Chapter 12: Analog-to-Digital Converter

A/D Conversion

- Almost any measurable quantity, for example, weight, humidity, pressure, mass or air flow, temperature, light intensity, speed, and so on, is analog and non-electrical in nature.
- A transducer is normally used to convert a non-electrical quantity into an electrical voltage so that the quantity can be processed by a computer.
- Analog signals must be represented in digital format in order to be processed by the digital computer.
- An analog to digital (A/D) converter can convert an electrical voltage to a digital value.
- The output from a transducer may not be appropriate for A/D conversion. A signal-conditioning circuit, including a voltage shifter and scaler, may be needed to transform the transducer output voltage into a range that can be best handled by the A/D converter.
- The accuracy of an A/D converter is dictated by the number of bits it used to represent the digital value (e.g., 10-bit A/D converter in HCS12).

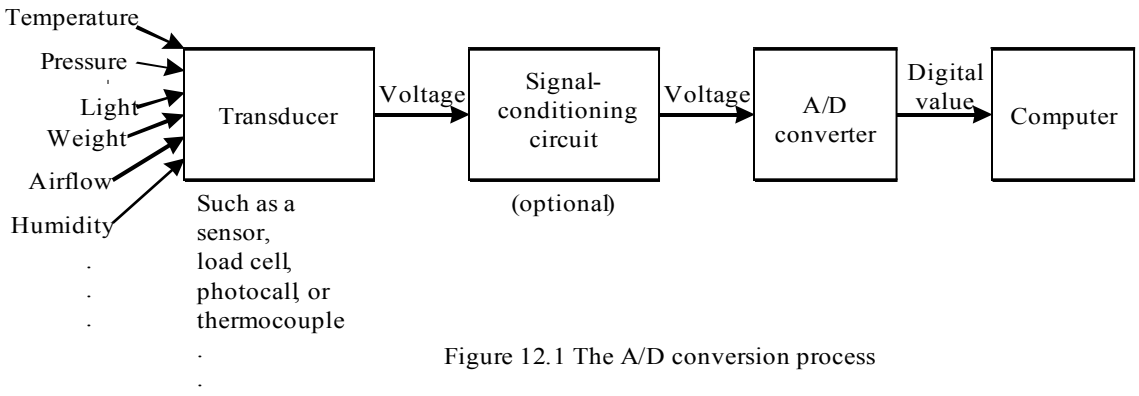

Figure 12.1 The A/D conversion process

Analog Voltage and Digital Code Characteristic

- An ideal A/D converter should have a linear input-output characteristic, but needing an infinite number of bits to represent the A/D conversion result.
- An n-bit A/D converter uses n bits to represent the A/D conversion result and has 2^n possible output digital values (or levels).
- The value of $V_{DD}/2^n$ is the resolution (or step size) of this A/D converter. The more bits in use, the higher the resolution and the smaller the quantization error will be.

- The area above and below the dotted line is called quantization error. This n-bit A/D converter has an average quantization error of $V_{DD}/2^{n+1}$.

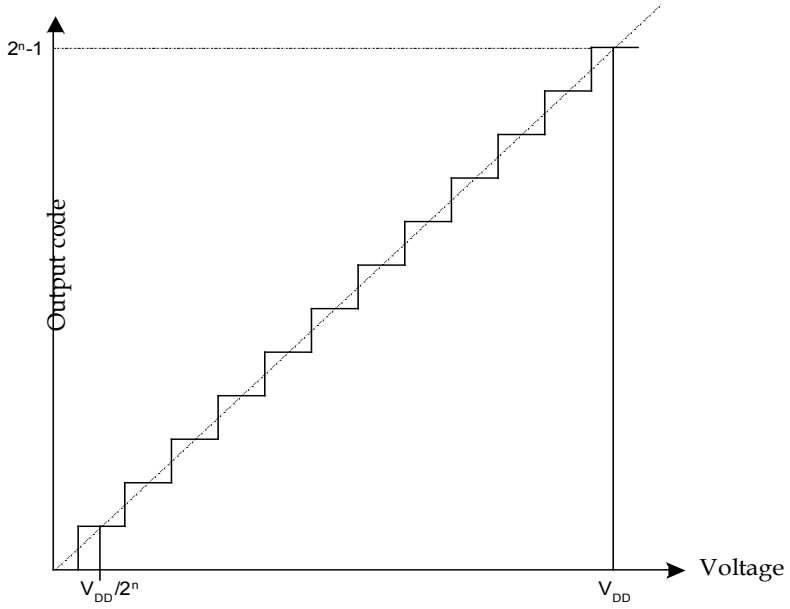

Figure 12.3 Output characteristic of an ideal n-bit A/D converter

Successive-Approximation Method

An A/D converter approximates the analog signal to a n-bit code in n steps. It may be used for low-frequency applications with large DC noise, such as an electrocardiography.

1. Initialize the successive-approximation register (SAR) to 0 and then performs a series of guessing.
2. Starting from the most significant bit of the SAR and work toward the least significant bit.
   - For each bit, guess the bit to be a 1.
   - Converts the digital value of the SAR to an analog voltage in the feedback circuit.
   - Compares the D/A output with the analog input.
   - Clears the bit to 0 if the D/A output is larger, which indicates that the guess is wrong.

