GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2017

Lab #10: Microcontroller Computational Benchmarking

Contents

1 Background Material

1.1 Data Types and Controller Computations

This lab project focuses on two tools of Code Composer Studio used by programmers to fully utilize the limited resources present on microcontrollers. To introduce this topic, first recall that mathematical algorithms are based on computations involving *real numbers*; such objects have a straightforward interpretation in mathematics, but they are not realizable inside a microcontroller. Prior to programming a microcontroller to execute a mathematical algorithm, one must first decide whether to approximate numerical values with *integer data types* or *floating-point data types*.

To provide some context, consider the computational requirements associated with approximating a state-space integral control algorithm on a microcontroller. As we have seen, such a control algorithm is generically described by equations involving real numbers of the form

$$\begin{aligned} u &= -K_1\hat{x} - K_2\sigma \\ \dot{\hat{x}} &= A\hat{x} + Bu - L\left(C\hat{x} - y\right) \\ \dot{\sigma} &= y - r \end{aligned}$$

where u is the computed signal (the actuator input), y is the measured signal (the sensor output), r is the reference command value, $\hat{x}$ and σ are the controller state variables, $\{K_1, K_2, L\}$ are the controller gain matrices and $\{A, B, C\}$ are the plant coefficient matrices. If the design model of the plant has n state variables, 1 actuator input and 1 sensor output, then using forward Euler numerical integration the state-space integral controller requires up to $n^2 + 4n + 2$ *scalar multiplications* and n^2+4n+2 *scalar additions* each time a corrective action is to be updated; the exact value will depend on the number of zero entries appearing in the various matrices involved. Note that the number of multiplications and additions grows quadratically with n, which may be interpreted as a measure of plant complexity; this agrees with intuition, since the ability to control a complex system should require more computations than to control a simple system. Any candidate microcontroller will have

an upper limit on system clock frequency and will complete multiplications and additions in a fixed number of system clock cycles; of particular practical interest is what impact the selection of data types might have on the number of system clock cycles required to complete a given computation (and this is determined by the underlying microcontroller hardware architecture).

Individual Terms	# of $\times$	# of $+$ or $-$
$-K_1\hat{x} - K_2\sigma$	$n+1$	n
$\hat{x} + (TA)\,\hat{x} + (TB)\,u - (TL)\,(C\hat{x} - y)$	$n^2 + 3n$	$n^2 + 3n$
$\sigma + T\,(y - r)$	1	2
Overall Totals	$n^2 + 4n + 2$	$n^2 + 4n + 2$

Some microcontrollers (e.g. the F28027) incorporate only integer processing units, providing hardware-based computational capability only for integer data types. Although these less-expensive microcontrollers are capable of performing computations on floating-point data types, such computations would require use of special subroutines from a device-specific software library; these subroutines would take up space in microcontroller memory and would also be relatively slow to execute. On the other hand, some more-expensive microcontrollers (e.g. the F28069) incorporate floating-point processing units in addition to integer processing units; such microcontrollers provide hardware-based computational capability for floating-point data types, thus eliminating the need for a software library approach when performing computations on floating-point data types.

Floating-point data types are perhaps the most natural choice for approximating real numbers in control algorithms; the benefits of using floating-point data types would be (i) the ability to represent all constants and signals in SI units so that the code most closely resembles the theoretical developments on which it is based, and (ii) the ability to disregard the magnitudes of all constants and signals anticipated during use without introducing potential numerical overflow concerns. However, it is possible to replace all floating-point data types with corresponding integer data types in control algorithms by introducing appropriate scaling factors. For example, if a particular voltage variable is expected to remain between $+24.0$ V and -24.0 V at all times, then the `int16` data type could be used to represent $+24.0$ V as the integer $+32767$ and -24.0 V as the integer -32768; this approach has the virtue of permitting efficient implementation on lower-cost microcontrollers, but the result is code that is more challenging to write and debug, and improper scaling factors could result in numerical overflow with catastrophic consequences.

The F28069 device costs $8.85 in quantity whereas the F28027 device costs only $3.05 in quantity; many features distinguish these two microcontrollers, not just the presence or absence of a floating-point processor, but still it is worth learning how these two microcontrollers compare from a computational viewpoint. Hence, this lab will employ comparative benchmarking experiments.

1.2 CCS Compiler Optimization

Chapter 3 of Optimizing C Compiler provides a detailed description of how to use compiler tools to optimize the executable code that the compiler generates from the C source code that you write, so as to improve execution speed or to reduce memory requirements. We have not explored these compiler tools this semester until now, and there are many different options and issues to consider, as described in the document mentioned above. For this lab, however, we will be content to consider just the two extremes from a range of possibilities; *no compiler optimization* versus *full compiler optimization* (for speed rather than for size).

The process for invoking compiler optimizations from within CCS is straightforward. Under project properties, go to `Build > C2000 Compiler > Optimization` and make selections.

