University of British Columbia
Electrical and Computer Engineering
ELEC291/292

Lab 3: RS232, Temperature, 32-bit Arithmetic

Dr. Jesús Calviño-Fraga P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

January 24, 2025

About Lab #3

- There are two versions of the lab (pick one):
  - Using Matlab. Unfortunately Matlab is not free but you may have it from another course. Installed in the lab computers.
  - Using Python. You can download Python (version 3) for free. The version I use is WinPython (http://winpython.github.io/)
- For this lab you need to use the serial port, the analog to digital converter (ADC) in the N76E003, Python/Matlab, and the temperature sensor LM335. Several examples are provided in Canvas.
- Since some math is required to convert the ADC value to temperature, a library of 32-bit unsigned integer arithmetic and conversion routines is provided.

Objectives

- The ADC in the N76E003.
- Use the serial port to connect the N76E003 to a computer and interchange information.
- Measure temperature with the LM335.
- Perform 32-bit unsigned arithmetic using a library in assembly language.

The ADC in the N76E003

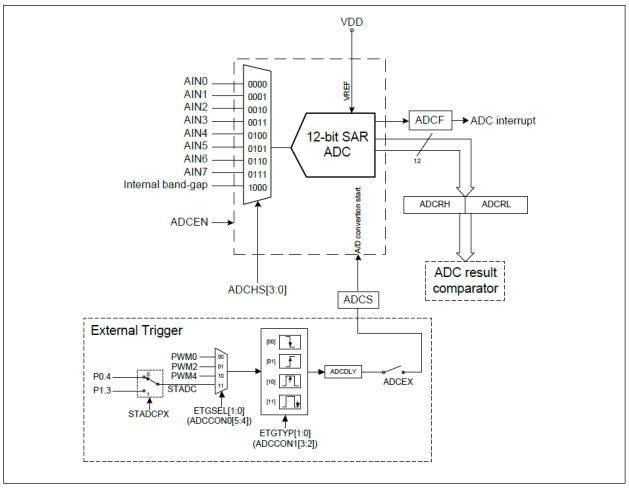
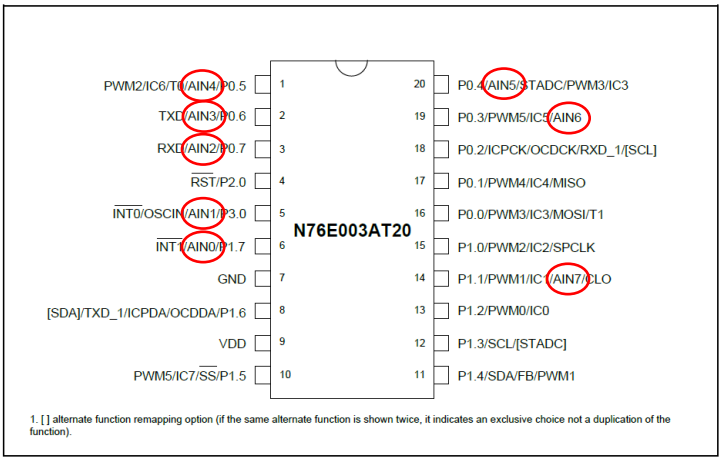


Figure 18.1-1. 12-bit ADC Block Diagram

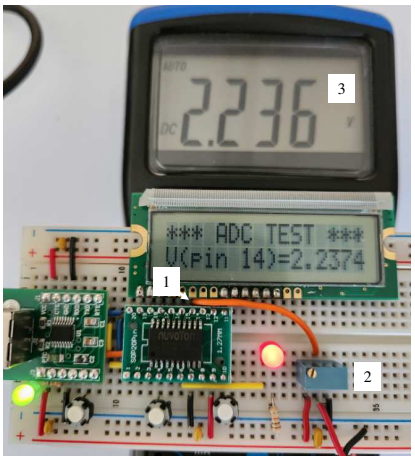
The ADC in the N76E003



1. [ ] alternate function remapping option (if the same alternate function is shown twice, it indicates an exclusive choice not a duplication of the function).

