

## A Brief Description of Library Functions

### Contents

A Brief Description of Library Functions.....	1
Functions defined in c6xdskdigio.c .....	3
void <b>Init_C6713DSK</b> (void) .....	3
void <b>Init_encoders</b> (int chip) .....	3
void <b>read_encoders</b> (int chip,float *enc1,float *enc2,struct encoder_parameters encpars).....	3
void <b>Init_pwm</b> (int chip).....	4
void <b>out_PWM</b> (int chip,int channel,float u).....	4
int <b>read_parallel_data</b> (void) .....	4
void <b>write_parallel</b> (int outdata).....	5
Functions defined in dac7724.c .....	6
void <b>Init_DAC7724</b> (void).....	6
void <b>writeDAC7724</b> (float dac1,float dac2,float dac3,float dac4).....	6
Functions defined in ad7864.c .....	7
void <b>ADC7864_Start</b> (void) .....	7
void <b>ADC7864_Read</b> (float *adc1, float *adc2, float *adc3, float *adc4).....	7
Functions defined in i2c.c .....	8
void <b>Init_i2c</b> (void).....	8
void <b>Init_LCD</b> (unsigned char contrast) .....	8
void <b>LCDPrintfLine1</b> (char *format,...).....	8
void <b>LCDPrintfLine2</b> (char *format,...).....	9
int <b>fire_UltrasonicSensor</b> (int I2Caddress) .....	9
void <b>read_UltraSonicLight</b> (int *intensity, int I2Caddress).....	9
void <b>read_UltraSonicRange</b> (int *range1, int *range2,int I2Caddress) .....	9
void <b>Init_UltrasonicSensor</b> (int I2Caddress, int gain, int range).....	9
int <b>atmel_set_mode</b> (unsigned char mode).....	10
int <b>atmel_set_rcservo</b> (int first, int count, unsigned char *buf) .....	10
int <b>atmel_get_ir</b> (int first, int count, unsigned char *buf).....	10
int <b>atmel_get_adc</b> (int first, int count, unsigned char *buf).....	11
int <b>atmel_get_all</b> (unsigned char *buf) .....	11
Functions defined in max3100uart.c.....	13
void <b>Init_UART1and2</b> (int BaudRate).....	13
void <b>Init_Wireless</b> (void).....	13
void <b>WirelessSend</b> (char *buffer, int size).....	13
int <b>SmallSprintf</b> (char *dest, char *format,...).....	13
Functions defined in switch_led.c.....	14
unsigned int <b>get_switchstate</b> (void) .....	14
void <b>set_LEDstate</b> (unsigned int leds).....	14
Functions defined in RCservo.c .....	15
void <b>Init_RCservo</b> (void).....	15
void <b>out_RCservo</b> (int channel, float u) .....	15
Functions defined in sharpir.c.....	16
void <b>Init_SharpIRs</b> (void) .....	16
void <b>StartIRs</b> (void).....	16
void <b>ReadSharpIR</b> (int *Read1,int *Read2,int *Read3).....	16

Functions defined in dspvisioncolor50Hz_cLCD.c.....	17
void <b>Init_colorvision</b> (void).....	17
void <b>SendImagetoUART2every25calls</b> (volatile bgr **ptrImage) .....	17
void <b>SendBWImagetoUART2every25calls</b> (volatile unsigned char **ptrImage).....	17
void <b>set_image_start</b> (int* row,int* col,int* ratio).....	18
Functions defined in color_LCD.c.....	19
void <b>SendImagetoColor_LCD</b> (volatile bgr **ptrImage) .....	19
DSP/BIOS Functions .....	20
unsigned long <b>CLK_gettime</b> (void).....	20
unsigned long <b>CLK_gettime</b> (void).....	20
void <b>LOG_printf</b> (LOG_Handle log,char * format) .....	20
void <b>MEM_alloc</b> (int segid, size_t size, size_t align).....	20
void <b>*QUE_get</b> (QUE_Handle queue).....	21
void <b>QUE_put</b> (QUE_Handle queue, void *elem) .....	21
Bool <b>SEM_pend</b> (SEM_Handle sem, unsigned int timeout).....	21
void <b>SEM_post</b> (SEM_Handle sem).....	21
void <b>SWI_post</b> (SWI_Handle swi) .....	22
void <b>TSK_sleep</b> (unsigned int nticks) .....	22
void <b>TSK_yield</b> (void) .....	22

## Functions defined in c6xdskdigio.c

### void Init\_C6713DSK(void)

Parameters: None

Return value: None

Description: Initializes the DSP's external memory interface for the C6713 daughter card.

*Needs to be the first function you call in your main() function.*

Example:

```
void main(void) {
    Init_C6713DSK();
}
```

### void Init\_encoders(int chip)

Parameters: chip 1 or 2 indicating which LS7266(encoder) chip to initialize on the daughter card

Return value: None

Description: Call this function before using the read\_encoder functions.

