Pie Tin & Sea Cow

John Juilfs & Mark Schmidt

EML5666
Final Report
December 8, 1999

Introduction

Though the human race does not live on the ocean, we do spend a significant amount of time there. From fishermen to romantic cruises to destroyers, the surface of the ocean has certainly been used by people for thousands of years. However, the ocean offers some problems. Covering two thirds of the Earth, the ocean is simply too large to be fully understood, at least with today's technology. In some places, pilots are still used to steer ships through a particular harbor because the captain does not know where all the shallows are. Currents, eddies, hurricanes, and tides tend to make the ocean a dynamic system, not static. Because of our lack of understanding and exploration, ships still hit icebergs and people are lost at sea.

It was our goal to begin exploring the possibilities of nautical robots. We wanted to create versatile boats that could potentially work together on the water. There are many possible ultimate uses of this. They could perform a coordinated search for a life boat or a rapid mapping of a harbor bottom. Of course, the military could certainly find a use.

Platform

Sea Cow

The platforms chosen for each of the two boats were quite different. Sea Cow was based on a catamaran. It has two identical hulls joined by an arch. The hulls are fairly hydrodynamic, and there is a motor in each hull. Each hull has three segments, the bow, midship, and stern. The bow is covered with balsa wood. The arch covers the midship and the stern. Inside the arch are the circuits controlling the robot. The part of the arch that is close to the water is solid, and everything is attached to that. The top portion of the arch is removable to parts can be accessed easily. With a little effort in sealing joints, Sea Cow could likely be operated in the rain.

Pie Tin

Pie Tin is far less hydrodynamic. On the top, it is an large octagon, and on the bottom it is a smaller square. There is a motor on each side near the bottom. The circuit boards fit into the boat on frames that are layered. On the bottom between the motors are the batteries. Above that are the sonar boards and a motor driver board. On the top layer are a motor driver board, the TJPro board, and the port expansion board. There were two reasons for this odd design. The first is so that the two could dock together possibly, and Pie Tin could fit under the arch of Sea Cow. With the symmetrical design, Sea Cow could approach Pie Tin from any direction. The second reason was simply for experimentation, to discover if, on such a small scale, the shape would make a significant difference. In the end, the two robots move at about the same speed, though Pie Tin is faster and has a much greater turning rate.

Design

From the initial design, the finished product was far away. Sea Cow was originally designed in 3-D on AutoCad, and then each segment was converted to 2-D. Pie Tin, because of it’s simple geometry, was never drawn in 3-D. From the AutoCad files, the parts were cut out on the T-tech. The parts formed a wooden structure of each boat. The wooden structure of Sea Cow consisted of the side panels and 3 supports for each hull. Pie Tin was only the walls because its shape was strong enough. However, Pie Tin did have some awkward angles that caused a problem because the wood had width. The solution was to sand the edges of the tabs at about 45 degrees. That allows for a tight fit. Each of us constructed our own wooden frames, but from there Mark took over with the toxic chemicals. First the outside of the boats was sanded until smooth. Then a layer of Bondo fiberglass was applied. Sea Cow was done first, and its first layer of fiberglass was not done very well, so another coat was applied.

After the fiberglass, the nozzles were inserted. This involved drilling holes in the appropriate places, filing them to fit the PVC pipe, and then attaching 1.5 - 2 inches of pipe to the hull. The most difficult was the holes in Sea Cow’s bows because of the awkward angle. The nozzles were then sealed with Goop.

Sea Cow then received a layer of Bondo UV Activated Body Filler. This did not work very well, either, but it did seal the cracks well enough. Pie Tin, however, was coated with Bondo red spot putty. This worked much better. It dried an a suitable rate and was not difficult to work with. Both of these surfaces were wet sanded, which was actually rather difficult, and then primed. John then painted Pie Tin, and Mark painted Sea Cow. The parts that would contact water were painted with spray enamel.

The motors were then fit into the boats. Flexible tube (5/8” OD, 1/2” ID) fit inside the PVC pipe and then joined to the motor with a coupling. The coupling was held to the motor and the flexible tube with JBQuick. The tube inside the PVC did not need a seal, and so the motors could be removed. However, Pie Tin insisted on leaking, and the entire tube area was covered with hot glue, so removing the motors there would be very difficult.
Sea Cow’s hulls were then attached to the arch. The fit was not perfect because of assembly stresses. However, the parts were close enough to allow glue to hold them in place.

The switches and computer ports were rather simple. The circuit boards were very easy to mount in Sea Cow, as there was plenty of open space. Pie Tin posed difficult. However, by tweaking the length of the braces, each layer could be put at the correct height.

Of the parts mentioned, they are all available in town. Lowes, Walmart, and automotive places carry Bondo fiberglass. Automotive supply stores carry Bondo spot putty and body filler. Walmart has the JBQuick and paint.

The propulsion for the robots is, at a glance, very simple. The motors are Graupner Bow Thrusters, available from Hobby-Lobby for $35. They are easily sealed, and much simpler than propellers. Having them set up in their particular configuration, they steer similarly to a two-wheeled robot. Reversing one motor will turn the boat in the direction of that motor. However, linear and rotational momentum make it very different from land robots, and are the real hidden problems. Once a boat turns, it tends to keep turning, particularly Pie Tin. Sea Cow will attempt to turn before hitting a wall, but if at full speed, it has difficulty slowing down enough before hitting the wall. These are problems that have been mostly fixed in software.

Sensors

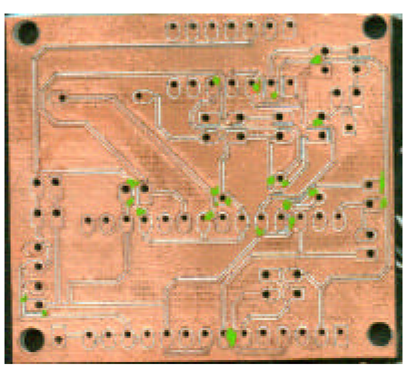
Overview

The primary sensors for Pie Tin and Sea Cow are sonar. Each robot has a bank of three sonar, one facing directly forward, and the other two at an angle to the left and right. The hope was that the sonar configured this way would be able to spot most large obstacles (such as walls) and avoid them. The sonar receiver used was the standard IMDL lab board printed out from the T-Tech, while the sonar emitter was the amplifier from the Toshiba manual that uses a single inverter chip to drive the transducer. Each sonar emitter / receiver pair was multiplexed / demultiplexed on a custom expansion board. This expansion board used port 0x6000 to select which emitter / receiver pair was active, as well as to choose the direction for the motors (this worked fine on Pie Tin, but proved difficult on Sea Cow: that is covered below).

The secondary sensor used was radio. Pie Tin emits radio (using Timer Output Compare 5 on Port A bit 3) and Sea Cow receives (it keeps track of the pulses with the pulse accumulator, and so is tied to Port A bit 7). The receiver could have also been attached to input capture, but at the time the software was written it was uncertain as to whether extraneous noise could filter in (not really, as it turned out), and the presence or absence of a large number of pulses could be more easily verified than whether a particular edge was accurate or not.