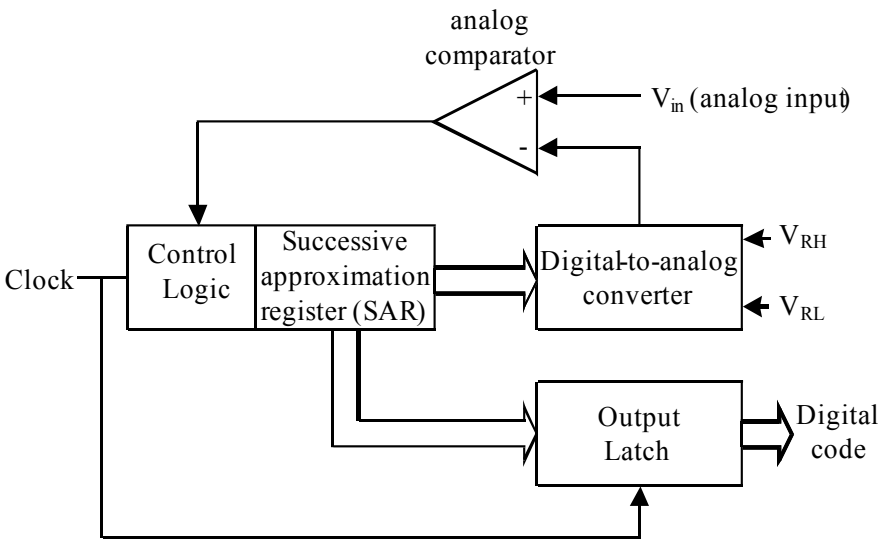

Figure 12.4 Block diagram of a successive approximation A/D converter

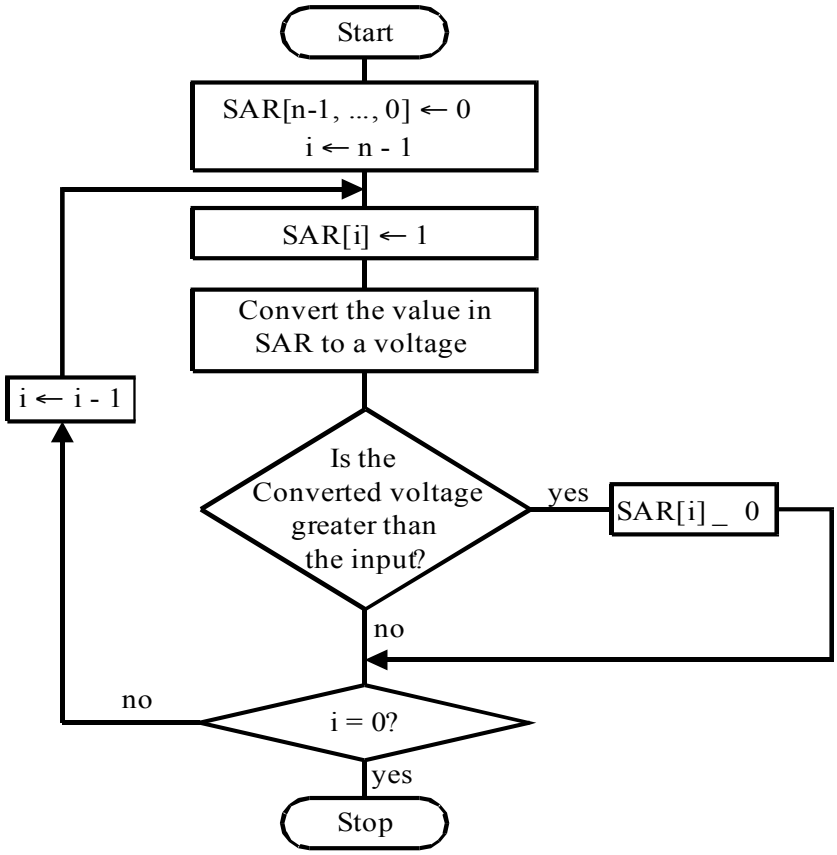

Figure 12.5 Successive approximation A/D conversion method

Signal-Conditioning Circuits

An A/D converter often needs a circuit to scale and/or shift the transducer output to fit its input range.

Optimal Voltage Range for the A/D Converter

- An A/D converter needs a low reference voltage (V_{LREF}), which is often set to ground 0V, and a high reference voltage (V_{HREF}), which is often set to the power supply V_{DD}, to operate.
- Most A/D converters are **ratiometric**:
  - A 0V analog input is converted to the digital value 0.
  - A V_{DD} analog input is converted to the digital value 2^n-1.
  - A V_k analog input is converted to the digital value $k = (V_k / V_{DD}) \times (2^n - 1)$.
  - n is the number of bits, so-called the **resolution** of the A/D converter.
- To get the most accurate conversion results, we should scale and shift the analog signal to the whole dynamic range $V_{LREF} \sim V_{HREF}$ of the A/D converter first.
- The A/D conversion result (digital value) k corresponds to an analog voltage V_k by

$$V_k = V_{LREF} + (V_{HREF} - V_{HREF}) \times k \div (2^n - 1)$$

Example: Suppose that there is a 10-bit A/D converter with $V_{LREF} = 1$ V and $V_{HREF} = 4$V. Find the corresponding voltage values for the A/D conversion results of 25, 80, 240, 500, 720, 800, and 900.

Whole dynamic range = $V_{HREF} - V_{LREF} = 4V - 1V = 3V$.

$V(25) = 1\text{ V} + (3 \times 25) \div (2^{10} - 1) = 1.07\text{ V}$
$V(80) = 1\text{ V} + (3 \times 80) \div (2^{10} - 1) = 1.23\text{ V}$
$V(240) = 1\text{ V} + (3 \times 240) \div (2^{10} - 1) = 1.70\text{ V}$
$V(500) = 1\text{ V} + (3 \times 500) \div (2^{10} - 1) = 2.47\text{ V}$
$V(720) = 1\text{ V} + (3 \times 720) \div (2^{10} - 1) = 3.11\text{ V}$
$V(800) = 1\text{ V} + (3 \times 800) \div (2^{10} - 1) = 3.35\text{ V}$
$V(900) = 1\text{ V} + (3 \times 900) \div (2^{10} - 1) = 3.64\text{ V}$

Voltage Scaling Circuit

- Some transducer has the an output voltage in the range of $0 \sim V_Z$, where $V_z < V_{DD}$.
- Because V_z sometimes can be much smaller than V_{DD}, the A/D converter cannot take advantage of the available full dynamic range, and therefore conversion results can be very inaccurate.

- The voltage scaling circuit can be used to improve the accuracy because it allows the A/D converter to utilize its full dynamic range.
- The voltage gain is given by $A_v = V_{out}/V_{in} = (R_1+R_2)/R_1 = 1 + R_1/R_2$.

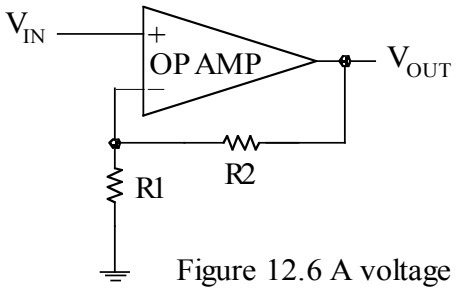

Figure 12.6 A voltage scaler

Example: Choose appropriate values of R1 and R2 to scale a voltage in the range of 0~200mV to 0~5V.

Set $V_{in} = 0.2$V and $V_{out} = 5$V. This requires are gain of (5V/0.2V) = 25.

Apply $A_V = 1 + R_2/R_1 = 25$ => R2/R1 = 24. Thus, choosing R_1 = 10,000 ohms and R_2 = 240,000 ohms will achieve the desired ratio.

Voltage Scaling and Shifting Circuit

- Some transducers have output in the range of V_1~V_2, where $V_2 > V_1$.
- V_1 can be negative and V_2 can be smaller than V_{DD}.
- To take advantage of the whole dynamic range of the A/D converter, the scaling and shifting circuit will shift and scale the transducer output to the range of 0~V_{DD}.
- By choosing appropriate values for V_1 and resistors, the transducer output can be scaled and shifted to 0~V_{DD} by the following voltage translation circuit.

