Notes - Unit 9

TIMER FUNCTIONS

HCS12D: ENHANCED CAPTURE TIMER

MAIN FUNCTIONS
- Input Capture
- Output Compare
- Pulse accumulator
- Modulus Down Counter

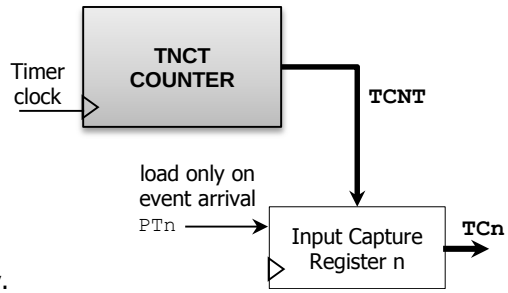
FEATURES
- 8 channels of Input Capture/Output Capture (`TC0, TC1, …, TC7`): a channel can only be configured as either Input Capture or Output Compare. Each channel can generate an interrupt.
- 16-bit counter (`TCNT`): Base Timer for all Channels.
- Pulse Accumulators:
  - ✓ Four 8-bit Pulse Accumulators (`PAC3, PAC2, PAC1, PAC0`), or
  - ✓ Two 16-bit Pulse Accumulator (`PACA=|PAC3|PAC2|, PACB=|PAC1|PAC0|`)
- Modulus Down Counter with pre-scaler (1,4,8, or 16)
- $Timer\ clock = \frac{E-clock\ (bus\ speed)}{Pre-scale\ fctor}$. The pre-scaler can be 1, 2, 4, 8, 16, 32, 64, 128.
- I/O Pins: `PORT T`: `PT7-PT0`. `PT7` is the Pulse accumulator input pin when used as a Pulse accumulator input. `PT3-PT0` can also be used as the pulse accumulator input for `PAC3-PAC0`. `PT7` can also be used for `PACA`, `PT0` for `PACB`.
- Interrupts:
  - ✓ Each Input Capture/Output Compare Channel can generate an Interrupt (`TC0 - TC7` Interrupt):
    - `$FFEE`: Enhanced Capture Timer channel 0
    - `$FFEC`: Enhanced Capture Timer channel 1
    - `$FFEA`: Enhanced Capture Timer channel 2
    - `$FFE8`: Enhanced Capture Timer channel 3
    - `$FFE6`: Enhanced Capture Timer channel 4
    - `$FFE4`: Enhanced Capture Timer channel 5
    - `$FFE2`: Enhanced Capture Timer channel 6
    - `$FFE0`: Enhanced Capture Timer channel 7
  - ✓ Enhanced Capture Timer Overflow (TOF) Interrupt:
    - `$FFDE`: Enhanced Capture Timer Overflow
  - ✓ Pulse Accumulator Overflow Interrupts: PACA, PACB
    - `$FFDC`: Pulse accumulator A overflow
    - `$FFC8`: Pulse accumulator B overflow
  - ✓ Pulse accumulator input Interrupt (`PT7` pin)
    - `$FFDA`: Pulse accumulator input edge Interrupt
  - ✓ Modulus Down Counter Interrupt:
    - `$FFCA`: Modulus Down Counter Underflow.

TIMER COUNTER REGISTER (TCNT)
- `TCNT ($0044:$0045)`: Read two bytes at once (e.g.: `ldd TCNT`), because `TCNT` doesn't stop during access operation.
- `TCNT` Configuration: `TSCR1, TSCR2, TLFG1, TFLG2`
  - ✓ `TSCR1`:
    - Bit 7 (TEN): '0' to disable timer, '1' to allow normal functioning.
    - Bit 4 (TFFCA):
      - 1: A read from Input Capture (or write to Output Compare) channel clears the *CnF* flag on `TLFG1`. Also, any access to `TCNT` clears the TOF flag.
      - 0: A specific *CnF* bit in `TFLG1` can only be cleared by writing a '1' to it. Also, TOF can only be cleared by writing a '1' to it.
  - ✓ `TLFG1`: When a selected edge (see `TCTL3, TCLT4`) arrives at the input-capture pin, the corresponding flag Channel 'n' Flag *CnF* (`n=0,1,…,7`) is set to '1'.
  - ✓ `TLFG2`: Only bit 7 (TOF) is implemented. When TCNT rolls over from `$FFFF` to `$0000`, TOF is set to 1. This flag can be cleared by writing a '1' to TOF.
  - ✓ `TSCR2`:
    - Bit 7 (TOI): Interrupt enable for TOF. '1' to activate interrupt when TOF=1, '0' to disable interrupt.
    - Bit 3 (TCRE): Timer counter reset enable. '0': Inhibits counter reset. '1': Counter reset by a successful Output-Compare 7. If `TC7=$0000` then `TCNT=$0000` indefinitely; if `TC7=$FFFF`, TOF is never 1 when TCNT rolls from `$FFFF` to `$0000`.
    - Bits 2 to 0: `|PR2|PR1|PR0|`: E-clock Pre-scale factor = $2^{4\times PR2+2\times PR1+PR0}$.

INPUT CAPTURE

8 Input Capture Channels. Each Channel includes:

- A 16-bit input Capture Register `TCn, n = 0 → 7`
- Input pin `PTn`
- Interrupt generation circuit.



