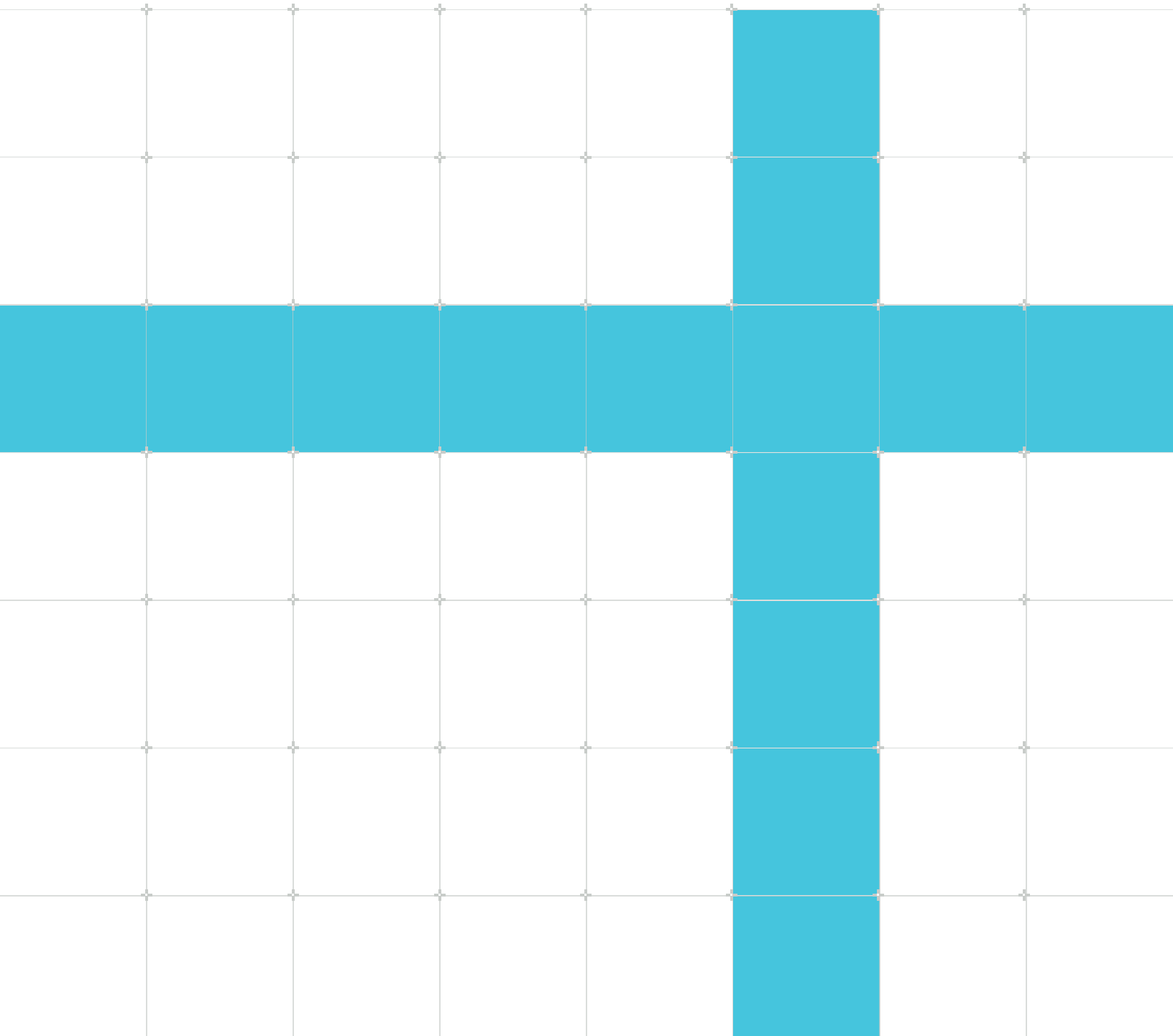
# Analyzing an example Streamline report with Arm DS

