PARALLAX SEMICONDUCTOR
www.parallaxsemiconductor.com
sales@parallaxsemiconductor.com
support@parallaxsemiconductor.com
phone: 916-632-4664 • fax:916-624-8003

Application Note AN019

Measuring Stack Space for Parallel Processes

Abstract: When programming the multicore Propeller P8X32A in Spin, each Spin-based parallel process that is launched needs its own stack space allocated in main memory. To develop optimal Spin-based multiprocessing objects, allocate plenty of stack space during development, then, as a last step, decrease it to an optimal size by measuring actual call stack requirements using the Stack Space.spin library object.

Introduction

A "stack," or stack space, is a block of memory used to store temporary data needed for proper program execution. The Propeller automatically handles the stack for the application cog, but launching additional cogs may require extra stack space allocated by the developer.

The following information demonstrates the need for stack space and describes the use of the Stack Length object to calculate the amount needed for a given process in a cog. The Stack Length object is included in the Propeller Tool Library[1].

The Need for Stack Space

Does the Propeller P8X32A use a call stack? The answer is yes and no, depending on the programming language selected.

Spin Uses a Call Stack

The Spin Interpreter implements a call stack to facilitate Spin method calling, parameter passing, and expression evaluation. The Propeller Application's stack is located in Main RAM immediately following the application's global variable memory. It expands and collapses as needed; growing towards higher addresses and shrinking towards lower addresses. Spin methods that are manually launched into other cogs store their stack starting at the *StackPointer* address given by the `COGNEW` or `COGINIT` command that launched them. Their stacks expand and contract in the same manner as with the Propeller Application stack. In both cases, the capacity of the stack (method nesting-depth, parameter list length, and expression complexity) is limited only by the amount of free memory available (for the application) or memory provided (by the developer).

Propeller Assembly Does Not Use a Call Stack

Propeller Assembly language does not implement a call stack since doing so would unnecessarily consume valuable memory and impose undue limitations on applications. Instead, Propeller Assembly provides a different mechanism through the `JMPRET` instruction (and also `CALL`, `JMP`, and `RET`) to maintain nested call history. This method requires more developer influence but allows the memory to be used more efficiently (optimizing for the specific application) and has a distinct advantage allowing the implementation of simple task-switching code in real-time systems using `JMPRET`.

Launching a Cog Requires Stack Space

Providing memory for stack space is necessary to execute a COGNEW or COGINIT command that launches Spin code (as opposed to Propeller Assembly code). The act of doing this is not difficult (see COGNEW and COGINIT documentation for examples), but determining the amount of space needed is a bit challenging.

Sizing the Stack

While developing an object that allocates a stack, it is good practice to make the stack larger than necessary, then optimize it after all development is done. This practice helps avoid strange behaviors that can easily be misinterpreted as a bug in the code when, in fact, the process simply has too small a stack for its purposes.

A very safe "initial" size is 128 longs of stack space for each cog launched. When you have made all your final code tweaks and are completely finished with your object, only then is it time to optimize the stack space. Often it can be optimized to as low as 9 or 10 longs.

Why bother optimizing stack space at all? Doing so will save valuable memory and perhaps development time in the long run. An un-optimized object may be fine for its initial intended use, but without optimization it may not be ready for future applications where memory might be scarcer.

Performing the Stack Length Test

Once development is complete and it's time to optimize stack space, use the Stack Length object to determine the appropriate size.

The Stack Length object makes testing easy:

- Just insert a small portion of test code into the object,
- Adjust the test code to watch the desired stack and call the object's starting method,
- Then exercise the object and get the resulting stack usage value.
- The Stack Length test code transmits the result (the actual stack space required) as a serial string on the transmit pin (typically P30).

It is convenient to view the serial string with the Parallax Serial Terminal included with the Propeller Tool.

Let's take a look at an example. The excerpt below is from Stack Length Demo.spin, which is included in the Propeller Tool L ibrary Demos.

The top section is temporary code to be inserted above the object code to be tested. In this case, the code to be tested begins at the VAR block.

