- **The MC9S12 A/D Converter**
    - Introduction to A/D Converters
    - Single Channel vs Multiple Channels
    - Singe Conversion vs Multiple Conversions
    - MC9S12 A/C Registers
    - Using the MC9S12 A/D Converter
    - A C program to use the MC9S12 A/D Converter

Analog to Digital Converters (A/D)

• An Analog-to-Digital (A/D) converter converts an analog voltage into a digital number

• There are a wide variety of methods used for A/D converters Examples are:

– Flash (Parallel)
– Successive Approximation
– Sigma-Delta
– Dual Slope Converter

• A/D converters are classified according to several characteristics

– Resolution (number of bits) — **typically 8 bits to 24 bits**
– Speed (number of samples per second) — several samples/sec to several billion samples/sec
– Accuracy — how much error there is in the conversion.

• High-resolution converters are usually slower than low-resolution converters

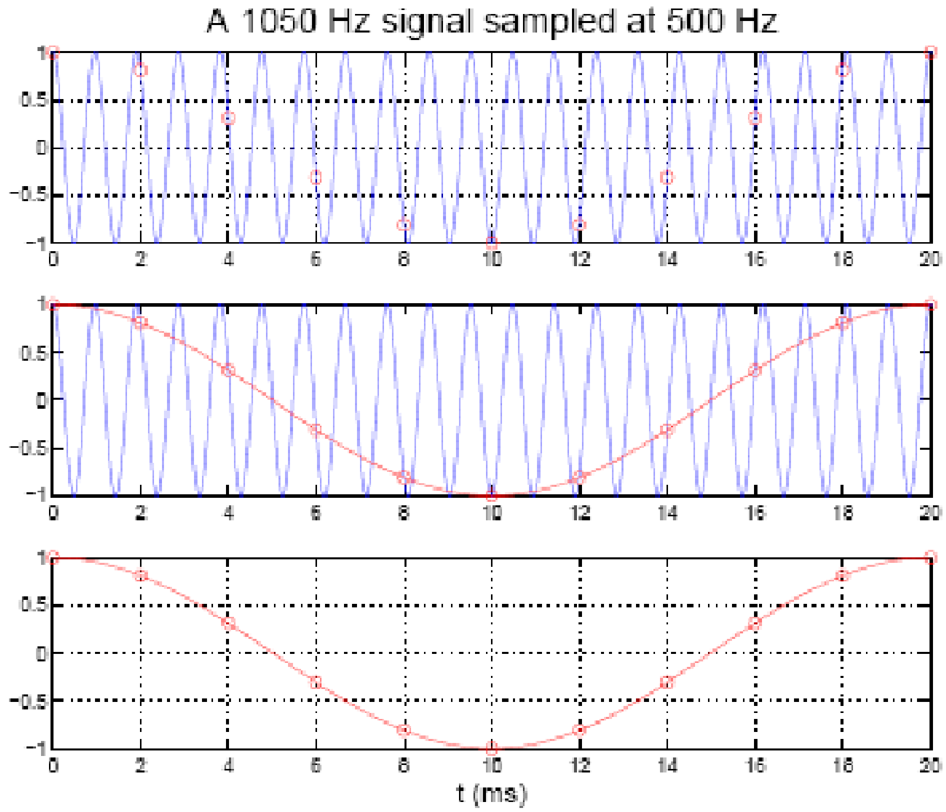
• The MC9S12 has two 10-bit successive approximation A/D converters (which can be used in 8-bit mode)

• The MC9S12 uses an analog multiplexer to allow eight input pins to connect to any of the A/D converters.

A/D Sampling Rate

• The rate at which you sample a signal depends on how rapidly the signal is changing

• If you sample a signal too slowly, the information about the signal may be inaccurate.



• A 1,050 Hz signal sampled at 500 Hz looks like a 50 Hz signal

<u>• To get full information about a signal you must sample more than twice the highest frequency in the signal. (Nyquist theorem).</u>

• Practical systems typically use a sampling rate of at <u>least four times the highest frequency in the signal.</u>

Comparator

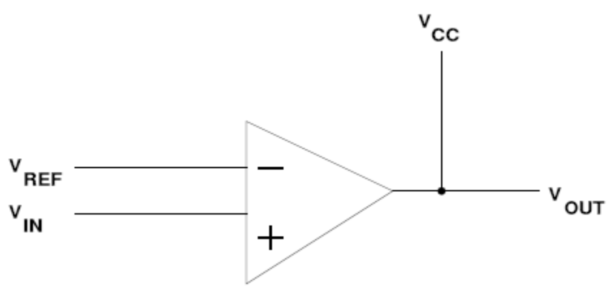
• A comparator is used in many types of A/D converters.