Version 1.0



**Issue 01**
102590_0100_01_en



# Analyzing an example Streamline report with Arm DS



## Release information

### Document history

| Issue | Date | Confidentiality | Change |
|---|---|---|---|
| 0100-01 | 30 March 2021 | Non-Confidential | First release |

## Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349|version 21.0)

## Confidentiality Status



## 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. Introduction

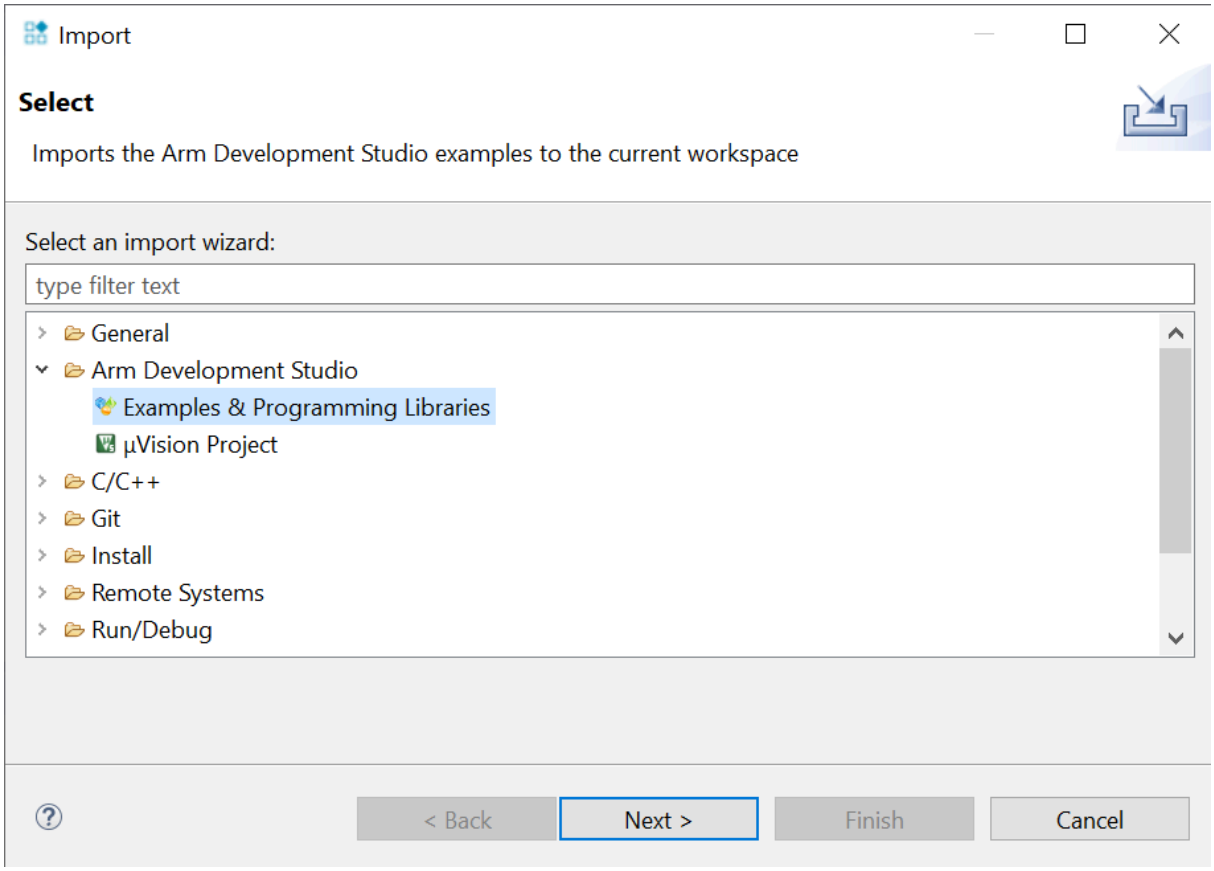
We use one of the example reports that are included with Streamline. These reports provide a quick way of exploring the features of the Streamline performance analyzer. The sample report used in this tutorial was captured by running the XaoS example included in Arm Development Studio (Arm DS) on a Cortex-A7 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:

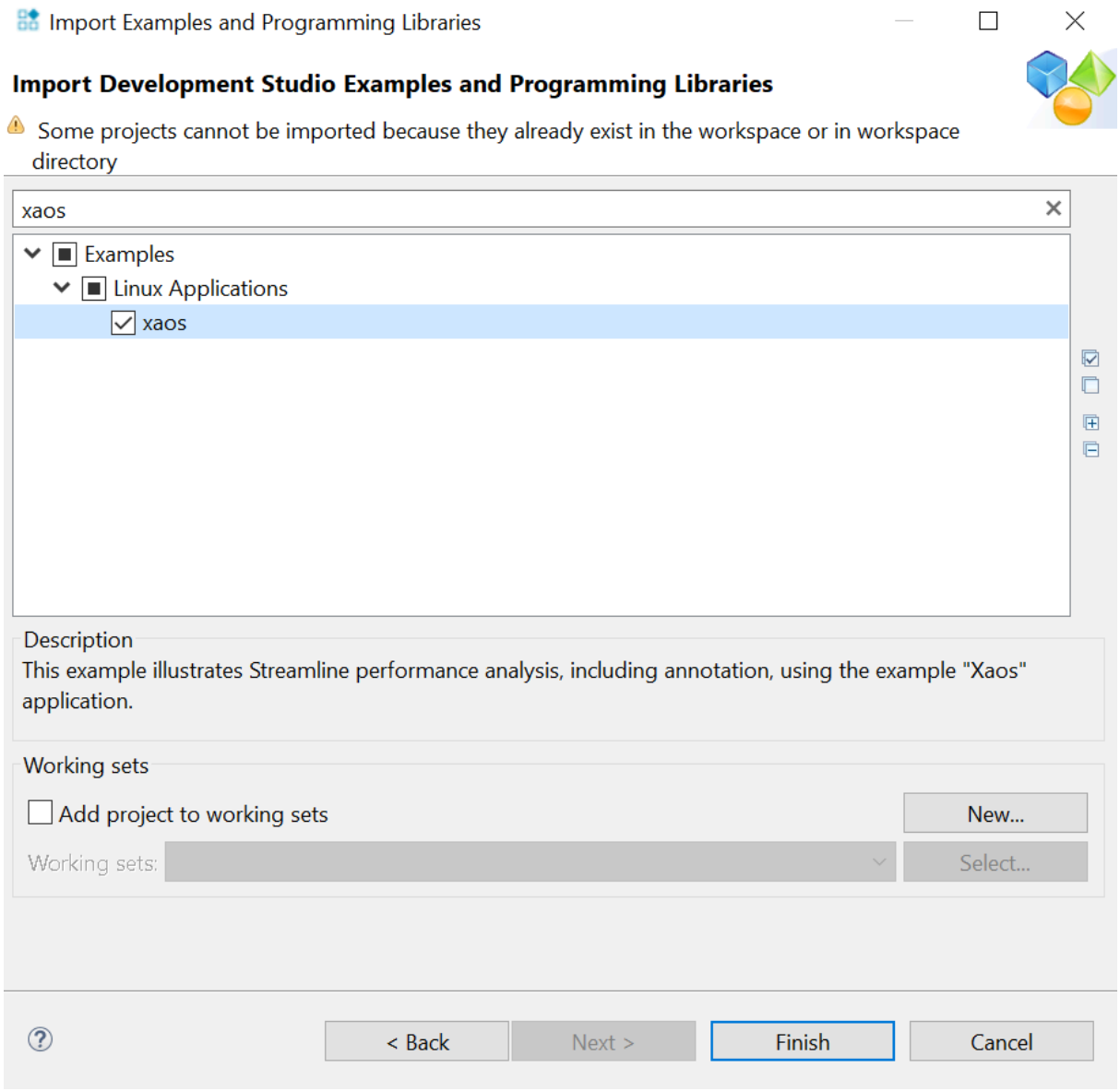
1. Open Arm DS IDE.
2. Click File > Import.
3. In the Import window, expand Arm Development Studio and select Examples & Programming Libraries. Click Next.