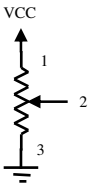
Figure 5.1-1. Pin Assignment of TSSOP-20 Package

Testing the ADC (one possible way)

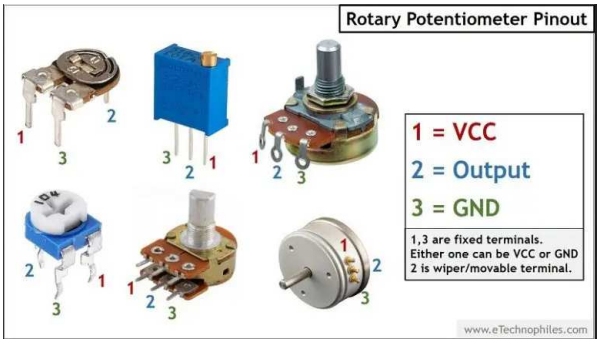


1. P1.1 (pin 14) is the analog input.
2. Potentiometer (I used 1k; 100k doesn't work very well). 'Fixed end 1' to 5V, 'fixed end 2' to GND, 'variable end' to P1.1.
3. Multi-meter and LCD show approximately the same voltage.

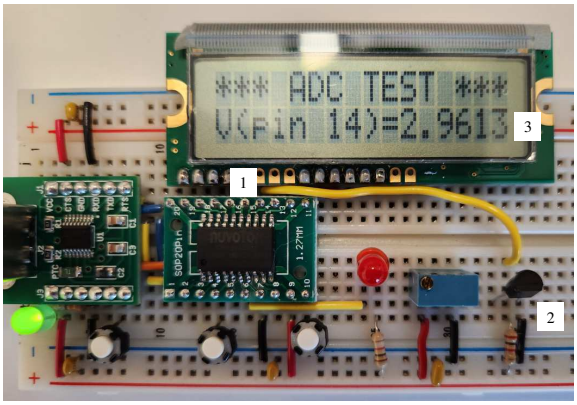
The potentiometer



Pins 1 and 3 are interchangeable.



Connecting the LM335



1. P1.1 (pin 14) is the analog input.
2. LM335 Temperature Sensor.
3. Voltage out of the LM335. Temperature is (2.96-2.73)*100=23°C

ADC Voltage Reference

- Using VCC as a voltage reference.

  VCC in this application comes from the USB port of the computer. It is approximately 5V (it changes a lot!). If you measure V_{CC} with the multimeter you can use it in your code:

$$V_{CH}=(ADC_{CH}*V_{CC})/4095$$

- Using the internal band-gap voltage reference. Section 18.1.4 of the N76E003 manual describes how to proceed:

Internal Band-Gap Reference

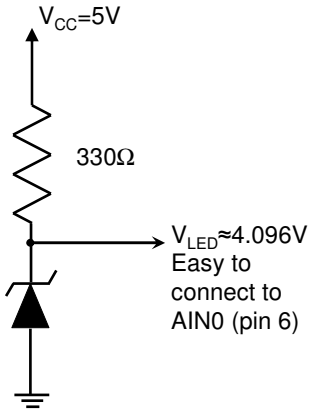
- Read the measured voltage during manufacturing stored in internal memory. Save it to a variable.
- Read the current band-gap value using the ADC. Must use the procedure in the manual! Save it to a variable.
- Use it in your code:

$$V_{CH}=(ADC_{CH}*V_{band\text{-}gap})/ADC_{band\text{-}gap}$$

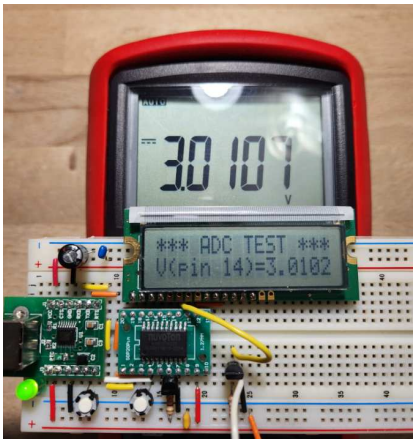
Bad news: The Internal Band-Gap Reference is not Very Good

- I hadn't had good results with the band-gap reference. An external reference seems to work better.
- LM4040-4.1 voltage reference. V_{REF}=4.0960V. Included in your kit.
- For the voltage calculation:

$$V_{CH}=(ADC_{CH}*V_{REF})/ADC_{REF}$$
Or
$$V_{CH}=(ADC_{CH}*40960)/ADC_{REF}$$



Using an external voltage reference



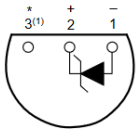
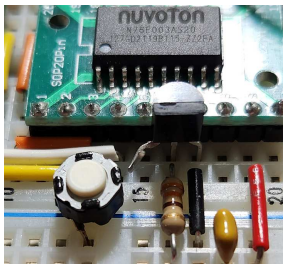


