RENESAS

APPLICATION NOTE

M16C/62

Using the M16C/62 Analog to Digital Converter in Single Sweep Mode

1.0 Abstract

The following article outlines the steps necessary to set up, perform, and read a single sweep conversion using the onboard analog to digital converter (ADC) of the M16C. The ADC is useful in measuring output voltages of sensors such as accelerometers or other analog instrumentation and converting them to digital values.

2.0 Introduction

The M16C line of devices features an onboard analog to digital converter (ADC). The ADC consists of one 10-bit successive approximation circuit with a capacitive coupled amplifier. There are eight analog input pins, selectable conversion clock speeds, sample and hold function, and several conversion modes. Figure 1 is an overview of the internal circuitry for the ADC block.

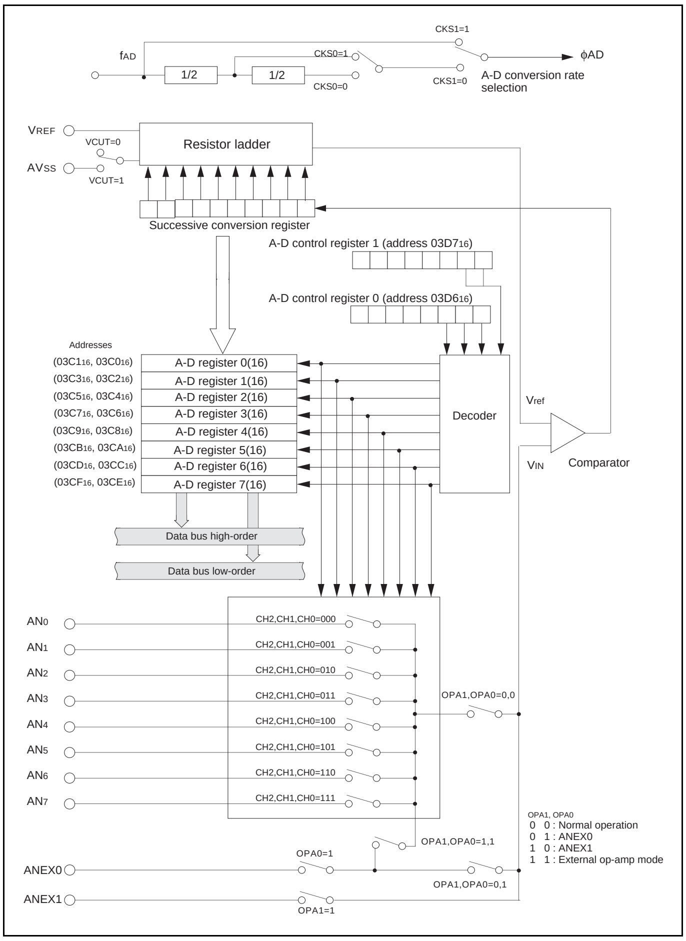
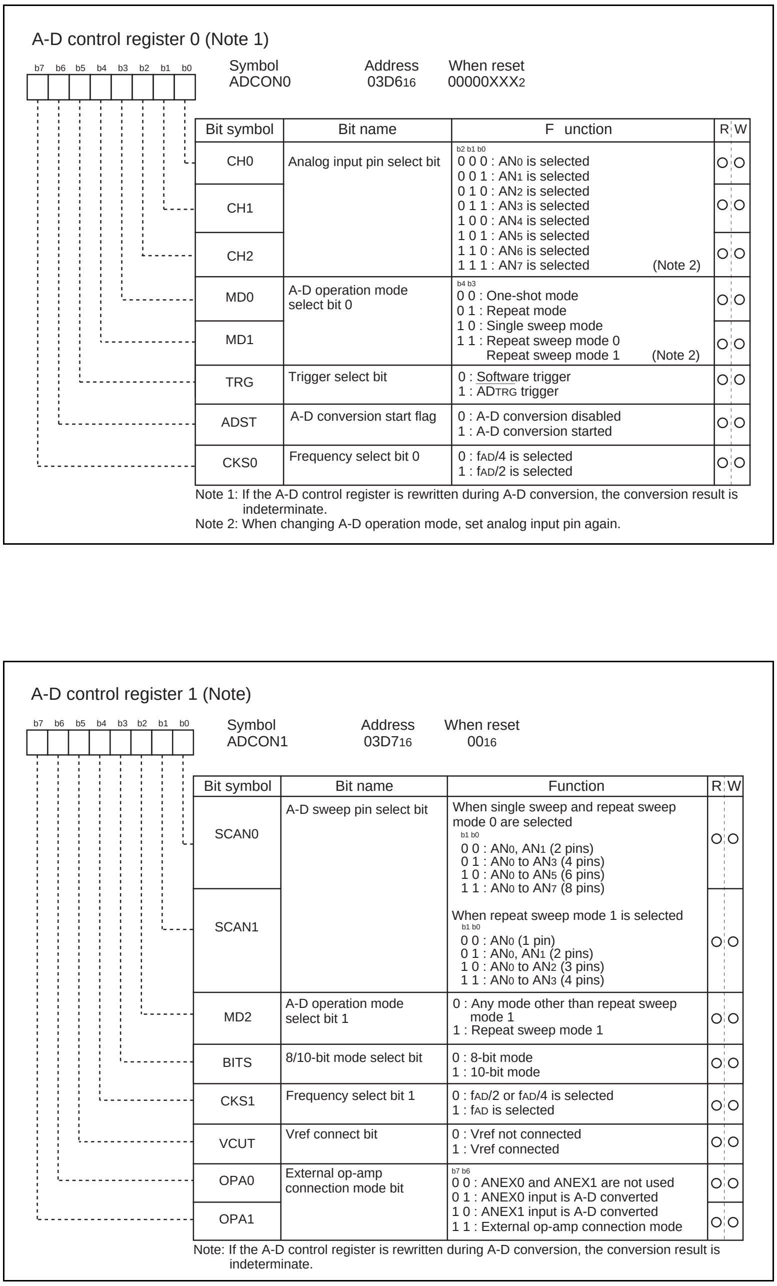