Init\_encoders enables both optical encoder channels on the specified chip. Sets up the chip in X4 quadrature decode mode. Zeros the encoder count register.

Example:

```
void main(void) {
    Init_encoders(1); // Chip 1
    Init_encoders(2); // Chip 2
}
```

### void read\_encoders(int chip,float \*enc1,float \*enc2,struct encoder\_parameters encpars)

Parameters:

chip 1 or 2 indicating which LS7266 chip to read on the daughter card.

\*enc1 A pointer to a variable that will receive the chip's Enc. 1 value in radians.

\*enc2 A pointer to a variable that will receive the chip's Enc. 2 value in radians.

encpars A structure of type encoder\_parameters which contains the needed parameters to convert the channel's encoder count to radians.

See c6xdskdigio.h for a description of encoder\_parameters.

Possible choices for you to use in GE 420 and GE 423 are:

g\_standard\_furuta,  
g\_standard\_DCMotor,  
g\_standard\_GearMotor (The mobile robot in GE 423)

Return value: None

Description: This function reads both encoder channels of the specified chip and returns the values in radians with the pointers enc1 and enc2.

Example:

```
// Define global variables (or they could be local)
float enc1 = 0;
float enc2 = 0;
float enc3 = 0;
float enc4 = 0;

//Then inside your control code call
// chip 1 Also notice that enc1 and enc2 are passed as reference
read_encoders(1,&enc1,&enc2,g_standard_GearMotor);
// chip 2
read_encoders(2,&enc3,&enc4,g_standard_GearMotor);
```

## void Init\_pwm(int chip)

Parameters:

chip 1 or 2 indicating which CTS82C54 (PWM Timer) chip to setup on the daughter card

Return value: None

Description: Call this function before calling any of the other PWM functions. You will need to do this for each chip you are using.

**This function can only be called inside the main() function.**

Example:

```
void main(void) {
    Init_pwm(1); // Chip 1
    Init_pwm(2); // Chip 2
}
```

## void out\_PWM(int chip,int channel,float u)

Parameters:

chip 1 or 2 indicating which CTS82C54 (PWM Timer) chip to command on the daughter card.

channel 1 or 2 indicating which PWM channel to command.

u The control effort as a value between -10 and +10.

Return value: None

Description: This function commands the duty cycle of a 20KHz carrier frequency PWM output. You will be using this PWM signal to command a 3952 PWM AMP chip to drive your DC motor.

Example:

```
// define global variables (or they could be local)
float u1=0;
float u2=0;
float u3=0;
float u4=0;

// in your control code assign u1 through u4 to the desired output value
// this assignment will be related to a control equation or an open loop output

// then in your code output the values
out_PWM(1,1,u1); //PWM channel 1
out_PWM(1,2,u2); //PWM channel 2
out_PWM(2,1,u3); //PWM channel 3
out_PWM(2,2,u4); //PWM channel 4
```

## int read\_parallel\_data(void)

Parameters: None

Return value: 8bit integer value of the 8 parallel port input pins PD0 to PD7.

Description: This function return the states of the 8 parallel port input pins. PD0 corresponds to bit 0, PD1->bit 1, ..., PD7->bit7. Each pin has a pull-up resistor. So your switch should wired so that when closed it connects the pin to ground. Details of the pin connections can be found on page 3 of the c6xdigio daughter-card schematic:

[http://coecsl.ece.uiuc.edu/ge423/datasheets/c6713\\_dgio.pdf](http://coecsl.ece.uiuc.edu/ge423/datasheets/c6713_dgio.pdf).

Pin out summary:

Parallel port pin	Data bit
2	PD0
3	PD1
4	PD2
5	PD3
6	PD4
7	PD5
8	PD6
9	PD7
18-25	GND

Example:

```
int mydata = 0;

mydata = read_parallel_data();
if ((mydata & 0x20) == 0x00) {
    //Do code that should be run when bit PD5 is pulled low.
}
if ((mydata & 0x2) == 0x2) {
```

```

    //Do code that should be run when bit PD1 is pulled high.
}

```

### void write\_parallel(int outdata)

Parameters: int outdata, 8 bit value that will set the state of the 8 parallel port output pins. 5 of these pins are found on the parallel port connector X1. The remaining 3 pins are found on connector SV5. Details of the pin connections can be found on page 3 of the c6xdigio daughter-card schematic: [http://coecsl.ece.uiuc.edu/ge423/datasheets/c6713\\_dqio.pdf](http://coecsl.ece.uiuc.edu/ge423/datasheets/c6713_dqio.pdf).

