EVB Pin		Port Bit	Bit Addresses & Labels
1	2	1.	
		2.	
3	4	3.	
		4.	
5	6	5.	
		6.	P1.4	0x94 POT
7	8	7.	
		8.	
9	10	9.	
		10.	P1.2	0x92 CEX2
11	12	11.	P1.3	0x93 CEX3
		12.	P1.0	0x90 CEX0
13	14	13.	
		14.	P0.6	0x86 SDA
15	16	15.	P0.7	0x87 SCL
		16.	
17	18	17.	
		18.	
19	20	19.	
		20.	
21	22	21.	
		22.	
23	24	23.	
		24.	
25	26	25.	
		26.	
27	28	27.	
		28.	
29	30	29.	
		30.	
31	32	31.	P3.6	0xB6 SS0
		32.	P3.7	0xB7 SS1
33	34	33.	
		34.	
35	36	35.	
		36.	
37	38	37.	P3.1	0xB1 BILED1
		38.	P3.0	0xB0 BILED0
39	40	39.	
		40.	
41 ⟷ 60			

Software Initializations

A) Port I/0

```
P1MDIN &= 0xEF;
P1MDOUT &= 0xEF
P1MDOUT |= 0x0D;
P1 |= ~0xEF;
P3MDOUT &= 0x3F;
P3MDOUT |= 0x03;
P3 |= ~0x3F;
```

B) Timers

C) Interrupts

```
EA = 1;
EIE1 |= 0x08;
```

D) A/D

```
REF0CN = 0x03;
ADC1CN = 0x80;
ADC1CF |= 0x01;
```

E) PCA

```
PCA0MD =0x81;
PCA0CPM2 = 0xC2;
PCA0CN = 0x40;
```

F) XBAR

```
XBR0 = 0x27;
```

G) I2C

```
SMB0CR = 0x93;
ENSMB = 1;
```

compile derivatives
  #include <studio.h>
  #include <<stdlib.h>
  #include <c8051_SDCC.h>
  #include <i2c.h>

declare global variables
  const unsigned char POT_ADDRESS, ACCEL_ADDRESS.
  unsigned char ACCEL_DATA[4], AD_VALUE, OUTPUT_MODE, ACCEL_READ_INDICATOR.
  unsigned int DISTANCE, HEADING, DRIVE_MOTOR_PW, STEERING_SERVO_PW, PCA_COUNTER, DRIVE_MOTOT_MAX, DRIVE_MOTOR_MIN, STEERING_SERVO_LEFT, DIRECTION.
  unsigned int STEERING_SERVO_RIGHT, TARGET_HEADING, TARGET_DRIVE_MOTOR_GAIN, TARGET_STEERING_SERVO_GAIN, PCA_START, DRIVE_MOTOR_NEUT AND STEERING_SERVO_NEUT.
  int HEADING_ERRO, AVG_GX, AVG_GY, GX, GY, KDX, KDY, KS.
  sbit POT, SS0, SS1, CF,BILED0, BILED1.

function prototypes
  void Port_Init(void);
  void PCA_Init(void);
  void XBR0_Init(void);
  void Interrupt_Init(void);
  void SMB_Init(void);
  void ADC_Init(void);
  void Accel_Init_C(void) from i2c.h;
  void PCA_ISR(void) __interrupt 9;
  void Drive_Motor(void);
  void Speed_Controller(void);
  void Read_Accel(void);
  void Drive_Motor_Init(void);
  void Steering_Servo(void);
  void Direction_Controller(void);
  void Steering_Servo_Init(void);
  void Read_AD_Input(void);
  void POT_Reader(void);
  void Read_Keypad(void);
  void Flight_Recorder(void);
  unsigned int Is_SS_On(void);
  void Turn_BILED_Green(void);
  void Turn_BILED_Red(void);
  void Turn_BILED_Off(void);
  void Reset_PCA_Counter(void);
  void Wait_For_1s(void);