Figure 4-2. LP Package
3-Pin TO-92
Bottom View



The Serial Port

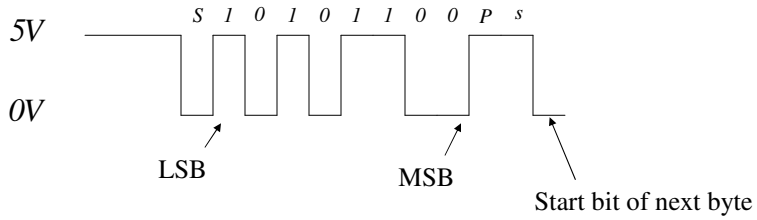
- The N76E003 as well as most popular microcontrollers have one or more serial ports.
- The serial port uses the RS-232 communication standard. It was introduced in 1962!
- Perhaps the easiest way to communicate between a microcontroller and a computer!
- RS-232 is asynchronous: the clock is not shared between the processors.

Asynchronous Data Communication Data Format

- A start bit used to synchronize the data. '0' or space.
- 5 to 8 data bits. For the standard 8051 the number of data bits is usually 8.
- Optional parity bit. Set or reset so that the number of ones transmitted is either odd or even. For the standard 8051 the parity is set to 'none' by default.
- One, one and half, or two stop bits. Always '1' or mark. For the standard 8051 is set to one stop bit.

Asynchronous Data Communication Data Format

- For example, transmit “00110101” using 8 bits, odd parity, one stop bit:



Baud Rate

$$BR = \frac{1}{t_{bit}}$$ Unit is ‘baud’

- Standard baud rates are: 110, 300, 600, 1200, 4800, 9600, 14400, 19200, 38400, and so on…
- The N76E003 with the correct oscillator frequency (for example 16.6 MHz) can generate all the standard baud rates up to 115200 baud!

Serial port in the N76E003

- To use the serial port in the N76E003:
  - Configure the baud rate using timer 1.
  - Configure the serial port mode using SFR SCON.
  - Transmit and receive using SFR SBUF.
  - For example if the microcontroller is running at 16.6 MHz:

```
; Configure serial port and baud rate
orl     CKCON, #0x10 ; CLK is the input for timer 1
orl     PCON, #0x80 ; Bit SMOD=1, double baud rate
mov     SCON, #0x52
anl     T3CON, #0b11011111
anl     TMOD, #0x0F ; Clear the conf. bits for timer 1
orl     TMOD, #0x20 ; Timer 1 Mode 2
mov     TH1, #0xF7 ; TH1=TIMER1_RELOAD;
setb    TR1 ; Start timer 1
```


Baud rate setup

- We use timer 1 as the baud rate generator:

$$TH1 = 256 - \frac{f_{osc}}{16 \times baud}$$

$$TH1 = 256 - \frac{16.6MHz}{16 \times 115200} = 247 = 0xF7$$

Timer 3 can be also used as baud rate generator.

Configure the serial port

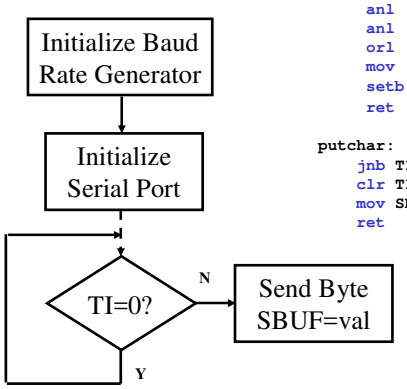
- The serial port in the 8051 has four operating modes. For RS-232 communications (same as personal computers) configure the serial port in mode 1.
- Use register SCON. The 8051 microcontroller serial port in 8-bit mode can be configured only for 8 data bits, no parity, one stop bit:
  - mov SCON, #52H; ; Mode 1, REN=1, TI=1

SCON SFR (Address 98H)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

- RI: If this bit is set, there is a newly received byte in register SBUF.
- TI: If this bit is set, the transmit buffer is empty. Writing a byte to SBUF will initiate transmission.
- RB8, TB8: The 9th bit in 9-bit UART mode.
- REN: Setting this bit to one enables serial reception.
- SM0, SM1: Configures serial port mode. For now set it to [0, 1], 8-bit UART
- SM2: Enables multiprocessor communication.

