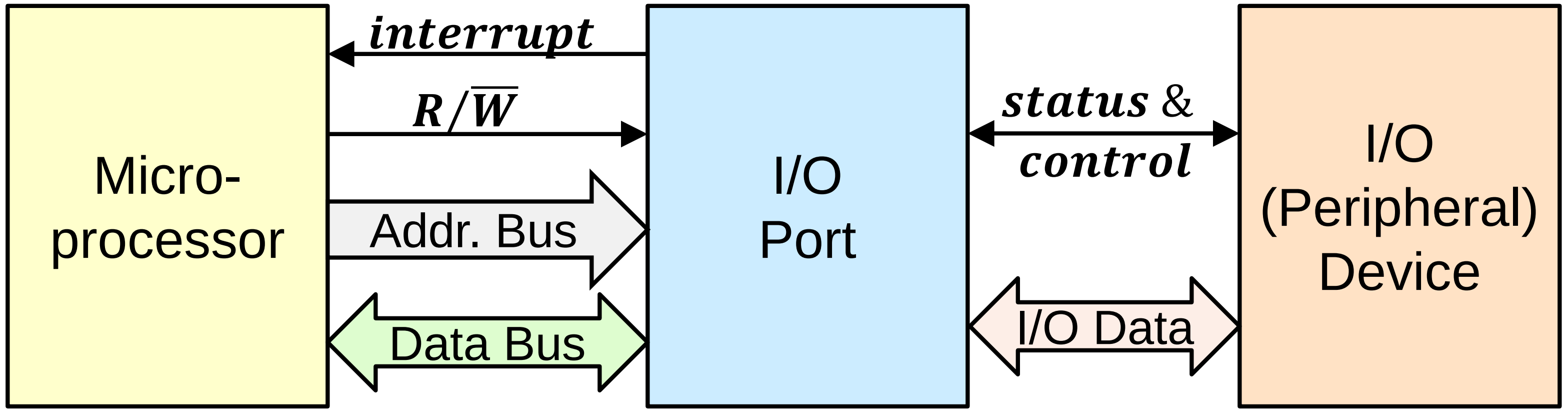
Lecture 11
Parallel Ports

Bryan Ackland
Department of Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ 07030

Adapted from HCS12/9S12 An Introduction to Software and Hardware Interfacing   Han-Way Huang, 2010

HCS12 Parallel Ports

- HCS12 is a microcontroller
  - CPU + on-chip memory + on-chip I/O devices and interfaces
  - On HCS12, I/O interfaces are called I/O Ports
  - I/O Ports appear as registers mapped into memory address space of microprocessor



  - I/O Device may be on-chip (part of the microcontroller) or off-chip – accessed though microcontroller's I/O pins
  - Unlike memory, I/O devices are not always ready to send to receive data
  - Need for some synchronization protocol

Synchronizing Microprocessor and I/O Port

- **Brute Force method**
  - No checking of whether I/O device is ready
  - For input – the CPU simply reads Port Data Register, assuming that there is a valid piece of data in the register waiting to be read
  - For output – the CPU simply writes the Port Data Register, assuming that the I/O device is ready to respond to this new value
  - Useful when timing is not important (e.g. LEDs, switches)
- **Polling Method**
  - The I/O Port maintains status bits (flags) to indicate the readiness of the I/O device to send or receive new data
  - For input, the CPU checks a “data ready” flag. When set this indicates that there is a new piece of input data available
  - For output, the CPU checks a “busy” flag. When set this indicates that the I/O device is still busy dealing with the previous value sent by the CPU. The CPU waits for the “busy” flag to clear
  - Simple to program, but the CPU spends most of its time checking these flags (polling) – not available to do other useful work

Synchronizing Microprocessor and I/O Port (2)

- **Interrupt method**
  - For input - the I/O Port interrupts the microprocessor whenever it has received new data from the input device
  - For output - the I/O Port interrupts the microprocessor whenever it is ready to receive new data from the microprocessor
  - More complex to code and debug – deadlocks, untested sequences, stack underflow/overflow etc.
  - Interrupt overhead with high speed I/O devices
  - No wasted polling cycles – more efficient use of CPU resources – CPU can do non-I/O related work while waiting for peripheral

Overview of HCS12 Ports

- HCS12 members have 48-144 I/O pins (depending on package) arranged to serve 3-12 Parallel Ports (A thru T)

- Limited pin-out means that pins have multiple functions

- Associated with each Port is an on-chip peripheral function
  - When peripheral function is not enabled, pins associated with that port are available for general purpose I/O
  - When peripheral function is enabled, pins are not available for general purpose I/O but reassigned to support that peripheral function

HCS12 I/O Ports

Port Name	#pins	Pin Names	Data Register Name	Alternate (Peripheral Function) Use of I/O Pins
A	8	PA7~PA0	PTA	Upper address & data bits
B	8	PB7~PB0	PTB	Lower address & data bits
E	8	PE7~PE0	PTE	Bus control and Ext. interrupt
H	8	PH7~PH0	PTH	Serial Peripheral Interface (SPI)
J	4	PJ7,6,1,0	PTJ	I^2C and CAN (auto) Network
K	6	PK5~PK0	PTK	Expanded address bits
M	8	PM7~PM0	PTM	CAN, SPIO and I^2C
P	8	PP7~PP0	PTP	Pulse Width Modulation
S	8	PS7~PS0	PTS	Serial Communications Interface
T	8	PT7~PT0	PTT	Timer IOC functions
AD0,1	16	PAD15~PAD0	PTAD0,1	A/D Converter(s)

Parallel Ports - General Purpose I/O

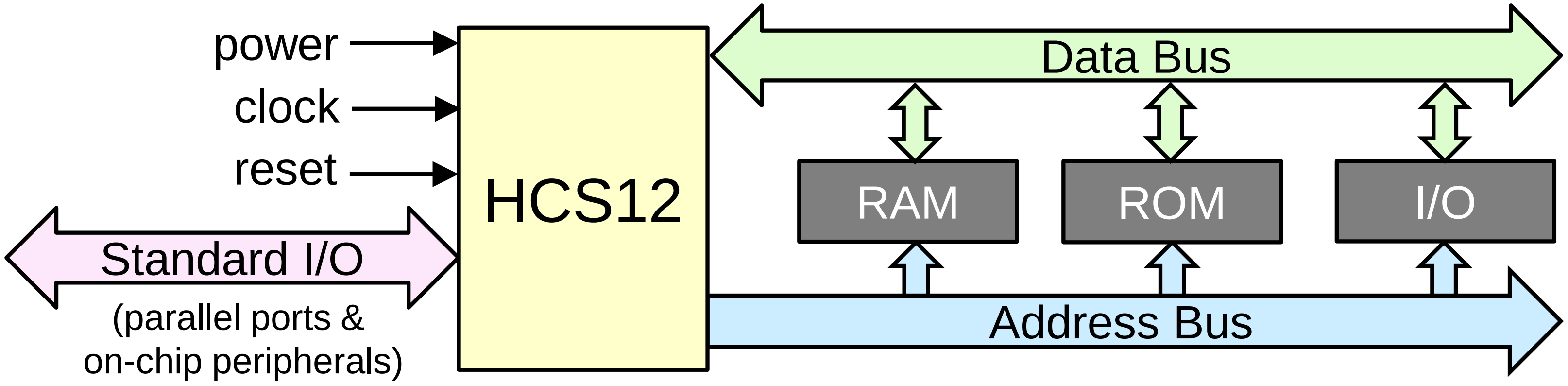
- When the peripheral function is not being used, every port has at least two registers associated with its operation as a parallel port:
  - a data register PTX (where X is the name of the port)
  - a data direction register DDRX
- To configure a specific pin for output (input), write a ‘1’ (‘0’) to associated bit in DDR register
  - e.g.

```
movb   #$81, DDRP    ; configure bits 0 & 7 of port P as output
                     ; all other bits will be inputs
```

