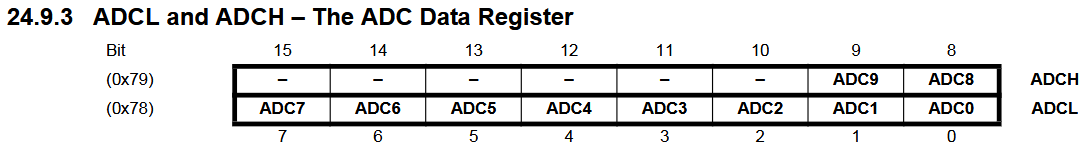
When an ADC conversion is complete, the result can be obtained from the 2-byte register pair **ADCH:ADCL**



As you know, the maximum value from this 10-bit ADC is 102310 or 11 1111 11112. The challenge for us to consider is how to use your UART knowledge and skill to display this value on the Arduino Serial Monitor such that the 2-byte binary number 11 1111 11112 is converted and transmitted as the ASCII character stream:  
 ‘3’, ‘2’, ‘0’, ‘1’ →

Fortunately, there is a well defined software solution known as the *Shift and Add-*3 or *Double Dabble* Algorithm. For the inspired hardware ACE, it must be noted that this Algorithm can also be implemented using logic gates!

The AVR Assembly code (AVR-as) code presented below sets the context for your implementation of the Double Dabble Algorithm. We’ll review this at the square table.

// PROJECT :DoubleDabble2025

// PURPOSE :Arduino IDE (avr-as) Double Dabble algorithm on the Nano

// COURSE :ICS4U-E

// AUTHOR :C. D'Arcy

// DATE :2025 03 30

// MCU :328P

// STATUS :Working. Ready for presentation to 24/25 ICS4U-E

// REFERENCE:http://darcy.rsgc.on.ca/ACES/TEI4M/Assembly/AVR8AssemblyLanguage.html

// :https://www.nongnu.org/avr-libc/user-manual/assembler.html

#include <avr/io.h> // to reference ports by name (-0x20 offset required)

.global main ;bypass Arduino C requirement for setup() & loop()

.equ BAUD\_PRESCALER, 103 ;precomputed for 16MHz OSC & 9600 BAUD

.equ ASYNCHRONOUS, 0<<UMSEL01 | 0<<UMSEL00

.equ SYNCHRONOUS, 0<<UMSEL01 | 1<<UMSEL00

.equ PARITY\_NONE, 0<<UPM01 | 0<<UPM00

.equ PARITY\_EVEN, 1<<UPM01 | 0<<UPM00

.equ PARITY\_ODD, 1<<UPM01 | 1<<UPM00

.equ CHAR\_SIZE8, 0<<UCSZ02 | 1<<UCSZ01 | 1<<UCSZ00 ;8-bit

.equ STOP\_BITS1, 0<<USBS0

.equ STOP\_BITS2, 1<<USBS0

.equ cTHREE,0x03 ;support for DD algorithm

.equ cTHREEZERO,0x30 ;support for DD algorithm

;avr-as syntax for SYMBOLIC Names of GP Registers

util = 16 ;assigns a symbolic name to a GP Register

bin0 = 18 ;

bin1 = 19 ;

BCD01 = 20 ;

BCD23 = 21 ;

BCD4 = 22 ;

offset = 23 ;

.text ;position for placement in Program Flash

.org 0x0000 ;position for start of Interrupt Vector Table (IVT)

rjmp main ;

.org \_VECTORS\_SIZE ;position beyond the IVT. See iom328p.h

data:

.word 65535 ;ADC Example. Maximum result from an ADC Conversion

converted:

.word 0x5535 ;hardcoded DD Conversion of data(above) for now

.word 0x0006 ;

.org 0x0200 ;provide the origin for the start of assembly code

main:

rcall init\_USART ;initialize baud, char size, parity, stop bits

ldi ZL,lo8(data) ;obtain the 16-bit data to be transmitted

ldi ZH,hi8(data) ;

lpm bin0,Z+ ;load the 16-bit source into R19:R18 register pair

lpm bin1,Z ;

;undertake the Double Dabble Algorithm

rcall doubleDabble ;convert

rcall display ;Confirmation on the Serial Monitor

rjmp .-2 ;hold (sleep would be better)

ret

;PreCondition: OR flags for UART initialization are defined

;PostCondition: UART is ready for service

init\_USART:

ldi util,hi8(BAUD\_PRESCALER) ;set the baud rate

sts UBRR0H,util ;"

ldi util,lo8(BAUD\_PRESCALER) ;"

sts UBRR0L,util ;"

ldi util,(1<<RXEN0) | (1<<TXEN0) ;enable receive and transmit

sts UCSR0B,util ;"

ldi util, ASYNCHRONOUS | CHAR\_SIZE8 | PARITY\_NONE | STOP\_BITS1

sts UCSR0C,util ;set Comm. Mode Default: 8N1

ret

;PreCondition: 16-bit value to be converted is in Register pair: R19:R18

;PostCondition: 3-Byte Packed BCD conversion are in Registers: R22:R21:R20

doubleDabble:

clr BCD01 ;zero the working target registers

clr BCD23 ;

clr BCD4 ;

;double dabble algorithm goes here...

ldi ZL,lo8(converted) ;for now, load the manually-defined conversion

ldi ZH,hi8(converted) ;

lpm BCD01,Z+ ;

lpm BCD23,Z+ ;

lpm BCD4,Z ;

ret

;PreCondition: the 3 DD working BCD registers are loaded with the conversion

;PostCondition: the 5 BCD values are transmitted through the UART

display:

ldi offset,'0' ;BCD values need an ASCII '0' (48) offset

mov r24,BCD4 ;prepare the most significant BCD digit

andi r24,0x0F ;mask off the high nibble

add r24,offset ;upgrade to ASCII character value

rcall TXCHAR ;echo (transmit) it

mov r24,BCD23 ;prepare the BCD 1000s digit

swap r24 ;exchange the high and low nibbles

andi r24,0x0F ;mask off the high nibble

add r24,offset ;upgrade to ASCII character value

rcall TXCHAR ;echo (transmit) it

mov r24,BCD23 ;prepare the BCD 100s digit

andi r24,0x0F ;mask off the high nibble

add r24,offset ;upgrade to ASCII character value

rcall TXCHAR ;echo (transmit) it

mov r24,BCD01 ;prepare the BCD 10s digit

swap r24 ;exchange the high and low nibbles

andi r24,0x0F ;mask off the high nibble

add r24,offset ;upgrade to ASCII character value

rcall TXCHAR ;echo (transmit) it

mov r24,BCD01 ;prepare the BCD 1s (units) digit

andi r24,0x0F ;mask off the high nibble

add r24,offset ;upgrade to ASCII character value

rcall TXCHAR ;echo (transmit) it

ret

; PreCondition: Character data (1 byte) to be transmitted is in R24

; PostCondition: Contents of R24 are transmitted via the USART (should appear on the Serial Monitor)

TXCHAR:

lds util,UCSR0A ;determine if the transmit buffer is empty

sbrs util,UDRE0 ;if the flag is set, exit the loop

rjmp .-8 ;loop back to the previous statement

sts UDR0, r24 ;transmit data in r24

ret

;PreCondition: None

;PostCondition: Character received in the input buffer is stored in R24

RXCHAR:

lds util,UCSR0A ;poll the RX flag

sbrs util,RXC0 ;is it set?

rjmp .-8 ;no? keep waiting...

lds r24,UDR0 ;load the incoming character into r24

ret