CPE 390: Microprocessor Systems

Spring 2018

Lecture 14

Analog to Digital Conversion

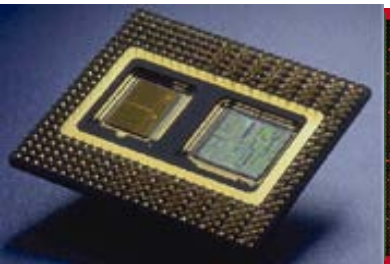
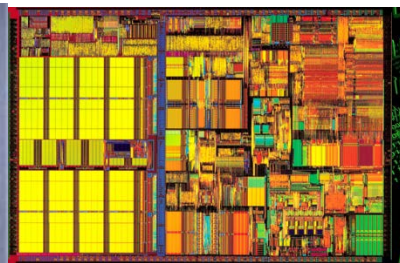
Bryan Ackland

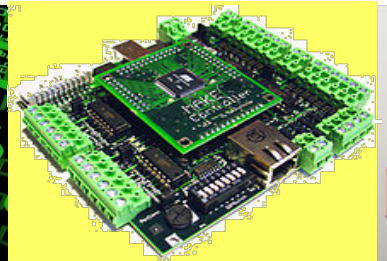
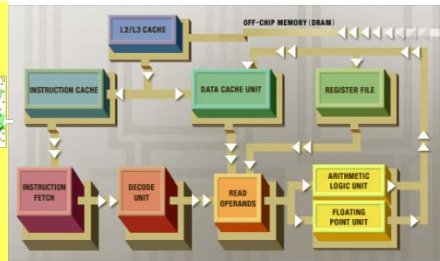
Department of Electrical and Computer Engineering

Stevens Institute of Technology

Hoboken, NJ 07030

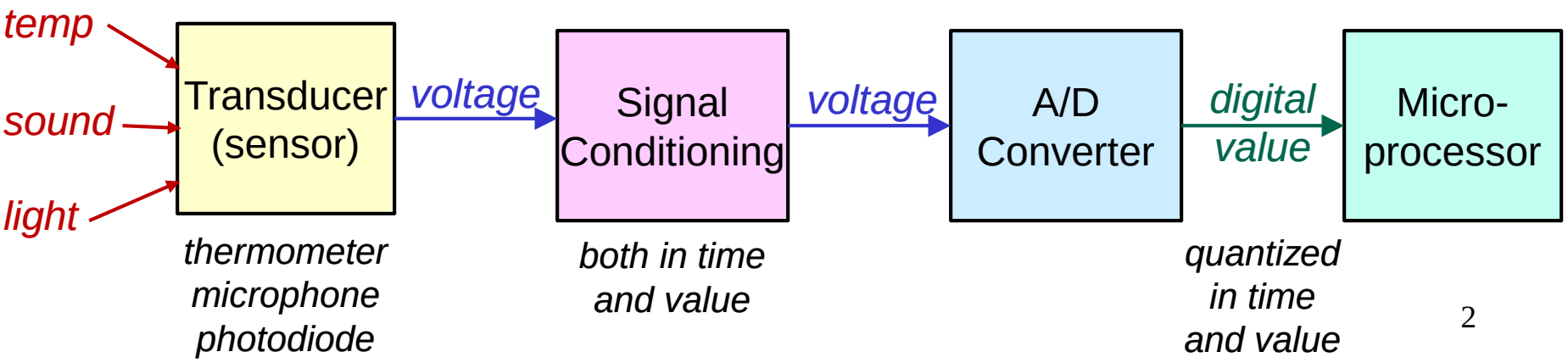
Adapted from HCS12/9S12 An Introduction to Software and Hardware Interfacing Han-Way Huang, 2010

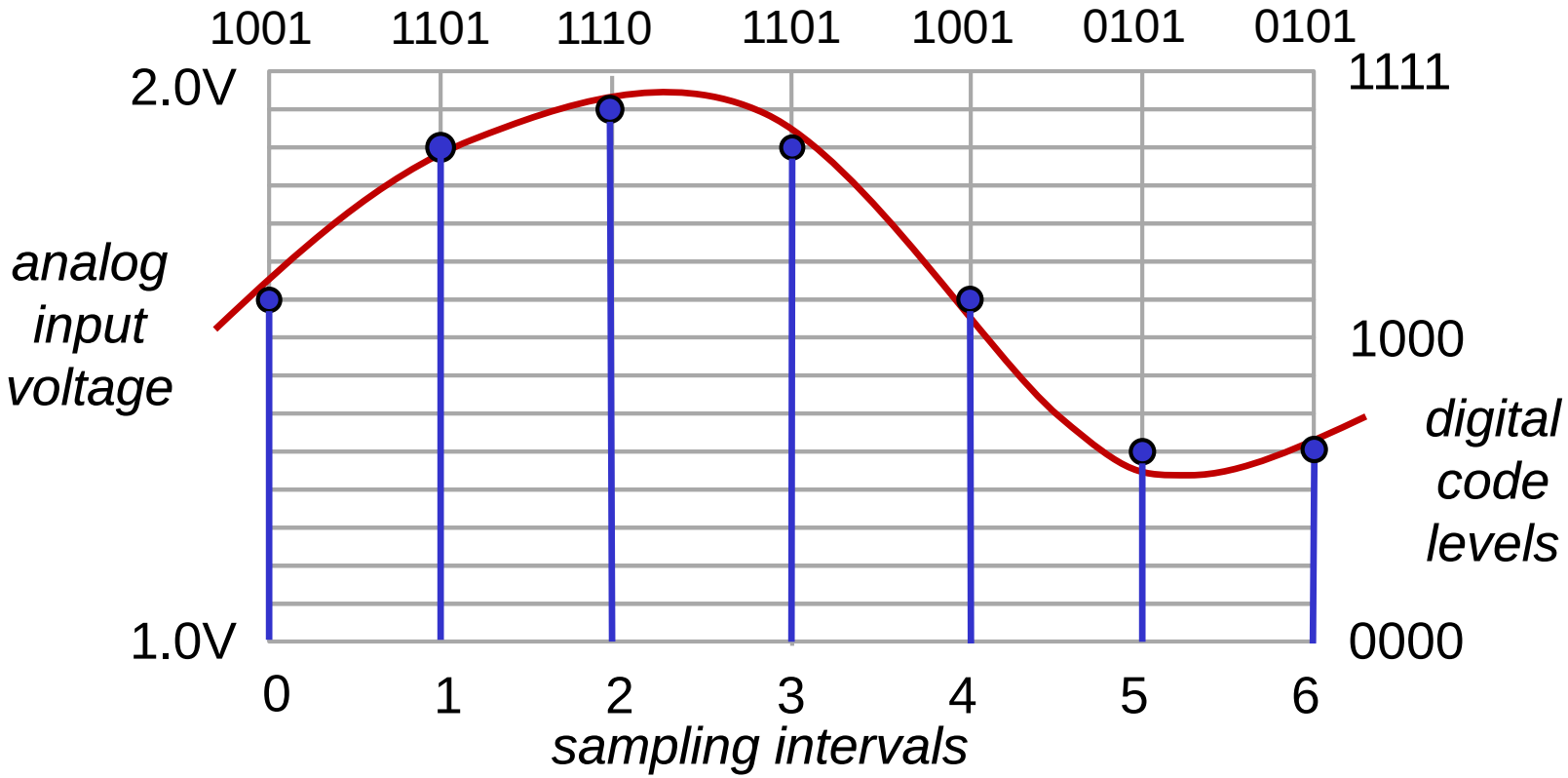
The Real World of Analog

- A microprocessor deals exclusively with digital data
  - finite precision representations of external real world and internal computational data
- A microcontroller in an embedded application takes inputs from real-world sensors
  - some of these are already digital (e.g. switches, keyboard, mouse)
  - many are analog (e.g. pressure, temperature, light intensity, microphone, airflow, engine speed, oxygen level)
- Analog-to-Digital converter (A/D) transforms analog signal into digital representation usable by microprocessor



