Helicopter Simulation Design Manual

By John Massy-Greene(z3215661)

Data Structures

This section shall describe the data structures used in designing the helicopter simulation, how they were implemented in terms of assembly code as well as reasons for the method of implementation.

Data Structure	Assembly Implementation	Reason For Implementation/Notes
X Position	xposH:xposL(r2:r3)	Positions in the helicopter simulation are stored in terms of millimetres. The room the helicopter flys in is a 50x50x10 meter. There is a 1000 millimeter to each meter. So the maximum value for a position is 50,000, which requires 16 bits or two registers each.
Y Position	yposH:yposL(r:4:r5)	
Z Position	zposH:zposL(r6:r7)	
Direction of helicopter	Direction (r8)	There are 6 possible ways the helicopter can move so one register was used to store the ASCII value of the direction of the helicopter
Speed of helicopter	Speed(r9)	4 possible speeds of the helicopter. So one register was used.
Signal to update the LCD screen	Update(r12)	While the program is running, Timer 0 needs a way to tell the main program to tell it to update the LCD with a possible new position or speed of the helicopter. Update was implemented to be used by both functions as method of communication.
Signal to increase/decrease speed	SpeedChange(r10)	This register is used by the buttons PB0 and PB1 to communicate with each other and Timer 0. PB0 sets this register with a value of 1 to increase speed. PB1 sets this value with -1 to decrease speed. Each interrupt routine makes sure the register has a value of 0 before setting the value. Timer 0 checks this value to see if it should time 0.5seconds before updating the speed.
Helicopter Statuses	Finehelicontrol(r11) – “fine-tuned helicopter control”	The helicopter can be in multiple states throughout its flight. Depending on the state of the helicopter certain actions can or cannot be taken. For example if the helicopter stops hovering it needs to go in the same direction and speed it was before it started to hover. Therefore in hover mode, keypad direction keys and

		speed increase buttons are disabled by checking the state of the helicopter and making no changes to the appropriate registers. These states are described below.
Takeoff Boolean	Finehelicontrol(r11 bit 1)	Setting this bit to 0 means the helicopter is landing. Setting this bit to 1 means the helicopter is taking off. Each state changes the direction and speed of the helicopter.
Hover Boolean	Finehelicontrol (r11 bit 2)	When the helicopter is hovering, the speed, direction and position of the helicopter cannot be changed. So the functions responsible for these changes must check this bit every time they are called.
Crash Boolean	Finehelicontrol(r11 bit 3)	When the helicopter exceeds the limits of the room, this bit is set. The main function of the program uses this bit to determine whether to disable further use of the helicopter and to flash the LEDS.
Initial Takeoff Boolean	Finehelicontrol(r11 bit 4)	When the board is turned on and the program is loaded, the START message will appear. The program needed a boolean to know when to change the LCD screen to display flight information as well as differentiate the difference between being in take-off mode during the flight and take-off mode at the start.
Successful Landing Boolean	Finehelicontrol(r11 bit 5)	The main function of the program uses this bit to determine when to stop the helicopter functioning and to print the flight time and distance.
LED pattern	LEDS(r13)	Holds the value used by Timer 0 to send to LEDs so the user can know when the helicopter has crashed.
Total time in milliseconds	thouL:thouH(r14:r15)	Every time Timer 0 has an interrupt this variable is updated by 1. This value is used because at the end of the simulation the user will want to know how many seconds their helicopter was flying for. So this value is tracked till it reaches 1000 before the variable totaltime is updated by 1.
Total distance in seconds	totalTime(2 bytes in data memory)	This variable keeps track of the total time the helicopter is in the air
Total distance in millimeters	dm (2 bytes in data memory) Note: originally tried to name this value	This variable is updated every time Timer 0 interrupts with the value of the current speed. When this variable

	distancemm but the assembler seems to have a problem with long data memory names	reaches 1000 it is cleared and the dt variable is increased by 1.
Total distance in meters	dt (2 bytes in data memory) Note: short for “distance total”.	This variable is updated by 1 every time the variable dm reaches 1000. It is used at the end of the program to tell the user how far they have flown the helicopter.
0.5 second wait time	Speedwait(2 bytes in data memory)	Timer 0 is set to interrupt every 1024 microseconds. 1000 interrupts is 1 second. So 500 interrupts is half a second. Speedwait is used by Timer 0 to measure time when the user wants to change the speed of the helicopter. It also doubles in usage for when the board needs to flash the LEDs on and off in a slow enough fashion that the user can see.
Data to be written to LCD	Data(r19)	A register dedicated solely to sending data/commands to the LCD.

