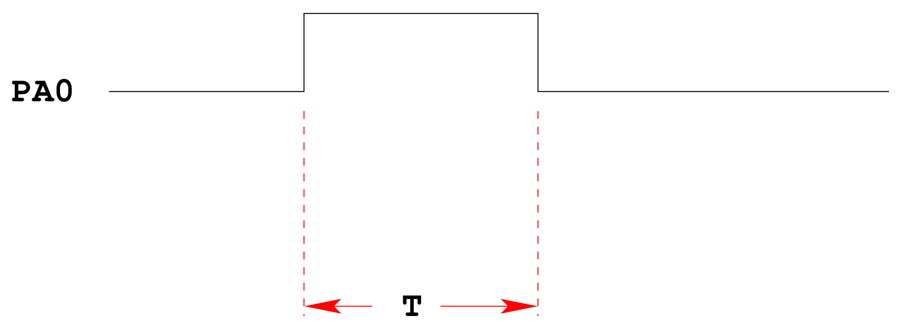
The MC9S12 Output Compare Function

;

Want event to happen at a certain time

Want to produce pulse pulse with width T

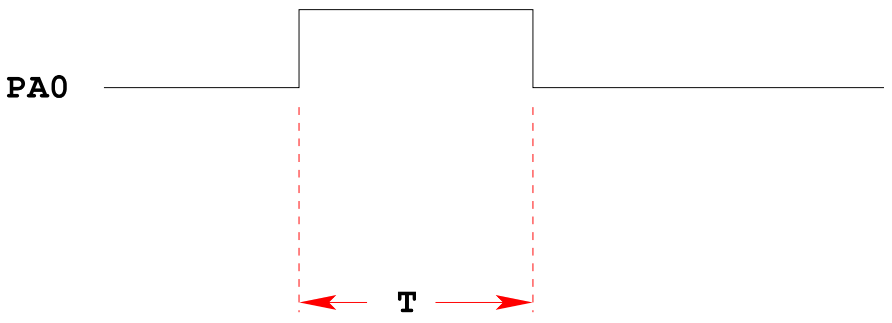


```
Wait until TCNT == 0x0000, then bring PA0 high

Wait until TCNT == T,      then bring PA0 low
```

```
  while (TCNT != 0x0000) ;
  PORTA = PORTA | BIT0;
  while (TCNT != T) ;
  PORTA = PORTA & ~BIT0;
```

Want event to happen at a certain time

Want to produce pulse pulse with width T



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

Wait until TCNT == T, then bring PA0 low

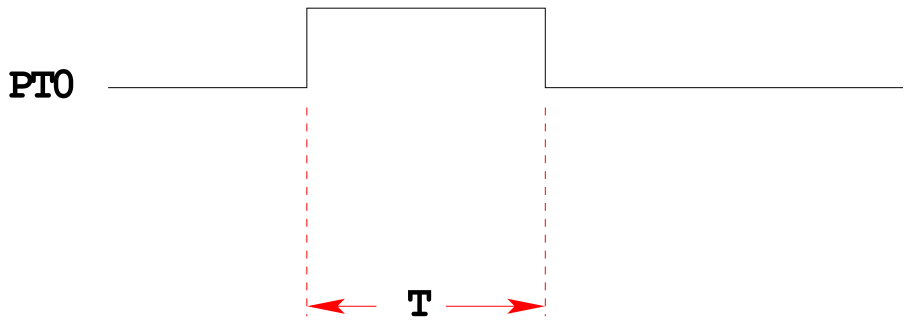
```
  while (TCNT != 0x0000) ;
  PORTA = PORTA | BIT0;
  while (TCNT != T) ;
  PORTA = PORTA & ~BIT0;
```

