Application Note

Analog-to-Digital Conversion Techniques Using ZiLOG Z8 MCUs

AN004001-Z8X0400
This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Windows is a registered trademark of Microsoft Corporation.

Information Integrity
The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZILOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZILOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZILOG Sales Office to obtain necessary license agreements.

Document Disclaimer
© 2000 by ZILOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZILOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZILOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZILOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.
Table of Contents

General Overview ................................................................. 1
Discussion ............................................................................... 1
  Theory of Operation ............................................................ 2
Summary ................................................................................ 4
Technical Support ................................................................. 4
  Source Code ........................................................................ 4
  Flow Charts ......................................................................... 23
Test Procedure ....................................................................... 26
  Equipment Used .................................................................. 26
  General Test Setup and Execution ....................................... 26
  Test Results .......................................................................... 26
References ............................................................................ 28
Appendix ................................................................................ 29
  PCB Artwork ....................................................................... 29
  Schematics .......................................................................... 30

Acknowledgements

Project Lead Engineer
Chip Curtis

Application and Support Engineer
Chip Curtis

System and Code Development
Chip Curtis
Analog-to-Digital Conversion Techniques Using ZiLOG Z8 MCUs

Many applications requiring analog-to-digital conversions can be achieved with 8-bit MCUs without compromising accuracy, speed, or system cost.

General Overview

Many embedded controller applications require that an analog voltage be measured. Depending on the application, a separate A/D converter (ADC) chip may be required because of the speed and resolution requirements. However, many designs do not require fast conversion speeds, and 8 to 11 bits of resolution is adequate. For instance, a digital thermostat samples the temperature periodically and turns the heater or air conditioner on or off when the temperature hits a trip point. Here, the measurement speed for the voltage across a thermistor is not critical, because the temperature is changing rather slowly. Conversion times on the order of milliseconds are acceptable. Capturing fast-changing signals, such as audio, requires a much faster conversion rate. If the highest audio frequency is 4 KHz coming into the ADC, the sample rate must be at least twice that frequency (8 KHz). Because of the limited processing time between samples (in this case 125 µs), the ADC must complete a conversion quickly, giving the MCU time to process the data before the next sample. Because most designs are cost-sensitive, especially in consumer electronics, there may not be the luxury of adding relatively expensive ADC chips to the design. Design engineers must look for a more integrated solution, and ZiLOG has the solution.

Discussion

The on-board dual analog comparators, along with one counter/timer, are used to implement the ADC routines. The analog comparators are multiplexed with the digital inputs on port pins P31, P32, and P33. Figure 1 illustrates that configuration. The analog comparators are selected via the P3M register. The comparators share a common reference pin, P33. The input range of the comparators is 0V–4V. The input offset voltage is typically 10 mV with $V_{cc}$ at 5.0V. The output of the comparators can be examined by a Test under Mask (TM) instruction on port P3. The outputs also generate an interrupt, based on the falling or rising edge of the comparator output. These outputs can connect to the P34 and P37 output pins under software control. The PCON register in the extended register file controls this connection (not available on C04/E04 and C08/E08). The comparators are enabled during HALT mode, but are disabled in STOP mode. Three ADC configurations are presented:
The software routines are designed around the Z86E08/C08, but can be adapted for other selected Z8 and Z8Plus™ MCUs. The ZiLOG CCP emulator (ZiLOG PN Z86CCP00ZEM) was used to test the routines, and the routines also include a simple serial output to display the ADC output.

**Figure 1. Port 3 Configuration**

![Port 3 Configuration Diagram]

**Theory of Operation**

**Successive Approximation ADC**

For applications requiring fast conversion times, consider the successive approximation method (Figure 2). This method uses a Digital-to-Analog Converter (DAC) in its feedback loop. The DAC is comprised of an R2R ladder connected to port P2. When a binary value is output at Port 2, a DC voltage proportional to the binary value appears at pin 1 of the ladder network. The DAC completes a binary search on the input voltage. This search is achieved by first setting the most significant bit (MSB) of the DAC and testing the comparator output. If the comparator output is 0, the DAC output for this bit is set to 0. If the comparator output is 1, then the DAC output for this bit is set to 1. The bits from output port 2 are individually tested in ascending order, performing the same test. When all the bits are tested, the conversion is complete. The output from the R2R resistor network becomes the comparator’s reference voltage. The analog voltage to be measured...
can be connected to either P31 or P32, the non-inverting inputs of the comparators.

To start the conversion, the MSB of P2 is set, resulting in a voltage of 0.5 \( V_{cc} \) at the \( V_{REF} \) input of the comparator. If \( V_{cc} \) is 5V, then the voltage is 2.5V. The non-inverting comparator input is tested. If High, then the analog voltage must be 2.5V–5.0V.

