Pulse Accumulator on the HC12

- To use the pulse accumulator connect an input to Port T7
- The pulse accumulator operates in two modes:
  1. Event-Count Mode
  2. Gated Time Accumulation Mode
- In Event-Count Mode, the pulse accumulator counts the number of rising or falling edges on Port T7
  - You can set up the pulse accumulator to select which edge to count
  - The counts are held in the 16-bit PACNT register
  - On each selected edge the PAIF flag of the PAFLG register is set
  - When PACNT overflows from 0xFFFF to 0x0000, the PAOVF flag of the PAFLG register is set
- In Gated Time Accumulation Mode the pulse accumulator counts clock cycles while in the input to Port T7 is high or low
  - In Gated Time Accumulation Mode the pulse accumulator uses the Timer Clock. To use the pulse accumulator in Gated Time Accumulation Mode you must enable the Timer Clock by writing a 1 to the TEN bit of TSCR
  - You can set up the pulse accumulator to count while PT7 is high or to count while PT7 is low
  - The clock for the pulse accumulator is the E-clock divided by 64
  - With an 8 MHz E-clock the clock frequency of the pulse accumulator is 125 kHz, for a period of 8 μs
  - For example, if the pulse accumulator is set up to count while Port T7 is high, and it counts 729 clock pulses, then the input to Port T7 was high for 729 x 8 μs = 5.832 ms

The Pulse Accumulator

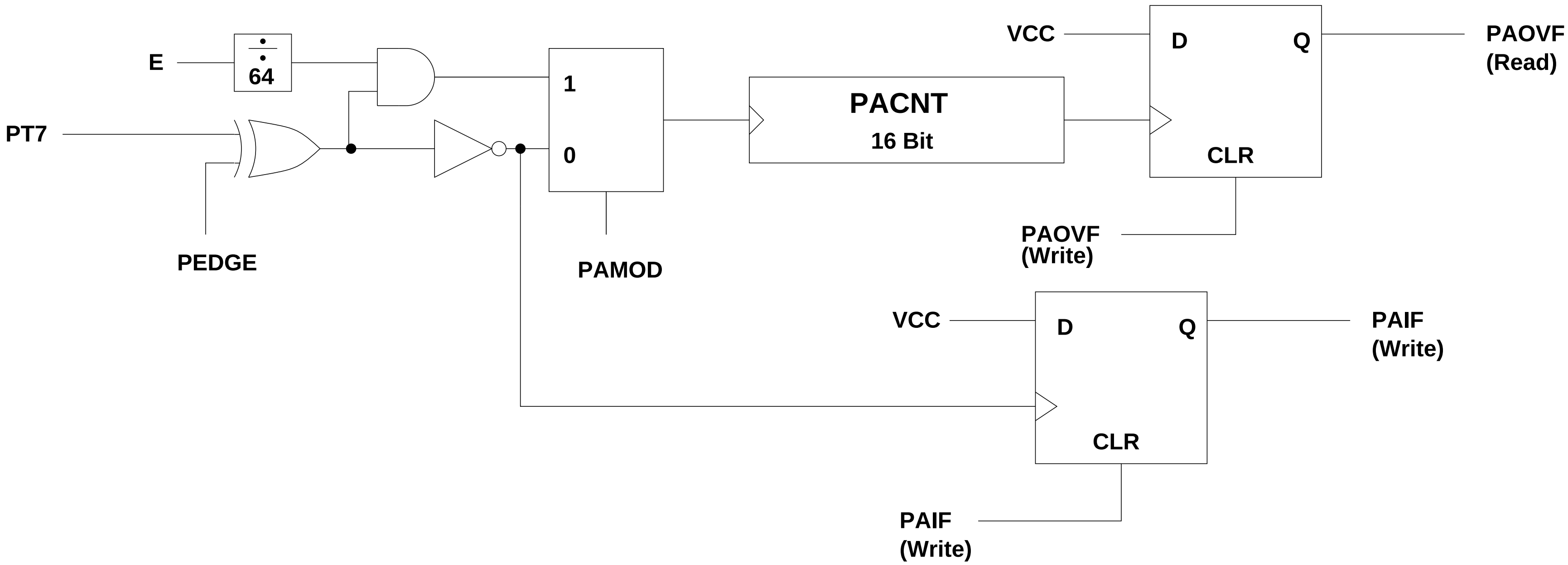
- The pulse accumulator uses PT7 as an input
  - To use the pulse accumulator make sure bit 8 of TIOS is 0 (otherwise PT7 used as output compare pin)
  - To use the pulse accumulator make sure bits 7 and 8 of TCTL1 are 0 (otherwise timer function connected to PT7)
- The pulse accumulator uses three registers: PACTL, PAFLG, PACNT
- To use the pulse accumulator you have to program the PACTL register
- The PAFLG register has flags to indicate the status of the pulse accumulator
  - You clear a flag bit by writing a 1 to that bit
- The count value is stored in the 16-bit PACNT register
  - You may write a value to PACNT
  - Suppose you want an interrupt after 100 events on PT7
  - Write -100 to PACNT, and enable the PAOVI interrupt
  - After 100 events on PT7, PACNT will overflow, and a PAOVI interrupt will be generated

