MACO

Multiple Agent Climbing Organism

Final Report

Ted Belser
University of Florida, Department of Computer Engineering
EEL 5666, Machine Intelligence Design Laboratory

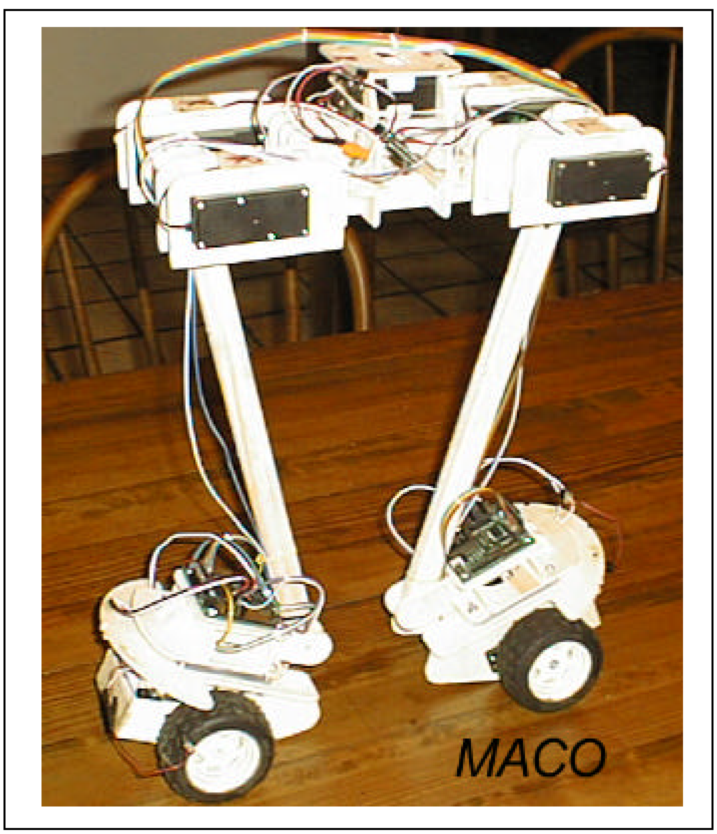


Table of Contents

Abstract

The design objective of this project is to develop an autonomous system of robots that can coordinate and perform the task of climbing a 30-inch wall. A mechanical design based on multiple constraints allows for three agents to be connected as a cooperative whole. Many design problems arose during the development of this system.

Executive Summary

The acronym MACO stands for 'multiple agent climbing organism.' The MACO design is based on an SBIR topic calling for a shape-shifting robot capable of reaching inaccessible locations. Inspired by this topic the design objective of MACO is to climb a thirty-inch wall. This objective was not achieved because of time limitations and design mistakes.

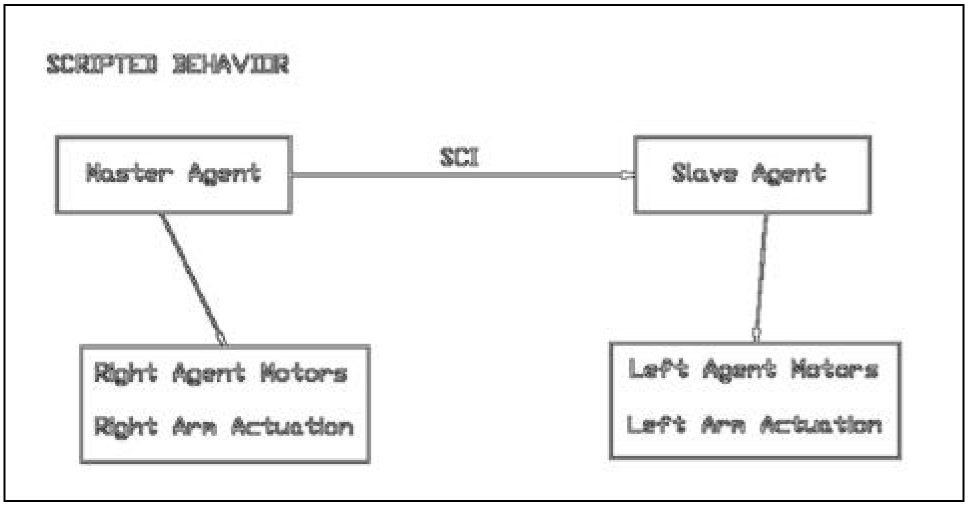
MACO is a project in progress, as such there is not much to report on the capabilities of the platform. The project, however, has resulted in many lessons learned.

Introduction

The design theme for MACO is influenced by an SBIR (http://www.darpa.mil/SBIR/sbir.html) topic calling for a shape-shifting robot capable of reaching inaccessible locations. Building a robot to actually meet this design description is beyond the scope and time of this class. Instead MACO is designed to be a proof of concept prototype on which to build future designs. The design goal of MACO is to climb onto a 30" wall. This objective was never realized because of the time constraints of a summer semester. The problems encountered will be addressed within the paper. Basic functionality in mechanical design however was achieved and information gathered during the design project will be useful for any future developments of climbing platforms. This paper primarily discusses the mechanical design process of MACO. For sensory input MACO uses standard IMDL sensors including analog IR and discrete bump sensors. Stimulus response is actuated by four large 330 oz-in servos and four 42 oz-in servos modified to run as motors. Behavior exhibited through actuation was not well developed by project's end. Overall MACO was an ambitious project to design and build within a three-month period.