The next bit, P26, is set, and the input port is tested again. If LOW, then bit P26 is reset. The process continues until all bits of P2 are tested. The resultant value at P2 is the digital representation of the analog input. With a crystal frequency of 8 MHz, the conversion time is approximately 110 \( \mu \)S. Even more resolution is available from 10- and 12-bit R2R networks. Of course, more resolution requires more port pins.

**Duty Cycle ADC**

When speed is not important, a Duty Cycle Analog to Digital Converter is the perfect solution. This method works by measuring the time it takes a capacitor to charge up above the input voltage and discharge below the input voltage. Because the charge time is compared to the discharge time, component tolerances have no effect on accuracy, and the reading is linear. In addition to the comparators, only one port pin is required to control the RC network, leaving the balance of the I/O free. This example performs a two-channel, 11 bit conversion.

Because this method is based upon measuring time, the duty cycle ADC requires a stable time base. The stable time base is accomplished using only one interrupt for the timer and ensuring that the software’s charge and discharge paths execute in the same amount of time.

At each timer interrupt period, the capacitor is compared to the input voltage. If the capacitor is greater than the input voltage, then the capacitor is discharged and the pass counter is decremented. If the capacitor is less than the input voltage, the capacitor is charged, the reading is incremented, and the pass counter is decremented. When the pass counter reaches 0 the conversion is complete.

For highest stability, the capacitor must be charged to the input voltage before measurement begins. The easiest way to charge the capacitor to the input voltage is to perform two conversions and discard the first one. Also, the total conversion time must be an even multiple of the line frequency (60Hz or 50Hz). This example is 2048 counts \( \times \) 130 \( \mu \)S = 266mS, approximately 16 cycles at 60 Hz, with a 1M resistor and a 0.1 \( \mu \)F capacitor. Total conversion time for both channels is 1.06S.
8-Bit Duty Cycle

The Duty Cycle conversion can be performed inline if a faster sampling rate is required. In this example, the time base is the code itself, so great care must be taken to ensure that each branch executes in the same number of clock cycles. The code contains a number of dummy instructions and balancing branches to keep all the timing consistent. Again, there can be no interrupts while the conversion is taking place. Because this routine executes faster, a different RC network is required. The value of the RC is not critical and is related to the total measurement period. On 5V systems, a good approximation for determining the time constant is \( RC = \frac{t}{2.75} \), where \( t \) is the total number of counts multiplied by the time for one count. This routine takes 108 clock cycles per count, or 27 \( \mu S \) on an 8-MHz resonator, for an RC of 2.5 mS (a 1 \( \mu F \) capacitor and a 25K resistor). This routine measures both channels in 28 mS.

Summary

Applications requiring analog-to-digital conversions can be achieved without compromising accuracy, speed, or system cost. Design engineers can experiment with the routines for performance.

Technical Support

Source Code

;TimerPorts.inc

;******************************Timers and Ports******************************

;These equates are for addressing individual bits

B0 equ 1b
B1 equ 10b
B2 equ 100b
B3 equ 1000b
B4 equ 10000b
B5 equ 100000b
B6 equ 1000000b
B7 equ 10000000b

;____________________________________Port0_______________________
Rint equ B0 ;Integrating resistor for Duty Cycle
Tx equ B1 ;Transmit

;____________________________________Port1_______________________

;Not available on an 08

;____________________________________Port2_______________________

;Port two is used for the successive approximation ADC, but does...
; not require individual bit assignments

Channel1 equ B1 ; Mask for channel 1
Channel2 equ B2 ; Mask for channel 2

Table 1. Init P01M Timer Control Register

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Field</th>
<th>R/W</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7–D6</td>
<td>P04–P07 Mode</td>
<td>R/W</td>
<td>00</td>
<td>P04–P07 Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00: Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01: Input</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1X: A12–A15</td>
</tr>
<tr>
<td>D5</td>
<td>External Timing</td>
<td>R/W</td>
<td>0</td>
<td>External Timing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0: Normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1: Extended</td>
</tr>
<tr>
<td>D4–D3</td>
<td>P10–P17 Mode</td>
<td>R/W</td>
<td>00</td>
<td>P10–P17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00: Byte output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01: Byte input</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10: AD0– = AD7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11: HiZ</td>
</tr>
<tr>
<td>D2</td>
<td>Internal Stack</td>
<td>R/W</td>
<td>0</td>
<td>1 Internal Stack</td>
</tr>
<tr>
<td>D1–D0</td>
<td>P00–P03 Mode</td>
<td>R/W</td>
<td>00</td>
<td>P00–P03 Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00: Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01: Input</td>
</tr>
</tbody>
</table>