• A comparator is the simplest interface from an analog signal to a digital signal

• A comparator compares two voltage values on its two inputs

• If the voltage on the + input is greater than the voltage on the - input, the output will be a logic high

• If the voltage on the + input is less than the voltage on the - input, the output will be a logic low.



If Vin > Vref then Vout = Vcc

If Vin < Vref then Vout = 0

Slope A/D Converter

• A simple A/D converter can be constructed with a counter and a D/A converter

• The counter counts from **0 to 2^b-1**

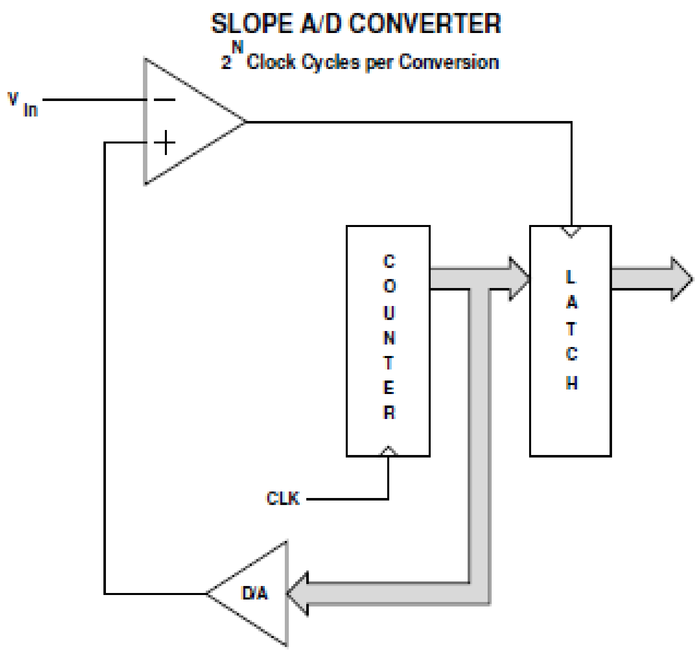
• The counter drives the input of the D/A converter

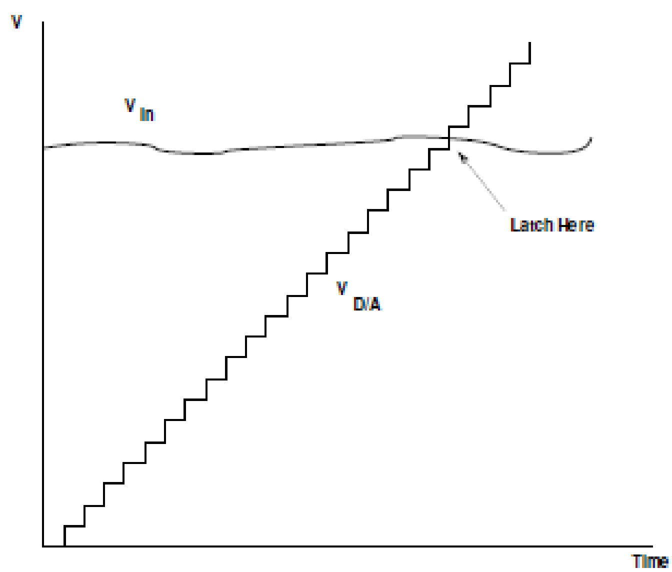
• The output of the D/A converter is compared to the input voltage

• When the output of the comparator switches logic level, the generated voltage passed the input voltage

• By latching the output of the counter at this time, the input voltage can be determined (with the accuracy of the quantization value of the converter)

• **Problem** with Slope A/D converter: <u>Could take 2^b clock cycles to test possible values of reference voltages</u>


SLOPE A/D CONVERTER
2^N Clock Cycles per Conversion
V In
C O U N T E R
L A T C H
CLK
D/A