Module Specification

In this section, we shall describe all the functions or macros used within the simulation. Information provided includes: function/macro name, inputs and outputs and description. See the table below

Function Name	Input	Output	Description
calculatepositon	xposH:xposL yposH:xposL zposH:zposL	ASCII value of the x, y or z position	This macro takes the values of the x, y or z positions which are in millimetres and converts it into the nearest meter. It then calculates the value of each digit and converts the digit into an ASCII value and sends that value to the LCD screen.
changemotspeed	Parameter 0 – a PWM value	Changes the PWM value in Timer 5	Takes a values and changes the PWM value in Timer 5 in order to increase/decrease the RPS of the motor.
clearhalfsec	Data memory location	Sets the value of a variable to 0	Takes a data memory location and clears the variable stored at that location.
lowstatus	Value of PORT C pins	N/A	Checks the value of the port C pins to make sure that keypad has been properly de-bounced.
addposition	- xposH:xposL or - yposH:yposL and - speed	New X or Y position Changes the crashed Boolean in the helicopter status register	This macro takes in the speed the helicopter is flying at and adds the value to the x or y positions to change the location of the helicopter. If the x or y locations exceed 50,000, it means the helicopter has crashed, so the macro sets the crashed bit of the helicopter status register to 1.
addZposition	speed	New Z position	Takes in speed of the helicopter and adds it to the current z position to find the new location of the helicopter. If the new value exceeds 10,000 the helicopter has crashed. The function sets the crashed bit of the helicopter status register to 1.

subposition	- xposH:xposL or - yposH:yposL and - speed	New x or y location	Takes the speed of the helicopter and subtracts the current position to find the new location of the helicopter. Used for when the helicopter is travelling backwards or to the left. If the speed is more than the current location it means the helicopter has crashed and thus the crashed bit of the helicopter status register will be set.
subZposition	Speed	New z position	Takes the speed of the helicopter and subtracts the current z position. If the speed is more than the current z position it means the helicopter has reached the ground. It then sets the landed boolean of the helicopter status register.
printdt	Data memory: - Total Time - Dt	Ascii value of each digit of a variable	This function takes in either the variable Total time or dt(total distance in meters) and finds each digit in the integer value(up to 9999), converts it into the appropriate ASCII value and sends that value to the LCD.
IRQ0	- User input - Speedchanges - Speed - helicopter status register	Sets the value of speedchanges to 1	The interrupt checks the helicopter status register to make sure the crashed and hover bits are cleared and checks that initial take off bit is set. If these bits aren't appropriately set or cleared the interrupt will end. Also checks that the speedchange register isn't already set by itself or IRQ1. If the value is set the interrupt will end. Lastly it will check that the value of speed isn't 4 so the user can't exceed the maximum speed.

			If the interrupt hasn't already ended, it will set the value of the speed change register to 1.
IRQ1			Interrupt 1 works the exact same way as interrupt 0 except that it will change the speedchange register to -1
Timer 0_INT	- helicopter status register - speedchanges - data memory variable speedwait - LED register	- sets the update variable to 1 - Sends the value of LED register to the LED hardware	If the speedchange register isn't cleared it will start adding 1 to the speedwait variable each time the interrupt is called until it reaches 500. 500 indicates that half a second has occurred. When half a second has occurred depending on whether the helicopter has crashed or the user wants to increase the speed, the interrupt will either send the value stored in the LED register to the LEDs or will call the dospeedupdate function. Depending on how the hovering, landing, or initial take off bits of the finehelicontrol register is set it will call the functions addtotaltime, addtotaldistance and findposition. Also sets the update register in order to communicate with the main function that the LCD screen needs to be refreshed with new information.
main	- Finehelicontrol register - Update register	- Can disable Timer 0 - And set Timer 5 PWM signal to 0	This function constantly checks the finehelicontrol register to check if the helicopter has landed or crashed. If the landed bit has been set it will disable Timer 0 and set the PWM of Timer 5 to 0. It will then call the printstats function

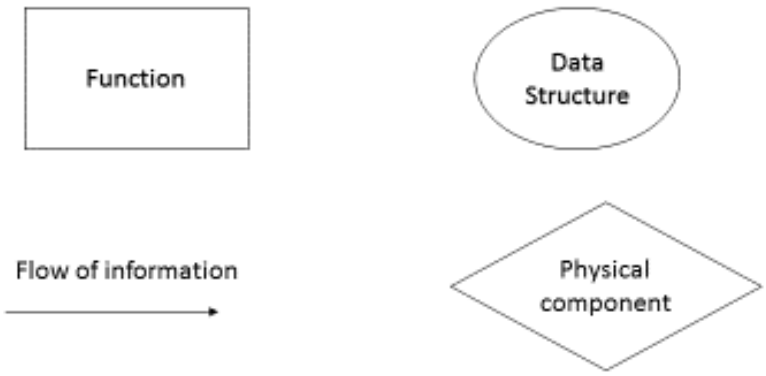
			If the crashed bit has been set it will set the PWM of Timer 5.
updateLCD	- xposH:xposL - yposH:yposL - zposH:zposL - speed - distance	- ASCII values to LCD - Clears update register	Calls the macro calculateposition to send the x,y and z positions to the LCD. Converts the values stored in the distance and speed registers into ASCII value and sends the ASCII value to the LCD.
dospeedupdate	- Speedchanges - Speed	- Calls the macro changemotspeed - Clears the value in speedchanges	Adds or subtracts the value stored in the speed register with the value stored in the speedchange. Uses the new speed value to call the macro changemotspeed.
Keypadcheck	- finehelicontrol - user input	- direction value - hover bit of finehelicontrol	Depending on which button the user presses, the function will either: change the value stored in the direction register, clear or set the hover bit of the finehelicontrol or call the takeoffOrLand function.
Findposition	- direction	N/A	Uses the current value stored in the direction register to determine which macro to call. The macros that it can call are: - addposition - subposition - addZposition - subZposition
takeoffOrLand	- finehelicontrol	- direction - speed - finehelicontrol - Timer 5 PWM value - Timer 0 mask - LCD	If the crashed, hover or landed bits of finehelicontrol are set then the function will end. If the takeoff bit is set to 0 the function will set it to 1. It will also set the direction to up and the speed to 2. It will also call the macro changemotspeed to reflect the new speed.

