

# **Rotator Controller Software**

**NASA  
Wallops Flight Facility  
Steven A. Bailey  
February 10, 1997**

**bailey@osb1.wff.nasa.gov**

# Table of Contents

<b>Introduction</b>	<b>3</b>
<b>Requirements</b>	<b>3</b>
<b>Design Philosophy</b>	<b>3</b>
<b>Design Structure</b>	<b>4</b>
<b>Timing</b>	<b>5</b>
<b>Code Description</b>	<b>6</b>
<i>rotator.asm</i>	7
<i>stejim19.c</i>	9
<b>Memory Map</b>	<b>14</b>
<b>Packets</b>	<b>15</b>
<i>uplink packet</i>	15
<i>downlink packet</i>	16
<b>Source Code</b>	<b>19</b>
<i>stejim19.c</i>	19
<i>stdio.h</i>	44
<i>io8096.h</i>	45
<i>string.h</i>	47
<i>ctype.h</i>	48
<i>math.h</i>	49
<i>control.h</i>	50
<i>rotator.asm</i>	51
<i>reg8096.inc</i>	55
<b>Cross Reference</b>	<b>57</b>
<i>rotator.out</i>	57
<b>Support Files</b>	<b>69</b>
<i>make.bat</i>	69
<i>stejim19.lk</i>	69
<b>Compilation</b>	<b>70</b>

## **Introduction**

This document describes software written to operate a custom built balloon rotator controller. This controller was designed and built ‘in house’ by NASA. It consists of off the shelf components chosen for both low power and low temperature operation. The core of this system is the Intel 80C196KB microcontroller. It is the cpu and is directly supported by 64 Kb of eeprom and 32 Kb of ram for program storage and variables. The ‘C’ programming language was used almost entirely for this project. Assembler was only used in select areas where speed was critical.

## **Requirements**

A 50 msec timing interval was determined adequate for control of the balloon rotator. A scheme was needed to provide accurate timing without losing the multitasking capabilities of the system. It was decided to use a hardware timer (within the cpu) to generate an interrupt every 50 msec. This way, all foreground processes could occur without the overhead needed if polling a timer was used instead.

Bi-directional communication to the controller board was also a requirement. To achieve this and maintain a consistent timing interval, an interrupt was assigned to the receiver port for uplink commands. This way, our main timing loop is only ‘interrupted’ when incoming characters are available from an uplink command. Assembler was used for this interrupt loop to keep cpu utilization as low as possible.

Downlink data is processed by a foreground loop in the program. Time not spent in either of the two previous interrupt loops is available to the foreground loop. See Figure 1 for a Data Flow Diagram.

## **Design Philosophy**

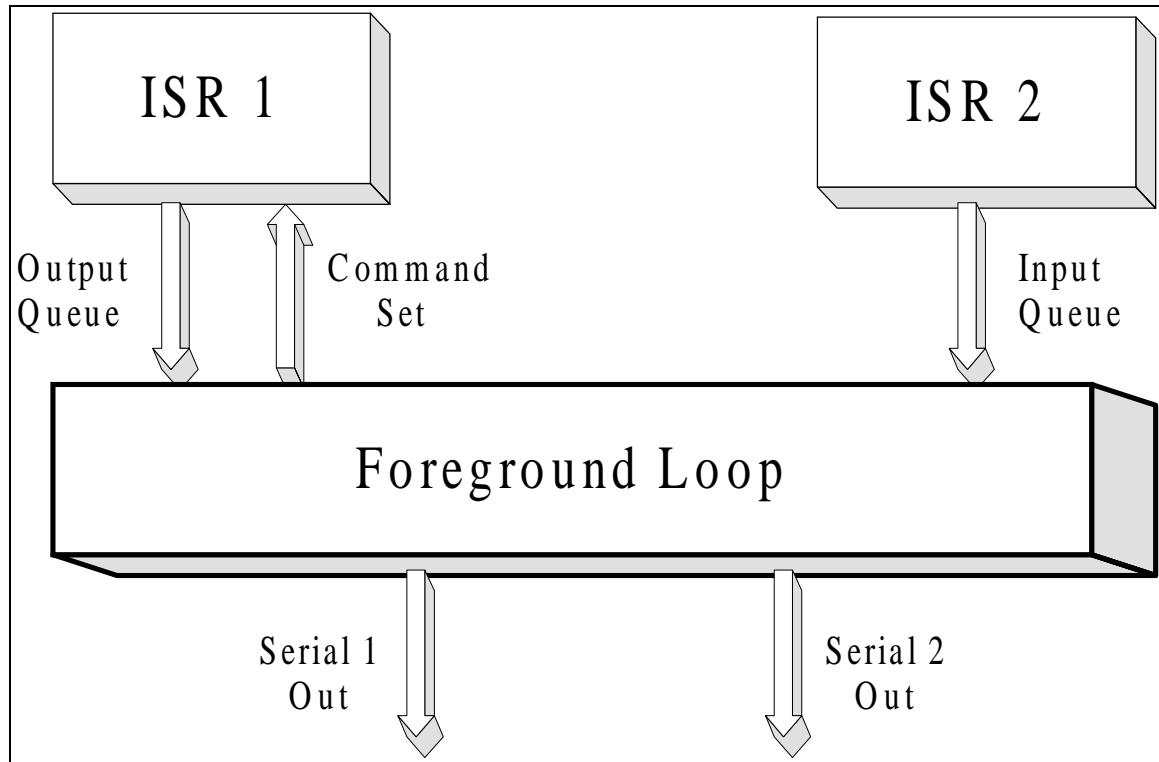
This system needed to be self contained, but access had to be available from a telemetry link. This link enabled commands to be sent to the controller and data from the controller to be sent to the ground. Even though the controller would run autonomously, absolute control was still needed from the ground in event of a catastrophic failure.

It was decided early on that the current ‘health’ of the system would be telemetered to the ground at some period rate. The ‘health’ would be in the form of a binary packet that included such things as sensor readings, control law variables, and general housekeeping data.

Commands telemetered to the system would also be in a binary format. Each binary packet contains a header, a command byte, and a data byte. Total length for an uplink packet is 8 bytes.

## Design Structure

The design of this system is broken down into 3 loops (ISR 1, ISR 2, and FGL). All 3 loops run asynchronously. ISR 1 occurs every 50 msec and is responsible for timing, data acquisition, and balloon control. This loop is the heart of the system. ISR 2 is another interrupt loop solely called when uplink commands are received by the controller. Finally, FGL is our foreground loop that monitors the 2 queues.



**Figure 1. Data Flow Diagram**

ISR 1 performs 4 functions every interrupt or every 50 msec. First, it strobes the external watchdog timer found on the controller board. The watchdog timer has a timeout period of 1.6 seconds. If it is not strobed within that time, it causes a system reset. Essentially, the rotator software does a restart or ‘warm boot’. Even though care has been taken in designing this software, the program could possibly ‘lockup’ for some unknown reason. In this unlikely event, the watchdog timer will timeout and the program will be reset.

The next function of ISR 1 is to read the rate sensor. This is needed for the control algorithm designed by Lanzi. 8 solar sensors are also read every 50 msec. Finally, the control algorithm is called and the appropriate PWM is applied to the rotator motor.

Every 20 interrupts or once per second, ISR 1 does the following data acquisition. It reads 3 temperature channels, 2 voltage monitors, 1 current monitor, and 1 heater status word. Heater control also occurs at this 1 Hz rate. Finally, the circular output queue is filled with 1 record every second. See Figure 2 for a flowchart of ISR 1.

ISR 2 maintains the input queue. When a character is received by the controller, an interrupt occurs and calls this ISR. This ISR reads this character and places it in a circular input queue. Nothing else occurs in this routine so minimum cpu time is used. To ensure high efficiency, this routine is written in assembler. See Figure 3 for a flowchart of ISR 2.

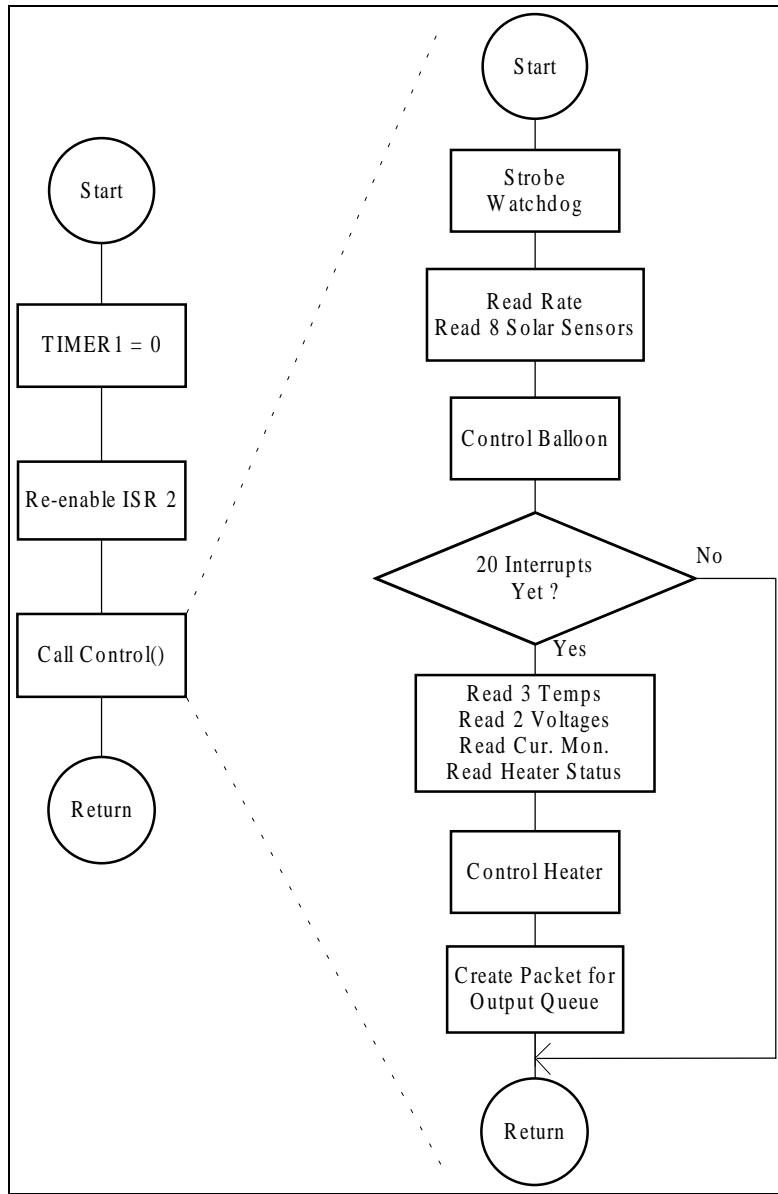
FGL is the foreground loop. Not only are the 2 queues serviced here, but this routine is the starting point of the main program. When a power up or reset of the controller is made, the FGL is called first. The FGL initializes all hardware, buffers, interrupts, and variables. Once this is complete, the FGL enters a loop where the 2 queues are serviced.

The input queue contains raw characters received from ISR 2. These characters are assembled into packets by the FGL and then processed. The FGL checks for bad or incomplete packets and throws them away. When a packet is deemed good, the FGL performs the appropriate action. This includes setting variables, controlling hardware directly, or causing a system reset. At present, there are 30 commands the FGL processes.

The output queue contains packets of data that have been formatted by ISR 1. The FGL simply checks to see if a packet is completely written. When it is written, the FGL is responsible for sending that packet out 2 serial ports. Currently, the low-rate serial port operates at 2400 baud and expects data at 12 records or packets per minute. The high-rate serial port operates at 1200 baud and expects data at 60 records or packets per minute. See Figure 4 for a flowchart of the FGL.

## Timing

The average cpu utilization of ISR 1 is 27.2%. This equates to 13.6 msec. out of every 50 msec. is used by ISR 1. Since ISR 2 is only called when uplink commands are received, I can only estimate its utilization. If a balloon flight occurs with no real problems, ISR 2 should use less than 1% of the cpu time. Assuming both serial ports are being written to at the aforementioned rates, the FGL will use on average 40% of the cpu resources. The total utilization is approximately 68%. I believe this leaves adequate ‘breathing room’ for future changes and additions to the software.



**Figure 2. ISR 1 Flowchart**

## Code Description

The source code for this program is found in 2 files. ‘stevjim18.c’ contains almost all of the code which is written in ‘C’. A very small portion of the code is found in an assembler file called ‘rotator.asm’. I will begin by describing ‘rotator.asm’.

## *rotator.asm*

**Assembler Directives** - this is where assembler specific directives are found. These are default directives for the ‘a8096’ assembler, so there is no need to change them.

**Register Definitions** - all system registers and user defined registers are found in the file ‘reg8096.inc’. This is simply an include file. Total space used for these registers is ‘2fh’ bytes.

**Equates & Externals** - the three ‘C’ functions ( ‘main’, ‘control’, and ‘SEG\_INIT\_L00’ ) are stated here so this assembler file has access to them. The two assembler created variables ( ‘Rcv\_Ptr’ and ‘Rcv\_Buffer’ ) are made public here so the ‘C’ file has access to them.

**Internal Ram Segment** - this section defines a 2 Kb ‘C’ stack and also defines an 80 byte serial port receive buffer in the ram space. Remember, ram space starts at C000h.

**Vectors & Chip Configuration Registers** - this section initializes memory locations which have special purposes. This particular memory falls between 2000h - 207fh. Locations 2000h - 2013h are the lower 10 interrupt vectors. Only 1 interrupt vector is activated here ( ‘Sftwr\_timer’ ). This vector contains the address of the function ‘Software\_Timer\_ISR’ found later in this file. All other vectors are initialized to the ‘Error\_ISR’ address. Any spurious calls to this function simply returns to the caller.

Locations 2030h - 203f are the upper 8 interrupt vectors. Only 1 interrupt vector is activated here ( ‘Ri’ ). This vector contains the address of the function ‘Rx\_ISR’ found later in this file. All other vectors are initialized to the ‘Error\_ISR’ address. Any spurious call to this function simply returns to the caller.

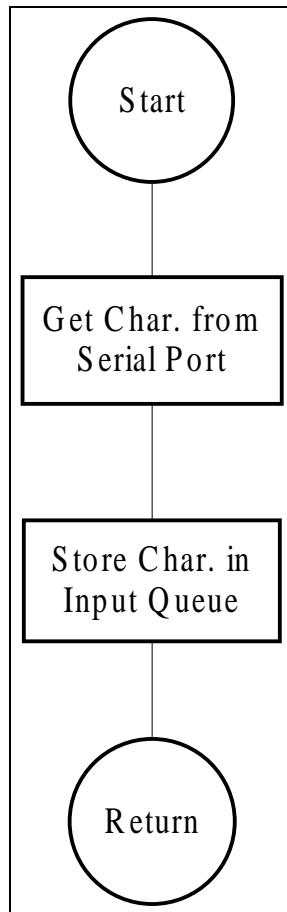
Locations 2040h - 2045h are the Software Version Date calculated by the assembler. Location 2018h is the chip configuration register and it is needed. Location 201ah - 201bh is a required word for this cpu.

**Code Segment & Reset Entry Point** - this section contains the entry point when the system powers up or is reset. Four things occur in this section. The stack pointer is initialized to the 2 Kb stack. All ‘C’ variables are initialized via the system function ‘SEG\_INIT\_L00’. The receive serial port pointer ( ‘Rcv\_Ptr’ ) is initialized to the top of the receive serial port buffer ( ‘Rcv\_Buffer’ ). Finally, the ‘C’ function ‘main()’ is called.

**Serial Receive ISR 2** - The sole purpose of this interrupt service routine is to grab a currently waiting character from the receive buffer ( ‘SBUF’ ) internal to the 80C196KB cpu. This character is then put in a circular queue. It is the job of the ‘C’ foreground program to remove characters from this queue. See Figure 3 for a flowchart of ISR 2.

**Error ISR** - this section simply contains a ‘ret’ opcode. Any call to this function will return to the caller without changing anything. This function is the default address of all interrupt vectors not being used.

**Software Timer ISR 1**- this interrupt service routine is called every 50 msec by a hardware timer called TIMER1. This routine resets TIMER1 to 0 so it can start counting up again for the next interrupt. Also, ISR 2 is re-enabled so received characters are not lost while doing data acquisition. Finally, a call is made to the ‘C’ function ‘control()’ where data acquisition takes place. See Figure 2 for a flowchart of ISR 1.



**Figure 3. ISR 2 Flowchart**

The following is a description of the source code found in file ‘stejim19.c’.

### ***stejim19.c***

**Memory Mapped Addresses** - these 5 addresses are for external devices. The first device ‘READ\_ATOD’ is the address of the 12bit A/D converter. Only the read address is needed because all other interaction with the A/D is done via port signals from the cpu. The remaining 4 addresses are for the PIA ( 8255 ) chip. This chip is a peripheral controller. It is needed because there are not enough controller lines available on the cpu. The PIA chip has 3 ports ( A, B, C ) of 8 bits each. The ‘PIA\_CONTROL’ address is used to set up direction and mode for each port.

