



Arm Streamline Tutorial

Revision: NA

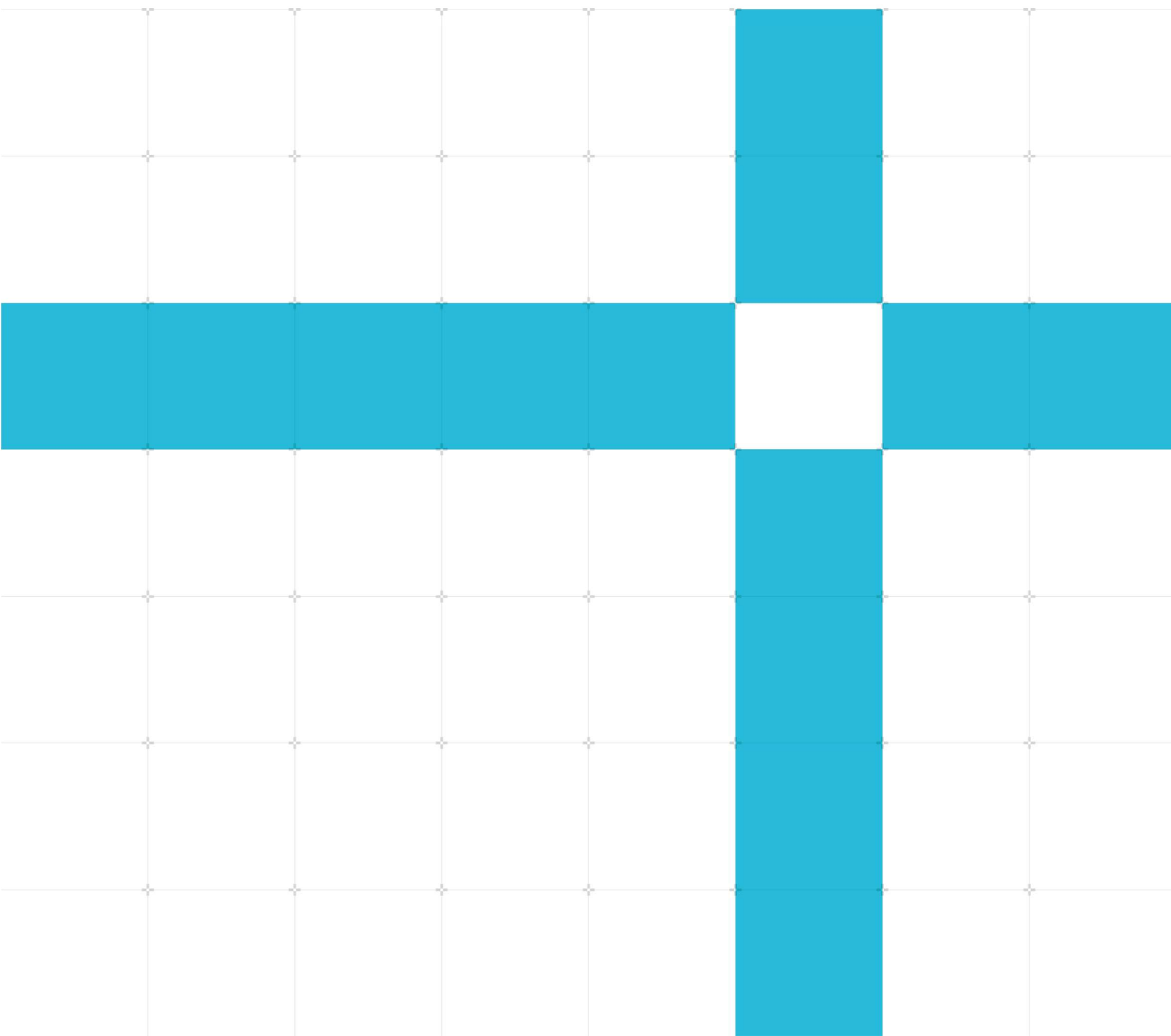
Analyzing an example Streamline report with Arm DS

Non-Confidential

Copyright © 2021 Arm Limited (or its affiliates).
All rights reserved.

Issue 0.0

NA



Arm Streamline Tutorial

Analyzing an example Streamline report with Arm DS

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Release information

Document history

Issue	Date	Confidentiality	Change
0.0	30 th of March 2021	Non- Confidential	First version

Confidential Proprietary Notice

This document is **CONFIDENTIAL** and any use by you is subject to the terms of the agreement between you and Arm or the terms of the agreement between you and the party authorized by Arm to disclose this document to you.

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information: **(i)** for the purposes of determining whether implementations infringe any third party patents; **(ii)** for developing technology or products which avoid any of Arm's intellectual property; or **(iii)** as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or **(iv)** for generating data for publication or disclosure to third parties, which compares the performance or functionality of the Arm technology described in this document with any other products created by you or a third party, without obtaining Arm's prior written consent.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly 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.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. 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.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <http://www.arm.com/company/policies/trademarks>.

Copyright © 2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20348

Confidentiality Status

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product Status

The information in this document is release quality.