			If the takeoff bit is set to 0 and the initial take off bit is set to 0 it will change the initial take off bit to 1. It will then clear the screen in order to ready it to display information for the helicopters flight. It will then call the functions write_line_one and updateLCD. If the takeoff bit is set to 1 it will set the bit to 0 and change the direction to down, the speed to 1 and call changemotspeed to reflect the new speed.
write_line_one	N/A	LCD message: "POS DIR SPD"	Writes the top line of the LCD screen which explains what the values on the second line of the LCD screen means.
write_start	N/A	LCD message: "START:"	Writes the message "START:" on the LCD
addtotaltime	- thouL:thouH - data memory variable totalTime	- Add 1 or clears thouH:thoL - Adds one to totalTime	Adds one to the register pair thouH:thouL each time the function is called till it reaches 1000. Once it reaches 1000 it will clear the register pair and add 1 to totalTime.
addtotaldistance	Data memory variable dm	Add 1 or clears dm	Add one to the variable dm each time the function is called until it reaches 1000. Once dm reaches 1000 it will be cleared and the function addmeter will be called.
addmeter	N/A	Add 1 to dt	This function adds 1 to the data memory variable dt. **NOTE:** The programmer would like to note that they wanted to include the functionality of addmeter to addtotaldistance but the assembler seemed to have trouble with trying to address multiple locations in data memory in the same function so creating multiple functions for separate data memory addresses appeared

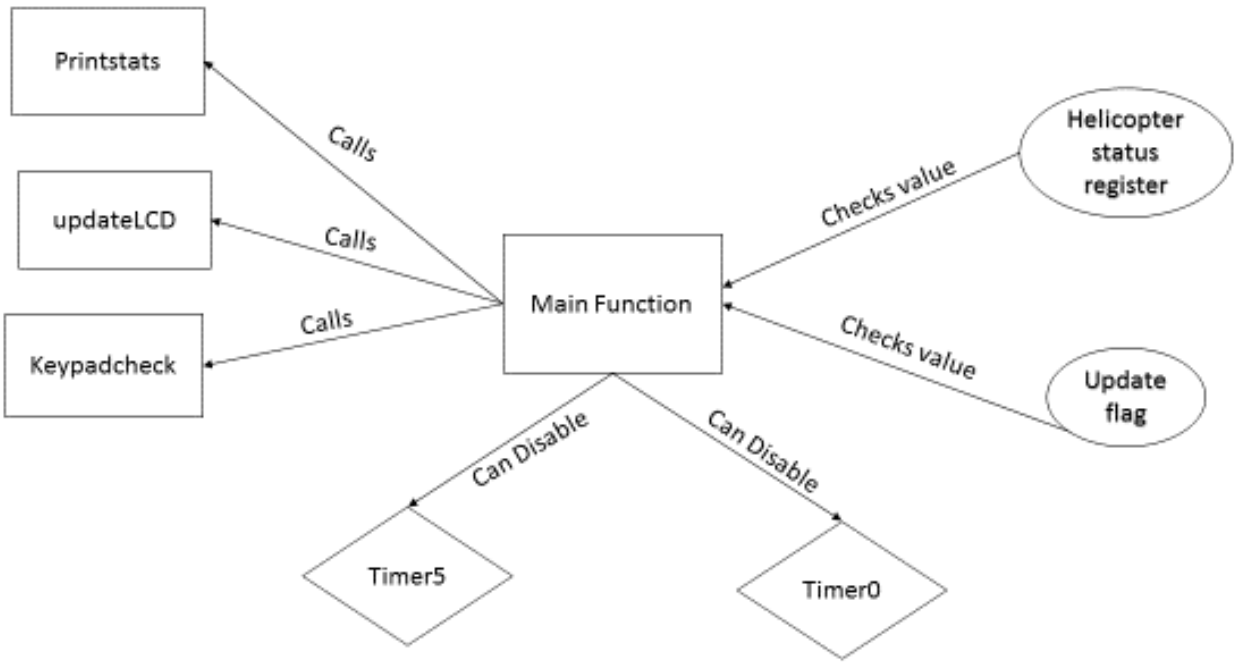
			to be the only viable solution for accurate calculations.
printstats	N/A	- Writes the message: “DISTANCE:” to the first line of the LCD. - Writes the message “DURATION:” to the second line of the LCD.	Writes the messages “DISTANCE:” and “DURATION:” to the LCD screen.
print_total_time	Data memory variable totalTime	N/A	Loads the variable totaltime and passes the value to the macro printDT for processing.
print_total_distance	Data memory variable dt	N/A	Loads the variable dt and passes the value to the macro printDT for processing.