- Many ports have (up to 6) additional registers that further define their operation

Operation Modes

- HCS12 can be operated in either:
- **Single Chip Computer** Mode
  - all memory is on-chip and all I/O is via supplied on-chip ports
  - address and data bus are not brought out to data pins
  - all pins available for standard (incl. general purpose) I/O
- **Expanded** Mode
  - address and data bus brought out to I/O pins
  - can add external RAM, ROM and custom memory mapped I/O
  - fewer pins available for standard I/O

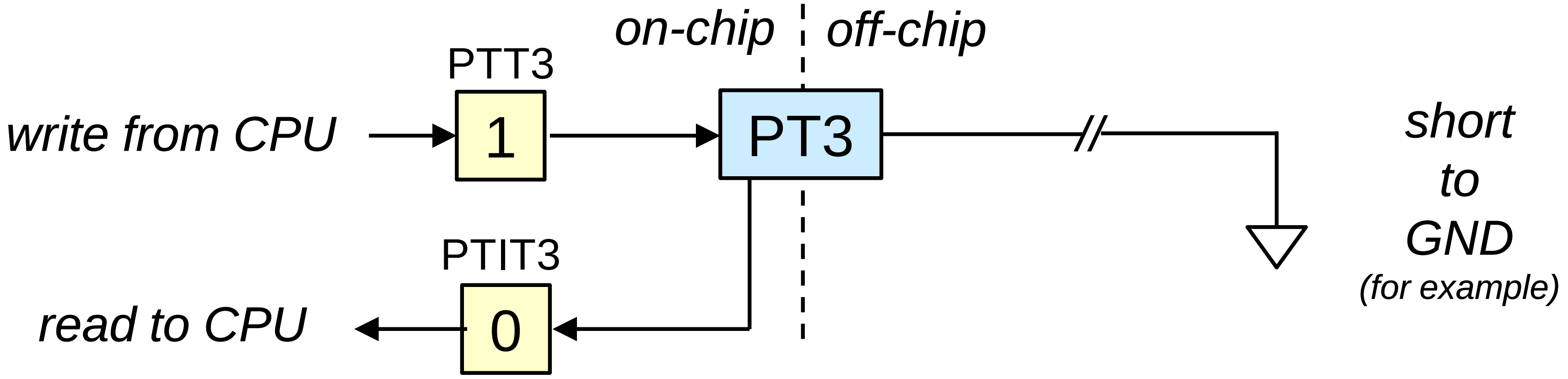


Ports A, B and K

- In single chip mode, ports A, B and K are available for general purpose I/O
  - Each port has a data register
  - PTA and PTB are 8 bits, PTK is 6-bit (PK5 ~ PK0)
  - each pin can be individually set for input or output
  - Each port has a data direction register (DDRA, DDRB and DDRK)
- No interrupt capability or status flags
  - Ideally suited for brute-force I/O
  - More sophisticated synchronization requires use of extra I/O pins (other ports, IRQ, etc.)
- In expanded mode, ports A, B and K are not available
  - port A pins carry time multiplexed upper address and data bus
  - port B pins carry time-multiplexed lower address and data bus
  - port K pins carry extended address bits
  - EVB (lab session) boards operate in expanded mode

Port T

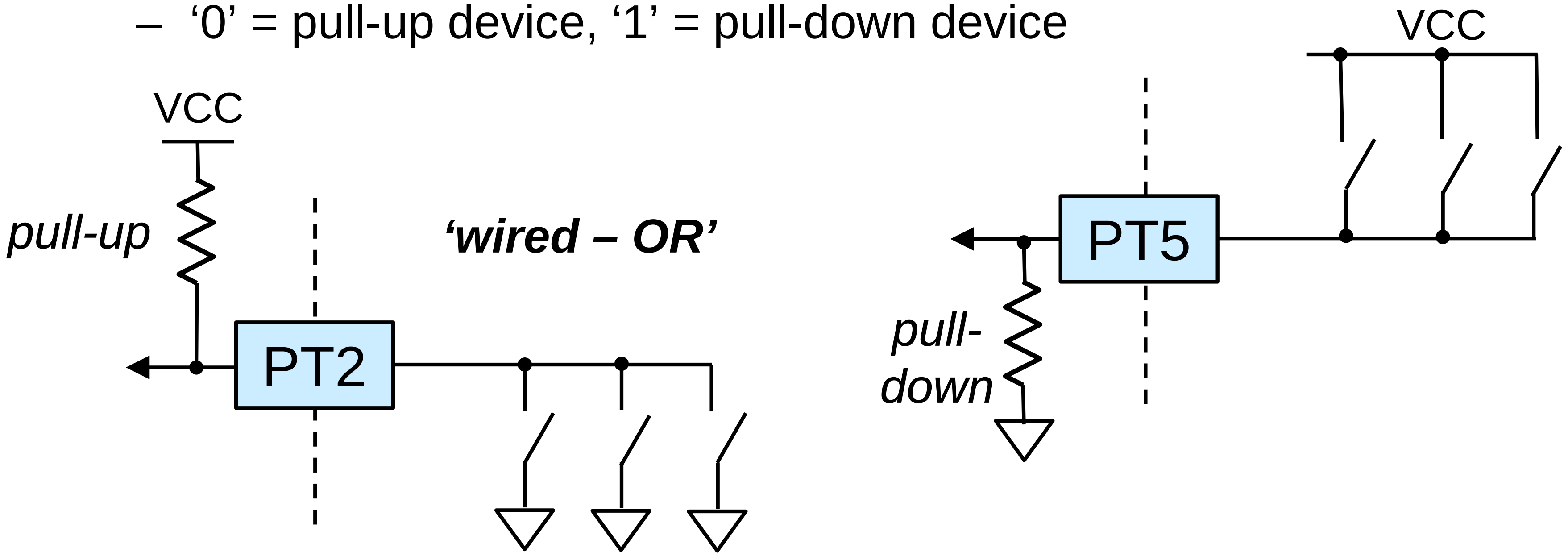
- In addition to PTT and DDRT, port T has 4 extra registers:
- **Port T Input Register (PTIT)** allows the program to read back the actual values on Port T pins when those pins are configured as output
  - can be used to check for overload, short etc. on output pins



- **Port T Reduced Drive Register (RDRT)** can be used to configure each output as full or reduced drive strength
  - '0' = full drive strength, '1' = approx. 1/3 drive strength (current)
  - can be used to slow I/O edges - leads to reduced EMI emissions

Port T (cont.)