APPLICATIONS:

- Period measurement
- Pulse-width measurement
- Duty cycle measurement
- Phase difference measurement between 2 signals with the same frequency.
- Recording of arrival time for several events. The number of events is limited by the number of Input Capture Channels.
- Interrupt generator: All input capture pins PT0-PT7 can serve as edge sensitive interrupt sources.

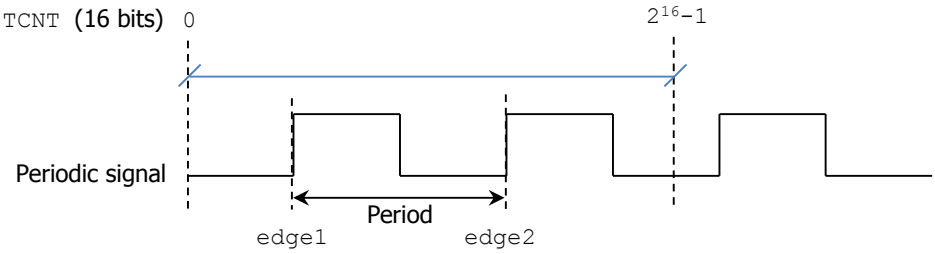
RELEVANT REGISTERS:

- Counter configuration: `TSCR1, TSCR2, TFLG1, TFLG2`.
- `TIOS`: Selects whether a pin is an Input Capture or Output Compare. 0 for Input Capture, 1 for Output Compare
- `TIE`: Timer Interrupt Enable Register. An input-capture channel can generate an interrupt request on the arrival of a selected edge if it is enabled by the corresponding TIE bit. 1 to enable (Local Enable), 0 to disable.
- Timer Control Registers: `TCTL3, TCTL4` (see Figure 8.6 in textbook). These registers specify what signal edge to capture. The selection of a channel edge is controlled by 2 bits: `EDGnB EDGnA`:
  - `EDGnB EDGnA = 00` → capture disabled on channel 'n'.
  - `EDGnB EDGnA = 01` → Rising edge on channel 'n'.
  - `EDGnB EDGnA = 10` → Falling edge on channel 'n'.
  - `EDGnB EDGnA = 11` → Both edges on channel 'n'

  When `EDGnB EDGnA` is not `00`, the associated pin `PTn` becomes an input tied to `ICn`, regardless of the state of `DDRT`.

Example: Period measurement on PT0.

- Assumption: period is shorter than 80 ms.
- We need to make sure that the count from $0\ to\ 2^{16}-1$ lasts at least 160 ms. If we set the Prescale factor to 64, then: $Timer\ clock = \frac{24\ MHz}{64} = 0.375\ Mhz \rightarrow Period = 2.67\ us$. Thus, our unit of measurement is 2.67 us. The smallest period we can measure is 2.67 us.
- A full count lasts 2^{16} cycles, which is $\frac{1}{0.375\times10^6} \times 2^{16} = 174.7626\ ms$.

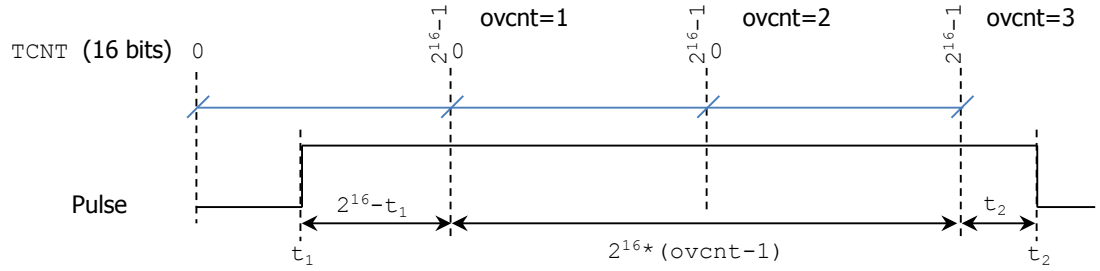


- Process:
  1. Enable Input Capture on Channel 0: `TIOS(0)=0` → `TIOS = 0x00`
  2. Select rising edge on Input Capture Channel 0: `TCTL4(1..0) = 01` → `TCTL4 = 0x01`
  3. Set pre-scaler to 64. → `TSCR2=0x06` (also TOF interrupt inhibited)
  4. Enable timer counter (starts from 0). Enable fast clear for TOF and *C0F*. → `TSCR1 = 0x90`
  5. Clear *C0F* flag (just in case): `TFLG1(0) = 1` → `TFLG1 = 0x01`
  6. Wait until `TFLG1(0) = 1` (we could have used an interrupt for this as well)
  7. edge1 = `TC0` (16-bit Input Capture Channel 0 Register). On the rising edge, `TC0` gets the counter value.
  8. Clear C0F flag (this is done by reading TC0 already)
  9. Wait until `TLFG(0) = 1`
  10. edge2 = `TC0`
  11. Period = edge2-edge1. Notice that edge2 is always greater or equal than edge1.

  C Code: `unit9a.c`

Example: Pulse width measurement on PT0.