SYSTEM CONTROL FLOW

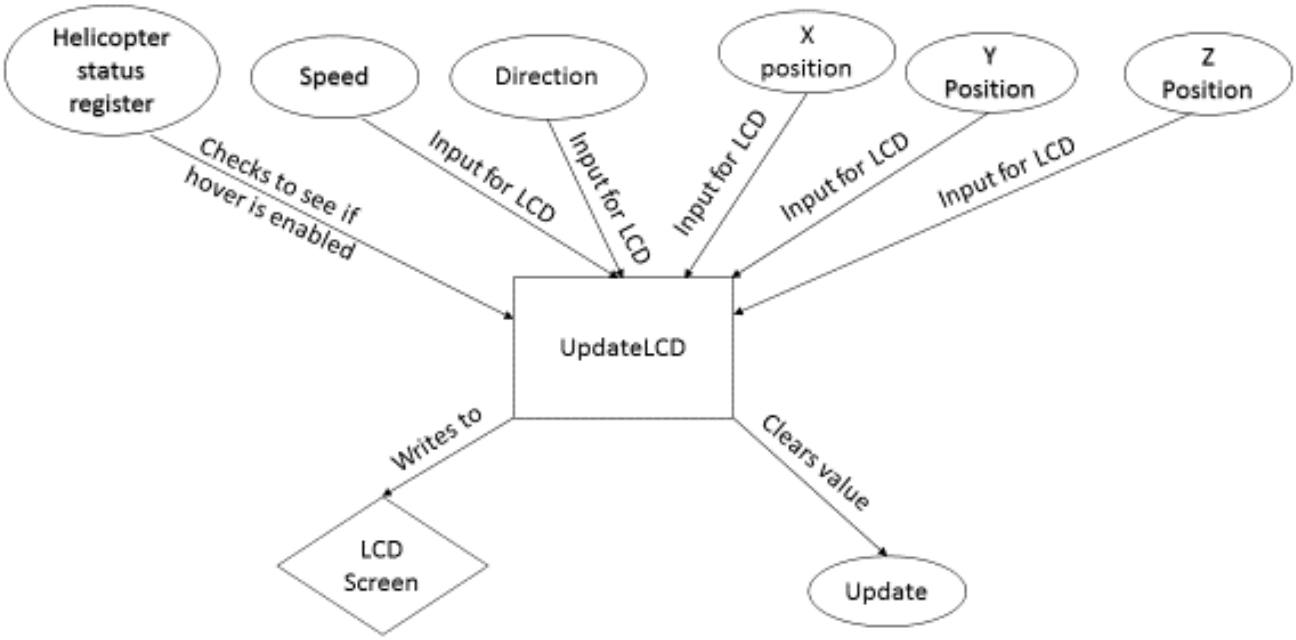
This section will graphically show a high level abstraction of the interactions between the major functions of the helicopter simulation program. It also shows the various inputs and outputs of these functions. In order to understand the diagrams please see the guide below that explains what each symbol means.



