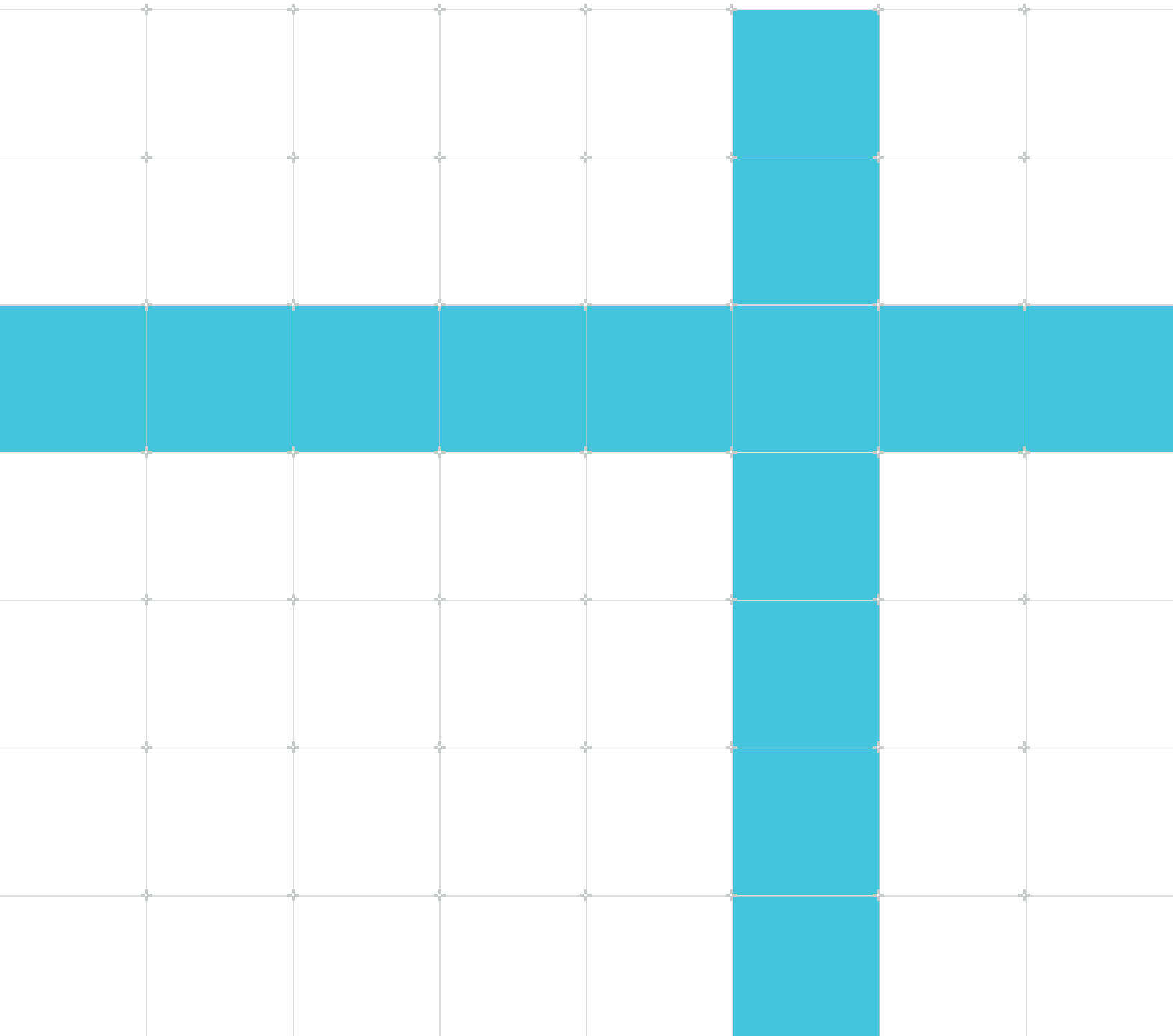
Using the PMU and the Event Counters in DS-5

Version 1.0

Issue 02
102601_0100_02_en



Using the PMU and the Event Counters in DS-5

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	1 January 2020	Non-Confidential	First version

Proprietary Notice

or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Using the PMU and the Event Counters in DS-5

This tutorial details how to use the Performance Monitoring Unit (PMU) and the Event Counters in Arm DS-5 Development Studio. They can provide valuable information regarding system events, that could prove useful when assessing the performance and resource efficiency of your system. By the end of this tutorial you should be able to implement event counters in your code and interpret their results.

