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

Application Note AN011

Simple Multicore Template for the Propeller P8X32A

Abstract: Use the simple_multicore_template.spin object as a starting framework for developing multicore Propeller applications that also comply with Parallax Semiconductor's Gold Object Standard. Several demonstration programs illustrate different aspects of parallel processing on the Propeller P8X32A microcontroller.

Introduction

Sometimes knowing where to begin programming when getting started with a new platform can be daunting. This template gives the reader an initial framework to take some of the "guess work" out of getting started with the Propeller chip and Spin language—just start filling in your code. The document is written from the perspective of the template being the top object file. However, depending on how code is filled in inside the template, a Parallax Gold Standard[1] compliant object can easily be developed from it.

The Template: simple_multicore_template.spin

```
{{
Object file:    simple_multicore_template.spin
Version:        0.1
Date:
Author:         [your name here]
Company:
Email:

Description:
This is a template for a simple multicore Propeller application.  Fill it
in to define the operation of your multicore application.

=============================================
        Connection Diagram
=============================================

N/A

Components:
N/A

=============================================
}}
CON

  'Set up the clock mode
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  '5 MHz clock * 16x PLL = 80 MHz system clock speed

VAR

  'Globally accessible variables
  long  cogStack[100]
  long  routine1_cog, routine2_cog
```

```
OBJ
  'Any object declarations go here

PUB Main
{{
  First public method in the top .spin file starts execution, runs in cog 0

  parameters:    none
  return:        none

  example usage: N/A - executes on startup

  Starts two cogs running in parallel with separate routines, then
  repeats forever.
}}

  'your code here

  'Start parallel routines
  routine1_cog := cognew(Parallel_Routine_1(6, 365, -1), @cogStack[0]) + 1
  routine2_cog := cognew(Parallel_Routine_2, @cogStack[50]) + 1


  'main loop - repeats forever
  repeat

PUB Parallel_Routine_1(param_var1, param_var2, param_var_etc) | local_var1, local_var2, local_var_etc
{{
  Does nothing yet. Add Spin statements to define the operation of this
  public method.  Meant to run independently in its own cog.

  parameters:    param_var1    = first parameter passed into Parallel_Routine_1
                 param_var2    = second parameter passed into Parallel_Routine_1
                 param_var_etc = third parameter passed into Parallel_Routine_1

  return:        none

  example usage: cognew(Parallel_Routine_1(42, 89, 314), @cogStack[0]) + 1

  expected outcome of example usage call: Does nothing yet.
}}

  'Your code here


PUB Parallel_Routine_2 | local_var1, local_var2, local_var_etc
{{
  Does nothing yet. Add Spin statements to define the operation of this
  public method.  Meant to run independently in its own cog.

  parameters:    none
  return:        none

  example usage: cognew(Parallel_Routine_2, @cogStack[50]) + 1

  expected outcome of example usage call: Does nothing yet.
}}

  'Your code here

PRI Private_Method_1 : return_result | local_var1, local_var2, local_var_etc
{{
  Does nothing yet. Add Spin statements to define the operation of this
  private method.

  parameters:    none
  return:        0 - alias is "return_result"

  example usage: Private_Method_1

  expected outcome of example usage call: Does nothing yet.
```

```
}}

  'Your code here

PUB Public_Method_1 : return_result | local_var1, local_var2, local_var_etc
{{
  Does nothing yet. Add Spin statements to define the operation of this
  public method.

  parameters:    none
  return:        0 - alias is "return_result"

  example usage: Public_Method_1

  expected outcome of example usage call: Does nothing yet.
}}

  'Your code here
```

The Constant Block

Typically, the optional CON constant declaration block appears as the first block within an object, but it can be placed anywhere. It can contain compiler directives and constant declarations for use within the object. Constant assignments use a single "=" operator instead of the ":=" variable assignment operator.

