EE4390 Microprocessors

Lessons 16 - 18

68HC12 Analog-to-Digital (ATD) Converter System

Overview

- Analog-to-digital fundamental concepts
- 68HC12 ATD system description
- ATD registers
  - control registers
  - status registers
  - result registers
  - test registers
- ATD programming

Analog-to-digital fundamental concepts

- Transducer interface design
- Big picture
- conversion process
  - sampling rate
  - encoding
  - quantizing and resolution
  - data rate
- successive approximation converter

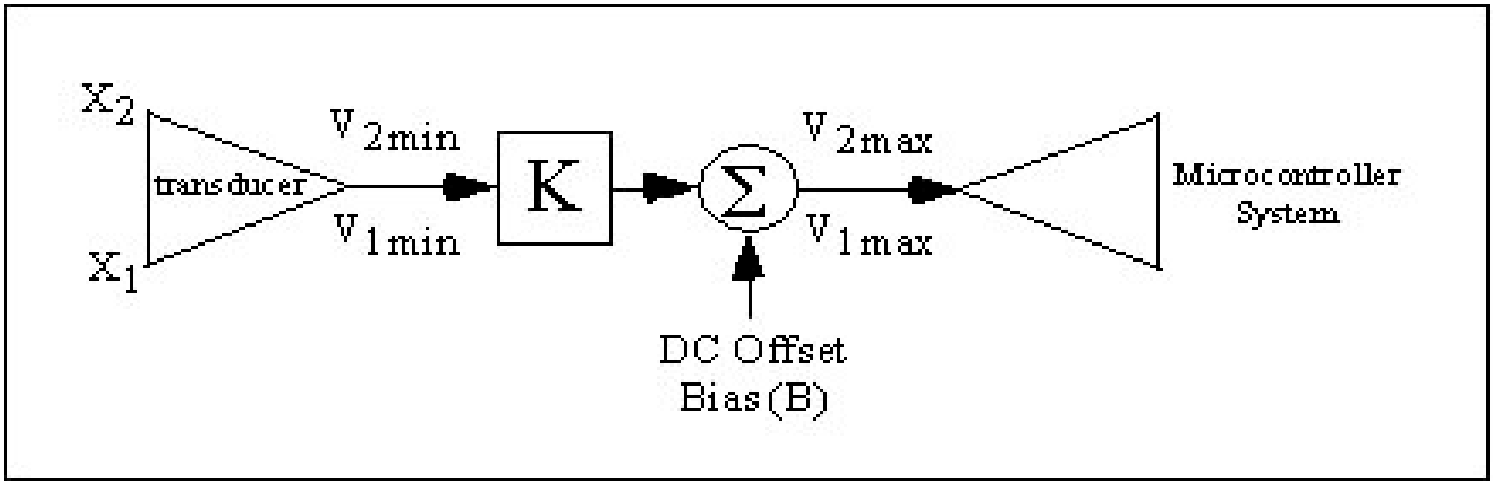
Analog-to-digital fundamental concepts Transducer Interface Design

- We live in an analog world!
- Physical variable requires conversion to digital representation
  - i.e. light, pressure, temperature -> voltage -> binary
- Requires transducer for conversion
- Transducer requires interface to microprocessor such that it fills conversion window (V_{RH} - V_{RL})