Web Address

<http://www.arm.com>

Contents

1 Introduction	5
2 Import the Linux example project	6
3 Import Streamline example report.....	8
4 Re-analyze the report.....	11
5 Exploring the sample report.....	13
6 Add the missing source code	14
7 Example - How to analyze the sample capture.....	15
8 Add the project path to Streamline locations	18

1 Introduction

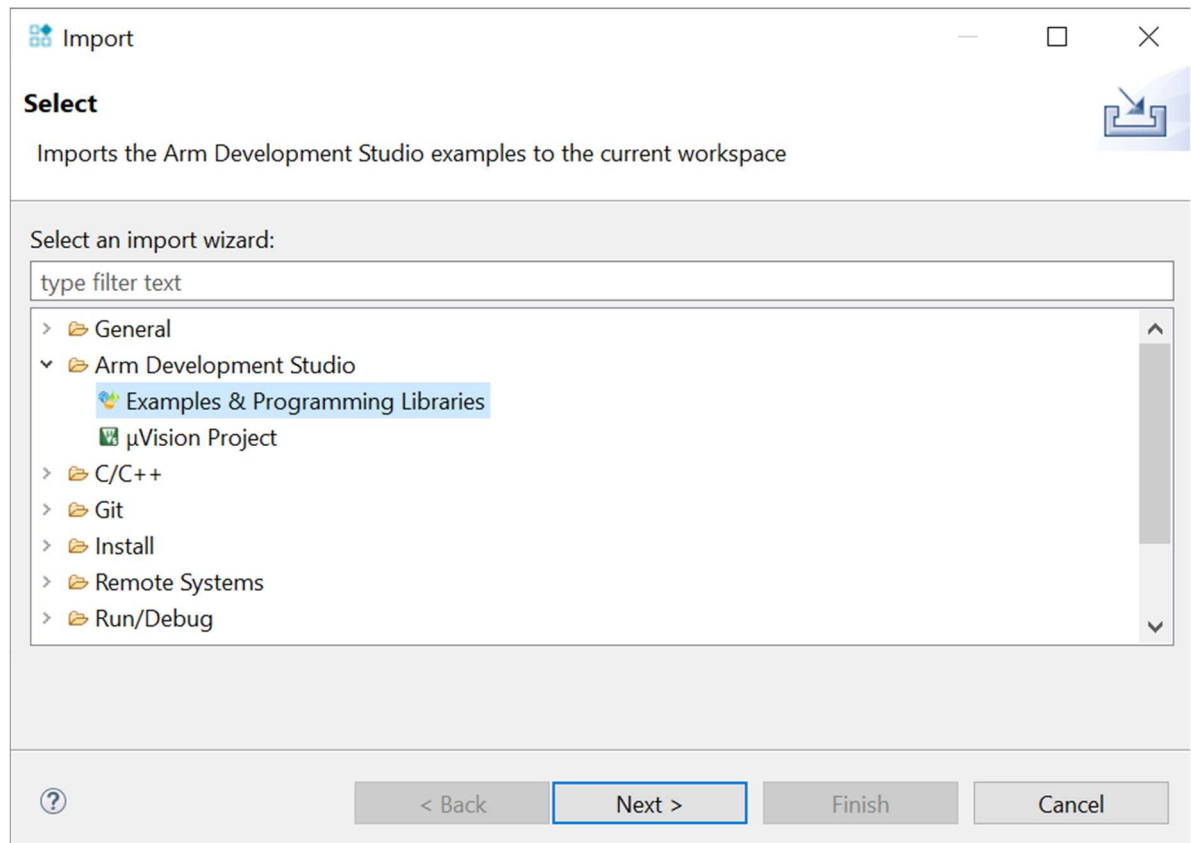
Learn how to import, re-analyze, and then explore the XaoS example report included in Streamline. The report was generated running the XaoS example included in **Arm Development Studio (Arm DS)**.

We use one of the example reports that are included with Streamline. These reports provide a quick way of exploring the features of the **Streamline performance analyzer**. The sample report used in this tutorial was captured by running the XaoS example included in Arm DS on a Cortex-A7 and Cortex-A15 Versatile Express board.

