Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

Lab 4: Waveform Generator

Overview/ Objective:

In this lab, we have setup the 9S12XD timer module to generate a timer channel 0 output compare interrupt every 0.1 msec. By doing so, we can generate waveforms. Throughout the lab, we had to build a program capable of taking in user input from the keypad to select a waveform as well as a one-byte integer, NINT, which represents the number of interrupts per Basic Time Interval (BTI). At the beginning of each BTI, the program updates the DAC input and outputs this value to the oscilloscope, creating our waveform. The waveforms we can create are a sawtooth wave, a 7-segmented sine wave, a 15-segmented sine wave, and a square wave. The program is also complete with error messages to create a user-friendly experience.

Tasks:

Task_1: Mastermind

States:

t1state0: Init
- Clears relevant variables
- Sets next state

t1state1: Hub
- Tests for CHAR_RDY
- If CHAR_RDY = 1, determines what that character will be used for, NINT or wave selection
- Branches to relevant section based on KEY_BUFF value
- Sets next state

t1state2:
- Checks if we are not able to delete the current character
- Sets next state

t1state3: Backspace state
- Checks BS_FLG and determines which state of the backspace process we are at
- Decrements BUFFCOUNT at the end of the backspace as well as set BS_FLG for display to start BS process
- Sets next state

t1state4: TBD error

- Clears BUFFCOUNT
- Sets next state

t1state5: No digits error
- clear BUFFCOUNT
- sets next state

t1state6: Zero error
- clear BUFFCOUNT
- sets next state

t1state7: Enter pressed
- Clears flags that got us here
- Uses ASC_BIN to convert our inputs from the keypad
- Checks for errors, branches if necessary
- Sets Run = 1
- Sets next state

t1state8: Error Delay
- Checks EDEL_FLG
- Stays in t1state8 if not equal to 0, meaning the delay isn't done
- Sets next state

t1state9: Delay
- sets flag for display to show error message
- sets next state

t1state10: Digit Handler
- Checks if accepting input still, ACCINPUT = 1
- Loads KEY_BUFF and evaluates if it is a backspace or a valid number
- if BUFFCOUNT is at a maximum value, no more numbers can be entered
- Branches if ACCINPUT = 0
- Sets next state

Task_2: Keypad

States:

t2state0: Initialization
- Initializes keypad
- Sets next state

t2state1: Waiting for key

- Tests LKEY_FLG
- JSR getchar to grab key
- Stores key in keybuff
- Sets CHAR_RDY = 1
- Sets next state

t2state2: CHAR clear waiting

- Checks for CHAR_RDY to get cleared
- Sets next state

Task_3: LCD Display

States:

t3state0: Initialization

- clears relevant task variables
- sets next state

t3state1: Hub

- Tests if we are in SETUP stil
- If yes, sets WAVES_FLG
- Checks all relevant display flags and branches accordingly
- If not, we skip SETUP
- Sets next state

t3state2: Base waves message

- Loads screen with base waves messages
- Sets next state

t3state3: Saw message

- Loads screen with saw selected messages
- Sets next state

t3state4: Sin7 message

- Loads screen with sin7 selected messages
- Sets next state

t3state5: Sin15 message

- Loads screen with sin15 messages
- Sets next state

t3state6: Square message

- Loads screen with square selected messages
- Sets next state

t3state7: Too big error

- Loads screen with TDB error message
- Sets next state

t3state8: Zero error

- Loads screen with Zero error message
- Sets next state

t3state9: No digits error

- Loads screen with no digits entered error message
- Sets next state

t3state10: Backspace

- Deletes one character through a multi step process of movin the cursor back one, replacing the value with a null character, moving the cursor back again, and waiting for another character to be entered
- Sets next state

t3state11: Echo

- Echoes values as they are typed into the keypad
- Sets next state

t3state12: Error delay

- Decrements EDELCOUNT and tests if it is equal to 0
- If equal to 0, delay is over, clear all relevant flags and return to main
- Otherwise, stay in this state and continue looping through and decrementing until we get to 0
- Sets next state

Task_4: Timer Channel 0

States:

t4state0: Initialization

- Bsets relevant interrupt settings
- Loads timer counter, adds interval, and stores back in D
- Sets next state
-

t4state1: waiting

- Loops indefinitely

Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

Task_5: Function Generator

States:

t5state1: Initialization

- Sets next state
-

t5state1: waiting for wave

- Tests if RUN = 1
- If RUN = 1, determines which wave is to be output based on wave specific flags
- Moves wave message to WAVEPTR
- Sets next state

t5state2: New wave

- Tests for DWAVE = 1
- Once DWAVE = 1, initializes CSEG, VALUE, LSEG, and SEGINC based on current wave
- Increments SEGPTR to the next segment
- Stores SEGPTR
- Sets flag to display NINT prompt
- Sets next state

t5state3: Waiting for NINT from keypad

- Tests RUN
- If RUN = 0, rts
- If RUN = 1, set next state
-

t5state4: Display wave

- Checks if RUN = 0, as it will then not display anything
- Decrement LSEG, CSEG, and compare them to 0
- If either are -, they will branch to t5s4a or t5s4b where they will start a new segment, or get the current DAC input value
- If not at the end of the segment, then we don't need to reinit the wave
- Sets next state

Inter-Task Communication Variables:

Variable Name	**Description**	**Task Set**	**State Set**	**Task Cleared**	**State Cleared**
SQR_FLG	Flag for square wave message	**1**	**1**	**3**	**6**
CHAR_RDY	Flag for character ready	**1**	**1**	**2**	**1**
DWAVE	Flag to start displaying a wave	**3**	**1**	**3**	**3 4 5**
ECHO_FLG	Flag for the echo display task	**1**	**10**	**3**	**11**

