Lecture Summary – Module 3

Microcontroller Peripherals

Learning Outcome: an ability to effectively utilize microcontroller peripherals

Learning Objectives:

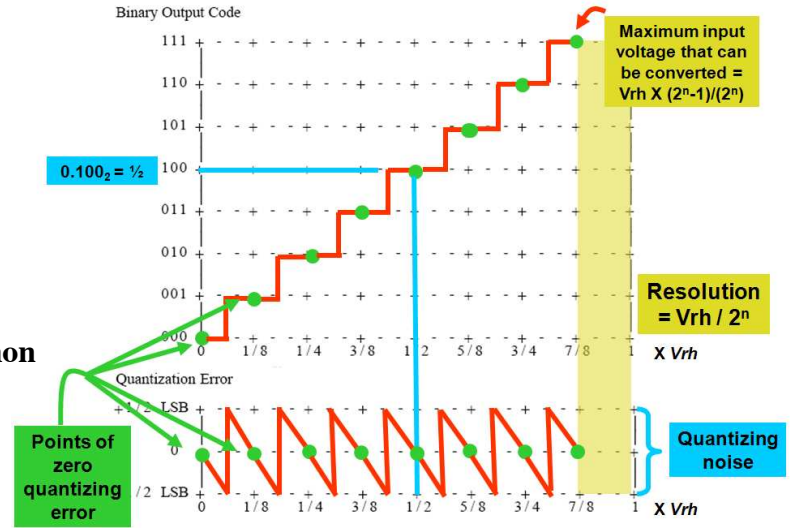
3-1. discuss key design considerations of a microcontroller-based data acquisition system: sampling, quantizing, and encoding
3-2. determine the sampling rate required (Nyquist rate) based on input signal spectral characteristics
3-3. describe aliasing and discuss how it can be prevented
3-4. estimate the dynamic range required based on converter resolution and calculate signal-to-quantizing-noise-ratio (SQNR)
3-5. describe the operation of a successive approximation analog-to-digital (ATD) converter
3-6. identify ATD features and operating modes
3-7. configure the ATD module to operate in a prescribed mode
3-8. create an ATD device driver routine for a prescribed application
3-9. describe asynchronous serial communication character frame format and justify the need for start/stop bits
3-10. compare non-return-to-zero (NRZ) serial data encoding with Manchester encoding
3-11. describe the difference between half-duplex with full-duplex communication
3-12. define baud rate and describe its relation to the local clock frequency
3-13. diagnose potential errors that can occur in asynchronous serial communications (parity, framing, overrun) and describe how they can be corrected
3-14. distinguish between single-ended and differential serial interfaces, and cite examples of each
3-15. describe the purpose and operation of a level translator (line driver) chip
3-16. identify SCI features and operating modes
3-17. configure the SCI to operate in a prescribed mode
3-18. create an SCI device driver routine for a prescribed application
3-19. describe the key features of synchronous serial communication
3-20. identify SPI features and operating modes
3-21. distinguish among MISO, MOSI, SISO, and MOMI entities as they pertain to the SPI module
3-22. configure the SPI to operate in a prescribed mode
3-23. create an SPI device driver routine for a prescribed application
3-24. describe the function of the TIM input capture mechanism
3-25. describe the function of the TIM output compare mechanism
3-26. describe the function of the TIM pulse accumulator mechanism
3-27. distinguish among input capture, output compare, and pulse accumulator timer applications
3-28. identify TIM features and operating modes
3-29. configure the TIM to operate in a prescribed mode
3-30. create a TIM device driver routine for a prescribed application
3-31. discuss why the TIM module can serve as a higher precision reference for timer applications than the RTI subsystem
3-32. define pulse width modulation (PWM) and describe potential applications
3-33. define natural sampling and contrast it with pulse-code modulation (PCM)
3-34. measure and compare the effects of different input and output sampling frequencies on signal integrity in an ATD-PWM digitization loop
3-35. describe how a PWM channel can be used as a digital-to-analog (DTA) converter
3-36. identify PWM features and operating modes
3-37. configure the PWM to operate in a prescribed mode

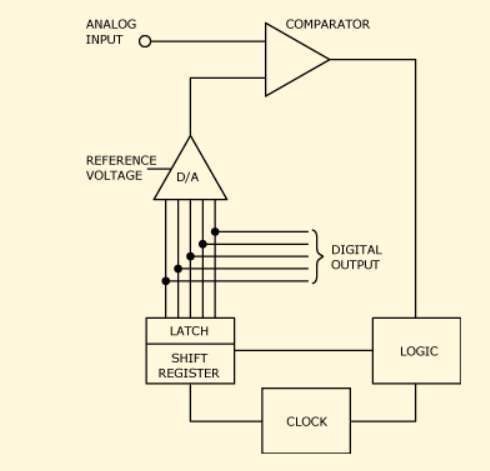
Lecture Summary – Module 3-A

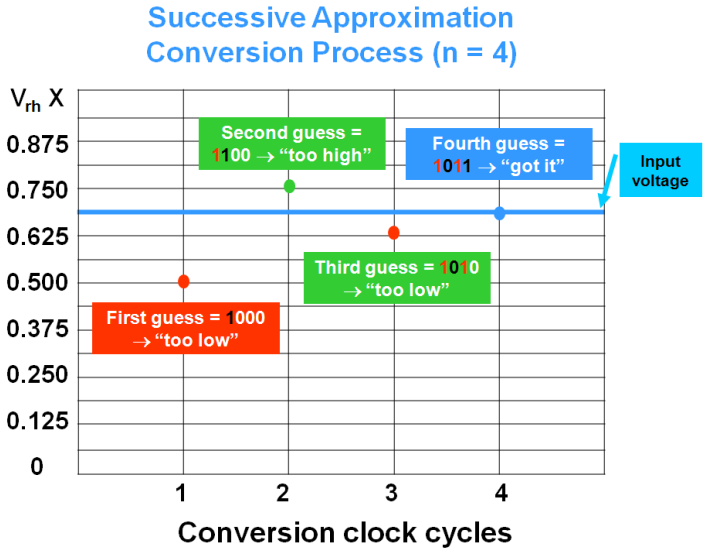
Analog-to-Digital (ATD) Converter

- **analog-to-digital conversion process**
  - **sampling**
  - **quantizing**
  - **encoding**
- **design issues**
  - **sampling rate (bandwidth)**
  - **number of bits (resolution)**
  - **converter type**
- **sampling**
  - **sample-and-hold circuit (analog memory)**
  - **sampling aperture = "shutter speed"**
  - **sampling rate (Nyquist)**
  - **anti-aliasing (low pass) filter**
  - **aliasing example**
- **quantization**
  - **number of bits needed - resolution**
  - **assignment of fixed amplitude level**
  - **relative to reference voltages**
  - **quantization error = noise**
  - **SQNR = 20 $\log_{10}(2^n / 1)$**
  - **about 6 dB of dynamic range/bit**
- **converter types**
  - **with or without DTA converter**
  - **successive approximation most common**
    - **high resolution**
    - **linear conversion time (n)**
    - **based on BGaN game**

Example in which higher frequency sinusoid aliases to a lower frequency

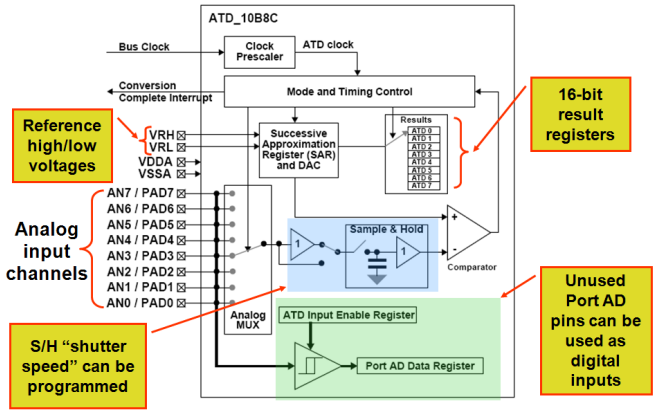


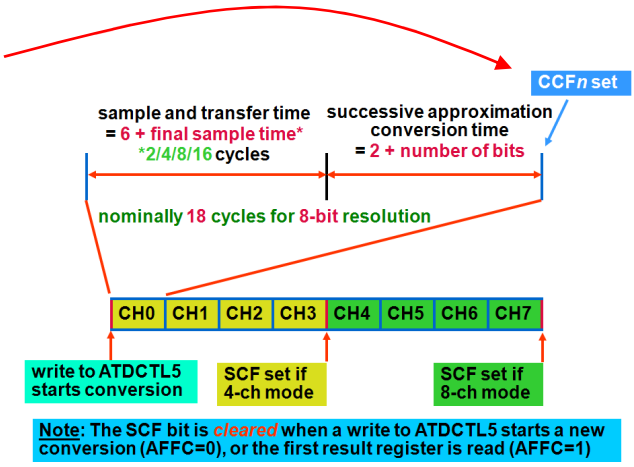




- **9S12C32 ATD features/modes**
  - **8 input channels**
  - **8/10 bit resolution**
  - **8 separate result registers**
  - **programmable sample time**
  - **fast conversion time (8 bits / 9 μs)**
  - **program- or interrupt-driven modes**
  - **port PAD pins also usable as digital inputs**
  - **conversion may be externally triggered**
- **operating modes**
  - **conversion sequence: 1 to 8 conversions**
  - **8 basic conversion modes**
    - **non-scan modes (program driven)**
    - **scan (continuous conversion → interrupt driven)**
    - **sequence complete flag (SCF) can be used as "polling" or interrupt device flag**
    - **each channel also has a conversion complete flag (CCF) which may be polled**
    - **conversion sequence is initiated by writing to ATD control register (ATDCTL5)**
- **reference voltages**
  - **V_{RH} (5 V) and V_{RL} (0 V)**
  - **resolution ("step size" or "one LSB change") = $(V_{RH} - V_{RL})/2^n$ (n is number of bits)**
- **control registers (mappings provided in skeleton file)**
  - **ATDCTL2**
    - **ADPU (bit 7) – overall enable**
      - **0 – ATD disabled**
      - **1 – ATD enabled**
    - **AFFC (bit 6) – "*fast* flag clear mode" (primarily useful for interrupt modes)**
      - **0 – "normal" CCF flag clearing mode (must read status register first before reading result reg.)**
      - **1 – "fast" CCF flag clearing mode (only need to read result reg - automatically clears CCF)**
    - **ASCIE (bit 1) – sequence complete interrupt enable**
      - **0 – ATD interrupt disabled**
      - **1 – ATD interrupt enabled**
    - **ASCIF (bit 0) – sequence complete interrupt device flag (read only bit)**
  - **ATDCTL3**
    - **(bits 6-3) – conversion sequence length**
    - **FIFO (bit 2) result register mode**
      - **0 – non-FIFO (results map into result registers based on *conversion sequence*)**
      - **1 – FIFO mode (conversion results placed in consecutive result registers modulo 8)**