Integrated System

MACO was designed as a multiple agent organism. Each of the three robots of MACO was to function independently of each other. Each individual would make a request of the other when a particular task was to be accomplished. Unfortunately this type of system was never implemented in MACO.



Two Motorola 68HC11 processors control MACO, one on each of the end agents, provide PWM signals to drive all 8 servos. Inter-processor communication is achieved through use of the SCI system. The communication protocol involves a master and a slave. The master controls the slave by sending single characters over the SCI line. Despite duplex capability MACO only sends data in one direction, from master to slave.

The code written for the end of semester presentation was developed in the ICC11 compiler using Mekatronix TJPro libraries. An RTI multitasking kernel was developed entirely in assembly but because of time constraints was never used. The RTI multitasking code works and is appended to this document.

An eight-segment LED display was built for decoding and user interface purposes. The TJPro board has four fully decoded active low outputs and four fully decoded active low input addresses. An output port (74HC574 8-bit latch) drives the display. The cathode of each LED in the display is connected to one of the output bits and the anode is connected to a 33kΩ resistor to ground. A software driver was written for the display. This driver ran successfully within the RTI kernel.

An assembly servo driver was also written for the RTI multitasking kernel but was never fully debugged. The driver controls servos using Output Compare ports OC2 – OC5.

Behaviors are scripted because MACO has no sensory capabilities. The integrated system design of MACO is largely undeveloped.

Mobile Platform

The mechanical design of MACO posed the greatest design challenge of the project.

3 Individual Agents

A 3-agent design simplifies the mechanics of a climbing robot. The middle robot acts as a fulcrum on which the end robots balance.

Large Servos

As a climbing robot MACO must have the mechanical power to lift itself to a particular height. The design goal is 30” but without a wall-clinging mechanism this must be achieved using a structure at least 30” tall. This is where the torque problem is introduced into the design. Torque is directly proportional to the length of the moment arm. A larger moment arm requires larger servos. This relationship is a design divergence typical to engineering. This divergence is easily quelled by noticing that as an arm reaches a vertical position the gravity component of the torque approaches zero. The largest load on the arm occurs when the arm is parallel to the ground. This produces the first design constraint.
Constraint 1-- The arm actuator must produce the torque required when the arm is parallel to the earth. This torque is calculated by multiplying the mass of the end robot by the length of the arm at zero degrees.
Constraint 2-- The largest servo within budget is a 350 oz-in Hitec 805BB+. Four of these servos were bought.

Extending Arm

Constraint 3-- A functional end robot weighs ~25 oz.
The above constraints result in a middle robot with two 350oz-in servos for each arm. This produces a total torque of 700 oz-in on each arm. 700 oz-in divided by 25 oz allows for a moment arm of 28 inches. Dividing again by 1.75 for safety allows for a 16 inch moment arm. This result is one-half of the 30 inch design goal. The simplest solution toward achieving the 30in span is to telescope the arm. A telescoping arm however was not achieved in this project.

Middle Agent

The function of the middle agent is solely to provide the mechanical means to climb onto a ledge. 4 servos, 2 on each arm perform one of two functions. Each pair is either lifting the end robot or lifting the middle robot. Each of these functions is performed in the various phases of the climbing behavior (See drawing 1).

End Agents

The end agents perform different functions depending on whether the robot is combined or undocked. As a part of the combined robot the end agent acts as a foot for the middle agent. The end agents also provide mobility for the organism when MACO is not performing the climbing behavior. The end agents are based on a TJ platform, modified to function as part of the MACO design. Undocked end agents were never realized in this project.

Actuation

As a climbing robot sufficient actuation is needed to achieve a workable climbing behavior. Agile actuation is a critical design objective of the MACO platform. The primary actuation of MACO is achieved through two servo pairs on each side of the middle robot. Each servo pair provides the speed and strength to move MACO with agility. The Hitec 805BB+ servos provide 350 oz-in of torque at 6.0 V each. The speed of the servos as specified by Hitec RCD is 0.16sec/60°. The Hitec servos perform well as an integral part of the MACO platform. The strength and speed of the servos are not only sufficient for quasi-static behavior but also have the potential to perform dynamic behaviors. Dynamic mechanical behaviors are untested at this stage in the development but the agility of the Hitec servos bodes probable success.

Some careful calibration was required to develop the software driving the arms. The two servos on each arm must move synchronously. This was achieved by moving each servo to its extents and recording the corresponding pulse-width. The extents are matched so that even though one servo may be able to move further it does not move beyond the extents of the other servo. Knowing both the clockwise and counter-clockwise extents of the servo a pulse-width range is calculated. This range is then divided by 300 producing a 300-point resolution to the movement of the arm. The following code snippet shows the implementation of the algorithm.

```
/* MoveRightArm */
void MoveRightArm(position)
{
      servo(0,(11.5*position)+1650);
      servo(1,5000-(11.3*position));
}
```

The power and speed of the Hitec servos may pose a problem in certain applications. When programming the motion of these servos one should consider gradually changing the pulse-width rather than sending a position and having the servo circuitry control the movement. This gradual change is easier to handle and the physics calculations are simpler.

Planar locomotion is achieved through hacked 42 oz-in servos, two on each end robot. The servo modification (hack) is described in Mekatronix reference material. These hacked servos drive 3” diameter, 1” wide wheels. The wheels are attached by compression fit to a disk screwed on the servo horn.

Sensors

The sensor suite of MACO is minimal and unimplemented. The sensors used are analog infrared sensors and bump sensors. A sonar circuit built for use on MACO was not implemented.

