MC9S12 Built-In Hardware

• Here is some of the hardware available on the MC9S12DP256:

– **General Purpose Input/Output (GPIO) Pins**: These pins can be used to read the logic level on a MC9S12 pin (input) or write a logic level to an MC9S12 pin (output).

- **Timer-Counter Pins**: The MC9S12 is often used to time or count events.

– **Pulse Width Modulation (PWM) Pins**: To make a motor turn at a particular speed you need to send it a PWM signal.

– **Serial Interfaces**: Used to talk to other digital devices (such as another computer) over a serial interface. The MC9S12 has two serial interfaces:

*An asynchronous serial interface (Serial Communications Interface, or SCI)

* A synchronous serial interface (Serial Peripheral Interface, or SPI).

– **Analog-to-Digital Converter (ADC)**: Useful to convert a voltage to a digital number for use by the MC9S12.

Timer inside the MC9S12:

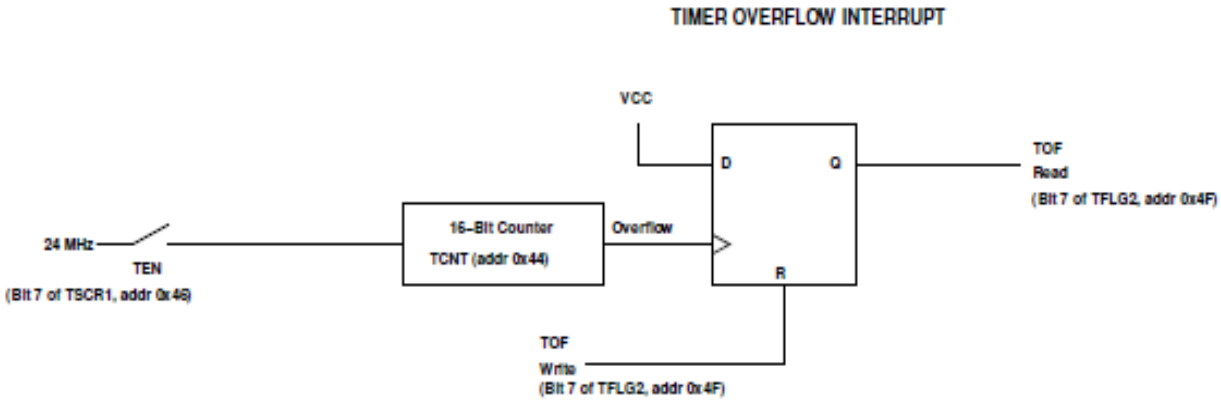
• When you enable the timer (by writing a 1 to bit 7 of TSCR1), you connect a 24−MHz oscillator to a 16−bit counter.

You can read the counter at address **TCNT**.

• To put in a delay of 2.7307 ms, you could wait from one reading of 0x0000 to the next reading of 0x0000.

• **Problem:** You cannot read the TCNT register quickly enough to make sure you will see the 0x0000.

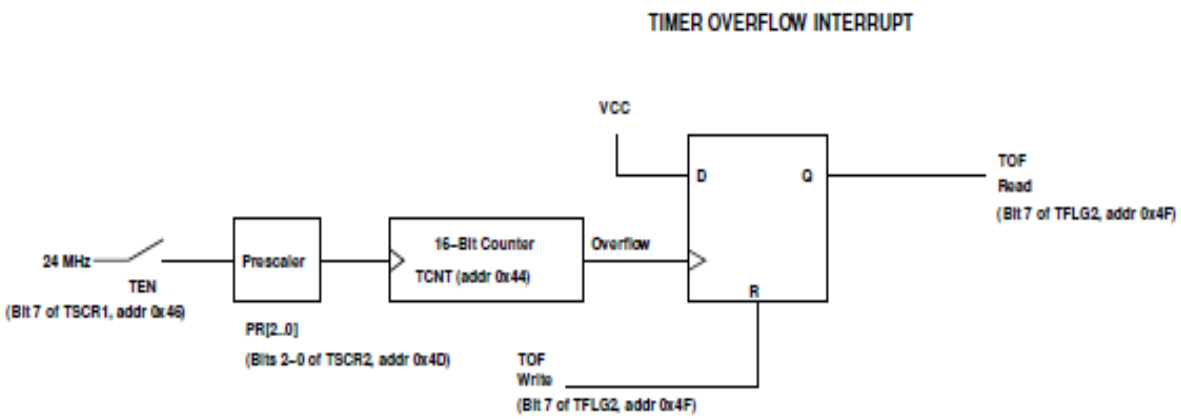
• **Solution:** The MC9S12 has built-in hardware which will set a flip-flop every time the counter rolls over from 0xFFFF to 0x0000.



• **Another problem:** Sometimes you may want to delay longer than 2.7307 ms, or time an event which takes longer than 2.7307 ms.

• **Solution:** The MC9S12 allows you to slow down the clock which drives the counter.

• You can **slow down the clock by dividing the 24 MHz clock** by 2, 4, 8, 16, 32, 64 or 128. You do this by writing to the prescaler bits (**PR2:0**) of the **Timer System Control Register 2** (**TSCR2**).



What Happens When You Reset the MC9S12?

• What happens to the MC9S12 when you turn on power or push the reset button?

• How does the MC9S12 know which instruction to execute first?

• On reset the MC9S12 loads the PC with the address located at address 0xFFFE and 0xFFFF.

Introduction to Interrupts

What happens when HCS12 gets an interrupt: HCS12 automatically jumps to part of the program which tells it what to do when it receives the interrupt (**Interrupt Service Routine**).

How does HCS12 know where the ISR is located: A set of memory locations called Interrupt Vectors tell the HCS12 the address of the ISR for each type of interrupt.