ATD Block Diagram





S8C	S4C	S2C	S1C	Number of Conversions per Sequence
0	0	0	0	8
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	8

- **ATDCTL4**
  - **S10BM (bit 7) – 8/10 bit resolution mode selection**
    - **0 – 10-bit resolution**
    - **1 – 8 bit resolution**
  - **SMP (bits 6 & 5) – sample time select ("sampling aperture")**
    - **00 - 2 ATD clock periods ("nominal sampling aperture")**
    - **01 - 4 ATD clock periods**
    - **10 - 8 ATD clock periods**
    - **11 - 16 ATD clock periods**
  - **PRS (bits 4-0) – clock pre-scalar (max ATD clock freq is 2 MHz)**
    - **maximum internal ATD clock frequency is 2 MHz**
    - **set to "00101" (divide by 12) if bus clock is 24 MHz (max CPU clock)**
- **ATDCTL5 (writing to it starts conversion)**
  - **DJM (bit 7) – data justification**
    - **0 – left justified data**
    - **1 – right justified data**
  - **DSGN (bit 6) – signed/unsigned**
    - **0 – unsigned data**
    - **1 – signed (radix) data**
  - **SCAN (bit 5) – single/scan**
    - **0 – single conversion sequence performed**
    - **1 – scan mode (continuous conversion → use for interrupt modes)**
  - **MULT (bit 4) – multi-channel mode**
    - **0 – sample only one channel (multiple times, based on conversion sequence length specified in ATDCTL3)**
    - **1 – sample multiple (successive) channels, starting at the input channel selected in bits 2-0 (use if sampling more than one channel)**
  - **INPUT CHANNEL SELECT (bits 2-0) – channel to start sampling sequence on if "MULT" mode selected (bit 4 = 1)**

SRES8	DJM	DSGN	Result Data Formats Description and Bus Bit Mapping
1	0	0	8-bit / left justified / unsigned - bits 8-15
1	0	1	8-bit / left justified / signed - bits 8-15
1	1	X	8-bit / right justified / unsigned - bits 0-7
0	0	0	10-bit / left justified / unsigned - bits 6-15
0	0	1	10-bit / left justified / signed - bits 6-15
0	1	X	10-bit / right justified / unsigned - bits 0-9

Input Signal Vrl = 0 Volts Vrh = 5.12 Volts	Signed 8-Bit Codes	Unsigned 8-Bit Codes	Signed 10-Bit Codes	Unsigned 10-Bit Codes
5.120 Volts	7F	FF	7FC0	FFC0
5.100	7F	FF	7F00	FF00
5.080	7E	FE	7E00	FE00
2.580	01	81	0100	8100
2.560	00	80	0000	8000
2.540	FF	7F	FF00	7F00
0.020	81	01	8100	0100
0.000	80	00	8000	0000

slide 81 - A. ADR0 - NON- FIFO
slide 81 - D - FIFO

- **status registers**
  - **ATDSTAT (high byte)**
    - **SCF (bit 7) – sequence complete flag**
      - **main "handshaking" flag to use in a program-driven device driver**
      - **flag is cleared by:**
        - **writing a "1" to this bit -or-**
        - **starting a new conversion (by writing to ATDCTL5) -or-**
        - **reading a result register (if AFFC=1)**
    - **ETORF (bit 5) – external trigger overrun flag**
    - **FIFOR (bit 4) – FIFO overrun flag**
    - **CC (bits 2-0) conversion sequence counter status (read only)**
  - **ATDSTAT (low byte)**
    - **CCF (bits 7-0) – conversion complete flags for each result register**
- **port registers**
  - **ATDDIEN (digital input enable for PTAD pins)**
    - **0 – corresponding pin is analog input**
    - **1 – corresponding pin is digital input**
  - **PORTAD (data input) register – returns digital value on PTAD pin if corresponding ATDDIEN pin is "1"**
- **result registers (16-bit): ADR0 through ADR7**
- **initialization routine for program-driven operation**

```
; Initializes ATD for program-driven operation

; STEP (1) Enable ATD for normal flag clear
;         mode with interrupts disabled
atd_ini   movb    #$80,atdctl2

; STEP (2) Set the conversion sequence length to one
          movb    #$08,atdctl3

; STEP (3) Select 8-bit resolution and nominal sample time
          movb    #$85,atdctl4
          rts
```

- **program-driven device driver (see note on non-FIFO mode!)**

```
; Channel number passed in B register
; Converted sample returned in A register
; STEP (1) Start conversion on desired channel

inatd   andb #$07     ;mask unwanted bits
        stab atdctl5  ;start conversion on
                      ;selected channel in
                      ;"default" mode

; STEP (2) Wait for conversion sequence to finish
await   brclr   atdstat,$80,await

; STEP (3) Read result register and exit
        ldaa   adr0h ;(A)=(selected result)
        rts
```

Lecture Summary – Module 3-B

Serial Communications Interface (SCI)

- **background and motivation**
  - **parallel data transmission is expensive**
  - **solution: transmit data serially (ideally, over single "twisted pair")**
  - **non-return-to-zero (NRZ) serial bit stream**
  - **PISO and SIPO shift registers**
  - **half duplex vs. full duplex**
  - **issue of local clock synchronization**
- **asynchronous serial transmission**
  - **sample data stream at a substantially higher rate (typically 16x) than the data bit rate**
  - **add "start" bit (always 0) and "stop" bit (always 1) to each character frame**
  - **set timing reference based on "approximate middle" of start bit**
  - **resynchronize on each character frame (on each start bit received)**
  - **baud rate = *bits per second* transmitted over serial link**
  - **local clock frequency**
  - **tolerance between transmitting and receiving clocks**
  - **option of adding parity bit to each frame to test integrity of data received**
    - **even parity – parity bit set so even number of ones in data + parity**
    - **odd parity – parity bit set so odd number of ones in data + parity**
    - **marking parity – parity bit is always the same value**
  - **possible errors that can occur**
    - **framing error (don't get stop bit at end of frame)**
    - **receiver overrun (CPU doesn't read data from shift register before next character starts shifting in)**
    - **parity error (noise on transmission line corrupts one or more bits in data stream)**
  - **most likely causes of transmission errors**
    - **framing: local clocks too far out of tolerance (or, wrong baud rate at either end)**
    - **overrun: data not read by program "fast enough"**
    - **parity: problem with cable/connector**
- **9S12 SCI features**
  - **software-selectable baud rates**
  - **advanced data sampling technique (takes three successive samples in middle of each data bit – declared valid if two out of three of the samples agree, but "NF" flag set to indicate vote was not unanimous)**
  - **supports special single-wire and loop modes**
    - **wired-OR, single-wire modes (Tx/Rx pin is open drain, allowing use of single twisted pair in multi-drop[configuration)**
    - **loop mode (used for testing Tx/Rx when external device not available)**
  - **uses standard NRZ format**
  - **full duplex operation**
  - **programmable word length**
  - **allows parity generation and checking**
  - **can be program- or interrupt-driven**
  - **transmitter and receiver are double-buffered**

- **9S12 SCI registers**
  - **data registers**
    - **SCIDRL – transmit/receive data register (low byte)**
      - **read (only): receive data register (RDR)**
      - **write (only): transmit data register (TDR)**
    - **SCIDRH – high byte of data register (used solely for 9-bit modes)**

  - **control registers**
    - **SCICR1 – control register #1**
      - **LOOPS (bit 7) – loop mode enable**
      - **WOMS (bit 6) – wired OR mode (OD)**
      - **RSRC (bit 5) – receiver source for loop mode**
      - **M (bit 4) – mode (character format)**
        - **0 – one start bit + 8 data bits + one stop bit**
        - **1 – one start bit + 9 data bits + one stop bit**
      - **ILT (bit 3) – idle line type**
      - **WAKE (bit 2) – wakeup mode**
      - **PE (bit 1) – parity enable**
        - **0 – parity is disabled**
        - **1 – parity is enabled**
      - **PT (bit 0) parity type (applicable only if PE=1)**
        - **0 – even parity**
        - **1 – odd parity**
    - **SCICR2 – control register #2**
      - **TIE (bit 7) – transmit interrupt enable**
        - **0 – disabled**
        - **1 – enabled**
      - **TCIE (bit 6) – transmit complete interrupt enable**
      - **RIE (bit 5) – receive interrupt enable**
        - **0 – disabled**
        - **1 – enabled**
      - **ILE (bit 4) – idle interrupt enable**
      - **TE (bit 3) – transmitter (section of SCI) enable**
        - **0 – disabled**
        - **1 – enabled**
      - **RE (bit 2) – receiver (section of SCI) enable**
        - **0 – disabled**
        - **1 – enabled**
      - **RWU (bit 1) – receiver wakeup control**
      - **SBK (bit 0) – break control (a "break" is a string of zeroes)**
    - **SCIBDH and SCIBDL – baud rate control (two bytes)**
      - **SCIBDH (upper 4-bits of baud rate divisor)**
      - **SCIBDL (low byte of baud rate divisor)**
      - **baud_rate_divisor = ECLK / (16 x baud_rate)**
      - **exercise: calculate local clock error resulting from truncation**