Sonar assembly and hardware:

Ten sonar boards were assembled: seven proved usable. The pre-designed sonar board is useful, but difficult. The T-tech machine has problems. The majority of the board is conductive, and only a small bit of removed copper allows for anything resembling a circuit. The true villains are "threads". The threads are bits of copper that were not completely cut away by the T-tech and cross over between traces and pads. These are best removed with a razor blade and a magnifying glass. The places where one is most likely to see a thread have been marked with green in this picture of a sonar board.



Sonar testing and software:

Since the board was only driving one sonar (due to the expansion board), Input Capture 1 was used to receive data (Port A bit 2). The 40kHz signal at port 0x7000 was used to turn the emitters on and off. Since the signal goes between a signal and high impedance, a pull down resistor was necessary to ensure that the signal was transmitted to the emitter: at first, the on board 330Ω resistor was used, but the current draw proved excessive for the board, and whenever the sonar emitter was turned on, the value kept in the latch attached to port 0x6000 (on the expansion board) would become corrupt, as the values of the data pins and Y4 would fluctuate at random. This was solved by using a 10kΩ resistor instead of the 330Ω one. This solution did not function for Sea Cow, so instead all of the sonar emitters were tied to the same enable pin, thus bypassing the necessity of using port 0x6000 to select the correct sonar emitter (thus making Sea Cow ping in all directions at once and listen at one, as during the listening stage the emitters were necessarily off, so the value at port 0x6000 was never corrupted).

The included sonar test assembler file “sonart.asm” will write port 0x7000 with all ones to turn on the 40kHz sonar, then wait 1ms, then turn it off by writing all zeroes. It will then wait another 1ms, then begin listening for sonar by polling TFLG1 bit 2. After this, it will either time out or receiver a sonar signal, and will print either the elapsed Eclocks or a message stating that it has timed out.

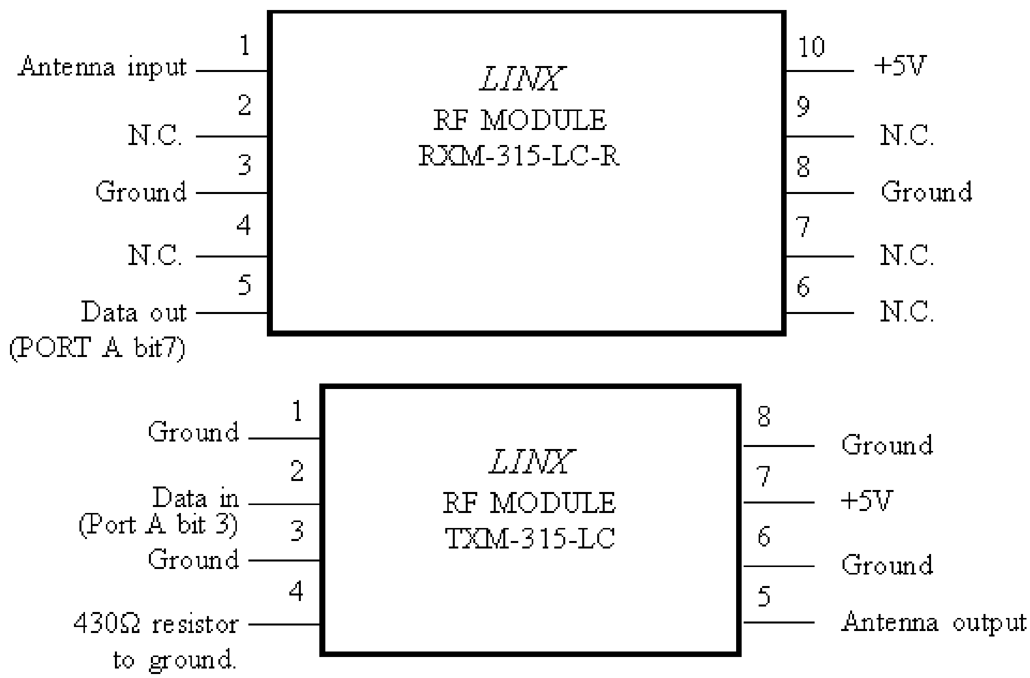
Extensive data was gathered on the properties of the sonar. The general conclusion of this data showed that the sonar had a hard time getting a receiving ping off of a surface with a face even slightly less than normal to the emitter, while receiving pings off of a surface normal to the sonar emitter but less than directly in front of it was not as difficult.

Radio assembly and hardware:

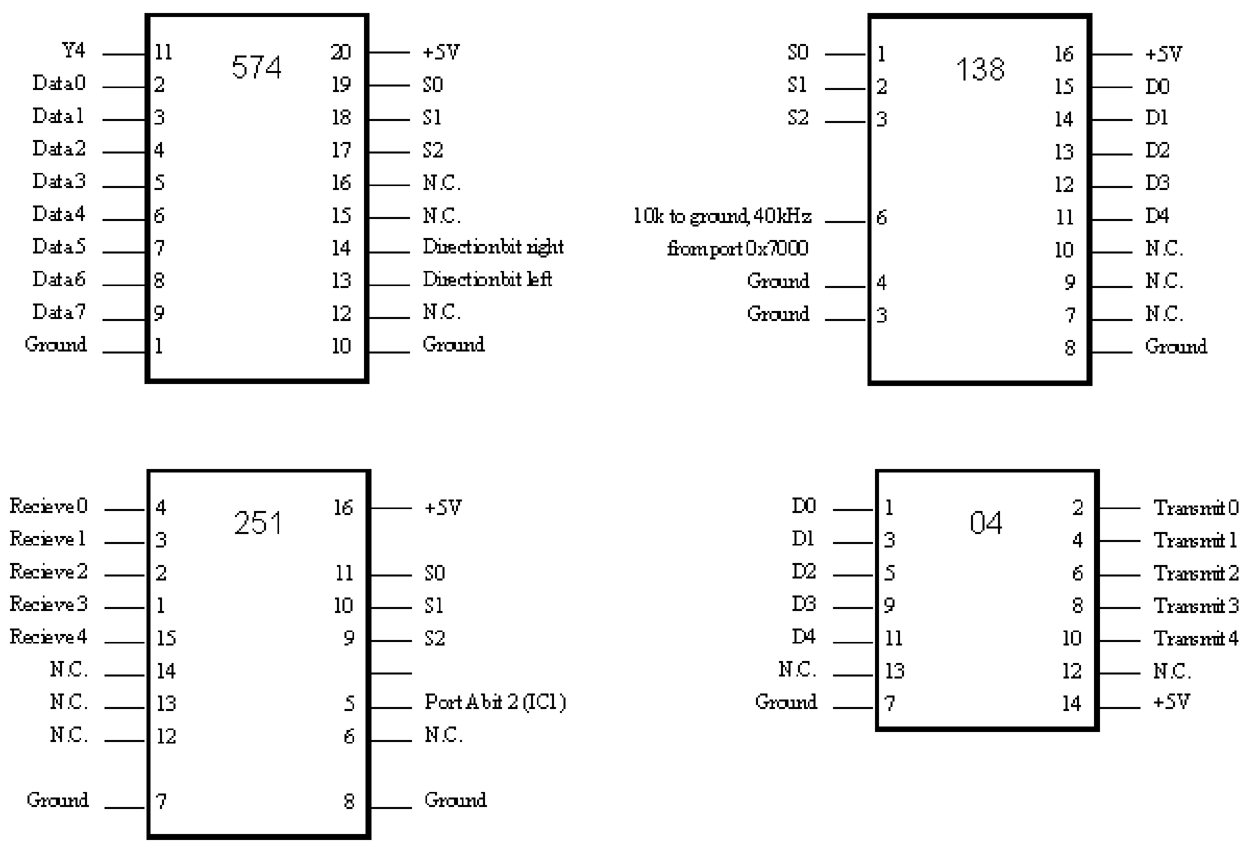
Since the radio used was the emitter and receiver pair by Linx Technologies (models RXM-315-LC-R and TXM-315-LC), the only assembly was tying the appropriate pins to ground, power, or data, as well as ensuring that the antenna had a good ground plane.