Pin out summery:

Parallel port pin	Data bit
15	PS0
13	PS1
12	PS2
10	PS3
11	PS4
18-25	GND

Connector SV5	Data bit
6	PS5
8	PS6
10	PS7
1-2	GND

Example:

```

// for this example, each output bit is connected to a relay that activates a solenoid
int solenoid0 = 0;
..
int solenoid7 = 0;
int outstatus = 0;

write_parallel(outstatus); // initially turn all solenoid off

// later in a period function or task
if (solenoid4 == 1) {
    outstatus = outstatus | 0x10;
    write_parallel(outstatus);
}
if (solenoid7 == 1) {
    outstatus = outstatus | 0x80;
    write_parallel(outstatus);
}

```

## Functions defined in dac7724.c

### void Init\_DAC7724(void)

Parameters: None

Return value: None

Description: Use this function to Enable the DAC channels.

**This function can only be called inside the main() function.**

Example:

```
void main(void) {  
    Init_DAC7724();  
}
```

### void writeDAC7724(float dac1,float dac2,float dac3,float dac4)

Parameters:

dac1 The desired voltage value to be output on DAC channel 1.

dac2 The desired voltage value to be output on DAC channel 2.

dac3 The desired voltage value to be output on DAC channel 3.

dac4 The desired voltage value to be output on DAC channel 4.

All inputs are saturated between +10V and -10V.

Return value: None

Description: Use this function to command the DAC channels  
with a -10Volt to +10Volt output signal.

Example:

```
// define global variables (or they could be local)  
float dac1=0;  
float dac2=0;  
float dac3=0;  
float dac4=0;  
  
// in your control code assign dac1 and dac2 to the desired output voltage  
  
// then in your code output the values  
writeDAC7724(dac1,dac2,dac3,dac4);
```

## Functions defined in ad7864.c

### void ADC7864\_Start(void)

Parameters: None

Return value: None

Description: Use this function to Start an ADC conversion. When the conversion of all 4 ADC channels is complete, hardware interrupt 7 (HWI7) is set.  
*Must be called each time you want to sample the ADC channels*

Example:

```
void myprd_func(void) {  
    ADC7864_Start();  
}
```

### void ADC7864\_Read(float \*adc1, float \*adc2, float \*adc3, float \*adc4)

Parameters: 4 de-referenced floats. Each corresponding to one channel of the 4 ADC inputs. On return each of these floats will have a value between -10V and 10V.

Return value: None

Description: Use this function to Read the sampled ADC channels inside hardware interrupt 7's interrupt service routine.

Example:

```
// global variables  
float myadc1=0,myadc2=0,myadc3=0,myadc4=0;  
void my_HWI_INT7_func(void) {  
    ADC7864_Read(&myadc1, &myadc2, &myadc3, &myadc4);  
    // other code that uses the ADC values  
    ...  
}
```

## Functions defined in i2c.c

### void Init\_i2c(void)

Parameters: None

Return value: None

Description: Initializes i2c port. Must be called before LCD, Compass and UltraSonic functions. **This function can only be called inside the main() function.**

Example:

```
void main(void) {
    Init_i2c();
}
```

### void Init\_LCD(unsigned char contrast)

Parameters: contrast: value between 0 and 250 setting the contrast of the LCD screen. 250 is dark and 0 is light. 120 is a good starting value.

Return value: None

Description: Initializes LCD Screen. See this source and the LCD manual to determine starting conditions for the LCD screen. **This function can only be called inside the main() function.**

Example:

```
void main(void) {
    Init_i2c();
    Init_LCD(80);
}
```

### void LCDPrintfLine1(char \*format,...)

Parameters:

format A formatting string for ANSI C, i.e. "foo=%f"

... The values to be substituted into your string.

See any C book that discusses the "printf" instruction to understand these parameters better.

Return value: None

Description: Starts a task to send a string of characters to the LCD or other device.

The formatted string of characters will be sent to the top line of the LCD screen.

Example:

```
// this example prints out encoder channels 1 and 2 to the LCD screen inside a period function "myprd"
float enc1=0;
float enc2=0;

void main(void) {
    Init_i2c();
    Init_LCD(80);
    Init_encoders(1);
}

void myprd(void) {
    read_encoders(1,&enc1,&enc2,g_standard_DCMotor);

    // Print to 20 character of Line 1
    LCDPrintfLine1("e1=%.2f,e2=%.2f",enc1,enc2);
}
```

**void LCDPrintfLine2(char \*format,...)**

Parameters:

format A formatting string for ANSI C, i.e. "foo=%f"

... The values to be substituted into your string.

See any C book that discusses the "printf" instruction to understand these parameters better.

Return value: None

Description: Starts a task to send a string of characters to the LCD or other device.

The formatted string of characters will be sent to the bottom line of the LCD screen.

Example: See example for LCDPrintfLine1

**int fire\_UltrasonicSensor (int I2CAddress)**

Parameters: I2CAddress: 7bit I2C address of the UltraSonic Sensor

Return value: 0 if no error; 1 if error firing UltraSonic.