Behaviors

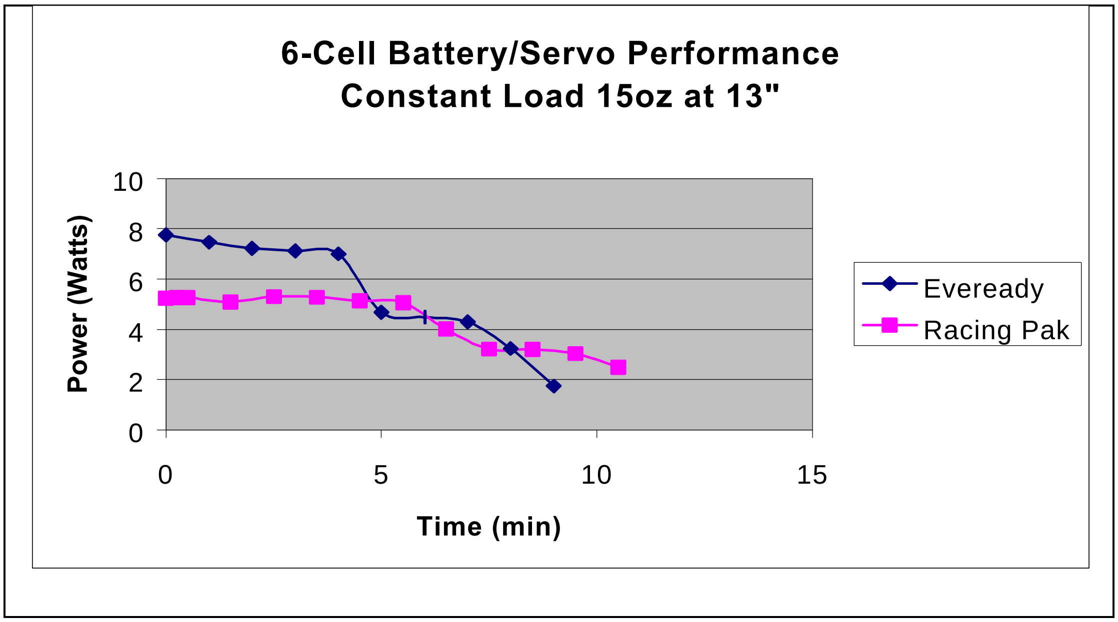
MACO can only perform one scripted behavior. This behavior converts MACO from a completely flat position to a raised position (see title page).

Experimental Layout and Results

Power Concerns

One major concern is the power consumption of the large 350 oz-in servos. The included data shows the battery performance of two battery packs. For each battery pack data was sampled for multiple loads and also for a constant load measured over time.

Graph 1 compares the performance of two battery packs, a lab standard 6-cell eveready pack and a 6-cell 1200 mAh Vinnic racing pack. The Vinnic pack was charged for 15 minutes on an AstroFlight quick charger. The eveready pack was charged by a dc power supply over 10 minutes. The data seems to indicate that the charges were insufficient.



The eveready pack provided an average of 1.06 Amps over 10 minutes. Power from the everyready was only sufficient during the first 5 minutes, averaging 1.74 Amps. This particular charge only provided 96mAh of sufficient power. There is no printed power rating on the eveready pack, however 96mAh is obviously well below expected performance.

The Vinnic pack provided an average of 1.12 Amps over 10.5 minutes. Power from the Vinnic was only sufficient for the first 6 minutes, averaging 1.19 Amps. This particular charge only provided 130 mAh of sufficient power. The Vinnic pack is rated at 1200 mAh.

Although the eveready pack did not last as long as the Vinnic it did provide more power to the servos. The 6-year age of the Vinnic pack may be attributable to the poor performance. The 10 minute quick-charge may be attributable to the poor performance of the eveready pack.

Conclusion

MACO is an unfinished project but it is a working prototype with the potential for further experimentation. The notable performance of the Hitec servos indicates that a climbing behavior is likely.

MACO was conceived as an alternative to the typical platforms of past IMDL projects. The design of the platform expended more time than expected. This attention to mechanical design resulted in many unfinished components. Time spent on conceptualizing mechanisms such as arm extension and docking compounded into a large amount of wasted time resulting in an unfinished project. Initially MACO's software was written in assembly because of the freedom and control of assembly source, but this also expended valuable time needed to produce a working project. This apparent failure however is a large lesson in design. Listed below are some important lessons learned when developing a new mechanical platform:

1. The acquisition of parts is the most time consuming activity in the design process. If the parts are not readily available the designer must find and posses them within one week or change the design.
2. A corollary to lesson 1: Design around parts that are readily available. Do not assume the existence of a part.
3. Use AutoCAD to visualize the entire project. Become familiar with the AutoCAD tools and use them to test all mechanical extents.
4. Spend only 50% of time on design and construction of the prototype. Spend the other 50% on design of software and the debugging of the prototype's design.
5. Make no assumptions.

Because MACO is an incomplete project there is much room for future work on the design. This report is a summary of a semester of work on MACO. Work on MACO will be continued to a satisfactory conclusion and a more comprehensive report will replace this one.