Note: R = Read, W = Write, X = Indeterminate.
InitP01M equ 100B ;Internal stack, P0 Output

Table 2. Init P3M Timer Control Register

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Field</th>
<th>R/W</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>Parity</td>
<td>R/W</td>
<td>0</td>
<td>0: Parity ON 1: Parity OFF</td>
</tr>
<tr>
<td>D6</td>
<td>P30</td>
<td>R/W</td>
<td>0</td>
<td>0: P30 = Input, P37 = Output 1: P30 = Serial in, P37 = Serial out</td>
</tr>
<tr>
<td>D5</td>
<td>P31</td>
<td>R/W</td>
<td>0</td>
<td>0: P31 = Input, P36 = Output 1: P31 = DAV2/RDY2, P36 = RDY2/DAV2</td>
</tr>
<tr>
<td>D4 - D3</td>
<td>P33</td>
<td>R/W</td>
<td>00</td>
<td>00: P33 = Input 11: P33 = DAV1/RDY1, P34 = RDY/DAV1 01 or 10: P34 = Out, P33 = In P34 = DM</td>
</tr>
<tr>
<td>D2</td>
<td>P32</td>
<td>R/W</td>
<td>0</td>
<td>0: P32 = Input, P35 = Output 1: P32 = DAV0/RDY0, P35 = RDY0/DAV0</td>
</tr>
<tr>
<td>D1</td>
<td>P3</td>
<td>R/W</td>
<td>0</td>
<td>0: Digital P3 input 1: Analog P3 input</td>
</tr>
<tr>
<td>D0</td>
<td>P2</td>
<td>R/W</td>
<td>0</td>
<td>0: Open drain P2 1: Push-pull P2</td>
</tr>
</tbody>
</table>

Note: R = Read, W = Write, X = Indeterminate.
InitP3M equ 00000011B ;Analog push-pull

Table 3. InitTMR Timer Control Register

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Field</th>
<th>R/W</th>
<th>Reset Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7–D6</td>
<td>TOUT</td>
<td>R/W</td>
<td>00</td>
<td>00: No T out</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01: T0 out</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10: T1 out</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11: Internal clock out</td>
<td></td>
</tr>
<tr>
<td>D4–D5</td>
<td>CLKIN</td>
<td>R/W</td>
<td>00</td>
<td>00: External clock input</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01: Gated input</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10: Trigger input, no retrigger</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11: Trigger input, retrigger</td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>T1</td>
<td>R/W</td>
<td>0</td>
<td>0: Disable T1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1: Enable T1</td>
</tr>
<tr>
<td>D2</td>
<td>T1</td>
<td>R/W</td>
<td>0</td>
<td>1: Load T1</td>
</tr>
<tr>
<td>D1</td>
<td>T0</td>
<td>R/W</td>
<td>0</td>
<td>0: Disable T0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1: Enable T0</td>
</tr>
<tr>
<td>D0</td>
<td>T0</td>
<td>R/W</td>
<td>0</td>
<td>1: Load T0</td>
</tr>
</tbody>
</table>

Note: R = Read, W = Write, X = Indeterminate.
InitTMR equ 00001111B ;Load and run both timers

comment^

;_____________________________________PRE0_____________________

InitPRE0 equ 1<<2|1 ;8MHz/2SCLK/4TCLK/

;1PRE0 = 1MHz Modulo

InitT0 equ 130 ;1µs*130 = 130µs

comment^

;_____________________________________PRE1_____________________

InitPRE0 equ 1<<2|1 ;8MHz/2SCLK/4TCLK/

;1PRE0 = 1MHz Modulo

InitT0 equ 130 ;1µs*130 = 130µs

Table 4. InitPRE0 Timer Control Register

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Field</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2–D7</td>
<td>PRE1</td>
<td>R/W</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D1</td>
<td>Reserved</td>
<td>R/W</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D0</td>
<td>Count Mode</td>
<td>R/W</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: R = Read, W = Write, X = Indeterminate.

^  

Bit Position | Bit Field | R/W | Reset Value | Description
D2–D7        | PRE1      | R/W | 00          | Prescaler 1–64 (01h–00h)
D1           | Reserved  | R/W | 0           | Reserved; must be 0
D0           | Count Mode| R/W | 0           | Count Mode

0: Single pass
1: Modulo

Table 5. InitPRE1 Timer Control Register

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Field</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2–D7</td>
<td>PRE1</td>
<td>R/W</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D1</td>
<td>T1</td>
<td>R/W</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D0</td>
<td></td>
<td>R/W</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: R = Read, W = Write, X = Indeterminate.