Analog to Digital Conversion

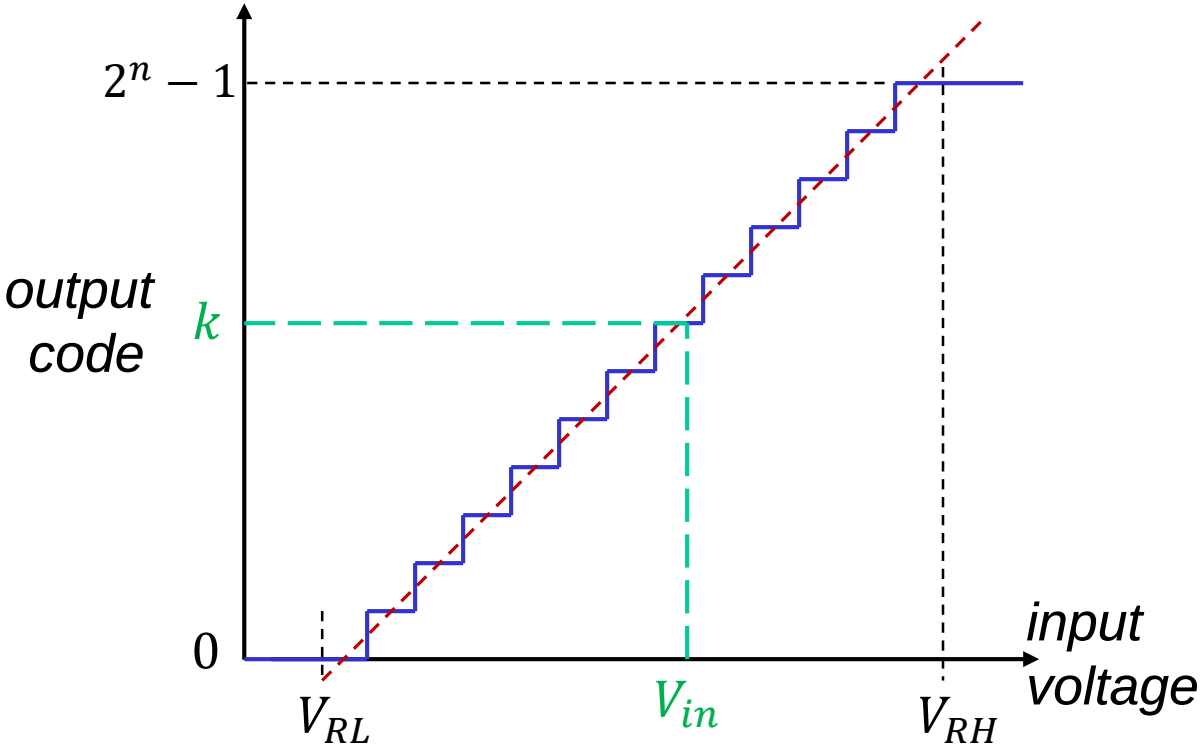
- An A/D converter samples an analog signal at regular intervals and generates a digital code which is its best (closest) approximation to the analog value at that instant

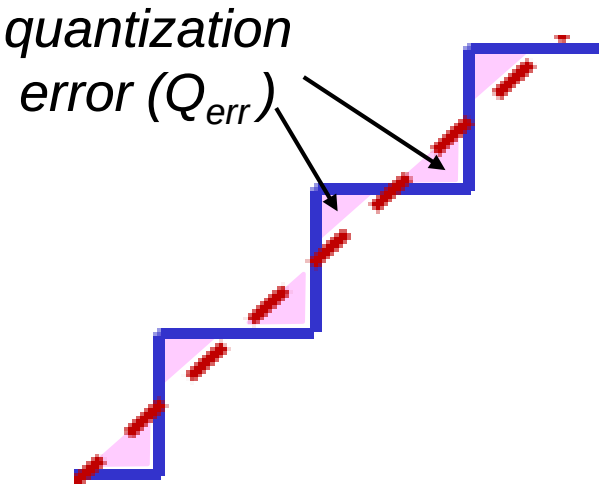


- Analog signal: continuous in time and value
- Digital signal: quantized in time and value

A/D Transfer Function

- An n-bit A/D converter has 2^n possible output codes
- Input voltage range typically defined by two reference voltages V_{RL} and V_{RH}





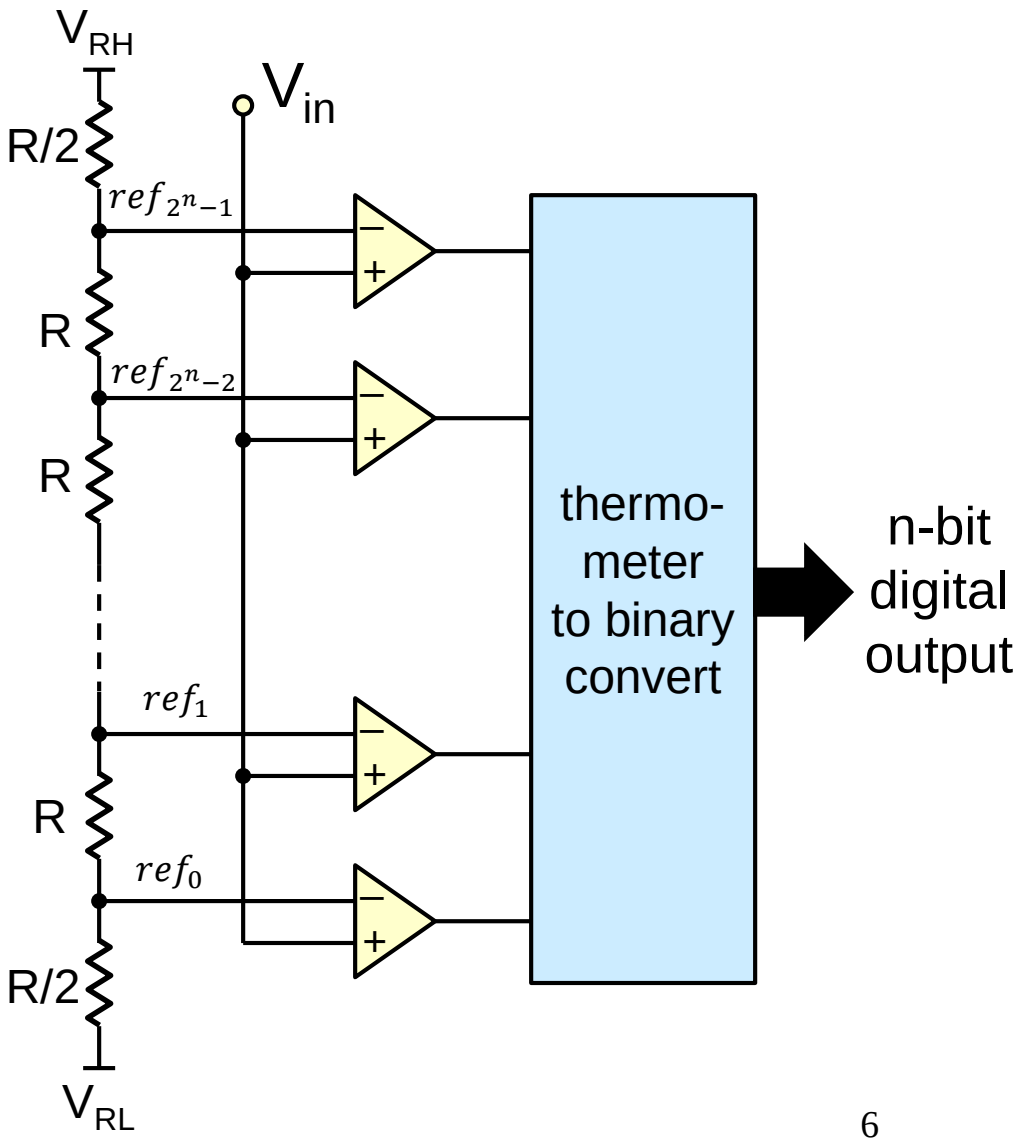
$$V_{in} = V_{RL} + \frac{(V_{RH} - V_{RL}).k}{2^n - 1} \pm Q_{err}$$

A/D Characteristics

- **Resolution**
  - often quoted in terms of # bits (e.g. 12-bit converter)
  - analog resolution is $(V_{RH} - V_{RL})/2^n$
  - average conversion error = $(V_{RH} - V_{RL})/2^{n+2}$
- **Conversion time**
  - how long does it take to produce digital code
- **Maximum Sampling Frequency**
  - How many samples per second
  - Limits maximum frequency component of analog input that is accurately captured by the A/D (see Nyquist)
- **Linearity**
  - staircase has uneven steps
  - generates error in addition to quantization error
  - significant non-linearity can lead to non-monoticity (e.g. a higher voltage generates a smaller code)

