

Week 11 Lab

CC2511

Task Description

You'll write a fast implementation of the vector dot product in assembly code. You will first implement the function in C, and then you will implement the same function in assembly code. The objective is to make the assembly code version as fast as possible.

The operation that you must implement is Eq. (1), in which there is repeated multiplication and addition:

$$\text{dot}(\mathbf{A}, \mathbf{B}) = \sum_i (\mathbf{A}_i \mathbf{B}_i), \quad (1)$$

where \mathbf{A} and \mathbf{B} are input arrays and the index i loops over all the items in these arrays.

Your task is to implement Equation (1) in fixed-point arithmetic¹. The input arrays are 32 bit unsigned integers in Q20 format, i.e. they have an implied denominator of 2^{20} and you must shift right by 20 bits after multiplication. To avoid overflows, your intermediate results must be 64 bits long. You can declare a 64 bit integer in C using

```
unsigned long long int accumulator = 0;
```

and in assembly by splitting the 64 bit result over two registers.

A starter project is provided on LearnJCU that includes an accuracy and speed test. Load this zip into Kinetis Design Studio using File -> Import -> General -> Existing projects into workspace -> Select archive file. Do not modify the `main.c` file. Write your implementations in the `c_dotproduct.c` and `asm_dotproduct.asm` files.

¹ Refer to the Week 9 lecture notes for details on fixed point arithmetic.

Motivation

Filters are a key part of many signal processing systems, for example, in processing audio, video, and sensor data such as vibrations measured by an accelerometer. A filter suppresses particular frequency components within a signal in order to isolate frequencies of interest. For example, a low-pass filter keeps the low frequencies that are below a specified cut-off, and attenuates high frequencies that are above that cut-off. In the case of audio data, this would eliminate high-pitched sounds while keeping the lower-pitched ones.

Many digital filter designs take the form of Eq. (1), where the vector \mathbf{A} is the filter coefficients that are fixed at design time, and

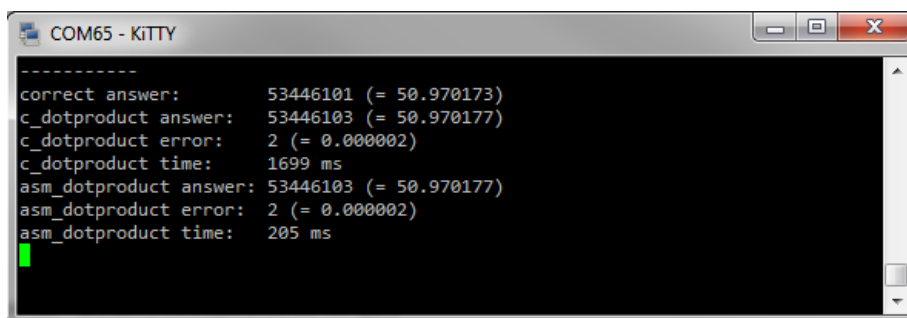
the vector **B** is the most recent n samples acquired from the sensor. You will learn how to design the filter coefficients in a future subject. Filtering is so common that microcontroller hardware often includes machine code instructions that accelerate the calculation of such equations.

Hints

- It's recommended that you implement the C version first.
- The correct answer hardcoded into `main.c` was calculated using double precision floating point arithmetic, and therefore might not exactly match the result computed using Q20 fixed point. Your error should be small (but not necessarily zero).
- You can perform a 32-bit x 32-bit multiply and a 64-bit addition in a single instruction using `umlal`.
- For the bit shifts, use the "logical shift left" and "logical shift right" instructions (`lsl` and `lsr`). These work with *unsigned* integers, in contrast with "arithmetic shifts" that work with signed integers.
- You will need to think about how to shift the 64-bit intermediate result into a 32-bit integer. You might find it helpful to draw a diagram showing the sequence of bits in the 64-bit number and how these are split across two registers. On your diagram, you can show how the two parts of this number need to be combined to produce the final result.

Assessment

A completed version of this project might look like the screenshot below.



```

-----
correct answer:      53446101 (= 50.970173)
c_dotproduct answer: 53446103 (= 50.970177)
c_dotproduct error:  2 (= 0.000002)
c_dotproduct time:   1699 ms
asm_dotproduct answer: 53446103 (= 50.970177)
asm_dotproduct error: 2 (= 0.000002)
asm_dotproduct time:  205 ms

```

To complete this lab task, you must demonstrate:

- A handwritten assembly function that calculates the same result but outperforms the C implementation.

Optional extension task

- Make your assembly code as fast as possible. What's the best time that you can achieve?
- Once you have both versions working, try adjusting the compiler optimisation level (in the Project Properties -> C/C++ Build -> Settings -> Optimization). After changing the compiler settings, Clean the project (on the right click menu in the Project Explorer) to force a full re-build. Can you beat an optimising compiler?