AVR Freaks

DESIGN NOTE #030

AUTHOR: DHANANJAY V. GADRE - DVGADRE@NETKRACKER.COM

KEYWORDS: UART, FREQUENCY SYNTHESIZER

Frequency Synthesizer

Introduction

Frequency Synthesizer using the concept of Numeric Controlled Oscillator. The Synthesizer is programmed through the serial port. An effective synthesizer for frequencies in the range of milli-Hz to 10s of KHz.

Can be used with Any AVR with a serial port uses a 40-bit summer value of the 40 bits is received from the serial port at 9600-8-N-1 ASCII encoded format: 123456789A Left to right --> MSB to LSB

Using the Synthesizer

For an 8MHz clock, and a 40-bit Adder as used in this Program, the user is required to send a data string from the PC in 9600-8-N-1 format. The string starts with " * " followed by a 10-digit number in ASCII. The 10 digit number "N" is calculated as follows: N = (2^40)*F*10^(-6) (i.e., N = 1099511.627776 * F). F is the required frequency and 10^(-6) is the cycle time for the adder loop.

Example: Lets assume you want 1KHz frequency. Therefore N = 004189374B. So transmit the following string to the AVR through the PC serial port: *004189374B and the AVR will generate 1 KHz signal on the PB7 pin. Similarly, to generate a 225 Hz signal, N = 000EBEDFA4. So the string is: *000EBEDFA4.

Code

Use the appropriate path of the 2313def.inc file on your system

```
;*****************************************
.include "c:\avrtools\appnotes\2313def.inc"
;*****************************************

.equ baudrate=$33 ; 9600 at 8MHz

.def sundry=r16
.def sum0=r20
.def sum1=r21
.def sum2=r22
.def sum3=r23
.def sum4=r24
.def rreg=r25
.def err_code=r26
.def rtemp=r16
.def f0=r1
```

```
.def f1=r2
.def f2=r3
.def f3=r4
.def f4=r5

.cseg
.org 0

rjmp STARTUP

;*********************************
;Receive data Interrupt Subroutine
.org 7
rjmp get_data
;*********************************

.org 20

STARTUP:
          ldi sundry, low(RAMEND) ; Stack pointer init.
          out SPL, sundry

          ldi sundry, 255         ;Output port for PORTB.
          out DDRB, sundry

;load default value for the NCO
          ldi sundry, $9c
          mov f0, sundry

          ldi sundry, $dc
          mov f1, sundry

          ldi sundry, $ef
          mov f2, sundry

          ldi sundry, $50
          mov f3, sundry

          ldi sundry, $00
          mov f4, sundry

          rcall init_uart
          sei
;************************************************
;Main Frequency Synthesizing Loop
loop:     add sum0, f0
          adc sum1, f1
          adc sum2, f2
```

```
          adc sum3, f3
          adc sum4, f4
          out PORTB, sum4         ; Output Frequency on PB7
          rjmp loop
;*************************************************

;*************************************************
;Initialize the UART for 9600-8-N-1 Data Format
init_uart:
          ldi rtemp, baudrate     ;setting baud rate
          out UBRR,rtemp
          ldi rtemp, $98          ;initialising UART control register
                                  ;RXCIE interrupt is enabled.
          out UCR, rtemp
          ret
;*************************************************

;*************************************************
;Serial Data Receive ISR
;Routine is invoked when a byte is received.
;The Routine then receives 7 more bytes, if the first
;byte is '*'. Else it returns back to the main program

get_data:
          in r0, sreg
          in sundry, ucr          ;disable further receive interrupt
          andi sundry, $7f
          out ucr, sundry

rxcomp:   sbis USR,RXC            ;poll to check if char received
          rjmp rxcomp
          in rreg,UDR             ;put received data in rreg
          out UDR, rreg           ;echo the character

          cpi rreg, 42            ;start of data header is: '*'
          brne go_back            ; If not, GO BACK.

          ldi err_code, 0

          rcall rx_byte
          cpi err_code, 0
          brne go_back
          mov f4, sundry

          rcall rx_byte
          cpi err_code, 0
          brne go_back
          mov f3, sundry
```

```
                rcall rx_byte
                cpi err_code, 0
                brne go_back
                mov f2, sundry

                rcall rx_byte
                cpi err_code, 0
                brne go_back
                mov f1, sundry

                rcall rx_byte
                cpi err_code, 0
                brne go_back
                mov f0, sundry
                clr sum0
                clr sum1
                clr sum2
                clr sum3
                clr sum4


go_back:
                in sundry, ucr
                ori sundry, $80         ;enable receive complete interrupt again
                out ucr, sundry
                out sreg, r0
                reti


rx_byte:
                sbis USR, 7             ;poll to check if char received
                rjmp rx_byte
                in rreg, UDR
                out UDR, rreg

                cpi rreg,$40
                brsh asnxt
                subi rreg,$30
                rjmp asnxt1
asnxt:          subi rreg,$37
asnxt1:         mov sundry, rreg
                swap sundry
                andi sundry, $f0
rx_byte1:
                sbis USR, 7             ;poll to check if char received
                rjmp rx_byte1
                in rreg, UDR
```

```
            out UDR, rreg
            cpi rreg,$40
            brsh asnxt2
            subi rreg,$30
            rjmp asnxt3
asnxt2:     subi rreg,$37
asnxt3:     andi rreg, $0f
            or sundry, rreg
            ret
```