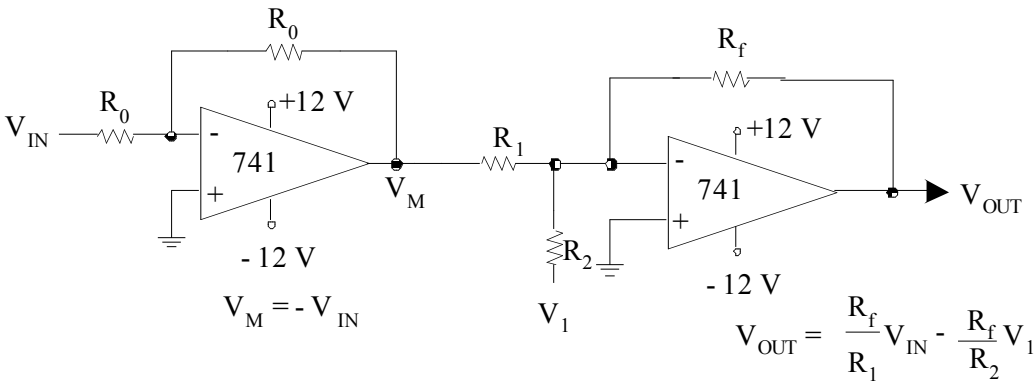

Figure 12.7 Level shifting and scaling circuit

Example: Choose the appropriate values of resistors and the adjusting voltage so that the shifting/scaling circuit can shift the voltage from –1.2V ~ 3.0V to 0V ~ 5V.

Applying $0 = -1.2\,(R_f/R_1) - (R_f/R_2)V_1$ & $5 = 3.0\,(R_f/R_1) - (R_f/R_2)V_1$

For example, by choosing $V_1 = -5V$, $R_1 = R_0 = 10{,}000$ ohms, and $R_f = 120{,}000$ ohms, R_2 is then found to be 50,000 ohms.

HCS12 A/D Converter

- A HCS12 member may have one or two 8-channel, 10-bit A/D converters. The successive-approximation method and ratiometric conversion are used.

- After the A/D converter has been enabled, the A/D conversion will be started when a control register is written into or by an external trigger input.

- A clock signal is required to control the A/D conversion that must have a maximum conversion frequency between 2 MHz and 500 kHz.
- The clock source to the converter is the E-clock divided by a prescale factor.
- At 2 MHz conversion clock, a sample may take 6 us (7 us) to complete a conversion for 8-bit (10-bit) resolution.

- When a sequence consists of multiple samples, they may be taken from one channel or multiple channels.
- The conversion result is stored in 16-bit result registers and can be right-justified unsigned, left-justified signed, and left-justified unsigned.

- The sample selected by the analog multiplexer is amplified before charging the sample capacitor in the sample and hold circuit.

- The control registers and associated logic select the conversion resolution, multiplexer input, and conversion sequencing mode, sample time, and ADC clock cycle.

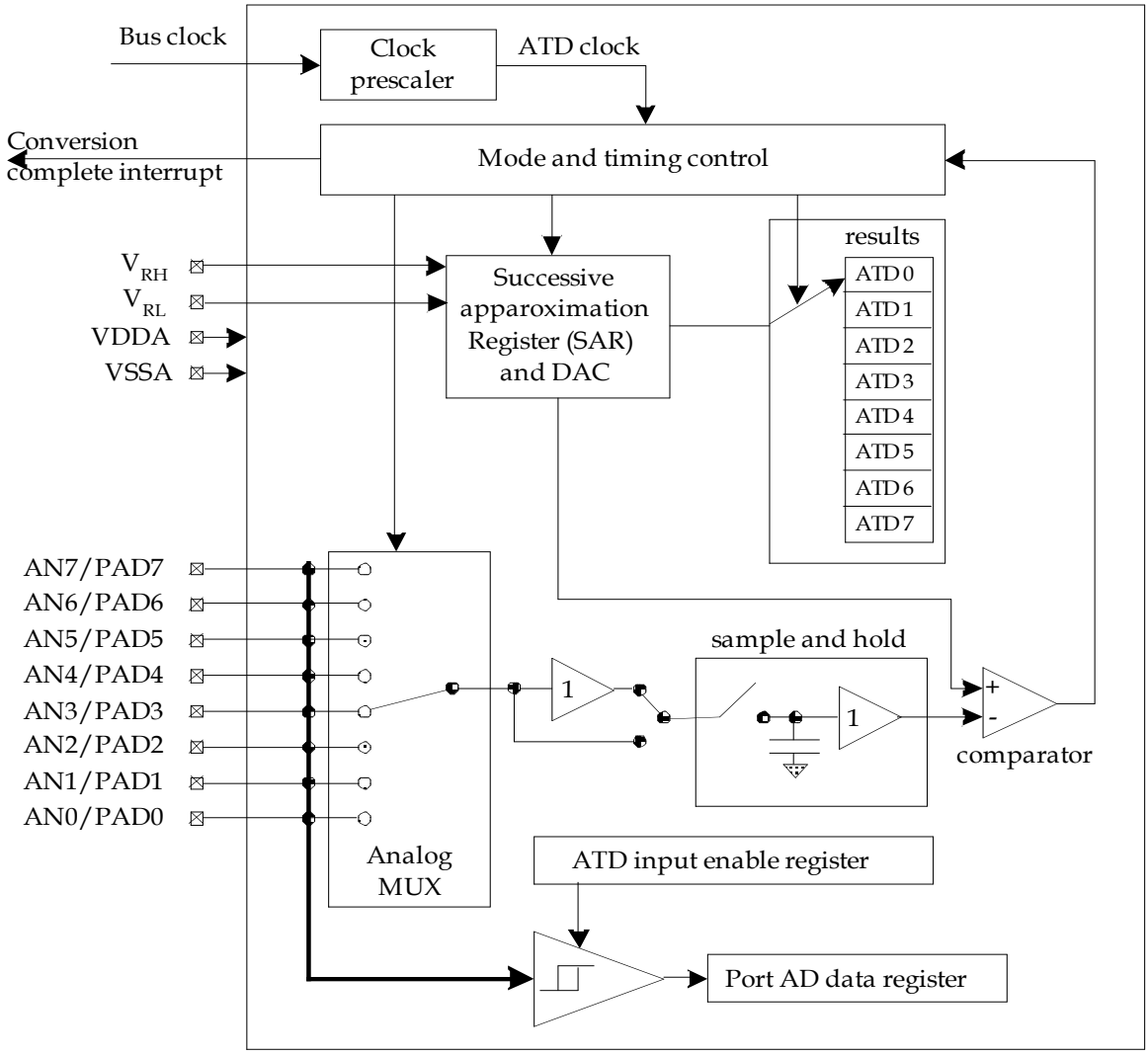

Figure 12.8 The HCS12 ATD block diagram

Signal Pins Related to A/D Converter

- Each of the two A/D modules has eight analog inputs.
- The AD0 module has analog input pins AN0 ~ AN7.
- The AD1 module has analog input pins AN8 ~ AN15.
- The AN7 pin can be optionally used as the trigger input pin for AD0 module.
- The AN15 pin can be optionally used as the trigger input pin for AD1 module.
- V_{RH} and V_{RL} are the high and low reference voltage input.
- V_{DDA} and V_{SSA} are power supply and ground inputs for the A/D converters.
- The high reference voltage must be between $V_{DDA}/2$ and V_{DDA}. The low reference voltage must be between V_{SSA} and $V_{DDA}/2$.
- Accuracy is only guaranteed for $V_{RL} = 0$ V and $V_{RH} = 5$ V.