- status registers
  - SCISR1 – status register #1
    - TDRE (bit 7) – transmit data register empty flag
      - 0 – *not* ready for new data (TDR "not empty")
      - 1 – ready for new data
      - TDRE flag is *cleared* by reading SCISR1 followed by writing to TDR (note: SCISR1 must be read before writing to TDR)
    - TC (bit 6) – transmit complete flag
    - RDRF (bit 5) – receive data register full flag
      - 0 –new data is *not* available
      - 1 – new data is available (RDR "full")
      - RDRF flag is *cleared* by reading SCISR1 followed by reading RDR
    - IDLE (bit 4) – idle line detected flag
    - OR (bit 3) – receiver overrun error flag
      - 0 – no overrun detected
      - 1 – overrun error detected
    - NF (bit 2) – noise flag
      - 0 – no noise detected in data bit sample
      - 1 – noise detected in data bit sample
    - FE (bit 1) – framing error flag
      - 0 – no framing error detected
      - 1 – framing error detected
    - PF (bit 0) – parity error flag
      - 0 – parity correct (or, parity disabled)
      - 1 – parity error detected
  - SCISR2 – status register #2
    - RAF (bit 0) – receiver active flag

- declarations, program-driven initialization, and device drivers

```
; Declarations

rxdrf   equ   $20   ; RDRF mask
txdre   equ   $80   ; TDRE mask
tron    equ   $0C   ; transmitter and
                    ; receiver enable mask
```

```
; Initializes SCI for program-driven operation

; STEP (1) Set transmission rate to 9600 baud
sci_pini   movb   #156t,scibdl

; STEP (2) Select character format (M bit) and parity
;      (bits 1 & 0) via SCI Control Register #1
           clr    scicr1

; STEP (3) Enable transmitter and receiver via SCI
;      Control Register #2 (with interrupts off)
           movb   #tron,scicr2
           rts
```

```
; inchar

; Inputs ASCII character from SCI port and
;    returns it in the A register
; No condition code bits or registers affected

; STEP (1) Check RDRF flag (wait until set)
inchar  brclr scisr1,rxdrf,inchar

; STEP (2) Load A with character in RDR
          ldaa  scidrl
          rts
```

```
; outchar

; Outputs character passed in A to the SCI port
; No condition code bits or registers affected

; STEP (1) Check TDRE flag (wait until set)
outchar brclr scisr1,txdre,outchar

; STEP (2) Load TDR with character in A register
          staa  scidrl
          rts
```

- **motivation for interrupt-driven mode**
  - **receive section**
    - **what a receive interrupt *means***
    - **rationale for *buffering* received data**
  - **transmit section**
    - **what a transmit interrupt *means***
    - **rationale for *buffering* data**
- **declarations and code**

```
; incBmod macro to increment (B) modulo
;   bufsize (substitution parameter)

incBmod   MACRO
          incb
          cmpb  #\1
          bne   $+3
          clrb
          ENDM
```

```
rxdrf   equ  $20   ; RDRF mask
txdre   equ  $80   ; TDRE mask
tron    equ  $0C   ; transmit/receive enable
rimask  equ  $20   ; receive  interrupt mask
timask  equ  $80   ; transmit interrupt mask

rsize  equ  40t  ; receive buffer size
rbuf   rmb  rsize
rin    fcb  0    ; receive buffer IN pointer
rout   fcb  0    ; receive buffer OUT pointer
tsize  equ  20t  ; transmit buffer size
tbuf   rmb  tsize
tin    fcb  0    ; transmit buffer IN pointer
tout   fcb  0    ; transmit buffer OUT pointer
```

```
;scbini (initializes SCI for interrupt-driven operation)
; STEP (1) Set transmission rate to 9600 baud
scbini movb  #156t,scibdl

; STEP (2) Select character format (M bit) and
; parity enable (and type, if parity enabled)
       clr   scicr1

; STEP (3) Enable transmitter and receiver,
; initially with receive interrupts enabled and
; transmit interrupts disabled
       movb  #tron,scicr2  ;enable trans/recv
       bset  scicr2,rimask ;enable recv intrs
       cli                 ;enable IRQ intrs
       rts
```

```
; sciisr (handles both receive and transmit sections of SCI in "polled" fashion)
; STEP (R1) Check RDRF flag
; STEP (R2) If RDRF=0, go to transmit section; else, continue with receive sec
sciisr brclr   scisr1,rxdrf,trans
; STEP (R3) Check status of RBUF
       ldab    rin
       incBmod rsize  ;(B) = (RIN+1) mod RSIZE
; STEP (R4) If FULL, error exit; else, continue
       cmpb    rout
       bne     rok
       swi
; STEP (R5) Input character from SCI RDR
rok    pshb           ; save (RIN+1) mod RSIZE
       ldaa  scidrl   ; read SCI RDR
; STEP (R6) Store character at RBUF[RIN]
       ldx   #rbuf
       ldab  rin
       staa  b,x      ; RBUF[RIN] = (RDR)
; STEP (R7) Increment RIN mod RSIZE
       pulb           ; restore from stack
       stab  rin      ; RIN = (RIN+1) mod RSIZE
; STEP (T1) Check TDRE flag
; STEP (T2) If TDRE=0, exit; else, continue
trans  brclr scisr1,txdre,scexit
; STEP (T3) Check status of TBUF
       ldab  tout
       cmpb  tin
; STEP (T4) If EMPTY, disable SCI transmit  interrupts and exit; else, continue
       bne   tok
       bclr  scicr2,timask ;disable trans intrs
       rti
; STEP (T5) Access character from TBUF
tok    ldx   #tbuf
       ldaa  b,x      ; (A) = TBUF[TOUT]
; STEP (T6) Output character to SCI TDR
       staa  scidrl   ; (TDR) = TBUF[TOUT]
; STEP (T7) Increment TOUT mod TSIZE
       incBmod  tsize
       stab     tout  ; TOUT=(TOUT+1) mod TSIZE
; STEP (T8) Exit
scexit rti
```

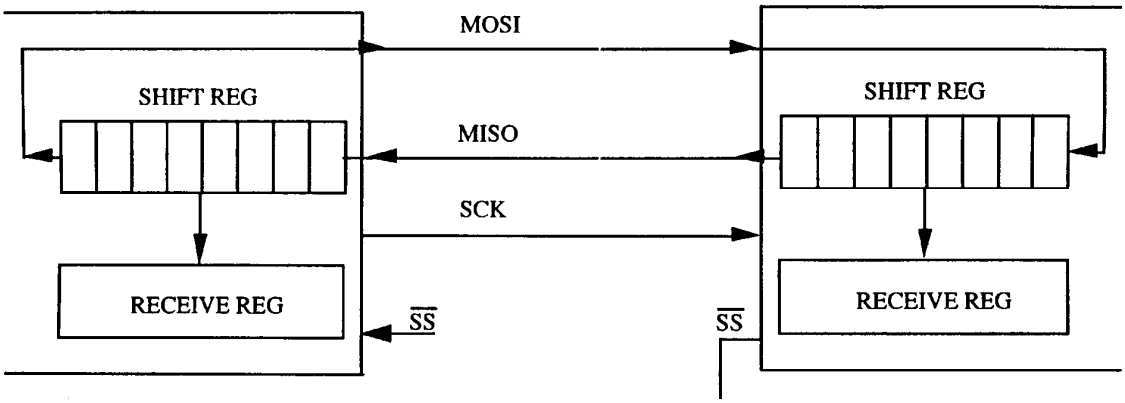
```
; bco (API routine that replaces inchar)
; Places character passed in A into TBUF
; STEP (1) Check TBUF status
bco    ldab    tin
       incBmod tsize  ;(B) = (TIN+1) mod
     TSIZE
       pshb           ;save on stack
; STEP (2) If FULL, wait for space
twait  cmpb    tout   ;perform FULL check
       beq     twait  ;wait if TBUF full
; STEP (3) Place character in TBUF
       ldx     #tbuf
       ldab    tin
       staa    b,x    ; TBUF[TIN] = (A)
; STEP (4) Increment TIN mod TSIZE
       pulb
       stab    tin
; STEP (5) Enable SCI transmit interrupts
       bset    scicr2,timask
; STEP (6) Exit
       rts
```

```
; bci (API routine that replaces outchar)
; Returns next character from RBUF in A register
; STEP (1) Check RBUF status
bci    ldab  rout
; STEP (2) If EMPTY, wait for character
rwait  cmpb  rin      ; perform EMPTY check
       beq   rwait    ; wait if RBUF empty
; STEP (3) Access next character from RBUF
       ldx   #rbuf
       ldaa  b,x      ;(A) = RBUF[ROUT]
; STEP (4) Increment ROUT modulo RSIZE
       incBmod  rsize
       stab  rout
; STEP (5) Exit
       rts
```

Lecture Summary – Module 3-C

Serial Peripheral Interface (SPI)