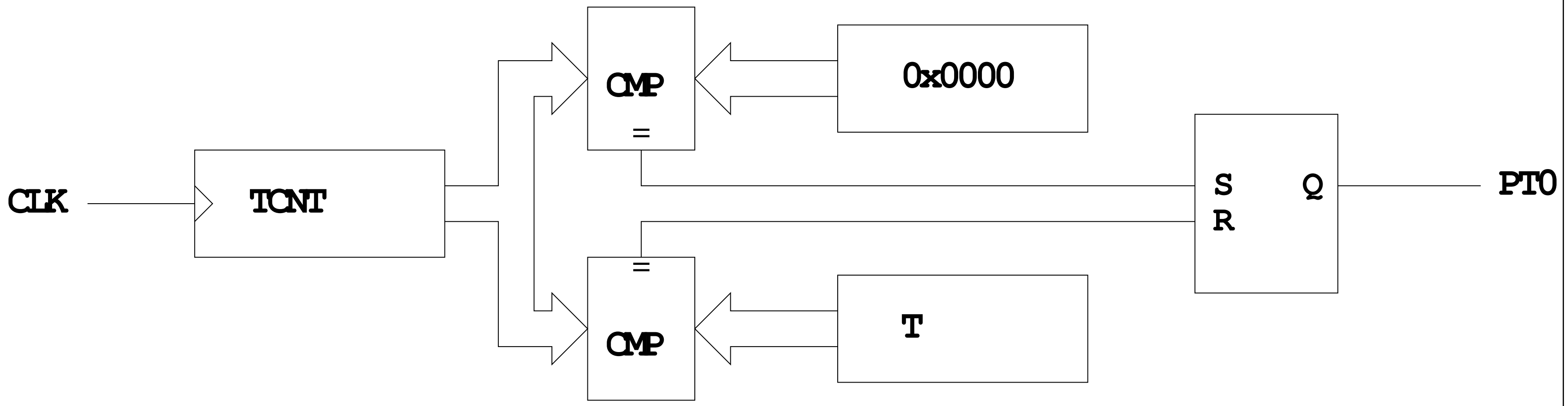
Problems:

1) May miss TCNT == 0x0000 or TCNT == T
2) Time not exact -- software delays
3) Cannot do anything else while waiting