2. Background

The PMU architecture uses event numbers to identify events. These numbers are used to configure the counters so that each one only monitors a single event at a time. The PMU and event counters are part of the Performance Monitors Extension, making them optional features for Armv7-A/Armv7-R/Armv8-A implementations. With this in mind, you should check the Technical Reference Manual for your processor before continuing.

There are a number of advantages to using event counters; they provide highly accurate information and are a non-invasive debug feature with minimal impact on performance.

There are several situations where the developer might benefit from the use of event counters. Here are some examples:

- The counters can provide the total number of clock cycles and the number of instructions executed, from which a cycles per instruction figure can be derived. This can be a good indicator of the core's efficiency in a particular section of code.
- The counters can provide the total number of L1 D/I-Cache refills and L1 D/I-Cache accesses, which can be used to determine the ratio of L1 D/I-Cache misses to L1 D/I-Cache accesses. This provides an indication of how efficiently the cache is being used and can potentially explain excessive data accesses to the external memory system that are slowing down your program.

3. The PMU architecture and events

While the PMU architecture defines a set of common events, each implementation can also define its own specific events. It is therefore essential that you consult your implementation's Technical Reference Manual.

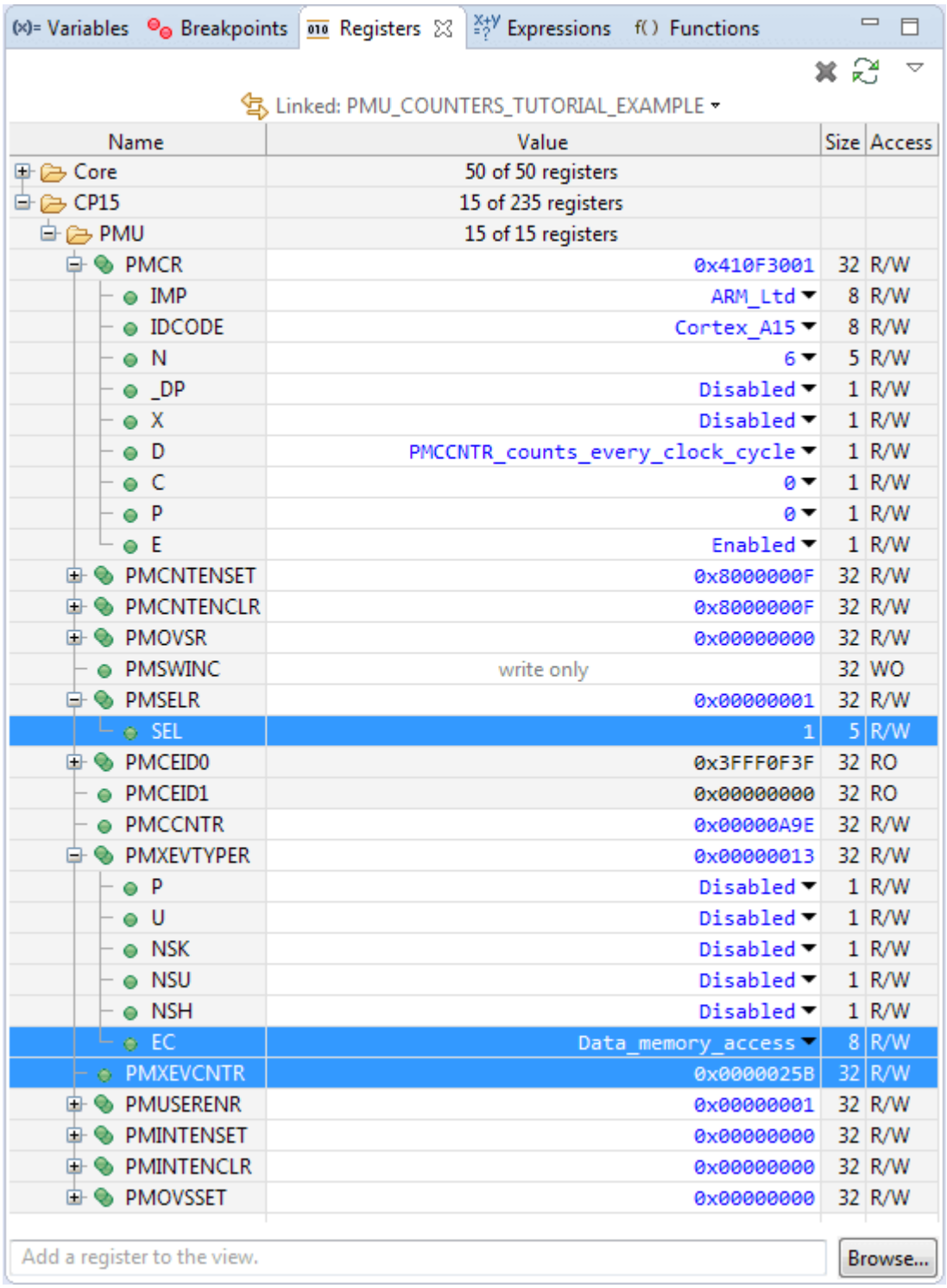
The counters are configured using the event numbers defined by the PMU architecture and specific implementation, and can each monitor any of the available events. It is important to note that there is an additional cycle counter that is not configurable and can only monitor cycles.