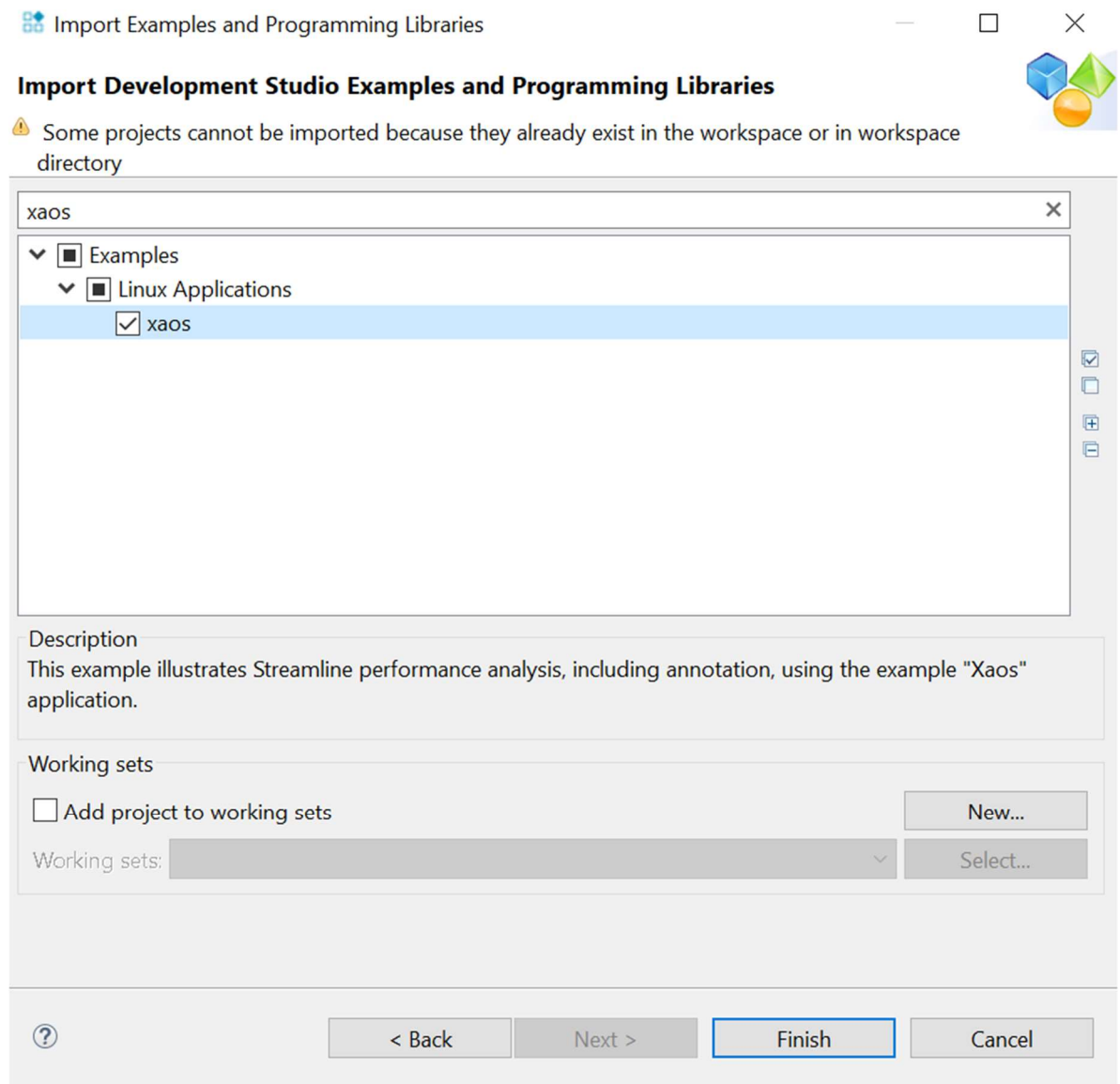
2 Import the Linux example project

The sample Streamline XaoS report was generated using the XaoS example project in Arm DS. Therefore, to work with the report, you must import the Arm DS XaoS example project to your Arm DS Workspace. To import the XaoS example project:

1. Open Arm DS IDE.
2. Click **File > Import**.
3. In the **Import** window, expand **Arm Development Studio** and select **Examples & Programming Libraries**. Click **Next**.



4. In the **Import Examples and Programming Libraries** window, expand **Examples > Linux Applications** and select **xaos**. You can also write "xaos" in the search bar to locate the example project. Click **Finish**.



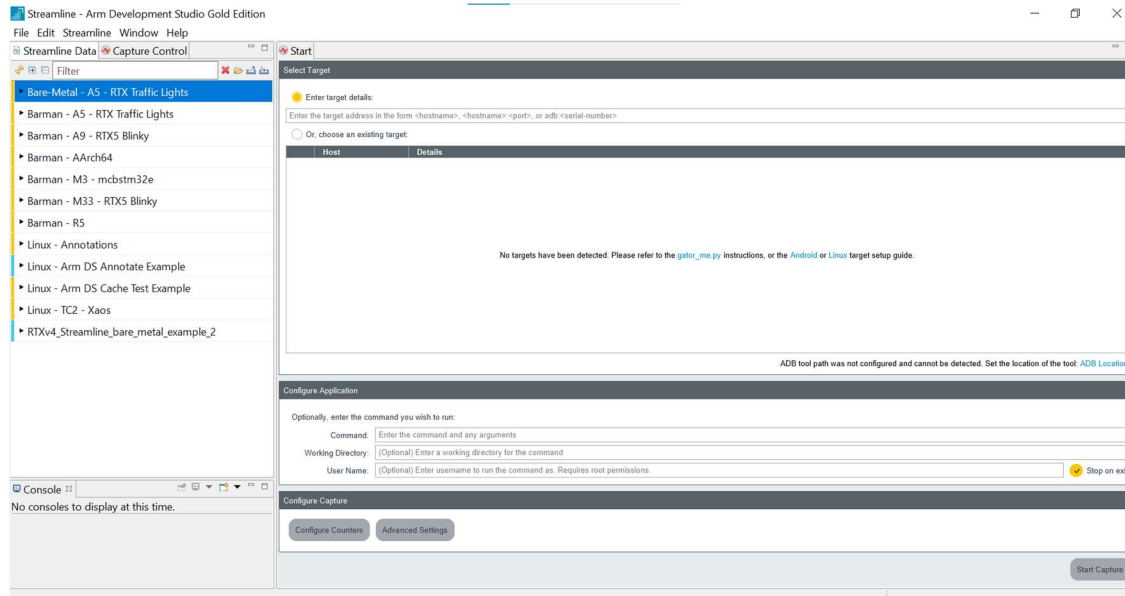
5. The xaos example project now appears in the **Project Explorer** view.

What is XaoS?:

XaoS is an open-source "interactive fractal zoomer" application. In our case, it runs multi-threaded to generate an interesting report.