**82050 Addresses** - these addresses are used for setup, control, and use of the external uart ( 82050 ) used for serial communication. This uart is our low speed port. Only data is written to this port. No data will be received here. Our high speed serial port is found on the 80C196KB cpu.

**Coefficients** - these 6 coefficients ( $a_0 - a_5$ ) are used to generate a 5<sup>th</sup> order polynomial. This polynomial is used to calibrate the termistors used for temperature monitoring and control. This polynomial is of the form:

$$\text{deg. } C = a_0 + a_1v + a_2v^2 + a_3v^3 + a_4v^4 + a_5v^5 \quad \text{where: } v = \text{voltage}$$

**General Purpose Defines** - these defines are pretty much self explanatory. They are used throughout the program for various purposes.

**Incoming Command Defines** - these 30 defines are used as command identifiers of incoming packets received from the ground station. A one line description is found in the source code.

**Port0 Defines** - these defines refer to port 0 of the cpu. This port has 8 lines which are analog inputs to the internal 10-bit A/D converter. Currently, only 5 lines are being used. These defines have the A/D line number and **NOT** the actual binary line number.

**Port1 Defines** - these defines refer to port 1 of the cpu. This port has 8 lines which are configured as TTL outputs. The lower 3 lines ( P1.0 - P1.2 ) are used as channel select lines for the temperature sensor MUX and the sun sensor MUX.

**Port2 Defines** - this single define refers to port 2 of the cpu. This particular line is configured as a TTL input.

**PIA Port A Defines** - these defines refer to the PIA ( 8255 ) chip port A. This port is configured as TTL output for all 8 lines.

**PIA Port B Defines** - these defines refer to the PIA ( 8255 ) chip port B. This port is configured as TTL output for all 8 lines.

**General Purpose Functions** - this list of 11 functions are general purpose in nature. They are higher level functions.

**I/O Functions** - this list of 16 functions are low level in nature and perform all I/O necessary for data acquisition.

**External Variables** - these 2 variables are created and found in the assembler file ‘rotator.asm’.

**Global Variables** - these are the general purpose variables used by the program. This includes buffers, flags, I/O variables.

**Port Variables** - these 6 variables hold the current states of the ports used for data acquisition.

**main()** - this is our ‘main’ function or foreground loop (FGL). Our program starts in this function. All hardware and variables are initialized here. A loop is then entered where both the input queue and output queue are checked for data availability. When data is available, due course is taken. Input or uplink data is parsed and acted on. Output data is written to the 2 serial ports. See Figure 4 for a flowchart of the FGL.

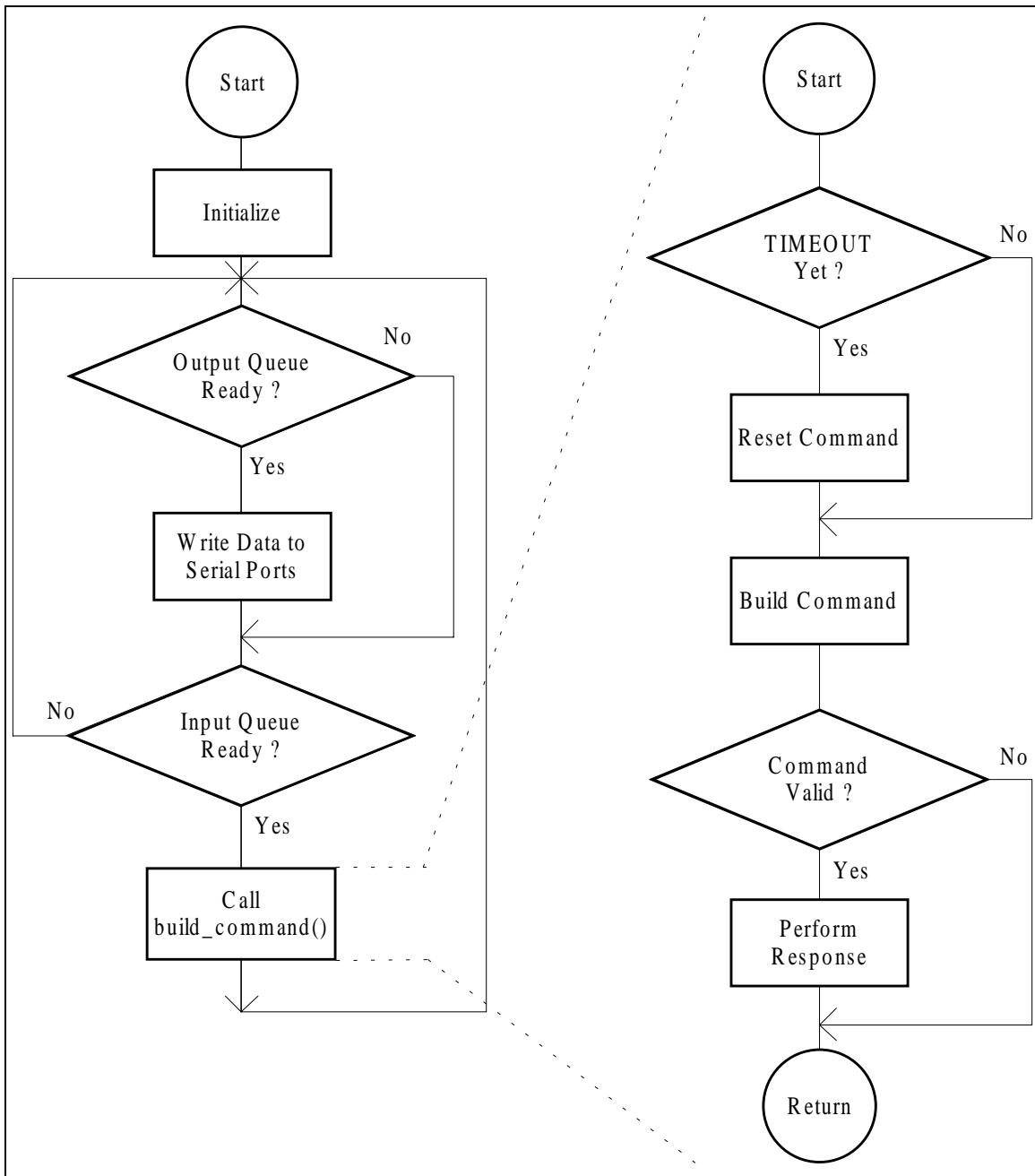
**build\_command( char c )** - this function is called from ‘main()’ with a character pulled out of the input queue. Within this function, this character is added to a ‘Command\_Buffer[]’. If too much time ( TIMEOUT ) has lapsed since the last character was received, the buffer index ( cnt ) is reset. This insures 2 or more different commands are not concatenated.

When the correct number of characters ( 7 ) has been received, this buffer is checked for data integrity. If integrity is good, the buffer is checked via a ‘switch’ statement for the appropriate command matchup. If the command is found within the current list, the appropriate action occurs.

**initialize()** - this function is called once during program startup or during a reset. It is called from ‘main()’. This function initializes all hardware and variables.

**putchar( int c )** - this function is needed by the ‘C’ function ‘printf()’. When ‘printf()’ outputs characters, they are directed to the function ‘putchar( int c )’.

The purpose of this function is to send characters out the internal serial port. This function is not currently used.



**Figure 4. FGL Flowchart**

**putbyte\_82050( UCHAR c )** - this function is solely used to output characters to the external serial port found on the 82050 uart. Only ‘main()’ uses this function.

**putbyte( UCHAR c )** - this function is solely used to output characters to the internal serial port found on the 80C196KB cpu. Only ‘main()’ uses this function.

**read\_heatstat()** - this function is used to read the heater status voltage. The internal 10-bit A/D is used. Currently, a binary voltage is found. Only ‘control()’ uses this function.

**read\_temp( UCHAR channel )** - this function is used to read 1 of 8 temperature channels via the temperature MUX and the internal 10-bit A/D. Only ‘control()’ uses this function.

**read\_volt\_mon( UCHAR voltage )** - this function is used to read 1 of 2 voltages (+5v & +12v) found on the controller board. A 2 pole electronic switch selects the appropriate voltage. The internal 10-bit A/D is used and only ‘control()’ uses this function.

**read\_cur\_mon()** - this function uses the internal 10-bit A/D to read a single current monitor. Only ‘control()’ uses this function.

**read\_lin\_temp()** - this function uses the internal 10-bit A/D to read linear temperature from the heater circuit. Only ‘control()’ uses this function.

**read\_solar\_sen( UCHAR channel )** - this function is used to read 1 of 8 solar sensors via the solar sensor MUX and the external 12-bit A/D. Only ‘control()’ uses this function.

**read\_rate()** - this function is used to read the rate sensor via the external 12-bit A/D. Only ‘control()’ uses this function.

**set\_heater( UCHAR state )** - this function simply turns the heater control line on and off. Only ‘control()’ uses this function.

**set\_brake( UCHAR state )** - this function simply turns the motor brake line on and off. Only ‘build\_command()’ uses this function.

**set\_dir( UCHAR state )** - this function simply changes the direction of the motor by toggling the TTL level of the direction line. Only ‘build\_command()’ uses this function.

**set\_enable( UCHAR state )** - this function simply turns the motor on or off by toggling the TTL level of the enable line. Only ‘initialize()’ uses this function.

**set\_shutdown( UCHAR state )** - this function simply turns the motor current monitor on or off by toggling the TTL level of the shutdown line. This function is not currently used.

**set\_pwm( UCHAR value )** - this function controls the PWM output of the pwm line of the cpu. Three functions use this functions. They are ‘build\_command()’, ‘control()’, and ‘IVLlaw()’.

**pause( int time )** - this function is simply a delay used . It is used throughout the program.

**control()** - this function is the ‘C’ portion of the interrupt service routine 1 ( ISR 1 ). This function is called by the assembly function ‘Software\_Timer\_ISR’ every 50 msec. Both 20 Hz and 1 Hz data acquisition occurs in this function. See Figure 2 for a flowchart of ISR 1.

**conv\_to\_temp( float voltage )** - this function is used to return a calibrated temperature given an uncalibrated input voltage. A 5<sup>th</sup> order polynomial is used in this calculation. See the section called **Coefficients** for a description of this polynomial. Only ‘control()’ uses this function.

**IVLlaw()** - this function performs the inner velocity control law to command the balloon into position by manipulating the motor PWM. Only ‘control()’ uses this function.

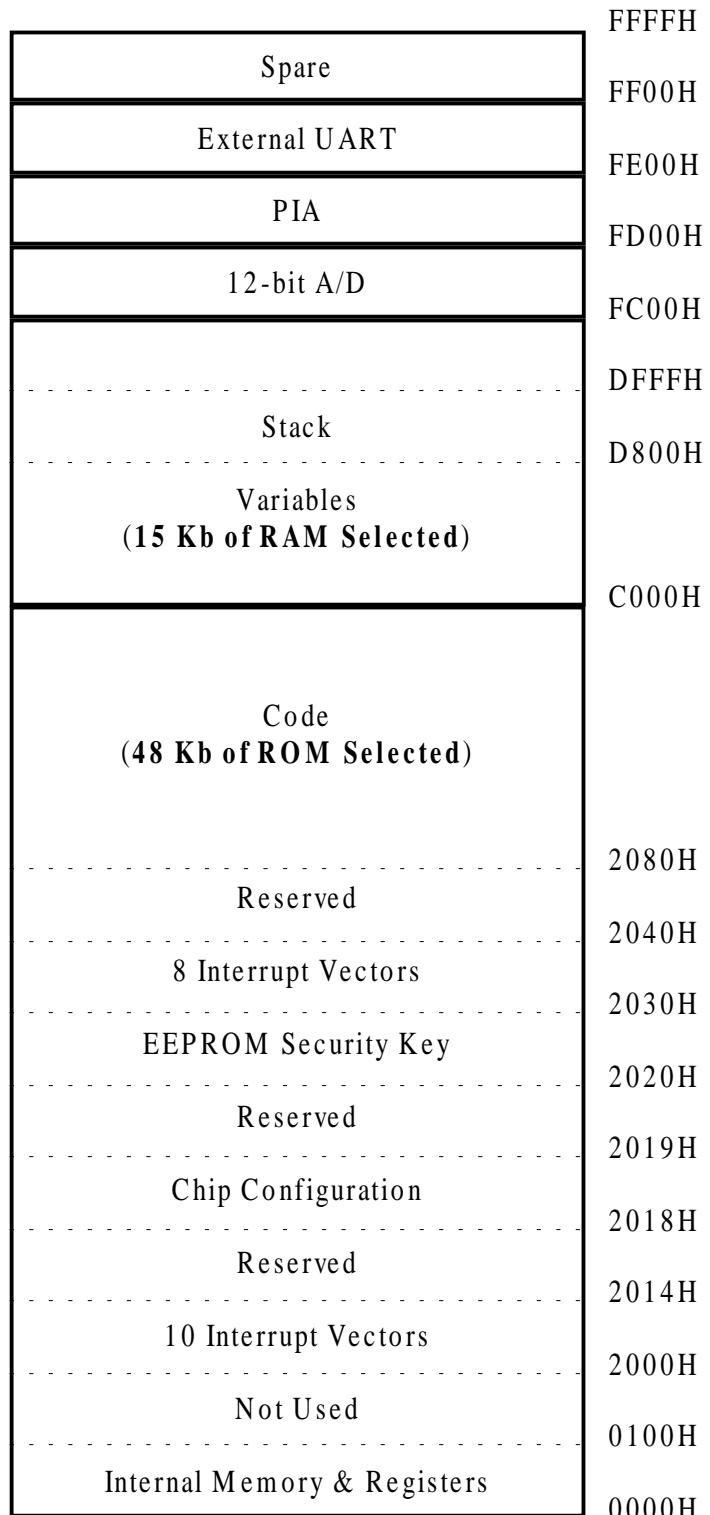
**RatEst()** - this function estimates the rate-of-change for motor PWM. Only ‘control()’ uses this function.

**PosEst()** - this function estimates the current position given data from 4 of 8 solar sensors. Only ‘control()’ uses this function.

**DataList()** - this function is called at a 1 Hz rate from ‘control()’. Its purpose is to place housekeeping data into a circular output queue in a binary packet format. Only ‘control()’ uses this function.

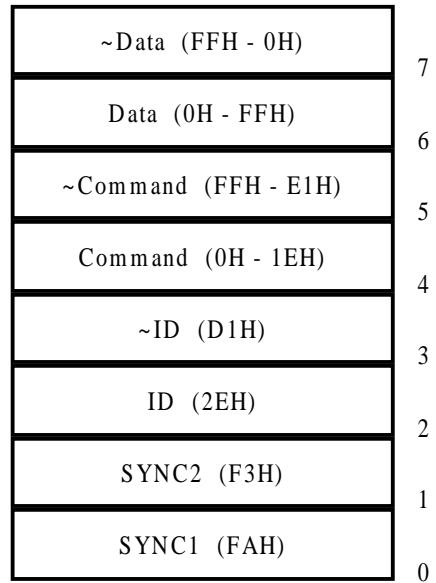
**encode( UCHAR ucData, UCHAR \*cksmData )** - this function is called from ‘DataList()’. Its purpose is to place the given character ( ucData ) into a packet data field. A checksum variable is then updated via a pointer to that variable. Only ‘DataList()’ uses this function.