Documentation

[1] Mekatronics TJPro Assembly Manual. http:\\www.Mekatronix.com

[2] Motorola HC11 Reference Manual, Motorola Inc. 1991

[3] Motorola HC11 Programming Reference Guide, Motorola Inc. 1991

[4] Katia P. Sycara, *Multiagent Systems,* AI Magazine, Vol. 18 No. 2, American Association for Artificial Intelligence, Summer 1998

Appendicies

Left Robot Demonstration Code

```
#include <tjpbase.h>
#include <servotjp.h>
#include <stdio.h>

//**** Behaviors and Arbitrator */

void MoveLeftArm(int);

//***** Main Loop *****/
char command;
void main(void)
{
      init_analog();
      init_clocktjp();
      init_motortjp();
      init_servotjp();
      while(1){
            command=getchar();
            if(command == 'g'){
                  motorp(0,100);
                  motorp(1,100);
            }
            else if(command == 's'){
                  motorp(0,0);
```

```
            motorp(1,0);
        }
        else if(command == 'l'){
            MoveLeftArm(10);
            motorp(0,-100);
            motorp(1,-100);
            wait(500);
            motorp(0,0);
            motorp(1,0);
        }
    }
}

/* MoveLeftArm */
void MoveLeftArm(position)
{
    servo(0,(11.5*position)+1650);
    servo(1,5040-(11.17*position));
}
```

Right Robot Demonstration Code

```
#include <tjpbase.h>
#include <servotjp.h>
#include <motortjp.h>
#include <stdio.h>

//**** Behaviors and Arbitrator */

void MoveRightArm(int);

//***** Main Loop *****/
void main(void)
{
    init_analog();
    init_clocktjp();
    init_motortjp();
    init_servotjp();
    printf("l");
    MoveRightArm(10);
    motorp(0,-100);
    motorp(1,-100);
    wait(500);
    printf("s");
    motorp(0,0);
    motorp(1,0);
    wait(500);
    printf("g");
    motorp(0,-100);
    motorp(0,-100);
    wait(2000);
    printf("s");
    motorp(0,0);
    motorp(0,0);
}

/* MoveRightArm */
```

```
void MoveRightArm(position)
{
      servo(0,(11.5*position)+1650);
      servo(1,5000-(11.3*position));
}
```