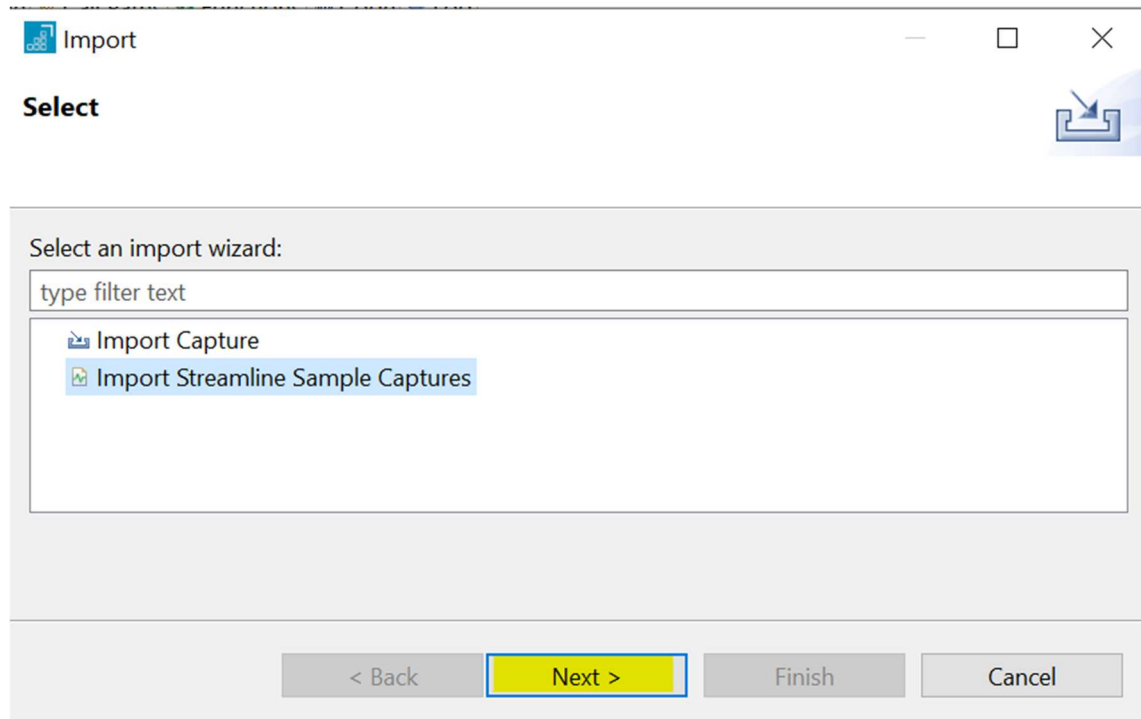
3 Import Streamline example report

Streamline is included with Arm DS installation. Now open Streamline. You will see the following IDE:




Now, you must import the Xaos Streamline example report. To import the example reports:

1. Click **File > Import...**
2. Select **Import Streamline Sample Captures** and click **Next**.



3. You can now select the sample Streamline captures available that you want to import. For this tutorial, select the **Linux > Arm DS Xaos Example**. To import the sample capture, click **Finish**.

—□×

Import Sample Captures

Select the sample captures to import from the list below:

▼ ☒ Linux

☒ Annotations Examples

☒ Arm DS Annotate Example

☒ Arm DS Cache Test Example

☒ Arm DS Xaos Example

▼ ☒ Bare-Metal

☒ Bare-Metal Agent Example running on Armv8 model

☒ RTXv4 Traffic Lights Example running on Cortex-A5

☒ Bare-Metal Agent Example running on Cortex-R5 model

☒ Bare-Metal Agent Example running on Cortex-M3 on MCBSTM32E board

☒ Bare-Metal Agent Example Blinky running on Cortex A9

☒ Bare-Metal Agent Example Blinky running on Cortex-M33

▼ ☒ Instruction Trace

☒ RTXv4 Traffic Lights Example running on Cortex-A5

< Back

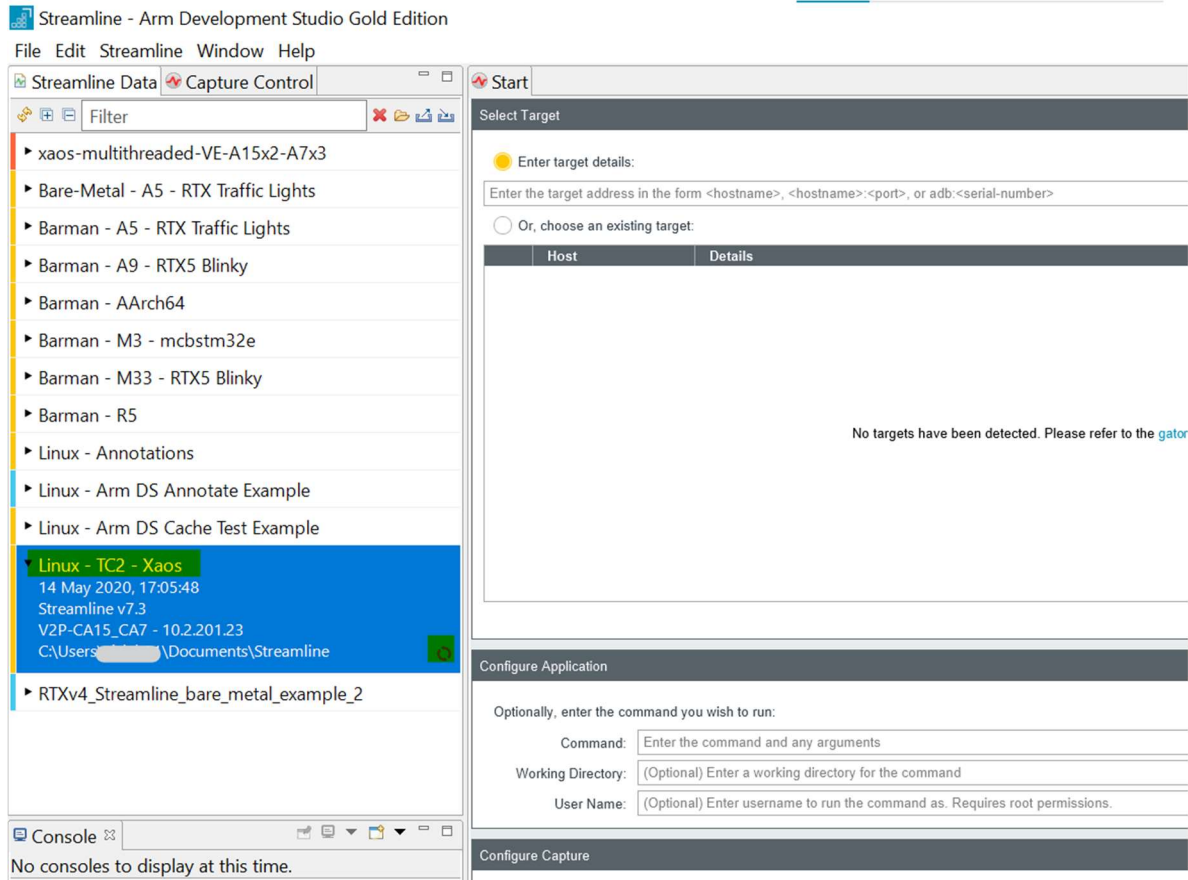
Next >

Finish

Cancel

4 Re-analyze the report

The **Linux - TC2 - Xaos** capture is listed in the **Streamline Data** view. Notice that the vertical bar at the left of the capture (in the list of captures) is yellow, meaning it must be re-analyzed.