Radio testing and software:

The radio worked perfectly, not picking up any stray signals. The software involved was also simple, either oscillating TOC5 to indicate a signal or remaining quiet on the transmitter end, and periodically checking on the pulse accumulator (once every TOI) on the receiving end.



(Above) Radio Transmitter and Reciever
(Below) Expansion board for sonar emitter / reciever pairs



Code / Behaviors

Overview

Both Pie Tin and Sea Cow are programmed in assembler, and run routines that are very similar or identical. Since much of the hardware is the same (the motors and the sonar are handled very similar on both), the routines that run these execute the same code on both machines, only doing special cases where a difference is unavoidable. The original code was interrupt driven, but the need for this was lessened when the number of timing specific actions was greatly reduced when LED communications was no longer a goal. The only interrupts currently used are TOI and TOC5.

The three main behaviors are common to both Pie Tin and Sea Cow, though the methods used to achieve them are different in some cases..

Main loop

The core code does initializations, looks to see if it can see other sonar (if it can it delays 150 ms to hopefully be out of phase with its own sonar: sonar emitting takes around 60 ms per sonar, and there is a delay of 300 ms during which the other boat has time to emit and receive sonar without interference), then falls into a loop of sending and receiving pings, making obstacle avoid decisions on those values, waiting so the other boat has time to ping sonar, and then looping back.

Interrupts

The Timer Overflow Interrupt sets up the width of the pulses sent to the motors (turned on with TOC1 and turned off with TOC2 and TOC3), and sets the direction pins (Pie Tin has the direction pins accessible on port 0x6000 bits 6 and 5: Sea Cow uses port A bits 4 and 3). It also keeps track of time, incrementing a 16 bit counter every time: since this won't wrap around until long after the motor batteries are drained, this is plenty of time (a little over half an hour). Additionally, Sea Cow looks at the pulse accumulator to see if Pie Tin is sending a message.

The Timer Output Compare 5 interrupt mostly serves to set itself to occur again: Port A bit 3 on Pie Tin serves as the radio output pin, and is either set to toggle or is simply left low to indicate the absence of signal.

Obstacle Avoidance

The first behavior, obstacle avoidance, just involves looking at the last set of pings and requesting a set of motor values that corresponds to the desired direction. There are two separate

“Avoid” functions, one for each robot, each of which has different motor values. Since Pie Tin’s motors are rigged “backwards” for symmetry about the y-axis (so they could fit), the values requested are not symmetrical: this does not solve the problem, but lessens it. Sea Cow’s motors could be fitted facing the same direction due to plentiful space, and so the values passed to its motors are the same: however, there is a more detailed turning section, as Sea Cow has much greater momentum than Pie Tin, and has more difficulty slowing down.

Sonar Noninterference

This behavior basically consists of a long delay on boot up while looking through each emitter in turn to see if any sonar is present. If sonar is seen, the boat seeing sonar waits for 150 ms before entering its main loop: if not, it simply enters after it has decided that no sonar is available to see. Since cycling through the sonars takes around 150 ms, and each robot waits 300 ms after using all three emitter / receiver pairs, this delay should put the second robot’s sonar scanning squarely in the middle of the first robot’s down time, assuming it can see the other sonar at boot up. This is poorly tested, as the odds of one robot’s sonar interfering negatively with the other is very low, due to the fact that the sonar would have to perfectly bounce off of a wall onto the other robot’s sensor and trigger an incorrect turn: after all, if the sonar hits the other robot’s receiver directly, all that will result is the second robot turning away from the first: effectively, obstacle avoidance with a actively pinging obstacle.