MACO Multitasking RTI kernel including buggy servo drivers and LED display driver

```
*********************************************************************
* Title          : MACO multi-tasking kernel
* Filename       : maco.asm
* Programmer     : Ted Belser
* Date           : 6-9-98
* Version        : A.0
*
* Drivers     :
*             Driver1 -- LED 7 segment display driver
*             Driver2 -- Servo Handler
*
***************************************************************************
* DEFINITIONS
***************************************************************************
*

REGS   EQU    $1000         ;
_TCNT  EQU    $0E           ; TCNT High byte
_TCTL1 EQU    $20           ;
_TCTL2 EQU    $21           ;
_TFLG1 EQU    $23           ;
_TFLG2 EQU    $25           ; Contains RTIF flag
_TMSK1 EQU    $22           ;
_TMSK2 EQU    $24           ; RTII enable flag
_PACTL EQU    $26           ; RTI Timer control
_BAUD  EQU    $2B           ; BAUD rate control register to set the BAUD rate
_SCCR1 EQU    $2C           ; Serial Communication Control Register-1
_SCCR2 EQU    $2D           ; Serial Communication Control Register-2
_SCSR  EQU    $2E           ; Serial Communication Status Register
_SCDR  EQU    $2F           ; Serial Communication Data Register
_OPTION       EQU    $39           ; Option Register

TCNT    EQU     $100E             ; TCNT High byte
TCTL1  EQU    $1020         ; Pin Control for TOC
TCTL2  EQU    $1021         ;
TFLG1  EQU    $1023         ;
TFLG2   EQU     $1025             ; Contains RTIF flag
TMSK1  EQU    $1022         ; TOC Interrupt enable
TMSK2   EQU     $1024             ; RTII enable flag
OC1D   EQU    $100D         ; TOC1
OC1M   EQU    $100C         ; TOC1
TOC1   EQU    $1016         ; TOC1 Register
TOC2   EQU    $1018         ; TOC2 Register
TOC3   EQU    $101A         ; TOC1 Register
TOC4   EQU    $101C         ; TOC2 Register
TOC5   EQU    $101E
PACTL   EQU     $1026             ; RTI Timer control
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
```

```
OPTION EQU    $1039         ; A/D Power Up (bit 7)
ADCTL  EQU    $1030         ; A/D Control Status
ADR1   EQU    $1031         ; A/D Results
ADR2   EQU    $1032         ;
ADR3   EQU    $1033         ;
ADR4   EQU    $1034         ;

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
BIT0   EQU    %00000001     ;
BIT1   EQU    %00000010     ;
BIT2   EQU    %00000100     ;
BIT3   EQU    %00001000     ;
BIT4   EQU    %00010000     ;
BIT5   EQU    %00100000     ;
BIT6   EQU    %01000000     ;
BIT7   EQU    %10000000     ;

*
************************************************************************
* Initialize Interrupt Jump Vectors
************************************************************************
* Buffalo jump vectors
*      ORG    $00EB
*      jmp    RTI_ISR
*      ORG    $00F7
*      jmp    ILL_OP_ISR
*      ORG    $00DC
*      jmp    TOC2_ISR
*      ORG    $00D9
*      jmp    TOC3_ISR
*      ORG    $00D6
*      jmp    TOC4_ISR
*      ORG    $00D3
*      jmp    TOC5_ISR
       org    $fff0
       fdb    RTI_ISR
       ORG    $fff8
       fdb    ILL_OP_ISR
       ORG    $ffe6
       fdb    TOC2_ISR
       ORG    $ffe4
       fdb    TOC3_ISR
       ORG    $ffe2
       fdb    TOC4_ISR
       ORG    $ffe0
       fdb    TOC5_ISR
       ORG    $fffe
       fdb    $8000

*
************************************************************************
* Define Strings and Reserve Variable memory space for system use
*   such as CPT, DSPT, CurPID, etc.
********************************************************************
        ORG     $8000
        jmp   Main
*
PID    rmb    1
CPT    rmb    16
DSPT   rmb    16
```

```
ClrScr  FCB    ESC,$5B,$32,$4A ; ANSI sequence to clear screen
        FCB    ESC,$5B,$3B,$48 ;   and move cursor to home
        FCB    EOS             ; EOS character
*
*******************************************************************
*               MAIN PROGRAM
*
* The main program performs the following tasks:
*  - Disable interrupts
*  - Initialize RTI
*  - Initialize SCI
*  - Zero out the initial Current Process Table
*  - Initialize system CurPID
*  - Initialize CPT[0] with DSPT[0]
*  - Initialize SP with CPT[0]
*  - Enables interrupts

*******************************************************************
Main   lds    #$41         ; Initialize Stack Pointer
       sei                 ; disable interrupts
       jsr    InitSCI             ; initialize the SCI system
       jsr    InitRTI             ; initialize the RTI system
       jsr    InitTables   ; Initialize tables
       ldaa   #0           ; Initialize the PID
       staa   PID          ; "
       ldx    DSPT         ; Initialize CPT[0]
       stx    CPT          ; "
       lds    CPT          ; Initialize SP with CPT[0]
       cli                 ; enable interrupts
* Execute the Autoexec
       jmp    Autoexec     ; Startup the system
*
*******************************************************************
* Subroutine:  InitTable
* Function:    Initializes Current Process Table
* Input:       None
* Output:      initializes the process table
*******************************************************************
*
InitTables
       psha
       pshb
       pshx

       ldab   #0                  ;
       ldx    #CPT                ; zero the CPT
Loop_Tables
       stab   0,X                 ;
       inx                        ;
       cpx    #CPT+16
       bne    Loop_Tables         ;
       ldy    #$90FF              ; create the DSPT
Loop_Tables_2
       sty    0,X                 ;
       inx                        ;
       inx                        ;
       ldab   #$FF                ;
       aby                        ;
       iny
       cpx    #CPT+32                    ;
       bne    Loop_Tables_2       ;

       pulx
```

```
        pulb
        pula
        rts
*
*******************************************************************
* Subroutine:  InitRTI
* Function:    This routine enables RTIs and sets the RTI rate to
*              32.77ms.
* Input:       None
* Output:      Initializes RTI
*******************************************************************
*
InitRTI
        pshx

        ldx    #REGS
        bset   _TMSK2,X %01000000  ; turn on RTI
        bset   _PACTL,X %00000011  ; set to 32.7 ms

        pulx
        rts
*
*******************************************************************
* Interrupt Service Routine (ISR): RTI_ISR
* Function:  This ISR services the Real-Time Interrupts.
* This ISR should do the followings:
*   - Clear RTI flag.
*   - Update current SP in CPT[Current PID]
*   - Find next PID
*   - Update CurPID
*   - Load New SP from CPT[Next PID]
*******************************************************************
*
RTI_ISR
        ldx    #REGS
        bset   _TFLG2,X %01000000  ; clear the RTI Flag
* save old stack ptr
        ldab   PID                 ; load PID into b
        ldx    #CPT                ; load CPT start in X
        lslb                       ; multiply PID by 2
        abx                        ; and add to X (X = CPT)
        sts    0,X                 ; store old stack ptr to the table
Set_new
        ldab   PID                 ;
        cmpb   #7                  ; Restart the PID at 0 if
        beq    Restart_PID         ; PID = 7.
* set new stack ptr
        inc    PID                 ; increment the PID
Set_new_2
        ldab   PID                 ; load the PID into b
        ldx    #CPT                ;
        lslb                       ;
        abx                        ;
        ldy    0,X                 ; load the stk ptr into y
        cpy    #$0000              ; check if is empty slot
        beq    Set_new                    ; go to next slot
        lds    0,X                 ; load stack ptr from table
        bra    END_RTI_ISR         ;
Restart_PID
        ldaa   #0                  ;
        staa   PID                 ; reset PID to 0
        bra    Set_new_2           ;
END_RTI_ISR
```

```
      rti                         ; return from interrupt
*
*********************************************************************
* Subroutine: Spawn
* Function:   generates a new process
* Input:      X: starting address of the process
* Output:     A: PID of the process just created, or
*                $FF if no slots are available
* Destroys:   Contents of A register
* Side effects: Creates the initial stack for the process.  This
*               stack must have the process PID in A, and %01000000