## Memory Map



## Packets

*uplink packet*



The uplink packet is 8 bytes in length and the ‘~’ symbol means ones complement.

cksmData	46
Unused	45
Commuted Byte	44
ID3 Byte	43
Data	42
Command	41
Commuted Byte	40
ID2 Byte	39
Commuted Byte	38
Commuted Byte	37
ID1 Byte	36
EstRate Word	34
IRSRate Word	32
Integ Word	30
Pos Word	28
8 Solar Words	12
Senstat Byte	11
Mode Byte	10
Port1 Byte	9
PWM Byte	8
Time Word	6
LS Data Length (28H)	5
MS Data Length (0H)	4
ID (26H)	3
(20H)	2
SYNC2 (F3H)	1
SYNC1 (FAH)	0

*downlink packet*

The following **pseudocode** shows how to decode the downlink packet. Remember to ‘cast’ downlink\_packet elements to the appropriate data type when decoding.

```

Time      = (downlink_packet[6] << 8) + downlink_packet[7];
PWM       = downlink_packet[8];
Port1     = downlink_packet[9];
Mode      = downlink_packet[10];
Senstat   = downlink_packet[11];
Solar0    = (downlink_packet[12] << 8) + downlink_packet[13];
Solar1    = (downlink_packet[14] << 8) + downlink_packet[15];
Solar2    = (downlink_packet[16] << 8) + downlink_packet[17];
Solar3    = (downlink_packet[18] << 8) + downlink_packet[19];
Solar4    = (downlink_packet[20] << 8) + downlink_packet[21];
Solar5    = (downlink_packet[22] << 8) + downlink_packet[23];
Solar6    = (downlink_packet[24] << 8) + downlink_packet[25];
Solar7    = (downlink_packet[26] << 8) + downlink_packet[27];
Pos       = (((downlink_packet[28] << 8) + downlink_packet[29])-32768) / 182.03;
Integ    = (((downlink_packet[30] << 8) + downlink_packet[31])-32768) / 128.0;
IRSRate  = (((downlink_packet[32] << 8) + downlink_packet[33])-32768) / 1310.68;
EstRate  = (((downlink_packet[34] << 8) + downlink_packet[35])-32768) / 1310.68;

ID1      = downlink_packet[36];

switch( ID1 ) {
    case 0:
        T1 = downlink_packet[37] / 2.086 - 56.6;
        break;
    case 1:
        T2 = downlink_packet[37] / 2.086 - 56.6;
        break;
    case 2:
        T3 = downlink_packet[37] / 2.086 - 56.6;
        break;
    case 3:
        V5 = downlink_packet[37] / 51.0;
        break;
    case 4:
        V12 = downlink_packet[37] / 51.0;
        break;
    case 5:
        Current = downlink_packet[37] / 51.0;
        break;
    case 6:
        Heat_min = downlink_packet[37] / 2.086 - 56.6;
        break;
    case 7:
        Heat_max = downlink_packet[37] / 2.086 - 56.6;
        break;
    case 8:
        Heat_stat = downlink_packet[37] / 51.0;
        break;
}

switch( (ID1 + 5) % 9 ) {
    case 0:
        T1 = downlink_packet[38] / 2.086 - 56.6;
        break;
    case 1:
        T2 = downlink_packet[38] / 2.086 - 56.6;
        break;
    case 2:
        T3 = downlink_packet[38] / 2.086 - 56.6;
        break;
    case 3:
        V5 = downlink_packet[38] / 51.0;
        break;
    case 4:
        V12 = downlink_packet[38] / 51.0;
        break;
    case 5:
        Current = downlink_packet[38] / 51.0;
        break;
}

```

```

        case 6:
            Heat_min = downlink_packet[38] / 2.086 - 56.6;
            break;
        case 7:
            Heat_max = downlink_packet[38] / 2.086 - 56.6;
            break;
        case 8:
            Heat_stat = downlink_packet[38] / 51.0;
            break;
    }

    ID2 = downlink_packet[39];

    switch(ID2) {
        case 0:
            Kp = downlink_packet[40] / 51.0;
            break;
        case 1:
            Ki = downlink_packet[40] / 25.5;
            break;
        case 2:
            Kd = downlink_packet[40] / 4.25;
            break;
        case 3:
            Clip = downlink_packet[40] / 10.2;
            break;
        case 4:
            PWM_lim = downlink_packet[40];
            break;
        case 5:
            PWM_off = downlink_packet[40];
            break;
        case 6:
            Pos_c = (downlink_packet[40] - 90.0) / 0.5;
            break;
        case 7:
            Rate_c = (downlink_packet[40] - 128.0) / 12.7;
            break;
        case 8:
            Dark_Awake = downlink_packet[40];
            break;
        case 9:
            Flux_min = downlink_packet[40] / 0.255;
            break;
        case 10:
            Fine = downlink_packet[40] / 100.0;
            break;
    }

    Cur_command = downlink_packet[41];
    Cur_data = downlink_packet[42];

    ID3 = downlink_packet[43];

    switch( ID3 ) {
        case 0:
            T4 = downlink_packet[44] / 2.086 - 56.6;
            break;
        case 1:
            T5 = downlink_packet[44] / 2.086 - 56.6;
            break;
        case 2:
            T6 = downlink_packet[44] / 2.086 - 56.6;
            break;
        case 3:
            T7 = downlink_packet[44] / 2.086 - 56.6;
            break;
        case 4:
            T8 = downlink_packet[44] / 2.086 - 56.6;
            break;
    }

    Unused = downlink_packet[45];
    CksmData = downlink_packet[46];

```

## Source Code

### *stejim19.c*

```
/*-----
*          Stejim19.c
*-----*/
*
*          NASA
*          Wallops Flight Facility
*          Steven A. Bailey
*          Jim Lanzi
*          February 7, 1997
*          Version 1.9
*/
#include <stdio.h>
#include <io8096.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <control.h>

/*-----
* These are memory mapped addresses for both the external
* 12bit A/D and the 8255A Pia chip.
*-----
*/
#define READ_ATOD      (* (unsigned char *)    ( 0xfc00 ))
#define PIA_PORTA      (* (unsigned char *)    ( 0xfd00 ))
#define PIA_PORTB      (* (unsigned char *)    ( 0xfd01 ))
#define PIA_PORTC      (* (unsigned char *)    ( 0xfd02 ))
#define PIA_CONTROL    (* (unsigned char *)    ( 0xfd03 ))

/*-----
* This set of defines are for the external Uart 82050
*-----
*/
#define TXD           (* (unsigned char *)    ( 0xfe00 ))
#define RXD           (* (unsigned char *)    ( 0xfe00 ))
#define BAL           (* (unsigned char *)    ( 0xfe00 ))
#define BAH           (* (unsigned char *)    ( 0xfe01 ))
#define IER           (* (unsigned char *)    ( 0xfe01 ))
#define IIR           (* (unsigned char *)    ( 0xfe02 ))
#define LCR           (* (unsigned char *)    ( 0xfe03 ))
#define MCR           (* (unsigned char *)    ( 0xfe04 ))
#define LSR           (* (unsigned char *)    ( 0xfe05 ))
#define MSR           (* (unsigned char *)    ( 0xfe06 ))
#define SCR           (* (unsigned char *)    ( 0xfe07 ))

/*-----
* This is a set of 6 coefficients used for thermistor
* calibration. These are used to generate a 5th order
* polynomial.
*-----
*/
#define A0            -62.2573
#define A1            116.8275
#define A2            -98.9833
#define A3             48.0809
#define A4            -11.0588
#define A5             0.9729
```

```

/*-----
 * General purpose defines
 *-----
 */

#define CR          0x0c      /* Carriage return */
#define LF          0x0a      /* Line feed */
#define SPACE       0x20      /* Space character */
#define BUFF_SIZE   80        /* Max command buffer size */
#define PACKET_SIZE 47        /* Size of Packet */
#define DATA_SIZE   100       /* Room for 100 packets */
#define MAXTIME    100        /* Max time to wait */
#define FULL         0         /* Output buffer full */
#define EMPTY        1         /* Output buffer empty */
#define ULONG        unsigned long /* Shorten data types */
#define UINT         unsigned int /* Shorten data types */
#define UCHAR        unsigned char /* Shorten data types */
#define VOLTS_5     0          /* Select 5 volts to monitor */
#define VOLTS_12    1          /* Select 12 volts to monitor */
#define OFF          0          /* OFF toggle for various devices */
#define ON           1          /* ON toggle for various devices */
#define CW           0          /* Clockwise direction */
#define CCW          1          /* Counter Clockwise direction */
#define SYNC1        0xfa      /* Sync 1 byte */
#define SYNC2        0xf3      /* Sync 2 byte */
#define ID           0x2e      /* ID byte */
#define TIMEOUT     100        /* 20 tics/sec * 5 = 5 seconds */
#define C12BIT_5V   0.001221001 /* 12 Bit rescale 5 / 4095 */
#define C10BIT_5V   0.004887585 /* 10 Bit rescale 5 / 1023 */
#define ZERORATE    2.43       /* Zero calibration coef. for rate */

/* Absolute value macro */
#define ABS( A ) ( (A) < (0) ? -(A) : (A) )

/*-----
 * Incoming command defines
 *-----
 */

#define MODE        0          /* Controls 6 (6-bits used) functions */
#define SENSOR      1          /* Controls 8 (8-bits used) solar sensors */
#define FDBK        2          /* Turns control algorithm ON or OFF */
#define RATE         3          /* Turns estimated rate ON or OFF */
#define ASLP         4          /* Turns auto-sleep ON or OFF */
#define PWM          5          /* Set PWM level */
#define DIR          6          /* Set motor direction CW or CCW */
#define BRAKE       7          /* Turn motor brake ON or OFF */
#define KP           8          /* Set proportional gain */
#define KI           9          /* Set integral gain */
#define KD          10          /* Set derivative gain */
#define CLIP         11         /* Set a rate limit for control algorithm */
#define INTEG       12          /* Reset integrator to 0 */
#define PWM_OFF     13          /* Set PWM offset */
#define PWM_LIMIT   14          /* Set PWM limit */
#define POS_C        15          /* Commanded position */
#define STOP         16          /* Stop motor and stop control algorithm */
#define RESET       17          /* Reset program...causes WDT timeout */
#define CRAT        18          /* Turn commanded rate ON or OFF */
#define RATE_C      19          /* Set commanded rate */
#define FLUX_MIN    20          /* Flux level needed for 'awake' */
#define HELP1       21          /* Not used here */
#define HELP2       22          /* Not used here */
#define FILT         23          /* Turn motor filter ON or OFF */
#define HEAT        24          /* Turn heater circuit ON or OFF */
#define HEAT_MIN    25          /* Thermostat min. temp */
#define HEAT_MAX    26          /* Thermostat max. temp */
#define ID1          27          /* Choose particular commutated channel now */
#define ID2          28          /* Choose particular commutated channel now */
#define FINE         29          /* Fine adjustment to commanded position */
#define NTIC        30          /* Set output rate of low-speed port (Hz) */

```

```

/*
 * Port0 defines
 */
#define HEAT_STAT      0          /* AD channel 0 */
#define MUX_TEMP       4          /* AD channel 4 */
#define VOLT_MON       5          /* AD channel 5 */
#define CURRENT_MON    6          /* AD channel 6 */
#define LINEAR_TEMP    7          /* AD channel 7 */

/*
 * Port1 defines
 */
#define SUN_CHIP_SEL   0x08      /* 00001000b */
#define SUN_RATE_SEL   0x10      /* 00010000b */
#define START_CONV     0x20      /* 00100000b */
#define HIGH_LOW_BYTE  0x40      /* 01000000b */
#define MOTOR_DIRECTION 0x80      /* 10000000b */

/*
 * Port2 defines
 */
#define CONV_STAT      0x80      /* 10000000b */

/*
 * Pia porta defines
 */
#define MOTOR_ENABLE    0x01      /* 00000001b */
#define WD_STROBE       0x02      /* 00000010b */
#define GAIN_SEL        0x04      /* 00000100b */
#define TEMP_MUX_SEL   0x08      /* 00001000b */
#define VOLT_MON_SEL   0x10      /* 00010000b */
#define MOTOR BRAKE    0x20      /* 00100000b */
#define MOTOR_SHUTDOWN  0x40      /* 01000000b */

/*
 * Pia portb defines
 */
#define HEATER_CTRL    0x03      /* 00000011b */
#define UART_RESET     0x04      /* 00000100b */
#define UART_RTS       0x08      /* 00001000b */

/*
 * General purpose functions
 */
void initialize();                      /* Initialize hardware */
void build_command( char c );           /* Build command function */
void pause( int time );                /* Pause function */
void init_Uart();                      /* Initialize external 82050 uart */
float conv_to_temp( float voltage );   /* Convert voltage to temp. (deg. C) */
void control();                        /* Main data acquisition control loop */
void IVLlaw();                         /* Inner velocity control law...Lanzi code */
void RatEst();                          /* Rate estimation algorithm...Lanzi code */
void PosEst();                          /* Position estimator...Lanzi code */
void DataList();                        /* Create output housekeeping packet */

/* Calc. checksum, then put in queue */
void encode( UCHAR ucData, UCHAR *cksmData );

```

```

/*
 * I/O functions
 */
UINT read_temp( UCHAR channel );           /* Read temperature sensors...possible 8 */
UINT read_volt_mon( UCHAR voltage );        /* Read voltage monitors...possible 2 */
UINT read_cur_mon();                      /* Read current monitor */
UINT read_lin_temp();                     /* Read linear temp. from heater */
UINT read_solar_sen( UCHAR channel );       /* Read solar sensors...possible 8 */
UINT read_rate();                         /* Read rate sensor */
UINT read_heatstat();                    /* Read heater status */

void set_brake( UCHAR state );            /* Set motor brake 1 = ON, 0 = OFF */
void set_dir( UCHAR state );              /* Set motor direction */
void set_enable( UCHAR state );           /* Set motor enable 1 = ON, 0 = OFF */
void set_shutdown( UCHAR state );          /* Set motor current monitor 1 = ON, 0 = OFF */
void set_pwm( UCHAR value );             /* Set PWM duty cycle 0 = 0%, 128 = 50% */
void set_heater( UCHAR state );           /* Set heater control 1 = ON, 0 = OFF */

void putbyte( UCHAR c );                  /* Output 1 byte to serial port */
void putbyte_82050( UCHAR c );            /* Output 1 byte to 82050 uart port */
int  putchar( int c );                   /* Output 1 byte to serial port for printf */

/*
 * External variables found in 'rotator.asm'
 */
/*-----*/
/* Used to receive raw chars. from */
/* ground station (commands) */
extern char *Rcv_Ptr, Rcv_Buffer[BUFF_SIZE];

/*
 * Global variables. Due to a bug in the Archimedes 'C'
 * compiler, initialization of variables when space is allocated
 * at startup does not work properly. The function 'initialize()'
 * is called immediately after program startup to programmatically
 * initialize all variables.
*/
/*-----*/
/*-----*/
/* Used for input queue */
UCHAR   *Command_Ptr, Command_Buffer[BUFF_SIZE];
/* Used for output queue */
UCHAR   Data_Buffer[DATA_SIZE][PACKET_SIZE];
/* Used for control access to */
UCHAR   Data_Tokens[DATA_SIZE];
/* output queue */

UCHAR   data;                           /* Save last uplink data */
UCHAR   command;                       /* Save last uplink command */
UCHAR   cnt;                            /* Used for building uplink command */
UCHAR   ntic;                           /* Rate (Hz) of low-speed port */

ULONG  new_tics;                      /* Used as timeout variables for */
ULONG  old_tics;                      /* uplink commands */

int    mode;                           /* 6 state (6-bit) function byte */
int    pwm_off;                        /* PWM offset */
int    pwm_lim;                        /* PWM limit */
int    pwm;                            /* PWM initial value */
int    senstat;                        /* 8 sensor enable/disable byte */
int    s[8];                           /* Raw sun sensor data */
int    dark;                            /* Dark flag */
int    awake;                           /* Awake flag */
int    uc0;                            /* Used for motor filter */
int    uf0;                            /* Used for motor filter */
int    krs;                            /* Used for stored position array */
int    fold;                           /* Used to prevent estimator saturation */

UINT   Id1;                            /* Index for 1st commutated byte */
UINT   Id2;                            /* Index for 2nd commutated byte */
UINT   Id3;                            /* Index for 3rd commutated byte */
UINT   outsiz;                         /* Used to index through output queue */

UINT   kps;                            /* Used to calc. rate every 200 ms */
UINT   data_index;                     /* Output queue ISR1 index */
UINT   out_index;                      /* Output queue FGL index */

```

```

UINT    time_index;                      /* Incrementing record timer */

UINT    loop_index;                     /* ISR1 loop timer */
UINT    time_out;                      /* Used for A/D timeout */

float   integ;                         /* Integral of rate error */
float   rate;                          /* Current rate */
float   pos;                           /* Position for control */
float   EstRate;                      /* Estimated rate */
float   IRSRate;                      /* Inertial rate */
float   int_lim;                      /* Integral limit */
float   clip;                          /* Rate limit */
float   pos_c;                        /* Commanded position */
float   rat_cx;                       /* Commanded rate */
float   coarse;                       /* Course adjustment to position */
float   fine;                         /* Find adjustment to position */
float   Kp;                            /* Proportional gain */
float   Ki;                            /* Integral gain */
float   Kd;                            /* Derivative gain */
float   timeout;                      /* 50 msec */
float   V5;                            /* 5 volt storage */
float   V12;                           /* 12 volt storage */
float   Current;                      /* Current storage */
float   flux_min;                     /* Flux level needed for awake */
float   gray_width;                  /* A function of flux_min */
float   heat_min;                     /* Thermostat min. temp. */
float   heat_max;                     /* Thermostat max. temp. */
float   heat_stat;                    /* Voltage of heater relay */
float   pos0;                          /* Current position for fold */

float   Temper[8];                   /* Storage for 8 temp. channels */
float   D[5];                        /* Used for estimated rate filter */
float   p[5];                        /* Used for estimated rate filter */
float   uu[5];                       /* Used for estimated rate filter */
float   ssfac[8];                   /* Sun sensor calibration variables */

/*-----
 * These are port specific variables
 *-----
 */

UCHAR  port0, port1, port2;          /* 80196 CPU port variables */
UCHAR  pia_a, pia_b, pia_c;          /* 8255A PIA port variables */