Description: **Note: This function can only be called inside a TSK function.** Commands the Ultrasonic sensor to take a measurement. Measurement takes approximately 75-85 ms.

Example: See Below

**void read\_UltraSonicLight (int \*intensity, int I2CAddress)**

Parameters: intensity, returned value of light intensity. Value between 0-255

I2CAddress: 7bit I2C address of the UltraSonic Sensor

Return value: None

Description: **Note: This function can only be called inside a TSK function.** Reads Light Intensity reading from UltraSonic sensor. Must wait 75-85ms after a fire command to read light.

Example: See Below

**void read\_UltraSonicRange (int \*range1, int \*range2, int I2CAddress)**

Parameters: range1, return value of first echoed distance.

Range2, return value of second echoed distance.

I2CAddress: 7bit I2C address of the UltraSonic Sensor

Return value: None

Description: **Note: This function can only be called inside a TSK function.** Reads first and second distance measurements from the UltraSonic sensor. Must wait 75-85ms after a fire command to read light.

Example: This example shows a way to program a TSK function to fire and read the Ultrasonic Sensors.

```
volatile int new_i2cdata = 0;
int ultral_dist1,ultral_dist2,ultral_light;
int ultra2_dist1,ultra2_dist2,ultra2_light;

void getI2CSensors(void) {
while(1) {
    if (new_i2cdata == 0) {
        while ( fire_UltrasonicSensor(0x70) == 1 ) {};
        TSK_sleep(100);
        read_UltraSonicLight((int *)&ultral_light,0x70);
        read_UltraSonicRange((int *)&ultral_dist1,(int *)&ultral_dist2,0x70);
        while ( fire_UltrasonicSensor(0x71) == 1 ) {};
        TSK_sleep(100);
        read_UltraSonicLight((int *)&ultra2_light,0x71);
        read_UltraSonicRange((int *)&ultra2_dist1,(int *)&ultra2_dist2,0x71);
        new_i2cdata = 1;
    } else {
        TSK_sleep(100);
    }
}
}
```

**void Init\_UltrasonicSensor(int I2CAddress, int gain, int range)**

Parameters: int I2CAddress: the address of the Devantech SRF08 UltraSonic sensor to interface

int gain: gain value between 1 and 31.

int range: range value between 1 and 255.

Return value: None

Description: Initialize Devantech SRF08 sensor. **Must be call in main() and after Init\_i2c().** See <http://coecsl.ece.uiuc.edu/ge423/DevantechSRF08UltraSonicRanger.pdf> for a

description of the gain and range parameters.

Example:

```
void main(void) {

    Init_UltrasonicSensor(0x70,40, 255);
    Init_UltrasonicSensor(0x71,40, 255);

}
```

### int atmel\_set\_mode(unsigned char mode)

Parameters: unsigned char mode: The mode of the RCservo or ADC pins of the Atmel chip on the robot's power board. Either enable all the 3 pin ports as RC servos (ATPWR\_MODE\_SERVOS) or as ADC inputs (ATPWR\_MODE\_ADCS)

Return value: number of bytes sent.

**Note: Currently all Atmels on the Robots are hard coded to be in the ATPWR\_MODE\_ADCS mode and cannot be commanded to change to the servo mode. If for some reason you need more than 6 RCservos ask your instructor to change the Atmel's firmware to allow this function to switch to the 14 RCservo mode!**

Description: Set the mode of the three pin ports of the ATMEL/Powerboard. **Note: This function can only be called inside a TSK function.**

Example: See atmel\_set\_rcservo function above.

### int atmel\_set\_rcservo(int first, int count, unsigned char \*buf)

Parameters: int first: first RCservo to set, 8 to 13  
int count: number of RCservos to set in order starting with first  
unsigned char \*buf: Character array that holds the values to send. Range is 0 to 255. You will have to calibrate each RC servo to find out what value corresponds to the RC servo's angle.

Return value: Number of bytes sent.

Description: Use this function to communicate over the I2C bus to the Atmel chip on the robot's power board. 6 RCservos can be controlled with this Atmel chip. **Note: This function can only be called inside a TSK function.**

Example:

```
void atmel_RCservo_task(void) {

    unsigned char RCbuf[14];
    TSK_sleep(500); // give I2C bus some initial rest time.
    atmel_set_mode(ATPWR_MODE_ADCS);

    while (1) {
        SEM_pend(&SEM_atmel_rcservo, SYS_FOREVER);

        RCbuf[0] = Atmel_RC1;
        RCbuf[1] = Atmel_RC2;
        // Could send more servos here
        atmel_set_rcservo(0, 2, RCbuf);

        TSK_sleep(50); // wait 50 ms before pending for next value
    }

}
```

### int atmel\_get\_ir(int first, int count, unsigned char \*buf)

Parameters: int first: first GP2D02 IR distance sensor to receive data from, 0 to 4  
int count: number of GP2D02 IR distance sensor to receive data from starting with first.  
unsigned char \*buf: Character array that receives the distances read. Range is 0 to 255. You will have to calibrate each IR distance sensor to find out what value corresponds to what distance.