*               in CCR.
*********************************************************************
*
Spawn
      pshy
      pshb
      pshx                        ; push pc on stack

      ldx    #CPT                 ; first find open slot
      ldaa   #0                   ;
Loop1_Spawn
      cpx    #CPT+16                     ;
      beq    Error_End            ;
      ldy    0,X                  ;
      cpy    #0                   ;
      beq    Start_Proc           ;
      inx                         ;
      inx                         ;
      inca                        ;
      bra    Loop1_Spawn          ;
Start_Proc
      tab                         ;
      lslb                        ;
      ldx    #DSPT                ; move default stack location
      abx                         ; to CPT.
      ldy    0,X                  ;
      ldx    #CPT                 ;
      abx                         ;
      sei                         ; disable interrupts
      sty    0,X                  ; store new stk ptr in CPT
      ldx    0,X                  ; load the stk ptr into x
      ldab   #%0100000            ; load init ccr value into b
      stab   1,X                  ;
      staa   3,X                  ;
      puly                        ; pull pc from stack
      sty    8,X                  ; set pc
      cli                         ; enable interrupts
      pshy                        ; copy pc to X
      pulx                        ;
      bra    End_Spawn
Error_End
      pulx                        ;
      ldaa   #$FF                 ;
End_Spawn
      pulb
      puly
      rts

*
*********************************************************************
* Subroutine: Kill
* Function:   removes a currently active process
```

```
* Input:      A: process ID to kill
* Output:     A: process ID just killed
* Destroys:   None
*********************************************************************
*
Kill
       pshx
       pshb
       sei                         ; disable interrupts

       ldx    #CPT                 ;
       tab                         ;
       lslb                        ;
       abx                         ;
       ldab   #0                   ;
       stab   0,X                  ;
       stab   1,X                  ;

       cli                         ; enable interrupts
       pulb
       pulx
       rts
*
*********************************************************************
* Subroutine: ILL_OP_ISR
* Function:   Prints a message if ther is an Illegal op-code
* Input:
* Output:
* Destroys:   Stops operation of all processes and returns to
*            Buffalo.
*********************************************************************
*
ILL_OP_ERR   fcc    @FATAL ERROR ::: Illegal Operation@
             fcb    EOS
ILL_OP_ISR
       ldx    #ClrScr
       jsr    OutStr
       ldx    #ILL_OP_ERR
       jsr    OutStr
       swi

** AUTOEXEC **
*********************************************************************
Autoexec
       ldx    #Driver1
       jsr    Spawn
       ldx    #Driver2
       jsr    Spawn
Auto_end
       bra    Auto_end
** DRIVERS **
*********************************************************************
* Driver1
* Function: Drives the one 7 segment display
* Input   :
* Output  : Outputs pattern to displays
* Note    :
*********************************************************************
* interface
D1MT   rmb    32                   ; 16 Message QUEUE
* Characters
LEDbp1 fcb    $48,$C7,$CD,$59,$9D,$9F,$C8,$DF,$DD,$DE
* Equates
```

```
LED1   EQU    $4000               ;

Driver1
* Initialization                  ;
       ldab   #0                  ;
       ldx    #D1MT               ;
D1_L_1
       stab   0,X                 ;
       inx                        ;
       cpx    #D1MT+32
       bne    D1_L_1              ;
* Driver Main Loop
D1_ML  ldx    #D1MT-2                    ;
D1_L_2 inx                        ;
       inx                        ;
       cpx    #D1MT+32            ;
       beq    D1_ML               ;
       ldy    0,X                 ;
       cpy    #0                  ;
       beq    D1_L_2              ;
       ldab   0,Y                 ; Read the number of display
D1_L_3 cmpb   #0                  ; patterns in message.
       beq    D1_L_2              ;
       decb                       ;
       iny                        ;
       ldaa   0,Y                 ;
       staa   LED1                ;
       iny                        ;
       ldaa   0,Y                 ;
D1_L_4 jsr    Delay               ;
       deca                       ;
       cmpa   #0                  ;
       bne    D1_L_4              ;
       bra    D1_L_3              ;
* End of Driver1 Code
*
*********************************************************************
* Driver2
* Function: Drives 4 servos using TOCs 1-4
* Input   :
* Output  :
* Note    :
*********************************************************************
* Interface
Duty2E rmb    2                   ; Other Processes may change
Duty3E rmb    2                   ; these values.
Duty4E rmb    2                   ;
Duty5E rmb    2                   ;
* Local Vars
Duty2  rmb    2                   ;
Duty3  rmb    2                   ;
Duty4  rmb    2                   ;
Duty5  rmb    2                   ;
Low2   rmb    2                   ;
Low3   rmb    2                   ;
Low4   rmb    2                   ;
Low5   rmb    2                   ;
Signal2       rmb    1                   ;
Signal3       rmb    1                   ;
Signal4       rmb    1                   ;
Signal5 rmb   1                   ;
* Data
D2_Message
```

```
       fcb    8,$80,1,$40,1,$01,1,$02,1,$04,1,$08,1,$01,1,$10,1
Driver2
* Initialization
       ldx    #REGS                ;
       bset   _PACTL,X %10001000   ; set data directions for
       bclr   _PACTL,X %00000100   ; porta.
       bclr   _TMSK1,X %11111111   ; disable the interrupts
       bset   _TFLG1,X %01111000   ; clear flags
       bclr   _TCTL1,X %11111111   ; set all pins low
       ldaa   #%00000000           ; disable pins 2-5
       staa   TCTL1                ;
       ldab   #0                   ;
       ldx    #Duty2E                     ;
D2_I_L1
       stab   0,X                  ; zero Vars
       inx                         ;
       cpx    #Signal5+1           ;
       bne    D2_I_L1                     ; end of zero Vars loop
       ldx    #D2_Message          ; ****** send message to
       stx    D1MT                 ; ****** LED driver
       ldd    #1000                ; ****** TOC test
       std    Duty2E               ; ****** TOC test
       std    Duty3E               ; ******
       std    Duty4E               ; ******
       std    Duty5E               ; ******
       ldx    #REGS                ;
* Driver Main Loop
D2_ML  ldd    Duty2E               ; first check to see which
       cpd    #0                   ; Servos are turned on
       bne    SetDuty2             ; (DutyXE!=0).
       bclr   _TCTL1,X %11000000   ; disable pins
       bclr   _TMSK1,X BIT6        ; disable interrupt
D2_L1  ldd    Duty3E               ;
       cpd    #0                   ;
       bne    SetDuty3             ;
       bclr   _TCTL1,X %00110000   ;
       bclr   _TMSK1,X BIT5        ;
D2_L2  ldd    Duty4E               ;
       cpd    #0                   ;
       bne    SetDuty4             ;
       bclr   _TCTL1,X %00001100   ;
       bclr   _TMSK1,X BIT4        ;
D2_L3  ldd    Duty5E               ;
       cpd    #0                   ;
       bne    SetDuty5             ;
       bclr   _TCTL1,X %00000011   ;
       bclr   _TMSK1,X BIT3        ;
D2_L4  bra    D2_L5                ;
SetDuty2
       ldd    #$9C40               ; Eclks in a 50Hz period
       subd   Duty2E               ; Subtract the duty cycle
       sei                         ; disable interrupts
       std    Low2                 ; store low cycle duration
       ldd    Duty2E               ;
       std    Duty2                ; store duty cycle duration
       cli                         ; enable interrupts
       bset   _TCTL1,X BIT6        ;
       bset   _TMSK1,X BIT6        ; clear local mask
       bra    D2_L1                ;
SetDuty3
       ldd    #$9C40               ; Eclks in a 50Hz period
       subd   Duty3E               ; Subtract the duty cycle
       sei                         ; disable interrupts
```

```
        std     Low3                    ; store low cycle duration
        ldd     Duty3E                  ;