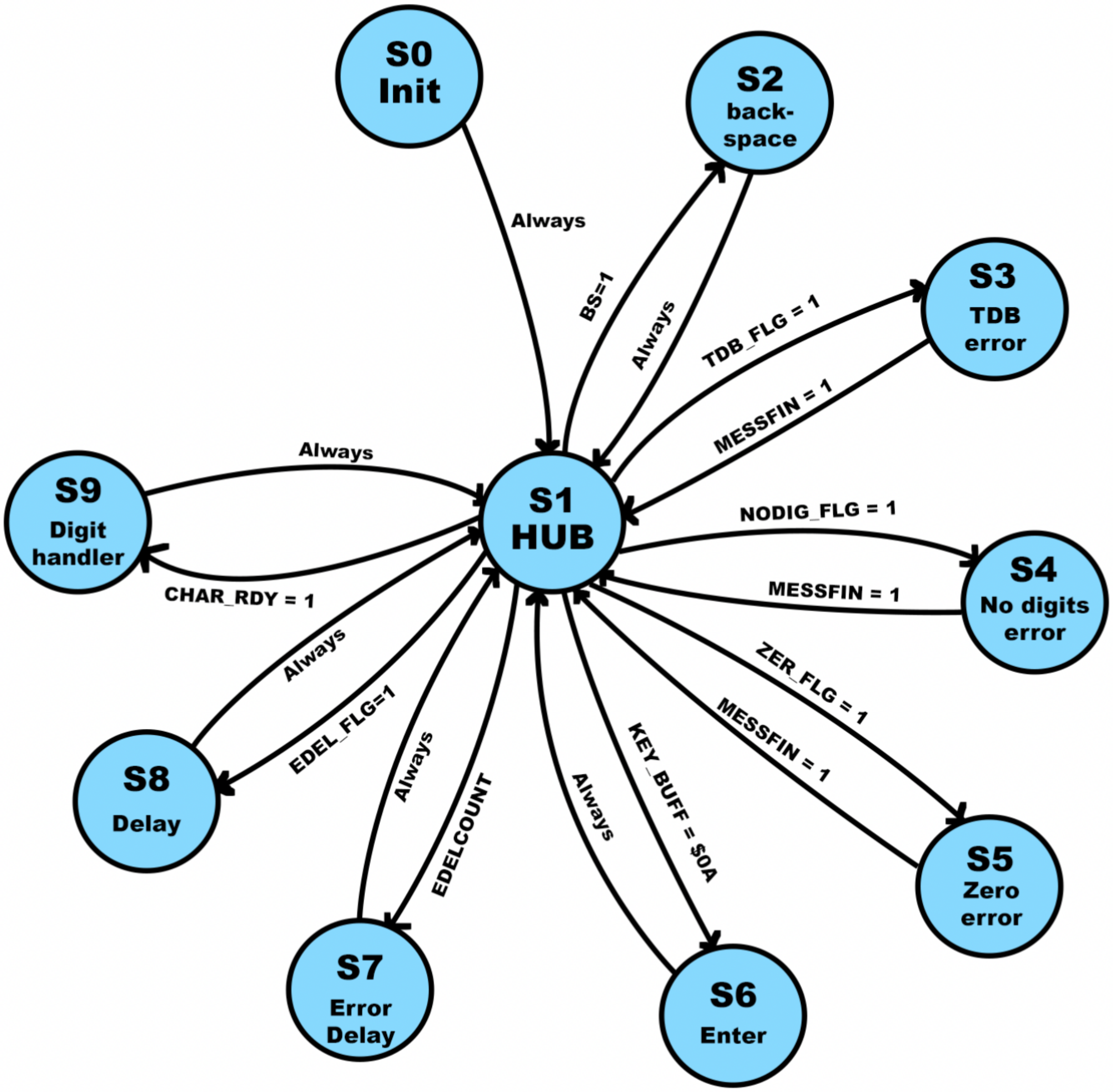
Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

RUN	Flag to control whether the timer and function generator should be running	**1**	**7**	**1**	**1**
SIN7_FLG	Flag to display 7 segment sin message	**1**	**1**	**3**	**4**
ZER_FLG	Flag to display zero error message	**1**	**7**	**3**	**8**
NODIG_FLG	Flag to display no digits error	**1**	**7**	**3**	**9**
TDB_FLG	Flag to display the too large of an input error message	**1**	**7**	**3**	**7**
NEWBTI	Flag to tell func. generator a BTI has passed	interrupt	**N/A**	**5**	**4**
MESSFIN	Flag to say that a message has finished displaying	**3**	**Dispcharsr**	**3**	**all**
EDEL_FLG	Flag to wait for a length of time while error message displays	**1**	**9**	**3**	**9**
BS_FLG	Flag to trigger a backspace from display	**1**	**3**	**1**	**3**
SAW_FLG	Flag to trigger saw wave prompt	**1**	**1**	**3**	**3**
SIN15_FLG	Flag to trigger 15 seg sine wave prompt	**1**	**1**	**3**	**5**
NINT_FLG	Flag to say that NINT is accepting input	**3**	**3**	**1**	**7**
SAW_WAVE	Flag to say saw wave is being displayed	**1**	**1**	**1**	**1**
SIN7_WAVE	Flag to say 7 segment sine wave is being displayed	**1**	**1**	**1**	**1**
SIN15_WAVE	Flag to say 15 segment sine wave is being displayed	**1**	**1**	**1**	**1**
SQR_WAVE	**Flag to say square wave is being displayed**	**1**	**1**	**1**	**1**

Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

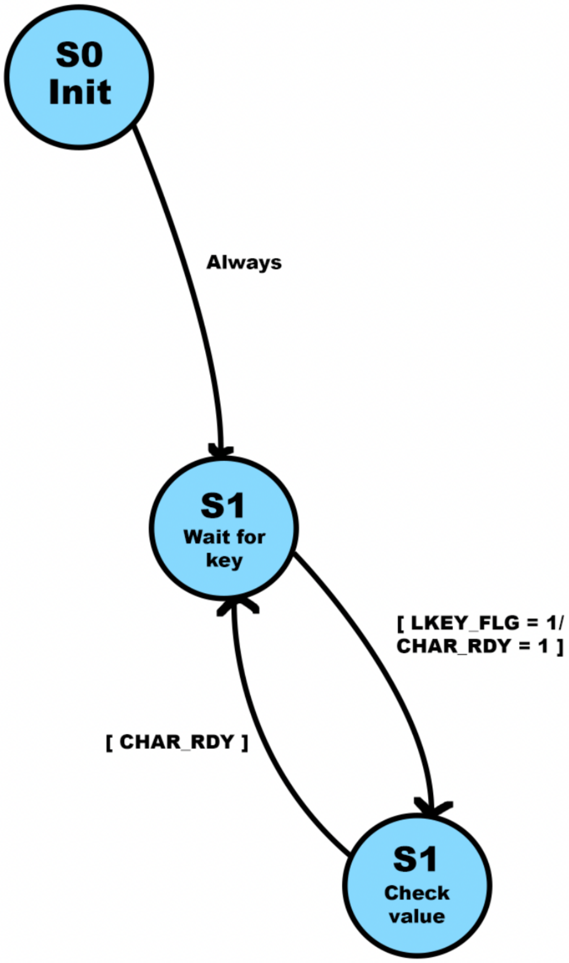
Finite State Diagrams:

Mastermind:



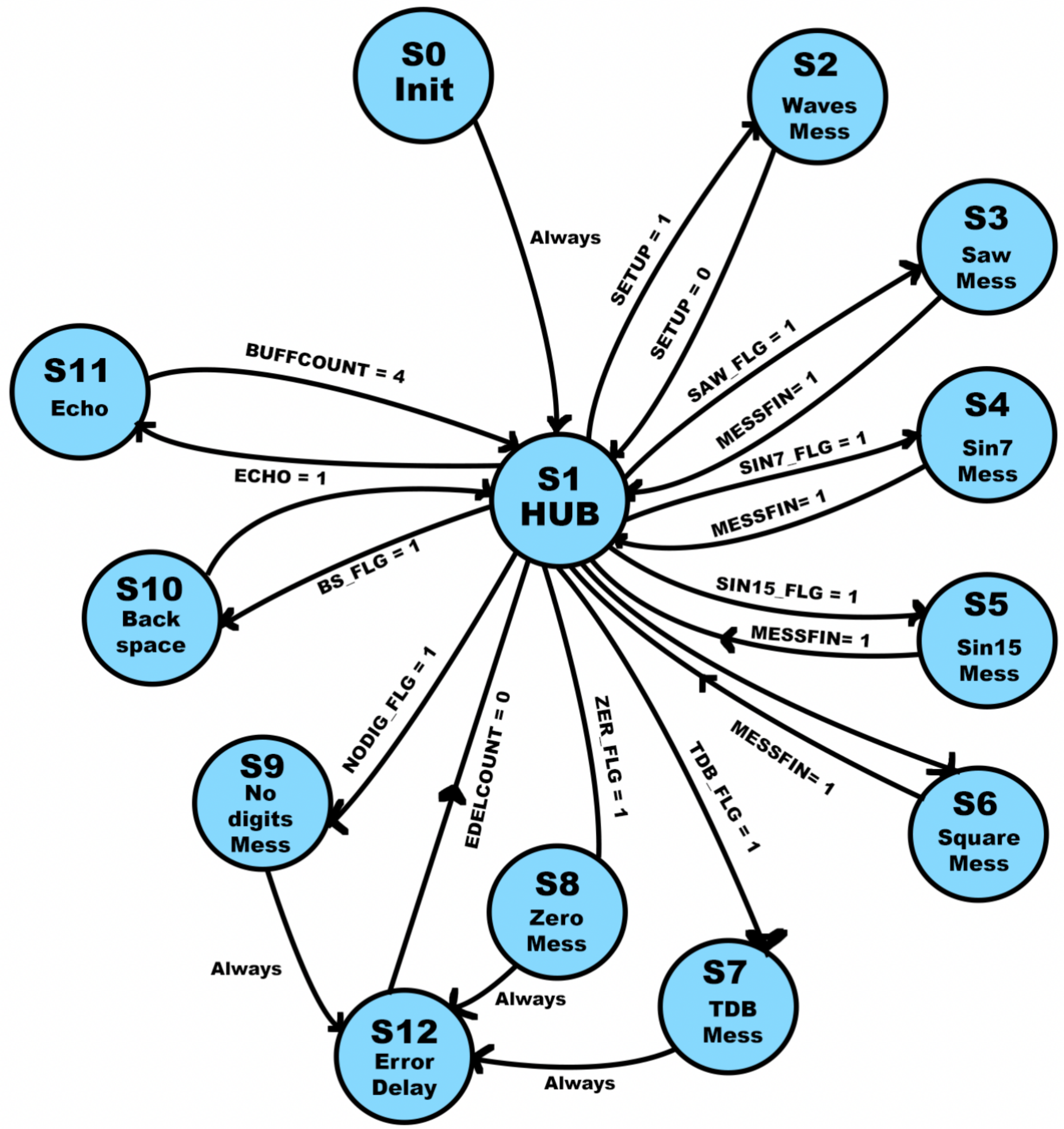
Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

Keypad



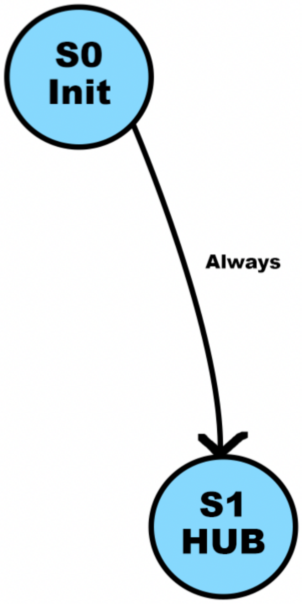
Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

LCD Display



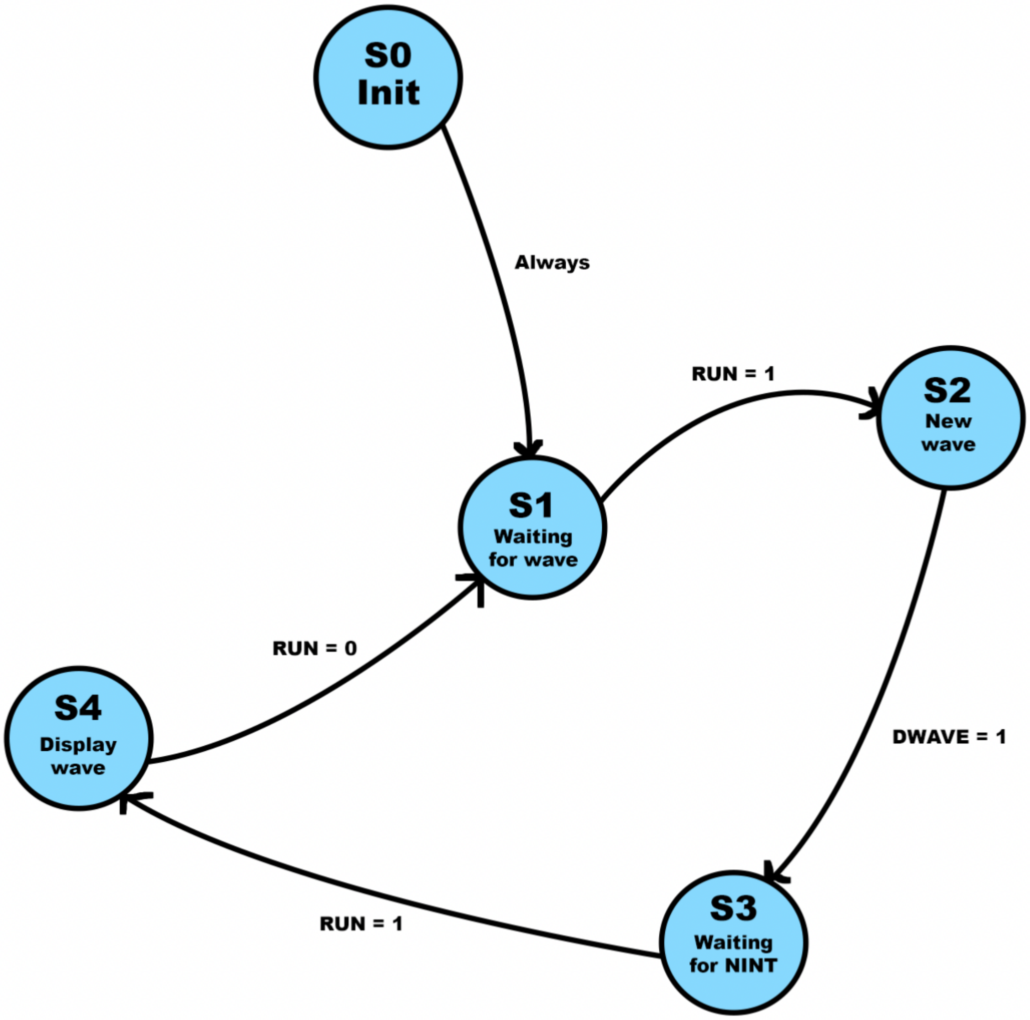
Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