Why do I have to re-analyze the report?:

Since the report was generated with a different version of Arm DS, you must re-analyze it in to update it.

In the **Analyze** dialog box, you can load extra program images. These images can be useful for gaining a more comprehensive idea of system execution, or alternatively, for focusing on one particular program. Streamline loads debug symbols if available.

To open the **Analyze** dialog, double-click on the Streamline capture. In the **Program Images** tab, select the ELF images **xaos** and **libpthread-2.17.so**. Then click **Analyze**. Re-analyzing the capture might take some time.

 Linux - TC2 - Xaos



Analyze




Choose the settings to produce a new report.









Analysis

☒ Process Extra Debug Information (when available)

Resolution mode: Normal

Program Images | Script Search Paths

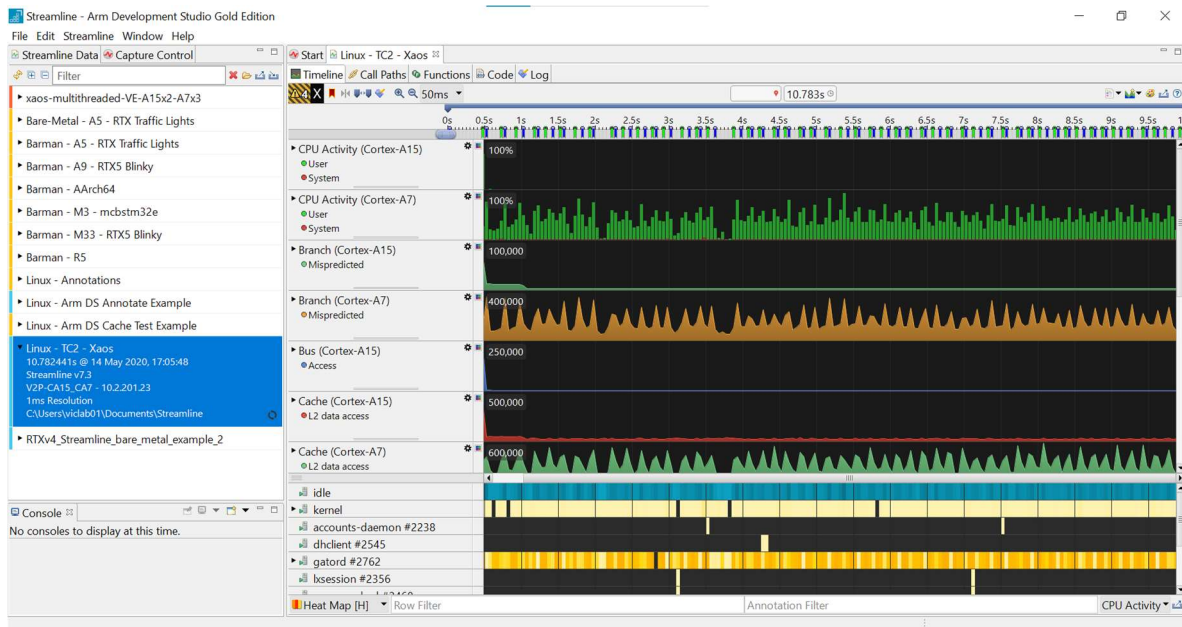
☒  Add ELF image...  Select Separate Debug Image...  Remove

Use	Name	Symbols	Debug Info	CFI	Remarks
<input checked="" type="checkbox"/>	 xaos				
<input checked="" type="checkbox"/>	 libpthread-2.17.so				Debug data in separate files

Analyze Cancel

5 Exploring the sample report

The Xaos report opens in a new tab automatically once you have re-analyzed it. It provides a summary view of CPU performance, branch mispredictions, memory and cache accesses, along with a sample of the frame buffer. Note that the vertical bar of the capture is now colored blue in the **Streamline Data** view, meaning it is up to date.



In the panel underneath these charts, there is a heatmap for each process and thread. Click individual processes or threads to reconfigure the charts to reflect them. You can also see annotations on each thread and the frame buffer, which are being passed from the source code. These annotations are a very useful profiling tool when you are working on your own code.