```

```

/*
 *-----*
 * The main program loop ( FGL - low priority )
 *-----*
 */

void main()
{
    int i;
    UCHAR j = 0;

    initialize();           /* Initialize 80c196 cpu */
    init_Uart();            /* Initialize external 82050 uart */

    while ( 1 ) {           /* This is the foreground loop */
        /* See if packets ready to go out port */
        if ( Data_Tokens[out_index] == FULL ) {

            /* Output every sec. to 82050 */
            for (i=0; i<PACKET_SIZE; i++)
                putbyte_82050( Data_Buffer[out_index][i] );

            if ( ++j >= ntic ) {      /* Output every 5 secs. */
                j = 0;
                for (i=0; i<PACKET_SIZE; i++)
                    putbyte( Data_Buffer[out_index][i] );
            }
        }

        Data_Tokens[out_index] = EMPTY;

        /* Watch for wrap around */
        if ( ++out_index == DATA_SIZE )
            out_index = 0;
    }

    /* Look for commands coming in port */
    while ( Command_Ptr != Rcv_Ptr ) {
        build_command( *Command_Ptr );

        if ( ++Command_Ptr == ( &Rcv_Buffer[0] + BUFF_SIZE ) )
            Command_Ptr = &Rcv_Buffer[0];
    }
}

/*
 *-----*
 * This routine builds a command string from the interrupt receive
 * buffer found in assembler routine Rx_ISR.
 *-----*
 */

void build_command( char c )
{
    /* If time between last char */
    /* and current char >= TIMEOUT, */
    /* then reset counter */

    if ( (new_tics - old_tics) >= TIMEOUT )
        cnt = 0;

    old_tics = new_tics;          /* Save copy of current new_tics */

    Command_Buffer[cnt] = c;

    if ( ++cnt == 8 ) {
        cnt = 0;
        /* Make sure command packet is valid */
        if ( (Command_Buffer[0] == SYNC1) &&
            (Command_Buffer[1] == SYNC2) &&
            (Command_Buffer[2] == ID) &&
            (Command_Buffer[3] == (UCHAR)~ID) &&
            (Command_Buffer[4] == (UCHAR)~Command_Buffer[5]) &&
            (Command_Buffer[6] == (UCHAR)~Command_Buffer[7]) ) {

            command = Command_Buffer[4] - 1; /* Remove Lanzi compatibility */
            data    = Command_Buffer[6];
        }
    }
}

```

```

switch( command ) {
    case MODE:                                /* Controls 6 (6-bits used) functions */
        if ( data <= 63 )
            mode = data;
        break;

    case ID1:                                 /* Choose commutated ID1 byte */
        if ( data <= 8 )
            Id1 = data;
        break;

    case ID2:                                 /* Choose commutated ID2 byte */
        if ( data <= 9 )
            Id2 = data;
        break;

    case SENSOR:                             /* Controls 8 (8-bits used) solar sensors */
        if ( data <= 15 )
            senstat = data;
        break;

    case FDBK:                               /* Turns control algorithm ON or OFF */
        if ( data & 1 ) {
            mode |= FDBK_BIT;
            integ = 0.0;
        }
        else {
            mode &= ~FDBK_BIT;
            set_pwm( 0 );
        }
        break;

    case RATE:                               /* Turns estimated rate ON or OFF */
        if ( data & 1 )
            mode |= RATE_BIT;
        else
            mode &= ~RATE_BIT;
        break;

    case ASLP:                               /* Turns auto-sleep ON or OFF */
        if ( data & 1 )
            mode |= ASLP_BIT;
        else
            mode &= ~ASLP_BIT;
        break;

    case CRAT:                               /* Turn commanded rate ON or OFF */
        if ( data & 1 )
            mode |= CRAT_BIT;
        else
            mode &= ~CRAT_BIT;
        break;

    case FILT:                               /* Turn motor filter ON or OFF */
        if ( data & 1 )
            mode |= FILT_BIT;
        else
            mode &= ~FILT_BIT;
        break;

    case NTIC:                               /* Set output rate of low-speed port (Hz) */
        if ( data > 0 )
            ntic = data;
        break;

    case DIR:                                /* Set motor direction CW or CCW */
        if ( data <= 1 )
            set_dir( data );
        break;

    case BRAKE:                             /* Set motor brake ON or OFF */
        if ( data <= 1 )
            set_brake( data );
        break;
}

```

```

case KP:                                /* Set proportional gain */
    Kp = (float)data / 51.0;
    break;

case KI:                                /* Set integral gain */
    Ki = (float)data / 25.5;
    break;

case KD:                                /* Set derivative gain */
    Kd = (float)data / 4.25;
    break;

case CLIP:                               /* Set rate limit for control algorithm */
    clip = (float)data / 10.2;
    break;

case PWM:                                /* Set PWM level */
    if ( data <= pwm_lim )
        set_pwm( data );
    break;

case PWM_OFF:                            /* Set PWM offset */
    pwm_off = data;
    break;

case PWM_LIMIT:                          /* Set PWM limit */
    pwm_lim = data;
    break;

case POS_C:                             /* Set commanded position */
    if ( data <= 180 ) {
        coarse = ((float)data - 90.0) * 2.0;
        pos_c = coarse + fine;
    }
    break;

case FINE:                               /* Set fine adjustment to commanded pos. */
    if ( data <= 200 ) {
        fine = (float)data * 0.01;
        pos_c = coarse + fine;
    }
    break;

case RATE_C:                            /* Set commanded rate */
    rat_cx = ((float)data - 128.0) / 12.7;
    break;

case FLUX_MIN:                           /* Flux level needed for 'awake' */
    flux_min = ((float)data * 3.9215);
    gray_width = 0.53846 * flux_min;
    break;

case HEAT:                               /* Turns heater circuit ON or OFF */
    if ( data & 1 )
        mode |= HEAT_BIT;           /* Turn ON */
    else
        mode &= ~HEAT_BIT;         /* Turn OFF */
    break;

case HEAT_MIN:                           /* Thermostat min. temp. */
    heat_min = ((float)data / 2.086) - 56.66;
    break;

case HEAT_MAX:                           /* Thermostat max. temp. */
    heat_max = ((float)data / 2.086) - 56.66;
    break;

case INTEG:                             /* Reset integrator */
    integ = 0.0;
    break;

case STOP:                               /* Stop motor and stop control algorithm */
    mode &= ~FDBK_BIT;
    set_pwm( 0 );
    break;

```

```

        case RESET:           /* Cause program reset by letting WDT timeout */
            INT_MASK = 0;
            break;

        default:
            break;
    }
}

/*
 * This routine initializes all registers and variables relative to
 * proper operation of 80196 microprocessor.
 */
void initialize()
{
    int i;

    disable_interrupt();

    /*
     * Below are program variable initializations
     */
    for (i=0; i<DATA_SIZE; i++)
        Data_Tokens[i] = EMPTY;          /* Init. token table */

    Command_Ptr      = Rcv_Ptr;        /* Set 'C' rec. loop ptr. to assembler ptr. */

    port0           = 0;              /* 80196 CPU port variables */
    port1           = 0;
    port2           = 0;

    pia_a           = 0;              /* 8255A PIA port variables */
    pia_b           = 0;
    pia_c           = 0;

    data             = 0;              /* Save last uplink data */
    command         = 0;              /* Save last uplink command */
    cnt              = 0;              /* Used for building uplink command */
    ntic             = 5;              /* Rate (Hz) of low-speed port */

    new_tics         = 0;              /* Used as timeout variables for */
    old_tics         = 0;              /* uplink commands */

    mode             = ASLP_BIT | FILT_BIT; /* 6 state (6-bit) function byte */
    pwm_off          = 0;              /* PWM offset */
    pwm_lim          = 115;             /* PWM limit */
    pwm              = 0;              /* PWM initial value */
    senstat          = 0;              /* 8 sensor enable/disable byte */
    dark             = FALSE;           /* Dark flag */
    awake            = TRUE;             /* Awake flag */
    uc0              = 0;              /* Used for motor filter */
    uf0              = 0;              /* Used for motor filter */
    krs              = 0;              /* Used for stored position array */
    fold             = 0;              /* Used to prevent estimator saturation */

    Id1              = 0;              /* Index for 1st commutated byte */
    Id2              = 0;              /* Index for 2nd commutated byte */
    Id3              = 0;              /* Index for 3rd commutated byte */
    outsiz           = 0;              /* Used to index through output queue */

    kps              = 0;              /* Used to calc. rate every 200 ms */
    data_index       = 0;              /* Output queue ISR1 index */
    out_index        = 0;              /* Output queue FGL index */
    time_index       = 0;              /* Incrementing record timer */

    loop_index       = 0;              /* ISR1 loop timer */
    time_out         = 0;              /* Used for A/D timeout */

    for (i=0; i<8; i++)
        s[i] = 0;                  /* Raw sun sensor data */
}

```

```

integ          = 0.0;           /* Integral of rate error */
rate           = 0.0;           /* Current rate */
pos            = 0.0;           /* Position for control */
EstRate        = 0.0;           /* Estimated rate */
IRSRate        = 0.0;           /* Inertial rate */
int_lim        = 32000.0;       /* Integral limit */
clip           = 5.0;           /* Rate limit */
pos_c          = 0.0;           /* Commanded position */
rat_cx         = 0.0;           /* Commanded rate */
coarse          = 0.0;           /* Course adjustment to position */
fine           = 0.0;           /* Find adjustment to position */
Kp              = 0.1;           /* Proportional gain */
Ki              = 2.0;           /* Integral gain */
Kd              = 30.0;          /* Derivative gain */
timeout        = 0.05;          /* 50 msec */
V5              = 0.0;           /* 5 volt storage */
V12             = 0.0;           /* 12 volt storage */
Current         = 0.0;           /* Current storage */
flux_min        = 20.0;          /* Flux level needed for awake */
gray_width     = 0.53846 * flux_min; /* A function of flux_min */
heat_min        = 0.0;           /* Thermostat min. temp. */
heat_max        = 15.0;          /* Thermostat max. temp. */
heat_stat       = 0;             /* Voltage of heater relay */
pos0            = 0.0;           /* Current position for fold */

for (i=0; i<8; i++)
    Temper[i] = 0.0;           /* Storage for 8 temp. channels */

for (i=0; i<5; i++) {
    D[i]  = 0.0;
    p[i]  = 0.0;
    uu[i] = 0.0;
}
ssfac[0] = 14.08;           /* Sun sensor calibration variables */
ssfac[1] = 14.86;
ssfac[2] = 11.70;
ssfac[3] = 11.09;
ssfac[4] = 12.30;
ssfac[5] = 15.67;
ssfac[6] = 10.80;
ssfac[7] = 10.31;

set_enable( ON );
set_heater( OFF );

/*-----
 * Below are the CPU specific initializations
 *-----
 */
WSR          = 0x00;           /* Window Select 0 */
IOC1          = 0x21;           /* Enable TxD pin for ser. port use & enab. PWM */
IOC2          = 0xC4;           /* 11000000 Lock Cam & enable & PWM freq. */
HSO_COMMAND   = 0xB8;           /* 10111000 Cam lock, Set HSO pin, Cause */
                                /* interrupt, & soft timer 0 using timer 1 */
HSO_TIME      = 37500;          /* CAM looks for this count from Timer1 */
                                /* before sending soft timer interrupt.~50 msec */
                                /* Ticks = 50 msec / (8 * 2 / 12 Mhz) */
INT_MASK      = 0x20;           /* 0010000b...0x20..enable software interrupt */
INT_MASK1     = 0x02;           /* 00000010b...0x02..enable RI serial interrupt */
SP_CON         = 0x09;           /* Enable receiver set to Mode 1 */
BAUD_RATE     = 0x38;           /* Set to 2400 baud...LSB sets baud */
BAUD_RATE     = 0x81;           /* MSB sets XTAL1 source */
                                /* Divisor = 12 Mhz / (Baud * 16) */
                                /* For 2400 baud = 0x0138 | 0x8000 = 0x8138 */
PIA_CONTROL   = 0x80;           /* Set PIA Ports A output, B output, C output */
enable_interrupt();
}

```

```

/*
 * Initialize external UART (82050)
 */
void init_Uart()
{
    disable_interrupt();

    pia_b     &= ~UART_RESET;          /* Set Uart RESET line low */
    pia_b     |=  UART_RTS;           /* Set Uart RTS line high */
    PIA_PORTB = pia_b;                /* Now, set portb */

    pause(1000);

    pia_b     |=  UART_RESET;          /* These cause Uart to use */
    pia_b     &= ~UART_RTS;           /* crystal oscillator */
    PIA_PORTB = pia_b;

    pause(100);

    pia_b     &= ~UART_RESET;          /* Return to normal state */
    pia_b     |=  UART_RTS;
    PIA_PORTB = pia_b;

    pause(400);

    IER      = 0x00;                  /* Disable all interrupts */
    MCR      = 0x00;                  /* Disable modem control bits */
    LCR      = 0x80;                  /* Set DLAB for baud change */
    BAL      = 0x53;                  /* Set 1200 baud...with 16.0 Mhz crystal */
    BAH      = 0x00;                  /* Divisor = (16.0 Mhz / 10) / (Baud * 16) */
    LCR      = 0x07;                  /* Clear DLAB, 8 data bits, no par., 2 stop bits */

    enable_interrupt();
}

/*
 * This is needed by printf to properly output chars to serial
 * port.
 */
int putchar( int c )
{
    while ( (SP_STAT & 0x08) == 0x00 );

    SBUF = (char)c;

    return c;
}

/*
 * Output 1 byte to 82050 uart port.
 */
void putbyte_82050( UCHAR c )
{
    while( (LSR & 0x20) == 0 );

    TXD = c;
}

/*
 * Output 1 byte to port.
 */
void putbyte( UCHAR c )
{
    while ( (SP_STAT & 0x08) == 0x00 );

    SBUF = c;
}

```

```

/*
 * Read heater status
 */
UINT read_heatstat()
{
    UINT total, high, low;

    time_out = 0;

    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    AD_COMMAND      = 0x08 | HEAT_STAT;
                    /* Set GO bit of 10 bit AD and OR in */
                    /* Channel number */
                    /* xxxx1000 Bits 0-2 channel number */
                    /* Bit 3 GO bit */

    pause(0);          /* Wait a little */
    time_out = 0;       /* Keep loop from freezing */
    /* Wait for conversion to complete */
    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    high  = (UINT)AD_RESULT_HI << 2;
    low   = (low >> 6) & 0x3;
    total = high + low;

    return total;
}

/*
 * Read 1 of possible 8 temperatures via MUX
 */
UINT read_temp( UCHAR channel )
{
    UINT total, high, low;

    port1      &= 0xf8;          /* Flush out bottom 3 bits */
    port1      |= channel;        /* OR in channel number */
    IO_PORT1   = port1;          /* Set PORT1 to approp. channel number */

    pia_a      |= TEMP_MUX_SEL; /* Select temperature MUX */
    PIA_PORTA  = pia_a;          /* Set PIA channel A */

    time_out = 0;

    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    AD_COMMAND      = 0x08 | MUX_TEMP;
                    /* Set GO bit of 10 bit AD and OR in */
                    /* Channel number */
                    /* xxxx1000 Bits 0-2 channel number */
                    /* Bit 3 GO bit */

    pause(0);          /* Wait a little */
    time_out = 0;       /* Keep loop from freezing */
    /* Wait for conversion to complete */
    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    high  = (UINT)AD_RESULT_HI << 2;
    low   = (low >> 6) & 0x3;
    total = high + low;

    pia_a      &= ~TEMP_MUX_SEL; /* De-select temperature MUX */
    PIA_PORTA  = pia_a;          /* Set PIA channel A */

    return total;
}

```

```

/*
 * Read 1 of possible 2 voltages via analog switch U7
 */
-----
```

UINT read\_volt\_mon( UCHAR voltage )

```

{
    UINT total, high, low;

    if ( voltage == VOLTS_5 )
        pia_a      &= ~VOLT_MON_SEL;          /* Choose 5v line */
    else
        pia_a      |= VOLT_MON_SEL;        /* Choose 12v line */

    PIA_PORTA     = pia_a;                /* Set PIA channel A */

    time_out = 0;

    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    AD_COMMAND     = 0x08 | VOLT_MON;
                    /* Set GO bit of 10 bit AD and OR in */
                    /* Channel number */
                    /* xxxx1000 Bits 0-2 channel number */
                    /* Bit 3 GO bit */

    pause(0);          /* Wait a little */
    time_out = 0;      /* Keep loop from freezing */
    /* Wait for conversion to complete */
    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    high  = (UINT)AD_RESULT_HI << 2;
    low   = (low >> 6) & 0x3;
    total = high + low;

    return total;
}
```

```

/*
 * Read current monitor
 */
-----
```