Timer Channel 0



Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

Function Generator



Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

Source Code:

```
;**************************************************************************************
;* Blank Project Main [includes LibV2.2]                                              *
;**************************************************************************************
;* Summary:                                                                           *
;*   -                                                                                *
;*                                                                                    *
;* Author: Noah Tanner, Cole Sterba                                                   *
;*   Cal Poly University                                                              *
;*   Spring 2023                                                                      *
;*                                                                                    *
;* Revision History:                                                                  *
;*   -                                                                                *
;*                                                                                    *
;* ToDo:                                                                              *
;*   -                                                                                *
;**************************************************************************************

;/------------------------------------------------------------------------------------\
;| Include all associated files                                                       |
;\------------------------------------------------------------------------------------/
; The following are external files to be included during assembly


;/------------------------------------------------------------------------------------\
;| External Definitions                                                               |
;\------------------------------------------------------------------------------------/
; All labels that are referenced by the linker need an external definition

              XDEF  main

;/------------------------------------------------------------------------------------\
;| External References                                                                |
;\------------------------------------------------------------------------------------/
; All labels from other files must have an external reference

              XREF  ENABLE_MOTOR, DISABLE_MOTOR
              XREF  STARTUP_MOTOR, UPDATE_MOTOR, CURRENT_MOTOR
              XREF  STARTUP_PWM, STARTUP_ATD0, STARTUP_ATD1
              XREF  OUTDACA, OUTDACB
              XREF  STARTUP_ENCODER, READ_ENCODER
              XREF  INITLCD, SETADDR, GETADDR, CURSOR_ON, CURSOR_OFF, DISP_OFF
              XREF  OUTCHAR, OUTCHAR_AT, OUTSTRING, OUTSTRING_AT
              XREF  INITKEY, LKEY_FLG, GETCHAR
              XREF  LCDTEMPLATE, UPDATELCD_L1, UPDATELCD_L2
              XREF  LVREF_BUF, LVACT_BUF, LERR_BUF,LEFF_BUF, LKP_BUF, LKI_BUF
              XREF  Entry, ISR_KEYPAD

;/------------------------------------------------------------------------------------\
;| Assembler Equates                                                                  |
;\------------------------------------------------------------------------------------/
; Constant values can be equated here

TIOS  EQU   $0040                      ; set timer channel 0 for output compare
TCNT  EQU   $0044                      ; current timer count high
TSCR1 EQU   $0046                      ; current timer count low
TCTL2 EQU   $0049                      ; enable channel timer 0 output compare interrupts
TIE   EQU   $004C                      ; Timer control register 2
TFLG1 EQU   $004E                      ; Timer system control register 1
TC0   EQU   $0050                      ; Timer input capture/output compare reg 0 High
TMSK1 EQU   $004C                      ; Timer marsk one to control interrupts
TSCR  EQU   $0046
TC0H  EQU   $0050
```

Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

```
;/------------------------------------------------------------------------------------\
;| Variables in RAM                                                                   |
;\------------------------------------------------------------------------------------/
; The following variables are located in unpaged ram

DEFAULT_RAM:  SECTION
SEGINC        DS.W                 1
BUFFCOUNT     DS.B                 1
NINT          DS.B                 1
VALUE         DS.B                 1
SQUARE_FLG    DS.B                 1
CHAR_RDY      DS.B                 1
SQR_FLG       DS.B                 1
DWAVE         DS.B                 1
ECHO_FLG      DS.B                 1
DSPCOUNT      DS.B                 1
WAVEPTR       DS.W                 1
NODIG_FLG     DS.B                 1
RUN           DS.B                 1
t1state       DS.B                 1
t2state       DS.B                 1
t3state       DS.B                 1
t4state       DS.B                 1
t5state       DS.B                 1
INTERVAL      DS.W                 1
SIN7_FLG      DS.B                 1
ERR_FLG       DS.B                 1
KEY_BUFF      DS.B                 1
ZER_FLG       DS.B                 1
WAVES_FLG     DS.B                 1
TDB_FLG       DS.B                 1
NEWBTI        DS.B                 1
SEGPTR        DS.W                 1
LSEG          DS.B                 1
BUFFER        DS.B                 3
MESSFIN       DS.B                 1
DSPSTART      DS.B                 1
DPRMT         DS.B                 1
EDEL_FLG      DS.B                 1
C0F           DS.B                 1
CINT          DS.B                 1
BS_FLG        DS.B                 1
SAW_FLG       DS.B                 1
EDELCOUNT     DS.W                 1
SIN15_FLG     DS.B                 1
CSEG          DS.B                 1
SETUP         DS.B                 1
SET_FLG       DS.B                 1
NINT_FLG      DS.B                 1
BIN_RES       DS.B                 1
ACCINPUT      DS.B                 1
TEMP          DS.B                 1
SAW_WAVE      DS.B                 1
SIN7_WAVE     DS.B                 1
SIN15_WAVE    DS.B                 1
SQR_WAVE      DS.B                 1

;/------------------------------------------------------------------------------------\
;|  Main Program Code                                                                 |
;\------------------------------------------------------------------------------------/
; Your code goes here

MyCode:       SECTION
main:
            clr  t1state
            clr  t2state
            clr  t3state
            clr  t4state
            clr  t5state

top:
            jsr  task_1
            jsr  task_2
            jsr  task_3
            jsr  task_4
            jsr  task_5
            bra  top


;spin:      bra   spin                     ; endless horizontal loop


;/-------------------------------------------------------------------------------------\
;| Subroutines                                                                         |
;\-------------------------------------------------------------------------------------/
; General purpose subroutines go here

;-------------------------------TASK_1 MASTERMIND-------------------------------------------

task_1:                                ; mastermind
            ldaa  t1state              ; get current state and branch accordingly
            lbeq  t1state0
            deca
            lbeq  t1state1
            deca
            lbeq  t1state2
            deca
            lbeq  t1state3
            deca
            lbeq  t1state4
            deca
            lbeq  t1state5
            deca
            lbeq  t1state6
            deca
            lbeq  t1state7
            deca
            lbeq  t1state8
            deca
            lbeq  t1state9
            deca
            lbeq  t1state10
            rts

t1state0:                              ; init
            clr  SAW_FLG               ; clear relevant state flags
            clr  SIN7_FLG
            clr  SQR_FLG
            clr  SIN15_FLG
            clr  KEY_BUFF
            clr  CHAR_RDY
            clr  ACCINPUT
            clr  NINT_FLG
            clr  ECHO_FLG
            clr  RUN
            clr  NEWBTI
            clr  CINT
            clr  LSEG
            clr  SAW_WAVE
            clr  SIN15_WAVE
            clr  SIN7_WAVE
            clr  SQR_WAVE
            clr  EDELCOUNT

            movb #$01, t1state          ; set hub state state
            rts

t1state1:                               ; hub
            tst  CHAR_RDY               ; check if digits entered/handled
            beq  t1s1SKIP               ; skip if nothing entered
            tst  NINT_FLG               ; test if value should go into NINT
            bne  t1s1NINT               ; branch to nint handler
            ldaa KEY_BUFF
            clr  CHAR_RDY
            clr  SAW_WAVE
            clr  SIN15_WAVE
            clr  SIN7_WAVE
            clr  SQR_WAVE
            cmpa #$31                   ; wave 1 pressed
            beq  saw
            cmpa #$32                   ; wave 2 pressed
            beq  sine7
            cmpa #$33                   ; wave 3 pressed
            beq  square
            cmpa #$34                   ; wave 4 pressed
            lbeq  sine15
            tst  SAW_WAVE
            bne  enter
            tst  SIN15_WAVE
            bne  enter
            tst  SQR_WAVE
            bne  enter
            tst  SIN7_WAVE
            bne  enter
            rts
enter:      cmpa #$0A                   ; enter key pressed
            lbeq set_enter

t1s1NINT:
            ldaa KEY_BUFF
            clr  CHAR_RDY
            cmpa #$0A                   ; enter key pressed
            beq  set_enter
            movb #$0A, t1state          ; set digit handler state
            movb #$01, ACCINPUT         ; accept keypad input
t1s1SKIP:   rts
t1s1ERR:
            tst  TDB_FLG
            beq  NO_TDB
            movb #$04, t1state          ; set TDB error state
NO_TDB:
            tst  NODIG_FLG
            beq  NO_NODIG
            movb #$05, t1state          ; set no digits error state
NO_NODIG:
            tst  ZER_FLG
            beq  NO_ZER
            movb #$06, t1state          ; set zero error state
NO_ZER:
            rts
saw:
            clr  BUFFER
            clr  BUFFCOUNT
            clr  RUN
            movb #$01, SAW_FLG          ; set flag to display saw message
            movb #$01, SAW_WAVE        ; set flag for task5
            rts
sine7:
            clr  BUFFER
            clr  BUFFCOUNT
            clr  RUN
            movb #$01, SIN7_FLG        ; set flag to display sine7 message
            movb #$01, SIN7_WAVE
            rts
square:
            clr  BUFFER
            clr  BUFFCOUNT
            clr  RUN
            movb #$01, SQR_FLG         ; set flag to display square message
            movb #$01, SQR_WAVE
            rts
sine15:
            clr  BUFFER
            clr  BUFFCOUNT
            clr  RUN
            movb #$01, SIN15_FLG       ; set flag to display sine15 message
            movb #$01, SIN15_WAVE
            rts
set_enter:
            movb #$07, t1state         ; set enter state
            rts

set_bspace:
            tst  BUFFCOUNT
            beq  dontdelete
            movb #$01, t1state         ; set backspace state
dontdelete:
            rts
t1state2:
            rts
t1state3:                              ; backspace state
            ldaa BS_FLG
            cmpa #$04
            beq  t1s3done
            tst  BS_FLG
            bne  t1s3skip
            movb #$01, BS_FLG          ; set bs flag to 1, display will initiat bs process