        std     Duty3                   ; store duty cycle duration
        cli                             ; enable interrupts
        bset    _TCTL1,X BIT4
        bset    _TMSK1,X BIT5           ; clear local mask
        bra     D2_L2                   ;
SetDuty4
        ldd     #$9C40                  ; Eclks in a 50Hz period
        subd    Duty4E                  ; Subtract the duty cycle
        sei                             ; disable interrupts
        std     Low4                    ; store low cycle duration
        ldd     Duty4E                  ;
        std     Duty4                   ; store duty cycle duration
        cli                             ; enable interrupts
        bset    _TCTL1,X BIT2
        bset    _TMSK1,X BIT4           ; clear local mask
        bra     D2_L3                   ;
SetDuty5
        ldd     #$9C40                  ; Eclks in a 50Hz period
        subd    Duty5E                  ; Subtract the duty cycle
        sei                             ; disable interrupts
        std     Low5                    ; store low cycle duration
        ldd     Duty5E                  ;
        std     Duty5                   ; store duty cycle duration
        cli                             ; enable interrupts
        bset    _TCTL1,X BIT0
        bset    _TMSK1,X BIT3           ; clear local mask
        bra     D2_L4                   ;
D2_L5   jmp     D2_ML                   ;
* Driver2 ISR's
TOC2_ISR
        ldx     #REGS                   ;
        bset    _TFLG1,X BIT6           ; clear flag
        ldaa    Signal2                         ; load the signal state
        cmpa    #0                      ; if it's 0 then set pin high
        beq     TOC2_ISR_HIGH           ; for the duty duration
        ldaa    #0                      ; Otherwise set it low
        staa    Signal2                         ; to complete a 50Hz wave.
        ldd     TCNT                    ;
        addd    Low2                    ;
        std     TOC2                    ;
        bra     TOC2_ISR_END
TOC2_ISR_HIGH
        ldaa    #1                      ;
        staa    Signal2                         ;
        ldd     TCNT                    ;
        addd    Duty2                   ;
        std     TOC2                    ;
TOC2_ISR_END
        rti
*
TOC3_ISR
        ldx     #REGS                   ;
        bset    _TFLG1,X BIT5           ; clear flag
        ldaa    Signal3                         ; load the signal state
        cmpa    #0                      ; if it's 0 then set pin high
        beq     TOC3_ISR_HIGH           ; for the duty duration
        ldaa    #0                      ; Otherwise set it low
        staa    Signal3                         ; to complete a 50Hz wave.
        ldd     TCNT                    ;
        addd    Low3                    ;
        std     TOC3                    ;
```

```
      bra    TOC3_ISR_END
TOC3_ISR_HIGH
      ldaa   #1                ;
      staa   Signal3                  ;
      ldd    TCNT              ;
      addd   Duty3             ;
      std    TOC3              ;
TOC3_ISR_END
      rti
*
TOC4_ISR
      ldx    #REGS             ;
      bset   _TFLG1,X BIT4     ; clear flag
      ldaa   Signal4                  ; load the signal state
      cmpa   #0                ; if it's 0 then set pin high
      beq    TOC4_ISR_HIGH     ; for the duty duration
      ldaa   #0                ; Otherwise set it low
      staa   Signal4                  ; to complete a 50Hz wave.
      ldd    TCNT              ;
      addd   Low4              ;
      std    TOC4              ;
      bra    TOC4_ISR_END
TOC4_ISR_HIGH
      ldaa   #1                ;
      staa   Signal4                  ;
      ldd    TCNT              ;
      addd   Duty4             ;
      std    TOC4              ;
TOC4_ISR_END
      rti
*
TOC5_ISR
      ldx    #REGS             ;
      bset   _TFLG1,X BIT3     ; clear flag
      ldaa   Signal5                  ; load the signal state
      cmpa   #0                ; if it's 0 then set pin high
      beq    TOC5_ISR_HIGH     ; for the duty duration
      ldaa   #0                ; Otherwise set it low
      staa   Signal5                  ; to complete a 50Hz wave.
      ldd    TCNT              ;
      addd   Low5              ;
      std    TOC5              ;
      bra    TOC5_ISR_END
TOC5_ISR_HIGH
      ldaa   #1                ;
      staa   Signal5                  ;
      ldd    TCNT              ;
      addd   Duty5             ;
      std    TOC5              ;
TOC5_ISR_END
      rti
* End of Driver2 Code


Message fcc  @ hello Victoria! @
      fcb    EOS
Driver3      jsr    InitSCI
      ldx    #LEDbp1
      ldaa   0,X
d3_ml staa   $4000
      ldaa   0,X
      inx
      pshx
```

```
      ldx    #Message
      jsr    OutStr
      pulx
      psha
      jsr    InChar
      pula
      bra    d3_ml

** UTILITIES **
********************************************************************
*
* Subroutine:  Delay
* Input:       None
* Output:      Provides a delay by simple looping
* Destroys:    None
* Note:        If you're looking to save memory, this function
*              may be rewritten or subsumed by Process since
*              Process is the only routine to call it.
********************************************************************
*
Delay   PSHA            ;
        PSHB            ;
        PSHX            ; Save registers
        PSHY            ;
        LDX    #00002   ; Load outer loop counter

Outer   LDY    #10000   ; Load inner loop counter
Inner   LDD    TCNT
        DEY             ; Decrement inner counter
        BNE    Inner    ; Branch if >0 to inner loop
        DEX             ; Decrement outer counter
        BNE    Outer    ; Branch if >0 to outer loop
        PULY            ; Restore registers
        PULX            ;
        PULB            ;
        PULA            ;
        RTS             ; Return from subroutine
*
*
************************************************************************
*                        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
        CLR    SCCR1    ; Set SCI Mode to 1 start bit,
*                       ;     8 data bits, and 1 stop bit.
        LDAA   #$0C     ; Enable SCI Transmitter
        STAA   SCCR2    ;     and Receiver
        PULA            ; Restore A register
        RTS             ; Return from subtoutine
*
************************************************************************
*                        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       ; Register A ==> SCI data
        PULB              ; Restore B register
        RTS               ; Return from subtoutine
*
************************************************************************
*                        SUBROUTINE -  OutStr2
* 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.  In order to print the
*          string properly with RTI, it automatically disables and
*          enables interrupts.
*
* Input         : Starting location of the string to be transmitted
*               : (passed in X register)
* Output        : Prints the string.
* Destroys      : Contents of X register.
* Calls         : OutChar.
************************************************************************
*
OutStr2 PSHA              ; Save contents of A register
        SEI               ; Disable interrupts
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               ; Increment index
        BRA    Loop2      ; Branch to Loop2 for the next char.
Done    CLI               ; Enable interrupts
        PULA              ; Restore A register
        RTS               ; Return from subtoutine
*
************************************************************************
*                        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
Loop_   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                ; Increment index
        BRA    Loop_       ; 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  LDAA   SCSR      ; Check status reg.
*                        ;     (load it into A reg)
        ANDA   #$20      ; Check if receive buffer full
        BEQ    InChar    ; Wait until data present
        LDAA   SCDR      ; SCI data ==> A register
        RTS              ; Return from subroutine
*

***********************************************************************
                   END OF CODE
***********************************************************************
```

