EXAMPLE – A/D CONVERTER

OBJECTIVE
This example will help you learn how to use the M68HC11 microcontroller to collect analog data in digital form. The following objectives are considered:

- Review the use of A/D conversion function of the M68HC11 microcontroller.
- Illustrate various modes of performing A/D conversion.

This example covers the following topics:

- Single channel, single conversion, Mult=0, Scan=0;
- Single channel, continuous conversion, Mult=0, Scan=1;
- Multiple channel, single conversion, Mult=1, Scan=0;
- Multiple channel, continuous conversion, Mult=1, Scan=1;

PROGRAM A/D CONVERTER

SINGLE CHANNEL, SINGLE CONVERSION, MULT=0, SCAN=0
In this section, we will study the single-channel single-conversion. The channel selected in this example will be the pin PE1. The program will do the following

a) Initialize the A/D conversion

b) Set ADCTL to reflect the following control bits:
   i) CD=0, CC=0, CB=0, CA=1 to select pin PE1
   ii) SCAN=0, i.e., no scanning
   iii) MULT=0, i.e., single channel

c) Check if the A/D conversion has finished. The conversion is finished when the flag CCF is set.

d) Load the results from the AD registers ADR1 – ADR4 and store them into memory locations VAL1 – VAL4.

e) Loop back to b)

FLOWCHART AND CODE
The program flowchart is show below. Two flowchart levels are presented: the big-picture and the details. The big-picture is used to understand the overall architecture of the program. The details are used to explain some of the blocks. (Details are given only for those blocks which are somehow new, and have not been used in previous programs.) The essential code for this program is shown to the right of the flowchart. The essential code was incorporated into the standard template asm to generate the code file Ex_AD_1.asm.
EXECUTION
Open THRSim11. Close the Commands window. View CPU registers, AD converter registers, port E pins, memory list, Sliders E port. Open and assemble Ex_AD_1.asm. Reset registers and memory. Set standard labels (Label/Set Standard Labels). Set display of ADCTL and accA to binary. Arrange windows for maximum benefit: Press the RESET button. Your screen should look like this:
Step through the program up to LABEL1. The AD conversion control register ADCTL has been set to perform a single A/D conversion on the pin PE1. Your screen looks like this:

![Image of the screen showing the program line by line]

We are now ready to perform the conversions. Set a breakpoint just after exiting the check CCF loop, i.e., at line $c010. Make the value of PE1 = 1000 mV. Run. The program performs the AD conversion and exists the loop. The registers ADR1 – ADR4 have the value $33, which is the hex value of $1000 \times \$ff/5000$ (check with your hex calculator!). The screen looks like this:

![Image of the screen showing the program line by line]

Step through the program until all the four AD registers get transferred into the memory storage locations VAL1 – VAL4. The screen looks like this:

Remove the breakpoint from $c010. Enter 1500 mV into PE1. Run. The program will run continuously. The converted value $4c (1500*$ff/5000=$4c, check with your hex calculator!) appears in the ADR1 – ADR4 registers and is transferred into VAL1 – VAL4. The screen looks like this:
Put value 2000 mV into PE1. The converted value $66 (2000*ff/5000=$66, check with your hex calculator) appears in ADR1 – ADR4 and then in VAL1 – VAL4. The screen looks like this:

Put value 2500 mV into PE1. The converted value $80 appears in ADR1 – ADR4 and then in VAL1 – VAL4 (2500*ff/5000=$7f with normal 1 bit convention, check with your hex calculator. But M68HC11 uses ½ bit accuracy convention, and hence rounds up to $80). The screen looks like this:

Put value 3500 mV into PE1. The converted value $b3 appears in ADR1 – ADR4 and then in VAL1 – VAL4 (3500*ff/5000=$b2 with normal 1 bit convention, check with your hex calculator. But M68HC11 uses ½ bit accuracy convention, and hence rounds up to $b3). The screen looks like this:
Put value 5000 mV into PE1. The converted value $ff appears in ADR1 – ADR4 and then in VAL1 – VAL4 (5000*$ff/5000=$ff, check with your hex calculator!). The screen looks like this:

At the end of this process, you have verified the conversion of six voltage values in the range 0 – 5 V, and have obtained the results indicated in Table 1.
Table 1