ATD Registers

Each one of the two A/D modules (labeled as x = 0 or 1) has the following registers:

- Six control registers, ATDxCTL0 ~ ATDxCTL5, that control the overall operation of the module. (ATDxCTL0 and ATDxCTL1 are used for factory testing only.)
- Two status registers, ATDxSTAT0 and ATDxSTAT1, that record the operation status of the converter.
- Two testing registers, ATDxTEST0 and ATDxTEST1, that are dedicated to factory testing purposes.
- One input enable register, ATDxDIEN, that enables analog pins to be used as digital input.
- One port data register, PortADx.
- Eight 16-bit result registers, ATDxDR0~ATDxDR7.

ATD Control Register 2 (ATD0CTL2 and ATD1CTL2)

- This register controls power down, interrupt, and the external trigger.
- Writes to this register will abort the current conversion sequence but will not start a new conversion sequence.
- Setting the ADPU bit (bit 7) of this register enables the A/D converter.
- By setting the bit 6 of this register, the fast ATD flag clear feature is enabled.
- Bit 0 indicates whether the A/D conversion is completed.
- A/D complete interrupt enabling/disabling is controlled by bit 1.
- A/D external triggering can be edge-triggering or level-triggering, chosen by bits 3 and 4 in this register.

Table 12.1 External trigger configurations

ETRIGLE	ETRIGP	External trigger sensitivity
0	0	falling edge
0	1	rising edge
1	0	low level
1	1	high level

	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 ATD
1 = normal ATD operation
AFFC: ATD fast flag clear all bit
0 = ATD flag is cleared normally, i.e., read the status register before reading the result register
1 = any access to a result register will cause the associated CCF flag to clear automatically if it is set at the time
AWAI: ATD power down in wait mode bit
0 = ATD continues to run when the HCS12 is in wait mode
1 = halt conversion and power down ATD during wait mode
ETRIGLE: External trigger level/edge control
This bit controls the sensitivity of the external trigger signal. Details are shown in Table 12.1.
ETRIGP: External trigger polarity
This bit controls the polarity of the external trigger signal. See Table 12.1 for details.
ETRIGE: External trigger mode enable
0 = disable external trigger on ATD channel 7
1 = enable external trigger on ATD channel 7
ASCIE: ATD sequence complete interrupt enable bit
0 = disables ATD interrupt
1 = enables ATD interrupt on sequence complete (ASCIF = 1)
ASCIF: ATD sequence complete interrupt flag
0 = no ATD interrupt occurred
1 = ATD sequence complete interrupt pending

Figure 10.9 ATD control register 2 (ATDxCTL2, x = 0 or 1)

ATD Control Register 3 (ATD0CTL3 and ATD1CTL3)

- This register sets the conversion sequence length, enables/disables the FIFO mode for result registers, and controls the ATD behavior in freeze mode.
- Writes to this register will abort the current conversion sequence but will not start a new conversion.
- If the FIFO bit is 0, the result of the first conversion appears in the first result register, the second conversion appears in the second result register, and so on.
- If the FIFO bit is 1, then the result of the first conversion appears in the result register specified by the conversion counter.
- The conversion counter, whose value is stored in bits 0-2 of the ATD status register 0, does not reset at the start of a new conversion sequence.

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

S8C,S4C,S2C,S1C: Conversion sequence limit
- 0000 = 8 conversions
- 0001 = 1 conversion
- 0010 = 2 conversions
- 0011 = 3 conversions
- 0100 = 4 conversions
- 0101 = 5 conversions
- 0110 = 6 conversions
- 0111 = 7 conversions
- 1xxx = 8 conversions

FIFO: Result register FIFO mode
- 0 = conversion results are placed in the corresponding result register up to the selected sequence length
- 1 = conversion results are placed in consecutive result registers (wrap around at end)

FRZ1 and FRZ0: background debug (freeze) enable bit
- 00: continue conversions in active background mode
- 01: reserved
- 10: finish current conversion, then freeze
- 11: freeze immediately when background mode is active

Figure 10.10 ATD control register 3 (ATDxCTL3, x = 0 or 1)

ATD Control Register 4 (ATD0CTL4 and ATD1CTL4)

- This register sets the conversion clock frequency, the length of the second phase of the sample time, and the resolution of the A/D conversion.
- Writes to this register will abort the current conversion but will not start a new conversion sequence.

- There are two stages in the analog signal sample time. The first stage is fixed at two conversion clock periods. The second stage is selected by the SMP1 and SMP2 bits of this register.

- Needs a clock signal to operate. When selecting the prescale factor, make sure that the resultant clock frequency is in the range of 500 kHz and 2 MHz.

ATD clock = 24 MHz / [2 (PRS + 1)] => PRS = 5 = %00101 for 2 MHz

```
MOVB  #$85, ATD0CL4     ; $85=1000 0101; 10-bit resolution, 2 conversion
                        ; clock periods, 2MHz clock frequency
```

	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 and SMP0: select sample time bits
These bits are used to select the length of the second phase of the sample time in units of ATD conversion clock cycles. See Table 12.2.

PRS4--PRS0: ATD clock prescaler bits
These five bits are the binary value prescaler value PRS. The ATD conversion clock frequency is calculated as follows:

$$\text{ATDclock} = \frac{[\text{bus clock}]}{\text{PRS} + 1} \times 0.5$$

The ATD conversion frequency must be between 500KHz and 2 MHz. The clock prescaler values are shown in Table 12.3.

Figure 12.11 ATD control register 4 (ATDxCTL4, x = 0 or 1)

Table 12.2 Sample time select

SMP1	SMP0	Length of 2nd phase of sample time
0	0	2 A/D conversion clock periods
0	1	4 A/D conversion clock periods
1	0	8 A/D conversion clock periods
1	1	16 A/D conversion clock periods

ATD Control Register 5 (ATD0CTL5 and ATD1CTL5)

- Selects the type of conversion sequence and the analog input channels sampled.
- Writes to this register will abort the current conversion sequence and start a new conversion sequence.
- Bits 2-0 (CC, CB, and CA) select which one of the AN0~7 channels to be converted.
- Bits 6 and 7 (DSGN and DJM), together with bit 7 (SRES8) of ATDxCTL4, select the result data formats, such as signed and unsigned, left justified and right justified output codes.
- When a single-channel mode is selected (MULT =0 in bit 4), the analog channel to be converted is specified by the CC~CA.
- When multichannel mode is selected, the first analog channel examined is specified by the value of CC~CA; subsequent channels sampled in the sequence are determined by incrementing the value of CC~CA.