Eveready Multiple Load

Torque	Volts	Amps	Watts
189.8	6.45	1.4	9.03
175.2	6.55	1.26	8.253
160.6	6.53	1.21	7.9013
146	6.63	1.07	7.0941
131.4	6.7	0.96	6.432
116.8	6.75	0.89	6.0075
102.2	6.83	0.77	5.2591
87.6	6.93	0.64	4.4352
73	7.09	0.48	3.4032
58.4	7.41	0.18	1.3338
43.8	7.45	0.15	1.1175

Eveready Constant Load

Torque	Volts	Amps	Watts	Time(min)
160.6	6.46	1.2	7.752	0
160.6	6.28	1.19	7.4732	1
160.6	6.23	1.16	7.2268	2
160.6	6.14	1.16	7.1224	3
160.6	6.04	1.16	7.0064	4
160.6	4.58	1.02	4.6716	5
160.6	4.44	1.01	4.4844	6

160.6	4.05	1.06	4.293	7
160.6	3.24	1	3.24	8
160.6	2.71	0.64	1.7344	9

Vinic Multiple Load

Torque	Volts	Amps	Watts
189.8	6.8	1.38	9.384
175.2	6.5	1.31	8.515
160.6	6.52	1.19	7.7588
146	6.45	1.12	7.224
131.4	6.46	0.97	6.2662
116.8	6.33	0.9	5.697
102.2	5.37	0.71	3.8127
87.6	5.43	0.62	3.3666
73	5.54	0.5	2.77
58.4	5.79	0.27	1.5633
43.8	6.07	0.05	0.3035

Vinic Constant Load

Torque	Volts	Amps	Watts	Time(min)
160.6	4.99	1.05	5.2395	0
160.6	4.97	1.06	5.2682	0.25
160.6	4.97	1.06	5.2682	0.5
160.6	4.94	1.03	5.0882	1.5
160.6	4.82	1.1	5.302	2.5
160.6	4.76	1.11	5.2836	3.5
160.6	4.71	1.09	5.1339	4.5
160.6	4.61	1.1	5.071	5.5
160.6	4.05	0.99	4.0095	6.5
160.6	3.15	1.02	3.213	7.5
160.6	3.1	1.03	3.193	8.5
160.6	3	1.01	3.03	9.5
160.6	2.86	0.87	2.4882	10.5