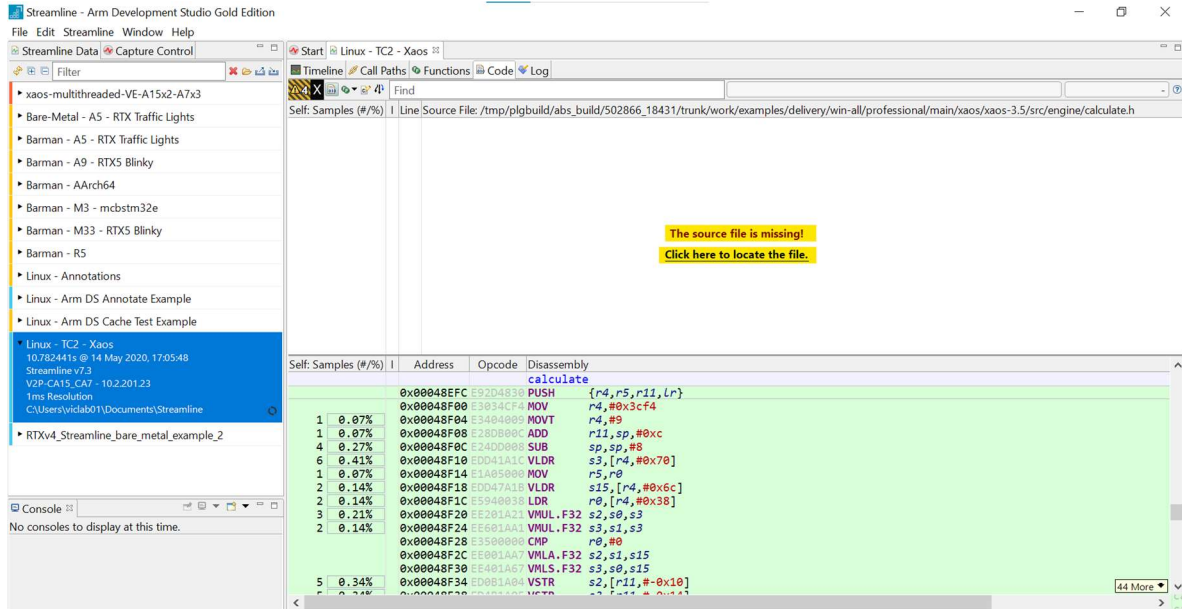
Along the top of the report, there are tabs which break down the data all the way to source code level. When using Streamline to profile your own software, look for areas where the CPU is being pushed hard, or where the cache is being heavily used. Then, try to identify which line of code is causing this.

I see numerous **<unknown code in "x">** in the Call Paths and Functions tabs :

This issue relates to the images that you use for analysis in the first place. By default, this capture only includes a couple of images (in this case, the images `xaos` and `libpthread-2.17.so`). So, while Streamline knows that function "x" is taking up a certain percentage of processor time, it does not know what this code is.

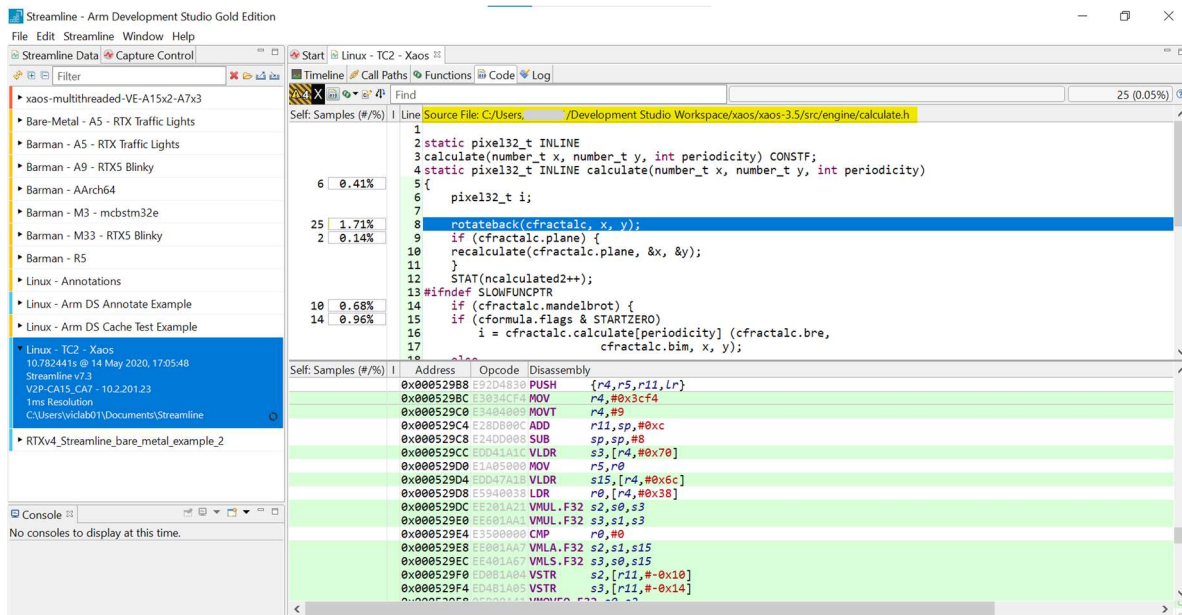
6 Add the missing source code

In the **Functions** tab, try double-clicking on a function name such as **calculate**. This action takes you to the corresponding line of source code. However, the source file is missing.



Follow the on-screen hint to **Click here to locate the file**. You can find the source code for the XaoS example project in your workspace. You can locate the file in `C:\Users\<User>\Development Studio Workspace\XaoS\XaoS-3.5\src\engine\calculate.h`.

