Introduction

- We want separate timing circuitry that runs independent of our main program
  - Our main program can't keep good timing due to unexpected things like IRQ's
- A timer is just a digital counter

Timer Overview

Timer functionality we will studied...

1) Main Free Running Timer (TCNT)
2) Timer Output Compare
3) Timer Input Capture
4) Pulse Accumulator
5) Real Time Interrupt
6) Pulse Width Modulator

Examples of application that use timer functionality

1) Count items on an assembly line
2) Generate a 30/70 duty cycle to control a motor
3) Measure phase of an incoming signal
4) Generate an edge every n seconds
5) Latch data from peripheral when an edge occurs
6) Generate interrupt every 50ms
An Overview of HC12 Timer

- 16-bit free-running counter based on system bus clock (e.g. E CLK)
- 8 timer channels, each configurable as:
  - Output compare: can generate variety of waveforms by comparing counter vs. programmable register
  - Input capture: latch value of counter on selected edge of timer input pins
- 16-bit pulse accumulator to count external events, or act as gated timer of internal pulses
- Programmable, periodic interrupt generator called RTI (real-time interrupt)

Programming HC12 Timer

- Most complex subsystem of the HC12, many control registers and bits
- All timer functions similarly programmed
- All have separate interrupt controls and vectors
  - Interrupts enabled/disabled by bit in control register
- All have flags that get set when some programmable condition is satisfied (reset by program)
- Thus, when operation of one timer is learned, procedures similar for all others

Basic Timer

- The heart of the timers is a 16-bit, free running, Main Timer (TCNT)
  - Other Timing functions are based off of this timer
- Its input clock from system bus clock, but may be prescaled via division by 1, 2, 4, 8, 16, or 32
  - PR2:PR1:PR0 prescale factors in control register TMSK2

<table>
<thead>
<tr>
<th>PR2:PR1:PR0</th>
<th>Bus Clock Divider</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
</tr>
<tr>
<td>001</td>
<td>2</td>
</tr>
<tr>
<td>010</td>
<td>4</td>
</tr>
<tr>
<td>011</td>
<td>8</td>
</tr>
<tr>
<td>100</td>
<td>16</td>
</tr>
<tr>
<td>101</td>
<td>32</td>
</tr>
</tbody>
</table>

Basic Timer

- TCNT ($84:$85) starts at $0000 on reset, runs continuously unless disabled or stopped (e.g. wait mode)
- Cannot be set by program in normal mode, but contents can be read at any time
- TCNT should be read by 16-bit read instruction (e.g. LDD $84) to fetch whole value
- Overflows after $FFFF, setting Timer Overflow Flag (TOP), which can use to extend range
Timer Overflow Hardware

TOF Polling Example

```assembly
;Dr. Tao Li
;Constant Equates
NTIMES: EQU 122 ; Number of TOF's

;I/O Register Equates
TFLG2: EQU $8F ; TFLG2 register
TSCR: EQU $86 ; Timer system ctrl. reg
TOF: EQU $10000000 ; Timer overflow flag
TEN: EQU $10000000 ; Timer enable

;Clear the TOF first
lda #TOF
staa TFLG2

;Enable the timer
bset TSCR, TEN

;Initialize the counter and wait for NTIMES
lda #NTIMES
staa counter

;spin WHILE TOF is set and
bus Clock

;After the TOP=1, clear TOF
lda #TOF
staa TFLG2

;IF counter != 0 spin
bne spin1
```

TOF Interrupt

```assembly
;Dr. Tao Li
;Constant Equates
TOFVect: EQU $FFDE ;TOF vector address
NTIMES: EQU 122 ;Number times to interrupt

;I/O Register Equates
TOF: EQU $10000000 ; Timer Overflow Flag
TMSK2: EQU $10000000 ; Timer Overflow 1st
TSCR: EQU $10000000 ; Timer control reg
TFLG2: EQU $10000000 ; Timer Enable

;Example: Use TOF interrupt to generate ~1s delay

See Next Page
```