<table>
<thead>
<tr>
<th>Voltage at pin PE1 (mV)</th>
<th>AD converted values (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000 $33</td>
</tr>
<tr>
<td>2</td>
<td>1500 $4c</td>
</tr>
<tr>
<td>3</td>
<td>2000 $66</td>
</tr>
<tr>
<td>4</td>
<td>2500 $80</td>
</tr>
<tr>
<td>5</td>
<td>3500 $b3</td>
</tr>
<tr>
<td>6</td>
<td>5000 $ff</td>
</tr>
</tbody>
</table>

Let the program run, and move freely the slider bar on PE1. Observe the reaction time of the MCU simulator, and how the converted values corresponding to the new simulated voltages appear first in the ADR1 – ADR4 registers, and then into the VAL1 – VAL4 memory locations.

**Single channel, continuous conversion, Mult=0, Scan=1**

You will now perform continuous AD conversion in SCAN mode. This mode does not require you to check the CCF flag. The program flowchart is showed below. Also two flowcharts are presented. The essential code for this program is shown to the right of the flowchart. The essential code was incorporated into the standard template asm to generate the code file Ex_AD_2.asm.
EXECUTION

Open and assemble Ex_AD_2.asm. Reset registers and memory. Set standard labels (Label/Set Standard Labels). Run the program. Notice that after addressing ADCTL once at the beginning, the program now loops only for storing the values of ADR1 – ADR4 into memory locations VAL1 – VAL4. Enter 1000 mV in PE1. The value is converted to $33 and stored in memory. The screen looks like this:

Continue with the other values in Table 1, until you are satisfied with the functioning of the program.
**MULTIPLE CHANNEL, SINGLE CONVERSION, MULT=1, SCAN=0**

We will now perform AD conversion on multiple channels, single conversion. This mode is selected by setting MULT=1 and SCAN=0 such that the conversion is performed once.

The program flowchart is shown below. Two flowchart levels are presented: the big-picture and the details. The essential code for this program is shown to the right of the flowchart. The essential code was incorporated into the standard template asm to generate the code file Ex_AD_3.asm.

![Flowchart Diagram]

**EXECUTION**

Open and assemble Ex_AD_3.asm. Reset registers and memory. Set standard labels (Label/Set Standard Labels). The screen looks like this:
Enter 2500 mV in PE0, 1000 mV in PE1, 3500 mV in PE2, 2000 mV in PE3. Run the program. The converted value $80, $33, $b3, $66 appear in ADR1 – ADR4, and then get stored in memory locations VAL1 – VAL4. The screen looks like this:
**MULTIPLE CHANNEL, SINGLE CONVERSION, MULT=1, SCAN=1**

We will now perform AD conversion on multiple channels continuous conversion. This mode is selected by setting MULT=1 and SCAN=1 such that the conversion is performed continuously.

The program flowchart is shown below. Two flowchart levels are presented: the big-picture and the details. The essential code for this program is shown to the right of the flowchart. The essential code was incorporated into the standard template asm to generate the code file Ex_AD_4.asm.

### EXECUTION

Open and assemble Ex_AD_4.asm. Reset registers and memory. Set standard labels (Label/Set Standard Labels). Run the program. Notice that after addressing ADCTL once at the beginning, the program now loops only for storing the values of ADR1 – ADR4 into memory locations VAL1 – VAL4. Enter 2500 mV in PE0, 1000 mV in PE1, 3500 mV in PE2, 2000 mV in PE3. The converted value $80, $33, $b3, $66 appear in ADR1 – ADR4, and then get stored in memory locations VAL1 – VAL4. The screen looks like this:
Repeat the process with other values in the four PE1 – PE4 pins. Obtain voltage values by moving the sliding controls on these pins up and down and observing the converted values being appearing in the ADR1 – ADR4 registers, and then into the VAL1 – VAL4 memory locations.

WHAT YOU HAVE LEARNED

In this exercise, you have:

- Review the use of A/D conversion function of the M68HC11 microcontroller.
- Illustrated 3 modes of performing A/D conversion:
  - Single channel single conversion
  - Single channel continuous conversion
  - Multiple channel single conversion
  - Multiple channel continuous conversion