UINT read\_cur\_mon()

```

{
    UINT total, high, low;

    time_out = 0;

    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    AD_COMMAND     = 0x08 | CURRENT_MON;
                    /* Set GO bit of 10 bit AD and OR in */
                    /* Channel number */
                    /* xxxx1000 Bits 0-2 channel number */
                    /* Bit 3 GO bit */

    pause(0);          /* Wait a little */
    time_out = 0;      /* Keep loop from freezing */
    /* Wait for conversion to complete */
    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    high  = (UINT)AD_RESULT_HI << 2;
    low   = (low >> 6) & 0x3;
    total = high + low;

    return total;
}
```

```

/*
 *-----*
 * Read linear temperature from heater
 *-----*
 */

UINT read_lin_temp()
{
    UINT total, high, low;

    time_out = 0;

    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    AD_COMMAND      = 0x08 | LINEAR_TEMP;
                    /* Set GO bit of 10 bit AD and OR in */
                    /* Channel number */
                    /* xxxx1000 Bits 0-2 channel number */
                    /* Bit 3 GO bit */

    pause(0);          /* Wait a little */
    time_out = 0;       /* Keep loop from freezing */
                        /* Wait for conversion to complete */
    while ( (((low = (UINT)AD_RESULT_LO) & 0x08) > 0) &&
           (time_out++ <= MAXTIME) );

    high  = (UINT)AD_RESULT_HI << 2;
    low   = (low >> 6) & 0x3;
    total = high + low;

    return total;
}

/*
 *-----*
 * Read 1 of 8 possible solar sensors
 *-----*
 */

UINT read_solar_sen( UCHAR channel )
{
    UINT      low, high, total;

    port1      &= 0xf8;           /* Flush out bottom 3 bits */
    port1      |= channel;        /* OR in channel number */
    port1      |= SUN_CHIP_SEL;   /* Select sun multiplexor */
    port1      &= ~SUN_RATE_SEL;  /* Select sun sensors */

    port1      |= START_CONV;     /* This section is new */
    IO_PORT1   = port1;
    pause(1);           /* Need small delay */

    port1      &= ~START_CONV;    /* Start AD conversion */
    IO_PORT1   = port1;
    READ_ATOD  = 0x00;           /* Make CS go low */

    port1      |= START_CONV;    /* Stop AD conversion */
    IO_PORT1   = port1;
    /* Set PORT1 */

    time_out = 0;                /* Keep loop from freezing */
                                /* Wait for conversion to complete */
    while( ((IO_PORT2 & 0x80) == 0) &&
          (time_out++ <= MAXTIME) );

    port1      &= ~HIGH_LOW_BYTE;
    IO_PORT1   = port1;           /* Make High byte valid */
    high       = (UINT)READ_ATOD << 8;

    port1      |= HIGH_LOW_BYTE;
    port1      &= ~SUN_CHIP_SEL;   /* De-select sun multiplexor */
    IO_PORT1   = port1;           /* Make Low byte valid */
    low        = (UINT)READ_ATOD; /* 4 bits are in upper nibble */

```

```

        total = (low + high) >> 4;           /* Create 16 bit word */
                                                /* and shift down to 12 bits */

        return total;
    }

/*
 * Read rate sensor
 */
UINT read_rate()
{
    UINT      low, high, total;

    port1      |= SUN_RATE_SEL;      /* Select rate sensor */
    port1      &= ~START_CONV;      /* Start AD conversion */
    IO_PORT1   = port1;            /* Set PORT1 */

    READ_ATOD   = 0x00;             /* Make CS go low */

    port1      |= START_CONV;      /* Stop AD conversion */
    IO_PORT1   = port1;            /* Set PORT1 */

    time_out = 0;                  /* Keep loop from freezing */
    /* Wait for conversion to complete */

    while( ((IO_PORT2 & 0x80) == 0) &&
          (time_out++ <= MAXTIME) );

    port1      &= ~HIGH_LOW_BYTE;
    IO_PORT1   = port1;            /* Make High byte valid */
    high       = (UINT)READ_ATOD << 8;

    port1      |= HIGH_LOW_BYTE;
    IO_PORT1   = port1;            /* Make Low byte valid */
    low        = (UINT)READ_ATOD; /* 4 bits are in upper nibble */

    total = (low + high) >> 4;      /* Create 16 bit word */
                                    /* and shift down to 12 bits */

    return total;
}

/*
 * Set heater control
 */
void set_heater( UCHAR state )
{
    if ( state == ON )
        pia_b      |= HEATER_CTRL; /* Turn heater on */
    else
        pia_b      &= ~HEATER_CTRL; /* Turn heater off */

    PIA_PORTB   = pia_b;           /* Set PIA channel B */
}

/*
 * Set motor brake
 */
void set_brake( UCHAR state )
{
    if ( state == ON )
        pia_a      |= MOTOR_BREAK; /* Turn motor brake on */
    else
        pia_a      &= ~MOTOR_BREAK; /* Turn motor brake off */

    PIA_PORTA   = pia_a;           /* Set PIA channel A */
}

```

```

/*
 *-----*
 * Set motor direction
 *-----*
 */

void set_dir( UCHAR state )
{
    if ( state == CW )
        port1      |= MOTOR_DIRECTION; /* Change motor direction to CW */
    else
        port1      &= ~MOTOR_DIRECTION; /* Change motor direction to CCW */

    IO_PORT1      = port1;           /* Set PIA channel A */
}

/*
 *-----*
 * Set motor enable
 *-----*
 */

void set_enable( UCHAR state )
{
    if ( state == ON )
        pia_a      |= MOTOR_ENABLE;   /* Turn motor brake on */
    else
        pia_a      &= ~MOTOR_ENABLE; /* Turn motor brake off */

    PIA_PORTA     = pia_a;          /* Set PIA channel A */
}

/*
 *-----*
 * Set motor current monitor shutdown
 *-----*
 */

void set_shutdown( UCHAR state )
{
    if ( state == ON )
        pia_a      |= MOTOR_SHUTDOWN; /* Turn motor shutdown on */
    else
        pia_a      &= ~MOTOR_SHUTDOWN; /* Turn motor shutdown off */

    PIA_PORTA     = pia_a;          /* Set PIA channel A */
}

/*
 *-----*
 * Set motor PWM
 *-----*
 */

void set_pwm( UCHAR value )
{
    pwm       = value;             /* Set motor PWM level */
    PWM_CONTROL = pwm;
}

/*
 *-----*
 * Pause program
 *-----*
 */

void pause( int time )
{
    static UINT i;

    for (i=0; i<time; i++)          /* Pause for 'time' */
}

```

```

/*
 *-----*
 * This is called by assembler interrupt routine Software_Timer_ISR.
 * Currently, this function is called every 50 msec.
 *-----*
 */

void control()
{
    UINT      i, usData;

    new_tics++;                                /* Used for timeout calc. */
                                                /* of incoming commands */

    pia_a     &= ~WD_STROBE;                  /* Strobe the Watch Dog Timer */
    PIA_PORTA = pia_a;

    pia_a     |= WD_STROBE;
    PIA_PORTA = pia_a;

    /*
     *-----*
     * This section reads the following sensors:
     * 1. Rate           - 1 channel using 12bit AD
     * 2. Solar Sensors - 8 channels using 12bit AD
     *-----*
     */

    usData  = read_rate();                      /* Read rate sensor & store */
    IRSRate = (float)usData * C12BIT_5V;        /* Convert counts to 5 volts */
    IRSRate = (IRSRate - ZERORATE) / 0.15;      /* Convert voltage to degrees */

    for (i=0; i<8; i++)                        /* Read 8 solar sensor channels */
        s[i] = read_solar_sen( i );             /* Don't rescale */

    /*
     *-----*
     * This section by Lanzi
     *-----*
     */

    PosEst();                                    /* Takes s[] & computes pos */

    if ( mode & ASLP_BIT ) {                   /* Auto on/off enabled? */
        if ( (dark == TRUE) && (awake == TRUE) ) {
            awake = FALSE;
            mode |= HEAT_BIT;                  /* Enable heater circuit */
            set_pwm( 0 );                    /* Stop driving motor */
            krs = 0;
            for(i=0;i<5;i++) {
                D[i] = 0.0;
                uu[i] = 0.0;
            }
        }                                     /* Wake Up !! */
        else if ( (dark == FALSE) && (awake == FALSE) ) {
            awake = TRUE;
            mode &= ~HEAT_BIT;              /* Disable heater circuit */
            set_heater( OFF );            /* Turn heater off immediately */
        }
    }
    else
        awake = TRUE;

    if ( awake == TRUE )
        RatEst();                            /* Takes pos & computes ESTRate */
    else
        EstRate = 0;

    if (mode & RATE_BIT)                      /* Use derived rate */
        rate = EstRate;
    else                                      /* Use digitized inertial rate */
        rate = IRSRate;

    if( (mode & CRAT_BIT) && (Kp > 0.0) ) /* Make sure no division by zero */
        pos_c = pos + rat_cx/Kp;
}

```

```

/* Put IVLlaw into code */

if ( (mode & FDBK_BIT) && (awake == TRUE) )
    IVLlaw();

/*-----*/
if ( !(loop_index++ % 20) ) {           /* Output to queue at 1Hz rate */

/*-----
 * This section reads the following sensors:
 * 1. Temperature - 8 channels using 10bit AD
 * 2. 5 Volt Monitor - 1 channel using 10bit AD
 * 3. 12 Volt Monitor - 1 channel using 10bit AD
 * 4. Motor Current - 1 channel using 10bit AD
 *-----
 */

for (i=0; i<8; i++) {                  /* Read 8 temp. channels */
    usData = read_temp( i );
    Temper[i] = (float)usData * C10BIT_5V; /* Rescale to 5 volt range */
}

Temper[0] = conv_to_temp( Temper[0] );   /* Convert to temperature (deg. C) */
                                         /* This sensor needed for thermostat */

usData = read_volt_mon( VOLTS_5 );      /* Read 5 volt line */
V5      = (float)usData * C10BIT_5V;    /* Rescale to 5 volt range */

usData = read_volt_mon( VOLTS_12 );     /* Read 12 volt line */
V12     = (float)usData * C10BIT_5V;    /* Rescale to 5 volt range */

usData = read_cur_mon();                /* Read current monitor */
Current = (float)usData * C10BIT_5V;    /* Rescale to 5 volt range */

usData = read_heatstat();               /* Read heater status */
heat_stat = (float)usData * C10BIT_5V; /* Rescale to 5 volt range */

                                         /* Control the heater */
if ( (mode & HEAT_BIT) && (heat_min < heat_max) ) {
    if ( Temper[0] <= heat_min )
        set_heater( ON );           /* Turn heater on */
    else if ( Temper[0] >= heat_max )
        set_heater( OFF );         /* Turn heater off */
}
else
    set_heater( OFF );             /* Keep heater off */

DataList();                            /* Output data to queue */

/*-----*/
Data_Tokens[data_index] = FULL;        /* Now foreground can write to port */

if ( ++data_index == DATA_SIZE )       /* Look for wrap around */
    data_index = 0;
}

/*-----
 * The following function converts voltage to temperature (deg. C). and
 * returns that temperature. Use a 5th order polynomial for calculation.
 *-----
 */

float conv_to_temp( float voltage )
{
    static float v2,v3,v4,v5;

    v2 = voltage * voltage;            /* Crunch multiplies for efficiency */
    v3 = voltage * v2;
    v4 = voltage * v3;
    v5 = voltage * v4;
                                         /* Return polynomial result */
    return ( A0 + A1*voltage + A2*v2 + A3*v3 + A4*v4 + A5*v5 );
}

```

```

/*
 * The following functions were written by Jim Lanzi and edited (somewhat)
 * by Steve Bailey.
 */
-----



/* IVLLAW-----
*
* ... Logic for Inner Velocity Loop Control Law
*
* Commanded Rate is Proportional to the Position Error.
* Control signal, PWM, is developed by an Inner Velocity Loop
* and is given by a linear combination of:
*     . Integral of Rate Error
*     . Rate Error
* Rate Error is given as the difference between the inertial
* gondola rate and the commanded rate from the outer position loop.
*
*-----
*-----*/
void IVLlaw(void)
{
    int u, uf;
    float pos_err, rat_err, rat_c;
/*
* ... Position Error
*/
    pos_err = pos_c - pos;

    if (pos_err > 180.0)
        pos_err -= 360.0;
    if (pos_err < -180.0)
        pos_err += 360.0;

/*
* ... Commanded Gondola Rate
*/
    rat_c = Kp*pos_err;
    if(rat_c > clip) rat_c = clip;
    if(rat_c <-clip) rat_c = -clip;
/*
* ... Rate Error
*/
    rat_err = rat_c - rate;
/*
* ... Integrate Rate Error
*/
    integ = integ + rat_err*((float)timeout);
    if(integ > int_lim) integ = int_lim;
    if(integ <-int_lim) integ = -int_lim;
/*
* ... Control Signal
*/
    u = (int) (Ki*integ + Kd*rat_err);
/*
* ... First Order Filter
*/
    if (mode & FILT_BIT) {
        /* Filter input to motor driver (1st order 1hz low-pass) */
        uf = (int)((88931*(u+uc0) + 477501*uf0) / 655361 );
        uf0 = uf;
        uc0 = u;
        u = uf;
    }
/*
* ... Stiction Compensation
*/
    if(u > 0 )
        u = u + pwm_off;
    else if(u < 0)
        u = u - pwm_off;
/*
* ... Clip Input to Motor Driver
*/
    if(u > pwm_lim) u =  pwm_lim;
}

```

```

        if(u <-pwm_lim) u = -pwm_lim;
/*
 * ... Set PWM
 */
    set_pwm( (unsigned char)ABS(u) );
/*
 * ... Set Direction Bit
 */
    if(u > 0)
        set_dir( CW );
    else if (u < 0)
        set_dir( CCW );
}

/*RATEST-----*/
/*
* ... Rate Estimation Algorithm
*      Rate filter derived using a full state observer w/ bias estimation.
*
*-----*/
void RatEst(void)
{
    int i;
/*
* ... Compute new rate every 4*50 ms
*/
    if (!(kps++ % 4)) {
/*
* ... 'krs' is used to load up the stored position array, p[],
* on start-up.
*/
        if(krs < 6) ++krs;
/*
* ... Update stored position values
*/
        for(i=4;i>0;i--) {
            p[i] = p[i-1];
        }
/*
* ... Increment the 'fold' variable when 'pos' passes through
* 180 deg/-180 deg. Prevents estimator from being saturated
* with large position changes.
*/
        if( ABS(pos - pos0) > 200.0 ) {
            if( pos > pos0)
                fold = fold - 1;
            else
                fold = fold + 1;
        }
/*
* ... Store current position for next fold calculation - pos0
* Compute folded post ion for use in rate filter      - p[0]
*/
        pos0 = pos;
        p[0] = pos + (float)(fold*360);
/*
* ... When p[] is loaded up, the Rate Estimator is enabled.
*/
        if(krs < 5) return;
/*
* ... Estimated Rate is calculated from:
*
*      1. Previous estimated rate values
*      2. Current and previous position values
*      3. Previous PWM levels
*
* The filter coefficients were computed from an optimal digital
* observer design.
*/
        EstRate =
-( -3.71456152e+0*D[0] + 5.29437242e+0*D[1] -3.57735109e+0*D[2] +
  1.13022883e+0*D[3] -1.32684396e-1*D[4] ) +
  4.52672481e-3*p[0] -1.24367324e-2*p[1] + 1.19351226e-2*p[2]
-4.64574884e-3*p[3] + 6.20633831e-4*p[4] +
  1.14505703e-3*uu[0] -3.11087881e-3*uu[1] + 2.95761808e-3*uu[2]
-1.14465790e-3*uu[3] + 1.52861593e-4*uu[4];