V
V In
Latch Here
V D/A
Time

Successive Approximation A/D Converter

• A successive approximation (SA) A/D converter **uses an intelligent scheme** to determine the input voltage

• It first tries a voltage half way between V_{RH} and V_{RL}

• It determines if the signal is in the **lower half** or the **upper half** of the voltage range

– If the input is in the upper half of the range, it sets the most significant bit of the output

– If the input is in the lower half of the range, it clears the most significant bit of the output

• The first clock cycle eliminates half of the possible values

• On the next clock cycle, the SA A/D tries a voltage in the middle of the remaining possible values

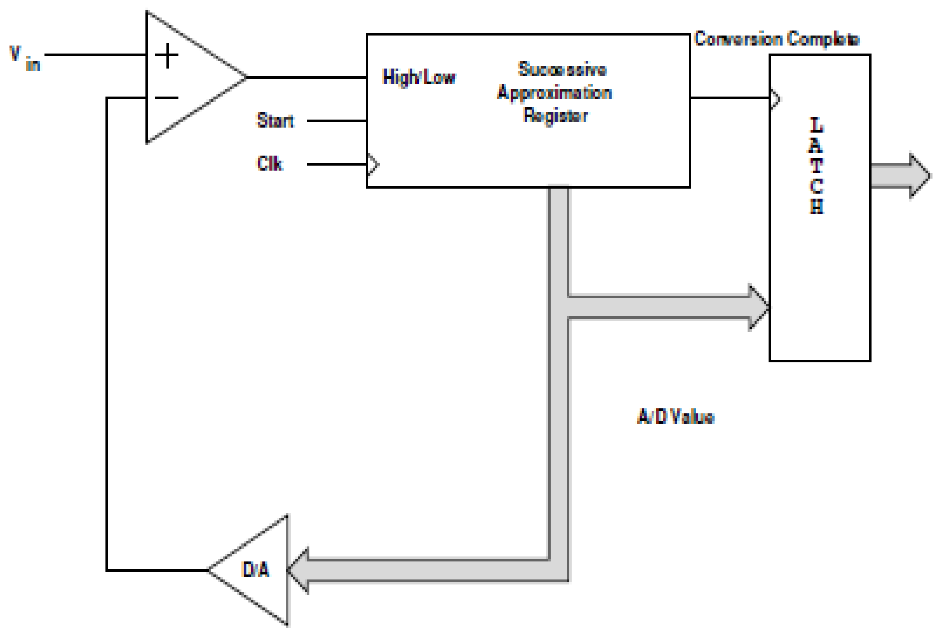
• The second clock cycle allows the SA A/D converter to determine the second most significant bit of the result

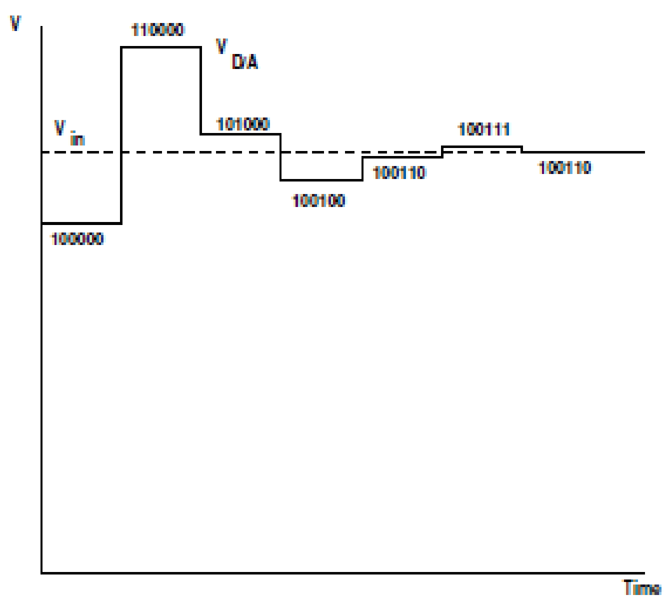
• <u>Each successive clock cycle reduces the range another factor of two</u>

• For a B-bit SA A/D converter, it takes B clock cycles to determine the value of the input voltage

SUCCESSIVE APPROXIMATION A/D CONVERTER

N Clock Cycles per Conversion





Successive Approximation A/D Converter

• An SA A/D converter can give the wrong output if the voltage changes during a conversion