Figure 1 Internal Circuitry for ADC Block—Overview

3.0 Single Sweep Mode Description

In single sweep mode, multiple pins of the ADC can be selected as the input source. Once triggered, a single conversion takes place on each of the selected pins and the result is stored in the ADC result registers corresponding to the selected channels. An interrupt is generated signifying the completion of the conversions. Figure 2 and Figure 3 are overviews of the registers that will be used in this example. These registers are detailed in the included sample code.



A-D control register 0 (Note 1)

Symbol	Address	When reset
ADCON0	$03D6_{16}$	$00000XXX_2$

Bit symbol	Bit name	Function	R	W
CH0	Analog input pin select bit	b2 b1 b0 0 0 0 : AN$_0$ is selected 0 0 1 : AN$_1$ is selected 0 1 0 : AN$_2$ is selected 0 1 1 : AN$_3$ is selected 1 0 0 : AN$_4$ is selected 1 0 1 : AN$_5$ is selected 1 1 0 : AN$_6$ is selected 1 1 1 : AN$_7$ is selected (Note 2)	○	○
CH1			○	○
CH2			○	○
MD0	A-D operation mode select bit 0	b4 b3 0 0 : One-shot mode 0 1 : Repeat mode 1 0 : Single sweep mode 1 1 : Repeat sweep mode 0 Repeat sweep mode 1 (Note 2)	○	○
MD1			○	○
TRG	Trigger select bit	0 : Software trigger 1 : AD$_{TRG}$ trigger	○	○
ADST	A-D conversion start flag	0 : A-D conversion disabled 1 : A-D conversion started	○	○
CKS0	Frequency select bit 0	0 : f$_{AD}$/4 is selected 1 : f$_{AD}$/2 is selected	○	○

Note 1: If the A-D control register is rewritten during A-D conversion, the conversion result is indeterminate.
Note 2: When changing A-D operation mode, set analog input pin again.