Flash (Parallel) A/D Converter

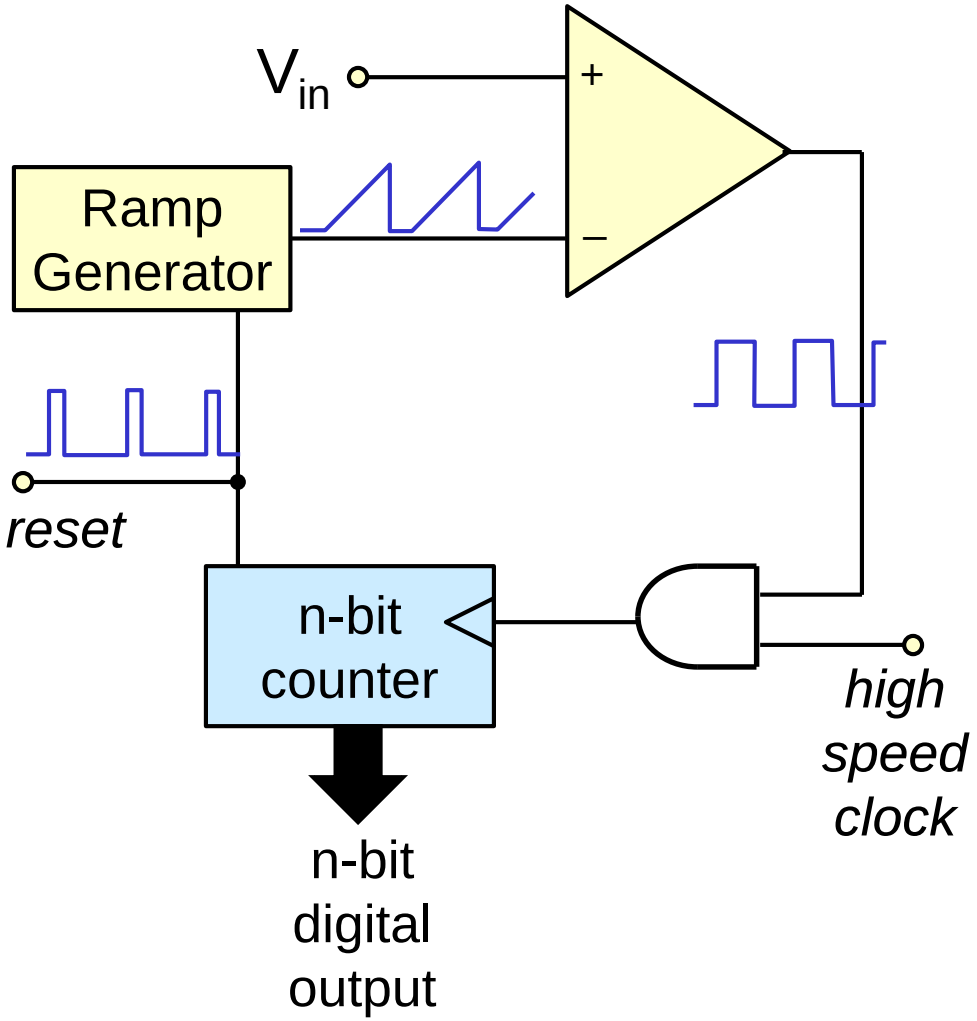
- Resistor ladder generates 2^n reference voltages
- 2^n comparators simultaneously compare input with each reference
- Comparator output k is high if Vin > ref_k
- Conversion logic generates code indicating greatest value of k for which comparator output is high
- Very high speed
- Expensive in area & power
- Limited to ~ 8-bits



Single Slope A/D Converter

- Compares input to linear ramp to generate a pulse width proportional to V_{in}
- Pulse used to gate clock to high speed digital counter
- Simple hardware – popular in low speed applications
- High resolution possible
- Performance limited by:

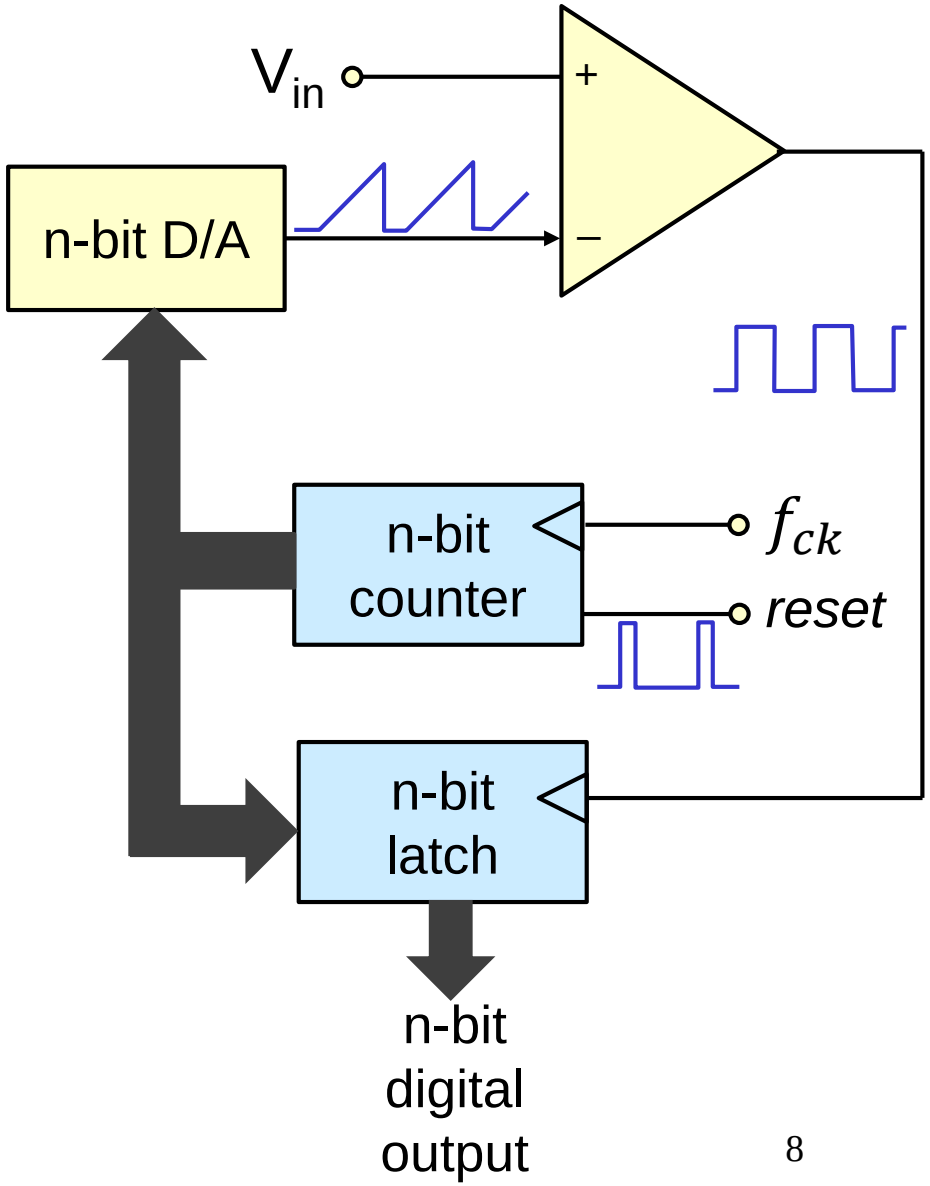
$$f_{ck} = f_{samp} \times 2^n$$



e.g. for f_{samp} = 1 MHz, a 12-bit converter requires f_{ck} = 4 GHz

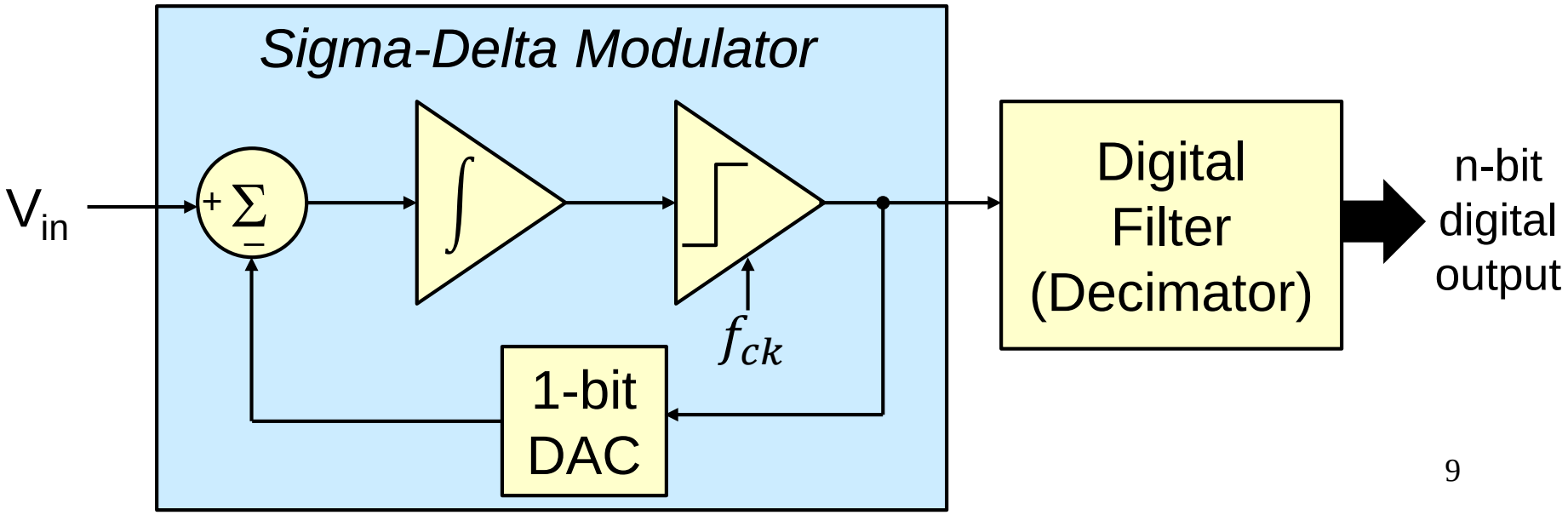
Counter Ramp A/D Converter

- Variant on single-slope converter
- Ramp is generated by counter driving a D/A converter
- When D/A output ramp crosses V_{in}, counter value is captured in n-bit latch
- Does not require precision analog ramp generation
- Precision limited by linearity of D/A