```

```

/*
 * ... Store current value of estimated rate and PWM for use
 *      in next rate filter sample.
 */
    for(i=4;i>0;i--) {
        D[i] = D[i-1];
        uu[i] = uu[i-1];
    }
    D[0] = EstRate;
    uu[0] = (float)pwm;
/*
 * ... Ensure proper sense is given to stored PWM values!!
 */
    if(!(port1 & MOTOR_DIRECTION))
        uu[0] = -uu[0];
}

/*POSEST-----*/
/*
 * ... Computes position from -180 to +180 deg from 4 photodiodes equally
 *      spaced around a ring. Sensors 4 thru 7 are backups to sensors 0 - 3.
 *
 *      Sensors 0 thru 3 are spaced clockwise (looking down) around the ring
 *      with sensor 0 located 90 degrees ccw from zero. Sensors 0 and 1 are
 *      the primary pointing sensors.
 */
void PosEst(void)
{
    int is,i;
    int offset;
    float yi,yj,sg,delta;
    float y[4];
/*
 * ... Set up Working Sensor Reading Array
 */
    for(i=0;i<4;i++) y[i] = (float)s[i] * ssfac[i];
/*
 * ... Use Backup Sensor/Signal if senstat flag is set
 */
    if(senstat & 0x01) y[0] = (float)s[4] * ssfac[4];
    if(senstat & 0x02) y[1] = (float)s[5] * ssfac[5];
    if(senstat & 0x04) y[2] = (float)s[6] * ssfac[6];
    if(senstat & 0x08) y[3] = (float)s[7] * ssfac[7];
/*
 * ... Locate Max Sensor
 */
    is = 0;
    yi = -9999.;
    for(i=0; i<4; i++) {
        if(y[i] > yi) {
            yi = y[i];
            is = i;
        }
    }
/*
 * ... Check All Sensors Dark Condition
 */
    if ( yi < flux_min * 24.576 ) {
        dark = TRUE;
        pos = 30.0;
        return;
    }
    else if ( yi > (flux_min + gray_width) * 24.576 )
        dark = FALSE;
/*
 * ... Locate Lesser Sensor and Compute Central Angle Offset
 */
    if( y[(is+3)%4] > y[(is+5)%4] ) {
        /* max is (+) angle from lesser */
        offset = 90*is-90;
        sg = +1.0;
        yj = y[(is+3)%4];
    }
}

```

```

    else {
        /* max is (-) angle from lesser */
        offset = 90*is;
        sg = -1.0;
        yj = y[(is+5)%4];
    }
/*
 * ... Compute delta angle based on linear approximation
 */
    delta = sg*45.0*(yi - yj)/(yi+yj);
/*
 * ... Nonlinear Correction
 */
    delta = delta + delta*(2.675674e-1
                           + delta*delta*(-1.823400e-4 + delta*delta*2.487085e-8));
/*
 * ... Compute Angle
*/
    pos = (float) offset + delta;
    while(pos > 180.) pos -= 360.;
    while(pos < -180.) pos += 360.;

}

/*DataList-----*/
/*
 * ... Encode Primary Housekeeping List to Output Queue
*/
void DataList(void)
{
    int i;
    unsigned short usData;
    unsigned char cksmData;

    cksmData = 0;           /* Initialize checksum */
    outsiz = 0;            /* Initialize element variable */

/*
 * ... Sync and ID bytes (Balloon ID = 0)
*/
    encode(0xfa, &cksmData);
    encode(0xf3, &cksmData);
    encode(0x20, &cksmData);

    cksmData = 0;

    encode(0x26, &cksmData);
    encode(0, &cksmData);
    encode(40, &cksmData);

/*
 * ... Data Bytes
*/
    encode( (unsigned char)(time_index >> 8), &cksmData);
    encode( (unsigned char) time_index++, &cksmData);
    encode( (unsigned char) pwm, &cksmData);
    encode( (unsigned char)((port1 & 0x80) | (pia_a & 0x20)), &cksmData);
    encode( (unsigned char) mode, &cksmData);
    encode( (unsigned char) senstat, &cksmData);
    for(i=0;i<8;i++) {
        encode( (unsigned char)(s[i] >> 8), &cksmData);
        encode( (unsigned char)s[i], &cksmData);
    }

    usData = (unsigned short)(182.03*pos+32768);
    encode( (unsigned char)(usData >> 8), &cksmData);
    encode( (unsigned char)usData, &cksmData);

    usData = (unsigned short)(128.0*integ+32768);
    encode( (unsigned char)(usData >> 8), &cksmData);
    encode( (unsigned char)usData, &cksmData);

    usData = (unsigned short)(1310.68*IRSRate+32768);
    encode( (unsigned char)(usData >> 8), &cksmData);
    encode( (unsigned char)usData, &cksmData);
}

```

```

usData = (unsigned short)(1310.68*EstRate+32768);
encode( (unsigned char)(usData >> 8),      &cksmData);
encode( (unsigned char)usData,                &cksmData);

encode( (unsigned char) Id1,                  &cksmData);

switch(Id1) {
    case 0:
        encode( (unsigned char) ((Temper[0] + 56.666) * 2.086), &cksmData);
        break;
    case 1:
        Temper[1] = conv_to_temp( Temper[1] ); /* Convert to temperature (deg. C) */
        encode( (unsigned char) ((Temper[1] + 56.666) * 2.086), &cksmData);
        break;
    case 2:
        Temper[2] = conv_to_temp( Temper[2] ); /* Convert to temperature (deg. C) */
        encode( (unsigned char) ((Temper[2] + 56.666) * 2.086), &cksmData);
        break;
    case 3:
        encode( (unsigned char) (51.0*v5),           &cksmData);
        break;
    case 4:
        encode( (unsigned char) (51.0*v12),          &cksmData);
        break;
    case 5:
        encode( (unsigned char) (51.0*Current),     &cksmData);
        break;
    case 6:
        encode( (unsigned char) ((heat_min + 56.666) * 2.086), &cksmData);
        break;
    case 7:
        encode( (unsigned char) ((heat_max + 56.666) * 2.086), &cksmData);
        break;
    case 8:
        encode( (unsigned char) (51.0*heat_stat),   &cksmData);
        break;
    default:
        break;
}

switch((Id1 + 5) % 9) {
    case 0:
        encode( (unsigned char) ((Temper[0] + 56.666) * 2.086), &cksmData);
        break;
    case 1:
        Temper[1] = conv_to_temp( Temper[1] ); /* Convert to temperature (deg. C) */
        encode( (unsigned char) ((Temper[1] + 56.666) * 2.086), &cksmData);
        break;
    case 2:
        Temper[2] = conv_to_temp( Temper[2] ); /* Convert to temperature (deg. C) */
        encode( (unsigned char) ((Temper[2] + 56.666) * 2.086), &cksmData);
        break;
    case 3:
        encode( (unsigned char) (51.0*v5),           &cksmData);
        break;
    case 4:
        encode( (unsigned char) (51.0*v12),          &cksmData);
        break;
    case 5:
        encode( (unsigned char) (51.0*Current),     &cksmData);
        break;
    case 6:
        encode( (unsigned char) ((heat_min + 56.666) * 2.086), &cksmData);
        break;
    case 7:
        encode( (unsigned char) ((heat_max + 56.666) * 2.086), &cksmData);
        break;
    case 8:
        encode( (unsigned char) (51.0*heat_stat),   &cksmData);
        break;
    default:
        break;
}

if (!(time_index % 9)) /* First commutated byte index */
    Id1 = (++Id1)%9;

```

```

encode( (unsigned char) Id2,           &cksmData);

switch(Id2) {
    case 0 :
        encode( (unsigned char) (51.0*Kp),      &cksmData);
        break;
    case 1 :
        encode( (unsigned char) (25.5*Ki),      &cksmData);
        break;
    case 2 :
        encode( (unsigned char) (4.25*Kd),      &cksmData);
        break;
    case 3 :
        encode( (unsigned char) (10.2*clip),     &cksmData);
        break;
    case 4 :
        encode( (unsigned char) pwm_lim,         &cksmData);
        break;
    case 5 :
        encode( (unsigned char) pwm_off,         &cksmData);
        break;
    case 6 :
        encode((unsigned char) (0.5 * coarse + 90.0), &cksmData);
        break;
    case 7 :
        encode((unsigned char) (12.7 * rat_cx + 128.0), &cksmData);
        break;
    case 8 :
        encode( (dark << 1 ) | awake, &cksmData);
        break;
    case 9 :
        encode((unsigned char)(0.255 * flux_min), &cksmData);
        break;
    case 10 :
        encode((unsigned char) (100.0 * fine), &cksmData);
        break;

    default:
        break;
}

if (!(time_index % 9))                  /* Second commutated byte index */
    Id2 = (++Id2)%11;

encode( command+1, &cksmData);          /* Echo back current command */
encode( data, &cksmData);              /* Echo back current data byte */

encode( (unsigned char) Id3,           &cksmData);

switch(Id3) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:                                /* Convert to temperature (deg. C) */
        Temper[Id3+3] = conv_to_temp( Temper[Id3+3] );
        encode( (unsigned char) ((Temper[Id3+3] + 56.666) * 2.086), &cksmData);
        break;
    default:
        break;
}

if (!(time_index % 9))                  /* Third commutated byte index */
    Id3 = (++Id3)%5;

encode(0, &cksmData);                 /* Spare data byte */

Data_Buffer[data_index][outsiz] = cksmData;
}

```

```
/*ENCODE-----*
*
* ... Lay down one byte of data onto the output queue.
*
*-----*/
void encode( UCHAR ucData, UCHAR *cksmData )
{
    Data_Buffer[data_index][outsiz++] = ucData;
    *cksmData = *cksmData + ucData;
}
```

## ***stdio.h***

```
/* - STDIO.H -
Subset of ANSI standard I/O function declarations.

Version: 3.00 [IANR]

*/
#ifndef NULL
#define NULL    (void *) 0
#endif

#ifndef EOF
#define EOF     (-1)
#endif

int    putchar(int);
int    getchar(void);
int    sprintf(char *,const char *,...);
int    printf(const char *,...);
int    scanf(const char *,...);
int    sscanf(const char *, const char *,...);
char   *gets(char *);
```

## **io8096.h**

```
/*
 - IO8096.H -
This file #defines the Special Function Registers
Version: 3.00 [A.R.]
*/
/* A/D converter */

#define AD_RESULT_LO    (* (unsigned char *) ( 0x02 ))
#define AD_RESULT_HI    (* (unsigned char *) ( 0x03 ))
#define AD_COMMAND       (* (unsigned char *) ( 0x02 ))

/* High Speed Input */

#define HSI_MODE         (* (unsigned char *) ( 0x03 ))
#define HSI_TIME          (* (unsigned int *) ( 0x04 ))
#define HSI_STATUS        (* (unsigned char *) ( 0x06 ))

/* High Speed Output */

#define HSO_TIME          (* (unsigned int *) ( 0x04 ))
#define HSO_COMMAND        (* (unsigned char *) ( 0x06 ))

/* Serial port receive/transmit Buffer */

#define SBUF              (* (unsigned char *) ( 0x07 ))
#define SBUF_RX            (* (unsigned char *) ( 0x07 ))
#define SBUF_TX            (* (unsigned char *) ( 0x07 ))

/* Interrupt Mask */

#define INT_MASK           (* (unsigned char *) ( 0x08 ))
#define INT_MASK1          (* (unsigned char *) ( 0x13 ))

/* Interrupt Pending */

#define INT_PENDING         (* (unsigned char *) ( 0x09 ))
#define INT_PENDING1        (* (unsigned char *) ( 0x12 ))

/* Watchdog Timer */

#define WATCHDOG          (* (unsigned char *) ( 0x0A ))

/* Timer 1 and Timer 2 */

#define TIMER1             (* (unsigned int *) ( 0x0A ))
#define TIMER2             (* (unsigned int *) ( 0x0C ))

/* Baud Rate */

#define BAUD_RATE          (* (unsigned char *) ( 0x0E ))

/* I/O ports */

#define IO_PORT0           (* (unsigned char *) ( 0x0E ))
#define IO_PORT1           (* (unsigned char *) ( 0x0F ))
#define IO_PORT2           (* (unsigned char *) ( 0x10 ))

/* Serial Port Status */
```

```

#define SP_STAT      (* (unsigned char *) ( 0x11 ))

/* Serial Port Control */

#define SP_CON       (* (unsigned char *) ( 0x11 ))

/* I/O status, HSO */

#define IOS0         (* (unsigned char *) ( 0x15 ))

/* I/O status, HSI */

#define IOS1         (* (unsigned char *) ( 0x16 ))

/* I/O Control Register 0 */

#define IOC0         (* (unsigned char *) ( 0x15 ))

/* I/O Control Register 1 */

#define IOC1         (* (unsigned char *) ( 0x16 ))

/* I/O Control Register 1 */

#define IOC2         (* (unsigned char *) ( 0x0b ))

/* Pulse Width Modulation Control */

#define PWM_CONTROL   (* (unsigned char *) ( 0x17 ))

/* Window Select Register */

#define WSR          (* (unsigned char *) ( 0x14 ))

#define SERIAL_FLAGS (* (unsigned char *) ( 0x34 ))

```

## **string.h**

```
/*
 - STRING.H -
The ANSI 'string' function declarations.

Version: 3.00 [IANR]

*/
#ifndef size_t
#define size_t int
#endif

char *strcat(char *s1, const char *s2);
char *strncat(char *s1, const char *s2, size_t n);
char *strcpy(char *s1, const char *s2);
char *strncpy(char *s1, const char *s2, size_t n);
size_t strlen(const char *s);

int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
```

## ***ctype.h***

```
/*
   - CTYPE.H -
The ANSI character testing function declarations.

Note: Only 7-bit ASCII is supported by these functions.

Version: 3.00 [IANR]

*/
#define _U      01
#define _L      02
#define _N      04
#define _S      010
#define _P      020
#define _C      040
#define _X      0100
#define _B      0200

extern const unsigned char _ctype_[129];

int toupper(int c); /* "toupper" is also available as a function */
int tolower(int c); /* "tolower" is also available as a function */

#define isalpha(c)    (((_ctype_+1)[c]&(_U|_L)))
#define isupper(c)   (((_ctype_+1)[c]&_U))
#define islower(c)   (((_ctype_+1)[c]&_L))
#define isdigit(c)    (((_ctype_+1)[c]&_N))
#define isxdigit(c)  (((_ctype_+1)[c]&(_N|_X)))
#define isspace(c)   (((_ctype_+1)[c]&_S))
#define ispunct(c)   (((_ctype_+1)[c]&_P))
#define isalnum(c)   (((_ctype_+1)[c]&(_U|_L|_N)))
#define isprint(c)   (((_ctype_+1)[c]&(_P|_U|_L|_N|_B)))
#define isgraph(c)   (((_ctype_+1)[c]&(_P|_U|_L|_N)))
#define iscntrl(c)   (((_ctype_+1)[c]&_C))
#define toupper(c)   (islower((c))? (c)&0x5F: (c))
#define tolower(c)   (isupper((c))? (c)|0x20: (c))
```

## ***math.h***

```
/*
   - MATH.H -
The ANSI-defined (+ a few additional) mathematical functions.

Version: 3.00 [IANR]

      Added abs function          4-17-92    JD
 */

double atan (double arg);
double atan2 (double arg1, double arg2);
double cos (double arg);
double exp (double arg);
double log (double arg);
double log10 (double arg);
double modf (double value, double *iptr);
double pow (double arg1, double arg2);
double sin (double arg);
double sqrt (double arg);
double tan (double arg);
double floor (double arg);
double ceil (double arg);
double frexp (double arg1, int *arg2);
double acos (double arg);
double asin (double arg);
double expl0 (double arg);
int     abs (int value);
double cos (double arg);
double exp (double arg);
double log (double arg);
double log10 (double arg);
double modf (double value, double *iptr);
double pow (double arg1, double arg2);
double sin (double arg);
double sqrt (double arg);
double tan (double arg);
double floor (double arg);
```

## ***control.h***

```
/*
 * ... Controller Mode Variable:
 *      Bit Assignment    Low        High
 *      --- ----- --- -----
 *      0 Feedback mode   Manual     IVL Feedback
 *      1 Rate Mode       IRS        Estimate Rate
 *      2 Night detect    off        on
 *      3 Command Rate    off        on
 *
 * ... Mode Masks
 */
#define FDBK_BIT 0x01
#define RATE_BIT 0x02
#define ASLP_BIT 0x04
#define CRAT_BIT 0x08
#define FILT_BIT 0x10
#define HEAT_BIT 0x20
/*
 * ... Controller Constants
 */
#define TRUE      1
#define FALSE     0

void  IVLlaw(void);
void  RatEst(void);
void  PosEst(void);
void  DataList(void);
void  encode(unsigned char usData, unsigned char *cksmData);
```