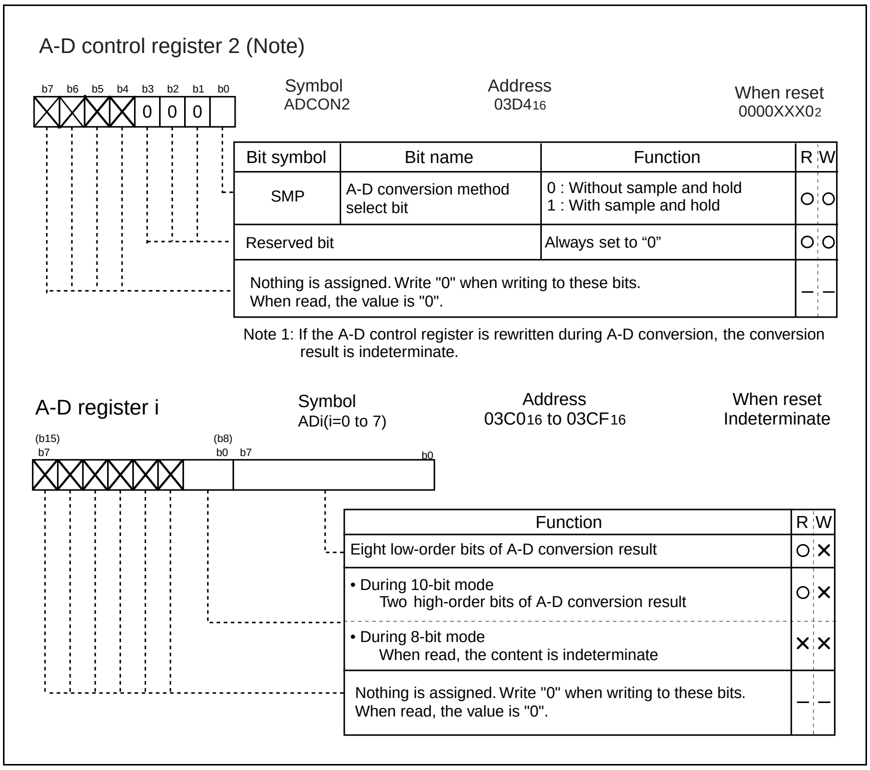
A-D control register 1 (Note)

Symbol	Address	When reset
ADCON1	$03D7_{16}$	00_{16}

Bit symbol	Bit name	Function	R	W
SCAN0	A-D sweep pin select bit	When single sweep and repeat sweep mode 0 are selected b1 b0 0 0 : AN$_0$, AN$_1$ (2 pins) 0 1 : AN$_0$ to AN$_3$ (4 pins) 1 0 : AN$_0$ to AN$_5$ (6 pins) 1 1 : AN$_0$ to AN$_7$ (8 pins)	○	○
SCAN1		When repeat sweep mode 1 is selected b1 b0 0 0 : AN$_0$ (1 pin) 0 1 : AN$_0$, AN$_1$ (2 pins) 1 0 : AN$_0$ to AN$_2$ (3 pins) 1 1 : AN$_0$ to AN$_3$ (4 pins)	○	○
MD2	A-D operation mode select bit 1	0 : Any mode other than repeat sweep mode 1 1 : Repeat sweep mode 1	○	○
BITS	8/10-bit mode select bit	0 : 8-bit mode 1 : 10-bit mode	○	○
CKS1	Frequency select bit 1	0 : f$_{AD}$/2 or f$_{AD}$/4 is selected 1 : f$_{AD}$ is selected	○	○
VCUT	Vref connect bit	0 : Vref not connected 1 : Vref connected	○	○
OPA0	External op-amp connection mode bit	b7 b6 0 0 : ANEX0 and ANEX1 are not used 0 1 : ANEX0 input is A-D converted 1 0 : ANEX1 input is A-D converted 1 1 : External op-amp connection mode	○	○
OPA1			○	○

Note: If the A-D control register is rewritten during A-D conversion, the conversion result is indeterminate.