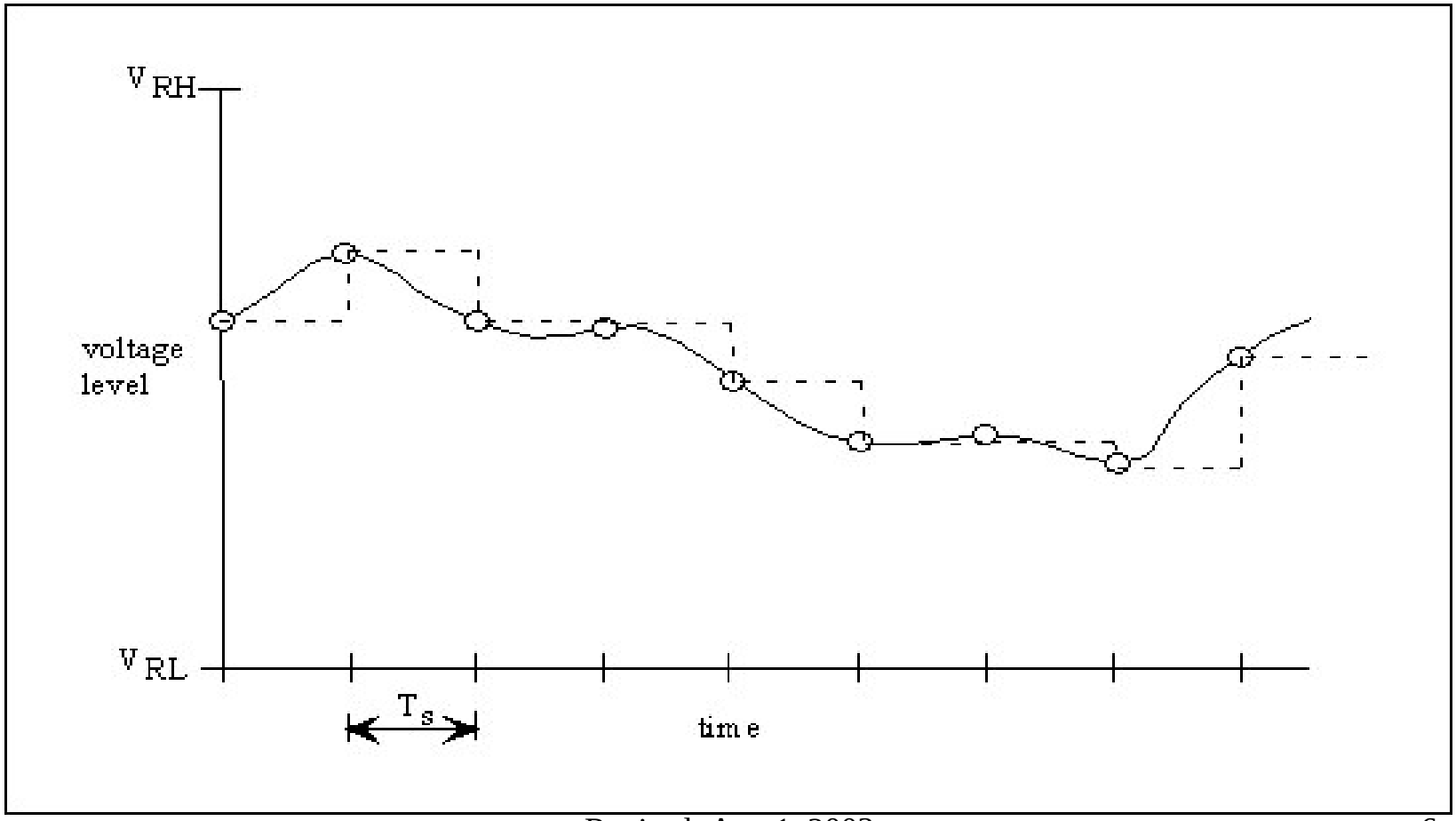
Analog-to-digital fundamental concepts
Transducer Interface Design (cont)

$$V_{2max} = V_{2min} * K + B$$

$$V_{1max} = V_{1min} * K + B$$



Analog-to-digital fundamental concepts
Big Picture



Analog-to-digital fundamental concepts conversion process

- conversion process
  - sampling rate
  - encoding
  - quantizing and resolution
  - data rate