4. Using DS-5 in conjunction with event counters

DS-5 and event counters are a powerful combination, allowing you to step through code, set breakpoints, and access the value of any counter when the target is stopped. Using these tools

together you can monitor events for any particular section of code, aiding in the optimization process by detecting potential inefficiencies.

Figure 4-1: PMU registers view.



The PMU registers aren't available by default, to add them click on **Browse**, expand **CP15**, select **PMU** and click **OK**.

It should also be noted that halting the processor and entering debug mode is an invasive process that can affect the counter values. It is therefore recommended that you do not halt the processor if high precision is required.

5. Setting up and using the event counters

This section outlines the steps required to setup and use the event counters on a **Cortex-A15 (Armv7-A)**. The steps for Armv8-A processors are similar, though may be subject to small variations.

You can choose not to activate the cycle counter (steps marked as optional). This will not affect the event counters since they are independent from the cycle counter. If you do not need a readout of the number of cycles, then you can leave the counter off, which will reduce the performance impact of the PMU on your system.

1. **(Not essential) Enabling PMU user access** - in the Performance Monitors User Enable Register (PMUSERENR), set the EN,bit[0] to 1.
2. **Enabling the PMU** - in the Performance Monitors Control Register (PMCR), set the E,bit[0] to 1.
3. **Configuring an event counter**
    a. In the Performance Monitors Event Counter Selection Register (PMSELR), write the counter number (0-5) to the SEL,bits[4:0] you wish to configure.
    b. In the Performance Monitors Event Type Select Register (PMXEVTYPER), write the event number (from the event list) to evtCount,bits[7;0], in order to select the event being monitored by the counter.
4. **Enabling a configured event counter** - in the Performance Monitors Count Enable Set Register (PMCNTENSET), set Px,bit[x] (where x corresponds to the counter to be enabled 0-5) to 1.
5. **(Optional) Enabling the cycle counter (CCNT)** - in the Performance Monitors Count Enable Set Register (PMCNTENSET), set the C,bit[31] to 1.
6. **(Optional) Resetting the cycle counter (CCNT)** - in the Performance Monitors Control Register (PMCR), set the C,bit[2] to 1.
7. **Resetting the event counters** - in the Performance Monitors Control Register (PMCR), set the P,bit[1] to 1. The counters are now configured and will monitor events of interest as execution continues.
8. **(Optional) Disabling the cycle counter (CCNT)** - in the Performance Monitors Count Enable Clear Register (PMCNTENCLR), set the C,bit[31] to 1.
9. **Disabling an event counter** - in the Performance Monitors Count Enable Clear Register (PMCNTENCLR), set Px,bit[x] (where x corresponds to the counter to be disabled 0-5) to 1.
10. **Reading the value of an event counter**
    a. In the Performance Monitors Event Counter Selection Register (PMSELR), write the counter number (0-5) to the SEL,bits[4:0] you wish to read.
    b. The value of the selected counter is stored in the Performance Monitors Selected Event Count Register (PMXEVCNTR).
11. **(Optional) Reading the value of the cycle counter (CCNT)** - the value of the cycle counter is stored in the Performance Monitors Cycle Count Register (PMCCNTR). Source code performing this operation can be found in this downloadable project that you can import to DS-5. Please import this as you will need it for the next part of this tutorial.

