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

Application Note #AN009

Code Execution Time on the P8X32A

Abstract: Determine the execution time of Spin or Propeller Assembly code on the P8X32A by using the system counter, by toggling an I/O pin, or by counting assembly instruction cycles. The techniques presented are vital for discovering or verifying routine timing and tuning code to an application's needs.

Introduction

While developing programming skills on the Propeller, it is not uncommon to become concerned about execution time of certain routines. Knowledge of the Propeller and programming in general may lead to experimentation with variations in code just to realize speed increases, but it can be a guessing game without solid feedback of actual execution time.

The techniques for calculating and measuring execution time introduced here can be applied in many ways, from timing just a single line of code to timing entire sophisticated routines.

Determining the execution time for a portion of Propeller Assembly code is quite easy, but more elusive for Spin code. There are three techniques for timing code with definitive results, each with its "ideal" application.

1. Counting and adding up determinant instruction times (Propeller Assembly only)
2. Using the System Counter (Spin and Propeller Assembly)
3. Toggling a pin and measuring the pulse-width (Spin or Propeller Assembly)

Technique 1: Counting and Adding Up Determinant Instruction Times

This technique applies only to Propeller Assembly. Since Propeller Assembly instructions are well defined and most consume a fixed number of clock cycles, it is possible to simply add up "clock cycles" while looking through the code. This is the preferred technique for assembly programs, especially in applications where precise timing is critical. Here's a small snippet from an example assembly routine that is serially transmitting a data byte:

```
                or      Data, #$100             '4
                shl     Data, #1                '4
                mov     BitCount, #10           '4
TxLoop          shr     Data, #1        wc      '4
                muxc    outa, Pin               '4
                djnz    BitCount, #TxLoop       '4,8 (12:120+4)
```

To determine the execution time of this snippet of code like this, look up the instruction times in the Propeller Quick Reference[1] or the Propeller Manual[2]. In the example above, the instruction times appear in the code's comments. Notice that most Propeller Assembly instructions take 4 clock cycles. However, the DJNZ instruction takes 4 cycles if it jumps and 8 cycles if it doesn't; since it will often jump and rarely not jump, the comment noted that as "4, 8."

So, it's easy to see that the first three instructions will take 4 + 4 + 4 = 12 clock cycles to execute. The last three instructions are a bit more complicated since they are in a loop, the TxLoop. Most of the iterations of TxLoop will take 12 cycles (4 + 4 + 4) to execute, and the last iteration will take 4 additional cycles for the non-jump. Since it will loop 10 times (as determined by BitCount and the DJNZ instruction), the total execution of the loop will be 10 × 12 + 4 = 124 cycles. We noted the single-iteration and full-loop times as "(12:120+4)" at the end of the loop.

These cycle counts give a straightforward way to determine actual execution time with respect to the clock speed in use. If this code was running with a system clock speed of 80 MHz, then each bit of data transmitted (via the MUXC instruction in the TxLoop) will be 12 cycles ÷ 80 MHz = 150 ns in width since it takes 12 cycles to get from one execution of MUXC to another. With overhead considered, the entire routine as shown will take 12 cycles (first 3 instructions) + 10 × 12 cycles (loop) + 4 cycles (leaving loop) = 136 cycles total, or 1.7 µs (136 cycles ÷ 80 MHz).

This is verifiable with an oscilloscope.

Spin Command Execution Times are Not Deterministic

Propeller Assembly instructions are very deterministic in their timing since they are the lowest level of execution within the cogs. Since Spin commands are constructed from potentially dozens of Propeller Assembly instructions with many possible execution paths, they are inherently difficult to document as a fixed amount of execution time. Even a single, specific Spin command can take a different amount of time to execute depending on the how its parameters are defined and with the varying complexity of constants used. For instance, constant values are compressed by the Spin compiler into a roughly proportional number of bytes, which affects the command's resulting execution time. For this reason, it is important to use the next techniques for timing Spin code.

Technique 2: Using the System Counter

This technique applies to both Spin and Propeller Assembly. The System Counter in the P8X32A is an immensely helpful tool for timing. Use it to time portions of code by noting the System Counter value just before and just after the target code and calculating the difference.

Developing Timing Code in Spin

It's always best to remember the technique of doing this kind of timing, rather than trying to remember the actual code that performs it. If the technique is well understood, it takes just a few moments to build the code when needed. Always perform a quick "test" to determine its validity in that particular situation.