main function
  declare local variables
    NONE
  initialize system, ports and PCA_COUNTER
    Sys_Init();
    putchar(' ');
    Port_Init();
    PCA_Init();
    XBR0_Init();
    Interrupt_Init();

```
    SMB_Init();
    ADC_Init();
  Turn off the BILED.
  Read the data from POT using POT_Reader.
  initialize the drive motor using Drive_Motor_Init.
  initialize the steering servo using Steering_Servo_Init.
  Determine the value of SS1 and set the direction to 0 if reverse and 1 if forward.
  Begin infinite loop
    If the SS0 is on
      Turn BILED to green
      if ACCEL_READ_INDICATOR is 1:
        call Read_Accel for the initial value
      Begin infinite loop while the SS0 is on
        call Read_Accel
        call Speed_Controller
        call Drive_Motor
        call Direction_Controller
        call Steering_Servo
    End if
    Else if the SS0 is off
      read the input from the keypad
      set DRIVE_MOTOR_PW to DRIVE_MOTOR_MAX
      call Drive_Motor
      set STEERING_SERVO_PW to STEERING_SERVO_NEUT
      call Steering_Servo
    End if
  End infinite loop

functions
  void Drive_Motor(void)
    Set the low byte of the CEX2
    Set the high byte of CEX2
  End function

  void Speed_Controller(void)
    DRIVE_MOTOR_PW = DRIVE_MOTOR_NEUT + KDY * GY + KDX * abs(GX).
  End function

  void Read_Accel(void)
    clear the variable AVG_GX and AVG_GY
    For 4 iterations
      Read status_reg_a into Data[0]
      If the 2 LSbits are high: (Data[0] & 0x03) == 0x03, then continue, otherwise reread the status.
      Read 4 registers starting with 0x28.
      i2c_read_data(addr_accel, 0x28|0x80, Data, 4);
      Clear sums: avg_gx = 0, avg_gy = 0
      Accumulate sum for averaging.
        AVG_GX += ((Data[1] << 8) >> 4); //a simple â€œ<< 4â€    WILL NOT WORK;
        AVG_GY += ((Data[3] << 8) >> 4); //it will not set the sign bit correctly
      Done with 4 iterations
    Finish calculating averages
      AVG_GX = AVG_GX/4.
      AVG_GY = AVG_GY/4.
    Set global variables and remove constant offset, if known.
```

```
        GX = AVG_GX.
        GY = AVG_GY.
End Read_Accel(void)

void Drive_Motor_Init(void)
    Set the PW of the drive motor to neutal
    Turn the drive motor on for neutral
    Wait for 1s
End function

void Steering_Servo(void)
    Update the low byte of CEX0
    update the high byte of CEX0
End function

void Steering_Servo_Init(void)
    Set the PW of the steering servo to neutral
    Turn the steering servo on for neutral
    Wait for 1s
End function

void Direction_Controller(void)
    STEERING_SERVO_PW = STEERING_SERVO_NEUT - KS * GX.
End function

void Read_AD_Input(void)
    Set P1.4 as the analog input for the POT
    Clear the "Conversion Completed" flag
    Initiate A/D conversion
    Wait for conversion to complete
    Set the AD_VALUE
End function

void POT_Reader(void)
    Read the AD_VALUE using Read_AD_Input().
    calculate the max of the drive motor
    calcualte the min of the drive motor.
    calculate the right PW for the steering servo.
    calculate the left PW for the steering servo.
End function

void Read_Keypad(void)
    Get the target heading from the keypad
    If the value is out of range
        Ask the user to input another value
    End if
    Get the KS from the keypad
    Get the KDX, KDY from the keypad
End function

void Flight_Recorder(void)
    if the MODE is not 1
        Output to user-friendly UI.
    End if
```

```
    else
        Print GX, GY, DRIVE_MOTOR_PW, STEERING_SERVO_PW
    End if
End function

unsigned int Is_SS_On(void)
    if SS is on
        return 1
    End if
    else if SS is off
        return 0
    End iff
End function

void Turn_BILED_Green(void)
    turn red LED off
    turn green LED on
End function

void Turn_BILED_Red(void)
    turn red LED on
    turn green LED off
End function

void Turn_BILED_Off(void)
    turn red LED off
    turn green LED off
End function

void PCA_Init (void)
    Enable SYSCLK/12 and enable interrupt.
    Enable CCM2 16bit PWM.
    Enable PCA counter.
    For 20ms period.
End function

XBR0_Init(void)
    Configure crossbar with UART, SPI, SMBus, and CEX channels.
End function

void Interrupt_Init(void)
    Enable general interrupt.
    Enable PCA overflow interrupts.
End function

void SMB_Init(void)
    Set the clock frequency to be 100kHz.
    Enable SMB.
End function

void ADC_Init(void)
    configure ADC1 to use VREF.
    Set a gain of 1.
    Enable ADC1.
```

```
End function

void PCA_ISR(void) __interrupt 9
   increment the PCA_COUNTER
   if 20ms has passed
      set ACCEL_READ_INDICATOR to 1
   End if
   if 1s has passed and switch is on
      call Flight_Recorder
   End if
   if CF
      Clear overflow flag.
      start count for 20ms period
   End if
   handle other PCA0 overflows
End function

void Reset_PCA_Counter(void)
   reset PCA_COUNTER to 0
End function

void Wait_For_1s (void)
   reset PCA_COUNTER
   Begin infinite loop while PCA_COUNTER less than 51
   wait for 1s
   End infinite loop
   reset PCA_COUNTER
End function
```