- Unlike the previous example, here we will consider the fact that the width of the pulse might be longer than 2^{16} cycles. The following procedure will also apply for period measurement when the period is longer than 2^{16} cycles.
- We know that by using the pre-scale factor of 64, the unit of measurement is 2.67 us. If we want to accurately measure pulses as small as 1 us (for example), we need to use a pre-scale of 16, this results in: $Timer\ clock = \frac{24\ MHz}{16} = 1.5\ Mhz \rightarrow Period = 0.67\ us$. Thus, our unit of measurement is 0.67 us. The smallest pulse width we can measure is 0.67 us. A full count then would last $\frac{1}{1.5\times10^6} \times 2^{16} = 43.6906\ ms = 2^{16}\ cycles$.
- The pulse can be longer than 2^{16} cycles (or it can be shorter than 2^{16} but it includes a timer overflow). So, we need to keep track of the number of times the counter (`TCNT`) overflows. A TOF Interrupt can take care of this (the ISR will count the number of instances TOF generated an interrupt). Each overflow ($ovcnt$ variable) adds 2^{16} cycles to the pulse width.



$$Pulse\ Width = (ovcnt - 1) \times 2^{16} + t_2 + (2^{16} - t_1) = (ovcnt) \times 2^{16} + (t_2 - t_1)$$

- If we store the Pulse Width variable as a 32-bit number, then the maximum pulse width we can detect is $2^{32} cycles = \frac{1}{1.5\times10^6} \times 2^{32} = 2863.311\ seconds \approx 47\ mins$.

Computer Arithmetic issue when implementing the formula:

- This formula works well for $t_2 \geq t_1$.
- For $t_2 < t_1$, there is a problem due to the use of 16-bit integer arithmetic:
  Here, the difference $t_2 - t_1$ is negative. If we are using unsigned numbers with 16 bits, the result will be also a 16-bit unsigned number, which will be: $(2^{16} + t_2) - t_1$ since a borrow out is assumed. So, for the pulse width formula to be correct, we need to get rid of the 2^{16} extra factor, hence:

$$Pulse\ Width = (ovcnt - 1) \times 2^{16} + (t_2 - t_1), \qquad when\ t_2 < t_1$$

- Another solution is to convert t_2 and t_1 to signed long integers (32 bits). This will allow to use the general formula without problems. However, it will restrict the maximum possible detectable pulse width to 2^{31} cycles.
- Another solution is to do $t_1 - t_2$ instead, and then change the formula to: $Pulse\ Width = (ovcnt) \times 2^{16} - (t_1 - t_2)$. This requires slightly more processing.

- Process:
  1. Enable Input Capture on Channel 0: `TIOS(0)=0` → `TIOS = 0x00`
  2. Select rising edge on Input Capture Channel 0: `TCTL4(1..0) = 01` → `TCTL4 = 0x01`
  3. Set pre-scaler to 16. → `TSCR2=0x04`
  4. Enable timer counter (starts from 0) → `TSCR1 = 0x80`
  5. Clear *C0F* flag: `TFLG1(0) = 1` → `TFLG1 = 0x01`
  6. Wait until `TFLG1(0) = 1`
  7. Here we enable the TOF Interrupt:
     - Clear `TOF`: `TFLG2 = 0x80`, write '1' on `TOF` to clear it.
     - Local Enable for TOF: `TOI=1` → `TSCR2 = TSCR2|0x80.`
     - Global Enable: `asm("cli");`
  8. edge1 = `TC0` (16-bit Input Capture Channel 0 Register). On the rising edge, `TC0` gets the counter value.
  9. Select falling edge on Input Capture Channel 0: `TCTL4(1..0) = 10` → `TCTL4 = 0x02`
  10. Clear *C0F* flag: `TFLG1(0) = 1` → `TFLG1 = 0x01`
  11. Wait until `TLFG1(0) = 1`
  12. edge2 = `TC0`
  13. diff = edge2-edge1.
  14. If edge2 < edge1 then overflow = overflow -1
  15. $Pulse\ Width = Overflow \times 2^{16} + diff$

- *Interrupt Service Routine*: The TOF interrupt was enabled on Step 7. The global variable overflow is initialized with 0, and every time the TOF interrupt arrives, the ISR clears the TOF flag and increments the count on overflow.

C Code: `unit9b.c`

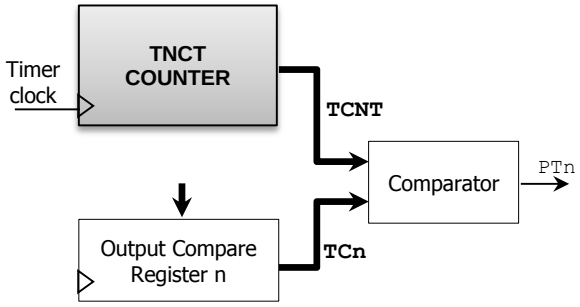
OUTPUT COMPARE

8 Output Capture Channels (OCs). Each Channel includes:

- A 16-bit Compare Register `TCn, n = 0 → 7`
- Output pin `PTn`
- Interrupt request circuit.
- A forced-compared function (`CFORCn`)

COMMON OPERATION:

- The user makes a copy of the current contents of `TCNT`
- Add to this copy a value that can generate a desired delay
- Stores the sum into an output-compare register (`TCn`), `n=0..7`
- The comparator compares `TCNT` and that `TCn` at every clock cycle. If they are equal:
  - A specified action is triggered on the `PTn` pin (to high, to low, toggle).
  - The associated flag in `TFLG1` is set to 1.
  - An interrupt requested is generated (if enabled).



APPLICATIONS:

- Single pulse generation
- Square wave generation
- Delay generation
- Event counting: Count number of events that occur during and interval.