**Figure 2-1: Examples & Programming Libraries option from import window**




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.

**Figure 2-2: xaos option from Import Examples and Programming Libraries window**




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

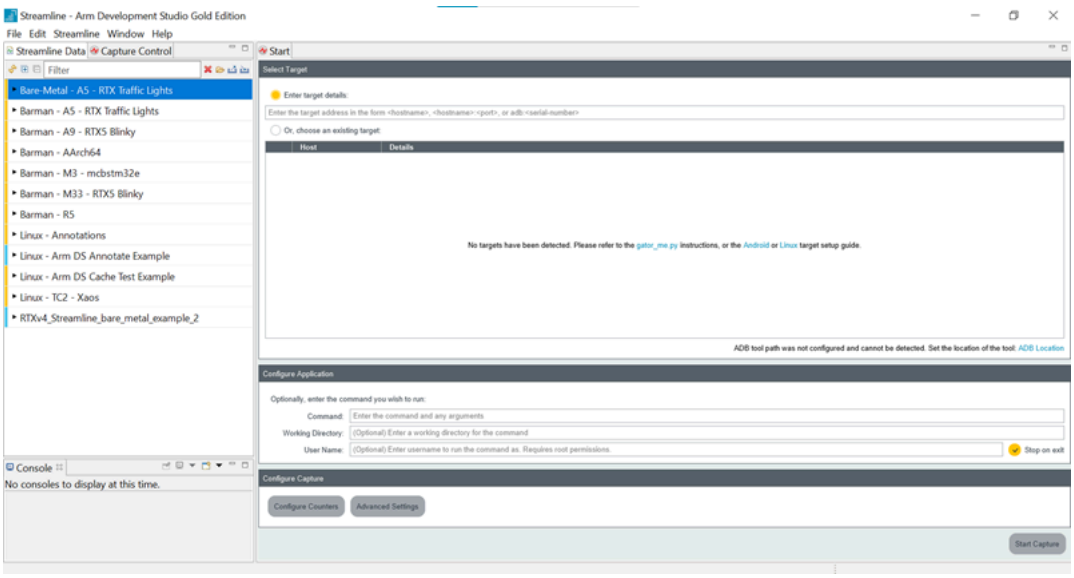
**What is XaoS?:**

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

