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 $\mu$s
  - 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 $729 \times 8 \mu s = 5.832$ 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

<table>
<thead>
<tr>
<th>0</th>
<th>PAEN</th>
<th>PAMOD</th>
<th>PEDGE</th>
<th>CLK1</th>
<th>CLK0</th>
<th>PAOVI</th>
<th>PAI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PACTL 0x00A0

| 0 | 0 | 0 | 0 | 0 | 0 | PAOF | 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

<table>
<thead>
<tr>
<th>PAMOD</th>
<th>PAEDGE</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Increment PACNT on falling edge of PAI</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Increment PACNT on rising edge of PAI</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Count E/64 if PT7 = 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Count E/64 if PT7 = 0</td>
</tr>
</tbody>
</table>
The Pulse Accumulator

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

```c
#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;
    DBG12FNP->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 special bit (the stop bit).
- The start bit and the stop bit are used to synchronize the data transfer.

Asynchronous Serial Communications

One byte requires 10 bit times

0xD6
11010110
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

\[\text{SC0DR (Write)}\quad \text{SC0DR (Read)}\]

\[
\begin{align*}
\text{SC0DR (Write)} & \quad \text{SC0DR (Read)} \\
\text{TxD Shift Reg} & \quad \text{RxD Shift Reg} \\
\text{PS1} & \quad \text{PS0} \\
\text{RxD Shift Reg} & \quad \text{TxD Shift Reg} \\
\text{PS0} & \quad \text{PS1} \\
\end{align*}
\]

\[\text{SC0DR receive and transmit registers are separate registers.}\]

\[\text{Overrun error if RxD shift register filled before SC0DR read}\]
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 $\mu$s
- When not transmitting the TxD line is held high
- When starting a transfer the transmitting device sends a start bit by bringing TxD low for one bit period (104 $\mu$s at 9600 baud)
- The receiver knows the transmission is starting when it sees RxD go low
- After the start bit, the transmitter sends 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 receiver 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 COMMUNICATIONS

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)

If no stop bit detected, Framing Error Flag set

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

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
<table>
<thead>
<tr>
<th>BTST</th>
<th>BSPL</th>
<th>BRLD</th>
<th>SBR12</th>
<th>SBR11</th>
<th>SBR10</th>
<th>SBR9</th>
<th>SBR8</th>
<th>SC0BDH - 0x00C0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBR7</td>
<td>SBR6</td>
<td>SBR5</td>
<td>SBR4</td>
<td>SBR3</td>
<td>SBR2</td>
<td>SBR1</td>
<td>SBR0</td>
<td>SC0BDL - 0x00C1</td>
</tr>
<tr>
<td>LOOPS</td>
<td>WOMS</td>
<td>RSRC</td>
<td>M</td>
<td>WAKE</td>
<td>ILT</td>
<td>PE</td>
<td>PT</td>
<td>SC0CR1 - 0x00C2</td>
</tr>
<tr>
<td>TIE</td>
<td>TCIE</td>
<td>RIE</td>
<td>ILIE</td>
<td>TE</td>
<td>RE</td>
<td>RWU</td>
<td>SBK</td>
<td>SC0CR2 - 0x00C3</td>
</tr>
<tr>
<td>TDRE</td>
<td>TC</td>
<td>RDRF</td>
<td>IDLE</td>
<td>OR</td>
<td>NF</td>
<td>FE</td>
<td>PE</td>
<td>SC0SR1 - 0x00C4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>RAF</td>
<td>SC0SR2 - 0x00C5</td>
</tr>
<tr>
<td>R8</td>
<td>T8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>SC0DRH - 0x00C5</td>
</tr>
<tr>
<td>R7/T7</td>
<td>R6/T6</td>
<td>R5/T5</td>
<td>R4/T4</td>
<td>R3/T3</td>
<td>R2/T2</td>
<td>R1/T1</td>
<td>R0/T0</td>
<td>SC0DRL - 0x00C7</td>
</tr>
</tbody>
</table>
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)
    Receiver disabled
    Transmitter enabled
    No IDLE Interrupt
    No Receiver Interrupt
    No Transmit Complete Interrupt
    No Transmit 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 */