Analog-to-digital fundamental concepts conversion process - sampling rate

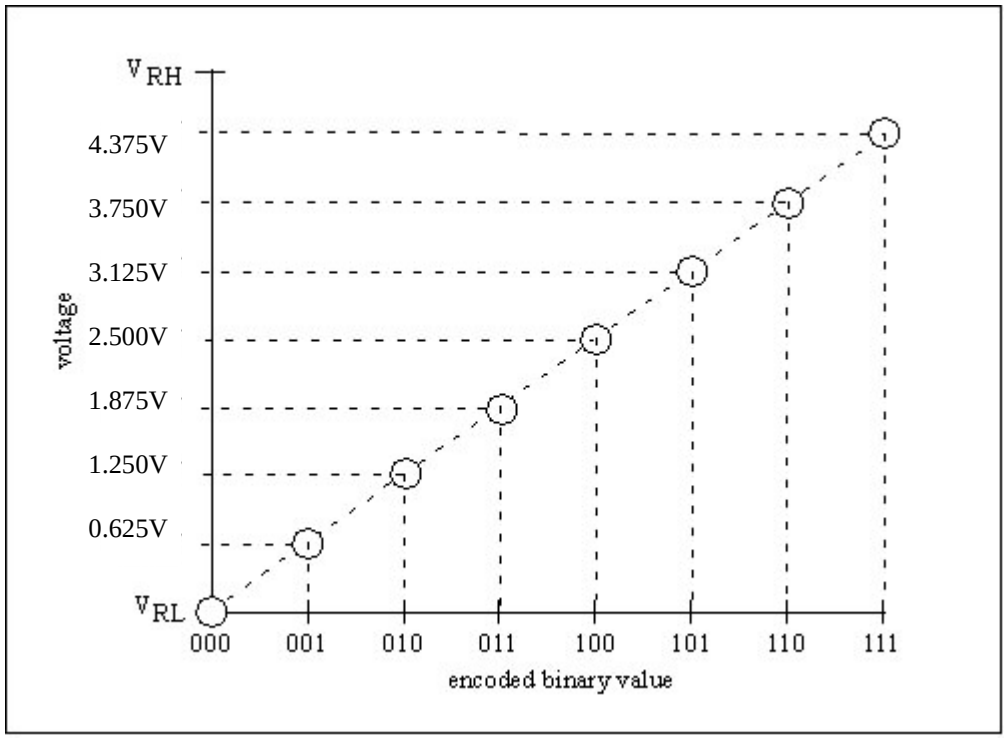
- strobe light example
- Nyquist criterion: sample signal at a minimum frequency of twice the highest frequency content of the sampled signal

  $f_s >= 2\ f_h$
- Time interval between samples

  $T_s = 1/\ f_s$
- anti-aliasing filter: use LPF with $f_{cutoff} = f_h$
  - phone company samples human voice at 8 KHz, uses 4 KHz LPF to prevent aliasing

Analog-to-digital fundamental concepts conversion process - encoding

- Provides unique binary code for every discrete voltage step between V_{RH} and V_{RL} $n = 2^b$



Analog-to-digital fundamental concepts
conversion process - quantizing, resolution, data rate