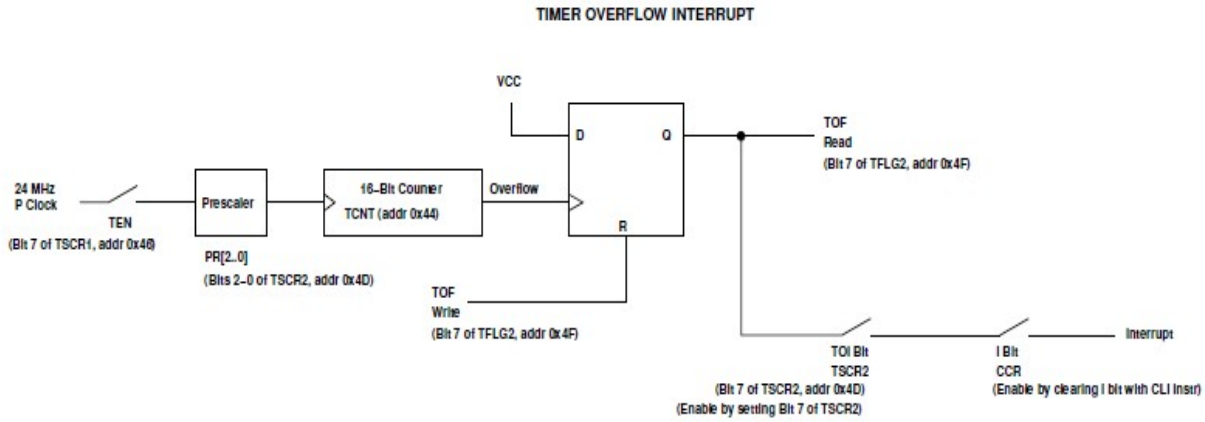
How does HCS12 know where to return to: Return address pushed onto stack before HCS12 jumps to ISR. You use the **RTI** (Return from Interrupt) instruction to pull the return address off of the stack when you exit the ISR.

What happens if ISR changes registers: All registers are pushed onto stack before jumping to ISR, and pulled off the stack before returning to program. When you execute the **RTI** instruction at the end of the ISR, the registers are pulled off of the stack.

What happens if you get an interrupt while in an ISR: MC9S12 disables interrupts (sets I bit of CCR) before it starts executing ISR.

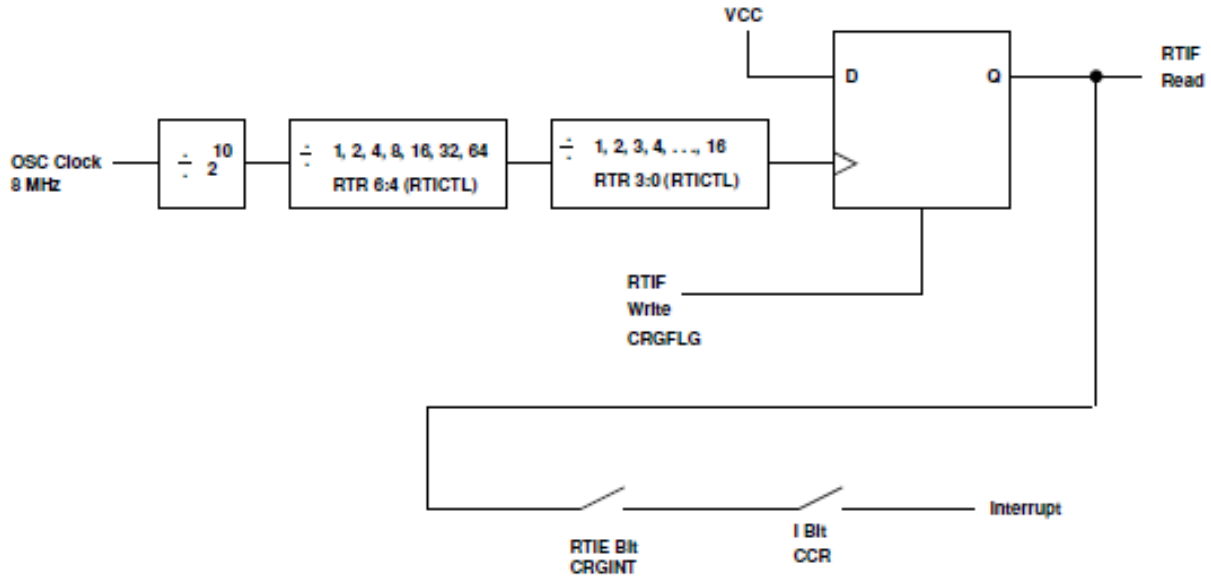
To Return from the ISR You must return from the ISR using the **RTI** instruction. The RTI instruction tells the HCS12 to pull all the registers off of the stack and return to the address where it was processing when the interrupt occurred.

How to generate an interrupt when the timer overflows



The Real Time Interrupt (RTI)

• Like the Timer Overflow Interrupt (TOF), the Real Time Interrupt (RTI) allows you to interrupt the processor at a regular interval.



• The specific interrupt mask for the Real Time Interrupt is the **RTIE bit** of the **CRGINT** register.

• When the Real Time Interrupt occurs, the **RTIF bit** of the **CRGFLG** register **is set**.

- To clear the Real Time Interrupt write a 1 to the RTIF bit of the CRGFLG register.

• The interrupt rate is set by the **RTR 6:4** and **RTR 3:0** bits of the **RTICTL** register. The RTR 6:4 bits are the Prescale Rate Select bits for the RTI, and the RTR 2:0 bits are the Modulus Counter Select bits to provide additional granularity.