Figure 2 A-D Converter Related Registers



A-D control register 2 (Note)

Symbol	Address	When reset
ADCON2	03D4$_{16}$	0000XXX0$_2$

Bit symbol	Bit name	Function	R	W
SMP	A-D conversion method select bit	0 : Without sample and hold 1 : With sample and hold	○	○
Reserved bit		Always set to “0”	○	○
Nothing is assigned. Write "0" when writing to these bits. When read, the value is "0".			–	–

Note 1: If the A-D control register is rewritten during A-D conversion, the conversion result is indeterminate.

A-D register i

Symbol	Address	When reset
ADi(i=0 to 7)	03C0$_{16}$ to 03CF$_{16}$	Indeterminate

Function	R	W
Eight low-order bits of A-D conversion result	○	×
• During 10-bit mode Two high-order bits of A-D conversion result	○	×
• During 8-bit mode When read, the content is indeterminate	×	×
Nothing is assigned. Write "0" when writing to these bits. When read, the value is "0".	–	–

Figure 3 A-D Converter Related Register

4.0 Example Program

This example program demonstrates how to perform a conversion using the ADC in the following environment:

Environment Setup

- Single sweep conversion
- 10-bit mode
- Analog inputs 0–3 used
- Sample and hold enabled
- Vref connected
- Conversion clock used will be $f_{AD}/2$
- Software conversion start

ADC Software Setup

- Set the ADCON0 register for single sweep mode 0 operation, $f_{AD}/2$ (0x90)
- Set the ADCON1 register for 10-bit mode, f_{AD} divided, AN0-3 sweep, and connect Vref (0x29)
- Set the ADCON2 register for sample and hold (0x01)
- Enable the A/D converter by setting the ADST bit to 1
- Read current A/D channel values in the variables ‘TempStore(x)’ in the AD Interrupt Service Routine

5.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheets

- M16C/62 datasheets, 62aeds.pdf

User's Manual

- NC30 Ver. 4.0 User's Manual, NC30UE.pdf
- M16C/60 and M16C/20 C Language Programming Manual, 6020EC.pdf
- M16C/62 User's Manual, 62eum.pdf
- Application Note: Writing Interrupt Handlers in C for the M16C