Spin Time

A time after bootup, Pie Tin will signal that “it is time to spin” using the radio. This time could have been determined quasi-randomly, but for testing purposes has been chosen at around 20 seconds. Upon signalling, it will assume that Sea Cow has received the message and begin spinning to the right. A while later, it will spin in the opposite direction. After it completes spinning, it will resume obstacle avoidance. Sea Cow will perform the same actions upon receiving the signal. The point was to demonstrate that the signal has been received and acted upon, as the act of spinning has no obvious practical value.

```
*Sonart.asm
        ORG     $FFFE   ;reset vector
        FDB     Main    ;point to main on reset


BAUD    EQU     $102B  ; BAUD rate control register to set the BAUD rate
SCCR1   EQU     $102C  ; Serial Communication Control Register-1
SCCR2   EQU     $102D  ; Serial Communication Control Register-2
SCSR    EQU     $102E  ; Serial Communication Status Register
SCDR    EQU     $102F  ; Serial Communication Data Register

PORTA   EQU     $1000   ;PORT A REGISTER
TCNT    EQU     $100e   ;Timer count
TFLG2  EQU    $1025
TFLG1   EQU     $1023
TIC1    EQU     $1010
TCTL2   EQU     $1021


EOS     EQU     $04    ; User-defined End Of String (EOS) character
CR      EQU     $0D    ; Carriage Return Character
LF      EQU     $0A    ; Line Feed Character
ESC     EQU     $1B    ; Escape Charracter



        ORG     $8000 ;start code at convienient place

bcount  RMB     1      ;the “big” counter:  incremented
*                      ;if TCNT wraps while waiting
first   RMB     2
tof     RMB     2
Msg     FCC     ‘Ready ‘
        FCB     EOS
Enter   FCB     CR,LF,EOS
TimeMSG FCC     ‘Time out’
        FCB     EOS


Main    lds     #$cfff
        ldd     #0
        ldx     #0
        ldy     #0

        ldaa    #20
        staa    TCTL2

        jsr     InitSCI

        ldaa    #$0             *turn emitter off at first
        staa    $7000
        ldx     #Msg            *output message
        jsr     OutStr
        ldaa    #250            *net total delay = 278250E = .14 seconds
dloop   ldab    #220            *[2]
inl     decb                    *[2]  [5]*220 = 1100
        bne     inl             *[3]
        brn     inl             *[3]
        deca                    *[2]
        bne     dloop           *[3]


        ldaa    #$FF          ;turn emitter on
        staa    $7000


        ldaa    #200            ;wait 1 ms
msla    nop                     ;[2]
        brn     msla            ;[3]
        deca                    ;[2]
        bne     msla            ;[3]
```

```
        ldaa    #$00          ;turn emitter off
        staa    $7000

        ldaa    #200            ;wait 1 ms
mslb    nop                     ;[2]
        brn      mslb           ;[3]
        deca                    ;[2]
        bne      mslb           ;[3]


        ldaa    #$04
        staa    TFLG1           ;clear any pending ICs
        ldaa    #$80
        staa    TFLG2           ;clear timer overflow
        ldd     TCNT            ;get current value of TCNT
        std     first
        clra
        staa    bcount          ;start at 0

inloop  ldaa    TFLG2         ;count up, while waiting for
        anda    #$80          ;response.
        beq     NoProblem

        staa    TFLG2           ;if TFLG goes high, acknowledge
        ldaa    bcount
        inca
        staa    bcount
        cmpa    #2              ;if we have had to TOIs, quit.
        beq     breakout

NoProblem

        ldaa    TFLG1
        anda    #$4
        beq     inloop
breakout
        ldd     TIC1
        subd    first
        std     tof

        ldaa    bcount
        cmpa    #2
        bne     NoTO            ;if 2, then timed out.
        ldx     #TimeMSG
        jsr     OutStr
        bra     DonePrint       ;skip ahead to carriage return
NoTO

*You will notice an apparently arbitrary "addd #$20" below this.  This is
*because sometimes a ping will be seen right away, or very near that
*(immediate response).  Usually this number is very low, such as 0x56.
*However, sometimes this number is very high (negative) due to inaccuracies
*reading TCNT.  Because of this, I add a small, arbitrary constant to the
*time of flight so that all immediate responses look about the same.

        ldd     tof             ;print out time of flight
        addd    #$20
        jsr     Hexph           ;print out high 4 bits of a
        jsr     Hexpl           ;print out low 4 bits of a
        tba
        jsr     Hexph           ;print out high 4 bits of a
        jsr     Hexpl           ;print out low 4 bits of a


DonePrint
        ldx     #Enter
        jsr     OutStr

        jmp     Main
```

```
**********************************************************************
*                       SUBROUTINE - InitSCI
* Description: This subroutine initializes the BAUD rate to 9600 and
*              sets up the SCI port for 1 start bit, 8 data bits and
*              1 stop bit.  It also enables the transmitter and receiver.
*              Effected registers are BAUD, SCCR1, and SCCR2.
* Input         : None.
* Output        : Initializes SCI.
* Destroys      : None.
* Calls         : None.
**********************************************************************
*
InitSCI PSHA                            * Save contents of A register

        ldaa #$30                       * Set BAUD rate to 9600
        staa BAUD
        ldaa #$0                        * Set SCI Mode to 1 start bit,
        staa SCCR1                      *     8 data bits, and 1 stop bit.
        ldaa SCCR2                      * Enable SCI Transmitter
        ora  #$0c                       *     and Receiver
        staa SCCR2

        PULA                            * Restore A register
        RTS                             * Return from subtoutine
```

```
**********************************************************************
*                       SUBROUTINES -  Hexph and Hexpl
* Description: Outputs the hex digit in high or low a after
*              checking if the Transmitter Data Register is Empty
* Input         : Data to be transmitted in register A.
* Output        : Transmit the data.
* Destroys      : None.
* Calls         : None.
**********************************************************************
*
Hexph   psha            ;save a away
        lsra
        lsra
        lsra
        lsra
        bra     hexs    ;after scaled, go to start
Hexpl   psha
hexs    anda    #$0F
        adda    #48     ;scale to ASCII 0
        cmpa    #58     ;if above or equal, need more scaling
        blt     hexl    ;jump over correction if OK
        adda    #7      ;now will print A-F properly
hexl    jsr     OutChar ;print out character
        pula            ;get a back
        rts             ;bye bye
```

```
**********************************************************************
*                       SUBROUTINE -  OutStr
* Description: Outputs the string terminated by EOS. The starting location
*              of the string is pointed by X register. Calls the OutChar
```

```
*               subroutine to display a character on the screen and
*               exit once EOS has been reached.
* Input          : Starting location of the string to be transmitted
*                : (passed in X register)
* Output         : Prints the string.
* Destroys       : contents of X register.
* Calls          : OutChar.
************************************************************************
*
OutStr  PSHA                        * Save contents of A register
Loop2   ldaa    0,x                 * Get a character (put in A register)
        cmpa    #EOS                * Check if it's EOS
        beq     Done                * Branch to Done if it's EOS

        JSR     OutChar             * Print the character by calling OutChar