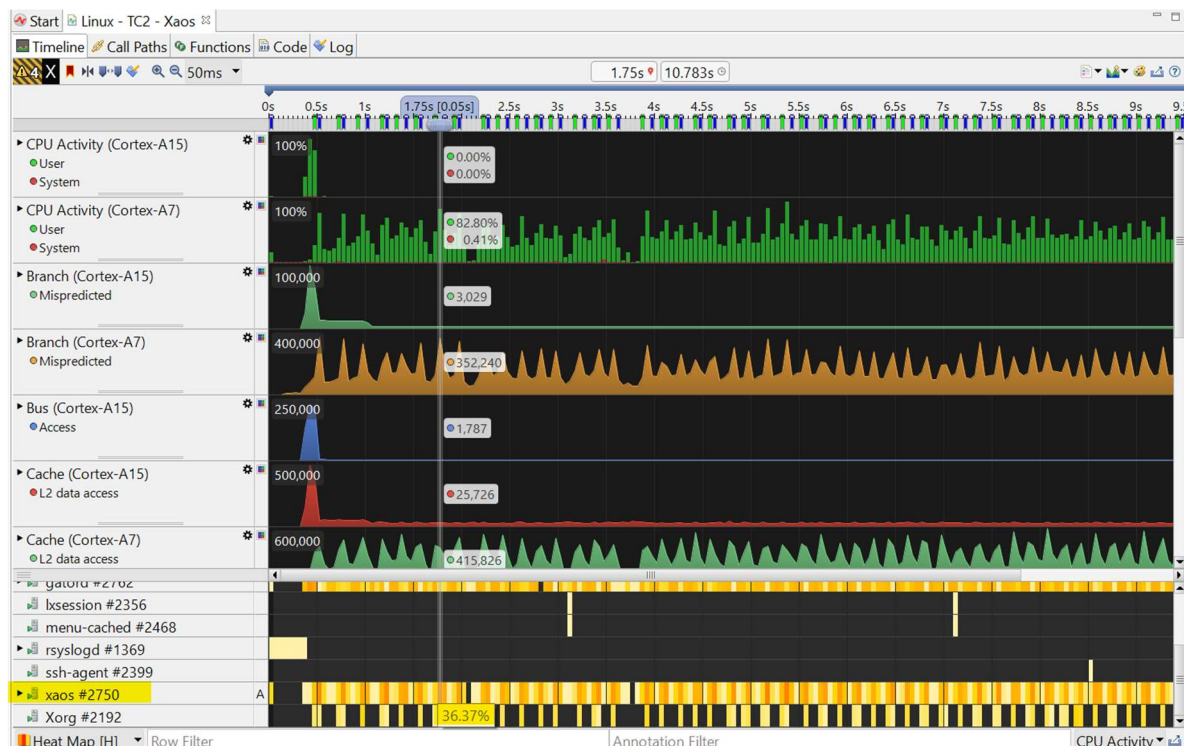
Once you have located the file, the source code appears:



7 Example - How to analyze the sample capture

In the **Timeline** view, you can see a timeline with information such as CPU activity, branch mispredictions, and instructions executed. You can click the timeline to set a cross-section market at 1.75 seconds of execution. That point in the execution time corresponds to one of the activity peaks in the Cortex-A7 core.

In the heat map below, you can see the CPU activity for the different processes. At 1.75s, you notice that the **xaos #2750** process is in orange color, using 36.37% of the **CPU Activity**.



Go to the **Call Paths** view selecting the Call Paths tab. The **main** code is contained within the **xaos #2750** process and thread. We want to see which functions in **main** are using more resources. If you click **main**, you can see that more than half of the samples (52.81%) in **main** correspond to the function **VisualAnnotateImage**. That corresponds to a 11.11% of samples in the **xaos #2750** process.

[Process]/[Thread]/Code	Total: Samples (#/%)	Self: Samples (#/%)	% Process: Samples	Stack	Location
idle	38,354 73.09%			-	
idle	38,354 73.09%		100.00%	-	
<unknown code in kernel>	38,325 73.04%	38,325 99.92%	99.92%	0	-
<unknown code in gator>	29 0.06%	29 0.08%	0.08%	0	-
xaos #2750	6,596 12.57%			-	
xaos #2750	3,315 6.32%		50.26%	-	
<unknown code in kernel>	1,418 2.70%	1,418 21.50%	21.50%	0	-
main	1,386 2.64%	0 0.00%	21.01%	44	ui.c:1086
main_loop	1,385 2.64%	0 0.00%	21.00%	56	ui.c:1724
ui_updatestatus	734 1.40%	0 0.00%	11.13%	92	ui.c:360
VisualAnnotateImage	733 1.40%	732 11.10%	11.11%	128	annotate.c:148
addBmpHeader	1 < 0.01%	1 0.02%	0.02%	164	annotate.c:84
uih_drawwindows	1 < 0.01%	0 0.00%	0.02%	128	wstack.c:339
getpos	1 < 0.01%	1 0.02%	0.02%	152	messg.c:18
uih_do_fractal	631 1.20%	0 0.00%	9.57%	88	ui_helper.c:912
do_fractal	568 1.08%	0 0.00%	8.61%	124	zoom.c:1575
calculatenewinterruptible	509 0.97%	5 0.08%	7.72%	160	zoom.c:1382
processqueue	421 0.80%	1 0.02%	6.38%	196	zoom.c:1334
calcolumn_32	234 0.45%	1 0.02%	3.55%	240	zoomd.c:145
calculate	232 0.44%	105 1.59%	3.52%	256	calculate.h:5

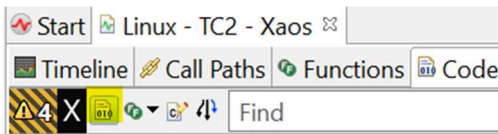
Function Name	Samples (#/%)	Instances	Location
VisualAnnotateImage	732 52.81%	1	annotate.c:148
calculate	285 20.56%	4	calculate.h:5
mand_calc	122 8.80%	2	xaos
mkrealloc_table	50 3.61%	1	zoom.c:457
calcolumn_32	40 2.89%	2	zoomd.c:145
fillline_32	40 2.89%	1	zoomd.c:296
calcline_32	30 2.16%	2	zoomd.c:39
look132	13 0.94%	2	autod.c:4

Now, select the **Functions** tab to see the **Functions** view. You can see that the function `VisualAnnotateImage` appears at the top and that it is one of the functions with the highest number of samples. Double-click the function to open the **Code** view for the `VisualAnnotateImage` function.