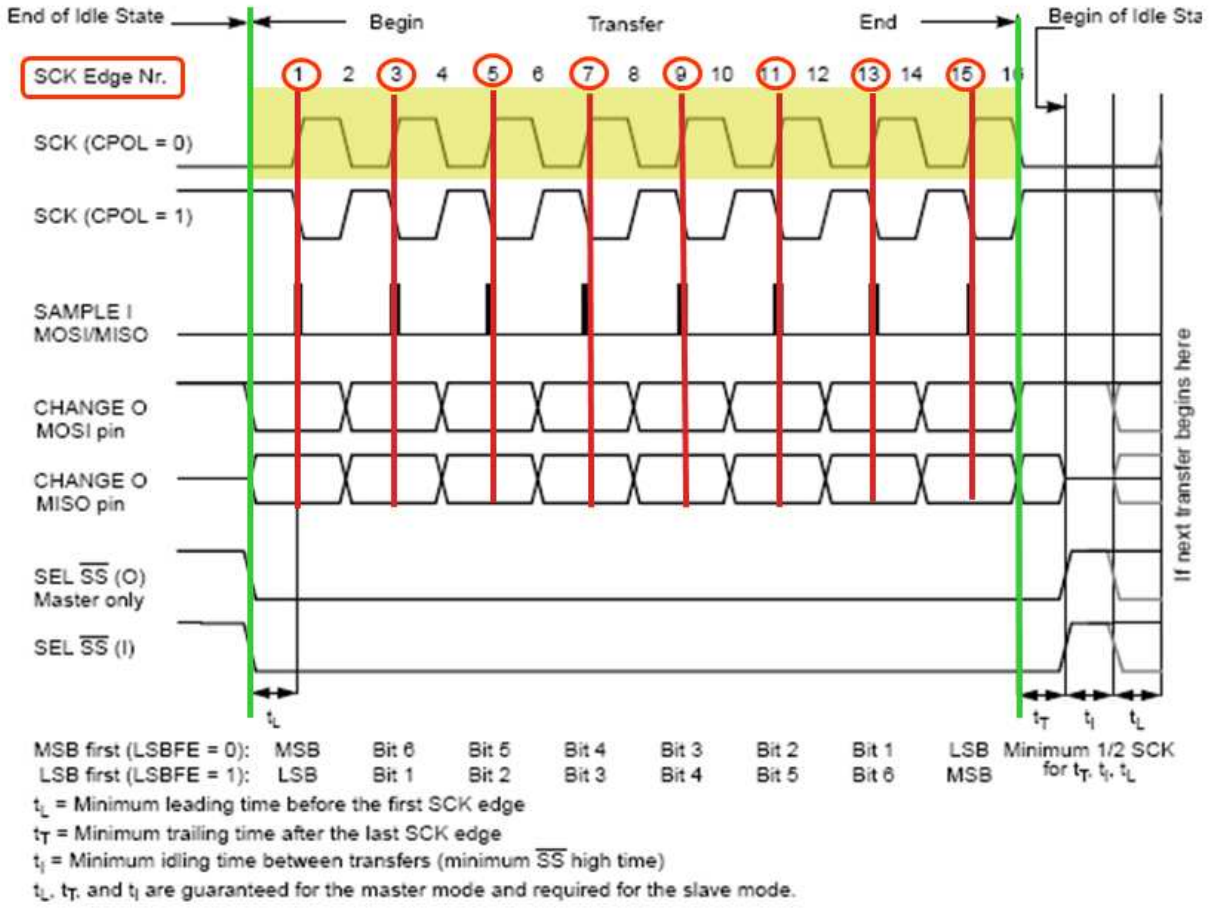
- **introduction**
  - **synchronous serial data transmission (faster – but requires common clock)**
  - **"master" and "slave" communicate by transmitting and receiving bit streams simultaneously (full duplex)**
  - **used in numerous applications (high speed data transfer with external peripherals)**



- **9S12 SPI features**
  - **flexible clock format**
  - **full or half duplex operation**
  - **program or interrupt driven**

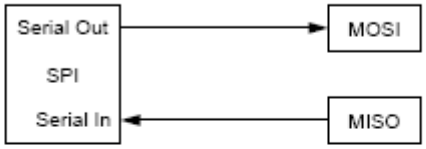
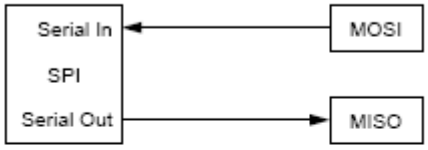
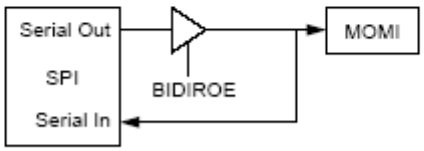
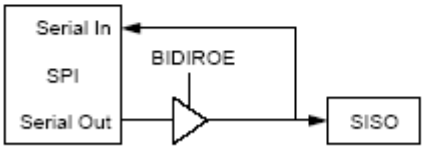
- **SPICR1 (control register #1)**
  - **SPIE (bit 7)**
    - **0 → SPI interrupts disabled**
    - **1 → SPI interrupts enabled**
  - **SPE (bit 6)**
    - **0 → SPI disabled ("power off")**
    - **1 → SPI enabled ("power on")**
  - **SPTIE (bit 5)**
    - **0 → SPI transmit empty interrupt disabled**
    - **1 → SPI transmit empty interrupt enabled**
  - **MSTR (bit 4)**
    - **0 → slave mode**
    - **1 → master mode**
  - **CPOL (bit 3)**
    - **0 → active high clock (SCK) – idles low**
    - **1 → active low clock (SCK) – idles high**
  - **CPHA (bit 2)**
    - **0 → data sampled at odd edges of SCK**
    - **1 → data sampled at even edges of SCK**
  - **SSOE (bit 1) – slave select (SS) output enable**
  - **LSBFE (bit 0)**
    - **0 → data transferred most significant bit first**
    - **1 → data transferred least significant bit first**

SPI Clock Format with CPOL=0, CPHA=0



t_L = Minimum leading time before the first SCK edge
t_T = Minimum trailing time after the last SCK edge
t_I = Minimum idling time between transfers (minimum $\overline{SS}$ high time)
t_L, t_T, and t_I are guaranteed for the master mode and required for the slave mode.

- **SPICR2 (control register #2)**
  - **MODFEN (bit 4) – "mode fault" detection enable**
  - **BIDIROE (bit 3) – MOSI/MISO output enable for bi-directional mode**
  - **SPC0 (bit 0) – bi-directional mode enable/disiable**

When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
Normal Mode SPC0 = 0		
Bidirectional Mode SPC0 = 1		

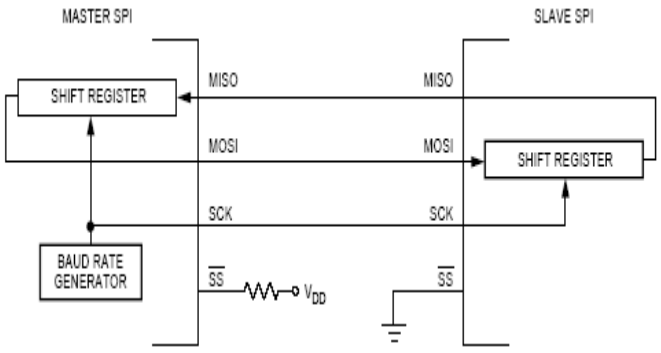
- **SPIBR (baud rate register)**
  - **SPRx (bits 2-0) – baud rate pre-selection bits**
  - **SPPRx (bits 6-4) – baud rate selection bits**

SPI BaudRate = BusClock / [(SPPR + 1) * $2^{(SPR+1)}$]

Example: if SPR=000 and SPPR=000 (default), BaudRate = BusClock / 2

- **SPISR (status register)**
  - **SPIF (bit 7) – data byte received ("receive data register full")**
    - **0 → transfer not yet complete**
    - **1 → received data available**
    - **SPIF is *cleared* by reading SPISR (status) followed by reading SPIDR (data)**
  - **SPTEF (bit 5) – transmit data register empty**
    - **0 → transmit data register not empty**
    - **1 → transmit data register is empty ("ready for next character")**
    - **SPTEF is *cleared* by reading SPISR (status) followed by writing SPIDR (data)**
  - **MODF (bit 4) – mode fault flag**
- **SPIDR (data register)**
  - **read: data received**
  - **write: data to transmit**
- **PORTM**
  - **MISO (bit 2)**
  - **SS (bit 3)**
  - **MOSI (bit 4)**
  - **SCK (bit 5)**

- **SPI interface to external "slave" device**



- **SPI initialization – program-driven mode**
- **SPI transmit/receive (I/O) routine**
- **Application: Lab 8 will utilize half-duplex communication with "slave" shift register that interfaces with LCD**

```
; Select 1.5 Mbps baud rate (24 MHz bus)
 spini  movb #$12,spibr

; Master mode, Interrupts off, CPOL=0,
; CPHA=0, slave select disabled, data
; transferred most significant bit first
        movb #$50,spicr1

; Normal (non-bidirectional) mode
        clr  spicr2
        rts
```

```
; Transmit data passed in A register
; Return data read in B register

; Transmit - wait for SPTEF to set
spio  brclr spisr,$20,spio
      staa  spidr     ;transmit data

; Receive - wait for SPIF to set
sprlp brclr spisr,$80,sprlp
      ldab  spidr       ;read data

      rts
```

Lecture Summary – Module 3-D

Timer (TIM)

- **definitions – key timer functions**
  - **output compare (event trigger)**
  - **input capture (time stamp)**
  - **pulse accumulator (event counter)**

- **9S12 TIM features and functions**
  - **8 channels – any mixture of functions**
  - **separate interrupt vector for each TIM channel**
  - **actions based on 16-bit free-running count register (TCNT)**
  - **Ch 7 is "special" – can generate precisely tuned periodic interrupts**
  - **single pulse accumulator (on Ch 7)**
  - **PA and OC functions can be used simultaneously on Ch 7**
  - **force output compare mechanism**

- **registers**
  - **TIOS (input capture/output compare select)**
    - **bit position in register corresponds to channel no.**
      - **0 → input capture mode**
      - **1 → output compare mode**

  - **TSCR1 (TIM system control register #1)**
    - **TEN (bit 7) – enable ("power on") bit**
      - **0 → disabled**
      - **1 → enabled**
    - **TFFCA (bit 4) – fast flag clear all**
      - **0 → normal**
      - **1 → IC read or OC write automatically clears corresponding channel flag**

  - **TSCR2 (TIM system control register #2)**
    - **TOI (bit 7) – timer overflow interrupt enable (TCNT wraps around)**
      - **0 → disabled**
      - **1 → interrupt generated when TOF flag set**
    - **TCRE (bit 3) – timer counter reset enable**
      - **0 → disabled (TCNT is free-running)**
      - **1 → TCNT reset when OC7 occurs**
    - **PR2-PR0 (bits 2-0) – clock prescalar**
      - **000 → prescale factor = 1**
      - **001 → prescale factor = 2**
      - **010 → prescale factor = 4**
      - **011 → prescale factor = 8**
      - **100 → prescale factor = 16**
      - **101 → prescale factor = 32**
      - **110 → prescale factor = 64**
      - **111 → prescale factor = 128**