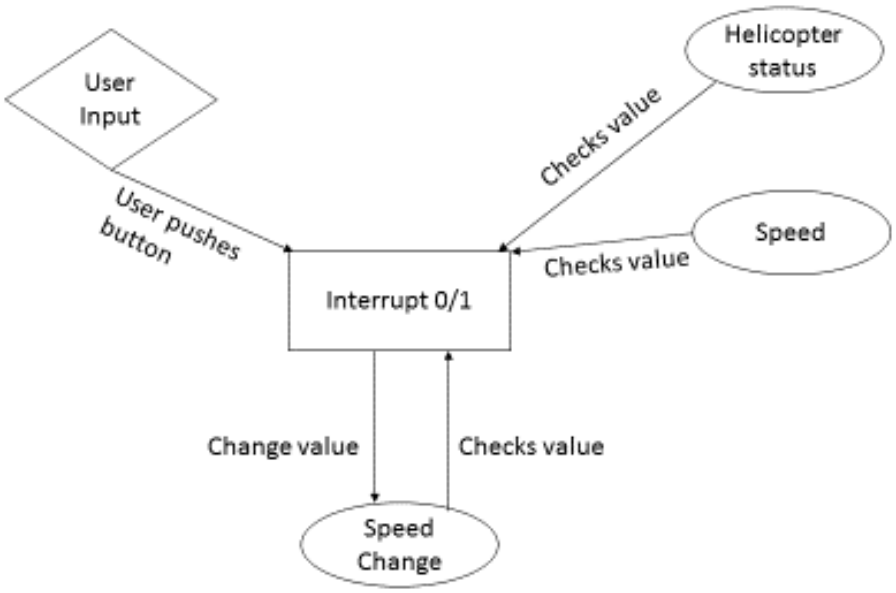
Main Function



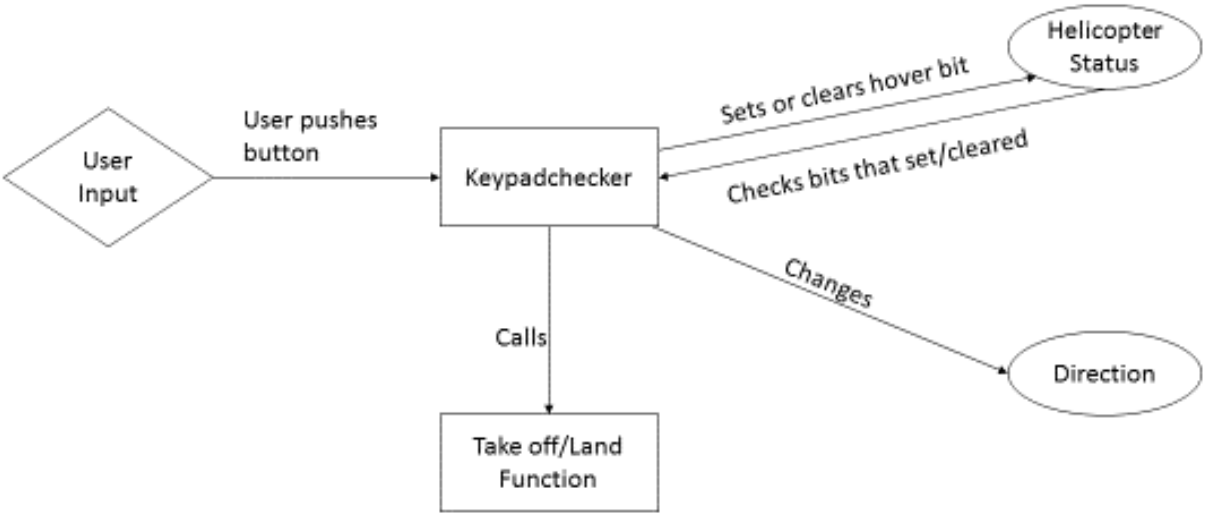
UpdateLCD Function



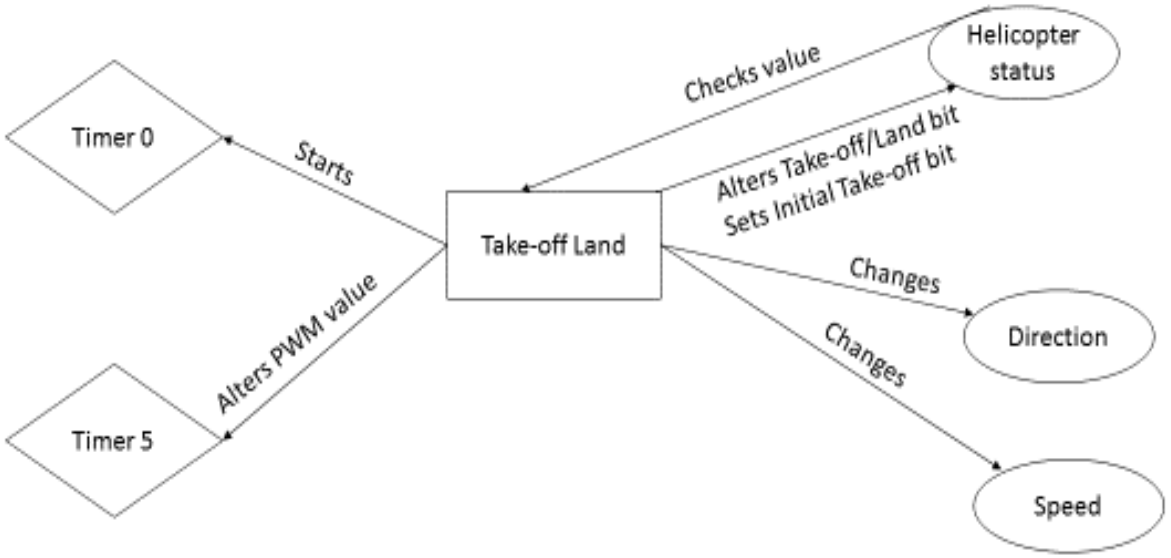
Interrupts 0 and 1



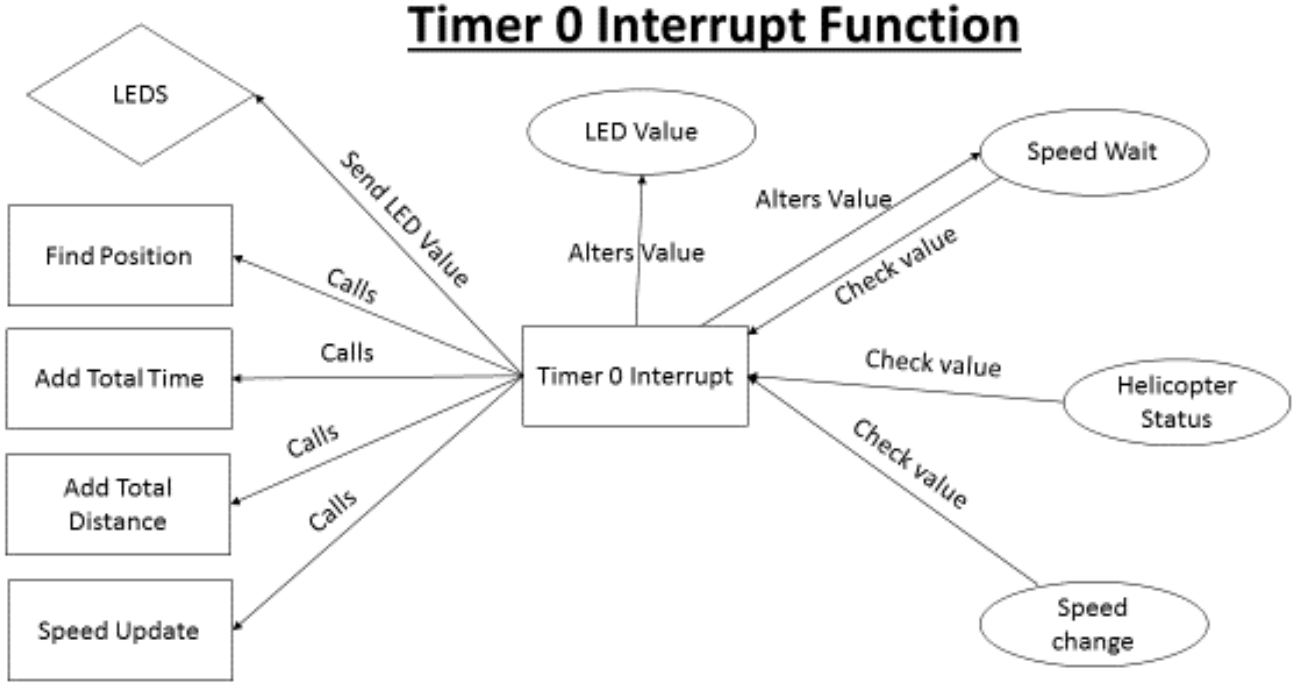
Keypadcheck Function



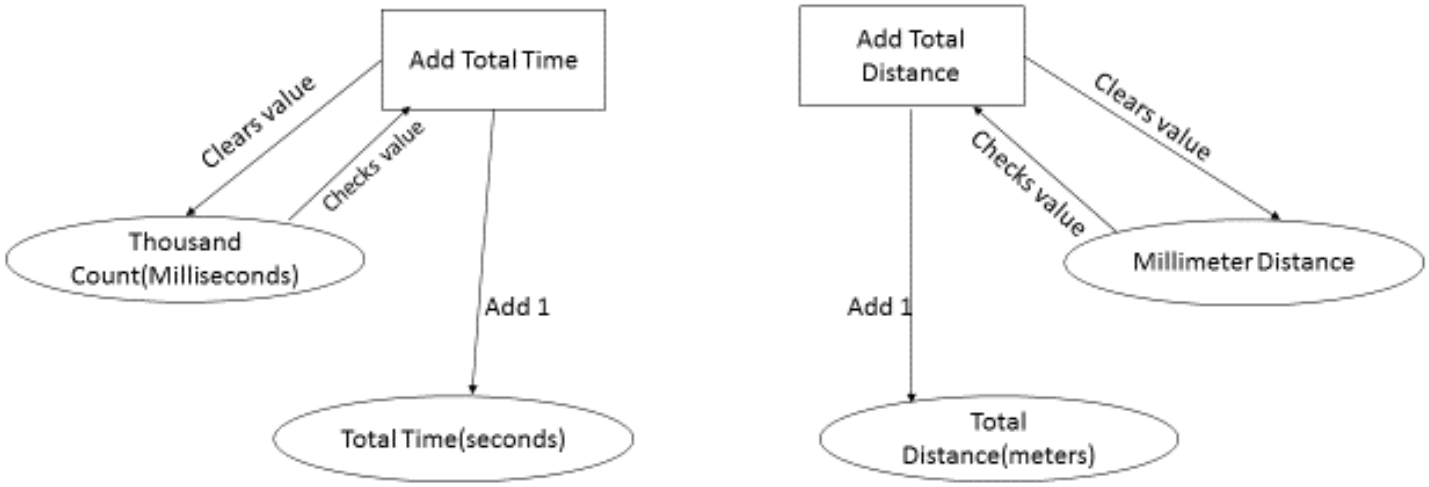
Take off/Land Function



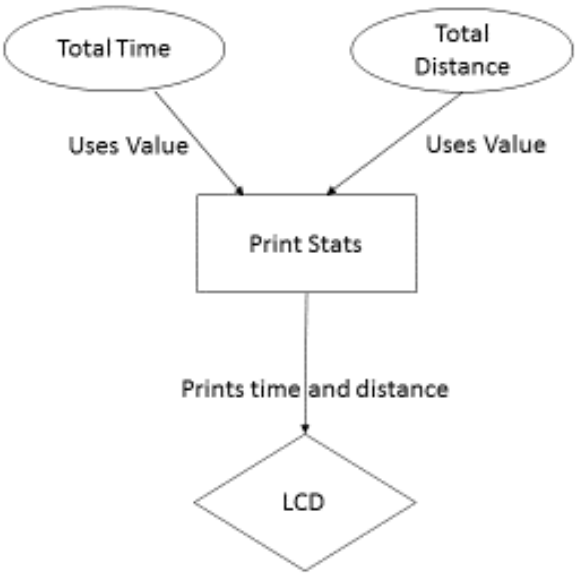
Timer 0 Interrupt Function



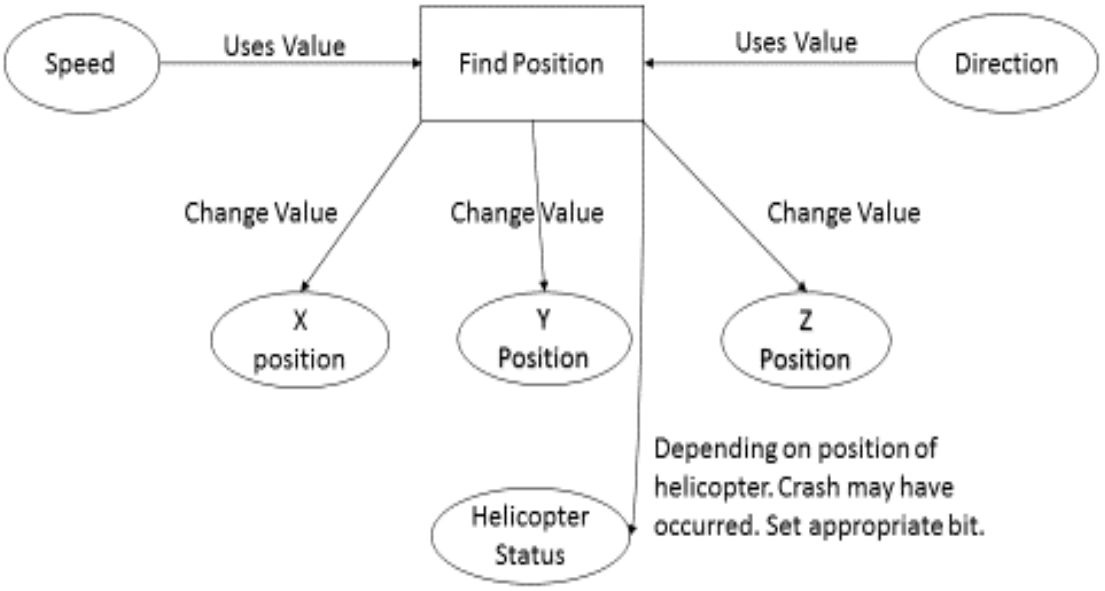
Add Total Time/Add Total Distance Functions



Print Stats Function



Find Position



Method and Module Algorithms

This section will give the high-level pseudo-code of the macros and functions used in the helicopter simulation assembler code.

calculatepositon

```
count = param1;
data = 0;
while(count < 10000) {
      count = count – 10000;
      data +=1;
}
data = data + “0”;
lcd_write(data);
data  = 0;
while(count < 1000) {
      count = count – 1000;
      data +=1;
}
data = data + “0”;
lcd_write(data);
```

changemotspeed

```
speed = param1;
Timer5.PWM = speed;
```

clearhalfsec

```
*x = 0;
```

lowstatus

```
ROWMASK = 15;
notLow = FALSE;
while(notLow == False) {
      temp1 = PORTC;
      if(temp1 == ROWMASK) {
            notLow = True;
      }
}
```

addposition

```
*XorYPosition = Null;
XorYPosition = &param1;
*XorYPosition = *XorYPosition + speed;
if(*XorYPosition >= 50000) {
      crashed = True;
      *XorYPosition = 50000;
```

```
}
```

addZposition

```
ZPosition = ZPosition + speed;
If(ZPosition >= 10000) {
      ZPosition = 10000;
      crashed = True;
}
```