RELEVANT REGISTERS:

- Counter configuration: `TSCR1, TSCR2, TFLG1, TFLG2`.
- Timer Control Registers: `TCTL1, TCTL2` (see Figure 8.18 in textbook). These registers specify the action to take on the output compare pin: to high, to low, toggle. The selection is controlled by 2 bits: `OMn OLn`:
  - `OMn OLn = 00` → No action (timer disconnected from output pin).
  - `OMn OLn = 01` → Toggle `PTn` pin.
  - `OMn OLn = 10` → Clear `PTn` pin to 0.
  - `OMn OLn = 11` → Set `PTn` pin to 1.
- When `OMn OLn` is not `00`, the associated pin `PTn` becomes an output tied to `OCn`, regardless of the state of `DDRT`.
- `TIOS`: Selects whether a pin is an Input Capture or Output Compare. 0 for Input Capture, 1 for Output Compare
- `TIE`: Timer Interrupt Enable Register. An Output Compare channel can request an interrupt when a comparison is successful. '1' to enable (Local Enable), '0' to disable.
- Reset `TCNT`: The `OC7` channel can reset `TCNT` when the `TC7=TCNT`. This is enabled by bit 3 (TCRE) of `TSCR2` register.

Example: Generate an active high 1-kHz digital waveform with a 30% duty cycle from the PT5 pin.

- 1KHz frequency amounts to a period of 1 ms. 30% duty cycle means that the signal is '1' for 300us and '0' for 700 us.
- With a pre-scale factor of 8, we have: $Timer\ clock = \frac{24\ MHz}{8} = 3\ Mhz \rightarrow Period = 0.33\ us = \frac{1}{3} us$. Then, we need to keep the signal high for HCYCLES = 900 cycles and low for LCYCLES = 2100 cycles.
- Wave generation: Signal set to high at the beginning, and then we add HCYCLES to TC5. When the comparison is successful, an interrupt is triggered. The ISR will add LCYCLES to TC5. When that comparison is successful, another interrupt is triggered. The ISR will add HYCLES to TC5. This will be repeated over and over.
- Process:
  1. Enable Output Compare on Channel 5: `TIOS(5)=1` → `TIOS = 0x20`
  2. Select OC5 action to pull to high: `TCTL1(3..2)=11` → `TCTL1 = 0x0C`
  3. Set Pre-scaler factor to 8. → `TSCR2=0x03`
  4. Enable timer counter (starts from 0). Enable fast clear for TOF and *C5F*. → `TSCR1 = 0x90`
  5. Clear all *CnF* flags (just in case) → `TFLG1 = 0xFF`
  6. Start an OC5 operation with a delay of 10 cycles: `TC5 = TCNT+10`. This is so that we start with 0 for just 10 cycles.
  7. Wait until `TLFG1(5) = 1`.
  8. Set OC5 pin to toggle. `TCTL1(3..2) = 01` → `TCTL1 = 0x04`
  9. Set new Output Compare operation with a delay of HCYCLES cycles: `TC5 = TC5+HCYCLES`
  10. HiLoflag = 0 (Global variable)
  11. Enable OC5 Interrupt:
      - Local Enable for OC5: `TIE(5)=1` → `TIE=0x20`
      - Global Enable: `asm("cli")`
- *Interrupt Service Routine*: The OC5 interrupt was enabled on Step 11. If HiLoflag = 0, then we add `LCYCLES` cycles to `TC5` and make HiLoflag=1. If HiLoflag=1, then we add `HCYCLES` cycles to `TC5` and make HiLoflag=0. Note that the addition will wraparound if it is larger than $2^{16} - 1$. Also, recall that TFLG1 is cleared every time we write on `TC5`.

C Code: `unit9c.c`

Notes about Digital Waveform Generation:

- We will use a Timer Clock period of $\frac{1}{3}us$ for the following results.
- The digital waveform created by the previous example has the following characteristics:
  - ✓ $HCYCLES, LYCLES$ determine the duty cycle. $Duty\ Cycle = \frac{HCYCLES}{TCYCLES} \times 100\%$
  - ✓ $TCYCLES = HCYCLES + LYCLES$ determines the period. $Frequency = \frac{1}{TCYCLES \times \frac{1}{3}us}$

TCYCLES	Frequency	HCYCLES	LCYCLES	Duty Cycle
3000	1 KHz	300	2700	10%
		600	2400	20%
		1500	1500	50%
		2400	600	80%
		2700	300	90%
1000	3 KHz	100	900	10%
		200	800	20%
		500	500	50%
		800	200	80%
		900	100	90%
500	6 KHz	50	450	10%
		100	400	20%
		250	250	50%
		400	100	80%
		450	50	90%
300	10 KHz	30	270	10%
		60	240	20%
		150	150	50%
		240	60	80%
		270	30	90%