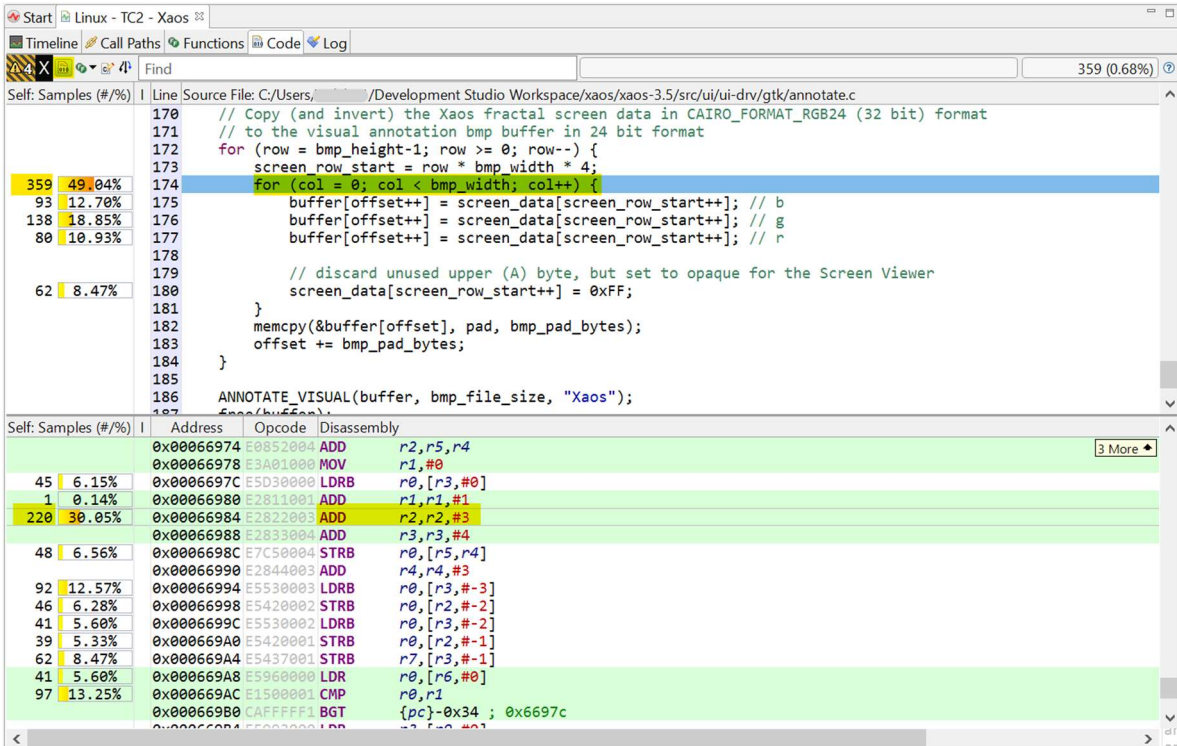
Function Name	Self: Samples (#/%)	Total: Samples (#/%)	Instances	Stack	Size	Location	Image
<unknown code in kernel>	47,322 90.18%	47,322 90.18%	23	0	-	-	-
calculate	1,462 2.79%	2,999 5.72%	21	16	240	calculate.h:5	xaos
VisualAnnotateImage	732 1.39%	733 1.40%	1	36	424	annotate.c:148	xaos
<unknown code in libc-2.17.so>	723 1.38%	723 1.38%	7	0	-	-	-
mand_calc	592 1.13%	592 1.13%	10	16	804	xaos	xaos
<unknown code in libpixmap-1.so.0.28.2>	485 0.92%	485 0.92%	2	0	-	-	-
fillline_32	246 0.47%	246 0.47%	9	4	124	zoomd.c:296	xaos
calcolumn_32	175 0.33%	1,376 2.62%	10	44	984	zoomd.c:145	xaos
calcline_32	107 0.20%	1,007 1.92%	10	44	932	zoomd.c:39	xaos
mkrealloc_table	85 0.16%	85 0.16%	5	52	2,704	zoom.c:457	xaos
moveoldpoints	79 0.15%	79 0.15%	5	40	244	zoom.c:954	xaos
mand_peri	48 0.09%	48 0.09%	11	16	1,236	xaos	xaos
plt [xaos]	46 0.09%	46 0.09%	5	0	2,456	xaos	xaos
_aeabi_uidivmod	39 0.07%	39 0.07%	5	0	28	xaos	xaos

One of the advantages of Streamline is that you can obtain a performance analysis from a high level and general perspective to the lowest level possible. In this example, we began detecting one of the processes with the highest CPU activity. Then, we detected one of the functions with more samples, within the main code that is executed in that process. Now, we can reach the lowest level possible, which is detecting which instructions are executed more frequently.

In the **Code** view, you see the code for the `VisualAnnotateImage` function, which is included in `annotate.c`. Select the highlighted icon in the following image to also show the assembly code:



The code in line 174 (`for` loop) corresponds to the 49.04% of the samples in the function. If you click the line 174, you can see the associated assembly code in green. You have been able to locate the instructions that are constantly executed in your code. This procedure can help you detecting bottlenecks in your code. The instruction `ADD r2, r2, #3` corresponds to the 30.05% of the samples in the `VisualAnnotateImage` function.



8 Add the project path to Streamline locations

If you generate and save your own Streamline reports in a project, you must include the location of the reports. You must indicate the path to the folders with the captures you want to analyze, so Streamline can find the captures in your file system.

As an example, in case you created and saved reports in your Xaos project:

1. In Streamline, click **Window > Preferences**.
2. To add or remove locations where Streamline analysis data can be found, select **Data Locations**.
3. Click **New** and select the *Streamline* folder with captures within the *xaos* project folder in your workspace. Then click **Apply and close**.
4. Your captures are now listed in the **Streamline Data** view.