Bit Position | Bit Field | R/W | Reset Value | Description
D2–D7        | PRE1      | R/W | 00          | Prescaler 1-64 (01h – 00h)
D1           | T1        | R/W | 0           | 0: External T1 source

1: Internal T1 source
D0           |           | R/W | 0           | 0: Single pass

1: Modulo

InitPRE1.equ(4*4)+3;(2sclk*4tclk/8MHz)*4 = 4µs, Int Mod

**************************************************************************
* Module Name:SDDAC.Asm
*       Copyright:(c)ZiLOG 1999, All Rights Reserved
*       Date:7/27/99
*       Created by:Chip Curtis
*       Modified by:
*       Description:This module performs a two channel Successive
       Approximation Analog to Digital Conversion
       When a conversion is complete the ASCII data
       is transmitted to an RS232 device.
****************************************************************
****************************************************************
****************************************************************
*       Include section
****************************************************************
****************************************************************
****************************************************************
include   "TimerPorts.inc";Timer and port definitions
****************************************************************
*       Global variables
****************************************************************
globals on
extern point_ah, point_al, point_bh, point_bl
****************************************************************
*       Global function declarations
****************************************************************
extern Xmit;Transmits conversion
****************************************************************
*       Interrupt Vectors
****************************************************************
vector  reset = Main
segmentcode
****************************************************************
*       Main
****************************************************************
Main:
;________________________________Initialization__________________
clr  imr ;No IQRs on, just in case
clr  rp  ;Reset rp just in case
ld   spl, #80H ;Initialize the stack. It should be
ld   p01m, #InitP01M ;empty at the top of the Main loop
clr  p2m  ;Initialize all port modes too
ld   p3m, #InitP3M
ld   pre1, #InitPRE1 ;Initialize T1 Prescaler
;________________________________Perform Conversion_____________
call SAADC ;Perform a Conversion
;________________________________Display______________________
clr point_ah            ;Only an 8 bit result
ld  point_al, channel1  ;Low byte
clr point_bh            ;Only an 8 bit result
ld  point_bl, channel2  ;Low byte
callXmit                 ;Send out ASCII
jrMain                    ;Do another conversion

*****************************************************************
* Function Name:SAADC
* Returns:channel1, channel2 loaded with ADC
* Entry Values: None
* Description: A two channel Successive Approximation Analog to
* to digital conversion is performed on P31 and P32
* *
* Notes         Depending on your application you may want to dis-
* able interrupts upon entry to this routine.
*****************************************************************
defineRam, SPACE= RFILE
segmentRam ;Our Ram
a2dPass   ds 1    ;Counter for passes through ADC loop
channel1  ds 1    ;Latched reading of channel 1
channel2  ds 1    ;Latched reading of channel 2
channelMaskds 1 ;Mask for addressing channels
segmentcode

;____________________________________Init________________________
SAADC:
    ld channelMask, #Channel1;Measure channel one first

SAADC40:
    clr P2 ;Clear the reference
    clr a2dPass ;Clear the loop counter
    scf ;Seed the counter

;_____________________________________Measure____________________
SAADC10:
    rrc a2dPass ;Next bit
    jr  C, SAADC20 ;All eight bits tested
    or P2, a2dPass ;Set the bit on port 2
    call SAADC50 ;Dummy call for a delay
    tm P3, channelMask ;Is the reference high or low?
    jr  NE, SAADC10 ;Still Low, leave ON, test a new bit
    xor P2, a2dPass ;Clear this bit
    jr  SAADC10 ;and test a new one
;___________________________Store Measurement____________________
SAADC20:
    cp channelMask, #Channel1; Did we finish channel 1?
    jr NE, SAADC30 ; Nope
    ld channel1, P2 ; Store the result from Port2
    ld channelMask, #Channel2; Read the next channel
    jr SAADC40 ; Start the measurement
SAADC30:
    ld channel2, P2 ; Store the result from P2
SAADC50:
    ret ; Conversion done

*****************************************************************
* Module Name: DutyCycle.Asm
* Copyright: (c) ZiLOG 1999, All Rights Reserved
* Date: 7/27/99
* Created by: Chip Curtis
* Modified by:
*
* Description: This is an example of a two channel 11 bit
* Duty Cycle Analog to Digital Conversion.
* When a conversion is complete the ASCII data
* is transmitted to an RS-232 device.
*
*****************************************************************
#include   "TimerPorts.inc"   ; Timer and port definitions
*****************************************************************
* Global variables
*****************************************************************
globals on
    extern point_ah, point_al, point_bh, point_bl
*****************************************************************
* Global function declarations
*****************************************************************
extern Xmit ; Transmits conversion
*****************************************************************
AN004001-Z8X0400
vector reset = Main
vector irq4 = A2D

Main:

;________________________Initialization________________________
ld sp1, #80H ;Initialize the stack.
ld pre0, #InitPRE0 ;Initialize T0 prescaler
ld pre1, #InitPRE1 ;Initialize T1 Prescaler
ld t0, #InitT0 ;Load T0 value
ld imr, #B4 ;Only T0 interrupt On
call InitADC ;Initial all ADC registers
ld tmr, #InitTMR ;Start the timers
ei ;Start the interrupts, ADC now on

;________________________Main Loop______________________________
clr rp ;Reset rp just in case
ld p01m, #InitP01M ;empty at the top of the Main loop
clr p2m ;Initialize all port modes too
ld p3m, #InitP3M

;_______________________Conversion Done?________________________
MainWait:
cp a2dPass+1, #Low 4096;Has the pass counter reset?
jr NE, MainWait ;Nope, wait
cp a2dPass, #High 4096 ;How about the high byte?
jr NE, MainWait ;Nope
cp channelMask, #Channel1;Is it the first channel?
jr NE, MainWait ;No, keep waiting

;________________________Display_______________________________
di ;Serial communication will start
ld point_ah, channel1 ;High byte
ld point_al, channel1+1 ;Low byte
ld point_bh, channel2 ;High byte
ld point_bl, channel2+1 ;Low byte
callXmit ;Send out ASCII

ei ;Serial communication finished

jrMainLoop ;Do another conversion

*****************************************************************
* Function Name: InitADC
* returns: None
* entry values: None
* Description: Initializes all ADC registers
* Notes
*****************************************************************

InitADC:

;Place functions in these sections, starting with init function. Example function called SysInit.

ret

*****************************************************************
; A2D Routine
*****************************************************************

;This routine performs a two channel A2D conversion by way of a timer ISR. The assumption will be made that the Timer and its interrupt have already been set-up, and that P3M has been configured as an analog port. As always, you should initialize your RAM too. The timing in this routine is critical, so don't mess with anything until AFTER the "Timing not critical" comment.

;This routine assumes that the timer is running at 130µs and that you are using a .1uF integrating capacitor and a 1M integrating resistor.

*****************************************************************
defineRam, ALIGN= 2, SPACE= RFILE

segmentRam

a2dReading dw 1
a2dPass dw 1 ;Counter for passes through ADC loop
channel1dw 1 ;Latched reading of channel 1
channel2dw 1 ;Latched reading of channel 2
channelMaskdb 1 ;Mask for addressing channels

segmentcode

Charge MACRO
  or P0, #Rint ;Change port assignment for your port
MACEND

DisCharge MACRO
  and P0, #~Rint ;Change port assignments for your port
MACEND

*****************************************************************
A2D:

```
  tm  P3, channelMask  ;10Cy Is the capacitor charged up?
  jr  NE, A2D10        ;10/12Cy Nope, still counting, keep
                      ;charging

;_______________________Discharge Path___________________________
  nop                   ;6Cy Delay so paths are equal
  nop                   ;6Cy Delay so paths are equal
  DisCharge            ;32Cy Total, Discharge the capacitor

;______________________Timing Not Critical_______________________
A2D30:
  cp  a2dPass, #HIGH 2048;Are we still settling the capaci
tor?
  jr  ULT, A2D20        ;No, we are counting
  clr a2dReading+1       ;Hold the reading at zero

A2D20:
  decw a2dPass            ;Next pass
  jr  NE, A2D5X           ;We aren't done yet

  cp  channelMask, #Channel1;Is this Channel1?
  jr  NE, A2D40           ;Nope, do channel two
  ld  a2dPass, #channel1;We'll use this as a pointer
  ld  channelMask, #Channel2;Now the other mask
  jr  A2D50               ;Reload

A2D40:
  ld  a2dPass, #channel2
  ld  channelMask, #Channel1 ;Now read channel one again

A2D50:
  ld  @a2dPass, a2dReading ;Store the HB
  inc  a2dPass            ;and
  ld  @a2dPass, a2dReading+1 ;the LB

  clr  a2dReading          ;Get ready to start all over
  clr  a2dReading+1

  ld  a2dPass, #HIGH 4096  ;2 passes of 2048 counts
  ld  a2dPass+1, #LOW 4096

A2DX:
  iret

;_______________________Charge Path______________________________
A2D10:
  incw a2dReading        ;10Cy We have a reading
  Charge                  ;32Cy Total, Charge Cap
  jr  A2D30               ;Rejoin common path
```