0	**PAEN**	**PAMOD**	**PEDGE**	CLK1	CLK0	**PAOVI**	**PAI**	**PACTL**	**0x00A0**
0	0	0	0	0	0	**PAOVF**	**PAIF**	**PAFLG**	**0x00A1**

```
PAEN:   1 => Enable PA

PAMOD:  0 => Event Count Mode
        1 => Gated Time Accumulator Mode

PEDGE:  0 => Falling Edge (Event)     High Enable (Gated)
        1 => Rising Edge (Event)      Low Enable (Gated)

PAOVI:  1 => Enable Interrupt when PACNT overflows

PAI:    1 => Enable Interrupt when edge on PT7
             If PEDGE == 0, interrupt on falling edge
             If PEDGE == 1, interrupt on rising edge


The 16-bit PACNT register is at address 0x00A2
```

The Pulse Accumulator

PULSE ACCUMULATOR LOGIC



PAMOD	PAEDGE	ACTION
0	0	Increment PACNT on falling edge of PAI
0	1	Increment PACNT on rising edge of PAI
1	0	Count E/64 if PT7 = 1
1	1	Count E/64 if PT7 = 0

The Pulse Accumulator

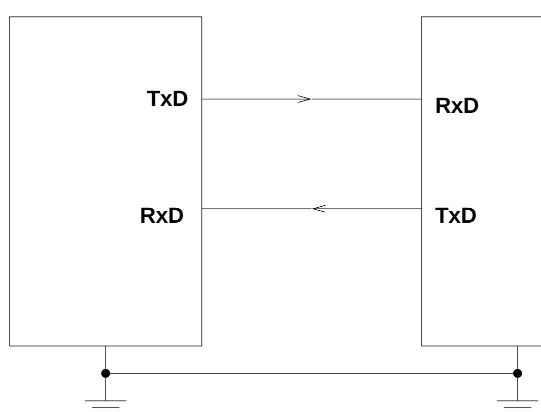
- Here is a C program which counts the number of rising edges on PT7:

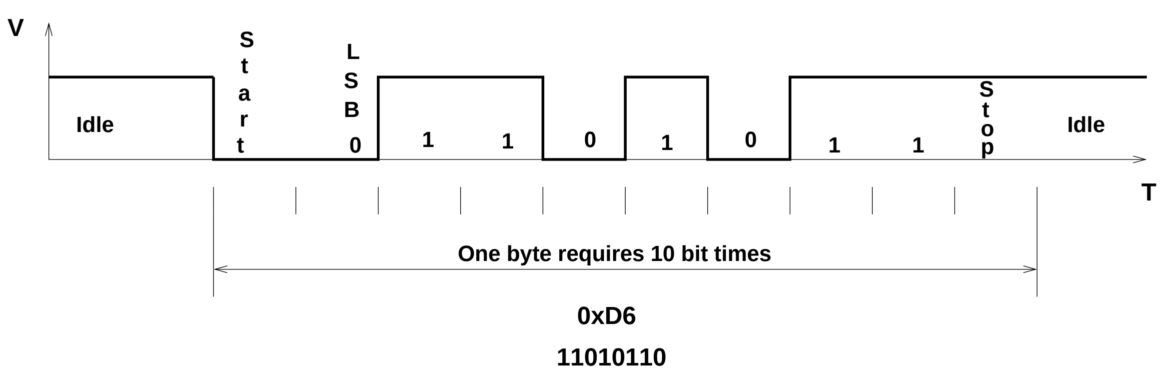
```
#include "hc12b32.h"
#include "DBug12.h"

int start_count,end_count,total_count;

main()
{
    int i;

    TIOS = TIOS & ~0x80;   /* PT7 input */
    TCTL1 = TCTL1 & ~0xC0  /* Disconnect IC/OC logic from PT7 */

    PACTL = 0x50;  /* 0 1 0 1 0 0 0 0                               */
                   /*   | | |     | |                               */
                   /*   | | |     | \_ No interurrupt on edge       */
                   /*   | | |     \___ No interurrupt on overflow   */
                   /*   | | \_________ Rising Edge                  */
                   /*   | \___________ Event Count Mode             */
                   /*   \_____________ Enable PA                    */

    start_count = PACNT;
    for (i=0;i<10000;i++) ;   /* Software Delay */
    end_count = PACNT;
    total_count = end_count - start_count;
    DBug12FNP->printf("Total counts = %d\r\n",total_count);
}
```

Asynchronous Data Transfer

- In asynchronous data transfer, there is no clock line between the two devices
- Both devices use internal clocks with the same frequency
- Both devices agree on how many data bits are in one data transfer (usually 8, sometimes 9)
- A device sends data over an TxD line, and receives data over an RxD line
  - The transmitting device transmits a special bit (the start bit) to indicate the start of a transfer
  - The transmitting device sends the requisite number of data bits
  - The transmitting device ends the data transfer with a specical bit (the stop bit)
- The start bit and the stop bit are used to synchronize the data transfer