	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 data in the result registers
1 = right justified data in the result registers
DSGN: Result register data signed or unsigned representation
0 = unsigned data representation in the result registers
1 = signed data representation in the result registers(not available in right justification)
SCAN: Enable continuous channel scan bit
0 = single conversion sequence
1 = continuous conversion sequences(scan mode)
MULT: Enable multichannel conversion bit
0 = sample only one channel
1 = sample across several channels
CC, CB, and CA: Channel select code
The channel selection is shown in Table12.4.

Figure 12.12 ATD control register 5 (ATDxCTL5, x = 0 or 1)

Table 12.4 Analog input channel select code

CC	CB	CA	analog input channel
0	0	0	AN0
0	0	1	AN1
0	1	0	AN2
0	1	1	AN3
1	0	0	AN4
1	0	1	AN5
1	1	0	AN6
1	1	1	AN7

Table 11.5 Available result data formats

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

Table 11.6 Left justified, signed and unsigned ATD output codes

input signal V_{RL} = 0 V V_{RH} = 5.12 V	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	7F	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

ATD Status Register 0 (ATD0STAT0 and ATD1STAT0)

- Each A/D channel has two status registers for indicating whether a sequence of conversion is complete.
- This read-only register contains the sequence complete flag, overrun flags for external trigger and FIFO mode, and the conversion counter.

- If the AFFC flag in the ATDxCLT4 control register is set to 1, the SCF flag can be cleared by reading a result register. Otherwise, it can be cleared by writing a 1 to SCF or writing to ATDxCTL5 (to start a new conversion sequence).
- While in edge-triggered mode, if addition active edges are detected while a conversion sequence is in progress, the ETORF flag is set to 1.
- The ETORF flag is cleared by writing a 1 to ETORF, writing to ATDxCTL2, ATDxCLT3, or ATDxCLT4 (a conversion sequence is aborted), or writing to ATDxCTL5 (a new conversion sequence is started).
- The setting of the FIFOR bit indicates that a result register has been written to before its associated conversion complete (CCF7~0) flag in the ATDxSTAT1 status register has been cleared.
- The FIFOR flag is cleared when writing a 1 to FIFOR or writing to ATDxCTL5.

	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
0 = conversion sequence not completed
1 = conversion sequence has completed
ETORF: External trigger overrun flag
0 = no external trigger overrun has occurred
1 = external trigger overrun has occurred
FIFOR: FIFO overrun flag
0 = no overrun has occurred
1 = an overrun has occurred
CC2, CC1, CC0: conversion counter
The conversion counter points to the result register that will receive the result of the current conversion.
In non-FIFO mode, this counter is reset to 0 at the begin and end of the conversion.
In FIFO mode, this counter is not reset and will wrap around when its maximum value is reached.

Figure 12.13 ATD status register 1 (ATDxSTAT0, x = 0 or 1)

ATD Status Register 1 (ATD0STAT1 and ATD1STAT1)

- This register contains the conversion complete flags for all channels.
- A CCFx flag can be cleared by one of the following:
  - Write to ATDxCTL5 register (a new conversion is started)
  - If AFFC = 0 and read of ATDxSTAT1 followed by read of result register ATDxDRy.
  - If AFFC = 1 and read of result register ATDxDRy

	7	6	5	4	3	2	1	0
	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
reset:	0	0	0	0	0	0	0	0

CCFx: conversion complete flag x (x = 7~0)
0 = conversion number x not completed
1 = conversion number x has completed, result in ATDyDRx

Figure 12.15 ATD status register 1 (ATDxSTAT1, x = 0 or 1)

ATD Test Registers (ATDTESTH/L, ATD0TESTH/L, and ATD1TESTH/L)

- Each A/D converter has a 16-bit test register to facilitate the factory testing.
- End users do not need to pay much attention to this register.

	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	SC
reset:	0	0	0	0	0	0	0	0

SC: Special channel conversion bit
If this bit is set, the special channel conversion can be selected using CC, CB, and CA of the ATDxCTL5 register. Table 12.7 shows the selection.
0 = special channel conversions disabled
1 = special channel conversions enabled

Figure 12.14 ATD test register 1 (ATDxTEST1, x = 0 or 1)

ATD Input Enable Register (ATD0DIEN and ATD1DIEN)

- This register allows the user to enable a Port ADx pin as a digital input.

	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: ATD digital input enable on channel x (x = 0~7)
0 = disable digital input buffer to PTADx
1 = enable digital input buffer to PTADx

Figure 12.16 ATD input enable register (ATDxDIEN, x = 0 or 1)

Port AD Data Register (PortAD0 and PortAD1)

- When the A/D converter is not enabled, the associated port can be used as an input port only.
- The values of these pins can be read from the port AD data register.
- The port pins are shared with analog inputs AN0~AN7 and AN8~AN15.

ATD Conversion Result Registers (ATD0DR7H/L~ATD0DR0H/L and ATD1DR7H/L~ATD1DR0H/L)

- Each ATD channel has eight 16-bit result registers to hold the conversion result.
- Each result register is 16-bit and consists of a high and low 8-bit result registers.
- When 8-bit mode is chosen, the conversion result is stored in the high result register.
- In 10-bit mode, the high result register holds the upper eight bits of the conversion whereas the *highest two bits* (bits 7 and 6) of the low result register hold the lowest two bits of the conversion result.
- The result can be stored right-justified or left-justified, signed or unsigned. However, the conversion result is unsigned when it is stored right-justified.

Sample and Hold Stage

- A sample and hold stage accepts analog signals from the input multiplexer and stores them as charges on the sample capacitor.
- The sampling process has two stages.
    - The **first stage** of the sampling process takes two A/D clock cycles to charge the sample capacitor.
    - The **second stage** of the sampling process stores the charge in the storage node for a programmable 2, 4, 8, or 16 cycles.
- The conversion time of a sample is given by

$$\text{conversion time} = \frac{(\text{no. of bits in resolution} + \text{no. of programmed sample clocks} + 2)}{\text{ATD clock frequency}}$$

- The A/D conversion time is the sum of the converter time and the sample time.

Table 12.8 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.

Input Channel Wrap Around

In the case of a multiple channel conversion sequence (MULT=1 in ATDxCTL5), when the input selector goes past channel AN7, it wraps around to channel AN0.

Example: Assuming that S8C~S1C (ATD0CTL3) are set to 0101 and CC~CA (ATD0CTL5) are set to 110, what is the conversion sequence for this setting?

The first channel to be converted is AN6 as CC~CA = 110.
There are five channels to be converted as S8C~S1C=0101 = 5 conversions.
The conversion sequence is AN6, AN7, AN0, AN1, and AN2.

FIFO Mode

- All eight 16-bit result registers are organized into a circular ring.
- The conversion counter in the ATDxSTAT0 register specifies the result register to hold the current conversion result.
- In the FIFO mode, the conversion is not reset to 0 when a new conversion sequence is started.
- In the FIFO mode, the first conversion result may not be stored in ATDxDR0 because one can set the length of a conversion sequence to a value between 1 and 8 in S8C~S1C (ATD0CTL3).