Serial Transmission




```
InitSerialPort:
     ; Configure serial port and baud rate
     orl  CKCON, #0x10 ; CLK is the input for timer 1
     orl  PCON, #0x80 ; Bit SMOD=1, double baud rate
     mov  SCON, #0x52
     anl  T3CON, #0b11011111
     anl  TMOD, #0x0F ; Clr the conf. bits for timer 1
     orl  TMOD, #0x20 ; Timer 1 Mode 2
     mov  TH1, #0xF7 ; TH1=TIMER1_RELOAD;
     setb TR1 ; Start timer 1
     ret

putchar:
    jnb TI, putchar
    clr TI
    mov SBUF, a
    ret
```


Serial Transmission (cleaner version)

```
CLK  EQU 16600000 ; Microcontroller system frequency in Hz
BAUD EQU 115200   ; Baud rate of UART in bps
TIMER1_RELOAD EQU (0x100-(CLK/(16*BAUD)))

InitSerialPort:
     ; Configure serial port and baud rate
     orl  CKCON, #0x10 ; CLK is the input for timer 1
     orl  PCON, #0x80 ; Bit SMOD=1, double baud rate
     mov  SCON, #0x52
     anl  T3CON, #0b11011111
     anl  TMOD, #0x0F ; Clr the conf. bits for timer 1
     orl  TMOD, #0x20 ; Timer 1 Mode 2
     mov  TH1, #0xF7 ; TH1=TIMER1_RELOAD;
     setb TIMER1_RELOAD ; Start timer 1
     ret

putchar:
    jnb TI, putchar
    clr TI
    mov SBUF, a
    ret
```

Example: Sending a String

```
$MODN76E003
org 0000H
   ljmp MainProgram

CLK  EQU 16600000
BAUD EQU 115200
TIMER1_RELOAD EQU (0x100-(CLK/(16*BAUD)))

InitSerialPort:
      ; Conf. all the pins for bidirectional I/O
      mov P3M1, #0x00
      mov P3M2, #0x00
      mov P1M1, #0x00
      mov P1M2, #0x00
      mov P0M1, #0x00
      mov P0M2, #0x00

      ; Some delay
      mov R1, #200
      mov R0, #104
      djnz R0, $ ; 4 cycles->4*60.285ns*104=25us
      djnz R1, $-4 ; 25us*200=5.0ms

      orl CKCON, #0x10 ; CLK is the input for timer 1
      orl PCON, #0x80 ; Bit SMOD=1, double baud rate
      mov SCON, #0x52
      anl T3CON, #0b11011111
      anl TMOD, #0x0F ; Clr the conf bits timer 1
      orl TMOD, #0x20 ; Timer 1 Mode 2
      mov TH1, #TIMER1_RELOAD
      setb TR1
      ret
```

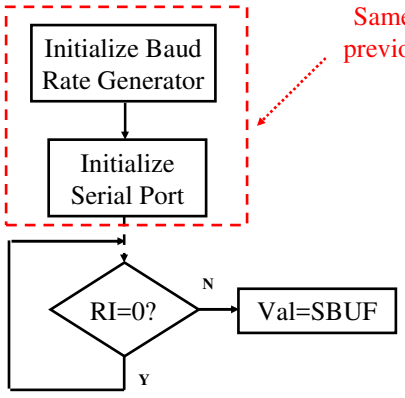
```
putchar:
    JNB TI, putchar
    CLR TI
    MOV SBUF, a
    RET

SendString:
    CLR A
    MOVC A, @A+DPTR
    JZ SSDone
    LCALL putchar
    INC DPTR
    SJMP SendString
SSDone:
    ret

Hello:  DB  'Hello, World!', 0AH, 0DH, 0
MainProgram:
    MOV SP, #7FH
    LCALL InitSerialPort
    MOV DPTR, #Hello
    LCALL SendString

    SJMP $
END
```

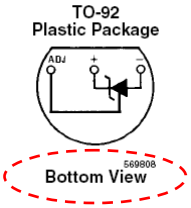

Serial Reception



Same code as previous slides.

```
getchar:
    jnb RI, getchar
    clr RI
    mov a, SBUF
    ret
```

LM335 Temperature Sensor



For the LM335:

+10mV/°K, -40°C<t<100°C

For the N76E003 :

ADC: 12-bit, 0.0V<V_{in}<4.096V

Un-calibrated temperature error: 2 to 6°C

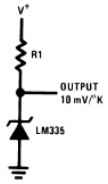
$$-40^0\,C = (273-40)^0\,K = 233^0\,K \rightarrow 2.33V$$

$$+100^0\,C = (273+100)^0\,K = 373^0\,K \rightarrow 3.73V$$

From the datasheet: "Included on the LM335 chip is an easy method of calibrating the device for higher accuracies. A pot connected across the LM335 with the arm tied to the adjustment terminal allows a 1-point calibration of the sensor that corrects for inaccuracy over the full temperature range."