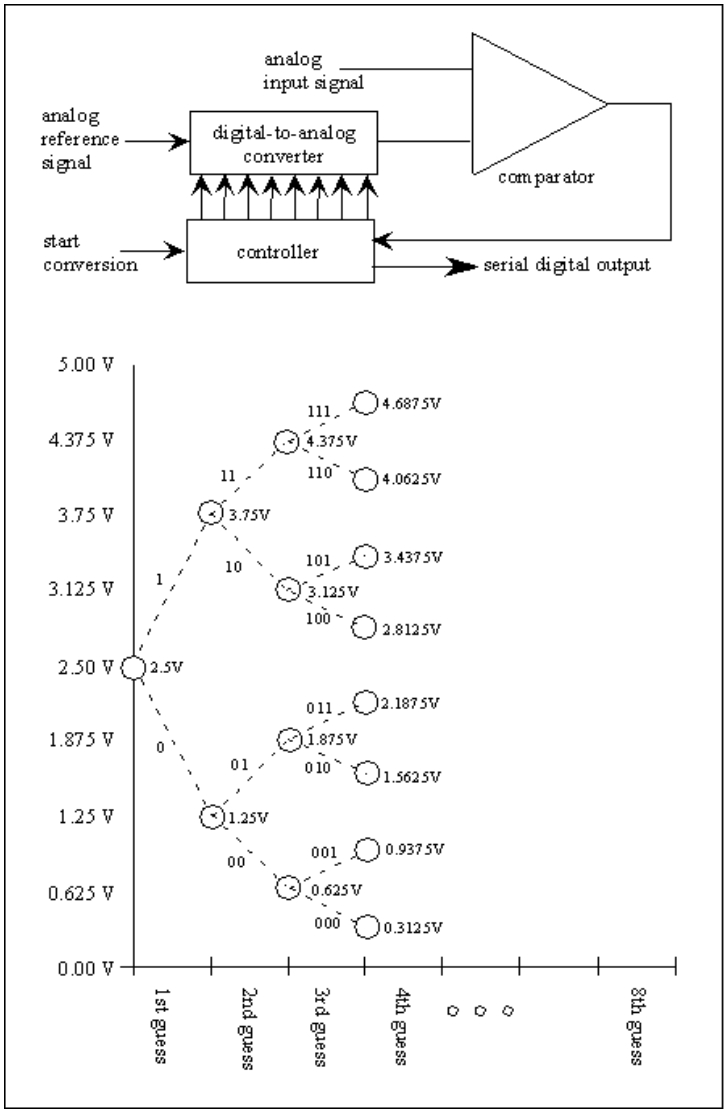
- Quantization: number of discrete levels the analog signal is divided into between V_{RH} and V_{RL}
  - More levels provide better representation of sampled signal

  EX] V_{RH} = 5 VDC and V_{RL} = 0 VDC, quantization 256 levels
  voltage per step = (5V - 0V)/(256 steps) = 19.53 mV/step

- Resolution: voltage per step

  Resolution = $(V_{RH} - V_{RL})$/number of steps = $(V_{RH} - V_{RL})/2^b$
- Data rate: $d = f_s b$

Analog-to-digital fundamental concepts
conversion process - successive approximation