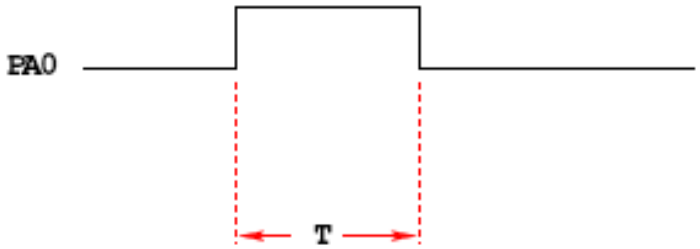
• To use the Real Time Interrupt, set the rate by writing to the RTR 6:4 and the RTR 3:0 bits of the RTICTL, and enable the interrupt by setting the RTIE bit of the CRGINT register

• The following table shows all possible values, in ms, selectable by the RTICTL register (assuming the system uses a 8 MHz oscillator):

RTR 3:0	RTR 6:4							
	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	Off	0.128	0.256	0.512	1.024	2.048	4.096	8.192
0001 (1)	Off	0.256	0.512	1.204	2.048	4.096	8.192	16.384
0010 (2)	Off	0.384	0.768	1.536	3.072	6.144	12.288	24.576
0011 (3)	Off	0.512	1.024	2.048	4.096	8.192	16.384	32.768
0100 (4)	Off	0.640	1.280	2.560	5.120	10.240	20.480	40.960
0101 (5)	Off	0.768	1.536	3.072	6.144	12.288	24.570	49.152
0110 (6)	Off	0.896	1.792	3.584	7.168	14.336	28.672	57.344
0111 (7)	Off	1.024	2.048	4.096	8.192	16.384	32.768	65.536
1000 (8)	Off	1.152	2.304	4.608	9.216	18.432	36.864	73.728
1001 (9)	Off	1.280	2.560	5.120	10.240	20.480	40.960	81.920
1010 (A)	Off	1.408	2.816	5.632	11.264	22.528	45.056	90.112
1011 (B)	Off	1.536	3.072	6.144	12.288	24.576	49.152	98.304
1100 (C)	Off	1.664	3.328	6.656	13.312	26.624	53.248	106.496
1101 (D)	Off	1.729	3.584	7.168	14.336	28.672	57.344	114.688
1110 (E)	Off	1.920	3.840	7.680	15.360	30.720	61.440	122.880
1111 (F)	Off	2.048	4.096	8.192	16.384	32.768	65.536	131.072

The MC9S12 Output Compare Function

Want event to happen at a certain time?
Want to produce pulse with width T?



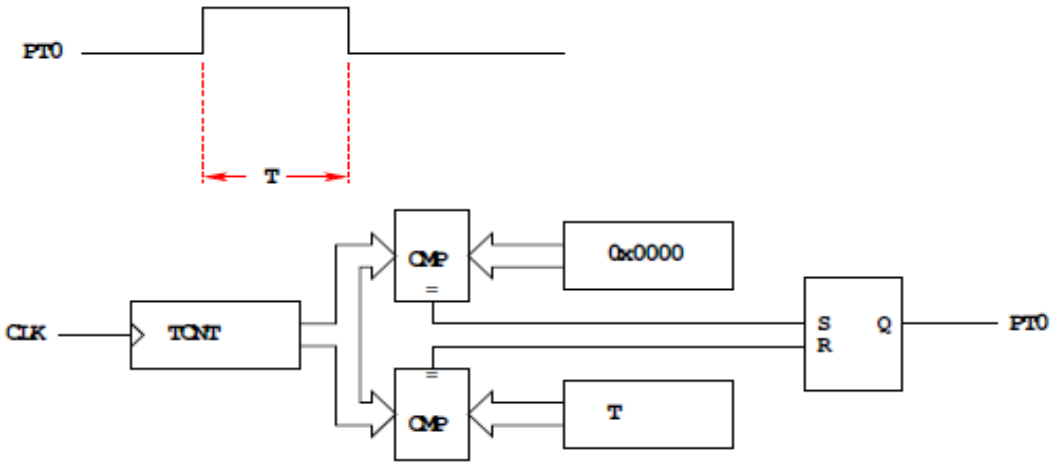
Wait until TCNT == 0x0000, then bring PA0 high
Wait until TCNT == T, then bring PA0 low

Problems:

1) May miss TCNT == 0x0000 or TCNT == T

2) Time not exact −− software delays

3) Cannot do anything else while waiting



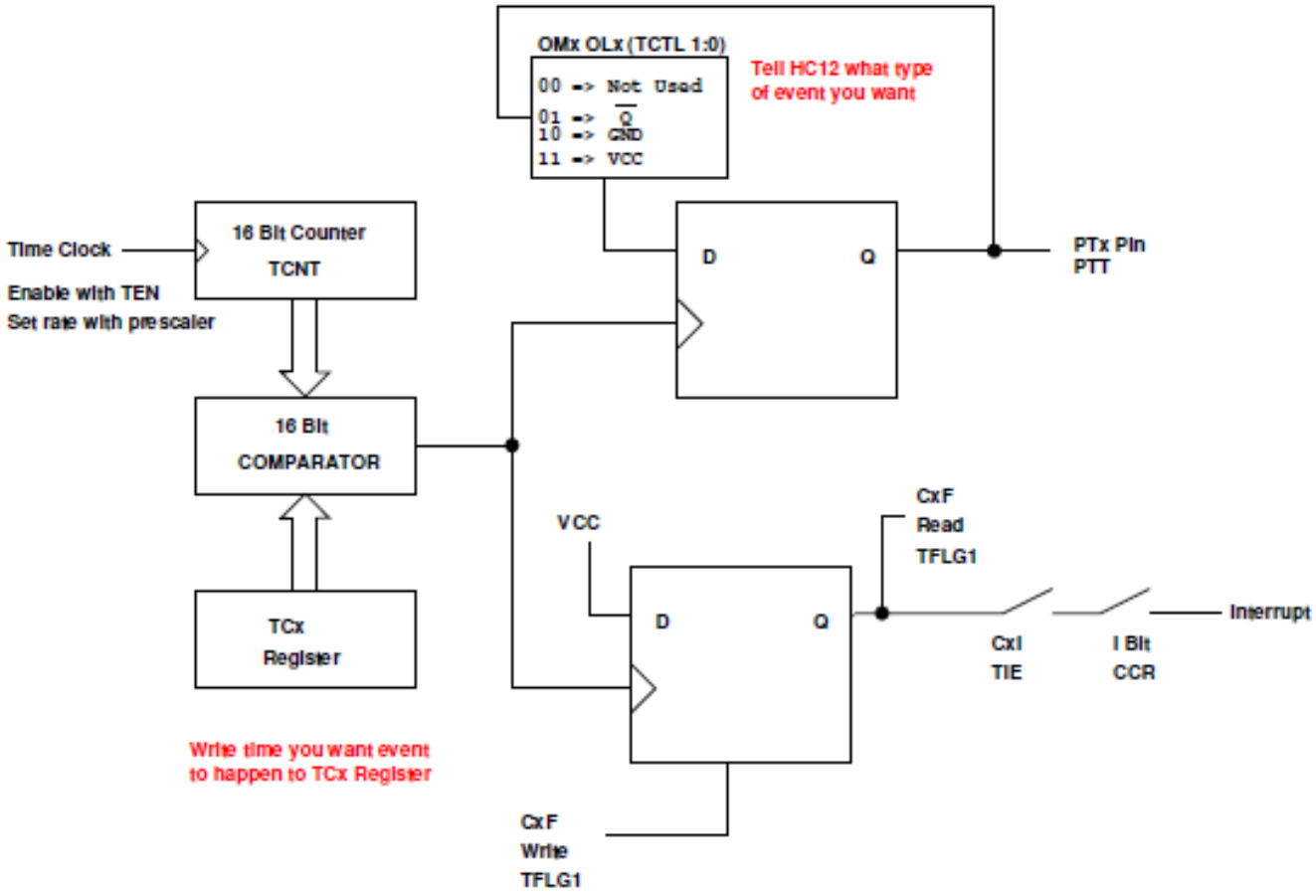
Wait until TCNT == 0x0000, then bring PA0 high

Wait until TCNT == T, then bring PA0 low

Now pulse is exactly T cycles long

Output Compare PORT T 0−7

To use Output Compare, you must set IOSx to 1 in TIOS



Using Output Compare on the MC9S12

1. In the main program:

   (a) Turn on timer subsystem (TSCR1 reg)
   (b) Set prescaler (TSCR2 reg)
   (c) Set up PTx as OC (TIOS reg)
   (d)Set action on compare (TCTL 1-2 regs, OMx OLx bits)

OMx	OLx	Action
0	0	Disconnected
0	1	Toggle
1	0	Clear
1	1	Set

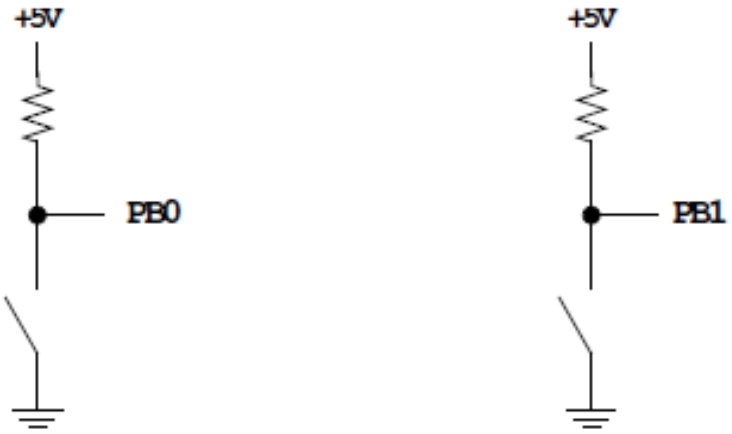
   (e) Clear Flag (TFLG1 reg)
   (f) To use interrupts: Enable int (TIE reg)

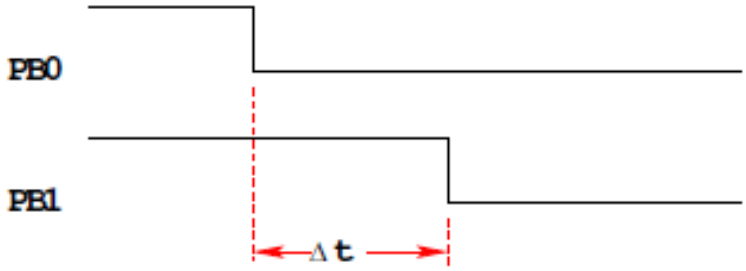
2. In interrupt service routine

   (a) Set time for next action to occur (write TCx reg)
   • For periodic events add time to TCx register
   (g) Clear flag (TFLG1 reg)

Capturing the Time of an External Event

Measure the time between two events





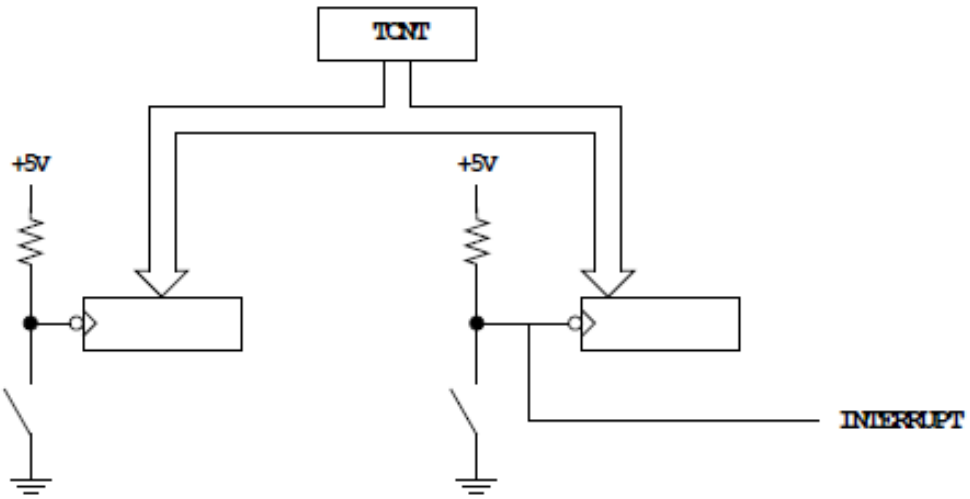
How to measure Δt?