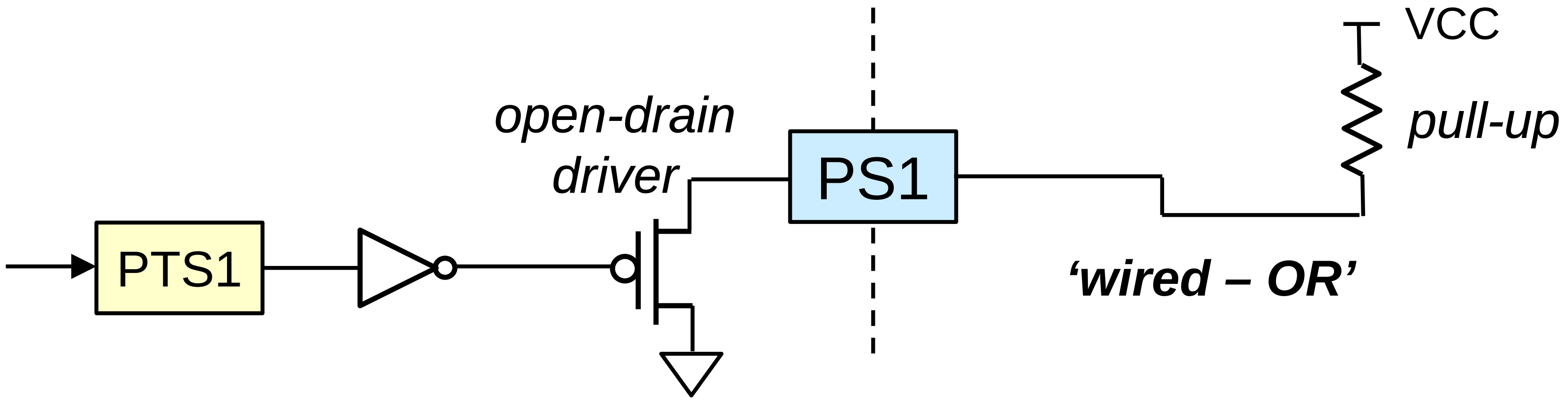
- **Port T Pull Device Enable Register (PERT)** configures each input pin to include a pull-up or pull-down device
  - '0' = pull-up /pull-down disabled, '1' = pull-up/pull-down enabled
- **Port T Polarity Select Register (PPST)** selects for each input pin whether a pull-down or pull-up is connected
  - only affects those inputs for which PERT bit is set to '1'
  - '0' = pull-up device, '1' = pull-down device



- Port T pins are also used as timer input capture/output compare pins

Port S

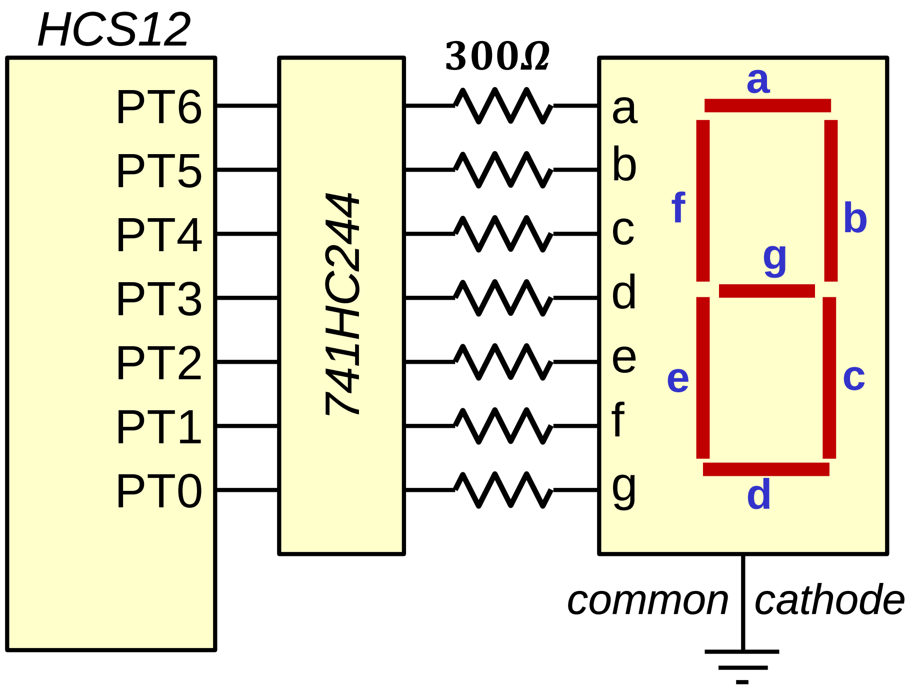
- Port S has the same 6 registers as Port T (PTS, DDRS, PTIS, RDRS, PERS and PPSS) plus:
- **the Port S Wired-OR Mode Register (WOMS)** configures each output pin as a pull-down driver suitable for wired-OR connection
  - ‘0’ = regular push-pull output, ‘1’ = open drain pull-down output



- Ports S and T are available on lab. EVB board
- But, Port S pins are used to support serial interface

Example: Driving a 7-Segment Display

- A common cathode 7-segment display can be driven from a parallel port using 74HC244 octal buffer and 300Ω current limiting resistors