# 3. Import the Streamline example report

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

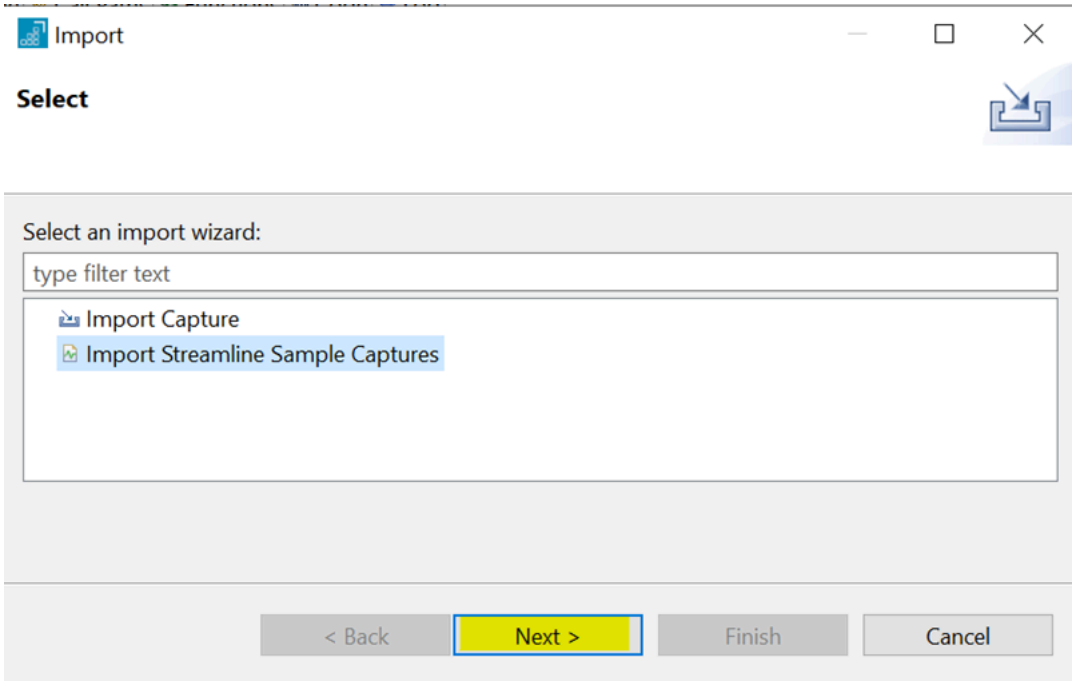
**Figure 3-1: Streamline IDE**




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

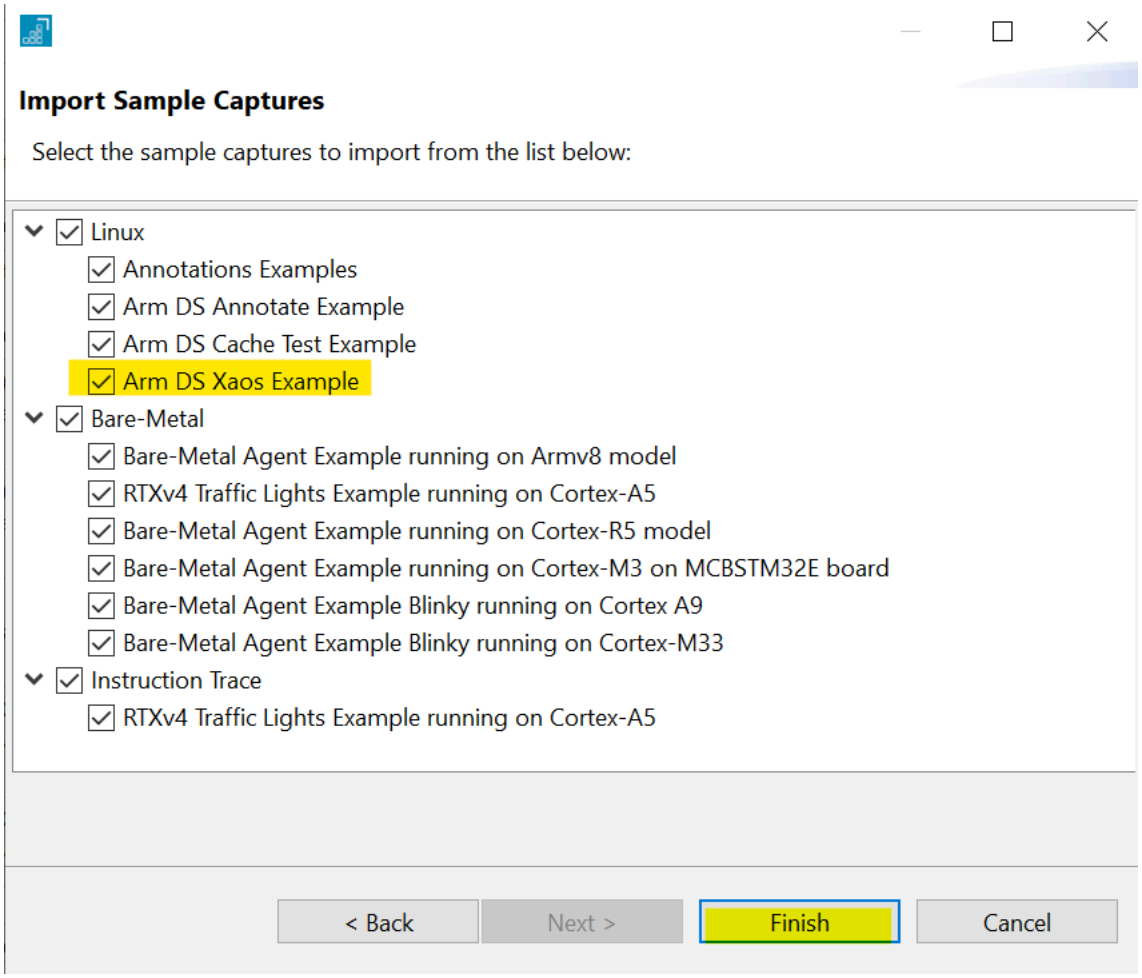
1. Click File > Import...
2. Select Import Streamline Sample Captures and click Next.