Be aware that each "read" of the System Counter takes time also, so the test result will naturally be off by some amount. Before relying on this timing method, first determine the amount of overhead it introduces.

To calculate overhead, the concept is to take two System Counter readings (Time1 and Time2), one immediately after the other, and subtract the two readings to determine how many clock cycles it took just to record the two values. Then use a library object like "Simple_Serial" or "Parallax Serial Terminal" to transmit the result as a decimal number to the computer screen.

Note that the difference between **Time1** and **Time2** can be calculated as **Time2** - **Time1**, but by applying a little algebra it is clear that -**Time1** + **Time2** is an equivalent expression. Using this knowledge and applying some powerful Spin operators gets it all done using just one **long** variable.

```
VAR
  long Time           'Holds elapsed clock cycles

PUB Timing
  Time := -cnt        'Read System Counter (Start)
  Time += cnt         'Read System Counter (End)
```

After the second "**Time**" line executes (above), **Time** will be equal to the number of clock cycles it took just to perform these two reads of the System Counter. As it turns out, it takes 544 clock cycles.

Keep in mind it's always important to test the overhead first, then account for it, rather than to always expect it to be 544. For example, the overhead will be different if **Time** were defined outside of the first eight longs of either the object's global variables or the **Timing** method's local variables (which includes parameters and the automatic **RESULT** variable).

So, now, after testing the overhead, modify the time checking code slightly to accommodate for it:

```
  Time := -cnt          'Read System Counter (Start)
  Time += cnt - 544     'Read System Counter (End)
```

Now, the clock cycles that elapse between the two statements will be calculated as 0, and adding any additional statements in between them will increase this value—a direct result of the time required by those additional statements. This is accurate as long as the clock cycles elapsed is less than 2^{31}.

Developing Timing Code in Propeller Assembly

A similar technique works in Propeller Assembly as it did in Spin. Start with the following:

```
AsmTiming         org 0
                  neg Time, cnt     'Read System Counter (Start)
                  add Time, cnt     'Read System Counter (End)

                  {more code here}

Time              res       1       'Holds elapsed clock cycles
```

The overhead is, not surprisingly, 4 clock cycles. Adjusting the time-checking code to accommodate for this leads to:

```
                  neg Time, cnt     'Read System Counter (Start)
                  add Time, cnt     'Read System Counter (End)
                  sub Time,  #4     'Adjust for overhead
```

Using it in Practice

Now, for timing Spin, precede the target code with:

```
Time := -cnt
```

...and follow the target code with:

```
Time += cnt - 544
```

..and then serially transmit the result with a communication object. If desired, grab a calculator and divide the result (clock cycles) by the system clock speed (cycles per second) to determine the actual time elapsed.

For timing Propeller Assembly code, follow a similar technique, except with:

```
neg Time, cnt
```

...preceding the target code, and following it with:

```
add Time, cnt
sub Time, #4
```

Technique 3: Toggling a Pin and Measuring the Pulse Width

This technique applies to both Spin and Propeller Assembly. Using an oscilloscope to monitor a toggling I/O pin is occasionally the handiest way to time Spin or Propeller Assembly code.

In Spin

In Spin, place something like this immediately before the target code:

```
dira[0]~~             'Set I/O pin 0 to output
!outa[0]              'Toggle pin 0
```

...and something like this immediately after the target code:

```
!outa[0]              'Toggle pin 0
```

In Propeller Assembly

In Propeller Assembly, place something like this immediately before the target code:

```
 or  dira, #%1        'Set I/O pin 0 to output
 xor outa, #%1        'Toggle pin 0
```

...and something like this immediately after the target code:

```
 xor outa, #%1        'Toggle pin 0
```

In both cases (Spin and Propeller Assembly) keep in mind that the toggling statement itself (**`!outa[0]`** or **`xor outa, #%1`**) takes some time to execute. Determine the overhead by first testing the pulse-width generated by two such toggling statements adjacent to each other; just subtract that overhead from all further readings.

References

1. The Propeller Quick Reference PDF is available from the Help menu of the Propeller Tool software; a free download from www.parallaxsemiconductor.com/software.
2. The Propeller Manual is available in print and free PDF; Parallax #122-32000; www.parallax.com. It is also available from the Help menu of the Propeller Tool software.

Revision History

Version 1.0: original document.