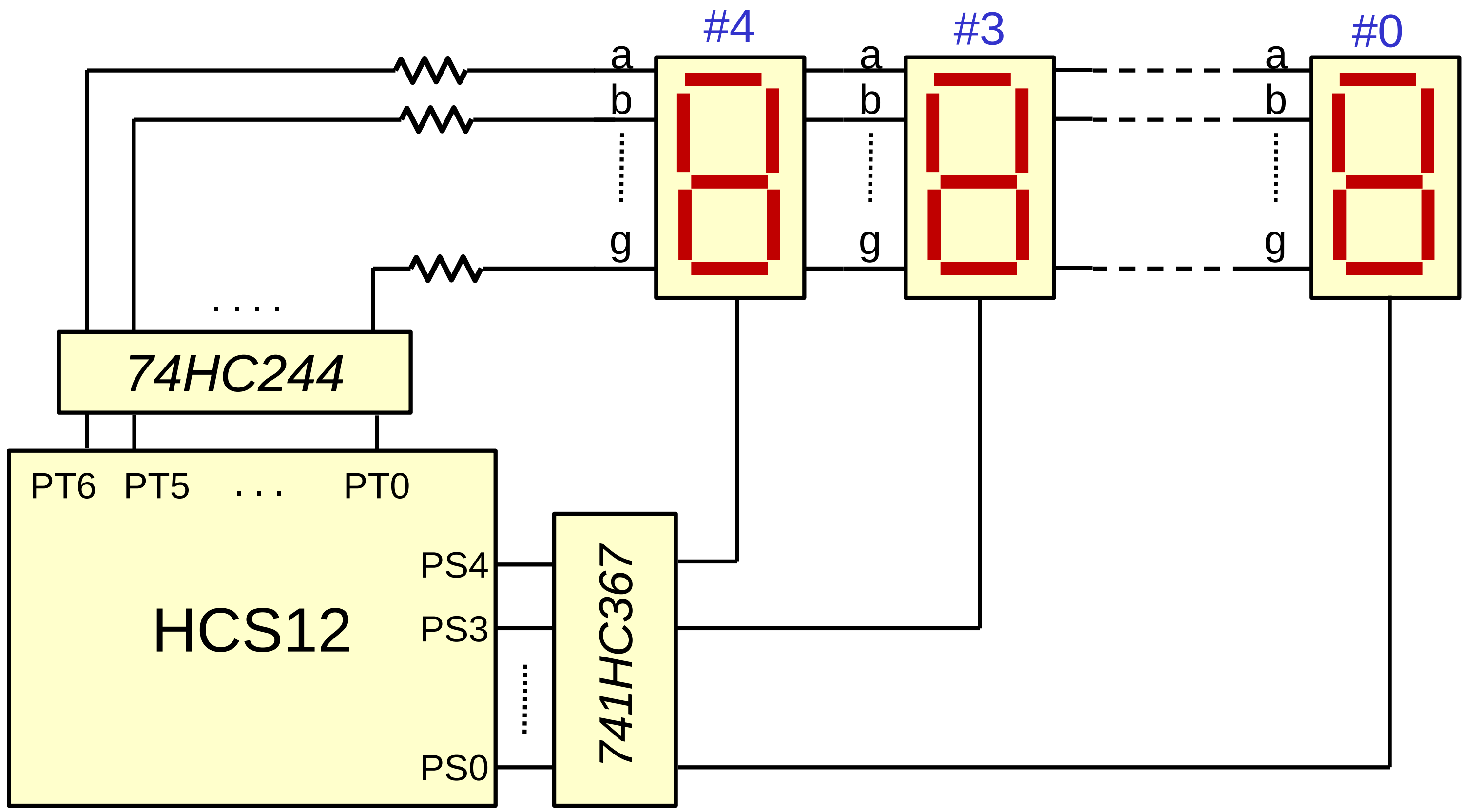


BCD Digit	Segments							Hex Code
	a	b	c	d	e	f	g	
0	1	1	1	1	1	1	0	$7E
1	0	1	1	0	0	0	0	$30
2	1	1	0	1	1	0	1	$6D
3	1	1	1	1	0	0	1	$79
4	0	1	1	0	0	1	1	$33
5	1	0	1	1	0	1	1	$5B
6	1	0	1	1	1	1	1	$5F
7	1	1	1	0	0	0	0	$70
8	1	1	1	1	1	1	1	$7F
9	1	1	1	1	0	1	1	$7B

```
four:     EQU    $33          ;seven-segment pattern for 4
          …
          movb   #$FF, DDRT   ;configure port T for output
          movb   #four, PTT   ;output 7-segment pattern
```

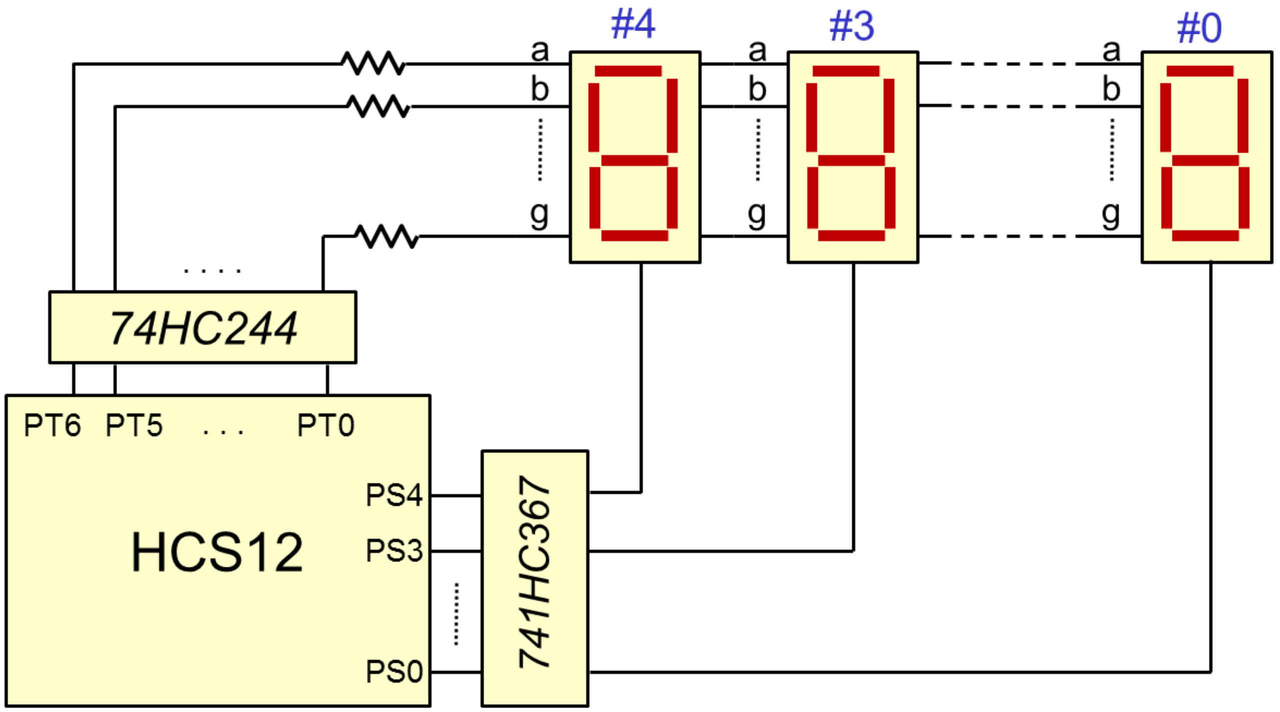
Driving Multiple 7-Segment Displays

- Connecting multiple 7-segments displays to separate output ports would require excessive number of I/O pins
- Solution is to share segment pattern
  - Turn on one display at a time - cycle quickly through different displays
  - Persistence of vision makes it appear they are all on simultaneously



Code Example: Driving 5 displays

- Write a program to display “14725” on the five seven-segment displays:



```
          include   hcs12.inc
          ORG       $800
data:     DC.B      1,4,7,2,5                                  ;digits to be displayed
patterns: DC.B      $7E,$30,$6D,$79,$33,$5B,$5F,$70,$7F,$7B   ;7-seg. patterns
dispen:   DC.B      $EF, $F7,$FB,$FD,$FE                       ;display enable codes
disptr:   DS.W      1                                          ;pointer to display codes
```