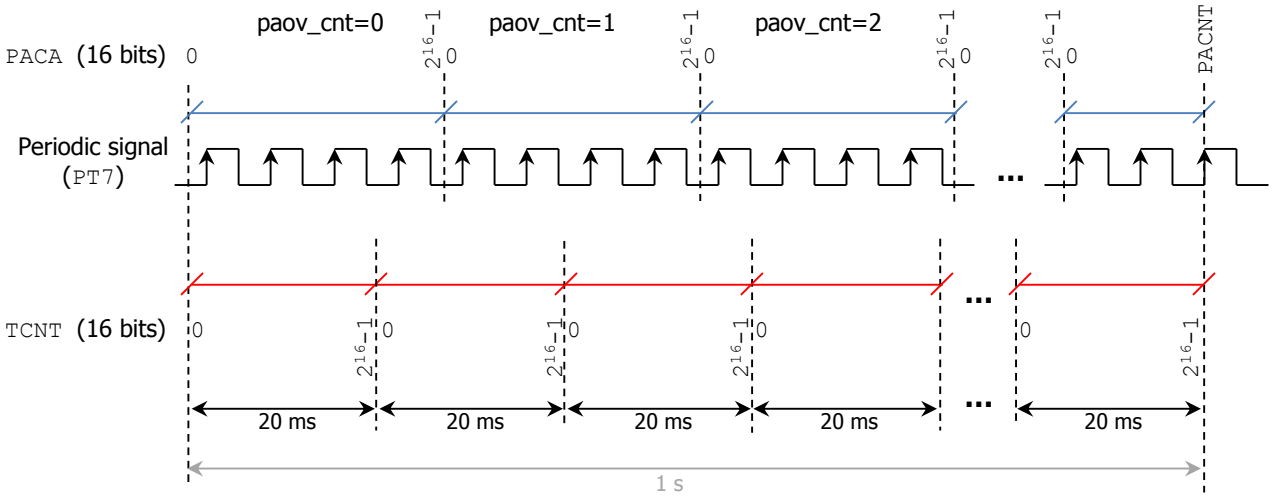
FORCED OUTPUT COMPARE:

- Useful if we want to immediately trigger an action on a particular Output Compare pin `PTn`.
- `CFORC` register: By writing a '1' on particular bit positions, the corresponding output compare channels will be forced (the particular PT pins will trigger an action).
- At the next Timer count after we write to `CFORC`, the forced channels will trigger the programmed pin actions to occur.
- The pin actions do not affect the timer flag (`TFLG1`) or generate an interrupt.

PULSE ACCUMULATOR

- It count the number of active edges (rising/falling edges) arriving on an input pin.
- HCS12D:
  - ✓ Four 8-bit Pulse Accumulators (`PAC3, PAC2, PAC1, PAC0`), or
  - ✓ Two 16-bit Pulse Accumulator (`PACA=|PAC3|PAC2|, PACB=|PAC1|PAC0|`)
- We can read and write on the `PACn` registers (n = 0, .. 3)
- Each Pulse Accumulator is assigned a particular input pin:
  - ✓ PT0 → PAC0, PT1 → PAC1, PT2 → PAC2, PT3 → PAC3
  - ✓ When PACA is used, PT7 is the input pin.
  - ✓ When PACB is used, PT0 is the input pin.
- Interrupts:
  - ✓ PT7 edge interrupt
  - ✓ PACA overflow
  - ✓ PACB overflow
- Operation modes:
  - ✓ Even-counting (rising/falling edges)
  - ✓ Gated-time accumulation mode (only for PACA).
- `TCTL4`: It controls the active edges selection for PAC0, PAC1, PAC2, PAC3, and PACB.
- `PACTL` register controls PACA operation. Bit 7 enables/disables PACA, bit 4 selects the active edge that cause the count to increment, bit 1 enables/disables the PAOV Interrupt (PACA overflow). PAFLG: tracks PACA status (overflow flag, pin flag). Writing '1' clears this register. If PAOV Interrupt is enabled, we usually clear this register in the ISR to keep PAFLG(1) from generating more interrupts.
- `PBCTL` register controls PACB operation. Bit 7 enables/disables PACB, bit 1 enables/disables the PBOV interrupt (PACB overflow). PBFLG records status similarly to PAFLG for PACA.

Example: Measuring frequency on the `PT7` pin. It also uses Output Compare for time reference and time delay.



- The 16-bit PACA will be used to count the number of rising edges (on `PT7`) in a 1 sec interval.
- The number of rising edges in a 1 sec interval will be most likely greater than 2^{16}, we keep track of the number of times the PACA counter overflows: We use the Pulse Accumulator A Overflow (PAOV) Interrupt.
- We need a time reference to create the 1 sec interval: we use the Output Compare Channel 0. With a pre-scale factor of 8, the Timer Clock period is $\frac{1}{3}us$. Thus, the largest frequency we can measure is 3 MHz. With a period of $\frac{1}{3}us$ we can create a delay of 60000 cycles, which results in a 20 ms delay. We need 50 of these delays for the 1 sec interval.
- At the end of the 1 sec time interval, the frequency is: $frequency = paov\_cnt \times 2^{16} + PACNT$

- Process:
  1. Enable Output Compare on Channel 0: `TIOS(0)=1` → `TIOS = 0x01`. Note that no action is set up on `PT0`.
  2. Set Pre-scaler factor to 8. → `TSCR2=0x03`
  3. Enable timer counter (starts from 0). Enable fast clear for TOF and `C0F`. → `TSCR1 = 0x90`
  4. Initialize PACA count: `PACN3 = PACN2 = 0x00`
  5. Enable PACA in event-counting mode for rising edges; enable PAOV Interrupt: `PACTL=0x52`.
  6. Configure `PT7` for input: `DDRT = 0x7F` (required when using the Pulse Accumulators)
  7. Global Enable Interrupts: `asm("cli")`
  8. oc_cnt = 50
  9. Add a delay of 60000 cycles to `TC0`: `TC0 = TCNT + 60000`
  10. while (oc_cnt ≠ 1) do:
      wait for `TLFG1=0x01`
      `TC0 = TCNT + 60000`
      oc_cnt = oc_cnt – 1
  11. Disable PA function: `PACTL=0x00`
  12. Disable interrupts: `asm ("sei")`
  13. $Frequency = paov\_cnt \times 2^{16} + PACNT,\ PACNT = PACN3 \times 0x100 + PACN2$

Interrupt Service Routine: The PAOV interrupt was enabled on Step 5. We clear the PAOVF flag and then increment the count on paov_cnt.