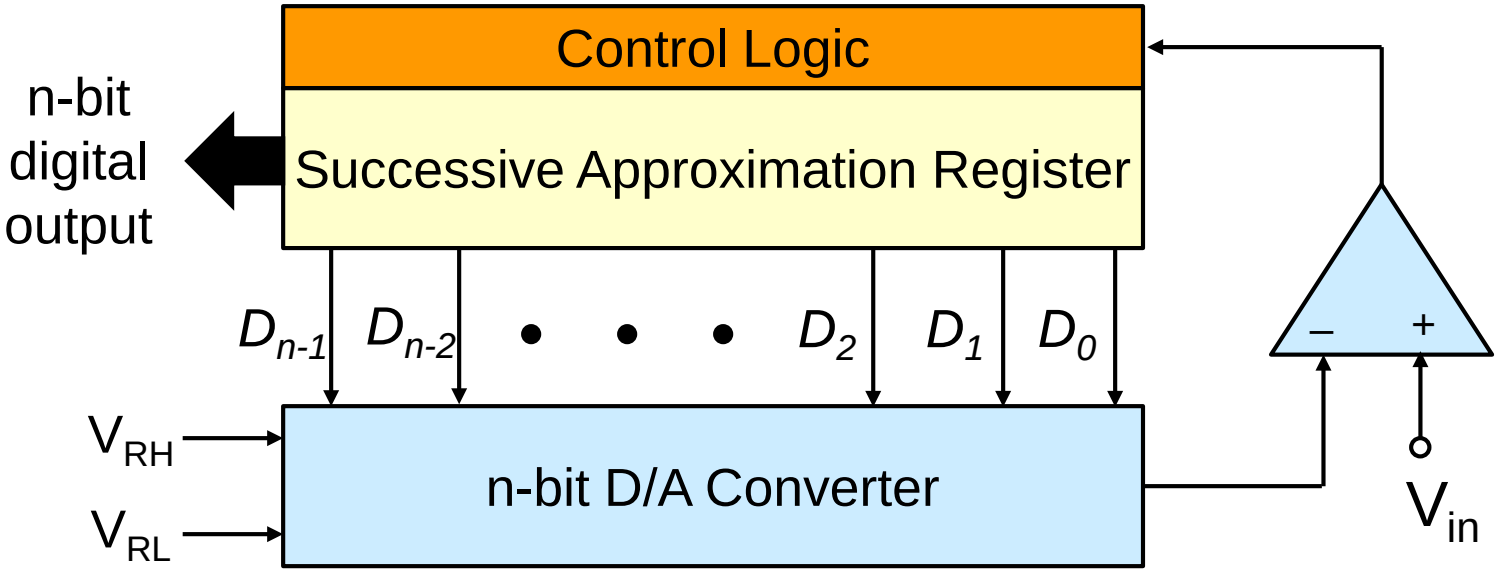
Sigma Delta (Oversampling) A/D Converter

- Sigma-delta modulator consists of summer, integrator, clocked comparator and a 1-bit DAC
- Modulator runs at many times (e.g. 16x – 1000x) the required sampling frequency to produce very high speed 1-bit waveform
- Digital filter coverts this to much slower n-bit digital output
- Since 1-bit DAC is perfectly linear, can produce very high resolution (up to 24-bit)
- Sampling frequency is limited by need to over-sample



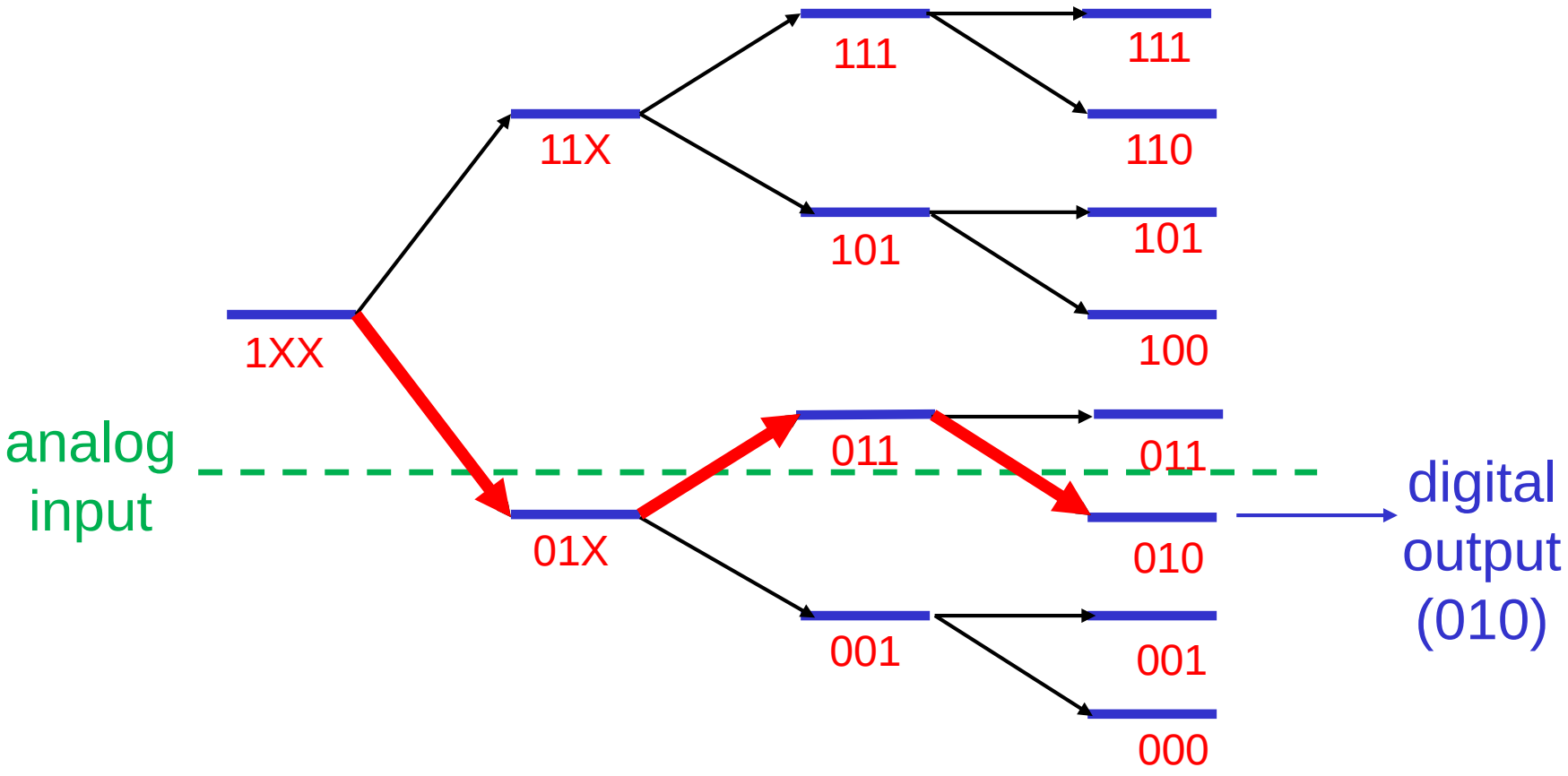
Successive Approximation A/D Converter

- Guesses and then corrects digital code in SAR one bit at a time



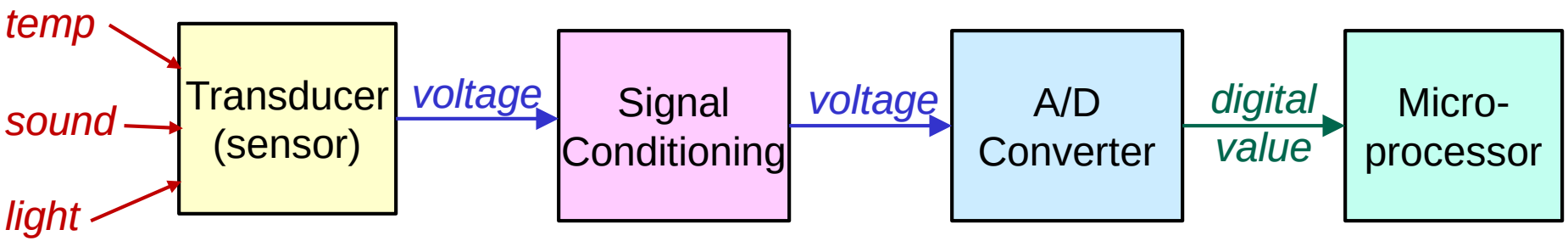
- Initially sets all bits in SAR to ‘0’
- Then starting with MSB, for each bit:
  - set bit to ‘1’ and convert output of SAR to analog value with D/A
  - compare output of D/A to input voltage
  - if D/A is larger, set this bit back to ‘0’ and go on to next (lesser sig.) bit
  - if input is larger, retain ‘1’ for this bit and go on to next bit