• An SA A/D converter **needs an input buffer** which holds the input voltage constant during the conversion

• This input buffer is called a **Track/Hold** or **Sample/Hold** circuit

• It usually works by charging a capacitor to the input voltage, then disconnecting the capacitor from the input voltage during conversion

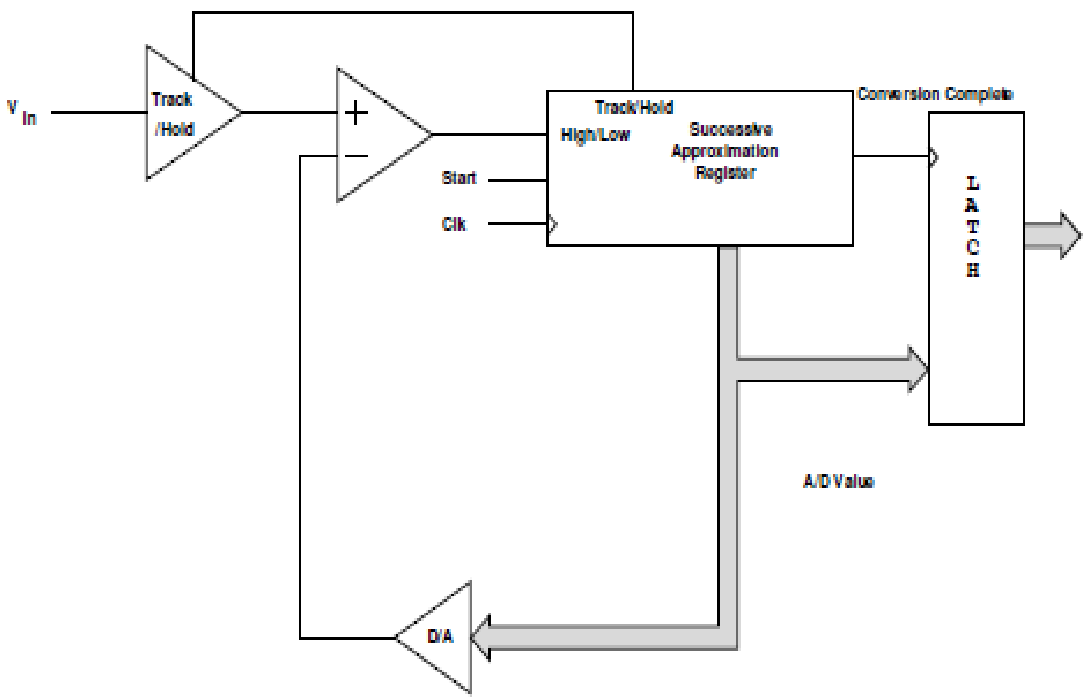
• The voltage on the capacitor remains constant during conversion

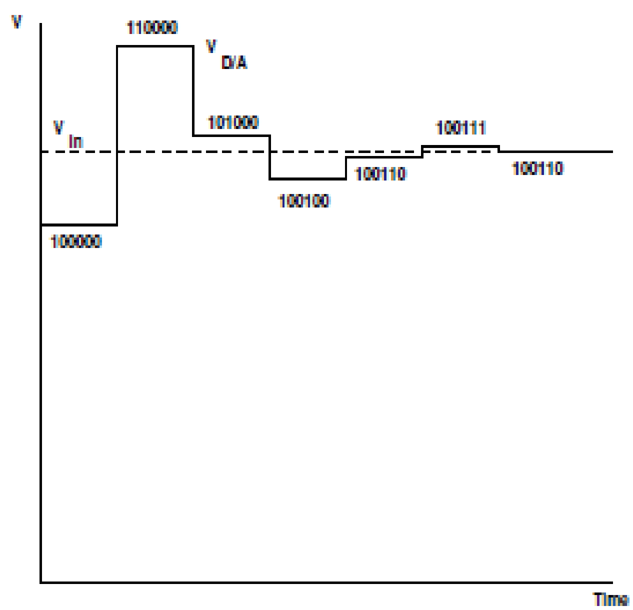
• The HC12 has a Track/Hold amplifier built in

• SA A/D converters have resolutions of up to 16 bits

• SA A/D converters have speeds up to several million samples per second

SUCCESSIVE APPROXIMATION A/D CONVERTER





MC9S12 Analog/Digital Converter

• A 10-bit A/D converter is used to convert an input voltage. The reference voltages are V_{RL} = 0V and V_{RH} = 5V.