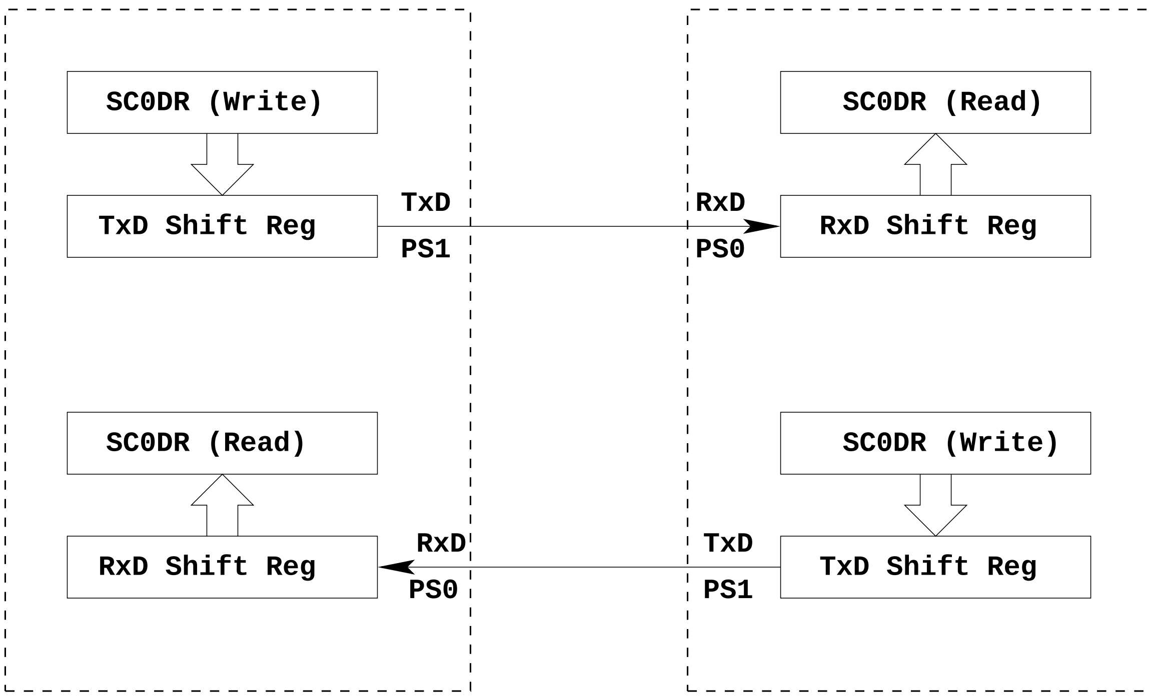
Asynchronous Serial Communications





Asynchronous Data Transfer

- The HC12 has an asynchronous serial interface, called the SCI (Serial Communications Interface)
- The SCI is used by D-Bug12 to communicate with the host PC
- When using D-Bug12 you normally cannot independently operate the SCI (or you will lose your communications link with the host PC)
- The D-Bug12 `printf()` function sends data to the host PC over the SCI
- The SCI TxD pin is bit 1 of Port S
- The SCI RxD pin is bit 0 of Port S
- In asynchronous data transfer, serial data is transmitted by shifting out of a transmit shift register into a receive shift register



SC0DR receive and transmit registers are separate registers.

Overrun error if RxD shift register filled before SC0DR read

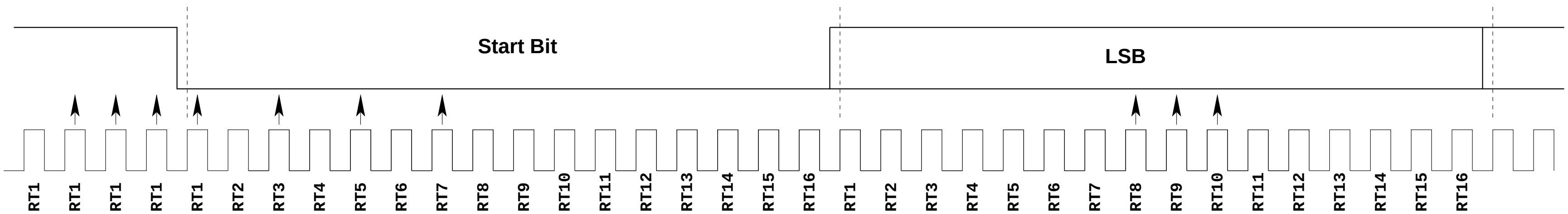
Timing in Asynchronous Data Transfers

- The BAUD rate is the number of bits per second
- Typical baud rates are 1200, 2400, 4800, 9600, 19,200, and 115,000
- At 9600 baud the transfer rate is 9600 bits per second, or one bit in 104 μs
- When not transmitting the TxD line is held high
- When starting a transfer the trasmitting device sends a start bit by bringing TxD low for one bit perios (104 μs at 9600 baud)
- The receiver knows the transmission is starting when it sees RxD go low
- After the start bit, the trasmitter send the requisite number of data bytes
- The receiver checks the data three for each bit. If the data within a bit is different, there is an error. This is called a noise error
- The transmitter ends the transmission with a stop bit, which is a high level on TxD for one bit period
- The reciever checks to make sure that a stop bit is received at the proper time
- If the receiver sees a start bit, but fails to see a stop bit, there is an error. Most likely the two clocks are running at different frequencies (generally because they are using different baud rates). This is called a framing error
- The transmitter clock and receiver clock will not have exactly the same frequency
- The transmission will work as long as the frequencies differ by less 4.5%(4% for 9-bit data)

Timing in Asynchronous Data Transfers

ASYNCHRONOUS SERIAL COMMUNIATIONS

Baud Clock = 16 x Baud Rate



Start Bit − Three 1's followed by 0's at RT1,3,5,7
(Two of RT3,5,7 must be zero −
If not all zero, Noise Flag set)

Data Bit − Check at RT8,9,10
(Majority decides value)
(If not all same, noise flag set)

If no stop bit detected, Framing Error Flag set

Baud clocks can differ by 4.5% (4% for 9 data bits) with no errors.