### ***rotator.asm***

```
;-----  
;  
; Rotator.asm  
;  
;  
;  
;  
; NASA  
; Wallops Flight Facility  
; Steven A. Bailey  
; January 28, 1997  
; Version 1.1  
;  
;
```

```
Page  
;  
;  
; Assembler Directives  
;  
;  
LSTOUT +  
LSTCOD +  
LSTEXP +  
LSTMAC +  
LSTWID -  
LSTPAG +  
LSTXRF  
PAGSIZ 56  
PTITL '  
Rotator Controller'
```

```
Page  
;  
;  
; Register Definitions  
;  
;  
$REG8096.INC ; Include file for 8096 registers  
  
RSEG    REGISTERS (1)      ; Register memory allocations  
DSB     2fh                ; Make room for this many registers
```

```
Page  
;  
;  
; Equates & Externals  
;  
;
```

```
extrn      main, control, ?SEG_INIT_L00  
public     Rcv_Ptr, Rcv_Buffer
```

```
Page  
;  
;  
; Internal Ram Segment  
;
```

```

;
;-----[REMOVED]-----[REMOVED]
;

        RSEG    CSTACK(1)
        DSB     2048           ; 2KB 'C' stack
Stack_Ptr:                                ; Stack grows 'down'

        RSEG    SER_BUF(1)
Rcv_Buffer:   DSB     80            ; Serial port receive buffer
Rcv_End:      ; This address marks end of buffer

        Page

;-----[REMOVED]-----[REMOVED]
;
;          Vectors & Chip Configuration registers
;
;-----[REMOVED]-----[REMOVED]

        RSEG    ROM (2)

Rom_Start:                                ; 8096 & 80c196 interrupt vectors

T_Overflow:      DCW    Error_ISR       ; Timer Overflow vector
AD_Complete:    DCW    Error_ISR       ; A/D Conversion Complete vector
HSI_Available: DCW    Error_ISR       ; HSI Data Available vector
HSO:             DCW    Error_ISR       ; High speed Output vector
HSI1:            DCW    Error_ISR       ; HSI.1 external interrupt vector
Sftwr_timer:   DCW    Software_Timer_ISR ; Software timer vector
Serial:          DCW    Error_ISR       ; Serial Port Vector
ExInt:           DCW    Error_ISR       ; External interrupt vector
Trap_Vector:    DCW    Error_ISR       ; Trap vector
OpCode:          DCW    Error_ISR       ; Unimplemented Opcode vector

        ORG    Rom_Start + 30h         ; 80C196 interrupt vectors

Ti:               DCW    Error_ISR       ; Serial Port Xmt Vector
Ri:               DCW    Rx_ISR          ; Serial Port Rcv Vector
HSI4:             DCW    Error_ISR       ; 4th HSI FIFO entry vecotr
T2cap:            DCW    Error_ISR       ; Timer #2 Capture vector
T2overflow:      DCW    Error_ISR       ; Timer #2 Overflow vector
ExInt_Pin:       DCW    Error_ISR       ; External interrupt pin vector
HSI_Full:        DCW    Error_ISR       ; HST FIFO Full Vector
NMI:              DCW    Error_ISR       ; NMI vectors through address 0000h
                                ; for 8096 compatability.

Month:            DCW    Date 5          ; Software Version Date
Day:              DCW    Date 4          ; Version date calculated by
Year:             DCW    Date 6          ; by assembler

        ORG    Rom_Start + 18h         ; Chip Configuration Register

CCR:              DCB    11011001b      ; Enable Power Down
                                ; Bus Width = 8 bit
                                ; /WrL & /WrH write strobe
                                ; ALE
                                ; Ready = 2 wait states
                                ; No Program Lock

        ORG    Rom_Start + 1Ah         ; Required Word for 80c196kc
        DCW    0FFFh

        Page

;-----[REMOVED]-----[REMOVED]
;
;          Code Segment & Reset Entry Point
;
;-----[REMOVED]-----[REMOVED]
;
; This entry point is called first before anything else. From
; here, 'C' main() is called.
;
;
```

```

;-----  

        ORG      Rom_Start + 80h          ; Code reset entry point  

        ld       SP, #Stack_Ptr           ; Set stack pointer to top of RAM...moves down  

        lcall    ?SEG_INIT_L00           ; Initialize 'C' variables  

        ld       Rcv_Ptr, #Rcv_Buffer   ; Set serial input buffer pointer to bottom  

Main:   lcall    main                ; Call 'C' main program  

        sjmp    Main

```

Page

```

;-----  

;                               Serial Receive ISR 2  

;  

;-----  

; inputs      | outputs     | routines called   | regs. altered -  

;-----  

; SBUF        | Rcv_Buff    |                   | Ax  

;-----  

;  

; This routine is interrupt driven and simply stuffs the current  

; char received from the serial port into a circular queue called  

; 'Rcv_Buff'.  

;  

;-----  

Rx_ISR:                                ; Handle Receive interrupt  

        pusha  

        push    Ax                  ; Save general register Ax  

        ei                  ; Re-enable interrupts  

        ldb    Ah, SBUF            ; Read character from serial port  

        stb    Ah, [Rcv_Ptr]+       ; Store character into buffer & incre. Ptr.  

        cmp    Rcv_Ptr, #Rcv_End   ; Check for buffer overflow  

        jlt    Cont                ; Not there yet  

        ld     Rcv_Ptr, #Rcv_Buffer ; Reset to top of buffer  

;  

Cont:  

        pop    Ax                  ; exit ISR.  

        popa  

        ret

```

Page

```

;-----  

;                               Error ISR  

;  

;-----  

; inputs      | outputs     | routines called   | regs. altered -  

;-----  

; none        | none        |                   | none  

;-----  

;  

; This is an un-implemented interrupt error handling routine  

;  

;-----  

Error_ISR:      ret

```

Page

```

;-----  

;                               Software Timer ISR  

;  

;-----  

; inputs      | outputs     | routines called   | regs. altered -
;
```

```

;      TIMER1    |    none     | control()      |   Ax      -
;          |          |                  |   Bx      -
;          |          |                  |   R0      -
;          |          |                  |   R2      -
;          |          |                  |   R4      -
;          |          |                  |   R6      -
;          |          |                  |   R8      -
;-----|-----|-----|-----|-----|-----|-----|
;
; This routine is interrupt driven by on board TIMER1. Currently,
; this routine is called every 50 msec. TIMER1 must be reset to 0
; everytime this routine is called because TIMER1 counts up from 0.
; If it is not reset, wrap-around occurs and an interrupt is issued
; every ~2.5 secs.
;
;-----|-----|-----|-----|-----|-----|-----|
Software_Timer_ISR:

    pusha           ; Save 2 system words
    push  Ax
    push  Bx
    push  R0           ; Save 'C' registers
    push  R2
    push  R4
    push  R6
    push  R8

    ldb   WSR, #15        ; Move to Window 15 first
    ld    TIMER1, #00h       ; Reset timer1 since counting up
    ldb   WSR, #00          ; Return to Window 0

    ldb   INT_MASK1, #00000010b ; Mask in RI serial interrupts
    ei                            ; Enable masked interrupts

    ld    R0, #control       ; Call 'C' function
    lcall control

STI_10: pop  R8           ; Restore 'C' registers
    pop  R6
    pop  R4
    pop  R2
    pop  R0
    pop  Bx
    pop  Ax
    popa                         ; Restore 2 system words
    ret

END

```

### ***reg8096.inc***

```
;      - REG8096.INC -
;
;      Include file containing 8096 register definitions
;

;
;      ICC Working registers
;
IR_BASE      DEFINE 01AH           ; Register base address

R8           DEFINE  (IR_BASE)
R9           DEFINE  (IR_BASE+1)

R0           DEFINE  (IR_BASE+2)
R1           DEFINE  (IR_BASE+3)
R2           DEFINE  (IR_BASE+4)
R3           DEFINE  (IR_BASE+5)
R4           DEFINE  (IR_BASE+6)
R5           DEFINE  (IR_BASE+7)
R6           DEFINE  (IR_BASE+8)
R7           DEFINE  (IR_BASE+9)

Ax           DEFINE  24h           ; Registers created for assembly functions
Ah           DEFINE  25h
Al           DEFINE  24h
Bx           DEFINE  26h
Cx           DEFINE  28h
Dx           DEFINE  2ah
Ex           DEFINE  2ch

Rcv_Ptr      DEFINE  2eh           ; Receive buffer pointer

;
;      Special Function Registers
;

SP           DEFINE  18H
STACKPOINTER  DEFINE  18H

AD_RESULT_LO   DEFINE  02H
AD_RESULT_HI   DEFINE  03H
AD_COMMAND     DEFINE  02H

HSI_MODE       DEFINE  03H
HSI_TIME       DEFINE  04H
HSI_STATUS     DEFINE  06H

HSO_TIME       DEFINE  04H
HSO_COMMAND    DEFINE  06H

SBUF          DEFINE  07H
SBUF_RX        DEFINE  07H
SBUF_TX        DEFINE  07H

INT_MASK       DEFINE  08H
INT_MASK1      DEFINE  13H
INT_PEND       DEFINE  09H
INT_PEND1      DEFINE  12H

WATCHDOG       DEFINE  0AH

TIMER1         DEFINE  0AH
TIMER2         DEFINE  0CH
T2CAP_LO       DEFINE  0Dh
T2CAP_HI       DEFINE  0Ch

BAUD_RATE      DEFINE  0EH

IO_PORT0       DEFINE  0EH
IO_PORT1       DEFINE  0FH
IO_PORT2       DEFINE  10H
```

SP_STAT	DEFINE 11H
SP_CON	DEFINE 11H
IOS0	DEFINE 15H
IOS1	DEFINE 16H
IOS2	DEFINE 17H
IOC0	DEFINE 15H
IOC1	DEFINE 16H
IOC2	DEFINE 0BH
PWM_CONTROL	DEFINE 17H
ZERO	DEFINE 00h
ZEROh	DEFINE 00h
ZEROl	DEFINE 01h
WSR	DEFINE 14h
T2CNTC	DEFINE 0CH

## Cross Reference

### *rotator.out*

```
#####
# Archimedes Universal Linker V4.11/DOS      07/Feb/97 14:37:06 #
#
# Target CPU      = 8096                      #
# List file       = rotator.out                #
# Output file 1  = rotator.hex                 #
# Output format   = intel-standard             #
# Command line   = -f stejim19.lk (-c8096 rotator.obj stejim19
#                   cl8096 -Z(CODE)RCODE,CONST=200
#                   -Z(CODE)ROM,CODE,CDATA,ZVECT,CSTR,CCSTR,CSTART=
#                   2000
#                   -Z(DATA)REGISTERS,RAM=0
#                   -Z(DATA)DATA,IData,UDATA,CDATA,ECSTR,WCSTR,
#                   TEMP,SER_BUF=C000
#                   -Z(DATA)CSTACK=D800 -Fintel-standard -o
#                   rotator.hex -xsni -l rotator.out -p58)
#
# (c) Copyright Archimedes Software Inc. 1988 #
#####
```

```
*****
*          *
*          CROSS REFERENCE
*          *
*****
```

Program entry at : 4C74 Relocatable, from module : CSTARTUP

```
*****
*          *
*          MODULE MAP
*          *
*****
```

FILE NAME : rotator.obj	PROGRAM MODULE, NAME : rotator	ADDRESS	REF BY MODULE
ABSOLUTE ENTRIES	=====	=====	=====
Rcv_Ptr		002E	stejim19
ABSOLUTE LOCALS	=====	=====	
IR_BASE		001A	
R8		001A	
R9		001B	
R0		001C	
R1		001D	
R2		001E	
R3		001F	
R4		0020	
R5		0021	
R6		0022	
R7		0023	
Ax		0024	
Ah		0025	
A1		0024	
Bx		0026	
Cx		0028	
Dx		002A	
Ex		002C	

SP	0018
STACKPOINTER	0018
AD_RESULT_LO	0002
AD_RESULT_HI	0003
AD_COMMAND	0002
HSI_MODE	0003
HSI_TIME	0004
HSI_STATUS	0006
HSO_TIME	0004
HSO_COMMAND	0006
SBUF	0007
SBUF_RX	0007
SBUF_TX	0007
INT_MASK	0008
INT_MASK1	0013
INT_PEND	0009
INT_PEND1	0012
WATCHDOG	000A
TIMER1	000A
TIMER2	000C
T2CAP_LO	000D
T2CAP_HI	000C
BAUD_RATE	000E
IO_PORT0	000E
IO_PORT1	000F
IO_PORT2	0010
SP_STAT	0011
SP_CON	0011
IOS0	0015
IOS1	0016
IOS2	0017
IOC0	0015
IOC1	0016
IOC2	000B
PWM_CONTROL	0017
ZERO	0000
ZEROh	0000
ZEROl	0001
WSR	0014
T2CNTC	000C

SEGMENTS IN THE MODULE

=====

REGISTERS

Relative segment, address : 0000 - 002E

CSTACK

Relative segment, address :	D800 - DFFF
LOCALS	ADDRESS
Stack_Ptr	E000

SER\_BUF

Relative segment, address :	D44C - D49B	
ENTRIES	ADDRESS	REF BY MODULE
Rcv_Buffer	D44C	stejim19
LOCALS	ADDRESS	
Rcv_End	D49C	

ROM

Relative segment, address :	2000 - 20DC
LOCALS	ADDRESS
Rom_Start	2000
T_Overflow	2000
AD_Complete	2002
HSI_Available	2004
HSO	2006
HSI1	2008
Sftwr_timer	200A
Serial	200C
ExInt	200E
Trap_Vector	2010
OpCode	2012
Ti	2030
Ri	2032
HSI4	2034

T2cap	2036
T2overflow	2038
ExInt_Pin	203A
HSI_Full	203C
NMI	203E
Month	2040
Day	2042
Year	2044
CCR	2018
Main	208B
Rx_ISR	2090
Cont	20A4
Error_ISR	20A8
Software_Timer_ISR	20A9
STI_10	20CD

\*\*\*\*\*

FILE NAME : stejim19.r15  
PROGRAM MODULE, NAME : stejim19

SEGMENTS IN THE MODULE  
=====

CODE

Relative segment, address :	20DE - 4C71	REF BY MODULE
ENTRIES	ADDRESS	
set_shutdown	31A5	Not referred to
set_pwm	31D7	Not referred to
set_heater	30E7	Not referred to
set_enable	3173	Not referred to
set_dir	314B	Not referred to
set_brake	3119	Not referred to
read_volt_mon	2D20	Not referred to
read_temp	2C30	Not referred to
read_solar_sen	2F28	Not referred to
read_rate	3030	Not referred to
read_lin_temp	2E8A	Not referred to
read_heatstat	2B92	Not referred to
read_cur_mon	2DEC	Not referred to
putchar	2B48	Not referred to
putbyte_82050	2B63	Not referred to
putbyte	2B7E	Not referred to
pause	31ED	Not referred to
main	20DE	rotator CSTARTUP
initialize	26FD	Not referred to
init_Uart	2A80	Not referred to
encode	4C39	Not referred to
conv_to_temp	35FE	Not referred to
control	320C	rotator
build_command	21DB	Not referred to
RatEst	39F2	Not referred to
PosEst	3DC9	Not referred to
IVLLaw	3717	Not referred to
DataList	4205	Not referred to

UDATA

Relative segment, address :	C000 - D44B	REF BY MODULE
ENTRIES	ADDRESS	
uu	D400	Not referred to
uf0	D33E	Not referred to
uc0	D33C	Not referred to
timeout	D390	Not referred to
time_out	D356	Not referred to
time_index	D352	Not referred to
ssfac	D414	Not referred to
senstat	D326	Not referred to
s	D328	Not referred to
rate	D35C	Not referred to
rat_cx	D378	Not referred to
pwm_off	D320	Not referred to
pwm_lim	D322	Not referred to
pwm	D324	Not referred to
pos_c	D374	Not referred to
pos0	D3B4	Not referred to
pos	D360	Not referred to

port2	D436	Not referred to
port1	D435	Not referred to
port0	D434	Not referred to
pia_c	D439	Not referred to
pia_b	D438	Not referred to
pia_a	D437	Not referred to
p	D3EC	Not referred to
outsiz	D34A	Not referred to
out_index	D350	Not referred to
old_tics	D31A	Not referred to
ntic	D315	Not referred to
new_tics	D316	Not referred to
mode	D31E	Not referred to
loop_index	D354	Not referred to
krs	D340	Not referred to
kps	D34C	Not referred to
integ	D358	Not referred to
int_lim	D36C	Not referred to
heat_stat	D3B0	Not referred to
heat_min	D3A8	Not referred to
heat_max	D3AC	Not referred to
gray_width	D3A4	Not referred to
fold	D342	Not referred to
flux_min	D3A0	Not referred to
fine	D380	Not referred to
data_index	D34E	Not referred to
data	D312	Not referred to
dark	D338	Not referred to
command	D313	Not referred to
coarse	D37C	Not referred to
cnt	D314	Not referred to
clip	D370	Not referred to
awake	D33A	Not referred to
V5	D394	Not referred to
V12	D398	Not referred to
Temper	D3B8	Not referred to
Kp	D384	Not referred to
Ki	D388	Not referred to
Kd	D38C	Not referred to
Id3	D348	Not referred to
Id2	D346	Not referred to
Id1	D344	Not referred to
IRSRate	D368	Not referred to
EstRate	D364	Not referred to
Data_Tokens	D2AE	Not referred to
Data_Buffer	C052	Not referred to
D	D3D8	Not referred to
Current	D39C	Not referred to
Command_Ptr	C000	Not referred to
Command_Buffer	C002	Not referred to