*****************************************************************
*       Module Name:DutyCycle8.Asm

AN004001-Z8X0400
* Copyright:(c)ZiLOG 1999, All Rights Reserved
* Date:7/27/99
* Created by:Chip Curtis
* Modified by:
* Description: This is an example of a two channel 8 bit
* Duty Cycle Analog to Digital Conversion.
* When a conversion is complete the ASCII data
* is transmitted to an RS-232 device.
*****************************************************************
*///////////////////////////////////////////////////////////////
*////////////////////////////////////////////////////////////////
*****************************************************************
* Include section
*****************************************************************
#include   "TimerPorts.inc" ; Timer and port definitions
*****************************************************************
* Global variables
*****************************************************************
globals on
extern point_ah, point_al, point_bh, point_bl
*****************************************************************
* Global function declarations
*****************************************************************
extern Xmit ; Transmits conversion
*****************************************************************
* Interrupt Vectors
*****************************************************************
vector  reset = Main
segmentcode
*****************************************************************
* Main
*****************************************************************
Main:
;________________________Initialization________________________
clr imr ; No IRQs on, just in case
clr rp ; Reset rp just in case
ld spl, #80H ; Initialize the stack. It should be
ld p01m, #InitP01M ; empty at the top of the Main loop
clr p2m ; Initialize all port modes too
ld p3m, #InitP3M
ld pre1, #InitPRE1 ; Initialize T1 Prescaler
;________________________Perform Conversion_________________
  callA2D ;Perform a Conversion
;__________________________Display_____________________________
  clr  point_ah ;Only an 8 bit result
  ld   point_al, channel1 ;Low byte
  clr  point_bh ;Only an 8 bit result
  ld   point_bl, channel2 ;Low byte
  callXmit ;Send out ASCII
  jrMain ;Do another conversion

***********************************************************
**       Module Name: Xmit.asm
**       Copyright: (c)ZiLOG 1999, All Rights Reserved
**       Date: 7/27/99
**       Created by: Chip Curtis
**       Modified by:
**       Description: This module is used to
** serial
*       port for testing. The two binary
*       words are passed to the routine,
*       to the routine, converted to BCD,
and then
*       transmitted.
*
***********************************************************
**       Include section
***********************************************************
**       include "TimerPorts.inc" ;Timer and port definitions
***********************************************************
**       Global variables
**globals** on

define Bank0, org= 00H, Space= RFILE; Ports are absolute
define Ram, Space= RFILE ; Anywhere else

segment Bank0
ds 4 ; Skip over ports

; The following assignments are used for passing arguments
; between functions using working register addressing. Their
; use is always considered temporary much like a multiple
; accumulator processor. They must reside in Bank0

point_ah ds 1 ; General purpose word high byte
point_al ds 1 ; General purpose word low byte
point_bh ds 1 ; General purpose word high byte
point_bl ds 1 ; General purpose word low byte
temp ds 1 ; General purpose single byte
loop ds 1 ; General purpose loop counter

***********************************************************
*
Xmit
***********************************************************
*
segment code

Xmit:
push point_bh ; Save all the data to transmit
push point_bl ; at a later time

; __________________________SendMessage_______________________

ld  point_bh, #High Ch1Msg; Point at the message
ld  point_bl, #Low Ch1Msg
call SendMsg          ; Print it

; __________________________ConvertMeasurement_____________________
call Bin2BCD           ; Make it BCD
call SendASCII         ; Transmit it
;____________________Send Message__________________________
ld R point_bh, #High Ch2Msg;Send the end
;of line
ld.R point_bl, #Low Ch2Msg;message
call  SendMsg

;_____________________Convert Measurement_____________________
pop  point_al ;Grab the other
;channel's data
pop  point_ah ;that was on the stack
call  Bin2BCD ;Convert it
call  SendASCII ;and send that too

ld .R point_bh, #High EOLMsg ;Send the end of
;line
ld .R point_bl, #Low EOLMsg ;message call
call  SendMsg
ret

******************************************************************************
*
Ch1Msg:
.asciz "Ch1: " ;Channel 1 prompt

******************************************************************************
*
*     Ch2Msg
******************************************************************************
*
Ch2Msg:
.asciz Ch2: " ;Channel 2 prompt
******************************************************************************
*
*     EOLMsg
******************************************************************************
*
EOLMsg:
**Function Name:** Bin2BCD
**Returns**: Result in pointb
**Entry Values**: 16 bit binary number in pointa
**Description**: Converts a 16 binary number to a 4 digit BCD number
**Notes**: Points and loop are destroyed in the process

```assembly
Bin2BCD:
clr point_bh ; Clear the destination
clr point_bl
ld .R loop, #16 ; Sixteen bits

Bin2BCD10:
rlc point_al ; Shift the LB over one
rlc point_ah ; and the HB, carry now
    ; has the bit
adc .R point_bl, .R point_bl ; add the result
    ; on itself
    ; for a X2
da point_bl
adc .R point_bh, .R point_bh ; and the HB
da point_bh
djnz.R loop, Bin2BCD10 ; Loop until done
ret
```