Handling Timer Overflow

- TOF (bit-7 in TFLG2 register) can be used in two ways: polling or interrupting
- Polling – program polls value of TOF; after asserted, must reset each time by writing '1' to TOF bit
- **Example:** Using TOF to generate delay of ~1s assuming 8MHz bus clock
  - 122 overflows, each of 64K periods of 125ns (8.192ms) ⇒ 122×64K×125ns ≈ 1 second!

```
Dr. Tao Li

TOF Interrupt

- TOF can generate interrupt if TOI bit in TMSK2 register enabled, TOF interrupt vector is initialized in vector table, and i-bit unmasked in CCR
- **Example:** Use TOF interrupt to generate ~1s delay

See Next Page
```

Dr. Tao Li

TOF Interrupt
### TOF Interrupt Example

1. Clear the TOF
   - `lda #TOF`
   - `sta TFLG2`

2. Enable the timer
   - `bset TSC0, TEN`
   - `bset TMRK2, TOF`
   - `bset TSCR, TEN`
   - `cli`

3. Enable the interrupt system
   - `sta TMRK2`

4. Clear the TOF bit
   - `lda #TOF`
   - `sta TFLG2`

5. Return to main program
   - `rti`

**Do Forever**

```
start:
  wai
  ; Wait for the interrupt

  ; When the counter incremented by the ISR
  ; reaches a maximum given
  ; by NTIMES, do some work and reset the
  ; counter value.

  ; IF Counter = maximum
  lda Counter
  cmpa #NTIMES
  bne endif

  ; DO SOME WORK HERE

  ; and reset the Counter

  clra
  sta Counter

  ; ENDIF Counter=maximum

  endif:
  bra start
```

### Output Compare Timers

- Each of 8 timer channels can be configured as input capture (from Port T) or output compare (to Port T), or Port T used for GP I/O as before if timers not used; choice via TIOS reg

- Output compare allows more accurate timing delays than the TOF

- Each of the 8 timer channels has a 16-bit timer capture/compare register (TCn: TC7...TC0), may be loaded or stored with 16-bit value

### Output Compare Timers

- **TCn** register compared with **TCNT** every clock cycle; when equal then flag for that channel (**CnF**: C7F...C0F) is set; can poll this flag, or...

- If interrupt enabled for that channel (**CnI**: C7I...C0I) and I-bit in **CCR** unmasked, then **OC** interrupt occurs

### Timer Output Compare Hardware

- **TCn** register compared with **TCNT** every clock cycle; when equal then flag for that channel (**CnF**: C7F...C0F) is set; can poll this flag, or...

- If interrupt enabled for that channel (**CnI**: C7I...C0I) and I-bit in **CCR** unmasked, then **OC** interrupt occurs
**OC Time Delays**

- Example: Use output compare to achieve 1ms delay (8000 cycles of 8MHz bus clock)

  **Constant Equates**
  
  I/O Register Equates
  
  TIOS: EQU $80 ; Input Cap/Out Compare Select
  
  TSCR: EQU $86 ; Timer System Control
  
  TFU[1]: EQU $88 ; TFU[1] register
  
  TCNT: EQU $84 ; TCNT register
  
  TFLG[1]: EQU $8E ; TFLG[1] register
  
  TC1: EQU $92 ; Timer channel 1
  
  C1F: EQU %00000010 ; Output compare 1 Flag
  
  TEN: EQU %10000000 ; Timer Enable

  **Bus Clock**

<table>
<thead>
<tr>
<th>Constant</th>
<th>I/O Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE_MS:</td>
<td>EQU</td>
<td>8000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  **Example:**

  - Enable the timer hardware
    1. bset TSCR.TEN
  - Enable Output Compare Channel 1
    2. bset TIOS.C1F
  - Just generate a 1 ms delay here
    3. ldd TCNT
    4. add #ONE_MS
    5. std TCNT
    6. ldd C1F
    7. stda TFLG1
    8. Wait until the flag is set
    9. spin: brclr TFLG1,C1F,spin