subposition

```
*XorYPosition = Null;
XorYPosition = &param1;
*XorYPosition = *XorYPosition – speed;
if(*XorYPosition <= 0) {
      crashed = True;
      *XorYPosition = 0;
}
```

subZposition

```
ZPosition = ZPosition - speed;
If(ZPosition <= 0) {
      ZPosition = 10000;
      landed = True;
}
```

printdt

```
distanceORTime = *param1;
data = 0;
while(distanceOrTime < 1000) {
      distanceOrTime = distanceOrTime – 1000;
      data += 1;
}
data = data+’0’;
print_LCD(data);

data = 0;
while(distanceOrTime < 100) {
      distanceOrTime = distanceOrTime – 100;
      data += 1;
}
data = data+’0’;
print_LCD(data);

data = 0;
while(distanceOrTime < 10) {
      distanceOrTime = distanceOrTime – 10;
      data += 1;
}
print_LCD(data);

data = distanceOrTime;
data = data+’0’;
print_LCD(data);
```

IRQ0

```
if(crashed == True or landed == True or landing == True) {
      return;
}
If(speedchanges!= 0 or speed == 4){
      Return
}
Speedchanges = 1;
```

IRQ1

```
if(crashed == True or landed == True or landing == True) {
      return;
}
If(speedchanges!= 0 or speed == 1){
      Return;
}
Speedchanges = -1;
```

Timer 0_INT

```
If(speedchanges != 0) {
      *speedwait += 1;
      if(*speedwait == 500) {
            speedwait = 0;
            if(crashed == True) {
                  LED = (LED x -1);
                  changeLEDS(LED);
            } else {
                  Dospeedupdate();
            }
      } else {
            Speedwait += 1;
      }
}
addtotaltime();
if(hover == False or initialTakeOff == True or Landed == False) {
      findPosition();
      addtotalDistance();
}
update = 1
```

main

```
while(True) {
	if(landed == True){
		changemotspeed(0);
		timer0.status = off;
		printstats();
		exit();
	}
	if(crashed == True) {
		changemotspeed(0);
		speedchanges += 1;
		update_LCD();
		exit();
	}
	Keypadcheck();
	if(update == 1){
		update_LCD();
	}
}
```

updateLCD

```
calculate_position(xposition);
data = “,”;
lcd_write(data);

calculate_position(yposition);
data = “,”;
lcd_write(data);

calculate_position(zposition);
if(direction == “H”){
	lcd_write(“H”);
else {
	lcd_write(direction);
}
lcd_write(speed+”ms”);
update = 0
```

dospeedupdate

```
temp = speedchanges;
temp += speed;
if(temp == 5 or temp == 0) {
	return;
} else {
	changeMotSpeed(temp);
	speed = temp;
}
Speedchanges = 0
```

Keypadcheck

```
keydirection = [1,2,3,4,6,8];
Keypressed = USER_INPUT();
found == False;
for(x = 0; x<6 and found==True; x++){
      if(keypressed == keydirection[x]){
            if(crashed!=True and landed==False and hover == False
               and initialtakeoff == True) {
                  found = True;
                  direction = keypressed;
      }
}
If(found == False){
      If(keypressed = “#”) {
            takeoffOrLand();
      }
      if(keypressed = “*”){
            if(initialtakeoff == True) {
                  hover = True if hover == False else false;
            }
      }
}
```

Findposition

```
If(direction == “Right” or direction == “Forward”) {
      if(direction == “Right”) {
            addPosition(xposition);
      } else {
            addposition(yposition);
      }
} else if( direction == “Left” or direction == “Back”) {
      If(direction == “Left”) {
            subPosition(xposition);
      } else {
            subPosition(yposition);
      }
} else {
      If(direction == “Up) {
            addZPosition();
      else {
            subZposition();
      }
}
```

takeoffOrLand

```
if(hover == True or crashed == True or landed == True) {
      return;
}
If(takeOff == False) {
      direction = “Up”;
      speed = 2;
      changemotspeed(speed);
      takeOff = True;
      if(initialtakeoff == False) {
            initialtakeoff = True;
            LCD.cleardisplay();
            Write_line_1();
            update_LCD();
            timer0.start();
      }
} else {
      direction = “Down”;
      speed = 1;
      changemotspeed(speed);
}
```

write_line_one

```
LCD.write(“POS  ”);
LCD.write(“DIR  ”);
LCD.write(“SPD”);
```

write_start

```
LCD.write(“START:”);
```

addtotaltime

```
thousandCount += 1;
if(thousandCount == 1000) {
      thousandCount == 0;
      *totalTime += 1;
}
```

addtotaldistance

```
*distancemm += speed;
if(*distancemm >= 1000) {
      *distancemm == 0;
      addmeter();
}
```

addmeter

```
*distanceInMeters += 1;
```

printstats

```
LCD.write(“DISTANCE:”);
print_total_distance();
LCD.write(“DURATION:”);
print_total_time();
```

print_total_time

```
printDT(totalTime);
```

print_total_distance

```
printDT(distanceInMeters);
```