        inx                         * Point to next character
        BRA     Loop2               * Branch to Loop2 for the next char.
Done    PULA                        * Restore A register
        RTS                         * Return from subtoutine



************************************************************************
*                SUBROUTINE  -  InChar
* Description: Receives the typed character into register A.
* Input          : None
* Output         : Register A = input from SCI
* Destroys       : Contents of Register A
* Calls          : None.
************************************************************************
*
InChar

pollrecv          ldaa SCSR             ; Check status reg.
                  anda #$20             ;     (load it into A reg)
                 cmpa #0              ; Check if receive buffer full
                 beq pollrecv         ; Wait until data present

                  ldaa SCDR             ; SCI data ==> A register
                 RTS                   ; Return from subroutine


************************************************************************
*                       SUBROUTINE -  OutChar
* Description: Outputs the character in register A to the screen after
*               checking if the Transmitter Data Register is Empty
* Input          : Data to be transmitted in register A.
* Output         : Transmit the data.
* Destroys       : None.
* Calls          : None.
************************************************************************
*
OutChar PSHB                         * Save contents of B register
Loop1   ldab    SCSR                 * Check status reg (load it into B reg)
        andb    #$80                 * Check if transmit buffer is empty
        BEQ     Loop1                * Wait until empty
        staa    SCDR                 * A register ==> SCI data
        PULB                         * Restore B register
        RTS                          * Return from subtoutine
```

```
*This is the main program, written for Pie Tin:  since all of the behaviors
```

```
*for both robots are here, it can be changed to Sea Cow’s code by changing
*the robot byte from 0 to 1 (it is FCBed as 0, and is a constant)

        ORG     $FFFE   ;reset vector
        FDB     Main    ;point to main on reset
        ORG     $FFDE   ;timer overflow vector
        FDB     TOIISR
        ORG     $FFE0
        FDB     TOC5ISR ;point to TOC5

BAUD    EQU     $102B  ; BAUD rate control register to set the BAUD rate
SCCR1   EQU     $102C  ; Serial Communication Control Register-1
SCCR2   EQU     $102D  ; Serial Communication Control Register-2
SCSR    EQU     $102E  ; Serial Communication Status Register
SCDR    EQU     $102F  ; Serial Communication Data Register

PORTA   EQU     $1000   ;PORT A REGISTER
TCNT    EQU     $100e   ;Timer count
TFLG2  EQU     $1025
TFLG1   EQU     $1023
TIC1    EQU     $1010
TCTL1   EQU     $1020
TCTL2   EQU     $1021
TOC1    EQU     $1016
TOC2    EQU     $1018
TOC3    EQU     $101A
TOC4    EQU     $101C
TOC5    EQU     $101E
TMSK1   EQU     $1022
TMSK2   EQU     $1024
OC1M    EQU     $100C
OC1D    EQU     $100D
PACTL   EQU     $1026
PACNT   EQU     $1027


EOS     EQU     $04    ; User-defined End Of String (EOS) character
CR      EQU     $0D    ; Carriage Return Character
LF      EQU     $0A    ; Line Feed Character
ESC     EQU     $1B    ; Escape Charracter
SPC     EQU     $20    ; Space


        ORG     $8000  ;start code at convienient place

bcount  RMB     1      ;the “big” counter:  incremented
*                      ;if TCNT wraps while waiting
first   RMB     10
tof     RMB     10



TOIFLG  RMB     1
mcount  RMB     2

mlwant  RMB     1
mrwant  RMB     1
mlcurr  RMB     1
mrcurr  RMB     1
paval   RMB     1
p6val   RMB     1
cursonar RMB    1
click   RMB     1
trans   RMB     1
spin    RMB     1
robot   FCB     0

Msg
        FCC     ‘Ready ‘
        FCB     EOS
Enter   FCB     CR,LF,EOS
```

```
Timeout
        FCC     '*TO*'
        FCB     EOS
Click   FCC     ' CL '
        FCB     EOS

gotablel
        FDB     $0100,$1000,$2000,$3000,$4000,$5000,$6000,$7000,
        FDB     $8000,$9000,$A000,$B000,$C000,$D000,$E000,$F800

gotabler
        FDB     $0100,$1000,$2000,$3000,$4000,$5000,$6000,$7000,
        FDB     $8000,$9000,$A000,$B000,$C000,$D000,$E000,$F800


***************************
******** Main *************
***************************
Main    lds     #$cfff
        ldd     #0
        ldx     #0
        ldy     #0

        clrb
        clra
        staa    spin
        staa    goingS
        std     StartSTime
        std     TimeSoFar
        staa    click
        staa    trans
        staa    PORTA
        staa    paval
        staa    $7000           *sonar starts off
        staa    $6000           *point to sonar 0 to start
        staa    p6val
        staa    cursonar        *start at sonar 0

        ldaa    #$48            *PORTA bit 3 needed as output for Sea Cow
        staa    PACTL           *And PortA bit 7 is input (0).

        ldaa    #$20            *capture sonar on falling edges
        staa    TCTL2

        ldaa    #$A0            *on TOC2 or TOC3, zero respective pin
        staa    TCTL1

        ldd     #0
        std     TOC1
        incb
        std     TOC2
        std     TOC3

        clra
        staa    OC1D
        ldaa    #$60
        staa    OC1M

        ldaa    #$10
        staa    mlcurr
        staa    mrcurr
        ldaa    #$10
        staa    mlwant
        staa    mrwant

        jsr     InitSCI
        clra                    *zero timer
        clrb
        std     mcount          *clear master count
        staa    PORTA           *start out at zero
```

```
        staa    paval



        ldaa    #$80
        staa    TMSK2
        cli                     *start interrupts

        jsr     Rinit

        ldaa    #$0         ;turn emitter off at first
        staa    $7000
        jsr     IsPing
Mloop

         ldx     #Msg
         jsr     OutStr

        jsr     Bdelay
        jsr     Ping
        jsr     SpinMonSpin     *See if we will be spinning.

        ldaa    spin
        bne     Mloop
        jsr     Avoid

*        jsr     TestAct
        bra     Mloop

***************************
******* Bdelay ************
***************************

*Sonar pings take around 150ms for all 3 to fire off.
*So we will delay for 300 ms each time so that if IsPing sees a
*sonar ping and delays for 150 ms, it will hopefully be inside this
*area and interfere less.

*Net delay is 601680E = 300ms
Bdelay  ldaa    #240
dloop   ldab    #250            *[2]
inl     decb                    *[2]  [10]
        brn     inl             *[3]
        nop                     *[2]
        bne     inl             *[3]
        deca                    *[2]
        bne     dloop           *[3]
        rts


***************************
*******SpinMonSpin*********
***************************

SpinMonSpin
        ldaa    robot           *Sea Cow is told when to spin
        bne     NoDecision

        ldd     mcount
        cpd     #800
        blt     NoDecision
        cpd     #900
        bgt     NoDecision
*Now we are Pie Tin, and want to signal a spin cycle...

        ldaa    #1
        staa    spin

        ldaa    TCTL1   *Pulse on TOC5:  Port A bit 3 is transmit
        oraa    #$1     *for Pie Tin, motor R direction for SeaCow
        staa    TCTL1
```

```
NoDecision
        ldaa    spin
        beq     DoneSpin
*At this point, spin is set.
        ldaa    goingS
        bne     AlreadyS
        inca
        staa    goingS
        ldd     mcount
        std     StartSTime
AlreadyS
        ldd     mcount
        subd    StartSTime
        std     TimeSoFar       *See where we are in the spin

*First thing we'll do is spin to the right for 10 seconds