6.0 Software Code

The sample software provided was written using the NC30 compiler. The program performs one set of conversions on reset. This code could be simply modified to use a timer for the trigger of the ADC to provide multiple conversions at specific intervals.

```
/*******************************************************************
 *
 *    DESCRIPTION: single_sweep.c
 *
 *    AUTHOR: Renesas Technology Corporation (June 2003)
 *
 *
 *    PURPOSE:Outlines how to use the M16C/62 ADC in single sweep
 *            mode. On reset, program stores the results of the
 *            conversions in variables that can be examined using
 *            KD30 and the MSV1632-62 Starter Kit
 *
 *******************************************************************/
#include "sfr62.h"

unsigned int TempStore0 = 0x0000;         // Location where AN0 result is stored
unsigned int TempStore1 = 0x0000;         // Location where AN1 result is stored
unsigned int TempStore2 = 0x0000;         // Location where AN2 result is stored
unsigned int TempStore3 = 0x0000;         // Location where AN3 result is stored

#pragma INTERRUPT ADCInt           /* compiler directive indicating
                                    the  proper  return  method  for  this  function
                            (REIT vs. RTS)*/
```

```
void ADCInt(void);

/*
 ** main
 *
 *  PARAMETERS: None
 *
 *  DESCRIPTION: Main function. Where program execution starts. Sets
 *               up the ADC then waits for interrupt to occur.
 *
 *  RETURNS: Nothing
 *
 */

void main (void){

        adcon0 = 0x90;  /*10010000  single sweep mode, software trigger, fAD/2
                          ||||||||______analog input select bit 0
                          |||||||_______analog input select bit 1
                          ||||||________analog input select bit 2
                          |||||_________A/D operation mode select bit 0
                          ||||__________A/D operation mode select bit 1
                          |||___________trigger select bit
                          ||____________A/D conversion start flag
                          |_____________frequency select bit */

        adcon1 = 0x29;  /*   00101001;    /* 10 bit mode, fAD divided, Vref connected,
AN0-3
                             ||||||||______A/D sweep pin select bit 0
                             |||||||_______A/D sweep pin select bit 1
                             ||||||________A/D operation mode select bit 1
                             |||||_________8/10 bit mode select bit
                             ||||__________frequency select bit 1
                             |||___________Vref connect bit
                             ||____________external op-amp connection bit 0
                             |_____________external op-amp connection bit 1 */

        adcon2 = 0x01; /* 00000001;  Sample and hold enabled
                           ||||||||______sample and hold select bit
                           |||||||_______reserved
                           ||||||________reserved
                           |||||_________reserved
                           ||||__________reserved
                           |||___________reserved
                           ||____________reserved
                           |_____________reserved */
```

```
       adic = 0x01;  /*00000001 Set Priority Level to Enable the ADC interrupt
                       ||||||||______interrupt priority select bit 0
                       |||||||_______interrupt priority select bit 1
                       ||||||________interrupt priority select bit 2
                       |||||_________interrupt request bit
                       ||||__________reserved
                       |||___________reserved
                       ||____________reserved
                       |_____________reserved */

    _asm ("   fset i") ;    // globally enable interrupts

    adst = 1;               // Start a conversion here

    while (1){}             // Program waits here forever
}

/*
 ** ADCInt
 *
 *  PARAMETERS: None
 *
 *  DESCRIPTION: Interrupt routine of the ADC. Here the converted value is
 *               loaded into a variable and masked off to show the result.
 *
 *  RETURNS: Nothing
 *
 */

void ADCInt(void){

       TempStore0= ad0 & 0x03ff;   // Mask off the upper 6 bits of the
                                   // variable leaving only the result
                                   // in the variable itself

       TempStore1= ad1 & 0x03ff;   // Mask off the upper 6 bits of the
                                   // variable leaving only the result
                                   // in the variable itself

       TempStore2= ad2 & 0x03ff;   // Mask off the upper 6 bits of the
                                   // variable leaving only the result
                                   // in the variable itself

       TempStore3= ad3 & 0x03ff;   // Mask off the upper 6 bits of the
                                   // variable leaving only the result
                                   // in the variable itself

}
```

In order for this program to run properly, the ADC interrupt vector needs to point to the function. The interrupt vector table is near the end of the startup file “sect30.inc”. Insert the function label “_ADCInt” into the interrupt vector table at vector 14 as shown below.

```
        :
        :
        :

;---------------------------------------------------------------
; variable vector section
;---------------------------------------------------------------
        .section      vector        ; variable vector table
        .org   VECTOR_ADR


        .lword dummy_int            ; BRK  (vector 0)
        .org   (VECTOR_ADR+16)
        .lword dummy_int            ; int3(for user)(vector 4)
        .lword dummy_int            ; timerB5(for user)(vector 5)
        .lword dummy_int            ; timerB4(for user)(vector 6)
        .lword dummy_int            ; timerB3(for user)(vector 7)
        .lword dummy_int            ; si/o4 /int5(for user)(vector 8)
        .lword dummy_int            ; si/o3 /int4(for user)(vector 9)
        .lword dummy_int            ; Bus collision detection(for user)(v10)
        .lword dummy_int            ; DMA0(for user)(vector 11)
        .lword dummy_int            ; DMA1(for user)(vector 12)
        .lword dummy_int            ; Key input interrupt(for user)(vect 14)
        .glb   _ADCInt
        .lword _ADCInt              ; A-D(for user)(vector 14)
        .lword dummy_int            ; uart2 transmit(for user)(vector 15)
        .lword dummy_int            ; uart2 receive(for user)(vector 16)
        .lword dummy_int            ; uart0 transmit(for user)(vector 17)

        :
        :
        :
```