– What is the quantization level of the A/D converter?

$\Delta V = (V_{RH} - V_{RL})/(2^b - 1) = 4.88$ mV

• If the value read from the A/D converter is 0x15A, what is the input voltage?

$Vin = V_{RL} + [(V_{RH} - V_{RL})/(2^b-1)]*ADvalue = 0$ V + 4.88 mV × 346 = 1.6894 V

• The MC9S12 has two 10-bit A/D converters (ATD0 and ATD1).

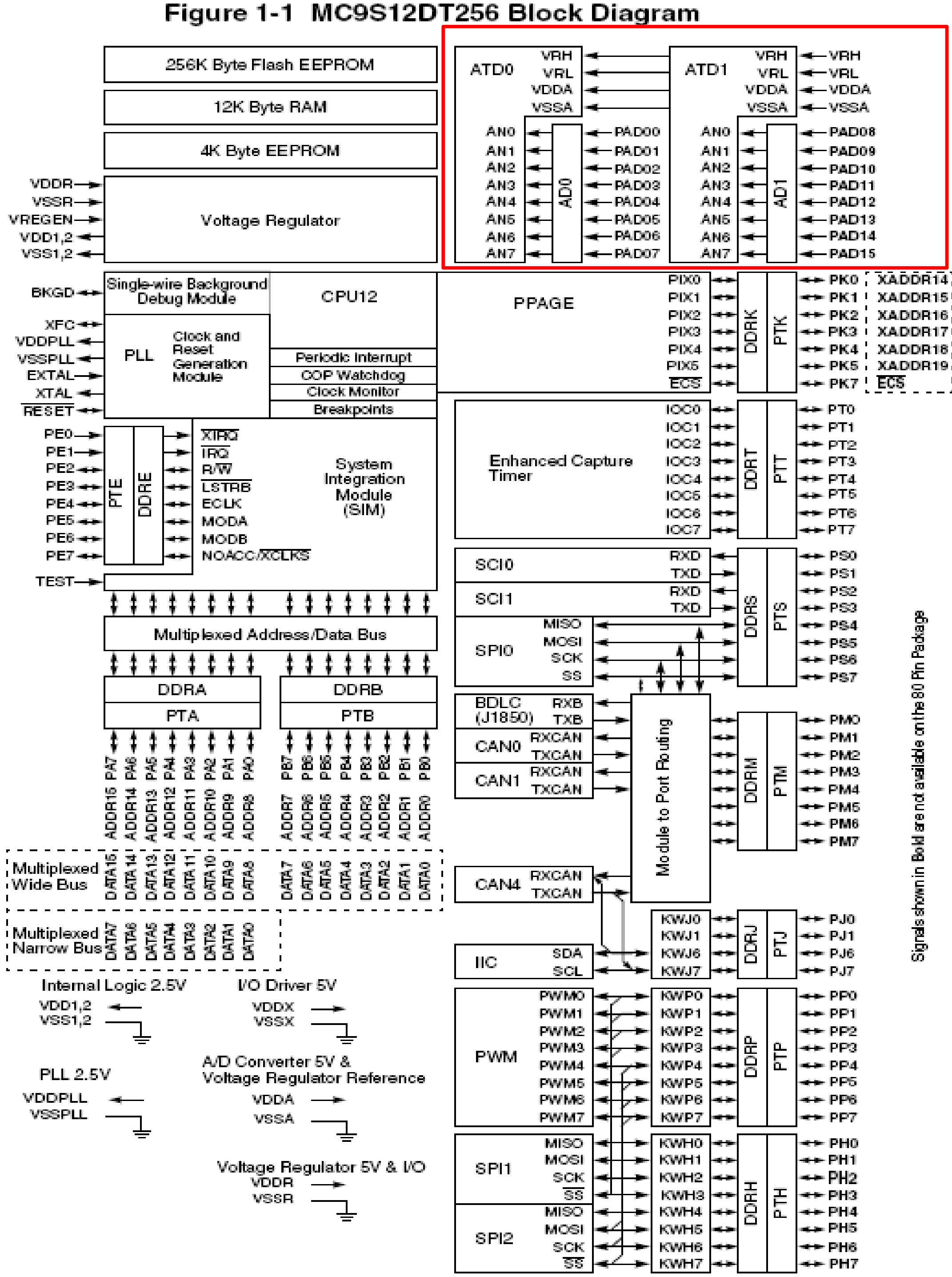
-Each A/D converter has an 8-channel analog mulitplexer in front of it, so each channel can convert 8 analog inputs (but not at exactly the same time).