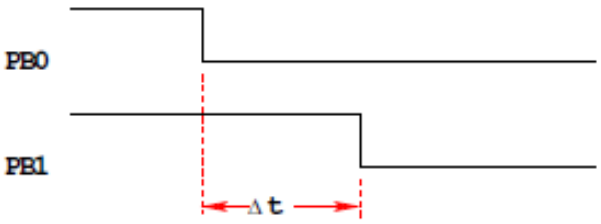
Wait until signal goes low, then measure TCNT

• Two problems with this:
1. Cannot do anything else while waiting
2. Do not get exact time because of delays in software

• To solve problems use hardware which latches TCNT when event occurs, and generates an interrupt.

Solution: Latch TCNT on falling edge of signal
Read latched values anytime later and get exact value
Can have MC9S12 generate interrupt when event occurs, so can do other things while waiting

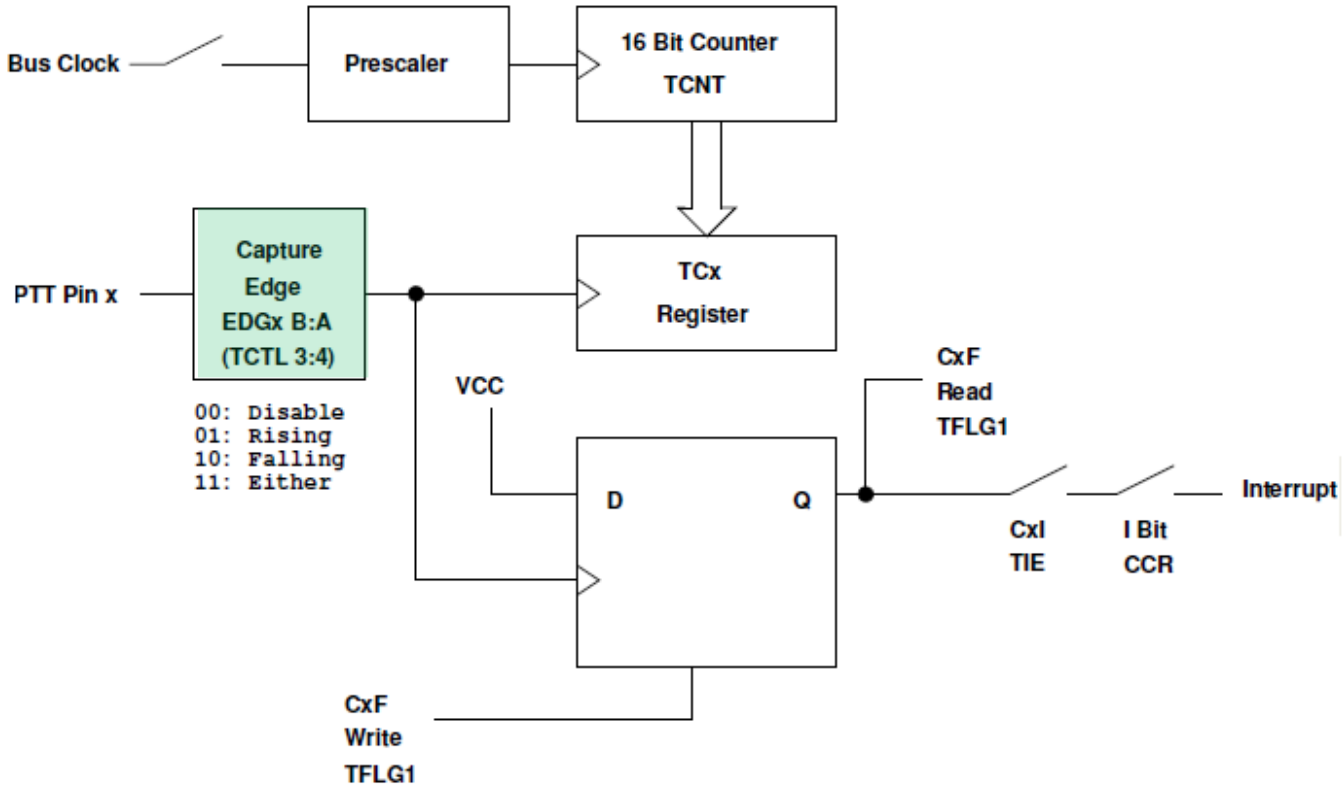




The MC9S12 Input Capture Function

Input Capture

Port T Pin x set up as Input Capture (IOSx = 0 in TIOS)



Using Input Capture on the MC9S12

To use Port T Pin x as an input capture pin:

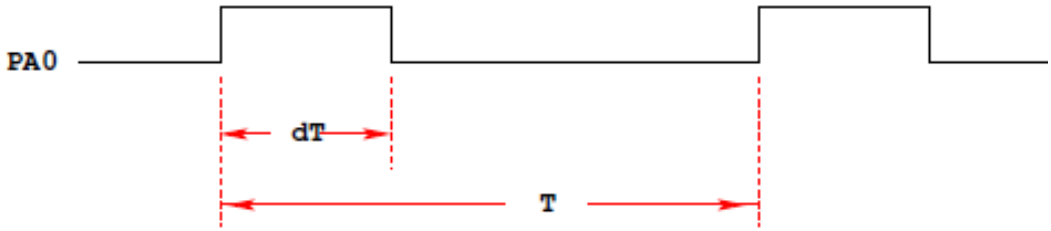
1. Turn on timer subsystem (1 -> Bit 7 of TSCR1 reg)
2. Set prescaler (TSCR2 reg). To get most accuracy set overflow rate as small as possible, but larger than the maximum time difference you need to measure.
3. Set up PTx as IC (0 -> bit x of TIOS reg)
4. Set edge to capture (EDGxB EDGxA of TCTL 3-4 regs)

EDGxB	EDGxA	
0	0	Disabled
0	1	Rising Edge
1	0	Falling Edge
1	1	Either Edge

5. Clear flag (1 -> bit x of TFLG1 reg, 0 -> all other bits of TFLG1)
6. If using interrupts
   (a) Enable interrupt on channel x (1 -> bit x of TIE reg)
   (b) Clear I bit of CCR (**cli** or **enable**())
   (c) In interrupt service routine,
      i. Read time of edge from TCx
      ii. Clear flag (1 -> bit x of TFLG1 reg, 0 -> all other bits of TFLG1)
7. If polling in main program
   (a) Wait for Bit x of TFLG1 to become set
   (b) Read time of edge from TCx
   (c) Clear flag (1 -> bit x of TFLG1 reg, 0 -> all other bits of TFLG1)

Using the MC9S12 PWM Function

Want a Pulse Width Modulated signal
Want to produce pulse with width dT, period T



• Because PWM is used so often the MC9S12 has a built-in PWM system

•The MC9S12 PWM does not use interrupts

• The PWM system on the MC9S12 is very flexible

• There are 33 registers used by the PWM subsystem

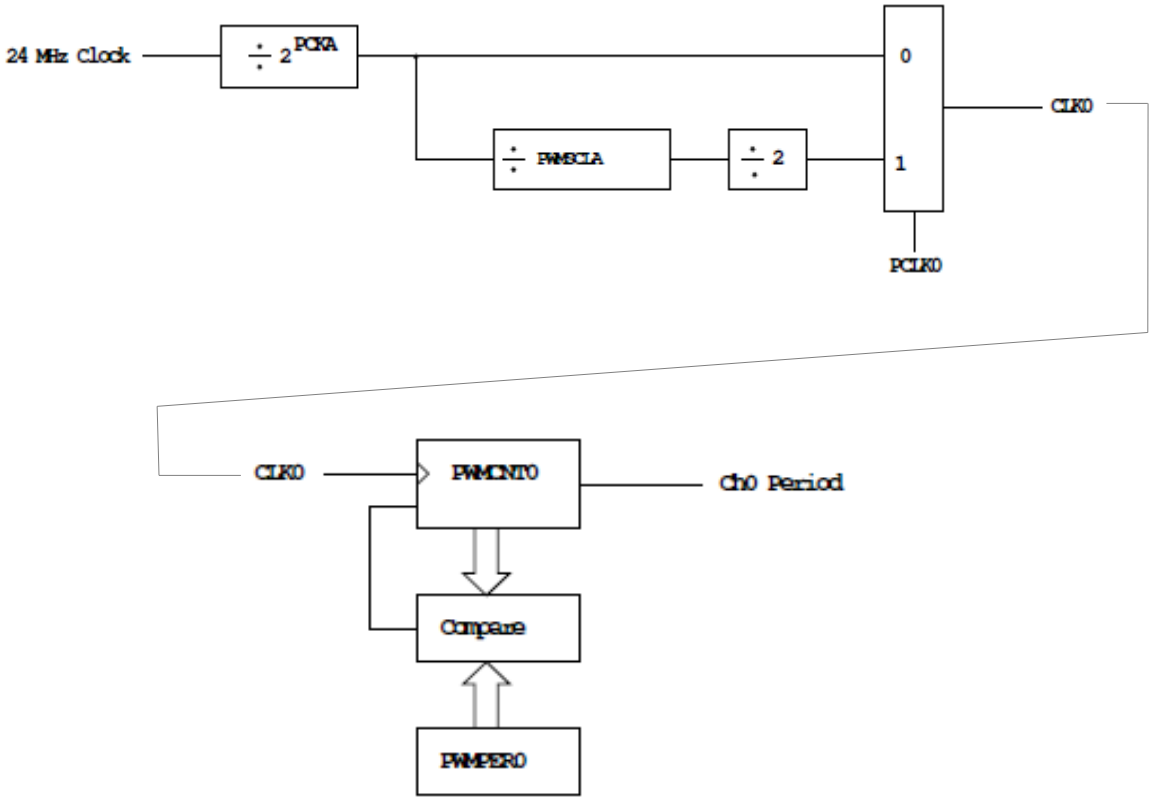
• You don't need to work with all 33 registers to activate PWM

• To select 8-bit mode, write a 0 to Bits 7, 6, 5 and 4 of PWMCTL register (no 16 concatenation).

• To select left-aligned mode, write 0x00 to PWMCAE.

• To select high polarity mode, write a 0xFF to PWMPOL register.