Return value: 0 no Error, 1 Error

Description: **Note: This function can only be called inside a TSK function.** Use this function to communicate over the I2C bus to the Atmel chip on the robot's power board. 5 GP2D02 IR distance sensors can be read with this Atmel chip. The GP2D02 sensor takes about 70 ms to convert to a new reading. If you read faster than that you will receive duplicate readings and also clog up the I2C bus with unneeded data transfer. Also if you are using both the ATMEL IRS and the ATMEL ADCs it is better to use the atmel\_get\_all function.

Example:

```
void atmel_IRs_task(void)
{
    unsigned char IRbuf[10];
```

TSK\_sleep(1000); // give I2C bus some initial rest time.

```

while (1) {
    if (new_irdata_i2c == 0) {
        if ( atmel_get_ir(0, 5, IRbuf) == 0 ) {
            ir1_i2c = IRbuf[0];
            ir2_i2c = IRbuf[1];
            ir3_i2c = IRbuf[2];
            ir4_i2c = IRbuf[3];
            ir5_i2c = IRbuf[4];
            new_irdata_i2c = 1;
            TSK_sleep(70);
        } else {
            TSK_sleep(5);
        }
    } else {
        TSK_sleep(70); // error condition, should not get to this code
    }
}
}

```

### int atmel\_get\_adc(int first, int count, unsigned char \*buf)

Parameters: int first: first ADC channel to receive data from, 0 to 7  
int count: number of ADC channels to receive data from starting with first.  
unsigned char \*buf: Character array that receives the ADC readings. Range is 0 to 255. The Atmel ADCs are 10bit, so this function returns the most significant 8 bits of the ten bit conversion.

Return value: 0 no Error, 1 Error

Description: **Note: This function can only be called inside a TSK function.** Use this function to communicate over the I2C bus to the Atmel chip on the robot's power board. 8 ADC channels can be read with this Atmel chip. Each ADC channel is sampled every 8ms, but you should not read the channels that often if you are using the I2C bus for other peripherals. If you read too fast you will clog up the I2C bus. Also if you are using both the ATMEL IRS and the ATMEL ADCs it is better to use the atmel\_get\_all function.

Example:

```

void atmel_ADCs_task(void)
{
    unsigned char ADCbuf[10];

    TSK_sleep(1000); // give I2C bus some initial rest time.

    while (1) {
        if (new_adcdata_i2c == 0) {
            if ( atmel_get_adc(0, 4, ADCbuf) == 0 ) {
                adc1_i2c = ADCbuf[0];
                adc2_i2c = ADCbuf[1];
                adc3_i2c = ADCbuf[2];
                adc4_i2c = ADCbuf[3];

                new_adcdata_i2c = 1;
                TSK_sleep(70);
            } else {
                TSK_sleep(5);
            }
        } else {
            TSK_sleep(70); // error condition, should not get to this code
        }
    }
}

```

### int atmel\_get\_all(unsigned char \*buf)

Parameters: unsigned char \*buf: Character array that receives the ir, adc and hardware interrupt readings. Range is 0 to 255. The Atmel ADCs are 10bit, so this function returns the most significant 8 bits of the ten bit conversion.

Return value: 0 no Error, 1 Error

Description: **Note: This function can only be called inside a TSK function.** Use this function to communicate over the I2C bus to the Atmel chip on the robot's power board. 8 ADC channels, 5 GP2D02 IR distance sensors and the status of 2 hardware interrupts can be read with this Atmel chip. Talk to your instructor for more information on the 2 hardware interrupts. The GP2D02 sensor takes about 70 ms to convert to a new reading. If you read faster than that you will receive duplicate readings and also clog up the I2C bus with unneeded data transfer.

Example:

```
void atmel_IR_ADCs_task(void)
{
    unsigned char ALLbuf[15];

    TSK_sleep(1000); // give I2C bus some initial rest time.

    while (1) {
        if (new_alldata_i2c == 0) {
            if ( atmel_get_all(ALLbuf) == 0 ) {
                ir1_i2c = ALLbuf[0];
                ir2_i2c = ALLbuf[1];
                ir3_i2c = ALLbuf[2];
                ir4_i2c = ALLbuf[3];
                ir5_i2c = ALLbuf[4];
                adc1_i2c = ALLbuf[5];
                adc2_i2c = ALLbuf[6];
                adc3_i2c = ALLbuf[7];
                adc4_i2c = ALLbuf[8];
                adc5_i2c = ALLbuf[9];
                adc6_i2c = ALLbuf[10];
                adc7_i2c = ALLbuf[11];
                adc8_i2c = ALLbuf[12];
                int1_i2c = ALLbuf[13];
                int2_i2c = ALLbuf[14];
                new_alldata_i2c = 1;
                TSK_sleep(70);
            } else {
                TSK_sleep(5);
            }
        } else {
            TSK_sleep(70); // error condition, should not get to this code
        }
    }
}
```