• ATD0 uses the eight bits of Port AD0, called PAD00 through PAD07

- PAD00 and PAD01 of ATD0 are used by DBug-12 at startup to determine whether to execute DBug-12, or to run code from EEPROM of the bootloader.

• ATD1 uses the eight bits of Port AD1, called PAD08 through PAD15

Figure 1-1 MC9S12DT256 Block Diagram



The MC9S12 Analog/Digital Converter

• We will discuss only ATD0. ATD1 is identical.

• ATD0 is an eight-channel 10-bit A/D converter.

– The A/D converter can also be used in 8-bit mode.

• There are eight inputs to the A/D converter.

• The inputs are fed through a multiplexer to the single A/D converter.

• There are inputs on the HCS12 for the reference voltages V_{RL} and V_{RH}

– In normal operation $V_{RL} = 0$ V and $V_{RH} = 5$ V.

– You must have $V_{SS} \le V_{RL} < V_{RH} \le V_{DD}$.

– The accuracy of the A/D converter is guaranteed only for $V_{RH} - V_{RL} = 5$ V.

• When using the A/D converter, you can choose between performing single or continuous conversion on a single channel, or multiple channels.

• The AD conversion results are stored in the registers ATD0DR0 through ATD0DR7

– You can choose whether to have the results left-justified or right justified.

• To program the HCS12 A/D converter you need to set up the A/D control registers **ATD0CTL2, ATD0CTL3, ATD0CTL4 and ATD0CTL5.**

• The registers ATD0CTL0 and ATD0CTL1 are used for factory test, and not used in normal operation.

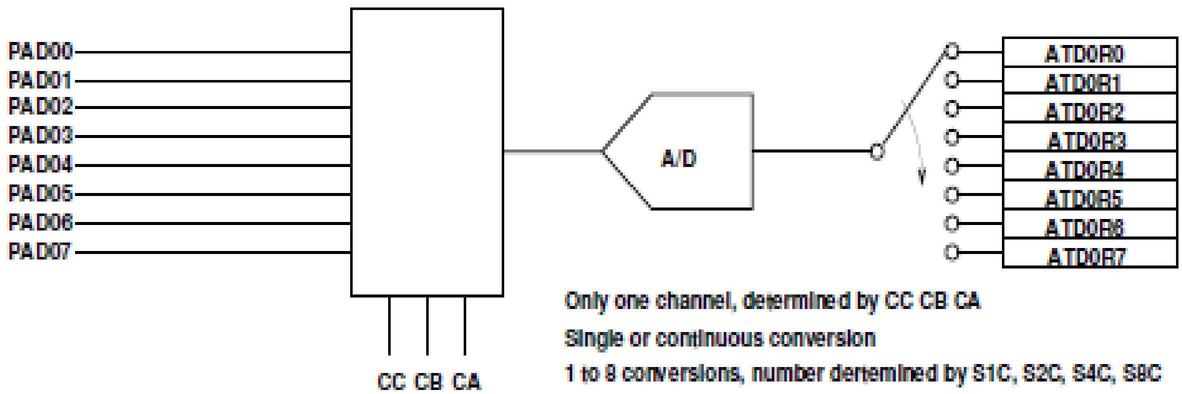
• When the AD converter is not used, Port AD0 can be used for a general purpose input (not as a GPIO – General Purpose I/O like the other ports).

- Register ATD0DIEN is used to set up Port AD0 pins for use as a general purpose inputs.