C Code: `unit9d.c`

MODULUS DOWN COUNTER

- MCCNT: 16-bit register. It counts in a downward fashion.
- It can generate an interrupt: Modulus Down Counter Underflow.
- Includes an independent clock with a pre-scaler (1,4,8,16).
- MCCTL: Controls operation:
  - ✓ Bit 7: enables/disables underflow interrupt.
  - ✓ Bit 6: If '0', the counter counts from a value written to MCCNT it down to `$0000`, and stops there. If '1', 'modulus mode' is enabled: when the counter reaches `$0000`, the next count is the latest value written into `MCCNT`.
  - ✓ Bit 3: enable bit. If '1', counter enabled; if '0', counter disabled and preset to `$FFFF`.
  - ✓ Bits 1 and 0 select the pre-scaler factor.
- MCTFLG: Modulus down counter Status Register. Bit 7: Underflow interrupt

Example: Generate periodic interrupts every 10 ms

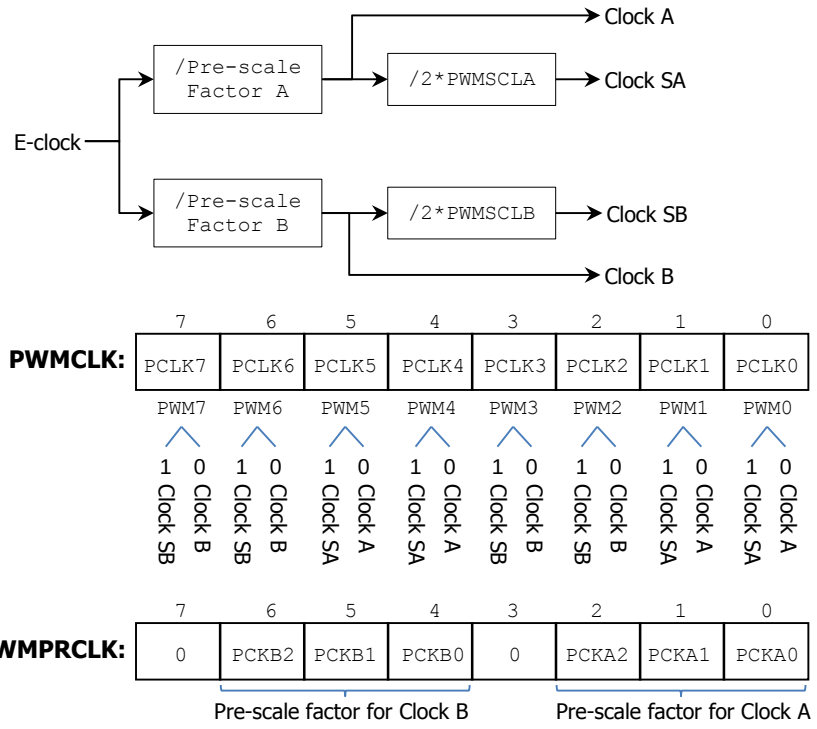
- We program a value VAL on the counter so that the count from VAL down to 0 amounts to a time interval of 10 ms. When '0' is reached, an interrupt will be generated, and the value VAL will be loaded again on the modulus down counter.
- If we use a pre-scale factor of 16, we have: $Timer\ clock = \frac{24\ MHz}{16} = 1.5\ Mhz \rightarrow Period = 0.67\ us = \frac{2}{3}us$. Then, we have: $VAL \times \frac{2}{3}us = 10ms \rightarrow VAL = 15000$.
- Process:
  1. Enable interrupt, enable 'modulus mode', enable counter, set pre-scale to 16. `MCCTL = 0xC7;`
  2. Set the counter to load 15000 every time it reaches 0. `MCCNT = 15000`
  3. Enable interrupts. → `asm("cli")`
- *Interrupt Service Routine*: Make sure to clear bit 7 of MCFLG by writing a 1 to it.

PULSE-WIDTH MODULATION (PWM)

- Creating a PWM signal using simple delay instructions requires frequent attention from the CPU (even if we use the Timer Output Compare Interrupt).
- The HCS12D contains a dedicated PWM Module. It can generate PWM signals without requiring frequent attention from the MCU.
- Eight PWM channels. Output pins: PORT P.
- Dragon12-Light Board: PP5 can be connected to the Buzzer via a Jumper.