68HC12 ATD system description

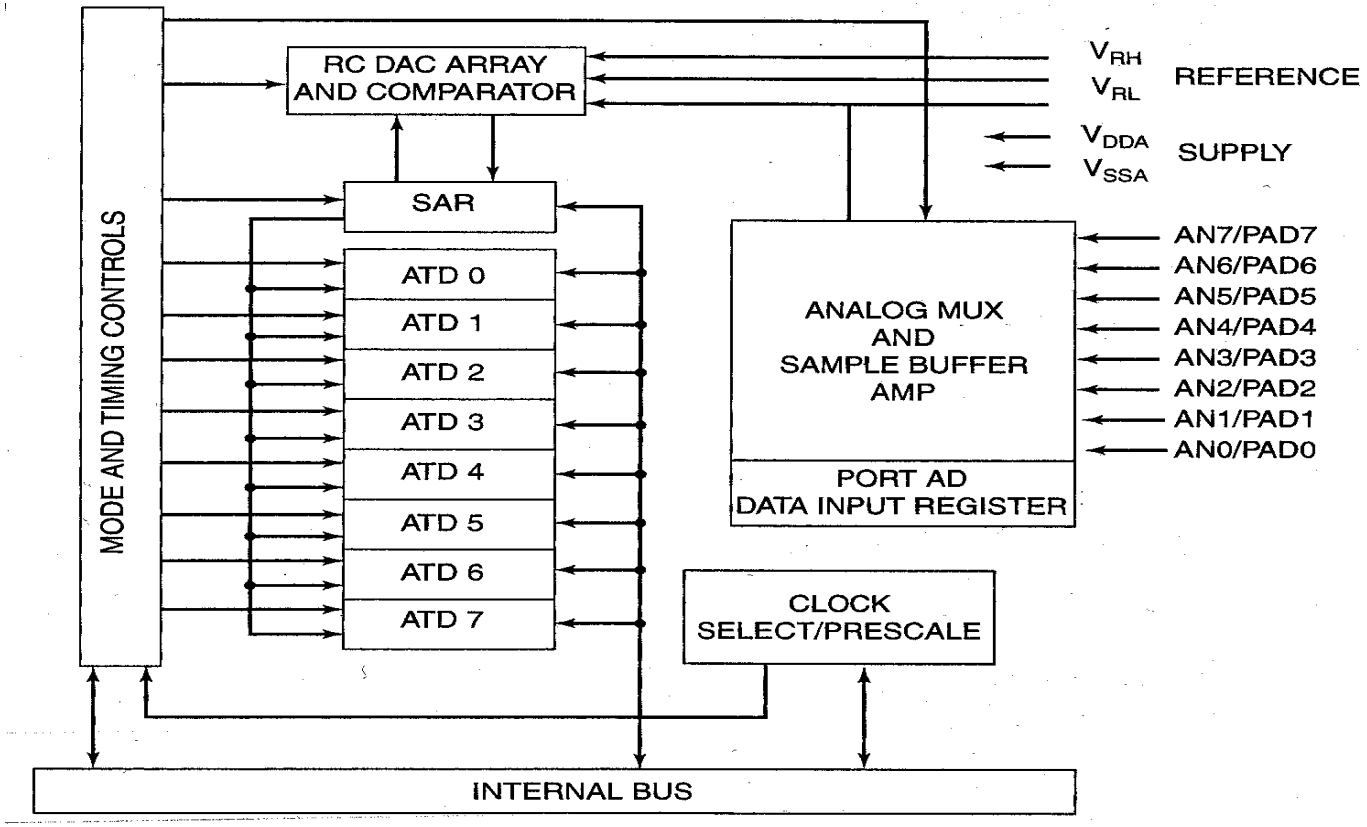


Figure 9.10 Analog-to-digital (ATD) block diagram. The eight ATD system analog inputs are located on pins AN0/PAD0 to AN7/PAD7. (Figure used with permission of Motorola, Incorporated.)

68HC12 ATD system description (cont)

- Eight ATD **analog** inputs on PORTAD PAD[7:0]
- Inputs fed to analog multiplexer
- Single signal fed to successive approximation converter
- Initiate conversion by writing to control register
- Upon conversion complete appropriate flags set in status registers
- Results available in results register

ATD registers

- control registers - configures ATD for specific operation (ATDCTL0 - ATDCTL5)
- status registers - two-byte register containing ATD status flags (ATDSTAT)
- result registers - contains binary weighted result after conversion (ADR0H - ADR7H)
- test registers - used in special modes

ATD registers
control registers

- Used to tailor an ATD conversion sequence
- We will concentrate on control registers ATDCTL 2, 4, and 5

ATD registers
control registers - ATDCTL2

- Memory address: $0062
  - ADPU: “on/off” switch
    - 0: off, 1: on (0 after processor reset)
  - must wait 100 us after “on” prior to using ATD
  - AFFC: ATD Fast Flag Clear
    - 0: normal clearing - write to ATDCTL5
    - 1: fast clearing - cleared when first result register read

Register: Analog-to-Digital Converter Control Register 2 (ATDCTL2) Address: $0062