Source code performing this operation can be found in this downloadable project that you can import to DS-5. Please import this as you will need it for the next part of this tutorial.

6. Worked example

The following chapter describes a worked example.

This example is written for the Cortex-A15 (Armv7-A), and runs on a CoreTile Express A15x2-A7x3 (TC2) on the Versatile Express platform. Arm DS-5 Development Studio 5.21.1 was used for testing.

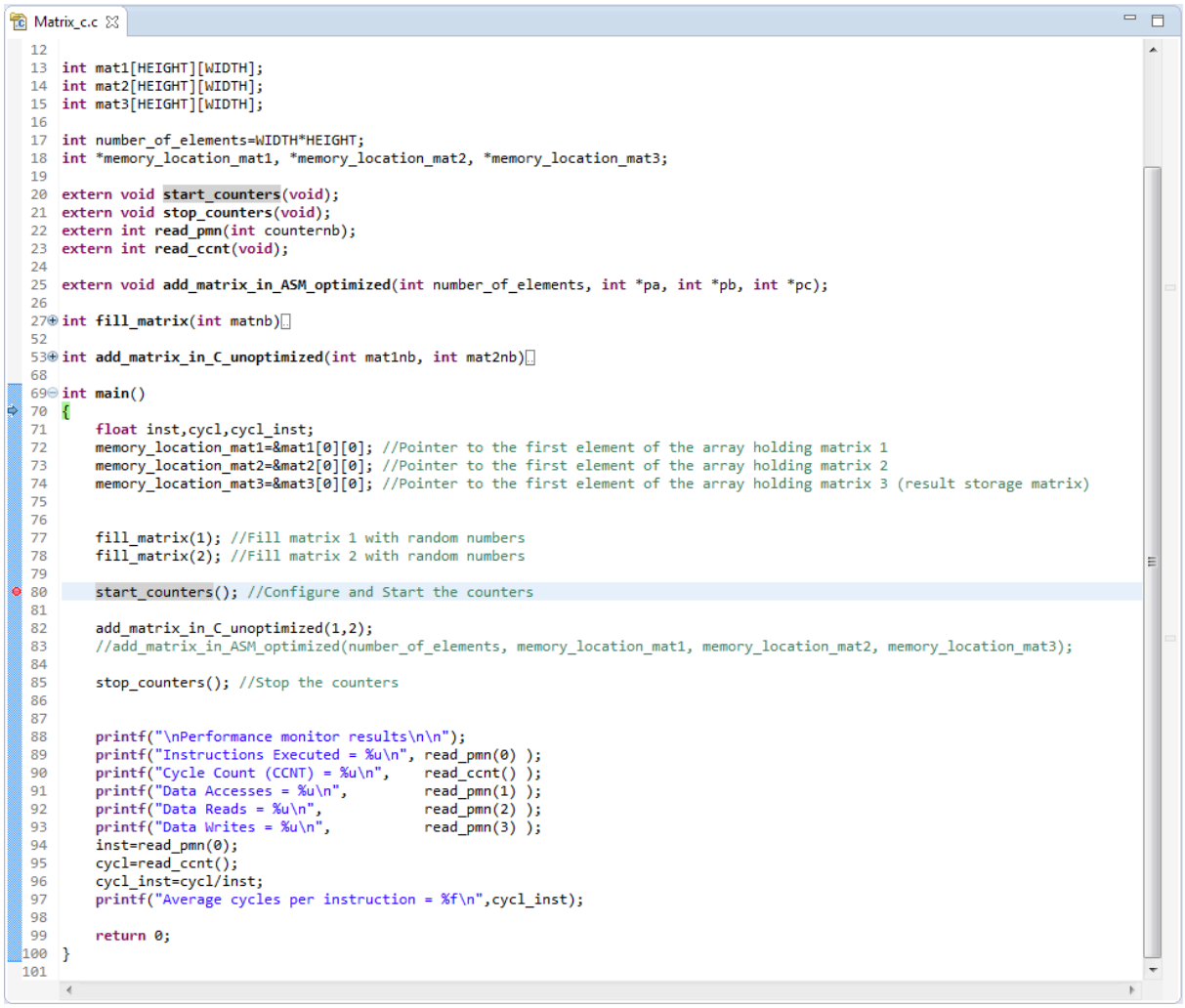
This example demonstrates setting up the event counters as described above. The counters will be used to measure the performance of a particular section of code, allowing for a comparison between its performance both before and after optimization.