1. For no use of compiler optimization
   (a) Set `Optimization level` to `off`
   (b) Set `Speed vs size trade-offs` to `0 (size)`
2. For full use of compiler optimization
   (a) Set `Optimization level` to `4 Whole Program Optimizations`
   (b) Set `Speed vs size trade-offs` to `5 (speed)`

1.3 CCS Profile Clock

When we implement controller calculations in a timer ISR, we implicitly assume that all required calculations will be completed within the interrupt period associated with that timer. As it is clear that an arbitrarily large number of calculations cannot be completed in an arbitrarily small interval of time, it is the programmer's responsibility to ensure that no timing problems arise when a specific controller is being implemented. CCS includes a *profile clock* feature that allows us to check how many system clock cycles are required to execute any line of code, so this tool can be used to accurately predict the time required to complete one cycle of controller calculations, thereby making sure that there is sufficient time available at the specified timer ISR frequency. In this lab we will execute pre-written computational benchmarking codes that feature numerous multiplications and additions, and we will use the profile clock to measure performance.

To use the profile clock, select `Run > Clock > Enable` during a debug session. A small yellow clock icon will appear along the bottom of the debug window. The number to the right of this icon is the number of elapsed system clock cycles. You can reset this system clock cycle counter to zero by double-clicking it. To measure your microcontroller's computational performance while executing a specific benchmark code, use this tool along with two strategically placed breakpoints in order to measure the elapsed time between the two breakpoints.[1]

2 Lab Assignment

2.1 Pre-Lab Preparation

Each individual student must work through the pre-lab activity and prepare a pre-lab deliverable to be submitted *by the beginning of the lab session*. The pre-lab deliverable consists of a brief typed statement, no longer than one page, in response to the following pre-lab activity specification:

1. What is the finite range of values that can be represented using the `float32` data type?
2. What is the finite range of values that can be represented using the `int32` data type?
3. In what way are the resolutions of the `float32` and `int32` data types fundamentally different?
4. Consider the supplied benchmarking codes.
   (a) Provide an overall description of what the codes actually do.
   (b) Use Matlab to determine the expected final value of `m3` for each code.

These questions are intended to guide your thinking and to provide context for the benchmarking experiments that you will conduct in the lab.

[1] If compiler optimization has been used, you may need to resume code execution several times to get from the first breakpoint to the second breakpoint. In such a situation, don't reset the profile clock at intermediate breaks.

2.2 Specification of the Assigned Tasks

2.2.1 Benchmarking the F28069

Two computational benchmarking codes have been written for the F28069 device, and they are to be run on the Peripheral Explorer kit; both codes have been posted to tsquare. The first code is used to benchmark computational performance using `int32` data types, whereas the second code is used to benchmark computational performance using `float32` data types. Place breakpoints at lines 25 and 66, and use the profile clock to determine the number of system clock cycles required to get from the first breakpoint to the second breakpoint. Conduct four separate benchmarking tests as follows: use `int32` data types with no compiler optimization; use `int32` data types with full compiler optimization; use `float32` data types with no compiler optimization; use `float32` data types with full compiler optimization. Record the required number of system clock cycles in the table provided, along with the corresponding time interval measured in seconds.

Instructor Verification (separate page)

2.2.2 Benchmarking the F28027

Two computational benchmarking codes have been written for the F28027 device, and they are to be run on the Launch Pad kit; both codes have been posted to tsquare. The first code is used to benchmark computational performance using `int32` data types, whereas the second code is used to benchmark computational performance using `float32` data types. Place breakpoints at lines 25 and 66, and use the profile clock to determine the number of system clock cycles required to get from the first breakpoint to the second breakpoint. Conduct four separate benchmarking tests as follows: use `int32` data types with no compiler optimization; use `int32` data types with full compiler optimization; use `float32` data types with no compiler optimization; use `float32` data types with full compiler optimization. Record the required number of system clock cycles in the table provided, along with the corresponding time interval measured in seconds.

Instructor Verification (separate page)

GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2017

Lab #10: Microcontroller Computational Benchmarking

Instructor Verification Page

Lab Section	Begin Date	End Date
L01, L02	November 28	November 28
L03, L04	November 30	November 30

To be eligible for full credit, do the following:

1. Submissions required by each student (one per student): Upload your pre-lab deliverable to tsquare before lab session begins on begin date.
2. Submissions required by each group (one per group): Submit a hard-copy of this verification page before lab session ends on end date.

Name #1: _____

Name #2: _____

F28069	No Optimization		Full Optimization	
	# Clock Cycles	# s at 90 MHz	# Clock Cycles	# s at 90 MHz
Integer Math				
Floating-Point Math				

Checkpoint: Verify completion of the task assigned in §2.2.1.

Verified: _____ **Date/Time:** _____

F28027	No Optimization		Full Optimization	
	# Clock Cycles	# s at 60 MHz	# Clock Cycles	# s at 60 MHz
Integer Math				
Floating-Point Math				

Checkpoint: Verify completion of the task assigned in §2.2.2.

Verified: _____ **Date/Time:** _____