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

ATD registers
control registers - ATDCTL4

- Memory address: $0064
- Controls sample timing for conversion sequence

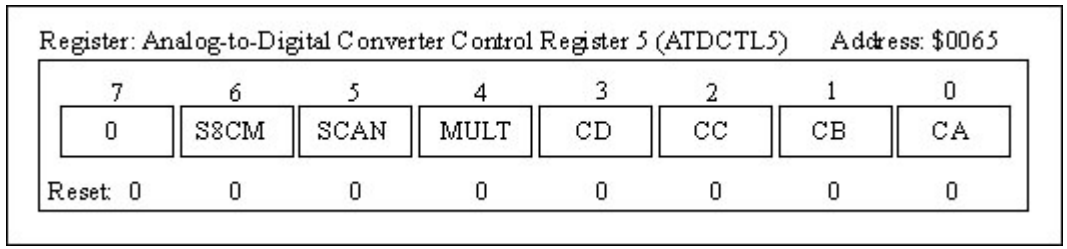
SMP1 SMP2	Final Sample Time	Total 8-bit Conversion Time
0 0	2 ATD clock periods	18 ATD clock periods
0 1	4 ATD clock periods	20 ATD clock periods
1 0	8 ATD clock periods	24 ATD clock periods
1 1	16 ATD clock periods	32 ATD clock periods

Register: Analog-to-Digital Converter Control Register 4 (ATDCTL4) Address: $0064

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

ATD registers
control registers - ATDCTL5

- Used to configure conversion mode for ATD
- Memory location: $0065
  - S8CM: select 8 channel mode 0: four, 1: eight conversions
  - SCAN: enable continuous scan 0: single 1: continuous conversion
  - MULT: enable multiple channel conversion 0: single channels, 1: multiple channels
  - CD,CC,CB,CA: specify channels for conversion



ATD registers control registers - ATDCTL5 (cont)

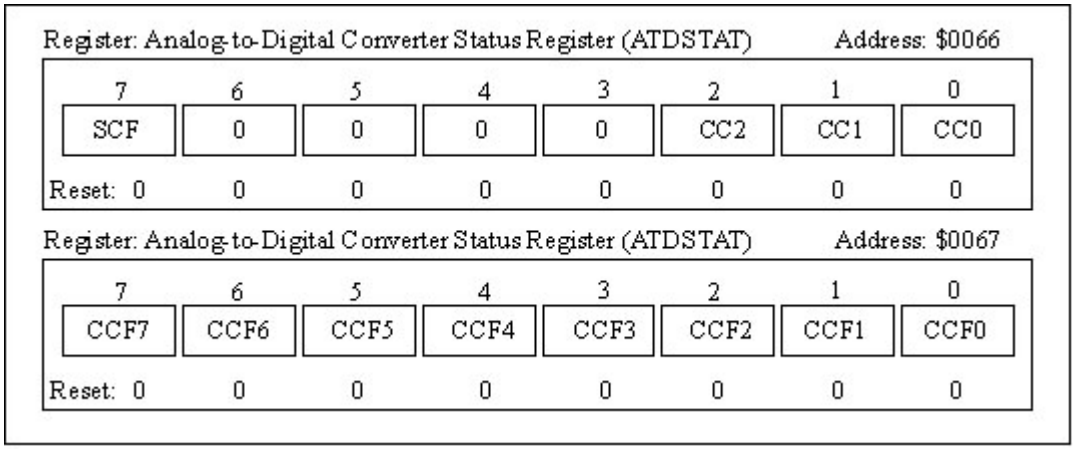
Table 17-4. Multichannel Mode Result Register Assignment

S8CM	CD	CC	CB	CA	Channel Signal	Result in ADRx if MULT = 1
0	0	0	0	0	AN0	ADR0
			0	1	AN1	ADR1
			1	0	AN2	ADR2
			1	1	AN3	ADR3
0	0	1	0	0	AN4	ADR0
			0	1	AN5	ADR1
			1	0	AN6	ADR2
			1	1	AN7	ADR3
0	1	0	0	0	Reserved	ADR0
			0	1	Reserved	ADR1
			1	0	Reserved	ADR2
			1	1	Reserved	ADR3
0	1	1	0	0	V_{RH}	ADR0
			0	1	V_{RL}	ADR1
			1	0	$(V_{RH} + V_{RL})/2$	ADR2
			1	1	Test/reserved	ADR3
1	0	0	0	0	AN0	ADR0
			0	1	AN1	ADR1
			1	0	AN2	ADR2
			1	1	AN3	ADR3
		1	0	0	AN4	ADR4
			0	1	AN5	ADR5
			1	0	AN6	ADR6
			1	1	AN7	ADR7
1	1	0	0	0	Reserved	ADR0
			0	1	Reserved	ADR1
			1	0	Reserved	ADR2
			1	1	Reserved	ADR3
		1	0	0	V_{RH}	ADR4
			0	1	V_{RL}	ADR5
			1	0	$(V_{RH} + V_{RL})/2$	ADR6
			1	1	Test/reserved	ADR7