**Function Name:** SendMsg
**Returns**: None
**Entry Values**: point_b points at asciz message
**Description**: Uses SendChr to send an asciz
:message
*
* Notes  :
********************************************************************
* SendMsg:
   ldc .R temp, @.RR point_bh;Grab a character
   cp temp, #0         ;Is it The
                    ;Terminator?
   jr Nz, SendMsg10    ;No, I'll be back
   ret

SendMsg10
   call SendChr ;Transmit the character
   incw .RR point_bh ;Next character
   jr SendMsg ;Loop until finished

********************************************************************
* * Function Name:    SendASCII
* * Returns  :        None
* * Entry Values   :point_b holds BCD to send
* * Description   :Transmits a 4 digit BCD
                     :number in point_b
* *
* * Notes  :Also calls SendChr and destroys
           :loop,temp and point_a
********************************************************************
*
segment Ram

chrptr db 0 ;Character pointer
  segment code

SendASCII:

;________________________________Unpack BCD____________________________

   ld .R point_ah, #0FOH ;Only look at high
                  ;nibble
   ld .R point_al, #0FH ;Only look at low nibble
and .R point_ah, .R point_bh; Convert
  ; 1000s
swap point_ah ; Should be in
  ; low nibble
and .R point_al, .R point_bh; Convert 100s

ld .R point_bh, #0F0H ; Only look at
  ; high nibble
and .R point_bh, .R point_bl; Convert 10s
swap point_bh ; put in low
  ; nibble
and point_bl, #0FH ; Convert 1s
ld chrptr, #point_ah ; Point at first
  ; character

; __________________________________________Send As ASCII____________________

SendASCII10:
  ld .R temp, #30H ; Gonna make it ASCII
  add .R temp, @chrptr ; Convert to ASCII
  call SendChr ; Send ASCII character
  inc chrptr ; Next character
  cp chrptr, #point_bl+1 ; past the end?
  jr NE, SendASCII10 ; Nope, keep going
  ret ; All done

******************************************************************************
* Function Name: SendChr
* Returns None
* Entry Values ; temp holds character to send, TMR1
  ; is in point_b
* Description ; Transmits a 4 digit BCD number
* Notes ; Also calls SendChr and destroys
  ; loop, temp and point_a
******************************************************************************

OneBit equ 26; Time for 1 bit, so if T1
  ; is 4µs * 26 = 104µs or about
  ; 9600 baud.
SendChr:
;__________________________Init____________________
  ld  T1, #255           ;Count down from here
  or  TMR, #00001100B    ;Load and start timer

;__________________________Send a Start Bit_________________
  or   P0, #Tx            ;Send the Start bit
  ld .R loop, #255-OneBit;We wait for 1 bits

;______________________Character Loop_____________________
SendChr30:
  cp  T1, loop            ;Is it time for this bit?
  jr UGT, SendChr30       ;No, T1 is greater
                           ;(counts down)
  scf                     ;This will push in a sto
  rrc temp                ;bit
  jr  NC, SendChr20       ;Bit is low
  and P0, #~ Tx           ;Its a high so send that
                           ;(invert)

SendChr10:
  sub  loop, #OneBit      ;Wait for next bit
  cp  loop, #(255-(12*OneBit));Have we had
       ;enough stop bits?
  jr  NE, SendChr30       ;Nope, go back

;__________________________Delay_________________________
  clr  .R loop            ;Extra delay loop for some PCs

SendChr40:
  djnz  .R loop, SendChr40;Wait here for a bit
                           ;before next chr
  ret                       ;All done

;______________________If Bit Low________________________
SendChr20:
  or  P0, #Tx             ;Make it high (low level invert)
  jr  SendChr10
Flow Charts

Figure 2. Successive Approximation

Start

Init ChannelMask and counters

Shift pass counter over 1 bit

Tested 8 bits?

Set the port bit being tested

Reference > Input?

Push the bit back down

Was this channel 1?

Store CH1 result, prepare for next channel

Store CH2 result

Finished
Figure 3. Duty Cycle Flow

Start

Initialize Channel Mask

Initialize Reading and Pass counters

Should the Cap be charged?

Yes

Charge the Cap

Increment reading

Completed 1st read?

No

Hold reading at 0

Yes

Decrement pass counter

Completed all counts?

No

Store Ch1, load mask for Ch2

Yes

Is this Channel1?