Driving 5 displays (2)

```
            ORG     $4000
            movb    #$FF, DDRT      ;set up Port T for output
            movb    #$1F, DDRS      ;set up lower 5 bits of Port S for output
forever:    ldy     #data           ;pointer to digits
            ldx     #dispen         ;set up display code pointer
            stx     disptr
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
next:       ldaa    1,y+            ;get digit & increment pointer
            ldx     #patterns       ;pointer to patterns
            ldab    a,x             ;get 7-segment code
            stab    PTT             ;output the pattern to displays
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            ldx     disptr
            ldaa    1,x+            ;get display code
            stx     disptr          ;increment display pointer
            staa    PTS             ;output display code to Port S
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            jsr     delayby1ms      ;hold pattern for 1ms
            cpy     #data+5         ;are we done yet?
            lbeq    forever         ;yes -start again on first digit
            bra     next            ;no – go to next digit
```

Ports H, J and P

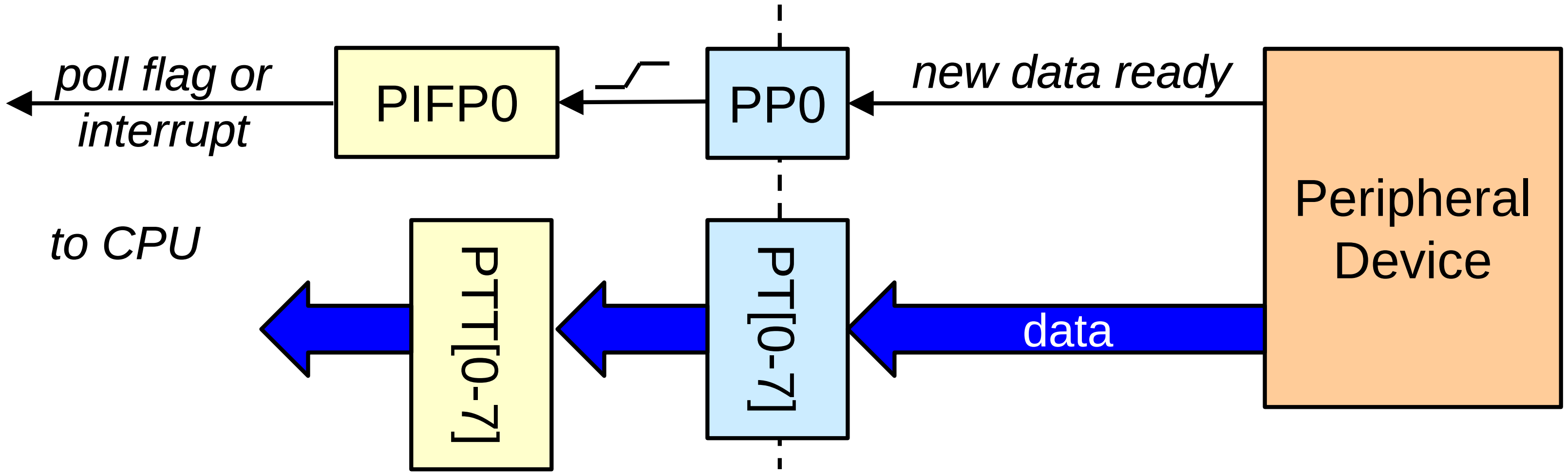
- Ports H, J and P have the same 6 registers as Port T (e.g. PTP*, DDRP, PTIP, RDRP, PERP and PPSP) plus 2 extra registers:

- **the Port P Interrupt Flag Register (PIFP)** indicates, on a per-pin basis when a rising (falling) edge has occurred on corresponding input pin in data register (PTP)
  - ‘0’ = edge has not occurred, ‘1’ = edge has occurred
  - indicates rising edge if corresponding bit in Polarity Select Register (PPSP) is ‘1’, otherwise indicates falling edge
  - can be used as a “new data ready” flag when polling
  - CPU must write a ‘1’ to clear the flag
  - causes an interrupt if corresponding bit is set in Interrupt Enable register (PIEP)

* Only Port P register names are given. For Ports H and J, just replace P in register names with H or J.

Ports H, J and P (cont.)

- **the Port P Interrupt Enable Register (PIEP)** enables interrupt to occur when corresponding bit in Interrupt Flag Register (PIFP) is set
  - ‘0’ = interrupt is disabled, ‘1’ = interrupt is enabled
  - each Port P pin can be used as an independent edge-sensitive interrupt source
  - all eight pins of the port share the same interrupt vector
  - when interrupt occurs, interrupt service routine checks PIFP register to see which bit(s) is set
  - Can be used to add interrupt to existing data port, e.g:



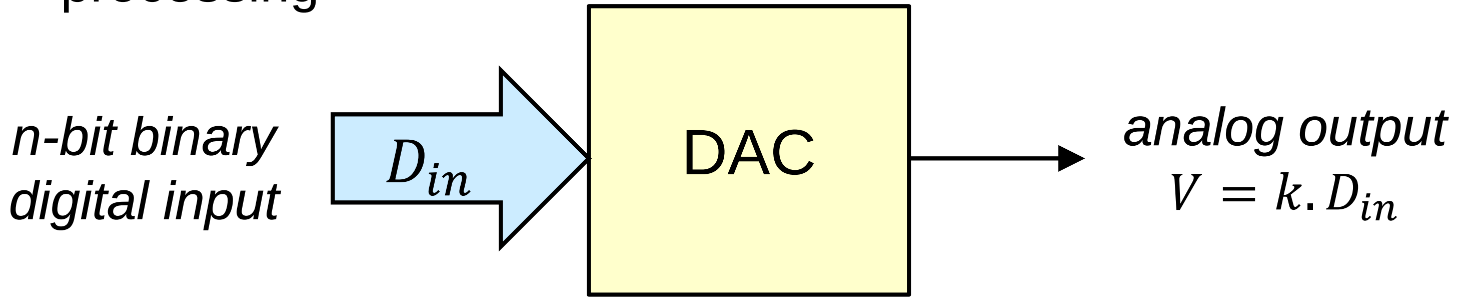
Ports AD0 and AD1

- HCS12 can have up to two on-chip, 8 channel, 10 bit A/D converters
  - 16 analog input pins AN7~AN0 (associated with AD0) and AN15~AN8 (associated with AD1)
- When A/D’s are not being used, these input pins can be redefined as digital <u>inputs</u> PAD15~PAD0 associated with parallel ports PTAD1 and PTAD0.