Successive Approximation Process



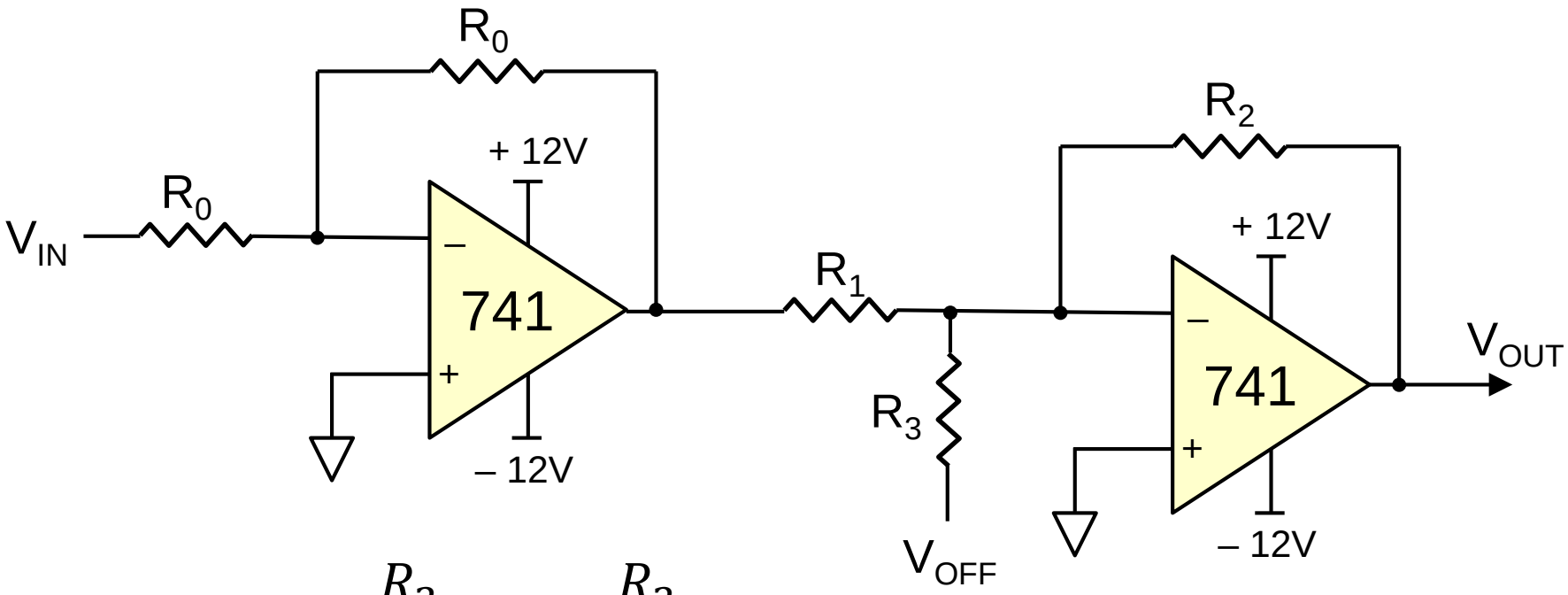
- SAR gives a good tradeoff between speed and precision
- One of most popular A/D techniques in embedded systems
- Used in HCS12

Signal Conditioning



- Signal Conditioning is process of matching transducer output to input characteristics of A/D
  - Need to match in voltage and time (frequency)
- Input range of A/D defined by V_{RH} and V_{RL}
  - Unlikely to match output range of transducer
  - e.g. transducer may output signal (-1 to +1 V), whereas A/D has input range of (0 to 4V).
  - Need “scale & shift circuit” to scale by x2 and shift by +2V.
- If sampling rate of A/D if f_s samples/sec, maximum frequency correctly captured is $f_s/2$
  - Any components of higher frequency content will be aliased
  - Signal conditioning may need low pass filter

Shift & Scale Circuit



$$V_{OUT} = \frac{R_2}{R_1}.V_{IN} - \frac{R_2}{R_3}.V_{OFF}$$

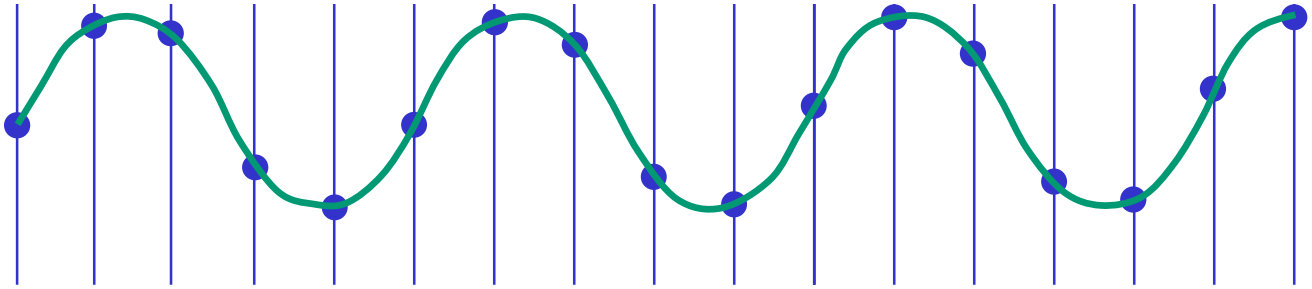
- From previous example, if $R_1 = R_3 = 10k\Omega$, $R_2 = 20k\Omega$, $V_{OFF} = -1V$:

$$V_{OUT} = (2 \times V_{IN}) + 2$$

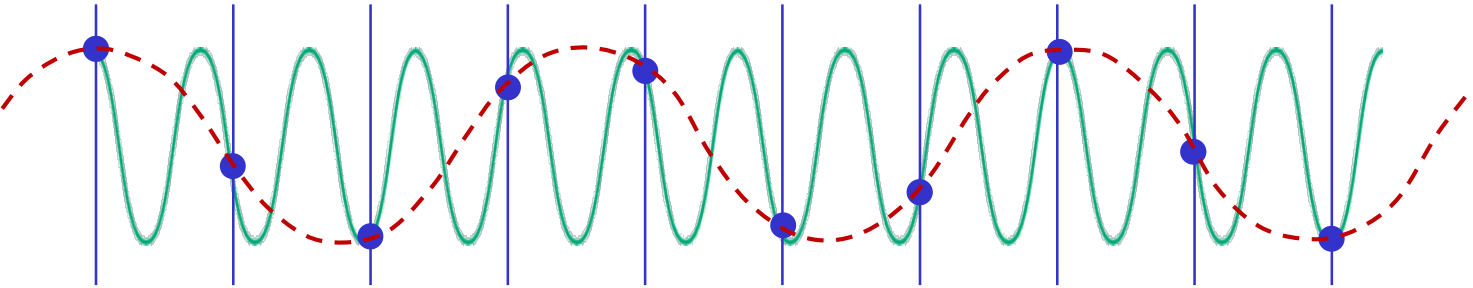
Nyquist Frequency

- If f_s is the sampling frequency, $f_s/2$ is known as Nyquist frequency
- Any signal component above Nyquist frequency will be aliased back into sampled waveform as a lower frequency component

Aliasing



$f_{sig} < f_s/2$



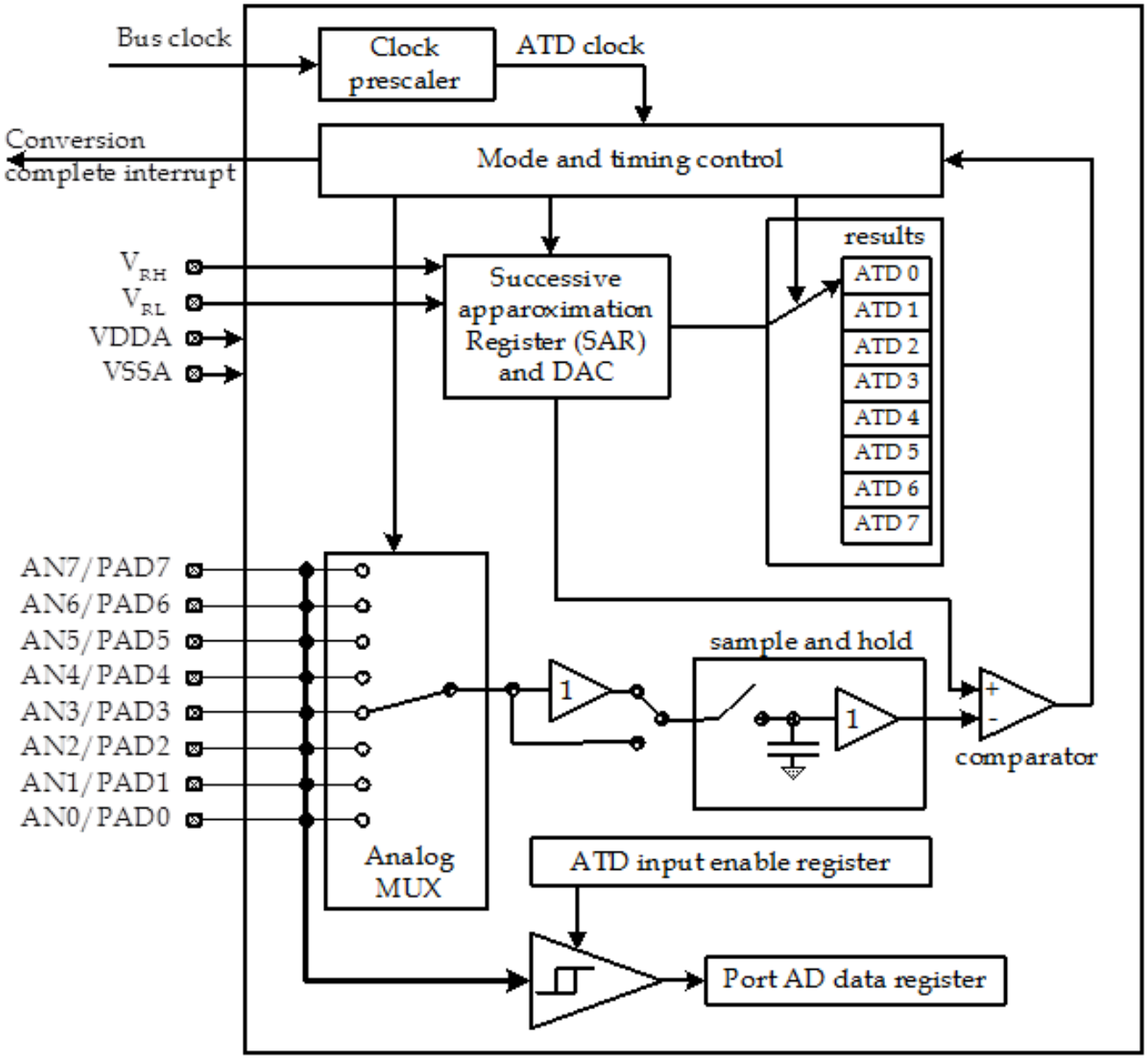
$f_{sig} > f_s/2$

- Even if desired signal does not contain components > Nyquist, there may be high frequency noise components which must be removed
- Signal conditioning circuits frequently include a sharp low-pass filter to take out any signal components > Nyquist