\*\*\*\*\*

FILE NAME : c18096.r15  
 PROGRAM MODULE, NAME : CSTARTUP

SEGMENTS IN THE MODULE

=====

CSTACK  
 Relative segment, address : E000 - E7FF

CSTART  
 Relative segment, address : 4C74 - 4C80

-----  
 LIBRARY MODULE, NAME : exit

SEGMENTS IN THE MODULE

=====

CODE

Relative segment, address : 4C72 - 4C73		
ENTRIES	ADDRESS	REF BY MODULE
?C_EXIT	4C72	CSTARTUP
exit	4C72	Not referred to

```

LIBRARY MODULE, NAME : ctype

SEGMENTS IN THE MODULE
=====
CODE
  Relative segment, address : Not applicable
-----
CONST
  Relative segment, address : 07A8 - 0828
    ENTRIES          ADDRESS      REF BY MODULE
    _ctype_           07A8        stejim19
-----
LIBRARY MODULE, NAME : ?SEG_INIT_L00

SEGMENTS IN THE MODULE
=====
UDATA
  Relative segment, address : Not applicable
-----
IData
  Relative segment, address : Not applicable
-----
ECSTR
  Relative segment, address : Not applicable
-----
TEMP
  Relative segment, address : Not applicable
-----
DATA
  Relative segment, address : Not applicable
-----
WCSTR
  Relative segment, address : Not applicable
-----
CDATA
  Relative segment, address : Not applicable
-----
ZVECT
  Relative segment, address : Not applicable
-----
CCSTR
  Relative segment, address : Not applicable
-----
CONST
  Relative segment, address : Not applicable
-----
CSTR
  Relative segment, address : Not applicable
-----
RCODE
  Relative segment, address : 0200 - 025C
    ENTRIES          ADDRESS      REF BY MODULE
    ?SEG_INIT_L00     0200        rotator
                                CSTARTUP
-----
LIBRARY MODULE, NAME : ?LIB_VERSION_L00
  ABSOLUTE ENTRIES      ADDRESS      REF BY MODULE
  =====                =====      =====
  ?CL8096_3_00_L00      0000        stejim19
                                ctype
-----
LIBRARY MODULE, NAME : ?F_MUL_L01

SEGMENTS IN THE MODULE
=====
RCODE
  Relative segment, address : 025E - 030C
    ENTRIES          ADDRESS      REF BY MODULE
    ?F_MUL_L01       025E        stejim19
-----
LIBRARY MODULE, NAME : ?F_DIV_L01

```

SEGMENTS IN THE MODULE  
=====

RCODE

Relative segment, address	ENTRIES	ADDRESS	REF BY MODULE
: 030E - 03B1	?F_DIV_SWAP_L01	030E	stejim19
	?F_DIV_L01	0311	stejim19

-----  
LIBRARY MODULE, NAME : ?F\_ADDSUB\_L01

SEGMENTS IN THE MODULE  
=====

RCODE

Relative segment, address	ENTRIES	ADDRESS	REF BY MODULE
: 03B2 - 0505	?F_ADD_L01	03B8	stejim19
	?F_SUB_SWAP_L01	03B2	stejim19
	?F_SUB_L01	03B5	stejim19

-----  
LIBRARY MODULE, NAME : ?F\_UNPACK\_L01

SEGMENTS IN THE MODULE  
=====

RCODE

Relative segment, address	ENTRIES	ADDRESS	REF BY MODULE
: 0506 - 0559	?F_UNPACK_L01	0506	?F_MUL_L01 ?F_DIV_L01 ?F_ADDSUB_L01

-----  
LIBRARY MODULE, NAME : ?F\_ROUND\_L01

SEGMENTS IN THE MODULE  
=====

RCODE

Relative segment, address	ENTRIES	ADDRESS	REF BY MODULE
: 055A - 05B9	?F_ROUND_L01	055A	?F_MUL_L01 ?F_DIV_L01 ?F_ADDSUB_L01

-----  
LIBRARY MODULE, NAME : ?F\_PACK\_L01

SEGMENTS IN THE MODULE  
=====

RCODE

Relative segment, address	ENTRIES	ADDRESS	REF BY MODULE
: 05BA - 05F0	?F_PACK_L01	05BA	?F_MUL_L01 ?F_DIV_L01 ?F_ADDSUB_L01

-----  
LIBRARY MODULE, NAME : ?FLT\_TO\_LONG\_L01

SEGMENTS IN THE MODULE  
=====

RCODE

Relative segment, address	ENTRIES	ADDRESS	REF BY MODULE
: 05F2 - 0640	?FLT_TO_UL_SWAP_L01	05F2	Not referred to
	?FLT_TO_UL_L01	05F5	Not referred to
	?FLT_TO_SL_SWAP_L01	05F2	Not referred to
	?FLT_TO_SL_L01	05F5	stejim19

-----  
LIBRARY MODULE, NAME : ?LONG\_TO\_FLT\_L01

SEGMENTS IN THE MODULE  
=====

RCODE

Relative segment, address
: 0642 - 06A0

ENTRIES	ADDRESS	REF BY MODULE
?UL_TO_FLT_SWAP_L01	0663	Not referred to
?UL_TO_FLT_L01	0660	stejim19
?SL_TO_FLT_SWAP_L01	0645	Not referred to
?SL_TO_FLT_L01	0642	stejim19

---

LIBRARY MODULE, NAME : ?F\_SWAP\_TOS\_NOS\_L01

SEGMENTS IN THE MODULE

---

RCODE

Relative segment, address :	06A2 - 06B0	
ENTRIES	ADDRESS	REF BY MODULE
?F_SWAP_TOS_NOS_L01	06A2	?F_DIV_L01 ?F_ADDSUB_L01 ?FLT_TO_LONG_L01 ?LONG_TO_FLT_L01

---

LIBRARY MODULE, NAME : ?F\_UNPACK\_OPER\_L01

SEGMENTS IN THE MODULE

---

RCODE

Relative segment, address :	06B2 - 06C5	
ENTRIES	ADDRESS	REF BY MODULE
?F_UNPACK_OPER_L01	06B2	?F_UNPACK_L01 ?FLT_TO_LONG_L01

---

LIBRARY MODULE, NAME : ?LONG\_MUL\_L02

SEGMENTS IN THE MODULE

---

RCODE

Relative segment, address :	06C6 - 06EE	
ENTRIES	ADDRESS	REF BY MODULE
?UL_MUL_L02	06C6	Not referred to
?SL_MUL_L02	06C6	stejim19

---

LIBRARY MODULE, NAME : ?SL\_DIV\_L02

SEGMENTS IN THE MODULE

---

RCODE

Relative segment, address :	06F0 - 0716	
ENTRIES	ADDRESS	REF BY MODULE
?SL_DIV_SWAP_L02	06FA	stejim19
?SL_DIV_L02	06F0	Not referred to

---

LIBRARY MODULE, NAME : ?UL\_DIV\_L02

SEGMENTS IN THE MODULE

---

RCODE

Relative segment, address :	0718 - 0789	
ENTRIES	ADDRESS	REF BY MODULE
?UL_DIV_SWAP_L02	0724	?SL_DIV_L02
?UL_DIV_L02	0718	?SL_DIV_L02

---

LIBRARY MODULE, NAME : ?RES\_SIGN\_L02

SEGMENTS IN THE MODULE

---

RCODE

Relative segment, address :	078A - 07A7	
ENTRIES	ADDRESS	REF BY MODULE
?RES_SIGN_L02	078A	?SL_DIV_L02

---

```

*****
*          SEGMENTS IN DUMP ORDER
*
*****
SEGMENT      START ADDRESS    END ADDRESS   TYPE  ORG  P/N  ALIGN
=====      ====== ======  =====  ==  ==  =====
RCODE        0200      -      07A7       rel  stc  pos  1
CONST         07A8      -      0829       rel  flt  pos  1
ROM           2000      -      20DC       rel  stc  pos  2
CODE          20DE      -      4C73       rel  flt  pos  1
ZVECT         Not applicable      rel  flt  pos  1
CSTR          Not applicable      rel  flt  pos  1
CCSTR         Not applicable      rel  flt  pos  1
CSTART        4C74      -      4C80       rel  flt  pos  1
REGISTERS     0000      -      002E       rel  stc  pos  1
RAM           Not applicable      dse  flt  pos  0
DATA          Not applicable      rel  stc  pos  1
IDATA          Not applicable      rel  flt  pos  1
UDATA          C000      -      D44B       rel  flt  pos  1
CDATA          Not applicable      rel  flt  pos  1
ECSTR         Not applicable      rel  flt  pos  1
WCSTR         Not applicable      rel  flt  pos  1
TEMP          Not applicable      rel  flt  pos  1
SER_BUF        D44C      -      D49B       rel  flt  pos  1
CSTACK        D800      -      E7FF       rel  stc  pos  1

```

```

?CL8096_3_00_L00
  ENTRY : 7
?C_EXIT
  ENTRY : 6
?FLT_TO_LONG_L01
  MODULE : 9 10
?FLT_TO_SL_L01
  ENTRY : 9
?FLT_TO_SL_SWAP_L01
  ENTRY : 9
?FLT_TO_UL_L01
  ENTRY : 9
?FLT_TO_UL_SWAP_L01
  ENTRY : 9
?F_ADDSUB_L01
  MODULE : 8 9
?F_ADD_L01
  ENTRY : 8
?F_DIV_L01
  MODULE : 8 9
  ENTRY : 8
?F_DIV_SWAP_L01
  ENTRY : 8
?F_MUL_L01
  MODULE : 8 9
  ENTRY : 8
?F_PACK_L01
  ENTRY : 9
?F_ROUND_L01
  ENTRY : 8
?F_SUB_L01
  ENTRY : 8
?F_SUB_SWAP_L01
  ENTRY : 8
?F_SWAP_TOS_NOS_L01
  ENTRY : 9
?F_UNPACK_L01
  ENTRY : 8
?F_UNPACK_OPER_L01
  ENTRY : 10
?RES_SIGN_L02
  ENTRY : 10

```

```

?SEG_INIT_L00
    ENTRY : 7
?SL_DIV_L02
    ENTRY : 10
?SL_DIV_SWAP_L02
    ENTRY : 10
?SL_MUL_L02
    ENTRY : 10
?SL_TO_FLT_L01
    ENTRY : 9
?SL_TO_FLT_SWAP_L01
    ENTRY : 9
?UL_DIV_L02
    ENTRY : 10
?UL_DIV_SWAP_L02
    ENTRY : 10
?UL_MUL_L02
    ENTRY : 10
?UL_TO_FLT_L01
    ENTRY : 9
?UL_TO_FLT_SWAP_L01
    ENTRY : 9
CCSTR
    SEGMENT : 7 11
CDATA
    SEGMENT : 7 11
CODE
    SEGMENT : 4 6 11
CONST
    SEGMENT : 7 11
CSTACK
    SEGMENT : 3 6 11
CSTART
    SEGMENT : 6 11
CSTR
    SEGMENT : 7 11
Command_Buffer
    ENTRY : 6
Command_Ptr
    ENTRY : 6
Current
    ENTRY : 6
D
    ENTRY : 6
DATA
    SEGMENT : 7 11
DataList
    ENTRY : 5
Data_Buffer
    ENTRY : 6
Data_Tokens
    ENTRY : 6
ECSTR
    SEGMENT : 7 11
EstRate
    ENTRY : 6
IDATA
    SEGMENT : 7 11
IRSRate
    ENTRY : 6
IVLLaw
    ENTRY : 5
Id1
    ENTRY : 6
Id2
    ENTRY : 6
Id3
    ENTRY : 6
Kd
    ENTRY : 6
Ki
    ENTRY : 6
Kp
    ENTRY : 6
PosEst
    ENTRY : 5

```

```

RAM
    SEGMENT : 11
RCODE
    SEGMENT : 7 8 9 10 11
REGISTERS
    SEGMENT : 3 11
ROM
    SEGMENT : 3 11
RateEst
    ENTRY : 5
Rcv_Buffer
    ENTRY : 3
Rcv_Ptr
    ENTRY : 2
SER_BUF
    SEGMENT : 3 11
TEMP
    SEGMENT : 7 11
Temper
    ENTRY : 6
UDATA
    SEGMENT : 5 7 11
V12
    ENTRY : 6
V5
    ENTRY : 6
WCSTR
    SEGMENT : 7 11
ZVect
    SEGMENT : 7 11
_ctype_
    ENTRY : 7
awake
    ENTRY : 6
build_command
    ENTRY : 5
clip
    ENTRY : 6
cnt
    ENTRY : 6
coarse
    ENTRY : 6
command
    ENTRY : 6
control
    ENTRY : 5
conv_to_temp
    ENTRY : 5
dark
    ENTRY : 5
data
    ENTRY : 5
data_index
    ENTRY : 5
encode
    ENTRY : 4
exit
    ENTRY : 6
fine
    ENTRY : 5
flux_min
    ENTRY : 5
fold
    ENTRY : 5
gray_width
    ENTRY : 5
heat_max
    ENTRY : 5
heat_min
    ENTRY : 5
heat_stat
    ENTRY : 5
init_Uart
    ENTRY : 4
initialize
    ENTRY : 4

```

```
int_lim
    ENTRY : 5
integ
    ENTRY : 5
kps
    ENTRY : 5
krs
    ENTRY : 5
loop_index
    ENTRY : 5
main
    ENTRY : 4
mode
    ENTRY : 5
new_tics
    ENTRY : 5
ntic
    ENTRY : 5
old_tics
    ENTRY : 5
out_index
    ENTRY : 5
outsiz
    ENTRY : 5
p
    ENTRY : 5
pause
    ENTRY : 4
pia_a
    ENTRY : 5
pia_b
    ENTRY : 5
pia_c
    ENTRY : 5
port0
    ENTRY : 5
port1
    ENTRY : 5
port2
    ENTRY : 5
pos
    ENTRY : 5
pos0
    ENTRY : 5
pos_c
    ENTRY : 5
putbyte
    ENTRY : 4
putbyte_82050
    ENTRY : 4
putchar
    ENTRY : 4
pwm
    ENTRY : 5
pwm_lim
    ENTRY : 5
pwm_off
    ENTRY : 5
rat_cx
    ENTRY : 5
rate
    ENTRY : 5
read_cur_mon
    ENTRY : 4
read_heatstat
    ENTRY : 4
read_lin_temp
    ENTRY : 4
read_rate
    ENTRY : 4
read_solar_sen
    ENTRY : 4
read_temp
    ENTRY : 4
read_volt_mon
    ENTRY : 4
```

```

s
    ENTRY : 5
senstat
    ENTRY : 5
set_brake
    ENTRY : 4
set_dir
    ENTRY : 4
set_enable
    ENTRY : 4
set_heater
    ENTRY : 4
set_pwm
    ENTRY : 4
set_shutdown
    ENTRY : 4
ssfac
    ENTRY : 5
time_index
    ENTRY : 5
time_out
    ENTRY : 5
timeout
    ENTRY : 5
uc0
    ENTRY : 5
uf0
    ENTRY : 5
uu
    ENTRY : 5

*****
*          *
*      END OF CROSS REFERENCE      *
*          *
*****
```

Errors: none  
 Warnings: none

## Support Files

### *make.bat*

```
a8096 rotator.asm rotator.lst rotator.obj x s  
c-96 stejim19.c -ml -vl -z -P -e -g -L -F -p58 -I\rotator\c-196\h\  
xlink -f stejim19.lk
```

### *stejim19.lk*

```
-!           -Control.lnk-  
  
First define CPU -!  
-c8096  
  
-! Object File Name -!  
rotator.obj  
stejim19  
cl8096  
  
-! First allocate the ROM segments -!  
-Z(CODE)RCODE,CONST=200  
-Z(CODE)ROM,CODE,CDATA,ZVECT,CSTR,CCSTR,CSTART=2000  
  
-! Then allocate the internal ram segments -!  
-Z(DATA)REGISTERS,RAM=0  
-Z(DATA)DATA,IData,UDATA,CDATA,ECSTR,WCSTR,TEMP,SER_Buf=C000  
-Z(DATA)CSTACK=D800  
  
-! Put code in file FILE.HEX in intel standard format -!  
-Fintel-standard  
-o rotator.hex  
  
-! Generate Cross Reference, Segment Map, Module Map, and Index -!  
-xsni  
-l rotator.out  
  
-! Set Listing page breaks at 58 lines -!  
-p58
```

## **Compilation**

The Archimedes C-96 Cross-Compiler Kit was used for this project. This kit includes both a ‘C’ compiler (‘c-96.exe’), an assembler (‘a8096.exe’), and a linker (‘xlink.exe’). The process of compilation simply involves invoking the following DOS command:

```
make
```

This is a DOS ‘batch’ file which assembles file ‘rotator.asm’, compiles the ‘C’ source file ‘stejim19.c’, and links the two into an Intel hex file called ‘rotator.hex’. This hex file is manually uploaded to an appropriate ROM burner that supports the ‘Atmel 29C512’ EEPROM.