No

Store channel 2 result

Yes

Discharge the Cap
Figure 4. 8-Bit Duty Cycle

- **Start**
- **Should the Cap be charged?**
  - Yes: **Charge the Cap** → Increment reading
  - No: **Discharge the Cap**
- **Completed 1st read?**
  - No: Hold reading at 0
  - Yes: **Decrement pass counter**
- **Completed all counts?**
  - Yes: **Store Ch2, load mask for Ch1** → Finish
  - No: **Is this Channel1?**
    - No: Store Ch1, load mask for Ch2 → Initialize Reading and Pass counters
    - Yes: **Initialize Reading and Pass counters**

The diagram illustrates the flow of operations for 8-bit duty cycle conversion, including initialization, reading, and handling of completed counts and channels.
Test Procedure

Equipment Used

- Z86CCP Emulator
- DMM
- 10 Turn Potentiometer
- PC with two serial ports.
- ZDS 2.11
- Hyperterminal

General Test Setup and Execution

A circuit board was constructed to test the Successive Approximation ADC, and a prototyping PCB was used for both Duty Cycle ADCs. A socket was used for the integrating resistor on the Duty Cycle PCB so that the two different resistors could be used. The emulator was configured for a Z86E08. Jumper J1 was in place to send $V_{cc}$ to the test board and an 8-MHz crystal was installed at the emulator’s X1. The potentiometer was configured as a voltage divider and its wiper was attached to the analog inputs of the ADC circuits. The serial output from the test board was connected to the PC and Hyperterminal was configured as Direct Connect, 9600 Baud, 8N1, no handshaking.

Each of the three programs was tested using the potentiometer to vary the analog input voltage and the DMM to measure the input voltage. The potentiometer should be installed across $V_{cc}$ and Ground, with the wiper attached to Channel 1 or Channel 2 input. The ADC was read from Hyperterminal and each ADC channel was tested at 5V increments. At the 1V level, the channel not being measured was shorted to $V_{cc}$ and then Ground to verify that there was no measurable interaction between the two channels. Error was calculated as $\text{Reading} \times (\text{Full Scale} (\text{Input Voltage} / V_{cc}))$. Error was calculated because the reading is ratio-metric, using $V_{cc}$ as its reference.

Test Results

Tables 6 through 8 illustrate the test results.
### Table 6. Successive Approximation Test Results

<table>
<thead>
<tr>
<th>Input</th>
<th>Ch1</th>
<th>Ch2</th>
<th>Average</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0.50</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>1.02</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>-1</td>
</tr>
<tr>
<td>1.54</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>-1</td>
</tr>
<tr>
<td>2.07</td>
<td>103</td>
<td>103</td>
<td>103</td>
<td>-1</td>
</tr>
<tr>
<td>2.56</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>3.06</td>
<td>153</td>
<td>153</td>
<td>153</td>
<td>0</td>
</tr>
<tr>
<td>3.49</td>
<td>174</td>
<td>174</td>
<td>174</td>
<td>-1</td>
</tr>
<tr>
<td>4.03</td>
<td>201</td>
<td>201</td>
<td>201</td>
<td>-1</td>
</tr>
</tbody>
</table>

### Table 7. Duty Cycle Test Results

<table>
<thead>
<tr>
<th>Input</th>
<th>Ch1</th>
<th>Ch2</th>
<th>Average</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>0.503</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>2</td>
</tr>
<tr>
<td>1.007</td>
<td>496</td>
<td>406</td>
<td>406</td>
<td>2</td>
</tr>
<tr>
<td>1.509</td>
<td>608</td>
<td>606</td>
<td>607</td>
<td>1</td>
</tr>
<tr>
<td>1.997</td>
<td>803</td>
<td>803</td>
<td>803</td>
<td>1</td>
</tr>
<tr>
<td>2.492</td>
<td>1001</td>
<td>1000</td>
<td>1000.5</td>
<td>0</td>
</tr>
<tr>
<td>3.015</td>
<td>1210</td>
<td>1209</td>
<td>1209.5</td>
<td>-1</td>
</tr>
<tr>
<td>3.509</td>
<td>1408</td>
<td>1409</td>
<td>1408.5</td>
<td>0</td>
</tr>
<tr>
<td>3.983</td>
<td>1599</td>
<td>1599</td>
<td>1599</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 8. 8-Bit Duty Cycle Test Results

<table>
<thead>
<tr>
<th>Input</th>
<th>Ch1</th>
<th>Ch2</th>
<th>Average</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>1.04</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td>1.55</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>1</td>
</tr>
<tr>
<td>2.02</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>0</td>
</tr>
<tr>
<td>2.47</td>
<td>124</td>
<td>124</td>
<td>124</td>
<td>0</td>
</tr>
<tr>
<td>2.99</td>
<td>150</td>
<td>150</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>3.54</td>
<td>177</td>
<td>177</td>
<td>177</td>
<td>0</td>
</tr>
<tr>
<td>4.02</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>-1</td>
</tr>
</tbody>
</table>

References

Appendix

PCB Artwork

Figure 5. Successive Approximation PCB
Schematics

Figure 6. Successive Approximation Schematic
Figure 7. Duty Cycle Schematic