PWM CLOCK SELECT

- E-clock: Input clock to the PWM Module.
- Each channel can select an specific clock using the `PWMCLK` and `PWMPRCLK` registers:



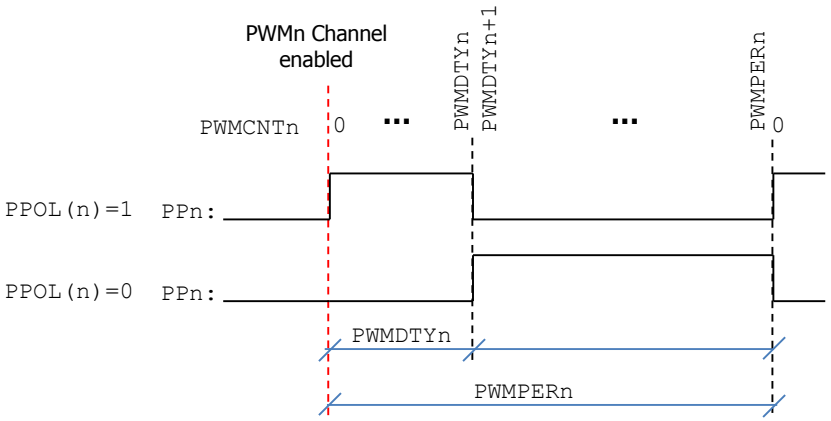
- Each PWM Channel (PWM7..0) is considered to be an 8-bit channel since an 8-bit counter is used to control the period and the duty cycle.
- `PWMCTL`: PWM Control Register. Bits 7-4 control whether all channels are independent 8-bit PWMs or whether we concatenate channels to create 16-bit PWMs. Bit 3 (PSWAI) enables(0)/disables(1) the PWM Module when the MCU is in wait mode. Bit 2 (PFRZ) enables(0)/disables(1) the E-clock when the MCU is in freeze mode.
  - ✓ Wait Mode: This is caused by the WAI (wait for interrupt) instruction. This instruction pushes the return address and CPU registers on the Stack and halts CPU execution. CPU resumes when an interrupt is detected. System clocks still work on this mode.
  - ✓ Freeze Mode: This is caused by the STOP instruction. This is similar to the WAI instructions, except that the system clocks are halted. Also, only RESET, /XIRQ, and /IRQ can resume CPU execution. The Bit S in CCR can enable(0)/disable(1) the STOP instruction.
- The input clock (E-clock) can also be disables for the PWM Module when all PWM Channels are disabled (`PWME=0x00`)

PWM CHANNELS:

- PWME: It enables (1)/disables(0) specific channels. This is regardless of the value of DDRP. Example: PWME=0xC0: Only Channels 7 and 6 are enabled.
- Each channel 'n' (n=0..7) contains:
  - ✓ An 8-bit counter: PWMCNTn
  - ✓ An 8-bit period register: PWMPERn
  - ✓ An 8-bit duty cycle register: PWMDTYn
- The counter runs at the rate of the selected clock source for the particular channel. Any value written to the 8-bit counter causes the counter to reset to 0x00 and to start counting up. Usually, we write 0x00 on this counter.
- PWMPOL: Polarity of each PWM channel. If '0', the PWM output starts with 0. If '1', the PWM output starts with '1'.
- PWMCAE: PWM Center Align Enable Register. It selects between 2 types of outputs: left aligned (0) and center aligned (1).

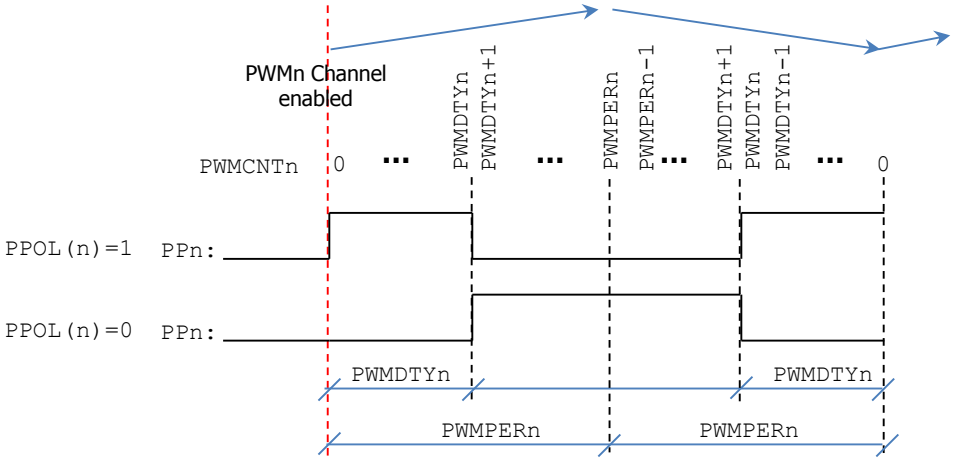
Left-aligned output: Up counter

- When the PWM channel is activated, the value on PWMCNTn is compared to the duty cycle register value (PWMDTYn) and the period register value (PWMPERn) at every clock cycle. A match between PWMCNTn and PWMDTYn causes the PWM waveform to toggle. A match between PWMCNTn and PWMPERn resets the counter, toggles the PWM waveform, and the process restarts.
- $PWMn\ frequency = \frac{PWM\ clock}{PWMPERn}$
- $PWMn\ Duty\ Cycle = \frac{PWMDTYn}{PWMPERn} \times 100\%,\ if\ PWMPOL(n) = 1$
- $PWMn\ Duty\ Cycle = \frac{PWMPERn - PWMDTYn}{PWMPERn} \times 100\%,\ if\ PWMPOL(n) = 0$