Even parity −− the number of ones in data word is even

Odd parity −− the number of ones in data word is odd

When using parity, transmit 7 data + 1 parity, or 8 data + 1 parity

SCI Registers

- The SCI uses 8 registers of the HC12
- Two registers are used to set the baud rate (`SC0BDH` and `SC0BDL`)
- One of the two control registers (`SC0CR1`) is used under normal operation
- (`SC0CR0` is used for special operation)
- One of the two status registers (`SC0SR0`) is used under normal operation
- (`SC0SR1` is used for special operation)
- The transmitter and receiver can be separately enabled in `SC0CR1`.
- Transmitter and receiver interrupts can be separately enabled in `SC0CR1`.
- `SC0SR0` is used to tell when a transmission is complete, and if any error was generated
- Data to be transmitted is sent to `SC0DRL`
- After data is received it can be read in `SC0DRL`

BTST	BSPL	BRLD	**SBR12**	**SBR11**	**SBR10**	**SBR9**	**SBR8**	**SC0BDH − 0x00C0**

SBR7	**SBR6**	**SBR5**	**SBR4**	**SBR3**	**SBR2**	**SBR1**	**SBR0**	**SC0BDL − 0x00C1**

LOOPS	WOMS	RSRC	**M**	**WAKE**	ILT	**PE**	**PT**	**SC0CR1 − 0x00C2**

TIE	TCIE	**RIE**	ILIE	**TE**	**RE**	**RWU**	SBK	**SC0CR2 − 0x00C3**

TDRE	TC	**RDRF**	IDLE	**OR**	**NF**	**FE**	**PE**	**SC0SR1 − 0x00C4**

0	**0**	**0**	**0**	**0**	**0**	**0**	RAF	**SC0SR2 − 0x00C5**

R8	**T8**	**0**	**0**	**0**	**0**	**0**	**0**	**SC0DRH − 0x00C5**

R7/T7	**R6/T6**	**R5/T5**	**R4/T4**	**R3/T3**	**R2/T2**	**R1/T1**	**R0/T0**	**SC0DRL − 0x00C7**

Example program using the SCI

```
#include <hc12b32.h>
/* Program to transmit data over SCI port */

main()
{
    /****************************************************************
     * SCI Setup
     ****************************************************************/
    SC0BDL = 0x34;    /* Set BAUD rate to 9,600 */
    SC0BDH = 0x00;

    SC0CR1 = 0x00;  /* 0 0 0 0 0 0 0 0
                       | | | | | | | |
                       | | | | | | | \____ Even Parity
                       | | | | | | \______ Parity Disabled
                       | | | | | \________ Short IDLE line mode (not used)
                       | | | | \__________ Wakeup by IDLE line rec (not used)
                       | | | \____________ 8 data bits
                       | | \______________ Not used (loopback disabled)
                       | \________________ Not used (loopback disabled)
                       \__________________ Normal (not loopback) mode
                    */


    SC0CR2 = 0x08;  /* 0 0 0 0 1 0 0 0
                       | | | | | | | |
                       | | | | | | | \____ No Break
                       | | | | | | \______ Not in wakeup mode (always awake)
                       | | | | | \________ Reciever disabled
                       | | | | \__________ Transmitter enabled
                       | | | \____________ No IDLE Interrupt
                       | | \______________ No Reciever Interrupt
                       | \________________ No Tranmit Complete Interrupt
                       \__________________ No Tranmit Ready Interrupt
                    */
    /****************************************************************
     * End of SCI Setup
     ****************************************************************/

    SC0DRL = 'h';   /* Send first byte */
    while ((SC0SR1 & 0x80) == 0) ;   /* Wait for TDRE flag */
    SC0DRL = 'e';   /* Send next byte */
    while ((SC0SR1 & 0x80) == 0) ;   /* Wait for TDRE flag */
```

```
    SC0DRL = 'l';   /* Send next byte */
    while ((SC0SR1 & 0x80) == 0) ;   /* Wait for TDRE flag */
    SC0DRL = 'l';   /* Send next byte */
    while ((SC0SR1 & 0x80) == 0) ;   /* Wait for TDRE flag */
    SC0DRL = 'o';   /* Send next byte */
    while ((SC0SR1 & 0x80) == 0) ;   /* Wait for TDRE flag */

}
```