  **In last example, delay limited to ≤8.192ms when 8MHz bus clock used (125ns x 64K)**

  **But, can divide TCNT increment of bus clock cycles using PR2:PR1:PR0 prescale bits**

  **For example:** for generation of a 10ms delay
  - Uses PR2:PR1:PR0 = 010 (i.e. ÷4) ⇒ each clock pulse is 500ns
  - ⇒ delay = 500ns/cycle × 20000 cycles = 10ms

**Changing the Timer Prescaler**

- In last example, delay limited to ≤8.192ms when 8MHz bus clock used (125ns x 64K)
- But, can divide TCNT increment of bus clock cycles using PR2:PR1:PR0 prescale bits
- For example: for generation of a 10ms delay
  - Uses PR2:PR1:PR0 = 010 (i.e. ÷4) ⇒ each clock pulse is 500ns
  - ⇒ delay = 500ns/cycle × 20000 cycles = 10ms

**Changing the Timer Prescaler**

- Get the current prescaler value and save it
  1. ldd TMSK2
  2. pulb TMSK2
  3. Set the prescaler to divide by 4
  4. bsr TMSK2,PR2:PR0
  5. bset TMSK2,PR1
  6. Now restore the original prescaler values
  7. pulb
  8. stab TMSK2

- Changes to prescaler bits will affect whole timer system, so may wish to restore afterwards
OC Interrupts

- Longer delays can also be generated by waiting for more output comparisons to be made.

- Example: generating 1s delay using OC flag to generate interrupt.
  - Achieved by waiting 250 complete 4ms delay times (125ns/cycle × 32000 cycles = 4ms) generated by OC.
  - ~8ms for setup, interrupt occurs every 4ms, total 250 interrupts.

OC Time Delays

- Example: generating 1s delay using OC flag to generate interrupt.

OCV2EC: EQU $FFEA; Timer channel 2 interrupt vector.

Constant Equates

NTIMES: EQU 250 ; Number of 4 ms delays
D_4MS: EQU 32000 ; Num clocks for 4 ms

I/O Register Equates

TIOS: EQU $80 ; In capt/out compare select
TCNT: EQU $84 ; TCNT register
TSCR: EQU $86 ; Timer control register
TMSK1: EQU $8C ; Timer mask reg
TFLG1: EQU $8E ; TFLG1 offset
TC2: EQU $94 ; Timer register 2
TEN: EQU %1000000 ; Timer enable bit
C2F: EQU %00000100 ; Output compare 2 Flag
C2I: EQU C2F ; Interrupt enable
IOS2: EQU C2F ; Select OC2

OC Time Delays

```assembly
; Enable the timer system
bset TSCR, TEN
; Enable output compare channel 2
bset TIOS, IOS2
; Generate a 1s delay
ldaa #NTIMES
staa counter
; Grab the value of the TCNT register
ldd TCNT
std TC2
; Now have 8 ms to set up the system
; Set up interrupts
ldaa #C2F
staa TFLG1
; Clear C2F
bset TMSK1, C2I ; Enable TC2 Interrupt
cli ; Unmask global interrupts

; Wait until the counter is 0
spin:
wai ; Wait for interrupt
tst counter
bne spin ; When out of the spin loop

; Decrement the counter
isr: dec counter

; Set up TC2 for the next interrupt
ldd TC2
addd #D_4MS
std TC2

; And clear the C2F
ldaa #C2F
staa TFLG1

; Return to wait for the next interrupt
bra spin
```

Timer Output Compare Hardware
Dr. Tao Li

OC Bit Operation

- OC flags can auto. set or reset Port T bits when flag is set
- When successful OC occurs, one of fours actions may occur at output pin on Port T: disconnected, toggled, cleared, or set
- Selection made for each Port T output via OMn and OLn control bits in TCTL1 and TCTL2 regs