Shaded bits are "don't care" if MULT = 1 and the entire block of four or eight channels makes up a conversion sequence. When MULT = 0, all four bits (CD, CC, CB, and CA) must be specified and a conversion sequence consists of four or eight consecutive conversions of the single specified channel.

ATD registers
status registers

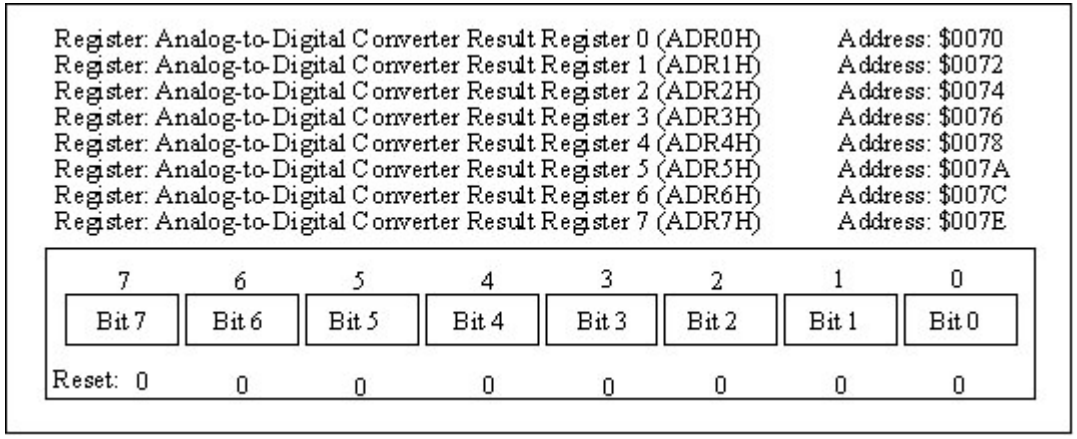
- two-byte register containing ATD status flags (ATDSTAT)
  - contains series of flags that indicate status of the ATD
  - Sequence Complete Flag (SCF) indicates specified conversion is complete
  - CCx: 3 bit counter which indicates channel currently undergoing conversion
  - CCFx: Conversion Complete Flag for each result register



ATD registers
result registers

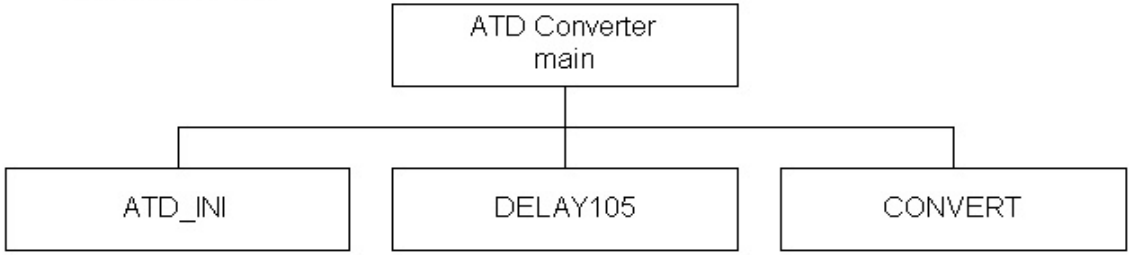
- After conversion results placed in ADR0H-7H
- Unsigned, weighted binary result