- **registers, continued…**
  - **TCTL1:TCTL2 (TIM control regs. #1 & #2)**
    - **OM:OL bit pairs – one for each channel (pertains to OC)**
      - **00 → timer channel disconnected from port pin**
      - **01 → toggle port pin on OC**
      - **10 → clear port pin on OC**
      - **11 → set port pin on OC**
  - **TCTL3:TCTL4 (TIM control regs #3 & #4)**
    - **EDGEB:EDGEA bit pairs – one for each channel (pertains to IC)**
      - **00 → capture disabled**
      - **01 → capture on rising edges only**
      - **10 → capture on falling edges only**
      - **11 → capture on rising and falling edges**
  - **TIE (TIM interrupt enable)**
    - **CI bit position in register corresponds to channel no.**
      - **0 → interrupt disabled**
      - **1 → interrupt enabled**
  - **TFLG1 (TIM interrupt flag #1)**
    - **CF bit position in register corresponds to channel no.**
      - **0 → flag not set**
      - **1 → flag set (when IC or OC event has occurred)**
      - **flag is cleared by writing a "1" to corresponding bit position (normal mode), or by just reading (IC mode) or writing (OC mode) the channel register (fast flag clear mode)**
  - **TFLG2 (TIM interrupt flag #2)**
    - **TOF (bit 7)**
      - **0 → flag not set**
      - **1 → flag set (when TCNT "wraps around")**
      - **flag is cleared by writing a "1" to bit 7 of TFLG2 register**
  - **CFORC (TIM compare force)**
    - **FOC bit position in register corresponds to channel no.**
      - **0 → no action**
      - **1 → forces OC action to occur immediately (instead of waiting on TCNT)**
  - **OC7M (OC 7 mask)**
  - **OC&D (OC 7 data)**
  - **TTOV (TIM toggle on overflow)**
    - **TOV bit position in register corresponds to channel no.**
      - **0 → feature disabled**
      - **1 → in OC mode, port pin toggles when TCNT overflows**
  - **TC0-TC7 (IC/OC 16-bit register for each channel)**
  - **PORTT (TIM port register)**
  - **DDRT (TIM data direction register)**
    - **TIM "takes over" any pins on PORTT programmed for OC**

- sample application: digital clock
  - use Ch 7 with TCRE bit set
  - prescaler of 16 divides 24 MHz to 1.5 MHz
  - set TC7 to 15,000 → interrupt every 10 ms
  - 100 interrupts = 1.00 sec
- pulse accumulator (PA) features and functions
  - shares ch 7 port pin
  - functions independently of ch 7 OC
  - two basic modes
    - event counting
    - gated accumulation
- PA registers
  - PACTL (PA control)
    - PAEN (bit 6) – PA enable ("power up")
      - 0 → disabled
      - 1 → enabled
    - PAMOD (bit 5) – PA mode
      - 0 → event counter mode
      - 1 → gated time accumulation mode
    - PEDGE (bit 4) – edge control
      - event mode
        - 0 → count on falling edges
        - 1 → count on rising edges
      - gated mode
        - 0 → H on pin enables accumulation
        - 1 → L on pin enables accumulation
    - CLK1:CLK0 (bits 3:2) – clock select
      - 00 → use timer prescalar as TIM clock
      - 01 → use PACLK as TIM clock clock
      - 10 → use PACLK/256 as TIM clock clock
      - 01 → use PACLK/65536 as TIM clock clock
    - PAOVI (bit 1) – PA overflow interrupt enable
      - 0 → interrupt disabled
      - 1 → interrupt enabled
    - PAI (bit 0) – PA interrupt enable
      - 0 → interrupt disabled
      - 1 → interrupt enabled
  - PAFLG (PA flag)
    - PAOVF (bit 1) – PA overflow flag
      - 0 → no overflow
      - 1 → PACNT register has overflowed ("wrapped around")
      - writing a "1" to this bit position clears the PAOVF flag
    - PAIF (bit 0) – PA interrupt flag
      - 0 → no edge detected pn PA input pin
      - 1 → edge detected on PA input pin
      - writing a "1" to this bit position clears the PAIF flag

```
; Initializes TIM for Digital Clock application

; STEP (1) Enable TIM subsystem
tim_ini     movb    #$80,tscr1

; STEP (2) Set prescale factor to 16, and enable
;                 counter reset after OC7
                movb    #$0C,tscr2

; STEP (3) Set Ch 7 for Output Compare
                movb    #$80,tios

; STEP (4) Enable Ch 7 interrupt
                movb    #$80,tie

; STEP (5) Set up Ch 7 to generate 10 ms interrupt
    rate
                movw    #15000t,tch7

; STEP (6) Enable system IRQ interrupts
                cli
                rts
```

PA application: digital tachometer

- PA used in event counting mode
- RTI or TIM used to determine integration period for estimate
- pulse count converted to RPM (in BCD)

- **code listing for "not so quick" clicker quiz**

```
;*****************************************************************
; In this exercise we will investigate using the TIM module as
;    a time base, and compare it to use of the RTI subsystem.
;*****************************************************************
;
; The objective of this problem is to implement stopwatch that can count
; in increments of 0.1 second up to 999.9 seconds.
; Here, both the RTI and TIM modules will be used as time bases, with the
; opportunity to compare the results obtained using each approach.
;
; The following docking board resources will be used:
; - left pushbutton (PAD7): stopwatch reset
; - right pushbutton (PAD6): stopwatch start/stop
; - left LED (PT1): stopwatch run/stop state
; - right LED (PT0): stopwatch maxed out (999.9) state
;
; The four-digit stopwatch value (NNN.N) is to be updated on the terminal
; screen every one-tenth second (display both RTI and TIM values).
;*****************************************************************
;
; RTI and TIM initializations given in the class notes
;
; set 8.192 ms RTI interrupt rate
     movb    #$70,rtictl
; enable RTI interrupts
     bset    crgint,$80
; enable TIM subsystem
     movb    #$80,tscr1
; set TIM prescale factor to 16, and enable counter reset after OC7
     movb    #$0C,tscr2
; set TIM Ch 7 for Output Compare
     movb    #$80,tios

; enable TIM Ch 7 interrupt
     movb    #$80,tie
; set TIM Ch 7 to generate 10.0 ms interrupt rate
     movw    #15000,tc7
; enable IRQ interrupts
     cli
;  If the "run/stop" flag is set, then
;    - If the "RFLG" flag is set, then
;      + clear the "RFLG" flag
;      + increment the RTI stopwatch value by one tenth (in BCD)
;    - Endif
;    - If the "TFLG" flag is set, then
;      + clear the "TFLG" flag
;      + increment the TIM stopwatch value by one tenth (in BCD)
;      + update the display
;    - Endif
;  Endif
main1
      brclr  RUNSTP,$01,main2

      brclr  RFLG,$01,main12
      clr    RFLG
      ldaa   rtime+1
      adda   #1
      daa
      staa   rtime+1
      ldaa   rtime
      adca   #0
      daa
      staa   rtime
```

- **code listing for NSQCQ, continued…**

```
;  If the "run/stop" flag is set, then
;    - If the "RFLG" flag is set, then
;      + clear the "RFLG" flag
;      + increment the RTI stopwatch value by one tenth (in BCD)
;    - Endif
;    - If the "TFLG" flag is set, then
;      + clear the "TFLG" flag
;      + increment the TIM stopwatch value by one tenth (in BCD)
;      + update the display
;    - Endif
;  Endif
main12
       brclr   TFLG,$01,main2
       clr     TFLG
       ldaa    ttime+1
       adda    #1
       daa
       staa    ttime+1
       ldaa    ttime
       adca    #0
       daa
       staa    ttime
       jsr     rtdisp

main2
;  If the left pushbutton ("reset stopwatch") flag is set, then:
;    - clear the left pushbutton flag
;    - clear the "run/stop" flag
;    - turn off both the "run/stop" and "maxed out" LEDs
;    - reset the terminal display to "000.0"
;  Endif
       brclr   LEFTPB,$01,main3
       clr     LEFTPB
       clr     RUNSTP
       clr     PTT
       clr     rtime
       clr     rtime+1
       clr     ttime
       clr     ttime+1
       jsr     rtdisp

main3
;  If the right pushbutton ("start/stop") flag is set, then
;    - clear the right pushbutton flag
;    - toggle the "run/stop" flag
;    - toggle the "run/stop" LED
;  Endif
       brclr   RGHTPB,$01,main4
       clr     RGHTPB
       ldaa    RUNSTP
       eora    #$01
       staa    RUNSTP
       ldaa    PTT
       eora    #$02
       staa    PTT

main4
;  If the TIM stopwatch has reached "999.9", then:
;    - clear the "run/stop" flag
;    - turn on the "time expired" LED
;    - turn off the "run/stop" LED
;  Endif
       ldaa    ttime
       cmpa    #$99
       bne     main41
       ldaa    ttime+1
       cmpa    #$99
       bne     main41
       clr     RUNSTP
       movb    #$01,PTT
main41
       jmp     main1   ; continue looping
```

- **code listing for NSQCQ, continued…**

```
;*****************************************************************
;
; TIM interrupt service routine
;
; Interrupt generated every 10 ms on TC7
;
; Use variable TIMCNT to count up to 10 of these
; and set TFLG when that happens
;
; NOTE: If the "runstp" flag is clear (stopwatch is "stopped")
;       this routine should DO NOTHING other than clear
;       the TC7 interrupt flag
tim_isr
        bset TFLG1,$80    ; clear TC7 interrupt flag
        brclr     RUNSTP,$01,tdone
        ldaa timcnt
        inca
        staa timcnt
        cmpa #10          ; count to 100 ms (10 x 10ms)
        blt  tdone
        clr  timcnt
        movb #$01,TFLG    ; set tenth-of-second flag
tdone
        rti

;*****************************************************************
; RTI interrupt service routine
;
;  Keeps track of when 0.1 second of RTI interrupts has accumulated
;    and sets RFLG
;
;  Also, samples state of pushbuttons (PTAD7 = left, PTAD6 = right)
;  If change in state from "high" to "low" detected, set pushbutton flag
;     LEFTPB (for PTAD7 H -> L), RGHTPB (for PTAD6 H -> L)
;     Recall that pushbuttons are momentary contact closures to ground
rti_isr
;  Using RTICNT, track when 0.1 second of RTI interrupts has accumulated
;  Set the "RFLG" flag when this occurs and clear RTICNT

       bset   crgflg,$80
       brclr  RUNSTP,$01,chkpb
       inc    rticnt
       ldaa   rticnt
        cmpa  #12    ; based on initial 8.192 ms interrupt rate
        blt   chkpb
       clr    rticnt
       movb   #$01,RFLG
;  Check the pushbuttons for a change of state (compare previous with current)
;  If no change detected, exit service routine
chkpb
       ldaa   PTAD   ; read current state of pushbuttons
       anda   #$C0   ; mask off PTAD7 and PTAD6
       cmpa   prevpb ; compare with previous PB state
       bne    didchg ; if current state != previous state
       rti           ; figure out which pushbutton involved
                     ; else, exit with no change
;  State of PB changed -- check if action necessary
;  Note: Not considering case when both PBs pressed simultaneously
; (more code)
rdone
       pula
       staa   prevpb   ; update PB state
       rti
```