            ldx  #BUFFER               ; load buffer address into x
            ldaa BUFFCOUNT             ; load BUFFCOUNT into A
            suba #$01                  ; subtract one from BUFFCOUNT
            ldab #$00                  ; load B with 0
            stab A,X                   ; index BUFFCOUNT - 1 past X, store value in B
            dec  BUFFCOUNT             ; decrement BUFFCOUNT
            rts
t1s3skip:
            rts
t1s3done:
            movb #$01, t1state
            clr  BS_FLG
            rts

t1state4:
            clr  BUFFCOUNT             ; TDB error state
            movb #$09, t1state         ; set delay state
            rts
t1state5:                              ; no digits entered state
            clr  BUFFCOUNT
            movb #$09, t1state         ; set delay state
            rts
t1state6:                              ; zero error state
```

```
            clr  BUFFCOUNT
            movb #$09, t1state         ; set delay state
            rts
t1state7:                              ; enter pressed state
            clr  TEMP
            clr  NINT_FLG
            clr  ACCINPUT
            ldx  #BUFFER
            clr  BIN_RES
            tst  BUFFCOUNT
            beq  NODIG_ERR
            jsr  ASC_BIN
            cmpa #$01
            beq  TDB_ERR
            cmpa #$02
            beq  ZERO_ERR
            movb #$01, t1state         ; no errors, set hub state
            movb BIN_RES, NINT         ; set new NINT
            movb #$01, RUN
            rts
NODIG_ERR:
            movb #$05, t1state
            movb #$01, NODIG_FLG       ; set flag to display no digits error
            rts
TDB_ERR:
            movb #$04, t1state
            movb #$01, TDB_FLG         ; set flag to display TDB error
            rts
ZERO_ERR:
            movb #$06, t1state
            movb #$01, ZER_FLG         ; set flag to display zero error
            rts

t1state8:
            tst  EDEL_FLG              ; Error Delay State
            bne  t1s8a
            movb #$01, t1state
            clr  BUFFCOUNT

t1s8a:      rts


t1state9:                              ; delay state
            movb #$01, EDEL_FLG        ; set flag for display to delay error emssage
            movb #$08, t1state         ; move back to hub state
            rts
t1state10:                             ; digit handler
            tst  ACCINPUT              ; see if we are still accepting input
            beq  t1s10done             ; check if buffer is still full
            ldab KEY_BUFF
            cmpb #$08
            beq  bs_press
            ldaa BUFFCOUNT
            cmpa #$03
            bhs  t1s10done             ;ensure extra values cannot be input
            movb #$01, ECHO_FLG
            ldx  #BUFFER
            ldab KEY_BUFF
            stab A, X
            inc  BUFFCOUNT
t1s10done:
            movb #$01, t1state         ; move back to hub state
            rts
bs_press:
            tst  BUFFCOUNT
            beq  no_BS
            dec  BUFFCOUNT
            movb #$03, t1state
            movb #$01, BS_FLG
            clr  ECHO_FLG
            rts
no_BS:      movb #$01, t1state
            clr  ECHO_FLG
            rts
;-------------------------------TASK_2 KEYPAD-----------------------------------------------

task_2:                                ; get current state, branch accordingly
            ldaa t2state
            beq  t2state0
            deca
            beq  t2state1
            deca
            beq  t2state2

t2state0:                              ; initialization of keypad
            jsr  INITKEY
            movb #$02, t2state         ; set next state
            rts

t2state1:
            tst  LKEY_FLG
            beq  exit_t2s1
            jsr  GETCHAR
            stab KEY_BUFF
            movb #$01, CHAR_RDY
            movb #$02, t2state
exit_t2s1:
            rts

t2state2:
            tst  CHAR_RDY              ; check CHAR_RDY flag
            bne  exit_t2s2             ; if it hasn't been cleared by M2, exit
            movb #$01, t2state         ; set next state
exit_t2s2:
            rts

;-------------------------------TASK_3 LCD DISPLAY------------------------------------------

task_3:                                ; get current state, branch accordingly
            ldaa  t3state
            lbeq  t3state0
            deca
            lbeq  t3state1
            deca
            lbeq  t3state2
            deca
            lbeq  t3state3
            deca
            lbeq  t3state4
            deca
            lbeq  t3state5
            deca
            lbeq  t3state6
            deca
            lbeq  t3state7
            deca
            lbeq  t3state8
            deca
            lbeq  t3state9
            deca
            lbeq  t3state10
            deca
            lbeq  t3state11
            deca
            lbeq  t3state12
            rts
t3state0:                                  ; initialization
            jsr  INITLCD                   ; initialize LCD
            jsr  CURSOR_ON                 ; set cursor on
            movb #$01, t3state             ; set next state
            movb #$01, SETUP               ; turn on setup flag
            clr  WAVES_FLG                 ; clear relevant task variables
            clr  SAW_FLG
            clr  SIN7_FLG
            clr  SIN15_FLG
            clr  SQR_FLG
            clr  TDB_FLG
            clr  NODIG_FLG
            clr  ZER_FLG
            clr  NODIG_FLG
            clr  DSPCOUNT
            clr  BUFFCOUNT
            clr  MESSFIN
            clr  ECHO_FLG
            clr  BUFFER
            clr  EDEL_FLG
            clr  BS_FLG

t3state1:                                  ; hub
            tst  SETUP
            beq  setupskip
            movb #$01, WAVES_FLG           ; set flag for waves base messages
            clr  SETUP                     ; only need to setup once
            tst  WAVES_FLG
            bne  DIS_WAVES_MESS
setupskip:
            tst  SAW_FLG                   ; check saw flag
            bne  DIS_SAW_MESS
            tst  SIN7_FLG                  ; check sin7 flag
            bne  DIS_SIN7_MESS
            tst  SIN15_FLG                 ; check sin15 flag
            bne  DIS_SIN15_MESS
            tst  SQR_FLG                   ; check square flag
            bne  DIS_SQUARE_MESS
            tst  TDB_FLG                   ; check TDB flag
            bne  DIS_TDB_ERR
            tst  ZER_FLG                   ; check zero entered flag
            bne  DIS_ZERO_ERR
            tst  NODIG_FLG                 ; check no digits flag
            bne  DIS_NODIG_ERR
            tst  EDEL_FLG                  ; check error delay flag
            bne  DIS_EDEL_MESS
            tst  BS_FLG                    ; check backspace flag
            bne  DIS_BS_MESS
            tst  ECHO_FLG                  ; check echo flag
            bne  DIS_ECHO_MESS
            rts

DIS_WAVES_MESS:
            movb #$02, t3state             ; set next state
            rts
DIS_SAW_MESS:
            movb #$01, DWAVE
            movb #$03, t3state             ; set next state
            rts
```

```
DIS_SIN7_MESS:
            movb #$01, DWAVE
            movb #$04, t3state             ; set next state
            rts
DIS_SIN15_MESS:
            movb #$01, DWAVE
            movb #$05, t3state             ; set next state
            rts
DIS_SQUARE_MESS:
            movb #$01, DWAVE
            movb #$06, t3state             ; set next state
            rts
DIS_TDB_ERR:
            movb #$07, t3state             ; set next state
            rts
DIS_ZERO_ERR:
            movb #$08, t3state             ; set next state
            rts
DIS_NODIG_ERR:
            movb #$09, t3state             ; set next state
            rts
DIS_EDEL_MESS:
            movb #$0C, t3state             ; set next state
            movw #$FFFF, EDELCOUNT         ; set initial delay count to 2000 m
            rts
DIS_BS_MESS:
            movb #$0A, t3state             ; set next state
            rts
DIS_ECHO_MESS:
            movb #$0B, t3state             ; set next state
            rts
t3state2:                                  ; base waves message
            ldx  #WAVES
            movb #$00, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s2done
            rts
t3s2done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, WAVES_FLG           ; reset display waves flag
            movb #$01, t3state
            rts
t3state3:                                  ; saw message
            ldx  #SAW_MESS
            movb #$40, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s3done
            rts
t3s3done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, SAW_FLG             ; reset display saw flag
            movb #$01, t3state             ; move back to hub state
            movb #$01, NINT_FLG            ; NINT ready to be set
            clr  DWAVE
            ldaa #$5A
            jsr  SETADDR                   ; set cursor to correct echo location
            rts
t3state4:                                  ; sin7 message
            ldx  #SIN7_MESS
            movb #$40, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s4done
            rts
t3s4done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, SIN7_FLG            ; reset display sin7 flag
            movb #$01, t3state             ; move back to hub state
            movb #$01, NINT_FLG            ; NINT ready to be set
            clr  DWAVE
            ldaa #$5A
            jsr  SETADDR                   ; set cursor to correct echo location
            rts

t3state5:                                  ; sin15 message
            ldx  #SIN15_MESS
            movb #$40, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s5done
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, SIN15_FLG           ; reset display sin15 flag
            rts
t3s5done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, SIN15_FLG            ; reset display sin15 flag
            movb #$01, t3state             ; move back to hub state
            movb #$01, NINT_FLG            ; NINT ready to be set
            clr  DWAVE
            ldaa #$5A
            jsr  SETADDR                   ; set cursor to correct echo location
            rts
t3state6:                                  ; square message
            ldx  #SQUARE_MESS
            movb #$40, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s6done
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, SQR_FLG             ; reset display saw flag
            rts
t3s6done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, SQR_FLG            ; reset display sin7 flag
            movb #$01, t3state             ; move back to hub state
            movb #$01, NINT_FLG            ; NINT ready to be set
            clr  DWAVE
            ldaa #$5A
            jsr  SETADDR                   ; set cursor to correct echo location
            rts
t3state7:                                  ; too big error
            ldx  #TDB
            movb #$55, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s7done
            rts
t3s7done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, TDB_FLG             ; reset display TDB error flag
            movb #$01, EDEL_FLG            ; set delay flag
            movb #$01, t3state             ; move back to hub state
            rts
t3state8:                                  ; zero error
            ldx  #ZERR
            movb #$55, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s8done

            rts
t3s8done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, ZER_FLG             ; reset display waves flag
            movb #$01, EDEL_FLG            ; set delay flag
            movb #$01, t3state             ; move back to hub state
            rts

t3state9:                                  ; no digits entered error
            ldx  #NODIG
            movb #$55, DSPSTART
            jsr  dispchar
            tst  MESSFIN
            bne  t3s9done
            rts
t3s9done:
            movb #$00, MESSFIN             ; reset message finished flag
            movb #$00, NODIG_FLG           ; reset display waves flag
            movb #$01, EDEL_FLG            ; set delay flag
            movb #$01, t3state             ; move back to hub state
            rts

t3state10:                                 ; backspace
            ldaa BS_FLG
            cmpa #$01
            bne  cp2
            ldab #$08
            jsr  OUTCHAR
            movb #$02, BS_FLG
            rts
cp2:
            cmpa #$02
            bne  cp3
            ldab #$20
            jsr  OUTCHAR
            movb #$03, BS_FLG
            rts
cp3:
            cmpa #$03
            ldab #$08
            jsr  OUTCHAR
            movb #$04, BS_FLG
            movb #$01, t3state
            clr  CHAR_RDY
            rts

t3state11:                                 ; echo
            ldx  #BUFFER                   ; load x w/ buffer
            ldaa BUFFCOUNT                 ; load a w/ buffcount
            cmpa #00
            beq  t3s11cont                 ; no keys entered yet, so continue
            cmpa #$04
            bhs  t3s11done
            suba #$01
            ldab A, X                      ; load b with the current character
            jsr  OUTCHAR                   ; echo current character
            clr  ECHO_FLG
            movb #$01, t3state
            rts
t3s11cont:
            clr  ECHO_FLG
            rts
t3s11done:
```

```
            clr  ECHO_FLG
            movb #$01, t3state             ; set hub state
            rts

t3state12:                                 ; error delay
            decw EDELCOUNT
            tstw EDELCOUNT
            beq  t3s12done
            rts
t3s12done:

            clr  ERR_FLG
            clr  EDEL_FLG
            tst  SAW_WAVE
            lbne DIS_SAW_MESS
            tst  SIN7_WAVE
            lbne DIS_SIN7_MESS
            tst  SIN15_WAVE
            lbne DIS_SIN15_MESS
            tst  SQR_WAVE
            lbne DIS_SQUARE_MESS
            rts

;-------------------------------TASK_4 TIMER CHANNEL 0------------------------------------------

task_4:     ldaa t4state                   ; get current state, branch accordingly
            beq  t4state0
            deca
            beq  t4state1
            rts

t4state0:                                  ; initialization
            movw  #1000, INTERVAL          ; set interval to 1000 ticks
            cli                            ; clear i bit
            bset  TIOS, #$01               ; set channel 0 for output compare
            bset  TCTL2, #$01              ; toggle output after successful output compare
            bset  TMSK1, #$01              ; allow flag to cause an interrupt
            bset  TFLG1, #$01              ; clears the timer flag
            bset  TSCR, #$A0               ; enables interrupts for channel 0
            movb  #$01, t4state

            ldd   TC0H
            addd  INTERVAL
            std   TC0H
            movb  #$01, t4state
            rts

t4state1:
            movb  #$01, t4state
            rts

;-------------------------------TASK_5 FUNCTION GENERATOR---------------------------------------

task_5:                                    ; function generator
            ldaa  t5state
            lbeq  t5state0
            deca
            lbeq  t5state1
            deca
            lbeq  t5state2
            deca
            lbeq  t5state3
            deca
            lbeq  t5state4
            rts
```

Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

```
t5state0:                                  ; initialization
            movb  #$01, t5state
            rts
t5state1:                                  ; waiting for wave
            tst   RUN                      ; test if it is time for a wave to be displayed
            beq   t5s1a                    ; if 0, rts
            movb  #$02, t5state            ; set next state if its time to RUN
            tst   SAW_WAVE                 ; check if display saw message
            bne   saw_disp
            tst   SQR_WAVE                  ; check if display sqr message
            bne   sqr_disp
            tst   SIN7_WAVE                 ; check if display sin7 message
            bne   sin7_disp
            tst   SIN15_WAVE                ; check if display sin15 message
            bne   sin15_disp
            rts
saw_disp:
            movw  #SAW, WAVEPTR
            movb  #$02, t5state
            rts
sqr_disp:
            movw  #SQUARE, WAVEPTR
            movb  #$02, t5state
            rts
sin7_disp:
            movw  #SIN7, WAVEPTR
            movb  #$02, t5state
            rts
sin15_disp:
           movw  #SIN15, WAVEPTR
            movb  #$02, t5state
            rts
t5s1a:
            rts

t5state2:                                   ; new wave
            tst   DWAVE                     ; wait for display of wave message
            bne   t5s2a
            ldx   WAVEPTR                   ; point to start of data for wave
            movb  0, X, CSEG                ; get number of wave segments
            movw  1, X, VALUE               ; get initial value for DAC
            movb  3, X, LSEG                ; load segment length
            movw  4, X, SEGINC              ; load segment increment
            inx                             ; inc SEGPTR to next segment
            inx
            inx
            inx
            inx
            inx
            stx   SEGPTR                    ; store incremented SEGPTR for next segment
            movb  #$01, DPRMT               ; set flag for display of NINT prompt
            movb  #$03, t5state             ; set next state
t5s2a:
            rts

t5state3:                                   ; waiting for NINT from keypad
            tst   RUN
            beq   t5s3a                     ; branch if zero
            movb  #$04, t5state             ; set next state to display wave
            rts
t5s3a:
            rts
t5state4:                                   ; display wave
                  tst   RUN
                  beq   t5s4c                     ; do not update function generator if RUN=0
                  tst   NEWBTI
                  beq   t5s4e                     ; do not update function generator if NEWBTI=0
                  dec   LSEG                      ; decrement segment length counter
                  bne   t5s4b                     ; if not at end, simply update DAC output
                  dec   CSEG                      ; if at end, decrement segment counter
                  bne   t5s4a                     ; if not last segment, skip reinit of wave
                  ldx   WAVEPTR                   ; point to start of data for wave
                  movb  0,X, CSEG                 ; get number of wave segments
                  inx                             ; inc SEGPTR to start of first segment
                  inx
                  inx
                  stx   SEGPTR                    ; store incremented SEGPTR
t5s4a:      ldx   SEGPTR                    ; point to start of new segment
                  movb  0,X, LSEG                 ; initialize segment length counter
                  movw  1,X, SEGINC               ; load segment increment
                  inx                             ; inc SEGPTR to next segment
                  inx
                  inx
                  stx   SEGPTR                    ; store incremented SEGPTR
t5s4b:      ldd   VALUE                     ; get current DAC input value
                  addd  SEGINC                    ; add SEGINC to current DAC input value
                  std   VALUE                     ; store incremented DAC input value
                  bra   t5s4d
t5s4c:      movb  #$01, t5state             ; set next state
t5s4d:      clr   NEWBTI
t5s4e:      rts
;-----------------------------------MISC FUNCTIONS---------------------------------------------;
ASC_BIN:

CLP:
            ldab  BIN_RES                  ; load RESULT into a
            ldaa  #$0A                     ; load 10 into b
            mul                            ; whats in a, mult by b, store in D
            tsta
            bne   LTDB                     ; branch if too big
            stab  BIN_RES                  ; store D in result
            ldaa  TEMP                     ; load TEMP into a
            ldab  A, X                     ; load into B, A + X
            subb  #$30                     ; subtract $30 from D
            ldaa  #$00                     ; remove temp from mathematics
            addb  BIN_RES                  ; add BIN_RES
            bcs   LTDB                     ; check carry flag and branch if necessary
            stab  BIN_RES                  ; store a in RESULT
            inc   TEMP                     ; increment TEMP
            dec   BUFFCOUNT                ; decrement count
            bne   CLP                      ; branch to CLP and repeat if COUNT is not 0
            cmpb  #$00                     ; compare b (holds result) to 0
            beq   LZERO                    ; if 0 , branch to error
            clr   TEMP                     ; clear temp for next time
            ldaa  #$00                     ; clear any possible errenous a value
            rts

LTDB:
            ldaa  #$01
            clr   TEMP                    ; clear temp for next time
            rts                            ; rts to main

LZERO:
            ldaa  #$02
            clr   TEMP                    ; clear temp for next time
            rts                            ; rts to main

dispchar:                                  ; code to display a message cooperatively
            ldaa  DSPCOUNT                 ; load A with DSPCOUNT
            tsta
            bne   skip                      ; skip set start on all counts but first
            ldaa  DSPSTART                  ; load a with starting point
            jsr   SETADDR                   ; set starting point
skip:
            ldaa  DSPCOUNT                  ; load accumulator A with DSPCOUNT
            ldab  A,X                       ; load accumulator B with X + DSPCOUNT
            tstb                            ; test B for ascii null
            beq   done                      ; branch to done if equal to 0
            jsr   OUTCHAR
            inc   DSPCOUNT                  ; inc COUNT
            rts
done:
            movb #$01, MESSFIN             ;set MESSFIN to 1
            movb #$01, t3state             ;hub state on next loop
            clr  DSPCOUNT
            rts

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                                        ;
;  This interrupt service routine for timer channel 0 output compare interrupts TC0     ;
;  to generate a new interrupt request every INTERVAL clock counts.                      ;
;                                                                                        ;
;  Every NINT calls this ISR which sends the current VALUE to the DAC, sets the NEWBTI  ;
;  flag to signify the start of a new BTI, and CINT is reset to NINT to be counter down ;
;  for the next BTI                                                                      ;
;                                                                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TC0ISR:
            tst   RUN                        ; check if function generator is running
            beq   NOT_YET                    ; if not, prepare next interrupt
            dec   CINT                       ; BIT completion check
            bne   NOT_YET
            ldd   VALUE                      ; get updated DACA input
            jsr   OUTDACA                    ; update DACA output
            movb  NINT, CINT                 ; reinitialize interrupt counter for new BTI
            movb  #$01, NEWBTI               ; set flag indicating beginning of a new BTI
NOT_YET:
            ldd   TC0H                       ; load d w/ current timer
            addd  INTERVAL                   ; add interval to d
            std   TC0                        ; store result back in TC0H
            bset  TFLG1, #$01                ; clear timer flag
            rti

;/------------------------------------------------------------------------------------\
;| ASCII Messages and Constant Data                                                    |
;\------------------------------------------------------------------------------------/
; Any constants can be defined here

WAVES         DC.B  "1: SAW, 2: SINE-7, 3: SQUARE, 4: SINE-15",$00   ; wave selection
SAW_MESS      DC.B  "SAWTOOTH WAVE        NINT:     [1-->255]", $00 ; sawtooth message
SIN7_MESS     DC.B  "7-SEGMENT SINE WAVE  NINT:     [1-->255]", $00 ; sine-7 message
SIN15_MESS    DC.B  "15-SEGMENT SINE WAVE NINT:     [1-->255]", $00 ; sine-15 message
SQUARE_MESS   DC.B  "SQUARE WAVE          NINT:     [1-->255]", $00 ; square message
TDB           DC.B  "MAGNITUDE TOO LARGE",$00                        ; TDB error message
NODIG         DC.B  "NO DIGITS ENTERED  ",$00                          ; no digit error message
ZERR          DC.B  "INVALID MAGNITUDE  ",$00                          ; zero error message

SIN15
              DC.B                 15         ; number of segments for SINE15
              DC.W                 2048       ; initial DAC input value
              DC.B                 10         ; length for segment_1
              DC.W                 41         ; increment for segment_1
              DC.B                 21         ; length for segment_2
              DC.W                 37         ; increment for segment_2
              DC.B                 21         ; length for segment_3
              DC.W                 25         ; increment for segment_3
              DC.B                 21         ; length for segment_4
              DC.W                 9          ; increment for segment_4
              DC.B                 21         ; length for segment_5
              DC.W                 -9         ; increment for segment_5
              DC.B                 21         ; length for segment_6
              DC.W                 -25        ; increment for segment_6
              DC.B                 21         ; length for segment_7
              DC.W                 -37        ; increment for segment_7
              DC.B                 20         ; length for segment_8
              DC.W                 -41        ; increment for segment_8
              DC.B                 21         ; length for segment_9
              DC.W                 -37        ; increment for segment_9
              DC.B                 21         ; length for segment_10
              DC.W                 -25        ; increment for segment_10
              DC.B                 21         ; length for segment_11
              DC.W                 -9         ; increment for segment_11
              DC.B                 21         ; length for segment_12
              DC.W                 9          ; increment for segment_12
              DC.B                 21         ; length for segment_13
              DC.W                 25         ; increment for segment_13
              DC.B                 21         ; length for segment_14
              DC.W                 37         ; increment for segment_14
              DC.B                 10         ; length for segment_15
              DC.W                 41         ; increment for segment_15

SIN7
              DC.B                 7          ; number of segments for SIN7
              DC.W                 2048       ; initial DAC input value
              DC.B                 25         ; length for segment_1
              DC.W                 33         ; increment for segment_1
              DC.B                 50         ; length for segment_2
              DC.W                 8          ; increment for segment_2
              DC.B                 50         ; length for segment_3
              DC.W                 -8         ; increment for segment_3
              DC.B                 50         ; length for segment_4
              DC.W                 -33        ; increment for segment_4
              DC.B                 50         ; length for segment_5
              DC.W                 -8         ; increment for segment_5
              DC.B                 50         ; length for segment_6
              DC.W                 8          ; increment for segment_6
              DC.B                 25         ; length for segment_7
              DC.W                 33         ; increment for segment_7

SQUARE
              DC.B                 4          ; number of segments for SQUARE
              DC.W                 0          ; initial DAC input value
              DC.B                 9          ; length for segment_1
              DC.W                 0          ; increment for segment_1
              DC.B                 1          ; length for segment_2
              DC.W                 3277       ; increment for segment_2
              DC.B                 9          ; length for segment_3
              DC.W                 0          ; increment for segment_3
              DC.B                 1          ; length for segment_4
              DC.W                 -3277      ; increment for segment_4
SAW
              DC.B                 2          ; number of segments for SAW
              DC.W                 173        ; initial DAC input value
              DC.B                 19         ; length for segment_1
              DC.W                 173        ; increment for segment_1
              DC.B                 1          ; length for segment_2
              DC.W                 -3287      ; increment for segment_2
```

Noah Tanner, Cole Sterba
Dr. Murray
ME-305-01
May 24th, 2023

```
;/------------------------------------------------------------------------------------\
;| Vectors                                                                            |
;\------------------------------------------------------------------------------------/
; Add interrupt and reset vectors here

        ORG   $FFFE                        ; reset vector address
        DC.W  Entry
        ORG   $FFCE                        ; Key Wakeup interrupt vector address [Port J]
        DC.W  ISR_KEYPAD
        ORG   $FFEE                        ; vector address for timer channel 0
        DC.W  TC0ISR                       ; interrupt service routine name
```