        cpd     #305
        bgt     nosR
        ldaa    #$1F
        staa    mlwant          *full forward
        ldaa    #$00
        staa    mrwant          *full reverse
        bra     DoneSpin
nosR

*After that, we'll spin to the left for 10 seconds
        cpd     #610
        bgt     nosL
        ldaa    #$00
        staa    mlwant          *full reverse
        ldaa    #$1F
        staa    mrwant          *full forward

        ldaa    TCTL1   *Don't pulse on TOC5 anymore:  stop signalling
        anda    #$fc    *for spin.
        staa    TCTL1
        bra     DoneSpin
nosL
        clra
        staa    spin
DoneSpin
        rts
goingS     RMB  1
StartSTime RMB  2
TimeSoFar  RMB  2




***************************
******* TOC5ISR ***********
***************************


TOC5ISR
        ldaa    TFLG1
        anda    #$08
        beq     DoneT5
        staa    TFLG1

        ldd     TOC5
        addd    #$1000
        std     TOC5

DoneT5  rti


***************************
******* TestAct ***********
```

```
**************************

TestAct
        ldd     mcount
        cpd     #400
        bgt     nostraight
        ldaa    #$19
        staa    mlwant
        staa    mrwant
        bra     donetest
nostraight
        cpd     #500
        bgt     noturn
        ldaa    #$14
        staa    mlwant
        ldaa    #$0B
        staa    mrwant
        bra     donetest
noturn
        ldd     #0
        std     mcount
donetest
        rts



**************************
******* Avoid ************
**************************

Avoid  ldx     #tof

*This next section marks all values past a certain point as infinite
*(zero), and also marks all the zeroes as FFFF (maxint) so that unsigned
*math can be used as comparisions.

        ldd     0,X
        cpd     #$4400
        blo     noclear0
        ldd     #0
noclear0
        addd    #$FFFF          *subtract one to make 0->FFFF (unsigned max)
        std     0,X


        ldd     2,X
        cpd     #$4400
        blo     noclear1
        ldd     #0
noclear1
        addd    #$FFFF          *subtract one to make 0->FFFF (unsigned max)
        std     2,X

        ldd     4,X
        cpd     #$4400
        blo     noclear2
        ldd     #0
noclear2
        addd    #$FFFF          *subtract one to make 0->FFFF (unsigned max)
        std     4,X

        ldaa    robot
        beq     DoPTAvoid
        jsr     AvoidSC
        bra     DACode
DoPTAvoid
        jsr     AvoidPT
DACode  rts

**************************
```

```
******* AvoidPT ***********
**************************

AvoidPT
*As input, X points to tof here.


*first see if they are all zero
        ldd     0,X
        cpd     #$FFFF
        bne     noallzeroPT
        ldd     2,X
        cpd     #$FFFF
        bne     noallzeroPT
        ldd     4,X
        cpd     #$FFFF
        bne     noallzeroPT

*We want to request all ahead.
*First zero moment, then accelerate

        ldaa    mlcurr
        cmpa    mrcurr
        blt     lagLPT          *if left<right, increase left
        bgt     lagRPT          *if right<left, increase right
*Ok, moment is zero, so accelerate
        ldaa    #$1D            *request all ahead
        staa    mlwant
        ldaa    #$1F
        staa    mrwant
        jmp     DAPT            *now quit

lagLPT  ldaa    mrcurr          *right is bigger, hold it steady..
        staa    mlwant
        staa    mrwant          *while we bring left up to speed.
        jmp     DAPT

lagRPT  ldaa    mlcurr          *left is bigger, so hold it steady...
        staa    mlwant
        staa    mrwant          *while we bring right up to speed.
        jmp     DAPT


noallzeroPT

        ldd     2,X             *get left motor value
        cpd     4,X             *compare to right motor value
        blo     gorightPT

*if here, left is >= right, so turn left

        ldaa    #$17            *+7 on left motor
        staa    mlwant
        ldaa    #$19            *+9 on right motor
        staa    mrwant
        jmp     DAPT

gorightPT
        ldaa    #$1A            *+10 on left motor
        staa    mlwant
        ldaa    #$16            *+6 on right motor
        staa    mrwant
        jmp     DAPT

DAPT    rts

**************************
******* AvoidSC **********
**************************
```

```
AvoidSC
*As input, X points to tof here.


*first see if they are all zero
        ldd     0,X
        cpd     #$FFFF
        bne     noallzeroSC
        ldd     2,X
        cpd     #$FFFF
        bne     noallzeroSC
        ldd     4,X
        cpd     #$FFFF
        bne     noallzeroSC

*We want to request all ahead.
*First zero moment, then accelerate

        ldaa    mlcurr
        cmpa    mrcurr
        blt     lagLSC          *if left<right, increase left
        bgt     lagRSC          *if right<left, increase right
*Ok, moment is zero, so accelerate
        ldaa    #$1F            *request all ahead
        staa    mlwant
        staa    mrwant
        jmp     DASC            *now quit

lagLSC  ldaa    mrcurr          *right is bigger, hold it steady..
        staa    mlwant
        staa    mrwant          *while we bring left up to speed.
        jmp     DASC

lagRSC  ldaa    mlcurr          *left is bigger, so hold it steady...
        staa    mlwant
        staa    mrwant          *while we bring right up to speed.
        jmp     DASC


noallzeroSC

        ldd     0,X
        cpd     #$3000
        bhi     RelaxSC

        ldd     2,X             *get left motor value
        cpd     4,X             *compare to right motor value
        blo     gorightSC

*if here, left is >= right, so turn left

        ldaa    #$00            *-15 on left motor
        staa    mlwant
        ldaa    #$08            *-7 on right motor
        staa    mrwant
        jmp     DASC

gorightSC
        ldaa    #$08            *-7 on left motor
        staa    mlwant
        ldaa    #$00            *-15 on right motor
        staa    mrwant
        jmp     DASC

RelaxSC
*This code runs when the value in front is far away, so the turn
*is not backwards

        ldd     2,X             *get left motor value
        cpd     4,X             *compare to right motor value
        blo     RLrightSC
```

```
*if here, left is >= right, so turn left

        ldaa    #$1A            *10 on left motor
        staa    mlwant
        ldaa    #$1F            *15 on right motor
        staa    mrwant
        jmp     DASC

RLrightSC
        ldaa    #$1F            *15 on left motor
        staa    mlwant
        ldaa    #$1A            *10 on right motor
        staa    mrwant
        jmp     DASC



DASC    rts



***************************
***** IsPing **************
***************************

IsPing  ldaa    #0
        staa    sval

Search
        ldaa    cursonar        ;get current sonar
        inca
        cmpa    #$3             ;3 sonar only
        bne     nozcs2
        clra
nozcs2  staa    cursonar
        ldab    p6val           ;get current p6
        andb    #$f8            ;kill 3LSBs
        aba                     ;a+b->a
        staa    p6val
        staa    $6000           ;record new sonar device

        ldaa    #$04
        staa    TFLG1           ;clear any pending ICs

*Now look for sonar
        jsr     Wait10ms
        ldaa    TFLG1
        anda    #$4
        beq     nothing
        jmp     foundfriend

nothing

        ldaa    sval
        inca
        staa    sval
        cmpa    #200
        bne     Search
        jmp     Donewait

foundfriend
        jsr     Wait150ms

Donewait
        rts
sval    RMB     1
```

```
***************************
******* Ping **************
***************************

Ping

*This area is involved in incrementing the sonar 0-1-2-0...
        ldaa    cursonar        ;get current sonar
        inca
        cmpa    #$3             ;3 sonar only
        bne     nozcs