## Functions defined in max3100uart.c

### void Init\_UART1and2(int BaudRate)

Parameters: BaudRate: Baud Rate for UART2. UART1 is fixed to 57600 for the Wireless Modem. UART2 can have three baud rates: BaudRate = 0 - 19200; BaudRate = 1 - 57600; BaudRate = 2 - 115200

Return value: None

Description: Initialize UART1 to a baud rate of 57600 bits/sec and UART2 to the specified rate. **This function can only be called inside the main() function.**

Example:

```
void main(void) {
    Init_UART1and2(2);
}
```

### void Init\_Wireless(void)

Parameters: None

Return value: None

Description: Call this function to setup the Wireless modem. It simply makes sure that UART1 is communicating with the Data port of the Wireless Modem. Call this once before calling WirelessSend. **This function can only be called inside the main() function.**

Example:

```
void main(void) {
    Init_UART1and2(2);
    Init_Wireless();
}
```

### void WirelessSend(char \*buffer, int size)

Parameters: buffer, pointer to a buffer of characters maximum size of buffer is 126. Size, number of chars to send

Return value: None

Description: Send the array of character pointed to by buffer over the wireless modem. This function adds the start character Chr(253) to the beginning of the string and the stop character Chr(255) to the end of the string.

Example: See SmallSprintf example

### int SmallSprintf(char \*dest, char \*format,...)

Description: Lower functionality version of the ANSI C standard sprintf. Has all the functionality we will need on the DSP and uses much less memory. Always use this function instead of sprintf.

Example:

```
// this example relates to the robots in Mechatronics
// this example sends a formatted string over the wireless card inside a period function "myprd"
float enc1=0;
float enc2=0;
char sendbuffer[40];

void myprd(void) {

    read_encoders(1,&enc1,&enc2,g_standardDCMotor);

    // First create the formatted string
    SmallSprintf(sendbuffer, "e1=%.2f,e2=%.2f",enc1,enc2);
    // then send the data over the wireless channel
    WirelessSend(sendbuffer,strlen(sendbuffer));
}
```

## Functions defined in switch\_led.c

### unsigned int get\_switchstate(void)

Parameters: None  
Return value: An integer value between 0 and 15 representing the state of the switches.  
Description: This function reads the state of the four DIP switches on the board.

Example:

```
int status=0;
status = get_switchstate();
switch (status) {
case 0:
    LCDPrintfLine1("Performing task 0");
    break;
case 1:
    LCDPrintfLine1("Performing task 1");
    break;
default:
    LCDPrintfLine1("State not Valid");
    Break;
}
```

### void set\_LEDstate(unsigned int leds)

Parameters: leds An integer between 0 and 15 representing the state of the four LEDs.  
Return value: None  
Description: This function sets the state of the four LEDs on the DSK board next to the four DIP switches.

Example:

```
int count = 0;

count = count +1;
set_LEDstate(count);
```

## Functions defined in RCservo.c

### void Init\_RCservo(void)

Parameters: None

Return Value: None

Description: This function sets up the second CTS82C54 timer chip on the C6XDIGIO daughter card to output a PWM signal specific to a RC servo motor. Note: Jumpers JP2 and JP3 need to have pins 1 and 2 jumpered together in order for the PWM signal to be used for an RC servo. **This function can only be called inside the main() function.**

Example:

```
void main(void) {
```

```
    Init_RCservo();
```

```
}
```

### void out\_RCservo(int channel, float u)

Parameters: int channel, Possible values 1 or 2. Determines which RC servo channel to command.

float u, Value between 0.6 and 3.0 (units milliseconds). This is the pulse width of the RC servo PWM signal. For an RC servo the length of the PWM pulse determines what angle the RC servo will turn to and stop.

Return Value: None

Description: This function controls the two RC servo channels produced by the second CTS82C54 chip on the C6XDIGIO daughter card.

Example:

```
float myvar1 = 1.2;
```

```
float myvar2 = 1.75
```

```
out_RCservo(1,myvar1);
```

```
out_RCservo(2,myvar2);
```

## Functions defined in sharpir.c

### void Init\_SharpIRs(void)

Parameters: None

Return Value: None

Description: This function sets up the TMS320C6713's GPIO pins 2,3,11 and 13 to be used to control three GP2D02 IR distance sensors. These pins are brought out on the camera daughter card. **This function can only be called inside the main() function.**

Example:

```
void main(void) {

    Init_SharpIRs();

}
```

### void StartIRs(void)

Parameters: None

Return Value: None

Description: This function starts the conversion of the three GP2D02 IR distance sensors attached to the TMS320C6713. This function must be called each time before reading the IR distance sensors. A conversion takes about 70 ms.