Example: Assume that the following setting was programmed before a new conversion is started:

- The conversion counter value in the ATD0STAT0 register is 5.
- The channel select code of the ATD0CTL5 register is 6.
- The conversion sequence limit of the ATD0CTL3 register is set to 5.
- The MULT bit of the ATD0CTL5 register is set to 1.

How would the conversion results be stored when the FIFO mode is selected or not selected?

MULT=1 selects multiple conversions.
The conversion counter specifies the ATD0DR5 result register to hold the first conversion result.
The channel select code specifies AN6 as the first channel to be converted.
The conversion sequence limit specifies five channels to be converted.

Table 12.9 Conversion results storage

analog channel	result stored in (FIFO mode)	result stored in (non FIFO mode)
AN6	ATD0DR5	ATD0DR0
AN7	ATD0DR6	ATD0DR1
AN0	ATD0DR7	ATD0DR2
AN1	ATD0DR0	ATD0DR3
AN2	ATD0DR1	ATD0DR4

Procedure for Performing A/D Conversion

Step 1: Connect the hardware properly. The A/D-related pins must be connected as follows:

V_{DDA}: connect to 5 V.
V_{SSA}: connect to 0 V.
V_{RH} (or V_{RHx}, x = 0 or 1): 5 V
V_{RL} (or V_{RLX}, x = 0 or 1): 0 V

If the transducer output is not in the appropriate range, then we should use a signal conditioning circuit to shift and scale it to between V_{RL} and V_{RH}.

Step 2: Configure ATD control registers 2 to 4 properly and wait for the ATD to stabilize (need to wait for 5 μs).

Step 3: Select the appropriate channel(s) and operation modes by programming the ATD control register 5. Writing into the ATD control register 5 also starts the A/D conversion.

Step 4: Wait until the SCF flag of the status register ATDxSTAT0 set, then collect the A/D conversion results and store them in memory.

Example: Write a subroutine to initialize the ATD0 converter and start the conversion with the following parameters:

- Nonscan mode; Select channel 7 (single channel mode)
- Enable ATD fast flag clear; Stop ATD in wait mode; Disable interrupt
- Perform 4 conversions in a sequence
- Disable FIFO mode
- Finish current conversion then freeze when BDM becomes active
- 10-bit operation and 2 A/D clock periods of the second-stage sample time
- Choose 2 MHz as the conversion frequency for 24 MHz E clock
- Result is unsigned and right justified

The setting of ATD control register 2 to 5 are as follows:

ATD0CTL2

- Enable ATD0 (bit 7 = 1)
- Enable fast flag clear (bit 6 = 1)
- Stop ATD0 when in wait mode (bit 5 = 1)
- Disable external trigger on channel 7 (bits 4~ 2 = 000)
- Disable ATD interrupt (bit 1 = 0)
- Clear all other bits
- MOVB #$E0,ATD0CTL2

ATDCTL3

- Perform four conversions (bits 6~3 = 0100)
- Disable FIFO mode (bit =0)
- When the freeze mode is active, complete the current conversion then freeze (bits1~0 = 10)
- MOVB #$22,ATD0CTL3

ATDCTL4

- Select 10-bit operation (bit 7 = 0)
- Set sample time to two ATD clock period (bits 6~5 = 00)
- Set prescaler to 5 to select ATD clock to 2 MHz. (bits 4~0 = 00101).
- MOVB #$05,ATD0CTL4

ATDCTL5

- Select result register right justified (bit 7 = 1)
- Result is unsigned (bit 6 = 0)

- Nonscan mode (bit 5 = 0)
- Single channel mode (bit 4 = 0)
- Select channel 7 (bits 2~0 = 111)
- MOVB #$87,ATDCTL5

```
openAD0     movb   #$E0,ATD0CTL2
            jsr    delay20us          ; wait for 20 us
            movb   #$22,ATD0CTL3
            movb   #$05,ATD0CTL4
            rts

dealy20us   movb   #$90,TSCR1         ; enable TCNT and fast timer flag clear
            movb   #$00,TMSK2         ; set TCNT prescaler to 1
            bset   TIOS,$01           ; enable OC0
            ldd    TCNT               ; start an OC0 operation
            addd   #480               ;
            std    TC0
            brclr  TFLG1,C0F,*        ; wait for 20us
            rts
```

Example: Write a program to perform A/D conversion on the analog signal connected to the AN7 pin. Collect 20 A/D conversion results and store them at memory locations starting from $1000. Use the same configuration as in the above example to configure AD0 to perform 4 conversions in a sequence on channel AN7.

Need to write into the ATD0CTL5 five times to collect twenty samples.

```
#include "reg9s12.h"

            org    $2000
            lds    #$2000
            ldx    #$1000          ; use index register X as a pointer to the buffer
            jsr    openAD0         ; initialize the ATD0 converter
            ldy    #5
loop5       movb   #$87,ATD0CTL5   ; start an A/D conversion sequence
            brclr  ATD0STAT,SCF,*
            movw   ADR00H,2,x+     ; collect and save the conversion results
            movw   ADR01H,2,x+     ; post-increment the pointer by 2
            movw   ADR02H,2,x+     ;        "
            movw   ADR03H,2,x+     ;        "
            dbne   y,loop5         ; run the loop 5 times to get 20 samples
            swi
            end
```

TC1047A Temperature Sensor

- A temperature sensor is a solid-state device that changes its output voltage whenever temperature changes.
- Has 3 pins with voltage output directly proportional to the ambient temperature.
- Can measure temperature in the range of -40°C to 125°C (i.e., span of 165 C) with a supply of 2.7~5.5V.
- Voltage output is typically 100mV at -40°C, 500mV at 0°C, 750mV at 25°C, and 1.75V at 125°C.