Center-aligned output: Up-down counter

- When the PWM channel is activated, the value on PWMCNTn is compared to PWMDTYn and PWMPERn at each clock cycle. The counter is first set to count up: a match between PWMCNTn and PWMDTYn causes the PWM waveform to toggle. A match between PWMCNTn and PWMPERn changes counter direction (down counter) and toggles the PWM waveform. When the decrementing counter matches PMDTYN again, the PWM waveform toggles. When the decrementing counter reaches 0, the counter direction changes to an up counter again, the PWM waveform toggles, and the process restarts.
- $PWMn\ frequency = \frac{PWM\ clock}{2 \times PWMPERn}$
- $PWMn\ Duty\ Cycle = \frac{PWMDTYn}{PWMPERn} \times 100\%,\ if\ PWMPOL(n) = 1$
- $PWMn\ Duty\ Cycle = \frac{PWMPERn - PWMDTYn}{PWMPERn} \times 100\%,\ if\ PWMPOL(n) = 0$



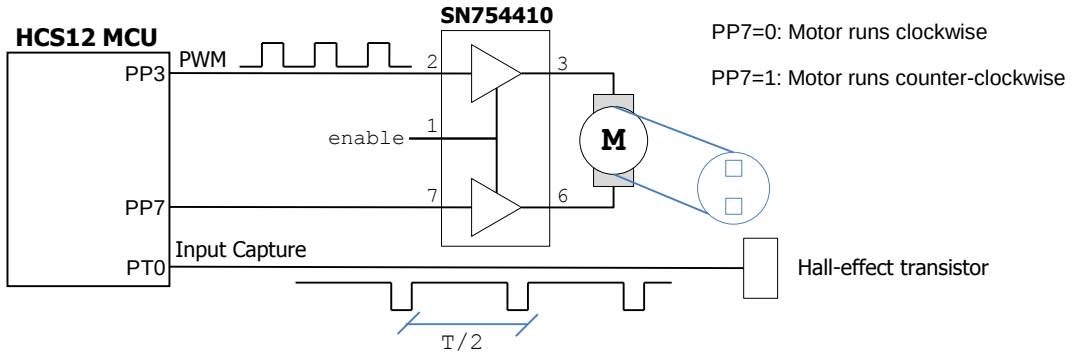
Example: PWM5 (PP5), 60% Duty Cycle, 100 KHz, E-clock = 24 MHz

Pre-scale factor for Clock A = 2: $PWM5\ clock = \frac{24\ MHz}{2} = 12MHz \rightarrow Period = \frac{1}{12}us$
PWM5 desired frequency is 100 KHz → PWM5 Waveform Period = 10 us
To get 10 us using a base period of 1/12 us, we need $\frac{10us}{1/12us} = 120\ cycles$
For 60% Duty Cycle, we need $120 \times 0.6 = 72$ cycles:

- Select Clock A as the clock source for PWM5: `PWMCLK = 0x00`
- Set clock A prescaler to 2: `PWMPRCLK = 0x01`
- Polarity of PWM5 set to '1': `PWMPOL = 0x20`
- Left aligned mode selected: `PWMCAE = 0x00`
- 8-bit individual PWMs enabled, stop PWM in wait and freeze mode: `PWMCTL = 0x0C`
- Set period value: `PWMPER5 = 120`
- Set duty cycle value: `PWMDTY5 = 72`
- Reset PWM5 counter: `PWMCNT5 = 0x00`
- Enable PWM Channel 5: `PWME = 0x20`

DC MOTOR CONTROL:

- DC motors: Speed can be controlled by changing the voltage level to the input of the motor. We can also control the direction of rotation by changing the polarity of the voltage applied to the motor.
- In a digitally controlled system, a voltage-controlled analog signal usually comes from a Digital-to-Analog Converter (DAC). Instead of using a DAC, we can use PWM so the average voltage controls the motor speed.
- The HCS12 cannot supply enough current to the DC motor nor enough voltage. A driver (e.g., SN754410) is used to supply the appropriate current. The SN754410 has 4 drivers. Each driver can supply 1A of current per channel (the HCS12 is usually in the tenths of mA), also they can provide voltage up to 36 V.
- The figure depicts how to control the speed of rotation, direction of rotation, and how to monitor the motor speed:
  - ✓ Speed of rotation (PP3). This is a PWM output. The duty cycle controls the speed. If PP7=0, the higher the duty cycle, the faster the speed.
  - ✓ Direction of Rotation (PP7). This is an ordinary output. If PP7=0, then the motor runs clockwise. If PP7=1, the motor runs counter-clockwise (here, the higher the duty cycle on PP3, the slower the speed).
  - ✓ Feedback line of motor (Input Capture Pin PT0): It provides information about the speed of the motor. A sensing device (optical encode, infrared detector, Hall-effect transistor) provides this information. The MCU can determine the speed and position of the motor in order to make adjustments: increase/decrease speed, reverse direction, stop motor. The example shows a particular configuration with a Hall-effect transistor and two magnets. Another typical configuration uses three Hall-effect transistors.



- If PP7=0, we can stop the motor if the PWM signal is 0. If PP7=1, we can stop the motor if the PWM signal is 1.
- The enable signal is usually set to 1. We can also control this signal. If enable is 0, the motor stops.
- Hall-effect transistor: It is mounted on the armature of the DC motor. Two magnets are mounted on the shaft. Every time the magnet passes by the Hall-effect transistor, a pulse is generated. These pulses are captured by the Input Capture. The time between two captures (T/2) is half of a revolution, this allows us to calculate the speed.
- In the figure, we need the two sides of the motor in order to control direction. But if we do not want to control direction, we can ground (or fix to power supply voltage) one of the ends of the motor. This way we avoid using a driver.