```
*******************************************************************************
*                                                                             *
*       Spin Architecture                                                     *
*                                                                             *
*******************************************************************************
```

This document describes the abstracted Propeller Chip environment in which Spin programming is used.

This is a preliminary and rough draft based upon an initial and liited evaluation of the architecture used and is incomplete and may be incorrect in places. It does however give a introduction to how things look like they work even if the specifics are not entirely correct.

```
*******************************************************************************
*                                                                             *
*       Objects                                                               *
*                                                                             *
*******************************************************************************
```

A Spin program consists of an object (and potentially sub-objects), even what would traditionally be thought of as a main program in other programming environments is really an object because it can be embedded within another main program as a sub-object in its own right.

Every object has the following format. A list of links, the first of which points to after the current object (long-aligned) and a set of links which point to each of the objects's PUB and PRI methods. PUB methods are always listed first, PRI methods second.

The PUB and PRI methods appear in the image in which they are within the source code. The content of DAT sections (assembly language code and so on) are inserted after the list of links and before the first method.

An object -

```
                        .----------.
        .---------------|__________|    +0      Pointer to next object
        |           .---|__________|    +1      Pointer to first PUB method
        |   .-------|---|__________|    +2      Pointer to second PUB method
        |   |   .---|---|__________|    +3      Pointer to first PRI method
        |   |   |   |   | DAT      |            Data Area
        |   |   |   |   |          |
        |   |   |   |   |----------|
        |   |   |   `-->| PUB a    |            First Method
        |   |   |       |          |
        |   |   |       |----------|
        |   |   `------>| PRI b    |            Second Method
        |   |           |          |
        |   |           |----------|
        |   `---------->| PUB c    |            Third method
        |               |          |
        |               `----------'
        `-------------->
```

The links are positive offsets from the base of the object to where the method appears.

A CALL opcode uses a numbered offset to indicate which method to call, so a call to the 'b' routine in this object would be coded as "CALL +3", which means call the routine whose start location is pointed to by the "+3" list entry. The first link (the pointer to the next object) is entry "+0", the second is "+1" and so on.

The links are all longs and long aligned. The link is actually formed from two words, the first being the offset (LSB first, LSB second) to the method it relates to.

```
*******************************************************************************
*                                                                             *
*       Sub-Objects                                                            *
*                                                                              *
********************************************************************************
```

Any object can include sub-objects. A sub-object is added after the end of the object which uses it and a link is added within the parent object which points to the start of the object. The object links are added after the links to PUB and PRI methods.

An object -

```
                        .----------.
        .---------------|__________|    +0      Pointer to next object
        |            .--|__________|    +1      Pointer to first PUB method
        |        .---|--|__________|    +2      Pointer to second PUB method
        |        | .-|--|__________|    +3      Pointer to first PRI method
        |        | | |  | DAT      |            Data Area
        |        | | |  |----------|
        |        | | `->| PUB a    |            First Method
        |        | `--->| PRI b    |            Second Method
        |        `----->| PUB c    |            Third method
        |               `----------'
        `--------------->
```

An object with sub-object -

```
                        .----------.
        .---------------|__________|    +0      Pointer to next object
        |            .--|__________|    +1      Pointer to first PUB method
        |        .---|--|__________|    +2      Pointer to second PUB method
        |        | .-|--|__________|    +3      Pointer to first PRI method
        |    .---|-|-|--|__________|    +4      Pointer to sub-object
        |    |   | | |  | DAT      |            Data Area
        |    |   | | |  |----------|
        |    |   | | `->| PUB a    |            First Method
        |    |   | `--->| PRI b    |            Second Method
        |    |   `----->| PUB c    |            Third method
        |    |          `----------'
        |    |
        |    |          .----------.
        `->  `-> .------|__________|    +0      Pointer to next object
                 |   .--|__________|    +1      Pointer to first method
                 |   |  |__________|
                 | .-|--|__________|    +3      Pointer to last method
                 | | |  | DAT      |            Sub-Object's Data Area
                 | | |  |----------|
                 | | `->| PUB      |
                 | |    :          :            Sub-Object's Methods
                 | `--->| PRI      |
                 |      `----------'
                 `----->
```

Note that the pointer to next object links form a chain through all objects.

CALL opcodes within the main program / top object and within the sub-object work as they do fro a single object. The PUB or PRI method to execute is found through the link which is at the start of the object the link is in.

To call a sub-object, the CALLOBJ opcode is used. This works in a similar way as CALL does but knows the link it references is not directly to a method but a pointer to another object. A second reference indexes the link within the object that is pointed to. In this way any method of any sub-object can be called.