  1/2FS 1/4FS 1/8FS….
- VDC = (contents ADRxH/256) * (V_{RH} - V_{RL})

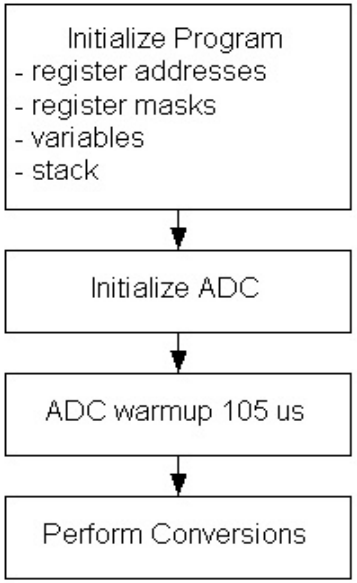


ATD Programming

Structure chart:



Flow chart:



ATD Programming

```
/* File Name: voltmeter.c
 * File Created: 04-14-02
 * File Modified:
 * Author(s): Abbie Wells
 * Carrie Hernandez, LCD Portions
 */

/* This program will create a simple voltmeter using the onboard
 * analog to digital converter in the HC12.  It will perform one
 * conversion and then the user will have to manually restart the
 * program to convert another voltage.  The voltage will then be
 * displayed to the LCD.
 */

 #include <hc12.h>
 #include <stdio.h>

 #define DECIMAL 0x2E  // define macro for a decimal point in ASCII
 #define V 0x56        // define macro for a "V" in ASCII

 void delay_100us(void);
 void ADC_convert(void);
void delay_5ms(void);
```

ATD Programming

```
void main(void)
 {
 printf("HELLO\n");
 ATDCTL2 = 0x80;           // power up the ADC and disable interrupts
 printf("ADC2\n");
 delay_5ms();              // wait for ADC to warm up
 printf("warmed up\n");
 ATDCTL3 = 0x00;           // select active background mode
 ATDCTL4 = 0x01;           // select sample time = 2 ADC clks and set
                           // prescaler to 4 (2 MHz)
 printf("ready\n");
 ADC_convert();            // perform conversion and change to usable
                           // value
 }
```

ATD Programming

```
/* void ADC_convert(void): function to perform a single conversion */

void ADC_convert()
{
unsigned int sumadr;
unsigned int avg_bin_voltage;
unsigned int int_voltage;
unsigned int ones_int;
unsigned int tenths_int;
unsigned int hundreths_int;
char ones;
char tenths;
char hundreths;


ATDCTL5 = 0x03;         // sets up ADC to perform a single conversion,
                        // 4 conversions on a single channel,
                        // and store the results ADR0H - ADR3H.


while((ATDSTAT & 0x8000) != 0x8000)  // Wait for conversion to finish
{
  ;
}

//printf("%x %x %x %x\n", ADR0H, ADR1H, ADR2H, ADR3H);
sumadr = ADR0H + ADR1H + ADR2H + ADR3H;
avg_bin_voltage = sumadr/4;
int_voltage = (100*avg_bin_voltage/256)*5;
ones_int = int_voltage/100;
ones = (char)(ones_int + 48);
tenths_int = (int_voltage - ones_int*100)/10;
tenths = (char)(tenths_int + 48);
hundreths_int = (int_voltage - ones_int*100 - tenths_int*10)/1;
hundreths = (char)(hundreths_int + 48);
printf("%c.%c%cV\n", ones, tenths, hundreths);
}
```

ATD Programming

```
/*************************************************************/
/*100us delay based on an 8MHz clock                         */
/*************************************************************/

void delay_100us(void)
{
int i;

for (i=0; i<400; i++)
  {
  asm("nop");
  }
}

/*************************************************************/
/*5 ms delay based on an 8MHz clock                          */
/*************************************************************/

void delay_5ms(void)
{
int i;

for (i=0; i<800; i++)
  {
  delay_100us();
  }
}
```