Example: See ReadSharpIR example.

### void ReadSharpIR(int \*Read1,int \*Read2,int \*Read3)

Parameters: The three distance readings from the three GP2D02 IR distance sensors that can be attached to the TMS320C6713. The integers are passed by reference in order that their value can be changed on return. The values range from 255 (close distance) to 0 (far distance).

Return Value: None

Description: Read the three GP2D02 IR distance sensors connected to the Camera Daughter Card.

Example:

```
int ir1,ir2,ir3;

void getIRs(void) {
    while(1) {
        if (new_irdata == 0) {
            StartIRs();
            TSK_sleep(70);
            ReadSharpIR(&ir1,&ir2,&ir3);

            new_irdata = 1; // tell period function that there is new data and do it only if period has received past
            //data

            TSK_sleep(5);
        } else {
            TSK_sleep(70); // error condition, should not get to this code
        }
    }
}
```

## Functions defined in dspvisioncolor50Hz\_cLCD.c

### void Init\_colorvision(void)

Parameters: None

Return Value: None

Description: Initializes the OV6620 CMOS camera and initializes the memory used for the images. **This function can only be called inside the main() function.**

Example:

```
void main(void) {
    void Init_colorvision();
}
```

### void SendImagetoUART2every25calls(volatile bgr \*\*ptrImage)

Parameters: pointer to a 2D multidimensional array of bgr (blue,green,red) values.

Return Value: none

Description: This function is passed a pointer to a 72 X 88 pixel color image. Because the serial port (UART2) is relatively slow (115200 bits/sec) when compared to the frame rate of the camera (25 frames per second) each captured image cannot be sent to the serial port. Instead, an image is sent to the serial every 25<sup>th</sup> time this function is called. So for 24 calls this function does nothing, but on the 25th call (and every 25 after that) the image is sent to the serial port. This equates to 1 frame sent to the serial port every second. The first frame the red pixels are sent, the second the green pixels and the third the blue pixels. So a full image is sent through the serial port in 3 seconds. **Note: This function cannot be called in the same picture frame if the function SendBWImagetoUART2every25calls is called.**

Example:

```
void userProcessColorImageFunc_laser(volatile bgr **ptrImage,volatile bgr **ptrLaser) {
    if (ptrImage != NULL) {
        SendImagetoUART2every25calls(ptrImage);
    }
}
```

### void SendBWImagetoUART2every25calls(volatile unsigned char \*\*ptrImage)

Parameters: pointer to a 2D multidimensional array of char (gray scaled) values.

Return Value: none

Description: This function is passed a pointer to a 72 X 88 pixel gray-scaled image. Because the serial port (UART2) is relatively slow (115200 bits/sec) when compared to the frame rate of the camera (25 frames per second) each captured image cannot be sent to the serial port. Instead, an image is sent to the serial every 25<sup>th</sup> time this function is called. So for 24 calls this function does nothing, but on the 25th call (and every 25 after that) the image is sent to the serial port. This equates to 1 gray scaled frame sent to the serial port every second. **Note: This function cannot be called in the same picture frame if the function SendImagetoUART2every25calls is called.**

Example:

```
volatile unsigned char **Thres_Image; // in Init_colorvision() memory is allocated for Thres_Image

void userProcessColorImageFunc_laser(volatile bgr **ptrImage,volatile bgr **ptrLaser) {
    if (ptrImage != NULL) {
        // Threshold the color image passed in ptrImage and save the Thresholded image in Thres_Image
        SendBWImagetoUART2every25calls(Thres_Image);
    }
}
```

```
void set_image_start(int* row,int* col,int* ratio)
```

Parameters: int \*row, starting row for a ratio = 2 image.  
int \* col, starting col for a ratio = 2 image.  
int \*ratio Image compression ratio. Either 2 or 4. If the compression ratio is 4, row and col are ignored. (Default value is 4)

Return Value: none

Description: The image functions for the OV6620 CMOS camera can either compress the image by 2 or by 4 times. The DSP functions are only written for an image of size 72 X 88 pixels. A divide by the ratio of 4 produces a 72 X 88 image so the row and col starting points are 0,0. But when divided by a ratio of 2 only a portion of the image (starting at row,col) can be processed. This allows the user to zoom in on a certain area of the image. All images processed after a call to set\_image\_start will use the new parameters.

Example:

```
int myrow;  
int mycol;  
int myratio;
```

```
myrow = 40;  
mycol = 25;  
myratio = 2;
```

```
set_image_start(&myrow,&mycol,&myratio);
```

## Functions defined in color\_LCD.c

**void SendImagetoColor\_LCD(volatile bgr \*\*ptrImage)**

Parameters: a pointer to a 2D multidimensional array of bgr (blue,green,red) values.