Example: OM2:OL2 = 01 to auto. toggle Port T-2 when OC occurs periodically
- Program outputs square wave w/ period of 2*OC delay (50% duty cycle)

```
TLCTL2: EQU $89
OM2: EQU %00100000
OL2: EQU %00010000

; Set up Output capture action to toggle the bit-2
bclr TLCTL2,OM2
bset TLCTL2,OL2
```

Input Capture

- Allows TCNT value latched when program-selected external event occurs via Port T
  - e.g. period of pulse train found by storing TCNT at start of period (i.e. rising or falling edge), then capture count at end of period (next rising or falling edge), and take difference
- Two bits for each IC channel, EDGnB (EDG7B...EDG0B) and EDGnA control when signal on Port T causes capture to occur (i.e. rising, falling or both edges activate)
- IC interrupts operate just like OC interrupts

Dr. Tao Li
Dr. Tao Li

Measure Waveform Period

1) Set TSCR, %10000000
2) Clear TIOS, %00000010
3) Load A with 100
4) Store A to TCTL4
5) Spin loop
6) Load D with TC1
7) Store D to First
8) Subtract First

Pulse Accumulator

- Port T, bit7 can be configured as a PA input
- A system that "counts" events
- There are two operating modes
  1) "Event Counting Mode"
     - Each time an edge occurs on PT7, a 16-bit counter is incremented
  2) "Gated Accumulator Mode"
     - When PT7 is asserted, a 16-bit counter will increment based on the Bus Clock / 64

Pulse Accumulator Operating Modes

- May write to or read from PA at any time, again should do so via 16-bit load/store
- May select the edge (positive or negative) for event counting, or the level (high or low) for gated time accumulation, via several control bits
- Two flags and corresponding interrupts available: (1) PA overflow (PAOVF flag); and (2) Selected input edge occurs (PAIF flag)
  - E.g., sensor on conveyor belt counting products as they pass, it wants to take action after 24 counts; initialize PA counter (PACNT) = -24, set up and use interrupt on PAOVG flag, and in ISR take appropriate action

Dr. Tao Li
Pulse Accumulator Setup

PACTL - "16-bit Pulse Accumulator Control Register"

- **PAEN** = PA System Enable Bit
  - **PAEN** = 0, disabled (default)
  - **PAEN** = 1, enabled

- **PAMOD** = PA Mode Select Bit
  - **PAMOD** = 0, Event Counter (default)
  - **PAMOD** = 1, Gated Accumulator

- **PEDGE** = PA Edge Control Bit
  - if (PAMOD = 0) "Event Counter"
    - **PEDGE** = 0, Falling Edge Count (default)
    - **PEDGE** = 1, Rising Edge Count
  - if (PAMOD = 1) "Gated Accumulator"
    - **PEDGE** = 0, Active HIGH (default)
    - **PEDGE** = 1, Active LOW

PA Flags & Interrupts

- **PA Flags**
  - A flag is set upon PACNT Overflow (PAOVF)
    - **PAOVF** = 1, event (reset by writing a "1")
    - **PAOVF** = 0, no event
  - A flag is set upon an input edge (PAIF)
    - **PAIF** = 1, event (reset by writing a "1")
    - **PAIF** = 0, no event

- **PA Interrupts**
  - IRQs can be generated upon PACNT Overflow (PAOVI)
    - **PAOVI** = 0, disabled (default)
    - **PAOVI** = 1, enabled
  - IRQs can be generated upon an input edge (PAI)
    - **PAI** = 0, disabled (default)
    - **PAI** = 1, enabled