There are three places in the `TestStack` method that must be adjusted to work with the object being tested and the serial display option being used:

1. `Stk.Init(@Stack, 32)`. Here, `(@Stack, 32)` was modified to match the name and size of the stack; `long Stack[32]` in the `VAR` block of the object being tested.
2. `Start(16, 500, 0)` is the code which calls test object's "starting" method and passes valid parameters.
3. `Stk.GetLength(30, 115200)` outputs the test results on P30 at 115,200 baud.

```
{••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••}
{••••••••••••••••••••••••  Temporary Code to Test Stack Usage  ••••••••••••••••••••••••••}
{••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••}

CON
  _clkmode      = xtal1 + pll16x         'Use crystal * 16 for fast serial
  _xinfreq      = 5_000_000              'External 5 MHz crystal on XI & XO

OBJ
  Stk   :       "Stack Length"           'Include Stack Length Object

PUB TestStack
  Stk.Init(@Stack, 32)                  'Initialize reserved Stack space (reserved below)
  Start(16, 500, 0)                    'Exercise code/object under test
  waitcnt(clkfreq * 2 + cnt)             'Wait ample time for max stack usage
  Stk.GetLength(30, 115200)              'Transmit results serially out P30 at 115,200 baud



{••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••}
{•••••••••••••••••••••••  Code/Object Being Tested for Stack Usage  •••••••••••••••••••••••}
{••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••}

VAR
  long  Stack[32]                                       'Stack space for new cog

PUB Start(Pin, DelayMS, Count)
{{Start new toggling process in a new cog.}}

  cognew(Toggle(Pin, DelayMS, Count), @Stack)

PUB Toggle(Pin, DelayMS, Count)
{{Toggle Pin, Count times with DelayMS milliseconds in between.
  If Count = 0, toggle Pin forever.}}

  dira[Pin]~~                                   'Set I/O Pin to output direction
  repeat                                        'Repeat the following
    !outa[Pin]                                  '  Toggle I/O Pin
    waitcnt(clkfreq / 1000 * DelayMS + cnt)     '  Wait for DelayMS milliseconds
  while Count := --Count #> -1                  'While Count-1 is not 0 (limit minimum to -1)
```

Steps for Using the Stack Length Demo

Here are the steps to follow for this demo with a Propeller Demo Board, PE Kit Platform, or any Propeller development system using a 5 MHz external crystal.

1. Select File → Open From → Library Demos and open the Stack Length Demo object.
2. Run the Parallax Serial Terminal software from the desktop icon, or from the Start → All Programs → Parallax Inc → Propeller Tool menu.
3. In the Parallax Serial Terminal, set the COM port to the Propeller chip's programming port (to find the COM port number, go back to the Propeller Tool and choose Run → Identify (F7).
4. In the Parallax Serial Terminal, set the Baud Rate field to 115200.
5. Position the Propeller Tool and Parallax Serial Terminal windows so they are not overlapping.
6. In the Propeller Tool, download the application to the Propeller; Run → Compile Current → Load EEPROM (F11).
7. Immediately click the blinking Enable button in the Parallax Serial Terminal.

A countdown message may appear until the Propeller Tool releases the serial port. Then, the stack usage message is displayed with the test results.

Tips for Testing Objects with the Temporary Code

Copy the temporary code out of the Stack Length Demo for testing your own objects. Here are some procedural tips:

- If the object provides stacks for more than one launched cog (running Spin code), test each stack individually using the test code and the following tips, below.
- Update the `Stk.Init` method call with the stack space name and size used in the object's `VAR` block.
- Update the call to `Start` with valid parameters for the object, or replace the whole call as needed if the object doesn't use a method named `Start`.
- Update the `Stk.GetLength` call as needed for the application's serial transmit pin (tx) and serial terminal baud rate—or adjust your serial terminal to match.
- The baud rate used must be supported by the frequency at which you are running your Propeller application. In this example, the system clock was set in the temporary code's `CON` section (5 MHz external crystal with 16x PLL, for 80 MHz operation). If your development board and application are using a different system clock configuration, change it here. Remember that serial communication requires an external crystal for timing accuracy.
- Make sure to fully exercise the object under test. In the above example, this is done simply by waiting long enough with the `waitcnt(clkfreq * 2 + cnt)` statement. In your application it may be required that you provide specific inputs via user controls or other devices attached. Failure to fully exercise the object before the test code calls `Stk.GetLength` may result in erroneous results as certain method and expressions will utilize the stack space more than others.
- After performing the test, reduce the stack space allotment in the object to the indicated amount in order to optimize memory usage.

References

1. The Stack Length and Stack Length Demo objects are included in the library files of the Propeller Tool software: www.parallaxsemiconductor.com/software

Revision History

Version 1.0: original document.