- **ATD Input Enable Register (ATD0DIEN and ATD1DIEN)** determines on a per-pin basis, whether a bit should be used as an analog or digital input
  - ‘0’ disables digital input on PTAD pin, ‘1’ enables digital input
  - AD0 and AD1 ports can only be used as input (no digital output)

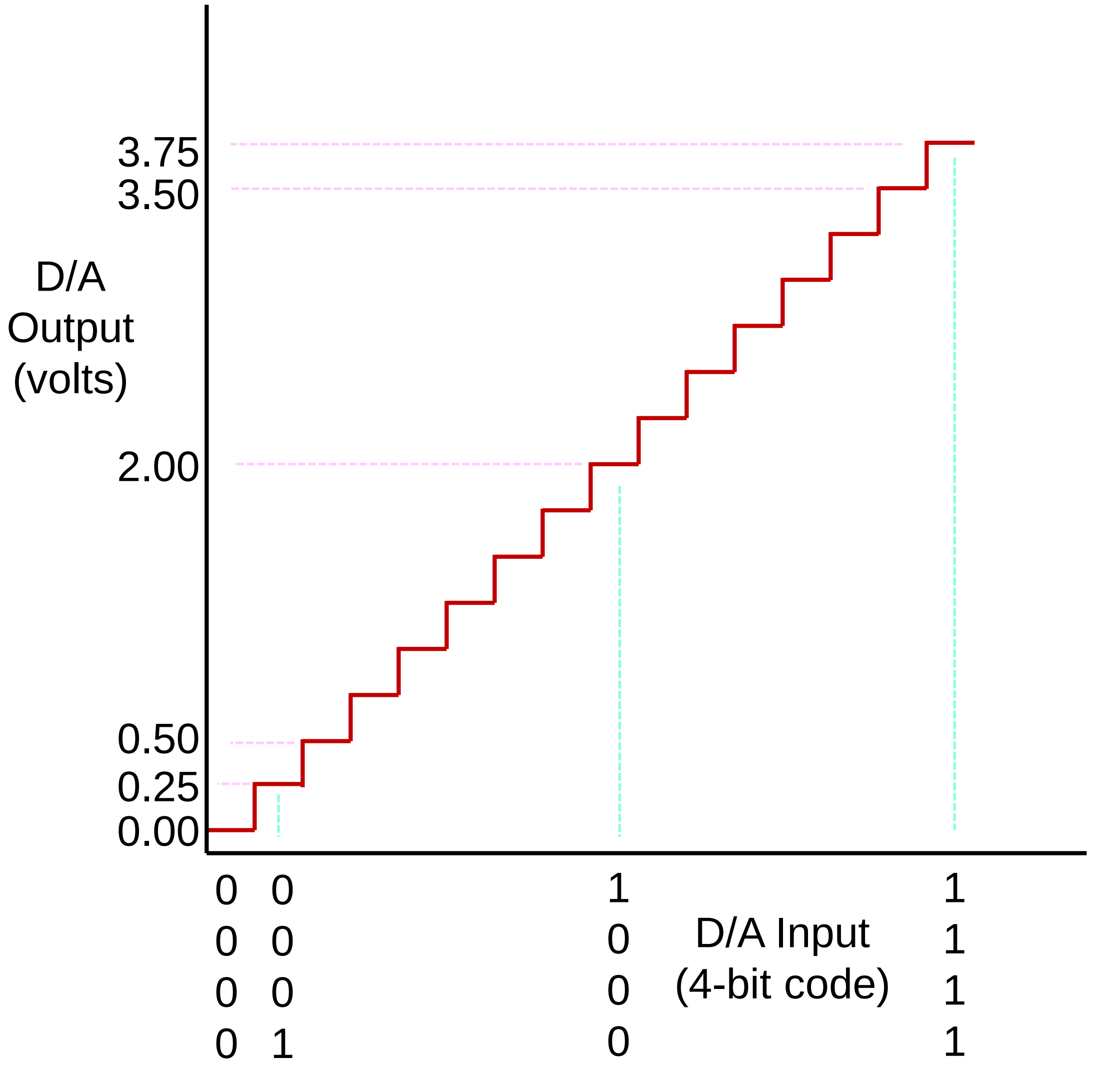
Interfacing to D/A Converter

- A Digital to Analog Converter (DAC or D/A) converts binary fixed-point numbers to an analog voltage or current
  - Analog output is a linear function of digital input
  - Input is often sampled digital values resulting from digital signal processing



- Specifications of a D/A converter include:
  - resolution (how many input bits)
  - number of channels (of output)
  - type of output (voltage, current, etc.)
  - speed (in response to changing digital input)
  - linearity (how much error in conversion process)
  - monotonicity (can an increase in input generate decrease in output?)
  - noise (superimposed on analog output)

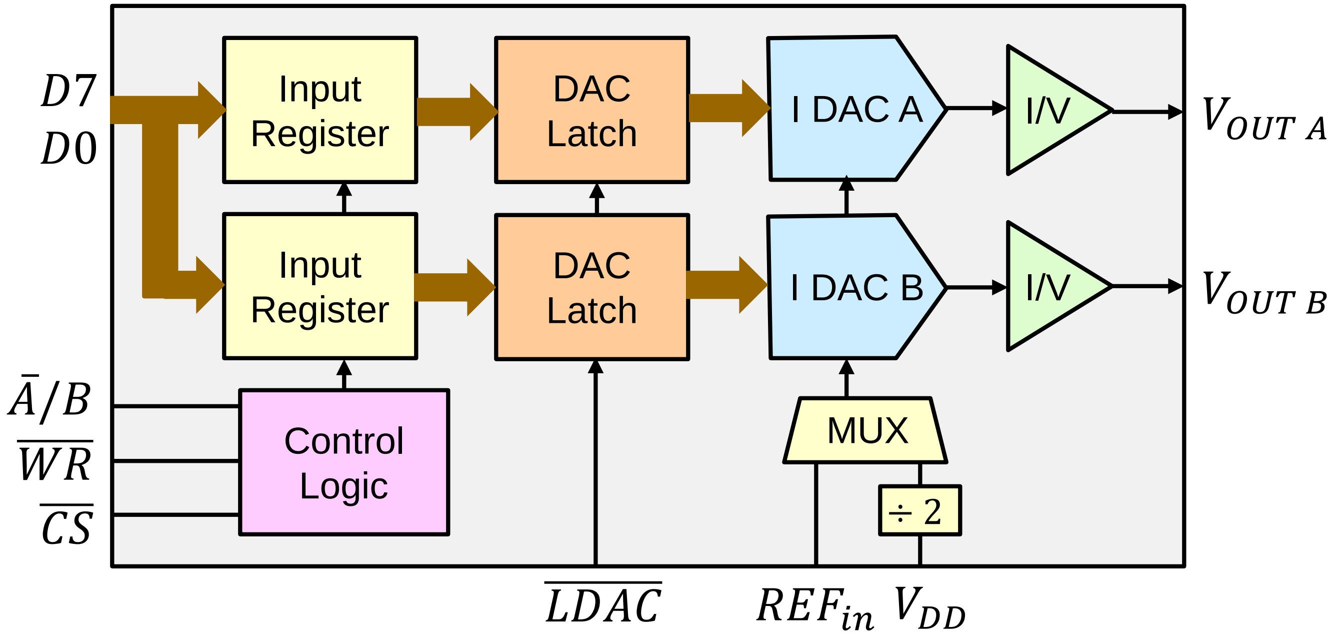
Example: 4-bit D/A Converter



- 16 possible 4-bit input codes
- translate into 16 different output voltages in the range of 0-4V.