Other Options for TCNT Clock Generator

- Two clock-select bits (CLK1:CLK0) in PACTL reg. to select clock source for TCNT reg
- When PA disabled (PAEN=0), bus clock prescaled by PR2:PR1:PR0 bits as before
  - Bus clock = 8MHz ⇒ clock TCNT @8MHz (0:0:0) down to 8MHz/32 = 250kHz (1:0:1)
- When PA enabled (PAEN=1), TCNT source can be derived from either an event signal on PT-7 in EC mode (PAMOD=0) or bus clock further divided (PAMOD=1)
- **PAEN:**CLK1:CLK0 are used for TCNT clock mux selection

TCNT Clock Generator
Real-Time Interrupt (RTI)

- RTI operates like TOF interrupt, except the periodic rate of generating interrupts is selectable
- Has its own vector in the vector table
- Enabled by RTIE bit, flag RTIF set at interval specified, and RTIF reset by ‘1’ as before
- RTI rate generated by a 13-bit counter that divides bus clock by $2^{13} = 8192$ or more via 3 RTI prescalar bits (RTR2:RTR0), where 000 = off, 001 = $\div 2^{13}$, 010 = $\div 2^{14}$, ..., 111 = $\div 2^{19}$

Real-Time Interrupt (RTI)

- Bus clock = 8MHz ⇒ intervals of 125ns $\times 2^{13} = 1.024$ms (001) up to 125ns $\times 2^{19} = 65.536$ms (111)
- Bus clock = 4MHz ⇒ intervals of 250ns $\times 2^{13} = 2.048$ms (001) up to 250ns $\times 2^{19} = 131.072$ms (111)
- Real Time Interrupt (RTI) vs. Timer Overflow Interrupt
  - Useful for slower Timer IRQ’s than TOF
  - This can be easier than counting many TOF’s when looking for slower events

External Interrupts using Timer Interrupts

- External inputs that generate timer interrupts may be used as GP vectored, external interrupts if pins are not o/w being used for I/O or timer functions
- PT-7 (PA input edge or IC7 interrupt), PT-6 (IC6 interrupt), ..., PT-0 (IC0 interrupt), using enable bits (PAI, C7I, ..., C0I), flags (PAIF, C7F, ..., C0F), and vectors as before
  - See Table 10.6

PWM on B32

- PWM (Pulse Width Modulator) module outputs up to 4 pulse-width modulated waveforms at once
PWM

- Once initialized/enabled, outputs automatically with no further action from program
- Useful for many applications (e.g. controlling stepper motors)
- Programmable PERIOD and DUTY CYCLE

\[
\text{Duty Cycle} = \frac{t_{\text{DUTY}}}{t_{\text{PERIOD}}} \times 100\%
\]

Programming PWM

1) We program period in PWM.PER register (s) - give in terms of clock cycles
2) We program duty cycle in PWMDTY register (s) - can select 25%, 50%, 75%
3) Can select Pulse Alignment - can select LEFT or CENTER
4) Can select Clock Input - Bus Clock / n

Example of PWM Waveforms

- Replicate
- Mirror

PWM Concatenation

- PWM registers and counter may be concatenated in pairs for 16-bit timing resolution
  - PWCT3 and PWCT2 together, PWCT1 and PWCT0 together
- Gives longer period and higher duty-cycle resolution
- May have four 8-bit, two 16-bit, or one 16-bit and two 8-bit PWM registers
Four clock sources derived from bus clock: Clock A, Clock B, Clock S0, and Clock S1

Clocks A and B each may be produced as bus clock divided by 1, 2, 4, ..., 64, 128

Clocks S0 and S1 produced by further dividing A and B, respectively, by 2, 4, 6, 8, ..., 512

**Question:**
What is the longest PWM period available assuming 8MHz bus clock and left-aligned waveform?

**Answer:**
Achieved by using concatenated PWPER register (16-bit) and slowest clock available (i.e. S0 or S1):
Bus clock period \( \times \) Clock A/B multiple of 128 \( \times \) Clock S0/S1 multiple of 512 \( \times \) max. # of counts
= 125ns \( \times \) 128 \( \times \) 512 \( \times \) 64K = 536.87s \( \approx \) 9 minutes!