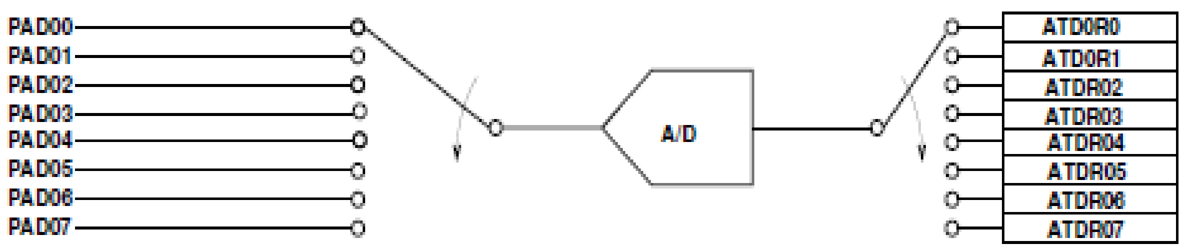
- The values on the pins are read from PORTAD0.

MC9S12 A/D Converter Setup

MULT = 0



MULT = 1



Several Channels.
Starting channel determined by CC CB CA
1 to 8 conversions, number determined by S1C, S2C, S4C, S8C

ATD0CTL2	ADPU	AFFC	ASWAI	ETRIGLE	ETRIGLP	0	ASCIE	ASCIF	0x0082
ATD0CTL3	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0	0x0083
ATD0CTL4	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0	0x0084
ATD0CTL5	DJM	DSGN	SCAN	MULT	0	CC	CB	CA	0x0085
ATD0STAT0	SCF	0	ETORF	FIFOR	0	CC2	CC1	CC0	0x0086
ATD0STAT1	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0	0x008B

To Use A/D Converter:

ADPU = 1 (Power up A/D)

SCAN = 0 => Single conversion sequence
SCAN = 1 => Convert continuously

S8C, S4C, S2C, S1C:
Number of conversions per sequence: 0001 −− 0111 (1 to 7)
0000 or 1xxx (8)
SRES8 = 0 = > 10 Bit Mode
SRES8 = 1 => 8 Bit Mode

DJM = 0 => Left justified data in the result registers
DJM = 1 => Right justified data in the result registers

DSGN = 0 => Unsigned data in the result registers

DSGN = 1 => Signed data representation in the result registers (only for left justified)

ATDCTL4= 0x85 => 2 MHz AD clock, 12 cycles per conversion, 8 bit mode
ATDCTL4= 0x05 => 2 MHz AD clock, 14 cycles per conversion, 10 bit mode

-Other values of ATDCTL4 will not work, or will result in slower operation of A/D

SCF Flag is set after a sequence of conversions is complete
-The SCF Flag is cleared when ATD0CTL5 is written, or by writing a 1 to the SCF bit

After writing to ATD0CTL5, SCF flag cleared and conversions start

Using the HCS12 A/D converter

1. Power up A/D Converter (ADPU = 1 in ATD0CTL2)

2. Select number of conversions per sequence (S8C S4C S2C S1C in ATD0CTL3)

    S8C S4C S2C S1C = 0001 to 0111 for 1 to 7 conversions
    S8C S4C S2C S1C = 0000 or 1xxx for 8 conversions

3. Set up ATD0CTL4

    • For 8-bit mode write 0x85 to ATD0CTL4
    • For 10-bit mode write 0x05 to ATD0CTL4
    • Other values of ATD0CTL4 either will not work or will result in slower A/D conversion rates

4. Select DJM in ATD0CTL5

    (a) DJM = 0 => Left justified data in the result registers
    (b) DJM = 1 => Right justified data in the result registers

5. Select DSGN in ATD0CTL5

    (a) DSGN = 0 => Unsigned data representation in the result register
    (b) DSGN = 1 => Signed data representation in the result register

The Available Result Data Formats are shown in the following table:

SRES8	DJM	DSGN	RESULT DATA FORMAT
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

6. Select MULT in ATD0CTL5:

- MULT = 0: Convert one channel the specified number of times
  - Choose channel to convert with CC, CB, CA of ATD0CTL5.

- MULT = 1: Convert across several channels. CC, CB, CA of ATD0CTL is the first channel to be converted.

7. Select SCAN in ATD0CTL5:

- SCAN = 0: Convert one sequence, then stop
- SCAN = 1: Convert continuously

8. After writing to ATD0CTL5, the A/D converter starts, and the SCF bit is cleared. After a sequence of conversions is completed, the SCF flag in ATD0STAT0 is set.

- You can read the results in ATD0DRx.