AD7302 D/A Converter

- Dual Channel 8-bit D/A converter made by Analog Devices
- Designed to be a memory mapped device:



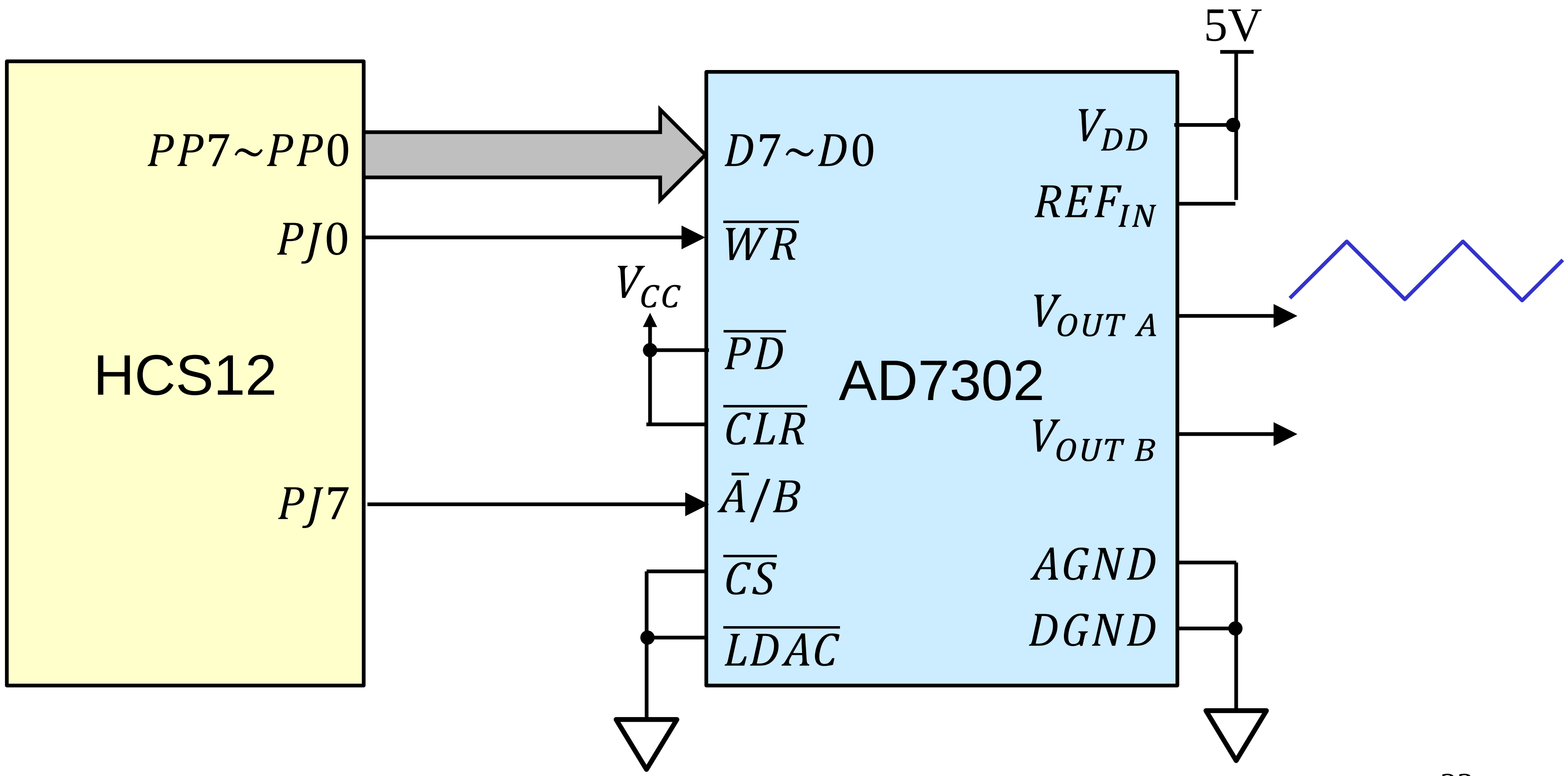
$$V_{OUT\ A/B} = V_{REF} \times (D/256)^*$$

- Data is latched into input register on rising edge of $\overline{WR}$
- Each conversion takes about $2\mu s$ to complete

**Note: If $REF_{in} \approx V_{DD}$, otherwise see text book*

Generate Triangular Waveform using D/A

- Configure Port P for digital output
- Use rising edge on PJ0 to latch data into D/A
- Select output A for triangular output