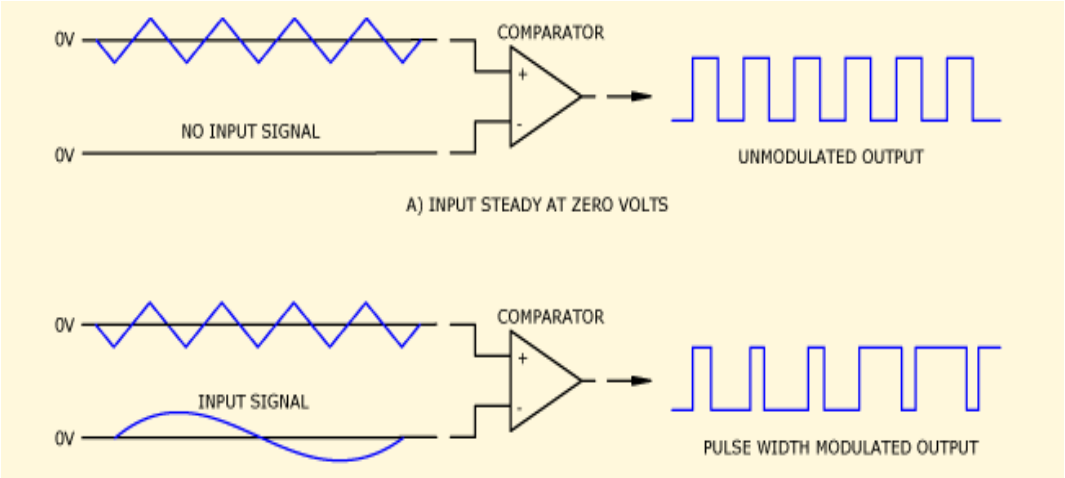
Lecture Summary – Module 3-E

Pulse Width Modulation (PWM)

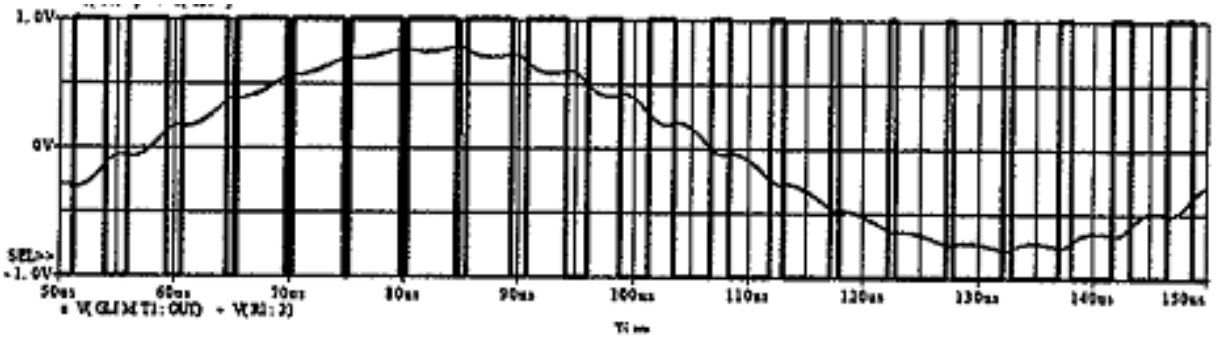
- **provides capability of producing square waves of varying *duty cycle* and *sampling frequency***
  - **0% duty cycle = always off**
  - **50% duty cycle = symmetric square wave**
  - **100% duty cycle = always on**

- **applications**
  - **D.C. motor speed control**
  - **LED backlight intensity control**
  - **DTA conversion (need LP filter)**

- **PWM encoding – natural sampling (*vs.* PCM = "pulse code modulation")**

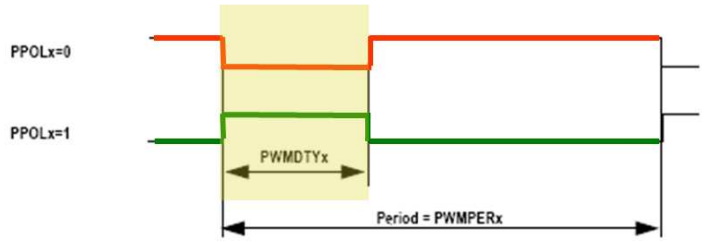


- **ATD – signal reconstruction – obtained when low-pass filter PWM output (sampling rate needs to be about an *order of magnitude higher* than the highest frequency component)**



- **9S12 PWM features and operating modes**
  - **six 8-bit channels with programmable period and duty cycle**
  - **pairs of adjacent channels can be combined to create a 16-bit channel**
  - **flexible clock generation**
  - **period and duty cycle registers are double-buffered (helps prevent "glitches" or discontinuities in PWM signal generated)**
  - **programmable output polarity and alignment**
  - **"emergency shutdown" capability**

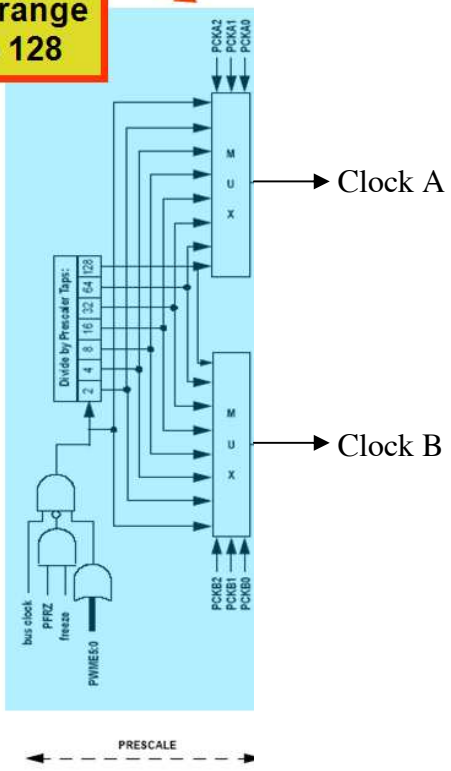
- **PWM registers**
  - **PWME (PWM channel enable) – corresponding bit for each channel**
    - **0 → disabled**
    - **1 → enabled**
  - **PWMPOL (PWM polarity select) – corresponding bit for each channel**
    - **0 → active low**
    - **1 → active high**
  - **PWMPER0..5 (period) – number of clock cycles that constitute *one complete period***
  - **PWMDTY0..5 (duty cycle) – number of clock cycles the PWM signal is *asserted***
  - **PWMPRCLK (prescale clock select)**
    - **PCKB2..0 (bits 6-4) – Clock B prescalar**
    - **PCKA2..0 (bits 2-0) – Clock A prescalar**
      - **000 – bus clock**
      - **001 – bus clock / 2**
      - **010 – bus clock / 4**
      - **011 – bus clock / 8**
      - **100 – bus clock / 16**
      - **101 – bus clock / 32**
      - **110 – bus clock / 64**
      - **111 – bus clock / 128**
  - **PWMCLK (clock source select)**
    - **PCLK5 (bit 5) – channel 5 clock select**
      - **0 → use Clock A**
      - **1 → use Clock SA**
    - **PCLK4 (bit 4) – channel 4 clock select**
      - **0 → use Clock A**
      - **1 → use Clock SA**
    - **PCLK3 (bit 3) – channel 3 clock select**
      - **0 → use Clock B**
      - **1 → use Clock SB**
    - **PCLK2 (bit 2) – channel 2 clock select**
      - **0 → use Clock B**
      - **1 → use Clock SB**
    - **PCLK1 (bit 1) – channel 1 clock select**
      - **0 → use Clock A**
      - **1 → use Clock SA**
    - **PCLK0 (bit 0) – channel 0 clock select**
      - **0 → use Clock A**
      - **1 → use Clock SA**



Duty Cycle (%) = 100 * (PWMDTY)/(PWMPER)

Sampling Frequency = (Input Clock)/(PWMPER)

Pre-scale divisor range is 1 to 128



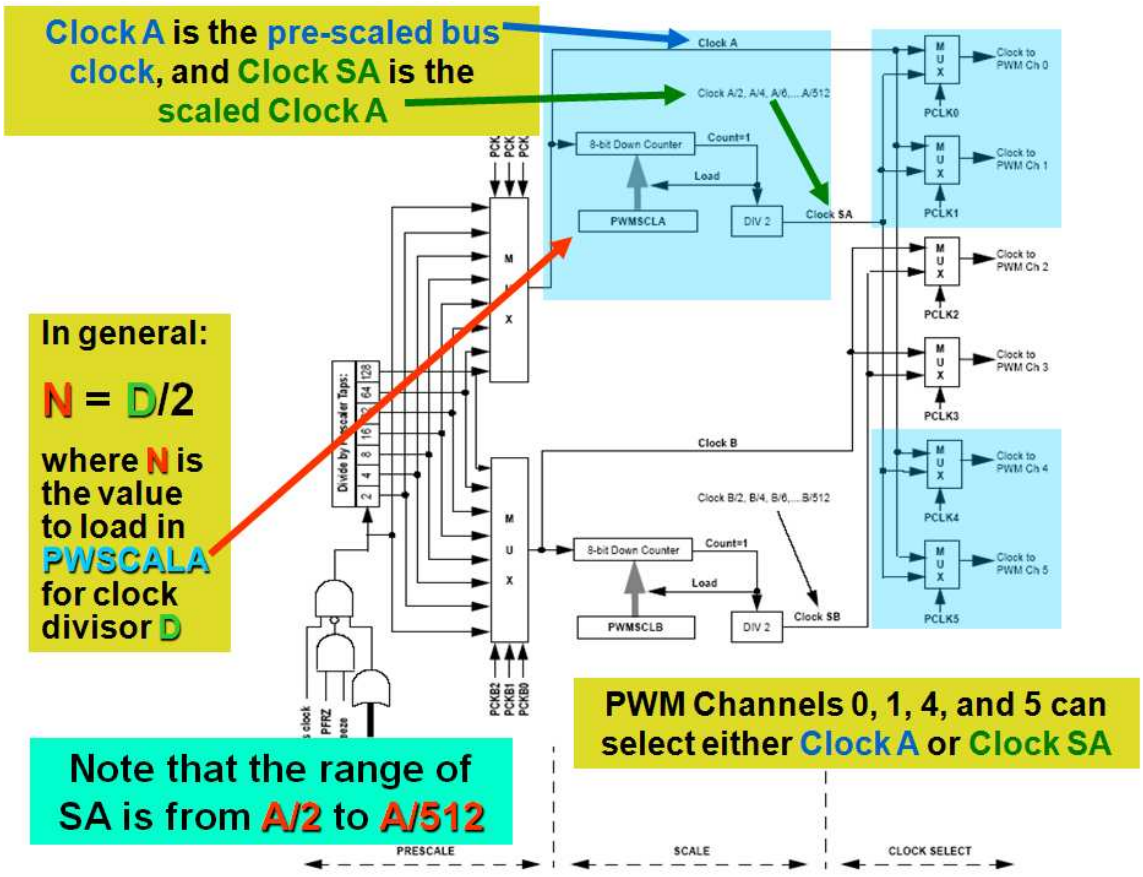
The "A" clocks are used by channels 0, 1, 4, and 5