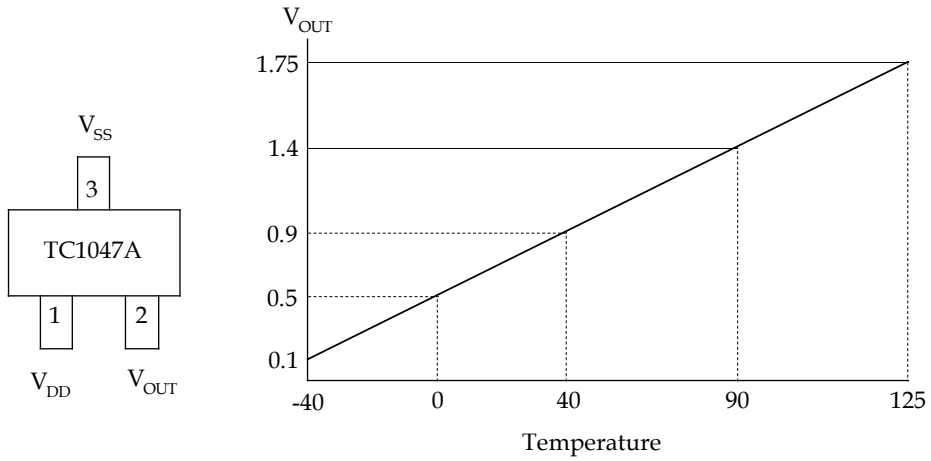

Figure 12.17 TC1047A V_{OUT} vs. temperature characteristic

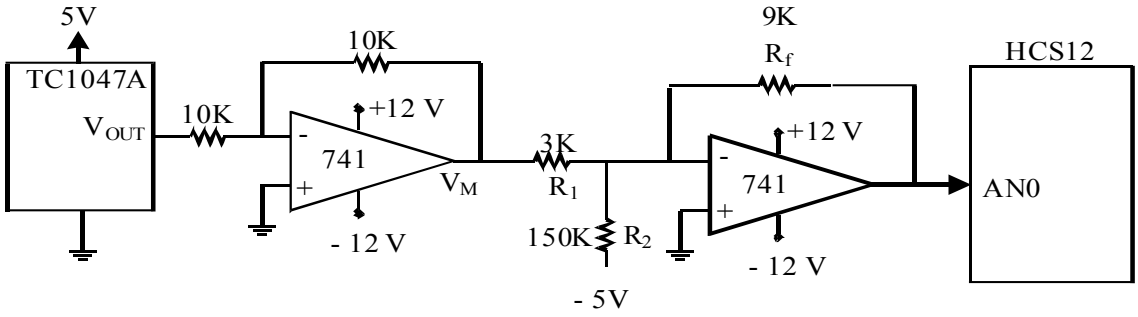

Figure 12.18 Circuit connection between the TC1047A and the HCS12

Example: Displaying the sampled value at ATD0 channel 5 (PAD5), which is connected to a temperature sensor in Dragon12-plus2, on the LEDs of port B. (Use 1 MHz for conversion frequency.)

#include “reg9s12.h”

```
R1      equ     $1001
R2      equ     $1002
R3      equ     $1003

        org     $2000
        lds     #$2000
        movb    #$FF,DDRB       ; set PTB as output for LEDs
        movb    #$FF,DDRJ       ; set PTJ as output (in Dragon12+)
        movb    #$00, PTJ       ; PTJ1=0 to allow LEDs

        movb    #$80,ATD0CTL2   ; initialize the ADC
        jsr     delay
        movb    #$08,ATD0CTL3   ; 8-bit resolution, 16-clock for 2nd stage
        movb    #$EB,ATD0CTL4   ; conversion frequency = 1MHz

H1      movb    #$05,ATD0CTL5   ; channel 5 of ATD0 (left-justified)
                                ; start an A/D conversion sequence
H2      brclr   ATD0STAT,$80,H2 ;
        ldaa    ADR00H          ; get the ADC result
        staa    PortB           ; display the ADC results on LEDs
        jsr     delay           ;

        bra     H1              ; keep reading the ADC

delay   psha
        ldaa    #1
        staa    R3
L3      ldaa    #1              ; 1 ms delay = (1/24MHz)x1x240x10
        staa    R2
L2      ldaa    #240
        staa    R1
L1      nop                     ; 1 E cycle
        nop
        nop
        dec     R1              ; 4 E cycles
        bne     L1              ; 3 E cycles
        dec     R2              ; total = 10 E cycles
        bne     L2
        dec     R3
        bne     L3
        pula
        rts
        end
```

Example: Displaying the sampled value at ATD0 channel 4 (PAD4), which is connected to a light sensor in Dragon12-plus2, on the LEDs of port B. (Use 1 MHz for conversion frequency)

```
#include "reg9s12.h"

R1      equ     $1001
R2      equ     $1002
R3      equ     $1003

        org     $2000
        lds     #$2000
        movb    #$FF,DDRB       ; set PTB as output for LEDs
        movb    #$FF,DDRJ       ; set PTJ as output (in Dragon12+)
        movb    #$00,PTJ        ; PTJ1=0 to allow LEDs

        movb    #$80,ATD0CTL2   ; initialize the ADC
        jsr     delay
        movb    #$08,ATD0CTL3   ; 8-bit resolution, 16-clock for 2nd stage
        movb    #$EB,ATD0CTL4   ; conversion frequency = 1MHz

H1      movb    #$04,ATD0CTL5   ; channel 4 of ATD0 (left-justified)
                                ; start an A/D conversion sequence
H2      brclr   ATD0STAT,$80,H2 ;
        ldaa    ADR00H          ; get the ADC result
        staa    PortB           ; display the ADC result on LEDs
        jsr     delay           ;

        bra     H1              ; keep reading the ADC

delay   psha
        ldaa    #1
        staa    R3
L3      ldaa    #1              ; 1 ms delay = (1/24MHz)x1x240x10
        staa    R2
L2      ldaa    #240
        staa    R1
L1      nop                     ; 1 E cycle
        nop
        nop
        dec     R1              ; 4 E cycles
        bne     L1              ; 3 E cycles
        dec     R2              ; total = 10 E cycles
        bne     L2
        dec     R3
```

```
        bne     L3
        pula
        rts
        end
```

Example: Displaying the sampled value at ATD0 channel 7 (PAD7), which is connected to a potentiometer in Dragon12-plus2, on the LEDs of port B. (Use 1 MHz for conversion frequency)

```
#include "reg9s12.h"

R1      equ     $1001
R2      equ     $1002
R3      equ     $1003

        org     $2000
        lds     #$2000
        movb    #$FF,DDRB       ; set PTB as output for LEDs
        movb    #$FF,DDRJ       ; set PTJ as output (in Dragon12+)
        movb     #$00,PTJ       ; PTJ1=0 to allow LEDs

        movb    #$80,ATD0CTL2   ;
        jsr     delay
        movb    #$08,ATD0CTL3   ; 8-bit resolution, 16-clock for 2nd stage
        movb    #$EB,ATD0CTL4   ; conversion frequency = 1MHz

H1      movb    #$07,ATD0CTL5   ; channel 7 of ATD0 (left-justified)

H2      brclr   ATD0STAT,$80,H2
        ldaa    ADR00H          ; get the ADC result
        staa    PortB
        jsr     delay
        bra     H1              ; keep reading the ADC

delay   psha
        ldaa    #1
        staa    R3
L3      ldaa    #1              ; 1 ms delay = (1/24MHz)x1x240x10
        staa    R2
L2      ldaa    #240
        staa    R1
L1      nop                     ; 1 E cycle
        nop
        nop
```

```
        dec     R1              ; 4 E cycles
        bne     L1              ; 3 E cycles
        dec     R2              ; total = 10 E cycles
        bne     L2
        dec     R3
        bne     L3
        pula
        rts
        end
```

Example: Convert the temperature and display it in three integral and one fractional digits using the LCD. Display the temperature in the whole range of the TC1047A. Update the display five times a second. (E clock is 24 MHz.)

