Pulse Accumulator on the HCS12

- A pulse accumulator counts the number of active edges at the input of its channel.
- The HCS12 has four 8-bit pulse accumulators, configurable as two 16-bit pulse accumulators. In the following we will discuss the 16-bit pulse accumulator A, PACA, made up of the two 8-bit pulse accumulators PACN3 and PACN2.
- To use the pulse accumulator connect an input to Port T7 (PT7).
- 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 PACA register
  - On each selected edge the PAIF flag of the PAFLG register is set
  - When PACA 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 bus clock divided by 64
  - With an 24 MHz bus clock, the clock frequency of the pulse accumulator is 375 kHz, for a period of 2.67 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 2.67 μs = 1.94 ms

The Pulse Accumulator

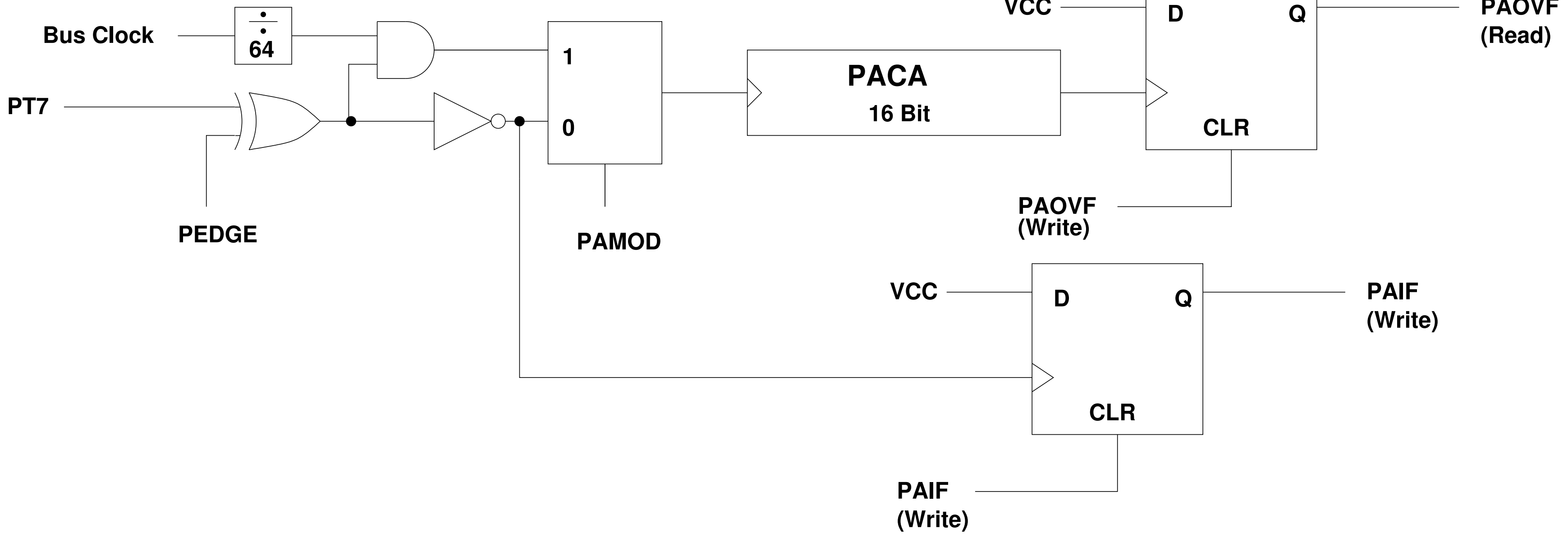
- The pulse accumulator uses PT7 as an input
  - To use the pulse accumulator make sure bit 7 of TIOS is 0 (otherwise PT7 used as output compare pin)
  - To use the pulse accumulator make sure bits 6 and 7 of TCTL3 are 0 (otherwise timer function connected to PT7)
- The pulse accumulator uses three registers: PACTL, PAFLG, PACA
- 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 PACA register
  - You may write a value to PACA
  - Suppose you want an interrupt after 100 events on PT7
  - Write -100 to PACA, and enable the PAOVI interrupt
  - After 100 events on PT7, PACA will overflow, and a PAOVI interrupt will be generated

0	**PAEN**	**PAMOD**	**PEDGE**	CLK1	CLK0	**PAOVI**	**PAI**	**PACTL**	**0x0060**
0	0	0	0	0	0	**PAOVF**	**PAIF**	**PAFLG**	**0x0061**

```
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 PACA 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 PACA register is at address 0x0062
```