Clock Select for PWM Channel 0



Using the HCS12 PWM

1. Choose 8-bit mode (PWMCTL = 0x00)
2. Choose high polarity (PWMPOL = 0xFF)
3. Choose left-aligned (PWMCAE = 0x00)
4. Select clock mode in PWMCLK:
   - **PCLKn = 0 for 2^N**,
   - **PCLKn = 1 for $2^{(N+1)} \times M$**,
5. Select N in PWMPRCLK register:
   - **PCKA** for channels **5, 4, 1, 0**;
   - **PCKB** for channels **7, 6, 3, 2**.
6. If PCLKn = 1, select M
   - PWMSCLA = M for channels 5, 4, 1, 0
   - PWMSCLB = M for channels 7, 6, 3, 2.
7. Select PWMPERn, normally between 100 and 255.
8. Enable desired PWM channels: PWME.
9. Select PWMDTYn, normally between 0 and PWMPERn. Then
   Duty Cycle n = (PWMDTYn / PWMPERn) × 100%
   Change duty cycle to control speed of motor or intensity of light, etc.
10. For 0% duty cycle, choose PWMDTYn = 0x00.

MC9S12 Analog/Digital Converter

• The MC9S12 has two 10-bit A/D converters (ATD0 and ATD1).

• ATD0 uses the eight bits of Port AD0, called PAD00 through PAD07

• ATD1 uses the eight bits of Port AD1, called PAD08 through PAD15

• A 10-bit A/D converter is used to convert an input voltage. The reference voltages are V_{RL} = 0V and V_{RH} = 5V.

– What is the quantization level of the A/D converter?

$$\Delta V = (V_{RH} - V_{RL})/(2^b - 1) = 4.88 \text{ mV}$$

• If the value read from the A/D converter is 0x15A, what is the input voltage?

$$Vin = V_{RL} + [(V_{RH} - V_{RL})/(2^b-1)]*ADvalue = 0 \text{ V} + 4.88 \text{ mV} \times 346 = 1.6894 \text{ V}$$

Using the HCS12 A/D converter

1. Power up A/D Converter (Bit 7 -> 1 in ATD0CTL2)
2. Select number of conversions per sequence (Bits 3,4,5,6 of ATD0CTL3)
   S8C S4C S2C S1C = 0001 to 0111 for 1 to 7 conversions
   S8C S4C S2C S1C = 0000 or 1xxx for 8 conversions
3. Set up ATD0CTL4
   - For 8-bit mode write 0x85 to ATD0CTL4
   - For 10-bit mode write 0x05 to ATD0CTL4
   - Other values of ATD0CTL4 either will not work or will result in slower A/D conversion rates
4. Select DJM, Bit 7 of ATD0CTL5
   (a) DJM = 0 => Left justified data in the result registers
   (b) DJM = 1 => Right justified data in the result registers
5. Select DSGN, Bit 6 of ATD0CTL5
   (a) DSGN = 0 => Unsigned data representation in the result register
   (b) DSGN = 1 => Signed data representation in the result register
6. Select MULT, Bit 4 of ATD0CTL5:
   - MULT = 0: Convert one channel the specified number of times
     – Choose channel to convert with CC, CB, CA of ATD0CTL5.
   - MULT = 1: Convert across several channels. CC, CB, CA of ATD0CTL is the first channel to be converted.
7. Select SCAN, Bit 5 of ATD0CTL5:
   - SCAN = 0: Convert one sequence, then stop
   - SCAN = 1: Convert continuously

8. After writing to ATD0CTL5, the A/D converter starts, and the SCF, Bit 7 of ATD0STAT0, is cleared. After a sequence of conversions is completed, the SCF flag in ATD0STAT0 is set.
    - You can read the results in ATD0DRxH.

9. If SCAN = 0, you need to write to ATD0CTL5 to start a new sequence. If SCAN = 1, the conversions continue automatically, and you can read new values in ATD0DRxH.

10. To get an interrupt after the sequence of conversions are completed, set ASCIE bit of ATD0CTL2. After the sequence of conversions, the ASCIF bit in ATD0CTL2 will be set, and an interrupt will be generated.

11. With 24 MHz bus clock and ATD0CTL4 = 0x05, it takes 7 μs to make one conversion, 56 μs to make eight conversions.

12. The conversion from Volts to Digital Values is

    ATD0DRx $=(V_{in} - V_{RL})/(V_{RH} - V_{RL}) \times 1024$. Normally,

    $V_{RL} = 0$ V, and $V_{RH} = 5$ V, so ATD0DRx $=V_{in}/5 \text{ V} \times 1024$

14. To use 10-bit result, set ATD0CTL4 = 0x05 (Gives 2 MHz AD clock with 24 MHz bus clock, 10-bit mode).

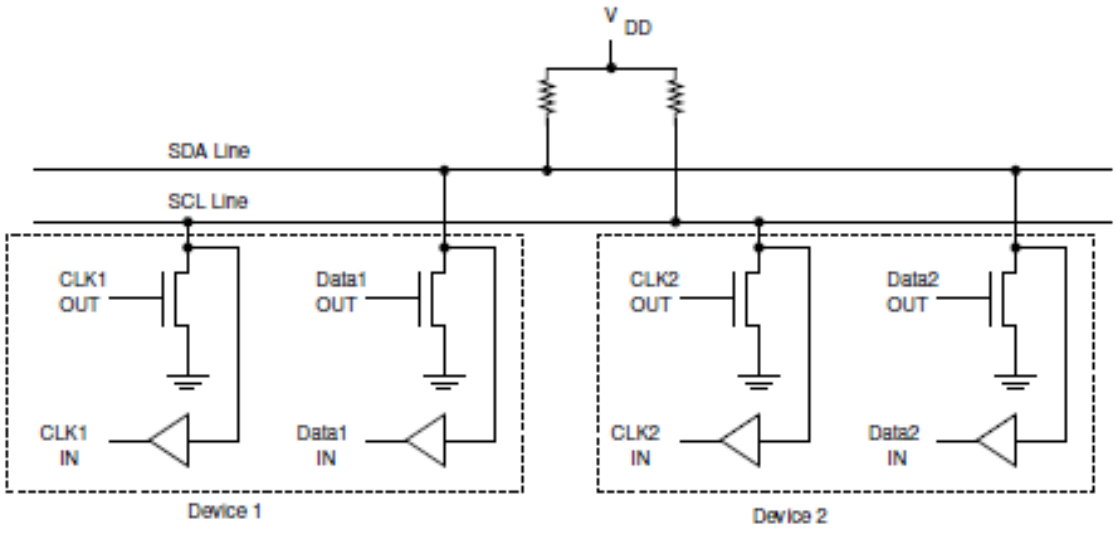
The HCS12 IIC Interface

• A popular synchronous serial interface is the Inter-Integrated Circuit (IIC or I2C) bus