A/D Conversion on HCS12

- HCS12 may have one or two 8-channel 10-bit A/D's
- Each uses successive approximation method
- A/D's runs off an ATD clock that can be set 500kHz ~ 2 MHz
  - At 2 MHz, ADC can perform an 8-bit conversion in 6μs or a 10-bit conversion in 7 μs.
- A/D conversion may be internally triggered (by writing to a control register) or externally triggered (via pins AN7 or AN15)
- May be a single conversion or a sequence of conversions
- Result(s) can be 8-bit or 10-bit, signed or unsigned:
  - (–128 to +127) or (–512 to +511) signed
  - (0 to 255) or (0 to 1023) unsigned
- Result(s) stored in 16-bit register(s)
  - either left or right justified

ATD Block Diagram



A/D Pins & Registers

- **Signal Pins:**
  - AD0 module uses pins AN0 ~ AN7
  - AD1 module uses pins AN8 ~ AN15
  - AN7 (AN15) pin can optionally be used to trigger AD0 (AD1) module
  - V_{RH} and V_{RL} are high and low reference voltage inputs
  - V_{DDA} and V_{SSA} are power supply and ground pins

- **Each A/D has following registers:**
  - Six control registers ATDxCTL0 ~ ATDxCTL5
    - (0 and 1 for factory testing only)
  - Two status registers ATDxSTAT0 ~ ATDxSTAT1
  - One input enable register ADTxDIEN
  - One port data register PTADx
  - Eight 16-bit result registers ATDxDR0 ~ ATDxDR7

where x= 0 or 1 (we will only describe AD0 registers in following slides)

ATD Control Register 2 (ATD0CTL2)

7	6	5	4	3	2	1	0
ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
Reset: 0	0	0	0	0	0	0	0

ADPU: ATD power-down bit ('0': power-down, '1': normal ATD operation)

AFFC: ATD fast flag clear bit
'0': ATD flag is cleared normally (i.e. read status before reading result)
'1': Any access to ATD result register will cause associated CCF flag to clear

AWAI: Power-down in wait mode ('0': ATD runs in wait, '1': does not run in wait)*

ETRIGLE: External trigger level/edge control (see next slide)

ETRIGP: External trigger polarity (see next slide)

ETRIGE: External trigger mode enable ('0': disable trigger on ATD channel 7, '1': enable trigger on ATD channel 7)

ASCIE: ATD sequence complete interrupt enable bit ('0': disabled, '1': enabled)

ASCIF: ATD sequence complete interrupt flag

* *We will not be using these bits*

Notes on ATD0CTL2

- ASCIF flag signals that a requested multi-sample sequence conversion sequence has completed (see ATD0CTL3)
  - If ASCIE is set, this will also cause interrupt
- Writing to ATD0CTL2 will abort any current conversion sequence

ETRIGLE	ETRIGP	External Trigger Sensitivity
0	0	Falling edge
0	1	Rising Edge
1	0	Low level
1	1	High level

ATD Control Register 3 (ATD0CTL3)

7	6	5	4	3	2	1	0
0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0
Reset: 0	0	1	0	0	0	0	0

S8C, S4C, S2C, S1C: Conversion Sequence Length

0000 = 8 conversions
0001 = 1 conversions
0010 = 2 conversions
0011 = 3 conversions
0100 = 4conversions
0101 = 5 conversions
0110 = 6 conversions
0111 = 7 conversions
1xxx = 8 conversions

FIFO: Result register FIFO mode (always set this bit to '0')*

FRZ1, FRZ0: Background debug freeze enable bit (ignore these bits)*

- Writing to ATD0CTL3 will abort any current conversion sequence

ATD Control Register 4 (ATD0CTL4)

7	6	5	4	3	2	1	0
SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
Reset: 0	0	0	0	0	1	0	1

SRES8: ATD resolution select bit ('0': 10-bit operation, '1': 8-bit operation)

SMP1, SMP0: Select Sample Time bits (see next slide)

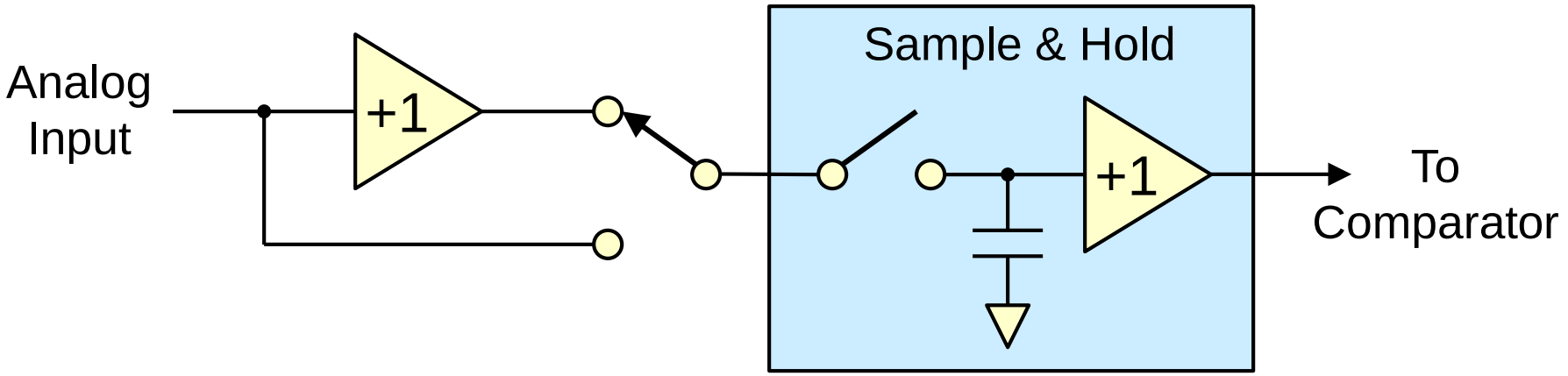
PRS4 ~ PRS0: ATD clock pre-scaler bits.

$$ATDclock = \frac{Eclock}{PRS + 1} \times 0.5$$

(ATD clock must be between 500kHz and 2 MHz)

- Writing to ATD0CTL4 will abort any current conversion sequence

Sample Time Select



- Sampling process has two stages:
  - Initially a unity gain sampling amplifier buffers input signal for two cycles to rapidly charge sample capacitor to almost the input potential
  - Sample buffer is then disconnected and input is connected directly to sample capacitor for 2, 4, 8 or 16 cycles

SMP1	SMP0	Length of 2nd Phase of Sample Time
0	0	2 ATD clock periods
0	1	4 ATD clock periods
1	0	8 ATD clock periods
1	1	16 ATD clock periods

ATD Control Register 5 (ATD0CTL5)

	7	6	5	4	3	2	1	0
	DJM	DSGN	SCAN	MULT	0	CC	CB	CA
Reset:	0	0	0	0	0	0	0	0

DJM: Result register data justification (‘0’: left justified, ‘1’: right justified)#

DSGN: Result register signed representation (‘0’: unsigned, ‘1’: signed)#

SCAN: Continuous channel scan bit (always set this bit to ‘0’)*

MULT: Enable multichannel conversion bit

‘0’: Sample only one channel (specified by CC/CB/CA)

‘1’: Sample across several channels (start with CC/CB/CA; number of channels specified by sequence length)

CC, CB, CA: Channel select code

Three bit code to select input channel (if MULT=0) or first channel in incrementing sequence (if MULT=1)

cannot be signed and right justified

Notes on ATD0CTL5

SRES8	DJM	DSGN	Result Data Format	Bit Mapping
1	0	0	8-bit/left-justified/unsigned	bits 15 ~ 8
1	0	1	8-bit/left-justified/signed	bits 15 ~ 8
1	1	x	8-bit/right-justified/unsigned	bits 7 ~ 0
0	0	0	10-bit/left-justified/unsigned	bits 15 ~ 6
0	0	1	10-bit/left-justified/signed	bits 15 ~ 6
0	1	x	10-bit/right-justified/unsigned	bits 9 ~ 0

- Writing to ATD0CTL5 will abort any current conversion sequence ***and start a new conversion sequence***

Input Analog Voltage to Output Code

- Left justified codes:

Input signal (V) V_{RL} **= 0.000 V** V_{RH} **= 5.120 V**	**Signed 8-bit codes (MSByte)**	**Unsigned 8-bit codes (MSByte)**	**Signed 10-bit codes**	**Unsigned 10-bit codes**
5.115	7F	FF	7FC0	FFC0
5.100	7F	FF	7F00	FF00
2.580	01	81	0100	8100
2.565	00	80	0040	8040
2.560	00	80	0000	8000
2.555	00	80	FFC0	7FC0
2.540	FF	7F	FF00	7F00
0.020	81	01	8100	0100
0.005	80	00	8040	0040
0.000	80	00	8000	0000

Input Analog Voltage to Output Code

- Right justified codes:

Input signal (V) V_{RL} = 0.000 V V_{RH} = 5.120 V	Signed 8-bit codes (LSByte)	Unsigned 8-bit codes (LSByte)	Signed 10-bit codes	Unsigned 10-bit codes
5.115	not allowed	FF	not allowed	03FF
5.100		FF		03FC
2.580		81		0204
2.565		80		0201
2.560		80		0200
2.555		80		01FF
2.540		7F		01FC
0.020		01		0004
0.005		00		0001
0.000		00		0000

ATD Status Register 0 (ATD0STAT0)

7	6	5	4	3	2	1	0
SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0
Reset: 0	0	0	0	0	0	0	0

SCF: Sequence complete flag

ETORF: External trigger overrun flag* (indicates trigger occurred while previous conversion still in progress)

FIFOR: FIFO overrun flag* (indicates result register has been written before corresponding conversion complete flag (CCF) has been cleared)

CC2, CC1, CC0: Conversion Counter*
Points to the result register that will receive the result of current conversion

- **ATD Result Registers ATD0DR0 ~ ATD0DR7**
  - eight 16-bit result registers

ATD Input Enable Register (ATD0DIEN)

7	6	5	4	3	2	1	0
IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN0
Reset: 0	0	0	0	0	0	0	0

IENx: When IENx=1, pin ANx can be used as a general purpose parallel port digital input pin rather than an A/D analog input pin

In this case, digital inputs are read from corresponding bit positions of **PTAD0**

ATD Conversion Timings:

ATD clock frequency	resolution	converter time	2+2 sample clocks	2+4 sample clocks	2+8 sample clocks	2+16 sample clocks
2 MHz 2 MHz	8-bit(1) 10-bit(2)	4 µs 5 µs	2 µs	3 µs	5 µs	9 µs
500 KHz 500 KHz	8-bit 10-bit	16 µs 20 µs	8 µs	12 µs	20 µs	36 µs

Note. 1. The fastest 8-bit resolution conversion time is 4 µs + 2 µs = 6 µs.
2. The fastest 10-bit resolution conversion time is 5 µs + 2 µs = 7 µs.

Multichannel Example

- Assume the following setup:
  - Channel select code (CC ~ CA) of ATD0CTL5 = 6
  - Conversion sequence length (S8C ~ S1C) of ATD0CTL3 = 5
  - MULT bit of ATD0CTL5 is set to ‘1’ (i.e. multiple channels)

- How would conversion results be stored in data registers?

Analog Channel	Result stored in: (non-FIFO mode)
AN6	ATD0DR0
AN7	ATD0DR1
AN0	ATD0DR2
AN1	ATD0DR3
AN2	ATD0DR4

Procedure for Performing A/D Conversion

1. Set up A/D power supply & reference voltages:
   - V_{DDA}: connect to 5V (nominal – normally same value as V_{DD})
   - V_{SSA}: connect to 0V (nominal – normally same value as V_{SS})
   - V_{RH}: connect to 5V (nominal) $V_{RH} \leq V_{DDA}$
   - V_{RL}: connect to 0V (nominal) $V_{RHL} \geq V_{SSA}$)
2. If transducer output is not in range (V_{RL} ~ V_{RH}), use a signal conditioning circuit to shift and scale it to match this range
3. Configure ATD control register 2 and wait 20μs for ATD circuits to stabilize
4. Configure ATD control registers 3 and 4.
5. Select appropriate channel(s) and operation modes by programming ATD control register 5. Writing to ATD0CTL5 starts conversion sequence
6. Wait until SCF flag of ATD0STAT0 is set, then collect conversion results and store in memory

Example: A/D Initialization

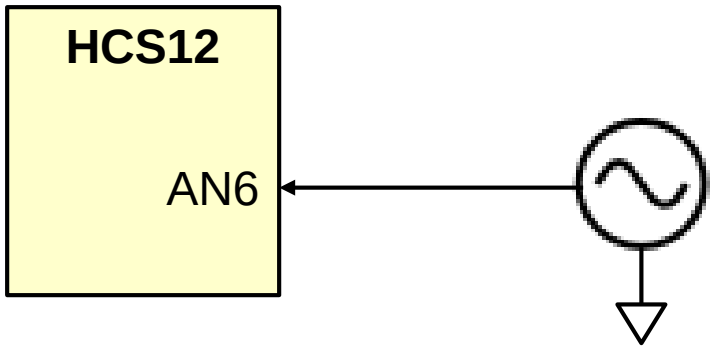
- Write a subroutine to initialize the AD0 converter with the following settings:
  - Software initiated conversions (no external trigger)
  - Fast ATD flag clear all
  - No interrupts
  - No FIFO mode
  - 10-bit operation
  - 2 MHz conversion clock (assume E-clock = 8 MHz)
  - 2 conversion clock periods for second stage sample time
  - Sequence of n ($1 \leq n \leq 8$) conversions where n is passed in acc A
- Need to write following data into ATD registers:
  - ATD0CTL2: $C0
  - ATD0CTL3: $00 + (n<<3)
  - ATD0CTL4: $01
- We do not write to ATD0CTL5 at this stage because that would actually start a conversion

A/D Initialization Code

```
            include  “hcs12.inc”

openAD0:    movb     $C0, ATD0CTL2
            jsr      wait20us          ; wait for A/D circuits to stabilize
            ldab     #8
            mult                       ; left shift sequence # by 3 places
            stab     ATD0CTL3
            movb     #$01, ATD0CTL4
            rts

wait20us:   movb     #$90, TSCR2       ; enable TCNT & fast timer clear
            movb     #0, TSCR1         ; set TCNT pre-scaler to 1
            bset     TIOS, $01         ; enable OC0
            ldd      TCNT              ; start OC0 operation
            addd     #160              ; 160 clock cycles = 20 us
            std      TC0
            brclr    TFLG1, $01, *     ; wait for time-out
            rts
```

Example: Perform 20 A/D Conversions

- Write a program to perform A/D conversions on the analog signal connected to pin AN6. Collect 20 10-bit conversion results and store them in consecutive 16-bit memory locations starting at $1000. Data should be unsigned and right justified.



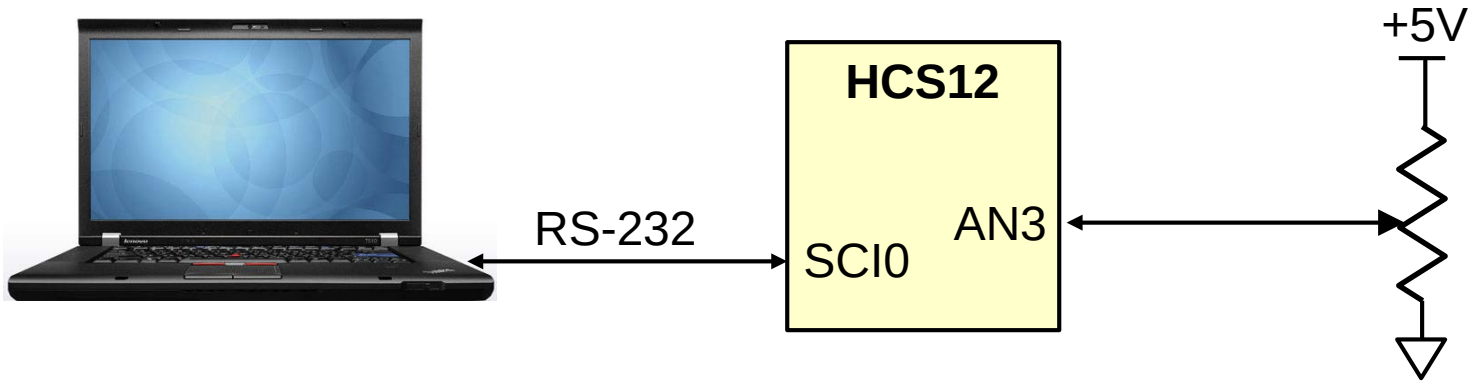
- Solution: Initialize A/D to perform sequence of four conversions and then initiate five conversion sequences

20 Conversions Code

```
        include  “hcs12.inc”

        ORG      $4000
        lds      #$4800
        ldx      #$1000              ; use X as data pointer
        ldaa     #4                  ; sequence length = 4
        jsr      openAD0             ; initialize A/D converter
        ldy      #5                  ; loop counter
loop5:  movb     #$86, ATD0CTL5      ; start  ch6 A/D conv. sequence
        brclr    ATD0STAT0, $80, *   ; wait for sequence complete flag
        movw     ATD0DR0, 2, x+      ; collect and save results
        movw     ATD0DR1, 2, x+      ; incrementing pointer by 2
        movw     ATD0DR2, 2, x+
        movw     ATD0DR3, 2, x+
        dbne     y, loop5            ; are we done?
        swi
```

