Part 3

# Overview

This section will focus a lot more on reading, with some doing sprinkled in. We will learn these topics and more: SD card data and play WAV files, propeller C functions, multicore approaches, what's a multicore?, library studies, counter modules applied and execution of code, examine the C interface with parallax assembly language. Given the nature of this section, most of the discussion topics will be summaries, insights, or both from what I was instructed to read.

# Discussion

## Steps and solution

**SD Card Data and Play WAV Files –**

**--------------------------------------------**

**Propeller C Functions –**

(Give output to function call program)

Can explain how IP pointer is manipulated

(Give output to function with parameter code)

(Define void counter function)

(Do modifications to adder)

(Do subtracter)

(Do global adder)

(Do global subtracter)

Understand the last multicore example including cogstart.

**Multicore Approaches –**

Cog\_run requirements: function cannot have parameters, retrun a value, should have all instructions in a while(1) loop. Should not use print, scan, or other calls using simpleIDE terminal. Must do so using specific functions meant to print from a cog.

(expand on cog example)

(expand once again output below)



Slightly confusing is their alternating use of cog\_run with and without dereferencing operator.

(expand on cog example)

(“Your turn” expansion)

Cores sharing data is a duplicate of the whats a multicore section. Skipped.

Terminal sessions can only one within one core or cog at a time. Provided example output below



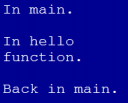
(Expand)

(“Your turn” expansion)

**What's a Multicore? –**

We already used multicore processing through the servo library.

Multitasking is diving a single core to multiple processes while multicore can achieve a few things in parallel.



(snip of multiple hellos)

At this point it feels like multicore approaches are simply from function calls.



(do the subtract function)

Ah ok, now they explain, it is not just a function call, you instruct the program to run the function call in the next available cog via int\* pointer name cog\_run( dereferenced memory location of function, stack space in int variable width). Then stop it with cog\_end(pointer name)

(Run the cog run example)

Attempt to declare function ahead of main to not use forward declarations.

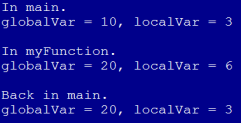
(Expand on the the cog run example to make p27 blink at diff rate)

(Expand once more to have main count while the other two blink)

Continue with page 138…

Explained variable scope, intro to global variables.

Local vs global code output below



(Expand on local vs global code)

Volatile keyword needed if to be shared between cogs. Example Volatile int sharedInt.

(Expand on pg 143 code to add a counter to blink, whicj updates reps to number of time LED blinks remember to change number of int spaces needed for cog call)

**Library Studies –**

**-------------------------------------**

**Counter modules applied and execution of code –**

This part is a bit nutty, 2 examples in this forum post to do.

**Examine the C interface with Parallax Assembly Language –**

(These files in GDrive)