The "B" clocks are used by channels 2 and 3

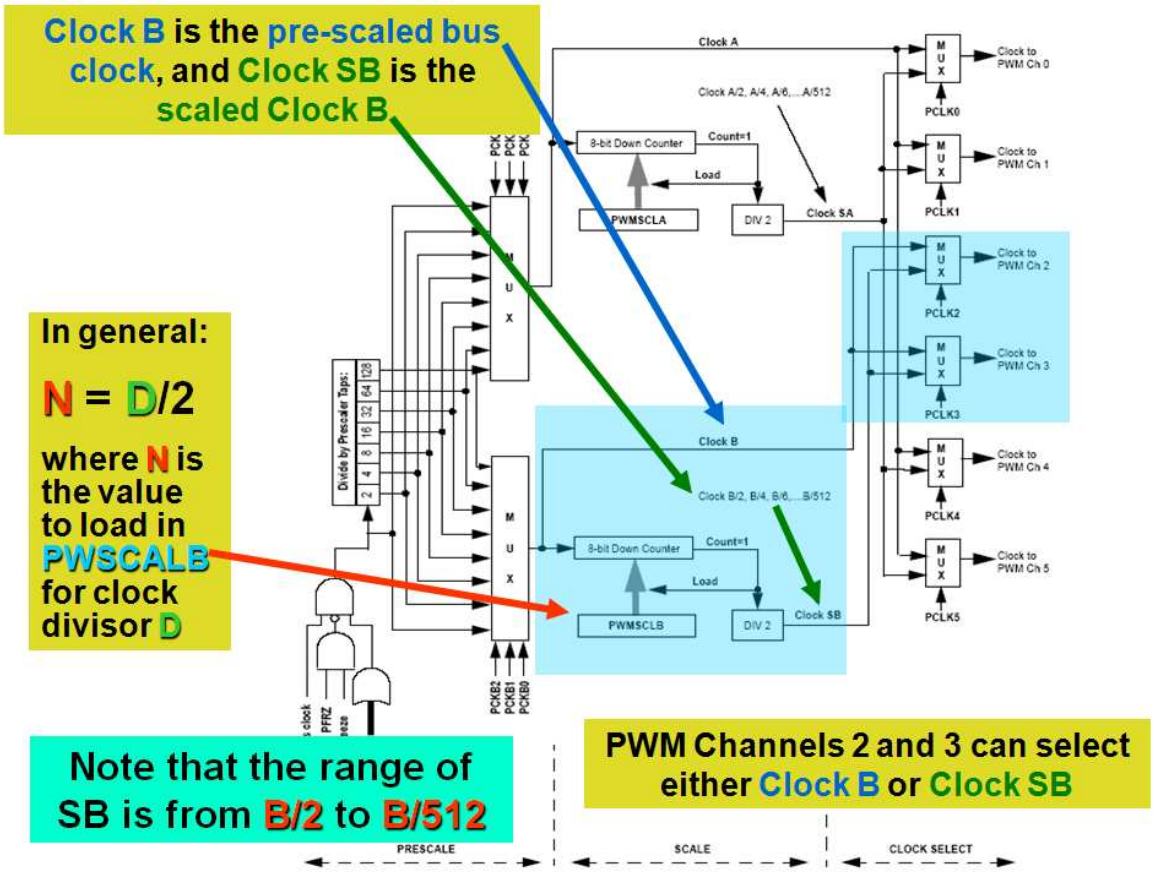
"Clock A" and "Clock B" are the *pre-scaled* clocks

"Clock SA" and "Clock SB" are the *scaled* clocks

- **the "A" clocks**

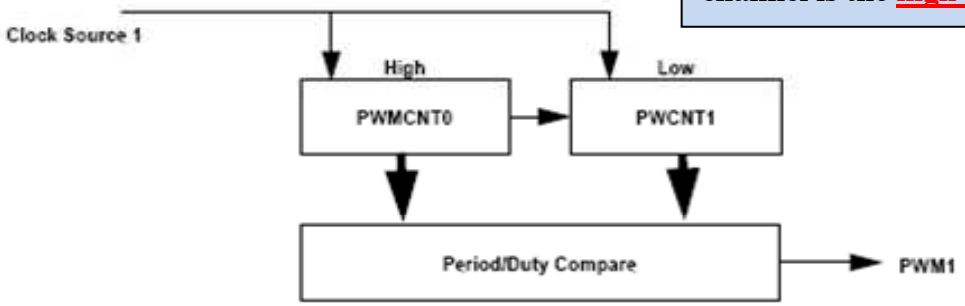


- **the "B" clocks**

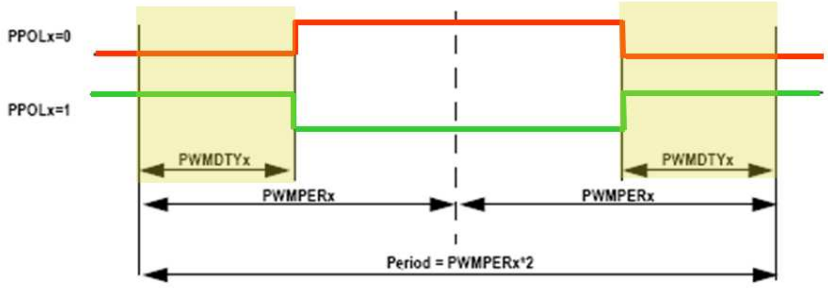


- **PWM registers, continued…**
  - **PWMSCLA (Scale A register)**
    - **8-bit programmable scale value for Clock A**
    - **Clock SA = Clock A / (2 * PWMSCLA)**
    - **$00 represents 256_{10}**
  - **PWMSCLB (Scale B register)**
    - **8-bit programmable scale value for Clock B**
    - **Clock SB = Clock B / (2 * PWMSCLB)**
    - **$00 represents 256_{10}**
  - **PWMCTL (PWM control)**
    - **CON45 (bit 6) – concatenate chs 4 & 5 for 16-bit mode**
      - **0 → normal 8-bit mode**
      - **1 → concatenated 16-bit mode**
    - **CON23 (bit 5) – concatenate chs 2 & 3 for 16-bit mode**
      - **0 → normal 8-bit mode**
      - **1 → concatenated 16-bit mode**
    - **CON01 (bit 4) – concatenate chs 0 & 1 for 16-bit mode**
      - **0 → normal 8-bit mode**
      - **1 → concatenated 16-bit mode**

In 16-bit ("concatenated") mode, the *lower numbered* channel is the high byte



  - **PWMCAE (center align enable) – corresponding bit for each channel**
    - **0 → operate in left-aligned mode (default)**
    - **1 → operate in center-aligned mode**



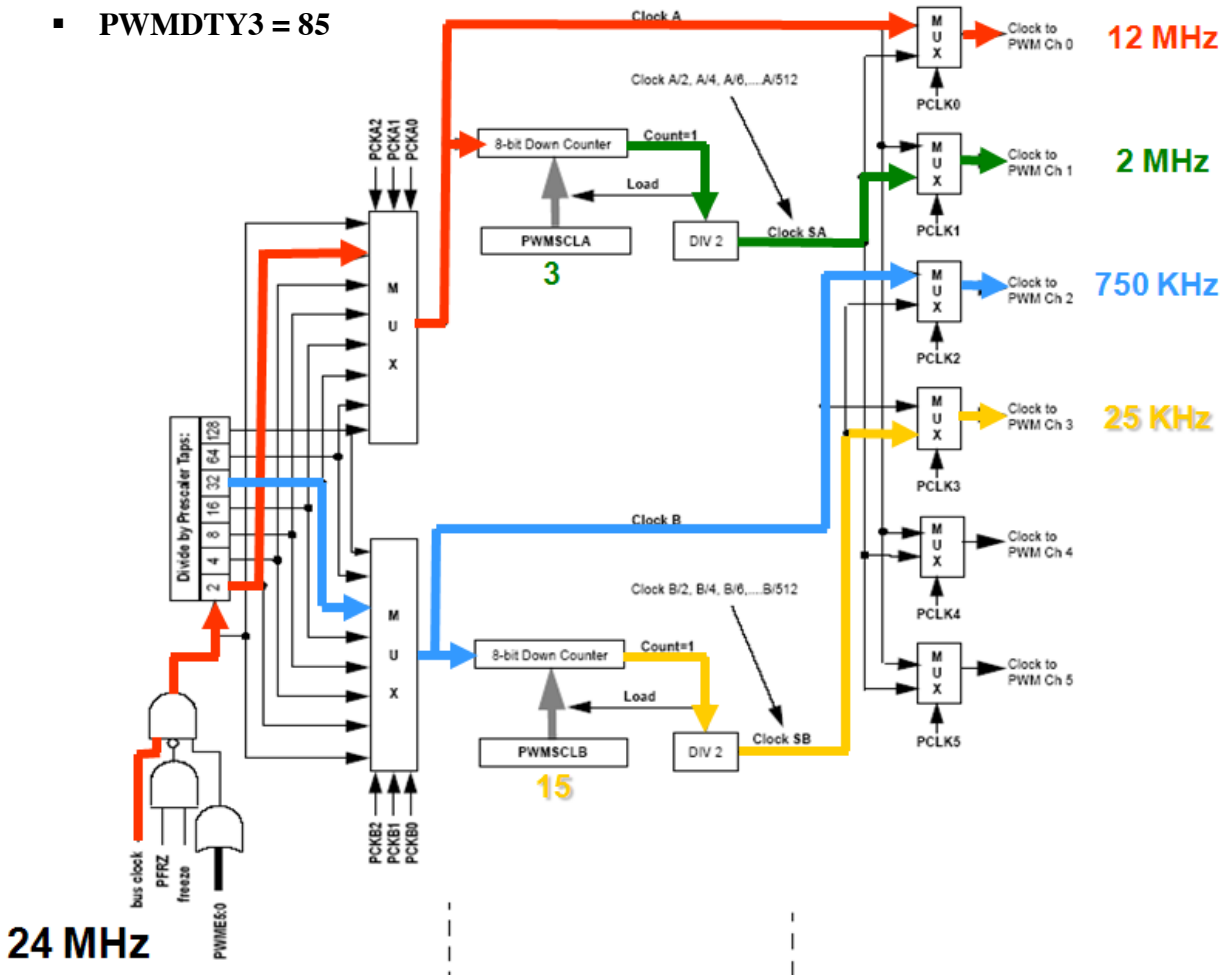
Useful in brushless motor speed control applications