Return Value: none

Description: This function is passed a pointer to a 72 X 88 pixel color image. This function then sends this image to the small color LCD on top of the robot's daughter cards. The frame rate that can be achieved by the small LCD is 12.5 frames per second, so every other image passed to this function is sent to the small color LCD.

Example:

```
void userProcessColorImageFunc_laser(volatile bgr **ptrImage,volatile bgr **ptrLaser) {  
    if (ptrImage != NULL) {  
        SendImagetoColor_LCD(ptrImage);  
    }  
}
```

## DSP/BIOS Functions

This is a list of the most common DSP/BIOS (operating system level) functions used in Mechatronics. For a full list see the help file C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp. Go to Section

### **API Modules->Reference->Functions**

#### unsigned long CLK\_gettime(void)

Syntax  
currtime = CLK\_gettime();

Parameters  
void

Return Value  
unsigned long currtime /\* low-resolution time \*/

Description: Returns the amount of time elapsed since the start of your application. The units of this low-resolution timer are operating system CLK ticks.

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

#### unsigned long CLK\_gethtime(void)

Syntax  
currtime = CLK\_gethtime();

Parameters  
void

Return Value  
unsigned long currtime /\* high-resolution time \*/

Description: Returns the amount of time elapsed since the start of your application. The units of this high-resolution timer are actual timer counts.

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

#### void LOG\_printf(LOG\_Handle log,char \* format)

Syntax  
LOG\_printf(log, format);  
or  
LOG\_printf(log, format, arg0);  
or  
LOG\_printf(log, format, arg0, arg1);

Parameters  
LOG\_Handle log; /\* log object handle \*/  
String format; /\* printf format string \*/  
Arg arg0; /\* value for first format string token \*/  
Arg arg1; /\* value for second format string token \*/

Return Value  
void

Description: Print ints and chars to a LOG window in CCStudio.

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

#### void MEM\_alloc(int segid, size\_t size, size\_t align)

Syntax  
addr = MEM\_alloc(segid, size, align);

Parameters  
int segid; /\* memory segment identifier \*/

```
size_t size; /* block size in MADUs */
size_t align; /* block alignment */
```

Return Value

```
void *addr; /* address of allocated block of memory */
```

Description: Allocate memory from the heap for large variables (such as camera picture)

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

### void \*QUE\_get(QUE\_Handle queue)

Syntax

```
elem = QUE_get(queue);
```

Parameters

```
QUE_Handle queue; /* queue object handle */
```

Return Value

```
void *elem; /* pointer to former first element */
```

Description: Gets and removes the first item from the QUE.

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

### void QUE\_put(QUE\_Handle queue, void \*elem)

Syntax

```
QUE_put(queue, elem);
```

Parameters

```
QUE_Handle queue; /* queue object handle */
void *elem; /* pointer to new queue element */
```

Return Value

```
void
```

Description: Puts an item at the end of the QUE.

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

### Bool SEM\_pend(SEM\_Handle sem, unsigned int timeout)

Syntax

```
status = SEM_pend(sem, timeout);
```

Parameters

```
SEM_Handle sem; /* semaphore object handle */
unsigned int timeout; /* return after this many system clock ticks */
```

Return Value

```
Bool status; /* TRUE if successful, FALSE if timeout */
```

Description: Waits (is put to sleep) until a Semaphore is posted.

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

### void SEM\_post(SEM\_Handle sem)

Syntax

```
SEM_post(sem);
```

Parameters

```
SEM_Handle sem; /* semaphore object handle */
```

Return Value

```
Void
```

Description: Posts a semaphore (signals that the highest priority code pending on this semaphore may run).

See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->

**API Modules->Reference->Functions** for more information.

**void SWI\_post(SWI\_Handle swi)**

Syntax  
 SWI\_post (swi);

Parameters  
 SWI\_Handle swi; /\* SWI object handle\*/

Return Value  
 void

Description: Posts a Software Interrupt (signals that the software interrupt *swi* is allowed to run).  
 See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->  
**API Modules->Reference->Functions** for more information.

**void TSK\_sleep(unsigned int nticks)**

Syntax  
 TSK\_sleep (nticks);

Parameters  
 unsigned int nticks; /\* number of system clock ticks to sleep \*/

Return Value  
 void

Description: Suspends operation of a TSK for *nticks* operating system ticks, default tick length is a millisecond.  
 See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->  
**API Modules->Reference->Functions** for more information.

**void TSK\_yield(void)**

Syntax  
 TSK\_yield();

Parameters  
 void

Return Value  
 void

Description: Suspends current TSK and allows other TSKs to run. The yielding task will not be reentered until all other TSKs suspend, yield, sleep or complete.  
 See C:\CCStudio\_v3.3\bios\_5\_32\_03\packages\ti\bios\help\doc\c6xbios.hlp->  
**API Modules->Reference->Functions** for more information.