LM335 Temperature Sensor

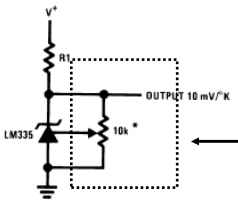
Figure 15. Basic Temperature Sensor



R1=2kΩ or 2.2kΩ, if you check the datasheet, most of the specs are @ 1 ma.

V^+=5V

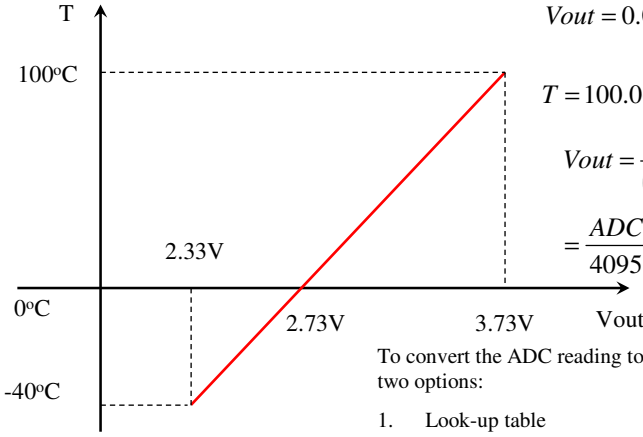
Figure 16. Calibrated Sensor



*Calibrate for 2.982V at 25°C

Calibration is not required for this lab, but it works really nice, especially for project 1! If you have a 10k resistor you can give it a try as it is very simple.

LM335 Transfer Function



$$Vout = 0.01\frac{V}{^{o}C} \times T + 2.73V$$

$$T = 100.0(Vout - 2.73V)\frac{^{o}C}{V}$$

$$Vout = \frac{ADC}{\left(2^{12}-1\right)} \times V_{REF}$$

$$= \frac{ADC}{4095} \times 5V$$

To convert the ADC reading to temperature you have two options:

1. Look-up table
2. Arithmetic

Math32 Library

- ***Math32.asm*** has the following functions:
  - **Hex2bcd**: Converts the 32-bit binary number in 'x' to a 10-digit packed BCD in 'bcd' using the double-dabble algorithm.
  - **Bcd2hex**: Converts the 10-digit packed BCD in 'bcd' to a 32-bit binary number in 'x'.
  - **add32**: x = x + y
  - **sub32**: x = x − y
  - **mul32**: x = x * y
  - **div32**: x = x / y
  - **x_lt_y**: mf=1 if x < y (mf is a bit)
  - **x_gt_y**: mf=1 if x > y
  - **x_eq_y**: mf=1 if x = y
  - **x_gteq_y**: mf=1 if x >= y
  - **x_lteq_y**: mf=1 if x <= y
- ***Math32.asm*** has the following macros:
  - **Load_X**: load x with a 32-bit constant
  - **Load_Y**: load y with a 32-bit constant

Math32 Test Program

- ***Mathtest.asm*** shows how to:
  - Define the x, y, bcd, and mf variables.
  - Include the math32 library in your program.
  - Use the Load_X and Load_X macros.
  - Convert a binary to BCD using bin2bcd and display it using the LCD.
  - Use the add32, sub32, mul32, and div32 functions.
  - Evaluate a formula using only integers.

Warning: You are using integer arithmetic!

$$Vout = \frac{ADC}{\left(2^{12}-1\right)} \times V_{REF} = \frac{ADC}{4095} \times 500$$

Suppose ADC=2524, compute Vout

$$Vout = \frac{2524}{4095} \times 500$$

Wrong!

Vout=(2524/4095) x 500 = (0) x 500 =0

Vout=(2524 x 500) / 4095 = (1262000) /4095 = 308 Right!

Final Remarks

- Lab 3 is an excellent starting point for project 1!