Duty Cycle (%) = 100 * (PWMDTY)/(PWMPER)

Sampling Frequency = (Input Clock)/(2* PWMPER)

  - **PWMSDN (PWM shutdown) – used for (externally triggered) emergency shutdown**
  - **PORTP (bits 0-5) – used as PWM outputs for enabled channels**
  - **DDRP – data direction register for PORTP**

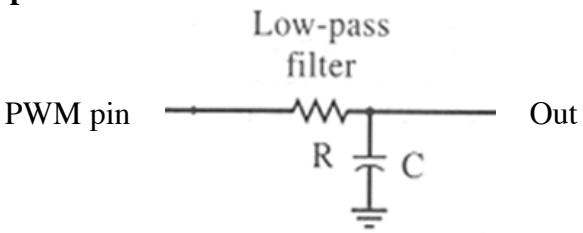
- **PWM initialization example**
  - **bus clock = 24 MHz**
  - **desired outputs (use left-aligned mode)**
    - **ch 0 – 120,000 Hz, 10%**
    - **ch 1 – 20,000 Hz, 30%**
    - **ch 2 – 7,500 Hz, 70%**
    - **ch 3 – 250 Hz, 85%**
  - **pick period (PMWPER) to be 100 clock ticks (gives 1% resolution)**
    - **means ch 0 clock freq = 120,000 ∗ 100 = 12 MHz → prescale of 2**
    - **means ch 1 clock freq (relative to ch 0) scaled by a factor of 6 (12,000,000 / 6 = 2,000,000) → scale A register set to half that value (3)**
    - **means ch 2 clock freq = 7,500 ∗ 100 = 750 KHz → prescale of 32**
    - **means ch 3 clock freq (relative to ch 2) scaled by a factor of 30 (750,000 / 30 = 25,000) → scale B register set to half that value (15)**
  - **duty cycles (PWMDTY) simply set to the desired duty cycle percentage (all base 10)**
    - **PWMDTY0 = 10**
    - **PWMDTY1 = 30**
    - **PWMDTY2 = 70**
    - **PWMDTY3 = 85**



- **code listing for clicker quiz**

```
;*************************************************************
; Analyze the following code
;*************************************************************
; initialize PWM Ch 0
       movb   #$01,MODRR    ; PT0 used as PWM Ch 0 output
       movb   #$01,PWME     ; enable PWM Ch 0
       movb   #$01,PWMPOL   ; set active high polarity
       movb   #$00,PWMCTL   ; no concatenate (8-bit)
       movb   #$00,PWMCAE   ; left-aligned output mode
       movb   #$FF,PWMPER0  ; set maximum 8-bit period (255)
       movb   #$7F,PWMDTY0  ; set 50% duty cycle
       movb   #$07,PWMPRCLK ; set Clock A = 24 MHz / 128
       movb   #$01,PWMCLK   ; select Clock SA for Ch 0
       movb   #$00,PWMSCLA  ; set Clock SA scalar to 512
; initialize TIM Ch 1
       movb    #$80,tscr1   ; enable TC1
       movb    #$07,tscr2   ; set TIM pre-scale factor to 128
       movb    #$02,tios    ; set TIM TC1 for OC mode
       movw    #$0004,tctl1       ; toggle PT1 on successful OC
       movw    #$0000,tc1   ; value for OC1
eloop  bra    eloop         ; infinite (do-nothing) loop
```

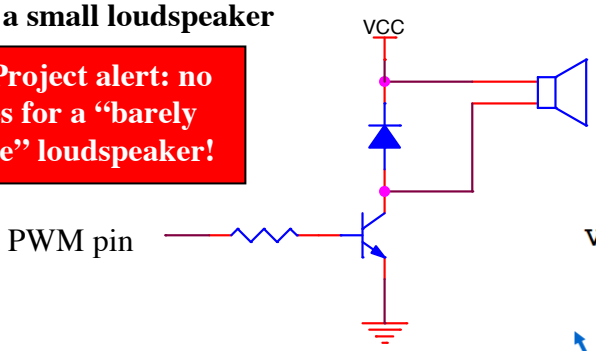
- **applications/interfaces**
  - **simple DTA**



$f_{-3\ dB} = 1 / 2\pi RC$

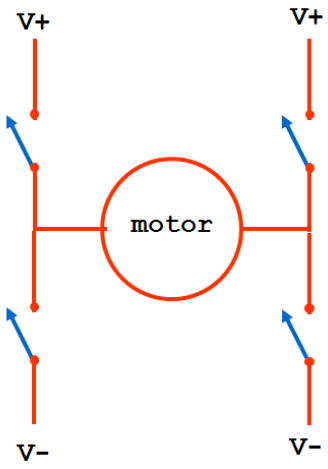
  - **driving a small loudspeaker**

Mini-Project alert: no excuses for a "barely audible" loudspeaker!



Loudspeaker will "mechanically" low-pass filter the PWM signal

  - **motor speed (and direction) control**
    - **use PWM control speed**
    - **use generic port pin to control direction via an "H bridge"**



Clicker Quiz Supplement

- **code listing for clicker quiz #1**

```
;*****************************************************************
; Analyze the following code
;*****************************************************************

; initialize PWM Ch 0
      movb    #$01,MODRR     ; PT0 used as PWM Ch 0 output
      movb    #$01,PWME      ; enable PWM Ch 0
      movb    #$01,PWMPOL    ; set active high polarity
      movb    #$00,PWMCTL    ; no concatenate (8-bit)
      movb    #$00,PWMCAE    ; left-aligned output mode
      movb    #$FF,PWMPER0   ; set maximum 8-bit period (255)
      movb    #$7F,PWMDTY0   ; set 50% duty cycle
      movb    #$07,PWMPRCLK  ; set Clock A = 24 MHz / 128
      movb    #$01,PWMCLK    ; select Clock SA for Ch 0
      movb    #$00,PWMSCLA   ; set Clock SA scalar to 512

; initialize TIM Ch 1
      movb     #$80,tscr1    ; enable TC1
      movb     #$07,tscr2    ; set TIM pre-scale factor to 128
      movb     #$02,tios     ; set TIM TC1 for OC mode
      movw     #$0004,tctl1  ; toggle PT1 on successful OC
      movw     #$0000,tc1    ; value for OC1

elp   bra      elp           ; infinite loop
```

- **code listing for clicker quiz #2**

```
; initialize ATD (8 bit, unsigned, nominal sample time, seq length = 2)
        movb    #$80,ATDCTL2    ; power up ATD
        movb    #$10,ATDCTL3    ; set conversion sequence length to TWO
        movb    #$85,ATDCTL4    ; select resolution and sample time

; initialize PWM Ch 0 (left aligned, positive polarity, max 8-bit period)
        movb   #$01,MODRR       ; PT0 used as PWM Ch 0 output
        movb   #$01,PWME        ; enable PWM Ch 0
        movb   #$01,PWMPOL      ; set active high polarity
        movb   #$00,PWMCTL      ; no concatenate (8-bit)
        movb   #$00,PWMCAE      ; left-aligned output mode
        movb   #$FF,PWMPER0     ; set maximum 8-bit period
        movb   #$00,PWMDTY0     ; initially clear DUTY register
        movb   #$00,PWMPRCLK    ; set Clock A = 24 MHz (prescaler = 1)
        movb   #$01,PWMCLK      ; select Clock SA for Ch 0
        movb   #$01,PWMSCLA     ; set Clock SA scalar to 2

; initialize TIM Ch 7 (periodic 0.1 ms interrupt)
        movb    #$80,TSCR1      ; enable TC7
        movb    #$0C,TSCR2      ; set TIM prescale factor to 16
                                ; and enable counter reset after OC7
        movb    #$80,TIOS       ; set TIM TC7 for Output Compare
        movb    #$80,TIE        ; enable TIM TC7 interrupt
        movw    #150,TC7        ; set up TIM TC7 to generate 0.1 ms interrupt rate

; enable IRQ interrupts
        cli        ; enable IRQ interrupts

; do nothing after initialization complete
elp    bra    elp ; infinite loop

; TIM interrupt service routine
; Interrupt generated every 0.1 ms (default - may be changed)
; Initiate ATD conversion on Chs 0 and 1
; Multiply analog input sample by potentiometer sample -> PWM Ch 0
;
tim_isr
        bset    tflg1,$80       ; clear TC7 interrupt flag
        movb    #$10,ATDCTL5    ; start conversion on ATD Ch 0
                                ; NOTE: "mult" (bit 4) needs to be set
await   brclr  ATDSTAT,$80,await
        ldaa    ATDDR0          ; analog input sample
        ldab    ATDDR1          ; potentiometer sample
        mul                     ; multiply input sample X pot setting
        adca #0                 ; round upper 8 bits of result
        staa    PWMDTY0         ; copy result to PWM Ch 0
        rti
```