Want event to happen at a certain time

Want to produce pulse pulse with width T





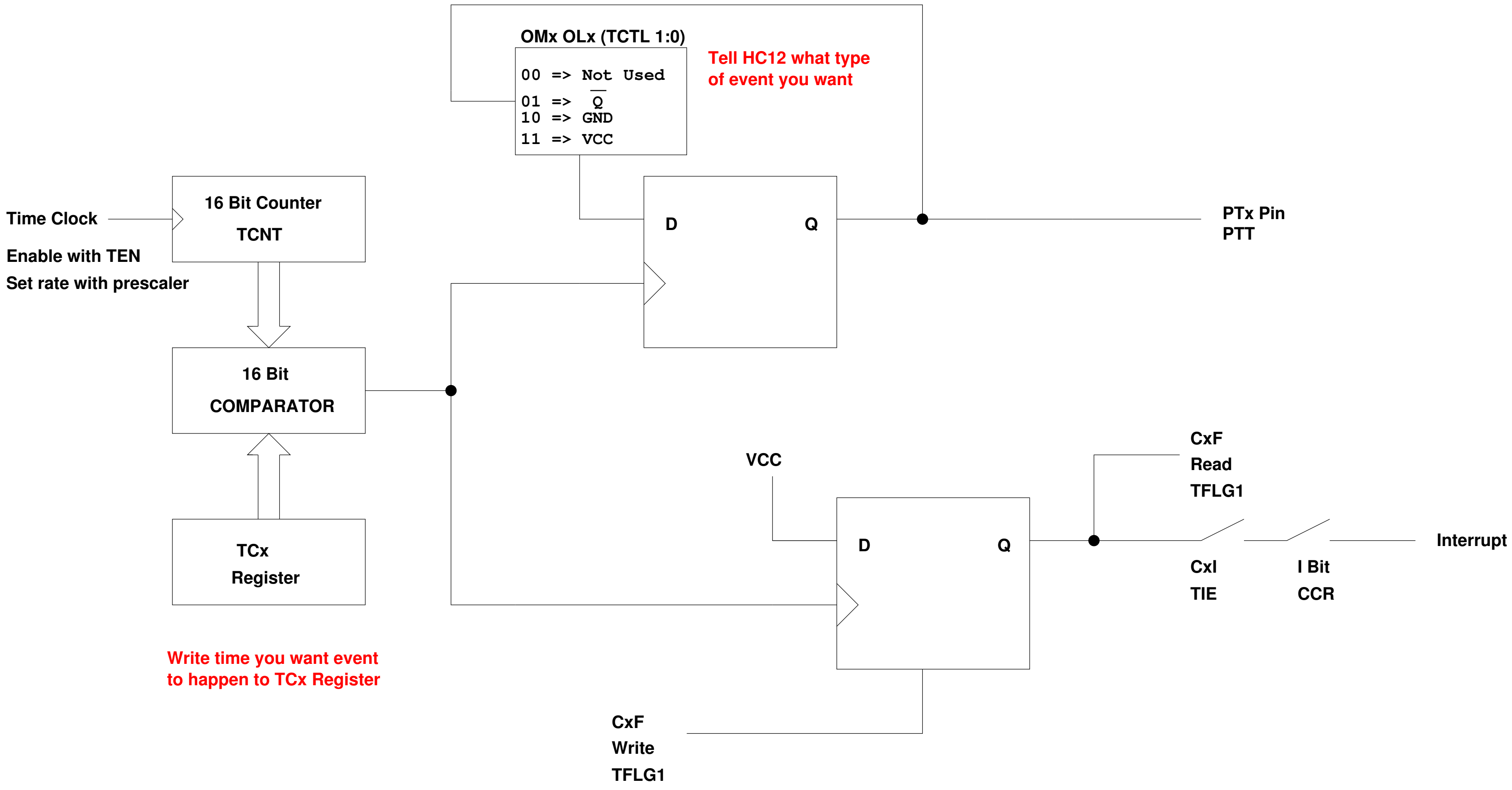
When TCNT == 0x0000, the output goes high

When TCNT == T, the output goes low

Now pulse is exaclty T cycles long

OUTPUT COMPARE PORT T 0–7

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



The MC9S12 Output Compare Function

- The MC9S12 allows you to force an event to happen on any of the eight `PTT` pins
- An external event is a rising edge, a falling edge, or a toggle
- To use the Output Compare Function:
  - Enable the timer subsystem (set `TEN` bit of `TSCR1`)
  - Set the prescaler
  - Tell the MC9S12 that you want to use Bit x of `PTT` for output compare
  - Tell the MC9S12 what you want to do on Bit x of `PTT` (generate rising edge, falling edge, or toggle)
  - Tell the MC9S12 what time you want the event to occur
  - Tell the MC9S12 if you want an interrupt to be generated when the event is forced to occur
- There are some more complicated features of the output compare subsystem which are activated using registers `CFORC`, `OC7M`, `OC7D` and
- Writing a 1 to the corresponding bit of `CFORC` forces an output compare event to occur, the same as if a successful comparison has taken place (Section 8.6.5 of Huang).
- Using `OC7M` and `OC7D` allow Timer Channel 7 to control multiple output compare functions (Section 8.6.4 of Huang).
- Using `TTOV` allows you to toggle an output compare pin when `TCNT` overflows. This allows you to use the output compare system to generate pulse width modulated signals.
- We will not discuss these advanced features in this class.