This program creates and populates two matrices, adds them together, then stores the result to a third matrix. The addition can be performed in two different ways:

- `add_matrix_in_C_unoptimized()` uses two C loops to perform the addition one element at a time
- `add_matrix_in_ASM_optimized()` is written in Arm assembler and adds four elements at a time using vector operations (NEON instructions) In theory the second approach should be more efficient, the counters can be used to prove this.

1. In `main()` set a breakpoint on the function `start_counters()` (see image below) and use **F8** to run to it.

Figure 6-1: Breakpoint on the function `start_counters()`



2. Use **F5** to step into `start_counters()` and note how the event counters are being initialized as described earlier.
3. Use **F5** to step into the subsequently called functions and notice how the program uses assembly to modify the Performance Monitors Extension's registers.
4. Once you have reached `add_matrix_in_C_unoptimized(1,2)` note how the `start_counters()` and `stop_counters()` functions are located around it. The counters will only measure the performance of this block of code.
5. Set a breakpoint on `stop_counters()` and use **F8** to run to it.
6. Use **F5** to step into `stop_counters()`; at this stage the event counters are being stopped.
7. Use **F5** to step into the subsequently called functions and note how the program uses assembler to modify the Performance Monitors Extension's registers.
8. When the `stop_counters()` function has completed, use **F8** to finish the program. Disconnect from the target.

9. As mentioned earlier halting the core is invasive and leads to imprecise counter values. So, remove all breakpoints, connect to the target and use **F8** to run the code without stepping.
10. Observe the output in the App Console, you should get something similar to below:

```
Performance monitor results

Instructions Executed = 190730
Cycle Count (CCNT) = 122772
Data Accesses = 60006
Data Reads = 50003
Data Writes = 10003
Average cycles per instruction = 0.643695
```

11. Now go to `main()` and comment out the call to `add_matrix_in_C_unoptimized()` and uncomment the call to `add_matrix_in_ASM_optimized()`. It should look like this:

Figure 6-2: `add_matrix_in_ASM_optimized()` function with comment removed

```
79
80      start_counters(); //Configure and Start the counters
81
82      //add_matrix_in_C_unoptimized(1,2);
83      add_matrix_in_ASM_optimized(number_of_elements, memory_location_mat1, memory_location_mat2, memory_location_mat3);
84
85      stop_counters(); //Stop the counters
86
```

12. Rebuild the program and remove all breakpoints.
13. Connect to the target and use **F8** to run the code without stepping. Note the output in the App Console, it should look something like this:

```
Performance monitor results

Instructions Executed = 22529
Cycle Count (CCNT) = 54710
Data Accesses = 10791
Data Reads = 5339
Data Writes = 5452
Average cycles per instruction = 2.428426
```

We can observe that the optimized version requires six times less data accesses to process the same amount of data, and processes that data in half the cycles.

The use of event counters has confirmed that the optimized version of the matrix adding algorithm performs better than the unoptimized version, as expected.

7. Summary

This tutorial has introduced the concept of event counters in Armv7-A, Armv7-R and Armv8-A, and how they can be used in conjunction with DS-5 in order to monitor certain aspects of system performance. The tutorial outlines the steps required to configure these timers, and a real-world example has been used to show how they can be useful when optimizing and testing code.

8. Further reading

A detailed list of the common PMU events and their explanations can be found in the Architecture Reference Manual.

In this example vectorization was performed manually through the use of NEON instructions in Arm assembler, but it can be performed automatically using compiler options such as the Arm Compiler's –vectorize option, further information can be found in your compiler's documentation.