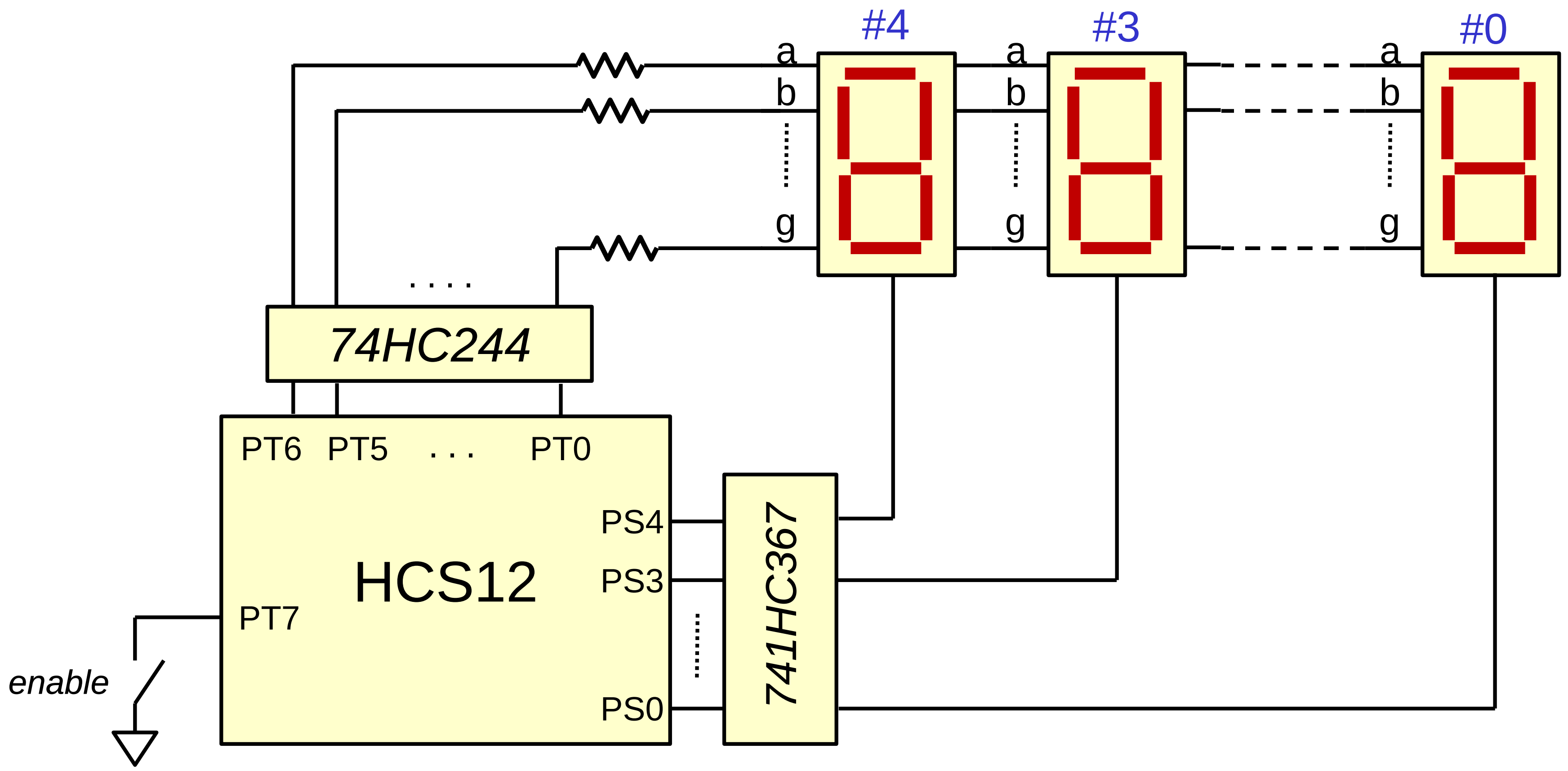


Code for Triangular Waveform

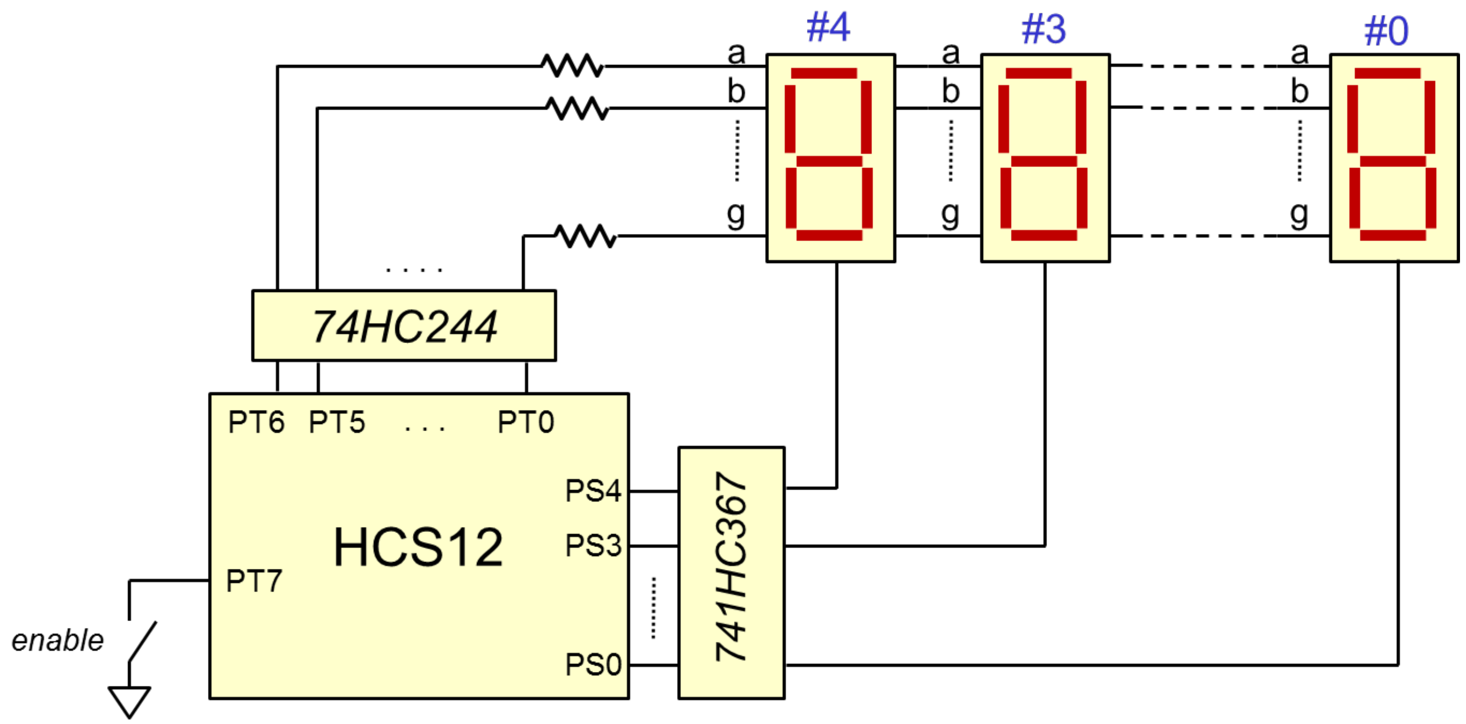
```
        ORG     $4000
        movb    #$FF, DDRP          ;configure Port P for output
        movb    #$81, DDRJ          ;configure PJ0 and PJ7 for output
        bclr    PTJ, $80            ;select VOUTA output
        clra                        ;initialize count
cntup:  staa    PTP                 ;send count to output
        bclr    PTJ, $01            ;generate rising edge on PTJ0
        bset    PTJ, $01            ;to load value into DAC
        jsr     waitfor1ms          ;hold value for 1ms
        inca                        ;increment count
        cmpa    #$FF                ;test for maximum
        bne     cntup
cntdn:  staa    PTP                 ;send count to output
        bclr    PTJ, $01            ;generate rising edge on PTJ0
        bset    PTJ, $01            ;to load value into DAC
        jsr     waitfor1ms          ;hold value for 1ms
        deca                        ;decrement count
        bne     cntdn               ;test for minimum
        bra     cntup
```

Driving 5 displays with “display enable” button

- Write a program to display “14725” on the five seven-segment displays
- Only turn on displays when “enable” button is pushed



Driving 5 displays with “display enable” button



```
          include   hcs12.inc
          ORG       $800
data:     DC.B      1,4,7,2,5                                    ;digits to be displayed
patterns: DC.B      $7E,$30,$6D,$79,$33,$5B,$5F,$70,$7F,$7B      ;7-seg. patterns
dispen:   DC.B      $EF, $F7,$FB,$FD,$FE                         ;display enable codes
disptr:   DS.W      1                                            ;pointer to display codes
```

Driving 5 displays (2)

```
          ORG     $4000
          movb    #$7F, DDRT       ;set PortT(0-6) for output, PorT(7) for input
          movb    #$80, PERT       ;set PT7 to have pull up/down device
          movb    #$00, PPST       ;set PT7 pull direction as pull-up
          movb    #$1F, DDRS       ;set up lower 5 bits of Port S for output
forever:  ldy     #data            ;pointer to digits
          ldx     #dispen          ;set up display code pointer
          stx     disptr
next:     ldaa    1,y+             ;get digit & increment pointer
          ldx     #patterns        ;pointer to patterns
          ldab    a,x              ;get 7-segment code
          stab    PTT              ;output the pattern to displays
          ldx     disptr
          ldaa    1,x+             ;get display code
          stx     disptr           ;increment display pointer
          brclr   PTT, $80, skip   ;branch if button pushed
          ldaa    #$1F             ;set display code to “all-off”
skip:     staa    PTS              ;output display code to Port S
```

Driving 5 displays (2)

```
jsr     delayby1ms      ;hold pattern for 1ms
cpy     #data+5         ;are we done yet?
lbeq    forever         ;yes -start again on first digit
bra     next            ;no – go to next digit
```