        clra
nozcs   staa    cursonar

*This area is responsible for requesting the correct sonar be written.
        ldab    p6val           ;get current p6
        andb    #$f8            ;kill 3LSBs
        aba                     ;a+b->a
        staa    p6val

*Sea Cow does not request a sonar to ping, merely turning all of them on
*at once and then listening on a specific reciever.
        ldab    robot           ;if Sea Cow, set sonar select to zero
        beq     noseloff
        anda    #$f8            ;we must always write a zero here...
noseloff
        staa    $6000           ;record new sonar device


*Turn on either selected emitter or emitters in general.
        ldaa    #$81            ;turn emitter on
        staa    $7000
        jsr     Wait1ms
        ldaa    #$00          ;turn emitter off
        staa    $7000
        jsr     Wait1ms

        ldaa    p6val           ;re-record value in case lost
        staa    $6000

        ldaa    #$04
        staa    TFLG1           ;clear any pending ICs
        ldaa    #$00
        staa    TOIFLG          ;clear timer overflow

        ldx     #first
        ldab    cursonar        ;write to first+2*cursonar
        lslb
        abx
        ldd     TCNT
        std     0,X

        clra
        staa    bcount          ;start at 0

inloop  ldaa    TOIFLG          ;count up, while waiting for
        anda    #$1             ;response.
        beq     NoProblem

        clra
        staa    TOIFLG

        ldaa    bcount
        inca
        staa    bcount
        cmpa    #2
        beq     breakout

NoProblem
```

```
        ldaa    TFLG1
        anda    #$4
        beq     inloop

breakout

        ldaa    bcount
        cmpa    #$2
        bne     gotping
        ldx     #tof
        ldab    cursonar
        lslb
        abx
        ldd     #0
        std     0,X             ;write tof as zero
        bra     checknext


gotping

        ldx     #first
        ldab    cursonar        ;write to first+2*cursonar
        lslb
        abx
        ldd     TIC1
        subd    0,X             ;write to first array
        cpd     #$100           ;if less than #$100, don't change (ImmRes)
        blo     delayloop
        std     10,X            ;write to tof array


delayloop
*Now we recieved the ping, wait the rest of the time for timing
        ldaa    TOIFLG          ;count up, while waiting for
        anda    #$1             ;response.
        beq     delayloop

        clra
        staa    TOIFLG

        ldaa    bcount
        inca
        staa    bcount
        cmpa    #2
        bne     delayloop

checknext
        ldaa    cursonar
        beq     Printloop
        jmp     Ping


Printloop

        ldx     #tof
        ldab    cursonar
        lslb
        abx
        ldd     0,X
        bne     okprint
        ldx     #Timeout
        jsr     OutStr
        bra     pastprint

okprint
        addd    #$20
        jsr     Hexph
        jsr     Hexpl
        tba
        jsr     Hexph
```

```
        jsr     Hexpl

pastprint
        ldaa    #$20    ;load space
        jsr     OutChar
        jsr     OutChar

        ldaa    cursonar
        inca
        staa    cursonar
        cmpa    #$3
        bne     Printloop
        clra
        staa    cursonar

        ldaa    #$20    ;load space
        jsr     OutChar
        jsr     OutChar
        jsr     OutChar
        ldaa    mlwant
        jsr     Hexph
        jsr     Hexpl
        ldaa    #$2F
        jsr     OutChar
        ldaa    mlcurr
        jsr     Hexph
        jsr     Hexpl


        ldaa    #$20    ;load space
        jsr     OutChar
        jsr     OutChar
        ldaa    mrwant
        jsr     Hexph
        jsr     Hexpl
        ldaa    #$2F
        jsr     OutChar
        ldaa    mrcurr
        jsr     Hexph
        jsr     Hexpl

        ldaa    #$20
        jsr     OutChar

        ldaa    PACNT
        jsr     Hexph
        jsr     Hexpl

        ldaa    #$20
        jsr     OutChar
        jsr     OutChar

        ldaa    click
        beq     nclick
pclick  ldx     #Click
        jsr     OutStr
        dec     click
        ldaa    click
        bne     pclick

nclick
        ldx     #Enter
        jsr     OutStr
        rts


***************************
*******Wait10ms************
***************************
Wait10ms
        jsr     Wait1ms
```

```
        jsr     Wait1ms
        jsr     Wait1ms
        jsr     Wait1ms
        jsr     Wait1ms
        jsr     Wait1ms
        jsr     Wait1ms
        jsr     Wait1ms
        jsr     Wait1ms
        jsr     Wait1ms
        rts

***************************
*******Wait150ms***********
***************************
Wait150ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        jsr     Wait10ms
        rts

***************************
*******Wait1ms*************
***************************

Wait1ms ldd     TCNT
        addd    #2000
        std     TOC4
        ldaa    #$10            ;clear current flag (so we wait)
        staa    TFLG1
ms1a    ldaa    TFLG1
        anda    #$10
        beq     ms1a
        rts

***************************
*******Radio***************
***************************

Radio
        ldaa    robot
        beq     radioD          *Pie Tin has nothing for this

        ldd     mcount          *Only check every 4th TOI
        andb    #$3
        bne     radioD
*Use this to determine the pulse density to determine behavoir mod
        ldaa    PACNT
        cmpa    #10
        blt     nobehave
        ldaa    #1
        staa    spin
nobehave
        clra
        staa    PACNT

radioD  rts
```

```
***************************
*******Rinit***************
***************************

Rinit
        ldaa    robot
        bne     SCrecv

        ldaa    PACTL
        oraa    #$8     *Set Port A bit 3 to output for emitter
        staa    PACTL

        ldd     #$2000
        std     TOC5

        ldaa    TMSK1   *Use TOC5
        oraa    #$8
        staa    TMSK1

        ldaa    OC1M
        oraa    #$80
        staa    OC1M

        bra     DRinit


SCrecv  ldaa    PACTL   *increment PACNT on falling edge
        anda    #$cf
        staa    PACTL

        clra
        staa    PACNT   *write a zero to Pulse accumulator

DRinit  rts



***************************
*********TOI ISR***********
***************************
TOIISR  ldaa    TFLG2
        anda    #$80
        bne     noleave
        jmp     TOIDONE
noleave
        staa    TFLG2
        ldd     mcount
        addd    #$1
        std     mcount
        ldaa    TOIFLG
        oraa    #$1
        staa    TOIFLG

        jsr     Radio

*only update once every two cycles
        ldd     mcount
        andb    #$1
        bne     outl

        ldaa    mlcurr
        cmpa    mlwant
        beq     donel
        blt     goingdownl
        deca
        bra     donel
goingdownl
```

```
        inca
donel   staa    mlcurr


outl    ldaa    mlcurr

*first write direction
        ldab    directionl
        anda    #$10    *get MSB to determine motor direction
        staa    directionl
        cba             *see if we just switched motor direction
        beq     noclickl
        inc     click
noclickl
*done recording whether a click occured or not


*a now contains direction in bit 4...
        ldab    robot
        bne     LeftSC  *seperate code for Sea Cow...
*Do Pie tin left motor code...
        lsla
        lsla
        staa    ltemp0
        eora    #$bf
        staa    ltemp1
        ldaa    p6val
        oraa    ltemp0
        anda    ltemp1
        staa    p6val
        staa    $6000
        bra     DoneDL

LeftSC
*Now will modify port A bit 4
        staa    ltemp0
        eora    #$ef
        staa    ltemp1
        ldaa    paval
        oraa    ltemp0
        anda    ltemp1
        staa    paval
        staa    PORTA


*Now done setting left direction pin