- The whole temperature span is 165°C (from the range: -40°C to 125°C).
- To translate from the A/D conversion result back to temperature, divide the result by 6.2, which can be done by multiplying the result by 10 and then divide the product by 62.

```
#include "reg9s12.h"

period      equ     $2E             ; ASCII code of period character
degree      equ     223             ; ASCII code of degree character
            org     $1000
quo         ds.b    1
rem         ds.b    1
sign        ds.b    1
fract       ds.b    1
buf         ds.b    8               ; to hold string to be output to LCD

            org     $2000
            lds     #$2000          ; set up stack pointer
            ldy     #2              ; wait for LCD to complete
            jsr     delayby100ms    ; internal configuration
            jsr     openLCD         ; configure LCD
            ldaa    #$80            ; set LCD cursor to upper
            jsr     cmd2LCD         ; left corner
            ldx     #msg1           ; output "Temperature = "
            jsr     putsLCD         ;         "
            jsr     openAD0         ; configure ATD0 module

forever     movb    #$20,buf            ; initialize the buffer contents to 0.0C
            movb    #$20,buf+1          ;         "
```

```
            movb  #$30,buf+2        ;       "
            movb  #period,buf+3     ;       "
            movb  #$30,buf+4        ;       "
            movb  #degree,buf+5     ; degree character
            movb  #$43,buf+6        ; letter 'C'
            movb  #0,buf+7          ; null character
            movb  #$87,ATD0CTL5     ; start an ATD conversion sequence
            movb  #0,sign           ; initialize sign to positive
            movb  #$30,fract        ; initialize fractional digit to 0
            brclr ATD0STAT,$80,*    ; wait for the conversion to complete
            ldd   ADR00H            ; read a conversion result
            ldy   #10               ; compute result x 10 / 62
            emul                    ;       "
            ldx   #62               ;       "
            ediv                    ;       "
            stab  rem               ; save the remainder
            tfr   Y,D               ; transfer quotient to B
            subb  #40               ; subtract temperature offset
            bhs   save_quo          ; if non-negative, don't touch remainder
            negb                    ; compute 2's complement of quotient
            stab  quo
            movb  #1,sign           ; temperature is negative
            ldab  rem               ; if remainder is 0, skip a few instruction
            beq   convert           ;
            ldab  #62               ; compute 62 - rem
            subb  rem               ;       "
            stab  rem               ;       "
            bra   cal_fract
save_quo    stab  quo               ; save updated quotient
cal_fract   ldab  rem
            beq   convert           ; come here when positive
            ldaa  #10               ; compute fractional digit
            mul                     ;       "
            ldx   #62               ;       "
            idiv                    ;       "
            cmpb  #31               ; round off fractional digit
            blt   no_round          ;       "
            inx                     ;       "
            cpx   #10               ;       "
            bne   no_round          ;       "
            inc   quo               ;       "
            bra   convert           ; prepare to separate integer digits
no_round    tfr   X,D               ; convert fractional digit to ASCII code
            addb  #$30              ;       "
            stab  fract             ;       "
convert     ldab  quo
```

```
                clra                    ;       "
                ldx     #10             ; use repeated divide by 10 to separate
                idiv                    ; integral digits
                addb    #$30            ;       "
                stab    buf+2           ; save the one's digit
                tfr     X,D             ; transfer quotient to D
                tstb                    ; is quo zero?
                beq     add_fra         ; if integral part is 0, then add fraction digit
                ldx     #10             ; separate the ten's digit
                idiv
                addb    #$30            ; convert and store the ten's digit
                stab    buf+1           ;       "
                tfr     X,D             ; test hundred's digit
                tstb                    ; is quotient 0?
                beq     add_fra
                movb    #$31,buf        ; hundreds digit, if any, is 1 only
add_fra         movb    fract,buf+4     ; insert fraction digit
                ldaa    sign            ; check the sign
                beq     out_it
                movb    #$2D,buf        ; when minus, add minus character
out_it          ldaa    #$C0                    ; set cursor to 2nd row
                jsr     cmd2LCD         ;       "
                ldx     #spaces         ; clear the 2nd row of the LCD
                jsr     putsLCD         ;       "
                ldaa    #$C5            ; set LCD cursor position
                jsr     cmd2LCD         ;       "
                ldx     #buf            ; output the temperature string
                jsr     putsLCD         ;       "
                ldy     #2              ; wait for 200 ms
                jsr     delayby100ms    ;       "
                jmp     forever         ; continue

openAD0         movb    #$E0,ATD0CTL2   ; enable AD0, fast flag clear, wait mode
                ldy     #2
                jsr     delayby10us     ; wait until AD0 is stabilized
                movb    #$0A,ATD0CTL3   ; perform one A/D conversion
                movb    #$25,ATD0CTL4   ; 4 cycles sample time, set prescaler to 12
                rts

delayby100ms:
                bset    TIOS,#$01       ; enable OC0
                ldd     TCNT
again           addd    #37500          ; start an output compare operation
                std     TC0             ; with 50 ms time delay
wait            brclr   TFLG1,$01,wait
                ldd     TC0
```

```
                dbne    y,again
                bclr    TIOS,#$01           ; disable OC0
                rts

delayby10ms:
                bset    TIOS,#$01           ; enable OC0
                ldd     TCNT
again1          addd    #3750               ; start an output compare operation
                std     TC0                 ; with 50 ms time delay
wait1           brclr   TFLG1,$01,wait1
                ldd     TC0
                dbne    Y,again1
                bclr    TIOS,#$01           ; disable OC0
                rts

delayby1ms:
                bset    TIOS,#$01           ; enable OC0
                ldd     TCNT
again2          addd    #375                ; start an output compare operation
                std     TC0                 ; with 50 ms time delay
wait2           brclr   TFLG1,$01,wait2
                ldd     TC0
                dbne    Y,again2
                bclr    TIOS,#$01           ; disable OC0
                rts

delayby10us:
                bset    TIOS,#$01           ; enable OC0
                ldd     TCNT
again3          addd    #3                  ; start an output compare operation
                std     TC0                 ; with 50 ms time delay
wait3           brclr   TFLG1,$01,wait3
                ldd     TC0
                dbne    Y,again3
                bclr    TIOS,#$01           ; disable OC0
                rts

delayby50us:
                bset    TIOS,#$01           ; enable OC0
                ldd     TCNT
again4          addd    #15                 ; start an output compare operation
                std     TC0                 ; with 50 ms time delay
wait4           brclr   TFLG1,$01,wait4
                ldd     TC0
                dbne    y,again4
                bclr    TIOS,#$01           ; disable OC0
```

```
                rts

#include “lcdutildragon12.asm”

msg1            dc.b    "Temperature = “,0
spaces          dc.b    "               “,0
                end
```

End of Chapter 12