Write a 1 to Bit 7 of TSCR1 to turn on timer

TEN	TSWAI	TSBCK	TFFCA					0x0046 TSCR1

To turn on the timer subsystem: TSCR1 = 0x80;

Set the prescaler in TSCR2

Make sure the overflow time is greater than the width of the pulse you want to generate

TOI	0	0	0	TCRE	**PR2**	**PR1**	**PR0**	0x004D TSCR2

PR2	PR1	PR0	Period (μ s)	Overflow (ms)
0	0	0	0.0416	2.73
0	0	1	0.0833	5.46
0	1	0	0.1667	10.92
0	1	1	0.3333	21.84
1	0	0	0.6667	43.69
1	0	1	1.3333	86.38
1	1	0	2.6667	174.76
1	1	1	5.3333	349.53

To have overflow rate of 21.84 ms:

TSCR2 = 0x03;

Write a 1 to the bits of TIOS to make those pins output compare

IOS7	IOS6	IOS5	IOS4	IOS3	IOS2	IOS1	IOS0	0x0080	TIOS

To make Pin 4 an output compare pin: TIOS = TIOS | 0X10;

Write to TCTL1 and TCTL2 to choose action to take

OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4	0x0048	TCTL1
OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0	0x0049	TCTL2

OMn	OLn	Configuration
0	0	Disconnected
0	1	Toggle
1	0	Clear
1	1	Set

To have Pin 4 toggle on compare:

TCTL1 = (TCTL1 | BIT0) & ~BIT1;

Write time you want event to occur to TCn register.

To have event occur on Pin 4 when TCNT == 0x0000: TC4 = 0x0000;

To have next event occur T cycles after last event, add T to TCn.

To have next event occur on Pin 4 500 cycles later: TC4 = TC4 + 500;

When TCNT == TCn, the specified action will occur, and flag CFn will be set.

To clear the flag, write a 1 to the bit you want to clear (0 to all others)

CF7	CF6	CF5	CF4	CF3	CF2	CF1	CF0	0x004E	TFLG1

To wait until TCNT == TC4: while ((TFLG1 & BIT4) == 0) ;

To clear flag bit for Pin 4: TFLG1 = BIT4;

To enable interrupt when compare occurs, set corresponding
bit in TIE register

C7I	C6I	C5I	C4I	C3I	C2I	C1I	C0I	0x004C	TIE

To enable interrupt when TCNT == TC4: TIE = TIE | BIT4;

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)
   (e) Clear Flag (TFLG1 reg)
   (f) 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

   (b) Clear flag (TFLG1 reg)

Setting and Clearing Bits in the Timer Subsystem

- Registers in the timer subsystem control multiple timer channels.
  - Usually, you want to use ANDS and ORS to change only that channel you are working on.
  - For example, to make Channel 2 an output compare, and set it to toggle on compare, do this:

    ```
    TIOS = TIOS | BIT2;              /* Configure PT2 as Output Compare */
    TCTL2 = (TCTL2 | BIT4) & ~BIT5;  /* Set up PT2 to toggle on compare */
    ```

  - Do not do this:

    ```
    TIOS = BIT2;                 /* Configure PT2 as Output Compare */
    TCTL2 = BIT4);               /* Set up PT2 to toggle on compare */
    ```

    This would set up Channel 2 as an output compare, toggle on successful compare. However, it will force all the other channels to input capture – this may not be what you want to do.

- **To clear a flag bit, do not use ORs!**
  - To clear Timer Channel 2 flag, do the following:

    ```
    TFLG1 = BIT2;
    ```

    This will clear Timer Channel 2 flag, and leave all other flags unaffected.
  - Do not do this:

    ```
    TFLG1 = TFLG1 | BIT2;   /* DO NOT DO THIS */
    ```

    This will clear Timer Channel 2 flag, but will also clear any other flag which is set. Suppose, for example, Timer Channel 2 and Timer Channel 3 flags are both set at the same time, so `TFLG1` register is `0x0C`. You want to deal the Timer Channel 2 first and Timer Channel 3 afterwards.
    The command:

    ```
    TFLG1 = TFLG1 | BIT2;   /* DO NOT DO THIS */
    ```

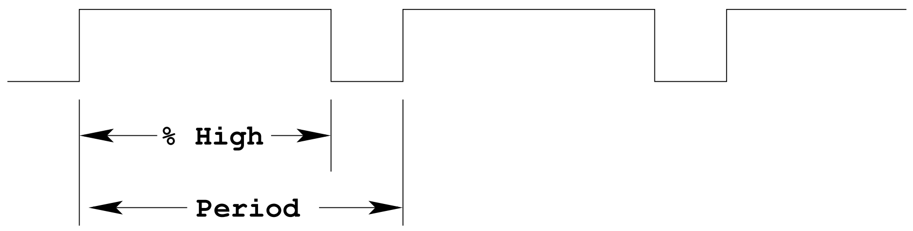
    will read TFLG1, which will return an `0x0C`. ORing that with a `0x04` (`BIT2`) will result in an `0x0C`. Writing that back to `TFLG1` will clear Timer Channel 2 flag and Timer Channel 3 flag. Now Timer Channel 3 flag is cleared, so you will never deal with the event which set Timer Channel 3 flag.

```
/*
 * Program to generate square wave on PT2
 * Frequency of square wave is 500 Hz
 * Period of square wave is 2 ms
 * Set prescale to give 0.667 us cycle
 * 2 ms is 3,000 cycles of 1.5 MHz clock
 *
 */
#include "hcs12.h"
#include "vectors12.h"

#define PERIOD      3000
#define HALF_PERIOD (PERIOD/2)

#define disable()  asm(" sei")
#define enable()  asm(" cli")

void INTERRUPT toc2_isr(void);

main()
{
    disable();

    TSCR1 = BIT7;                     /* Turn on timer subsystem */
    TSCR2 = 0x04;                     /* Set prescaler to 16 (0.666 us) */

    TIOS = TIOS | BIT2;              /* Configure PT2 as Output Compare */
    TCTL2 = (TCTL2 | BIT4) & ~BIT5;  /* Set up PT2 to toggle on compare */
    TFLG1 = BIT2;                    /* Clear Channel 2 flag */
    /* Set interrupt vector for Timer Channel 2 */
    UserTimerCh2 = (unsigned short) &toc2_isr;
    TIE = TIE | BIT2;                /* Enable interrupt on Channel 2 */

    enable();

    while (1)
    {
        asm("wai");
    }
}

void INTERRUPT toc2_isr(void)
{
    TC2 = TC2 + HALF_PERIOD;
    TFLG1 = BIT2;
}
```

Pulse Width Modulation

- Often want to control something by adjusting the percentage of time the object is turned on
- For example,
  - A DC motor — the higher the percentage, the faster the motor goes
  - A light – the higher the percentage, the brighter the light
  - A heater – the higher the percentage, the more heat output
- Can use Output Compare to generate a PWM signal
- Because PWM is used so often the MC9S12 has a built-in PWM system
- The PWM system on the MC9S12 is very flexible
  - It allows you to set a wide range of PWM frequencies
  - It allows you to generate up to 8 separate PWM signals, each with a different frequency
  - It allows you to generate 8-bit PWM signals (with 0.5% accuracy) or 16-bit PWM signals (with 0.002% accuracy)
  - It allows you to select high polarity or low polarity for the PWM signal
  - It allows you to use left-aligned or center-aligned PWM signals
- Because the MC9S12 PWM systes is so flexible, it is fairly complicated to program
- To simplify the discussion we will only discuss 8-bit, left-aligned, high-polarity PWM signals.

Pulse Width Modulation



Need a way to set the PWM period and duty cycle

The HC12 sets the PWM period by counting from 0 to some maximum count with a special PWM clock

PWM Period = PWM Clock Period x (Max Count + 1)

Once the PWM period is selected, the PWM duty cycle is set by telling the HC12 how many counts it should keep the signal high for

PWM Duty Cycle = (Count High + 1)/(Max Count + 1)

The hard part about PWM on the HC12 is figuring out how to set the PWM Period