DoneDL
        ldaa    mlcurr
        ldab    directionl
        bne     noflipl *flip if direction bit not set
        nega
        adda    #$15
noflipl anda    #$0F
        staa    magnitude
        cmpa    #0
        bne     nozerol

        ldaa    #$40
        staa    TFLG1
        ldaa    OC1D
        anda    #$bf
        staa    OC1D
        bra     doright

*not zero, set it up to be ok...
nozerol ldaa    #$40
        staa    TFLG1
        ldaa    OC1D
        oraa    #$40
        staa    OC1D
        ldab    magnitude
```

```
        lslb
        ldx     #gotablel
        abx
        ldd     TCNT
        addd    0,X
        std     TOC2


*now do the right motor

doright

*only update once every four cycles
        ldd     mcount
        andb    #$3
        bne     outr

        ldaa    mrcurr
        cmpa    mrwant
        beq     doner
        blt     goingdownr
        deca
        bra     doner
goingdownr
        inca
doner   staa    mrcurr


outr    ldaa    mrcurr

*first write direction
        ldab    directionr
        anda    #$10    *get MSB to determine motor direction
        staa    directionr
        cba             *see if we just switched motor direction
        beq     noclickr
        inc     click
noclickr

*a now contains direction in bit 4...
        ldab    robot
        bne     RightSC  *seperate code for Sea Cow...
*Do Pie tin right motor code...

        lsla
        staa    ltemp0
        eora    #$df
        staa    ltemp1
        ldaa    p6val
        oraa    ltemp0
        anda    ltemp1
        staa    p6val
        staa    $6000
        bra     DoneDR

RightSC
        lsra                    *Scale to portA bit 3
        staa    ltemp0          *save as 08 or 00
        eora    #$f7            *save as ff or f7
        staa    ltemp1
        ldaa    paval
        oraa    ltemp0
        anda    ltemp1
        staa    paval
        staa    PORTA
DoneDR


        ldaa    mrcurr
        ldab    directionr
        bne     noflipr *flip if direction bit not set
```

```
        nega
        adda    #$15
noflipr anda    #$0F
        staa    magnitude
        cmpa    #0
        bne     nozeror

        ldaa    #$20
        staa    TFLG1
        ldaa    OC1D
        anda    #$df
        staa    OC1D
        bra     TOIDONE

*not zero, set it up to be ok...
nozeror ldaa    #$20
        staa    TFLG1
        ldaa    OC1D
        oraa    #$20
        staa    OC1D
        ldab    magnitude
        lslb
        ldx     #gotabler
        abx
        ldd     TCNT
        addd    0,X
        std     TOC3



TOIDONE rti
magnitude       RMB     1
directionl      RMB     1
directionr      RMB     1
ltemp0          RMB     1
ltemp1          RMB     1
***************************



************************************************************************
*                       SUBROUTINE - InitSCI
* Description: This subroutine initializes the BAUD rate to 9600 and
*              sets up the SCI port for 1 start bit, 8 data bits and
*              1 stop bit.  It also enables the transmitter and receiver.
*              Effected registers are BAUD, SCCR1, and SCCR2.
* Input         : None.
* Output        : Initializes SCI.
* Destroys      : None.
* Calls         : None.
************************************************************************
*
InitSCI PSHA                            * Save contents of A register

        ldaa #$30                       * Set BAUD rate to 9600
        staa BAUD
        ldaa #$0                        * Set SCI Mode to 1 start bit,
        staa SCCR1                      *     8 data bits, and 1 stop bit.
        ldaa SCCR2                      * Enable SCI Transmitter
        ora  #$0c                       *     and Receiver
        staa SCCR2

        PULA                            * Restore A register
        RTS                             * Return from subtoutine
```

```
***********************************************************************
*                       SUBROUTINES -  Hexph and Hexpl
* Description: Outputs the hex digit in high or low a after
*              checking if the Transmitter Data Register is Empty
* Input         : Data to be transmitted in register A.
* Output        : Transmit the data.
* Destroys      : None.
* Calls         : None.
***********************************************************************
*
Hexph   psha            ;save a away
        lsra
        lsra
        lsra
        lsra
        bra     hexs    ;after scaled, go to start
Hexpl   psha
hexs    anda    #$0F
        adda    #48     ;scale to ASCII 0
        cmpa    #58     ;if above or equal, need more scaling
        blt     hexl    ;jump over correction if OK
        adda    #7      ;now will print A-F properly
hexl    jsr     OutChar ;print out character
        pula            ;get a back
        rts             ;bye bye


***********************************************************************
*                       SUBROUTINE -  OutStr
* Description: Outputs the string terminated by EOS. The starting location
*              of the string is pointed by X register. Calls the OutChar
*              subroutine to display a character on the screen and
*              exit once EOS has been reached.
* Input         : Starting location of the string to be transmitted
*               : (passed in X register)
* Output        : Prints the string.
* Destroys      : contents of X register.
* Calls         : OutChar.
***********************************************************************
*
OutStr  PSHA                        * Save contents of A register
Loop2   ldaa    0,x                 * Get a character (put in A register)
        cmpa    #EOS                * Check if it’s EOS
        beq     Done                * Branch to Done if it’s EOS

        JSR     OutChar             * Print the character by calling OutChar
        inx                         * Point to next character
        BRA     Loop2               * Branch to Loop2 for the next char.
Done    PULA                        * Restore A register
        RTS                         * Return from subtoutine


***********************************************************************
*                SUBROUTINE  -  InChar
* Description: Receives the typed character into register A.
* Input         : None
* Output        : Register A = input from SCI
* Destroys      : Contents of Register A
* Calls         : None.
***********************************************************************
*
InChar

pollrecv          ldaa SCSR              ; Check status reg.
```

```
                  anda #$20              ;     (load it into A reg)
                 cmpa #0              ; Check if receive buffer full
                 beq pollrecv         ; Wait until data present

                  ldaa SCDR              ; SCI data ==> A register
                 RTS                    ; Return from subroutine


************************************************************************
*                       SUBROUTINE -  OutChar
* Description: Outputs the character in register A to the screen after
*              checking if the Transmitter Data Register is Empty
* Input         : Data to be transmitted in register A.
* Output        : Transmit the data.
* Destroys      : None.
* Calls         : None.
************************************************************************
*
OutChar PSHB                        * Save contents of B register
Loop1   ldab    SCSR                * Check status reg (load it into B reg)
        andb    #$80                * Check if transmit buffer is empty
        BEQ     Loop1               * Wait until empty
        staa    SCDR                * A register ==> SCI data
        PULB                        * Restore B register
        RTS                         * Return from subtoutine
```