Global Variables

Define global variables in the VAR section of the object. These variables are globally accessible to all Spin methods in the object, including methods that might be executing in other cogs. Be aware that another cog may change global variable's value if multiple cogs share the same memory location, resulting in a possible memory collision. Even though the Propeller's hub guarantees a cog mutually exclusive access to a variable residing in main memory, nested operations (like post increment) are not always apparent and take multiple accesses to complete.

Spin provides a set of lock commands to prevent memory collisions; see the Propeller Manual for more information [2].

Stack Space Variables

When launching a Spin method into a new cog with the COGNEW or COGINIT commands, the Spin interpreter needs dedicated RAM space allocated for a call stack. For relatively small Spin routines running independently in a cog, 50 longs of allocated space is plenty. Since the template launches two additional cogs, an array of 100 longs has been reserved. For more information about stack space requirements, see AN019: Stack Space [3].

Object Declarations

Included object declarations go in the OBJ section. The template does not pre-include any objects, but the other demonstration programs in the .zip file declare the Parallax Serial Terminal.spin object[4] for serial communication with a serial terminal. Note that any method within the object can make use of an included child object's public (PUB) methods, even if the calling methods are executing in a different cog. Be aware that possible multi-threaded hazards can exist when making calls to a child object's methods from multiple

cogs. Executing the simple_multicore_template_demo1.spin application on the Propeller demonstrates how multiple calls to the Parallax Serial Terminal from parallel routines can cause unexpected results.

First Public Method in the Top Object File

The first public Spin method in the top object file serves as the entry point for execution on the Propeller. The Spin Interpreter, which always starts in cog 0, executes the Spin instructions. Since these Spin instructions are the first things executed, at least one public method must exist in every object.

Subsequent Public Methods and Private Methods

All methods within an object can be launched into a cog or called from any other method in that object. Private (PRI) methods can only be called from within the object that contains them. Declaring a method private prevents inappropriate calls from outside the object, where such a call could negatively affect the integrity of the object itself.

Method Parameters

The Parallel_Routine_1 method in the template demonstrates a method that requires three parameters be passed to it. Parameter values passed into a method are locally scoped and can be referenced within the method by the parameter's symbol defined symbol name. Method parameters are typed as a LONG by default and are capable of holding signed 32-bit values.

Local Variables

A local variable is valid only within the method that declares it. Any data stored in a local variable is valid only while that method is being executed. Declare local variables on a method's header line by inserting "| *LocalVar*, …" where *LocalVar* is the desired name of each local variable declared. In the code template above, each method has a few local variables defined, except for the Main method.

Return Values and the Result Alias

All methods return a value upon completion, even if an explicit return command is not issued. The value returned is stored in a special local RESULT variable within each method. By default, a method returns with zero unless the RESULT variable is changed or a RETURN *Value* command is issued. To aid in program readability, the RESULT variable can be aliased to a variable name of your choosing. This is done on the method's header line by inserting "**:***RValue*" where *RValue* is the desired symbol, after the method's parameters but before local variable definitions. Public_Method_1's result value in the template is aliased to **return_result**, which can have assignments made to it just as any other local variable. Remember, Spin commands are not methods; some commands like LONGMOVE do not return a value and cannot be assigned to a variable.

Symbol Name Scope

Symbols defined in an object have a scope limited to that object, but extends to the object's methods running in other cogs. Access to some symbols (public methods and constants) in included objects is only possible through the use of the *object*.*method* or *object*#*constant* syntax. See the Propeller Manual v1.1 for more information.[2]

Resources

The following files are available for download from the zip file archive at this application note's web page: www.parallaxsemiconductor.com/an011.

simple_multicore_template.spin
simple_multicore_demo1.spin
simple_multicore_demo2.spin

References

1. Gold Standard objects: www.parallaxsemiconductor.com/goldstandard
2. Propeller Manual version 1.1: www.parallaxsemiconductor.com/docs
3. AN019: Stack Space: www.parallaxsemiconductor.com/an019
4. Parallax Serial Terminal.spin is a Propeller Tool library object; download the Propeller Tool from www.parallaxsemiconductor.com/software.

Revision History

Version 1.0: Original document