When CALLOBJ executes, it updates the global 'base of object pointer' (OBJ) to point to the start of the object which will be executed. The previous object pointer having been pushed to the stack to be popped when the called objet's method returns. In this way, any PUSH, POP and other opcodes which deal with variables referenced from OBJ will relate to the object which is executing.

```
********************************************************************************
*                                                                              *
*       Multiple Sub-Objects                                                   *
*                                                                              *
********************************************************************************
```

Every object that is added has another link added which points to its base and the new object is added after the last object that was added.

If the object added is exactly the same as an already added object, because all objects are position independant and re-entrant, only one copy of the object is added although a new link to the object is added. Links are added in the order objects are defined for inclusion.

```
********************************************************************************
*                                                                              *
*       Arrays of Sub-Objects                                                  *
*                                                                              *
********************************************************************************
```

Arrays of sub-objects are no different to adding the same object a number of times. The first object will be added as previously described. Every dditional object will cause a link to the same object to be added which will point to the object already defined.

To facilitate accessing the correctly required object, an indexed version of CALLOBJ is available ("CALLOBJ []"). This allows the index of the object being referenced to be pushed to the stack before executiong "CALLOBJ []", and the opcode will pop the top of stack and add it to the offset index into the link list when determining which object to get.

```
********************************************************************************
*                                                                              *
*       Wrapping up a main program                                             *
*                                                                              *
********************************************************************************
```

Although it was earlier said that a main program was no more than an object in its own right, when placed in an image file to be executed as a single application an application description block is prefixed to its front.

This description block describes the operating frequency and system clock source to use after the image is loaded, how the initial object base, variable base, program pointer and stack pointer should be loaded. It also includes a checksum of the entire image.

The checksum is the sum of all bytes in the image with byte overflow ignored with the sum finally negated. Verifying the correctness of an image is a simple matter of summing all bytes in the image with byte overflow ignored and checking that the checksum is zero.

Note that the description block is a mix of byte, word and long entries.

```
                           .----------.
                           |__________|         Long : Frequency (Hz)
                           |__|                 Byte : XTAL mode
                           |__|__               Byte : Checksum
                .----------|_____|              Word : Base of program
            .---|----------|_____|              Word : Base of variables
          .-|---|----------|_____|              Word : Base of stack
          | | .-|----------|_____|              Wopd : Initial program counter
        .-|-|-|-|----------|     |              Word : Initial stack pointer
        | | | | |          `-----'
        | | | | |
        | | | | |          .----------.
        | | | | `-> .------|__________|    +0   Pointer to next object
        | | | |     |   .--|__________|    +1   Pointer to first method
        | | | |     |   |  |__________|
        | | | |     | .-|--|          |    +N   Pointer to last method
        | | | |     | | |  |----------|
        | | | |     | | |  | DAT      |         Data Area
        | | | |     | | |  |----------|
        | | | `---> | | `->| PUB      |
        | | |       | |    :          :         Methods
        | | |       | `--->| PRI      |
        | | |       |      `----------'
        | | |       |      .----------.
        | | `-----> `----->| vars     |        Main Program Variables
        | |               |          |
        | |               `----------'
        | |               .----------.
        | `--------------->| stack    |         Stack Space
        `----------------->|          |
                           `----------'
```

Main program variables are placed after the object and the stack space after that.

Note that the main program link to next object when there is no sub-object points to the main program's vraiable base. The last object of any sub-objects included also points to the main program's variable base.

```
********************************************************************************
*                                                                              *
*       Stack                                                                  *
*                                                                              *
********************************************************************************
```

... more ...

```
********************************************************************************
*                                                                              *
*       Main Program Variables                                                 *
*                                                                              *
********************************************************************************
```

... more ...

```
********************************************************************************
*                                                                              *
*       Variables in Sub-Objects                                               *
*                                                                              *
********************************************************************************
```

** THIS SECTION MAY NOT BE ENTIRELY CORRECT

Variable allocation for objects is a little complex. While the amount of space required for variable storage for the main program and any sub-objects is known at compile time, at run time sub-objects need to be able to set the base of that area so thay can access their varaibles referenced to that base (VAR).

Every object will have its own variable space pre-defined in memory at compile time, and multiple objects or arrays of object each require their own unique and separate space regardless of whether or not they share the same executable bytecode.

A sub-object's variable space is placed after the main program's variable space and in order to adjust the objects variable base when the object is called, the second half of the word in the link which vectors the CALLOBJ opcode stores a number to increment the variable base by when the object is called.

This is why it is necessary to have a separate link for all objects used even when objects utilise the exact same bytecode; calls to each individual link will adjust the variable base as required for that specific object.