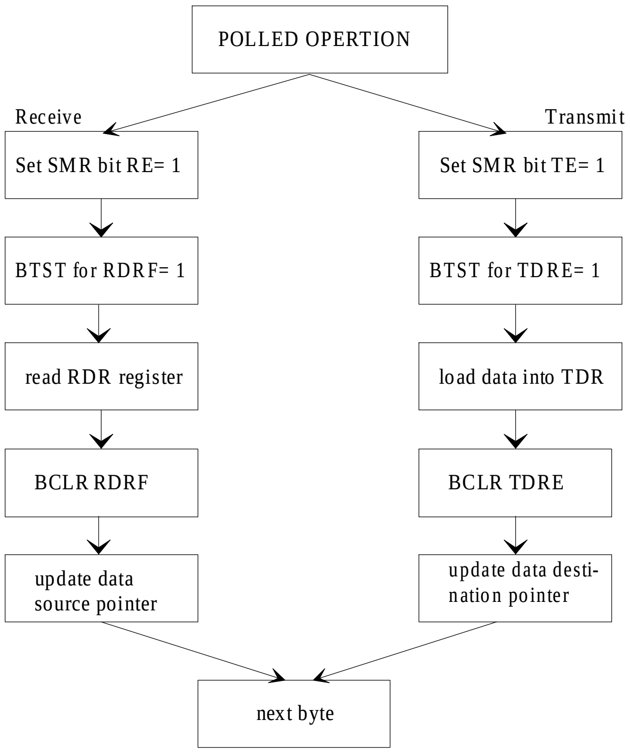

Figure 7b

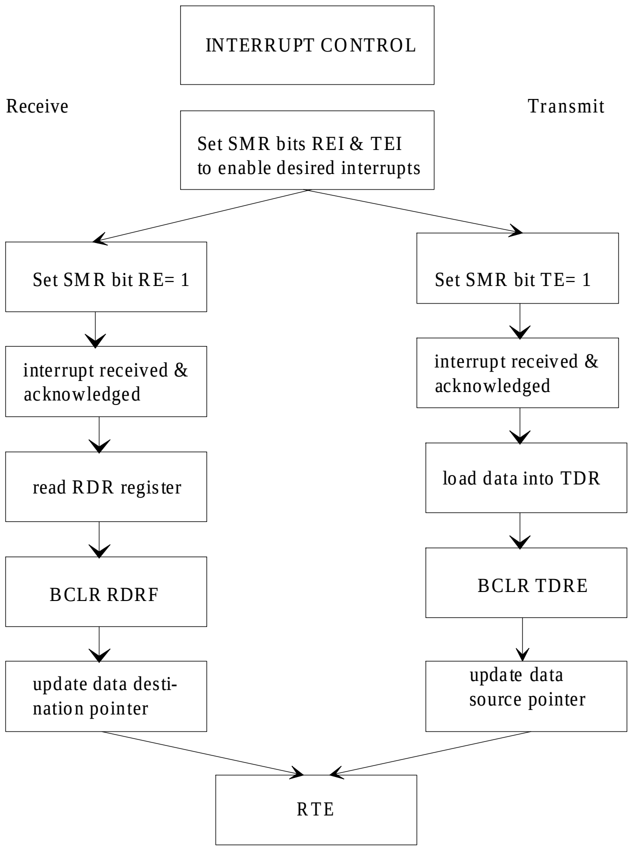
INTERRUPT CONTROL
Receive
Transmit
Set SMR bits REI & TEI to enable desired interrupts
Set SMR bit RE= 1
interrupt received & acknowledged
read RDR register
BCLR RDRF
update data destination pointer
Set SMR bit TE= 1
interrupt received & acknowledged
load data into TDR
BCLR TDRE
update data source pointer
RTE

Figure 7c

The information contained in this document has been carefully checked, however the contents of this document may be changed and modified without notice. Hitachi America, Ltd. shall assume no responsibility for inaccuracies, or any problem involving patent infringement caused when applying the descriptions in this document. This material is protected by copyright laws. Hitachi America, Ltd. reserves all rights.