9. If SCAN = 0, you need to write to ATD0CTL5 to start a new sequence. If SCAN = 1, the conversions continue automatically, and you can read new values in ATD0DRx.

10. To get an interrupt after the sequence of conversions are completed, set ASCIE bit of ATD0CTL2. After the sequence of conversions, the ASCIF bit in ATD0CTL2 will be set, and an interrupt will be generated.

11. With 24 MHz bus clock and ATD0CTL4 = 0x05, it takes 7 µs to make one conversion, 56 µs to make eight conversions.

12. On MC9S12 EVBU, AD0 channels 0 and 1 are used to determine start-up program (D-Bug12, EEPROM or bootloader). Do not use AD0 channels 0 or 1, unless absolutely necessary (if you need more than 14 A/D channels).

13.

$$\text{ATD0DRx} = (V_{in} - V_{RL})/(V_{RH} - V_{RL}) \times 1024$$

Normally, $V_{RL} = 0$ V, and $V_{RH} = 5$ V, so

$$\text{ATD0DRx} = V_{in}/5\text{ V} \times 1024$$

Example: ATD0DR0 = 448 => $V_{in} = 2.19$ V

14. To use 10-bit result, set ATD0CTL4 = 0x05 (Gives 2 MHz AD clock with 24 MHz bus clock, 10-bit mode).

15. You can get more accuracy by averaging multiple conversions. If you need only one channel, set MULT = 0, set S8C, S4C, S2C, S1C, bits for eight conversions, then average all eight result registers. The following assumes the data was right justified:

```
int avg;

avg = (ATD0DR0 + ATD0DR1
       ATD0DR2 + ATD0DR3
       ATD0DR4 + ATD0DR5
       ATD0DR6 + ATD0DR7) >> 3;
```

```
/* Read temperature from PAD4. Turn on heater if temp too low,
 turn off heater if temp too high. Heater connected to Bit 0 of Port
A. */

#include "hcs12.h"
#define TRUE 1
#define SET_POINT 72  /* Temp at which to turn heater */
                      /* on or off */
main() {
        ATD0CTL2 = 0x80; /* Power up A/D, no interrupts */
        ATD0CTL3 = 0x00; /* Do eight conversions */
        ATD0CTL4 = 0x85; /* 8-bit mode */
        ATD0CTL5 = 0xA4; /* 1 0 1 0 0 1 0 0
                            | | | |   \___/
                            | | | |     |
                            | | | |     \__ Bit 4 of Port AD
                            | | |  \____ MULT = 0 => one channel only
                            | |  \___ Scan = 1 => continuous conversion
                            | \_____DSGN = 0 => unsigned
                            \______ DJM = 1 => right justified
                         */

        /*******************************************************/

        DDRA = 0xff; /* Make Port A output */
        PORTA = 0x00; /* Turn off heater */

        /*******************************************************/
        while (TRUE) {
                if (ATD0DR0 > SET_POINT)
                        PORTA &= ~BIT0;
                else
                        PORTA |= BIT0;
        }
}
```

```
/* Convert signals on Channels AD08 through AD15.  Set up for
10-bit, multi-channel do one set of scans, save values in variables
*/

#include "hcs12.h"
main()
{
        unsigned int ch[8]; /* Variable to hold result */
        ATD1CTL2 = 0x80; /* Power up A/D, no interrupts */
        ATD1CTL3 = 0x40; /* Do eight conversions */
        ATD1CTL4 = 0x05; /* 10-bit mode, 7 us/conversion */
        ATD1CTL5 = 0x92; /* 1 0 0 1 0 0 1 0
                            | | | |   \__/
                            | | | |     |
                            | | | |     \_ First channel = 2
                            | | |  \___ MULT = 1 => multiple channels
                            | |  \__ SCAN = 0 => one set of conversions
                            |  \_____DSGN = 0 => unsigned
                            \_______DJM = 1 => right justified    */

/***********************************************************/

        while ((ATD1STAT0 & BIT7) == 0 ) ; // Wait for sequence to finish

        ch[0] = ATD1DR0;
        ch[1] = ATD1DR1;
        ch[2] = ATD1DR2;
        ch[3] = ATD1DR3;
        ch[4] = ATD1DR4;
        ch[5] = ATD1DR5;
        ch[6] = ATD1DR6;
        ch[7] = ATD1DR7;
}
```