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 PACA

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

```
#include "hcs12.h"
#include "DBug12.h"

#define PACA *(unsigned int *) 0x62;   /* pulse accumulator A3 count */

int start_count,end_count,total_count;

main()
{
    int i;

    TIOS = TIOS & ~0x80;   /* PT7 input */
    TCTL3 = TCTL3 & ~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 PACA (16 bit mode)   */

    start_count = PACA;
    for (i=0;i<10000;i++) ;   /* Software Delay */
    end_count = PACA;
    total_count = end_count - start_count;
    DB12FNP->printf("Total counts = %d\r\n",total_count);
}
```

The Pulse Accumulator PACA

- Here is a C program which determines how long the input on PA7 is high:

```
#include "hcs12.h"
#include "DBug12.h"

#define PACA *(unsigned int *) 0x62;   /* pulse accumulator A3 count */

int start_count,end_count,total_count;

main()
{
    int i;

    TSCR = 0x80;           /* Turn on timer clock */
    TIOS = TIOS & ~0x80;   /* PT7 input */
    TCTL3 = TCTL3 & ~0xC0  /* Disconnect IC/OC logic from PT7 */

    PACTL = 0x60;  /* 0 1 1 0 0 0 0 0                           */
                   /*   | | |     | |                           */
                   /*   | | |     | \_ No interurrupt on edge       */
                   /*   | | |     \___ No interurrupt on overflow   */
                   /*   | | \_________ Count while input high       */
                   /*   | \___________ Gated Counter Mode           */
                   /*   \_____________ Enable PACA (16 bit mode)    */

    start_count = PACA;
    while ((PTT & 0x80) == 0) ;  /* Wait until input goes high */
    while ((PTT & 0x80) == 0x80) ;  /* Wait until input goes low */
    end_count = PACA;
    total_count = end_count - start_count;
    DB12FNP->printf("Total clock cycles = %d\r\n",total_count);
}
```

Using Floating Point Numbers the the Gnu C Compiler

For the final lab, you need a control loop which looks like this:

```
void INTERRUPT rti_isr(void)
{
    Code to read potentiometer voltage and convert into RPM

    Code to measure speed Sm in RPM

    Code which sets duty cycle to

        DC = DC + k*(Sd-Sm)
        if (DC > 1.0) DC = 1.0;
        if (DC < 0.2) DC = 0.2;

    Code which writes the PWM Duty Cycle Register
    to generate duty cycle DC.

    Code which clears RTI flag
}
```

The desired speed is given by:

$$S_d = \left(0.2 + 0.8\frac{AD}{AD_{max}}\right) S_{max}$$

Suppose your code to read the potentiometer looks something like this:

```
speed_desired = (0.2 + 0.8*(ATD1DR0/255))*speed_max;
```

What will `speed_desired` be if the potentiometer is set to the half-way point? It will be 0, because `ATD1DR0` will be 128, and $128/255 = 0$ when done as fixed-point arithmetic.

How can you fix this? First, be sure to declare `speed_desired` and `speed_max` to be floating point:

```
volatile float speed_desired, speed_max;
```

Then write the equation in this way:

```
    speed_desired = (0.2 + 0.8*(ATD1DR0/255.0))*speed_max;
```

When the C compiler sees `ATD1DR0/255.0`, it recognizes that 255.0 is meant to be a floating point number. It cannot divide the fixed-point number `ATD1DR0` by the floating point number 255.0, so it will convert the fixed-point number to a floating-point number before it does the calculation. When doing arithmetic with mixed types, C automatically converts numbers to the type with the most precision. If doing arithmetic with `short` and `char`, it will convert the `char` to a `short` before doing the arithmetic.

The original equation had the floating point numbers 0.2 and 0.8 in it. However, C does calculations inside parentheses first, so it does the calculation `ATD1DR0/255.0` (which results in a zero if ATD1DR0 is less than 255). It then converts the fixed-point 0 to a floating-point 0.0, multiplies that by 0.8 (resulting in 0.0), adds 0.2 (resulting in 0.2) and finally multiplies by `speed_max`. Thus, `speed_desired` will be 20% of `speed_max` when the pot is less than 5 V, or `speed_max` when the pot is set at five volts.

The better way to write the equation is:

```
    speed_desired = (0.2 + 0.8*((float) ATD1DR0/255.0))*speed_max;
```

This explicitly tells the C compiler to convert `ATD1DR0` to a floating point before doing the division.

When you print out the desired speed, do this:

```
    DB12FNP->printf("Desired Speed:  %4d\r\n",(short) speed_desired);
```