Example: Record Potentiometer Output Voltage

- Write a program to read the voltage from a potentiometer connected between 0 and +5 volts. The wiper of the potentiometer should be connected to the AN3 pin of AD0. Read the voltage once every 2 seconds and print out (to 2 sig. figs.) on a terminal using the SPI0 RS-232 serial output



- Solution: The conversion result 1023 will correspond to +5V. To convert A/D output to volts, divide by 204.6.
  - Since we don't have floating point, this can be done by multiplying by 10 and then dividing by 2046

Potentiometer Output Code

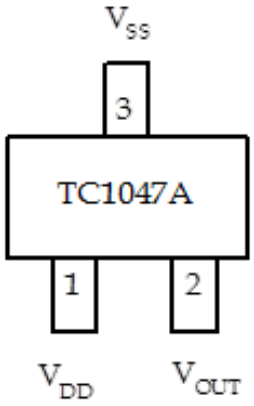
```
            include   “hcs12.inc”

CR:         equ       $0D                ; ascii carriage return
LF:         equ       $0A                ; ascii line feed

            ORG       $5000
headbuf:    dc.b      “Voltage = ”, 0
numbuf:     dc.b      “0.0 V”, CR, LF, 0 ; calculated digits stored here

ORG         $4000
            lds       #$4800
            ldaa      #1                 ; sequence length = 1
            jsr       openAD0            ; initialize A/D converter
forever:    movb      #$83, ATD0CTL5     ; start  ch3 A/D conv. sequence
            brclr     ATD0STAT0, $80, *  ; wait for sequence complete flag
            ldd       ATD0DR0            ; get 10 bit data
            ldy       #10                ; multiply data by 10
            emul                         ; L.S. result in D
            ldx       #2046              ; divide by 2046
            idiv                         ; quotient in X, rem in D
```

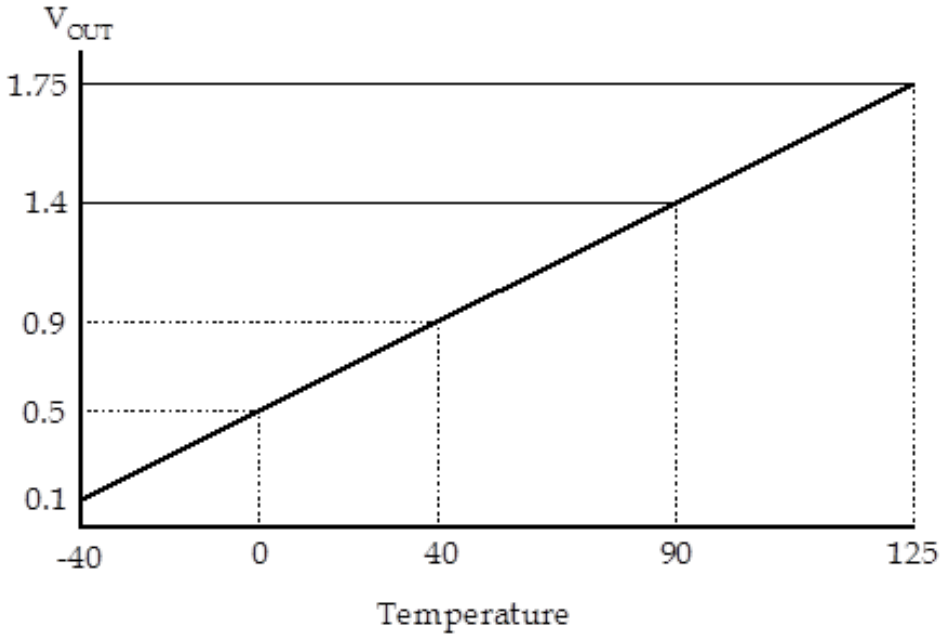
Potentiometer Output Code (cont.)

```
exg     x, d            ; swap X and D
addd    #$30            ; convert MSD to ascii
stab    numbuf          ; store in string buffer

tfr     x, d            ; remainder back in D
ldy     #10             ; multiply remainder by 10
emul
ldx     #2046           ; and divide by 2046
idiv                    ; quotient in X
tfr     x, d            ; fractional digit in D
addd    #$30            ; convert LSD to ascii
stab    numbuf+2        ; store in string buffer

ldx     #headbuf
jsr     putsSCI0        ; output header string
ldx     #numbuf
jsr     putsSCI0        ; output number string
ldy     #20
jsr     delayby100ms    ; wait for 2 seconds
bra     forever
```

Example: Temperature Sensor TC1047A

- Three pin temperature sensor whose output voltage is directly proportional to ambient temperature
- Measures temp. in range of -40º C to +125º C

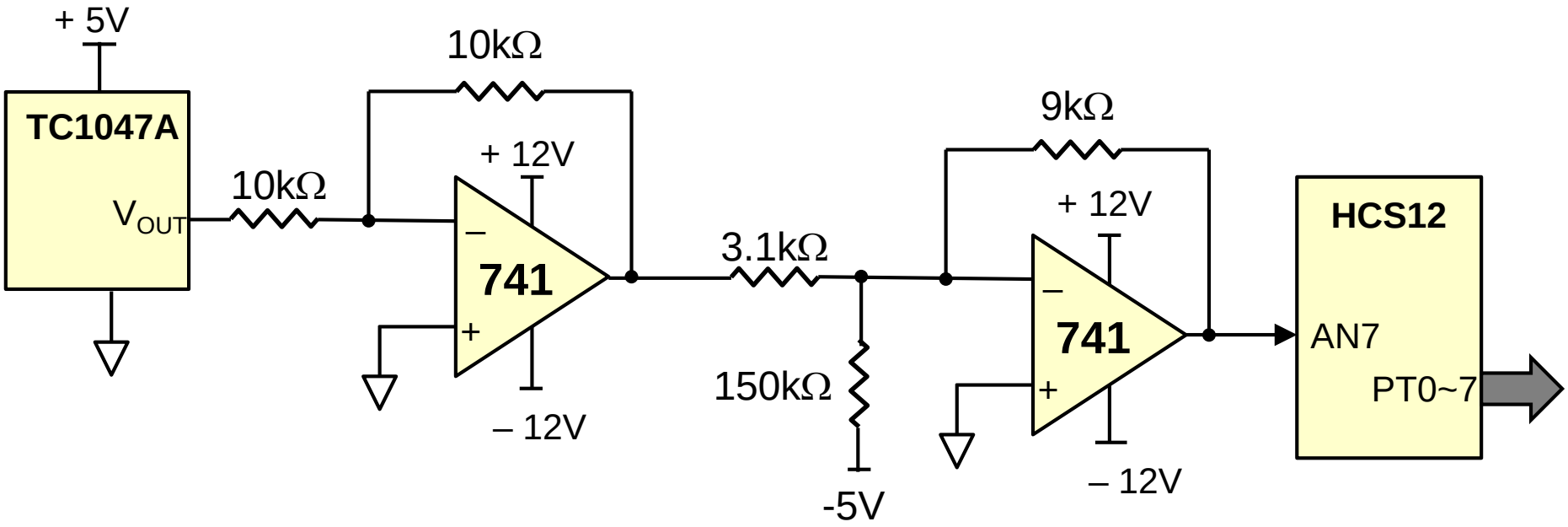




- Describe circuit connection and write program to read temperature five times per second and output temperature (signed and rounded to nearest degree C) to external peripheral connected to port T.

Temperature Sensor: Signal Conditioning

- Output of temperature sensor is 0.1V ~ 1.75V
- To effectively use input range of A/D (0 ~ 5V), need to:
  - multiply by (5/1.74 = 2.874) which gives a range of 0.29 ~ 5.29V
  - offset by (–0.29V) which gives a range of 0 ~ 5V



Temperature Sensor: Digital Data Scaling

- Following signal conditioning, range of analog input is 0 ~ 5V
- Range of digital result is 0 – 1023
- Represents temperature range of 165 ºC
- Need to divide by 1023/165 = 6.2
  - i.e. multiply by 10 and divide by 62
  - this gives a range of 0 ~ 165
- Finally subtract 40 to give a range of – 40 to + 125

Temperature Sensor Code

```
            include   “hcs12.inc”

ORG         $4000
            lds       #$4800
            movb      $FF, DDRT          ; set up Port T as 8-bit output
            ldaa      #1                 ; sequence length = 1
            jsr       openAD0            ; initialize A/D converter
```

Temperature Sensor Code (cont.)

```
forever:  movb   #$87, ATD0CTL5      ; start  ch7 A/D conv. sequence
          brclr  ATD0STAT0, $80, *   ; wait for sequence complete flag
          ldd    ATD0DR0             ; get 10 bit data
          ldy    #10                 ; multiply data by 10
          emul                       ; L.S. result in D
          addd   #31                 ; to round divisor
          ldx    #62                 ; divide by 62
          idiv                       ; quotient in X, rem in D
          tfr    x, d                ; rounded quotient in D
          subd   #40                 ; to give signed result
          stab   PTT
          ldy    #2                  ; wait for 200 ms
          jsr    delayby100ms
          bra    forever
```