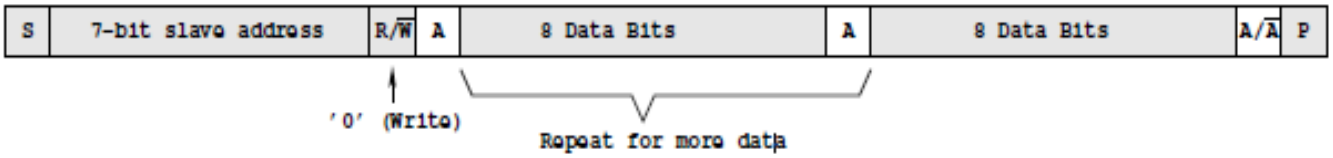
– The IIC bus can control multiple devices using only two wires
– The two wires are Clock and Data
*The devices connect to the wires using a *wired AND* method
* The lines are normally high. Any device on the bus can bring them low.

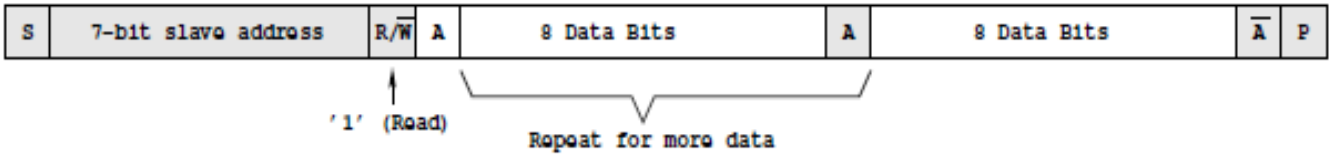
– Each device on the bus has a unique address.

– An IIC master starts the process by sending out a serial stream with the seven-bit address of the slave it wants to talk to, and an eight bit indicating if it wants to write to the slave or read from the slave.



The IIC Interface





From master to slave

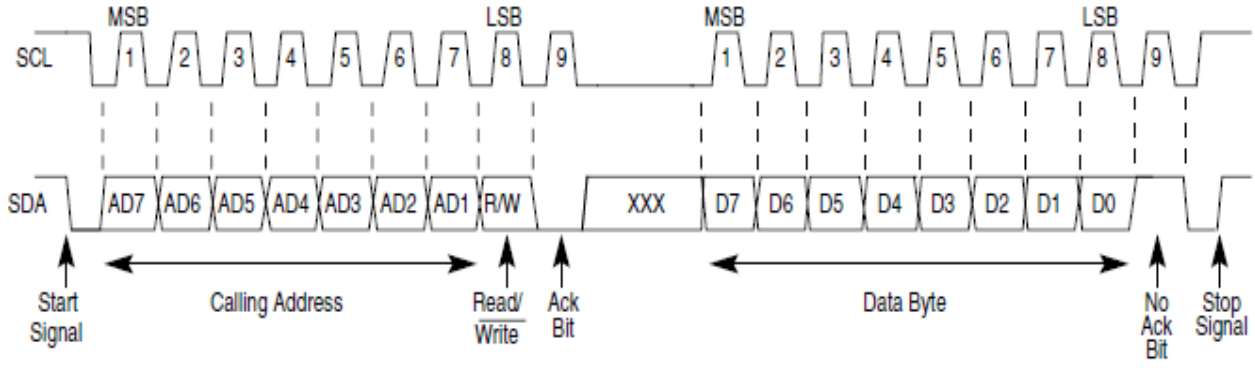
From slave to master

A = Acknowledge (SDA low)

$\overline{A}$ = Not Acknowledge (SDA high)

S = Start condition

P = Stop condition



Using the IIC Interface on the MC9S12

• The IIC uses five registers

1. **IBAD** (IIC Bus Address Register). This is the address of the IIC when it is addressed as a slave.

2. **IBFD** (IIC Bus Frequency Divide Register). This register determines the speed of the transfers. Table 3-4 of the data sheet shows what the divide times and hold times are for all possible values of IBFD.

3. **IBCR** (IIC Bus Control Register) This register controls the IIC.
   - **IBEN**: Enable the IIC Bus
   - **IBIE**: Enable interrupts
   - **MS/SL**: Switch into master mode
   - **Tx/Rx**: Switch between transmit and receive
   - **TXAK**: Send an acknowledge

4. **IBSR** (IIC Bus Status Register). This register indicates the status of the IIC, and is used to clear interrupts bits.
   - **TCF**: Transmit complete flag. Interrupt generated when TCF goes from low to high when IBEN is set.
   - **IAAS**: Addressed as slave
   - **IBB**: IIC Bus Busy
   - **IBAL**: Arbitration lost
   - **SRW**: Slave read/write
   - **IBIF**: Interrupt flag. Set when arbitration is lost. Clear by writing a 1 to this bit
   - **RXAK**: Received Acknowledge

5. **IBDR** (IIC Bus Data Register)
   – For both write to and read from the slave. First write to IBDR must be slave address plus R/W bit.
   – Write data to this register to send to slave
   – Read data from this register to receive from slave