**Figure 3-2: Import Streamline Captures option.**




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.

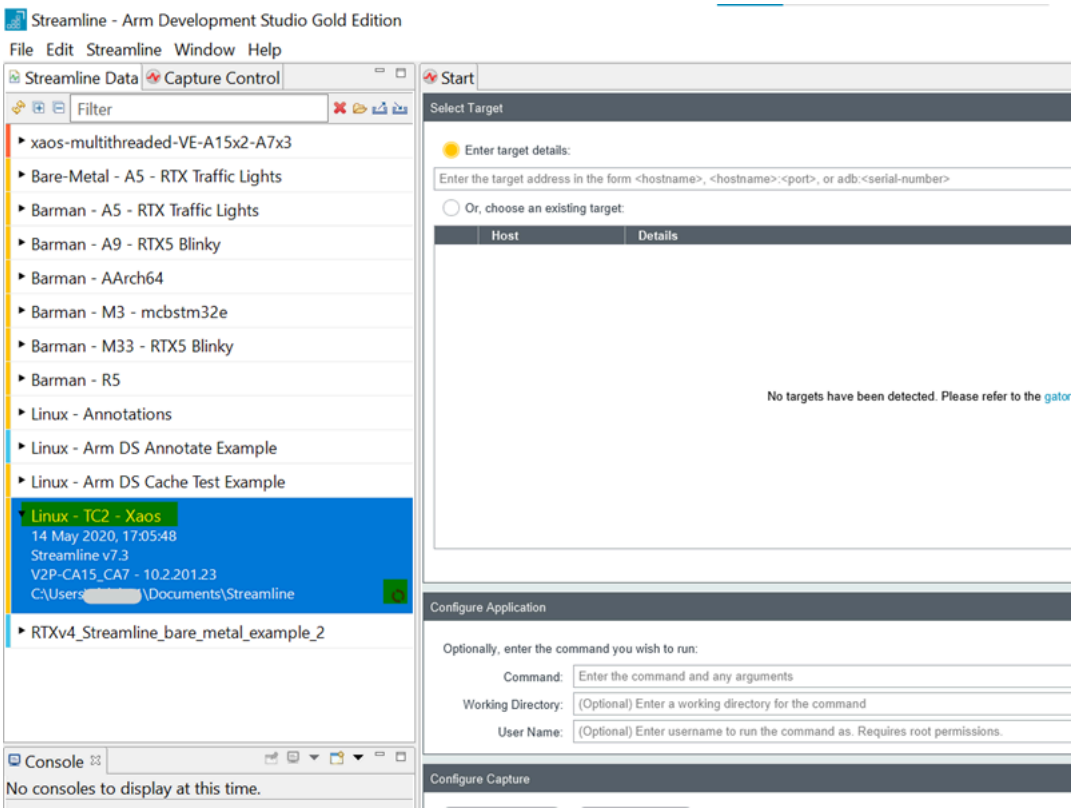
**Figure 3-3: Arm DS Xaos Example option**



# 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.

**Figure 4-1: Streamline Data view yellow bar**

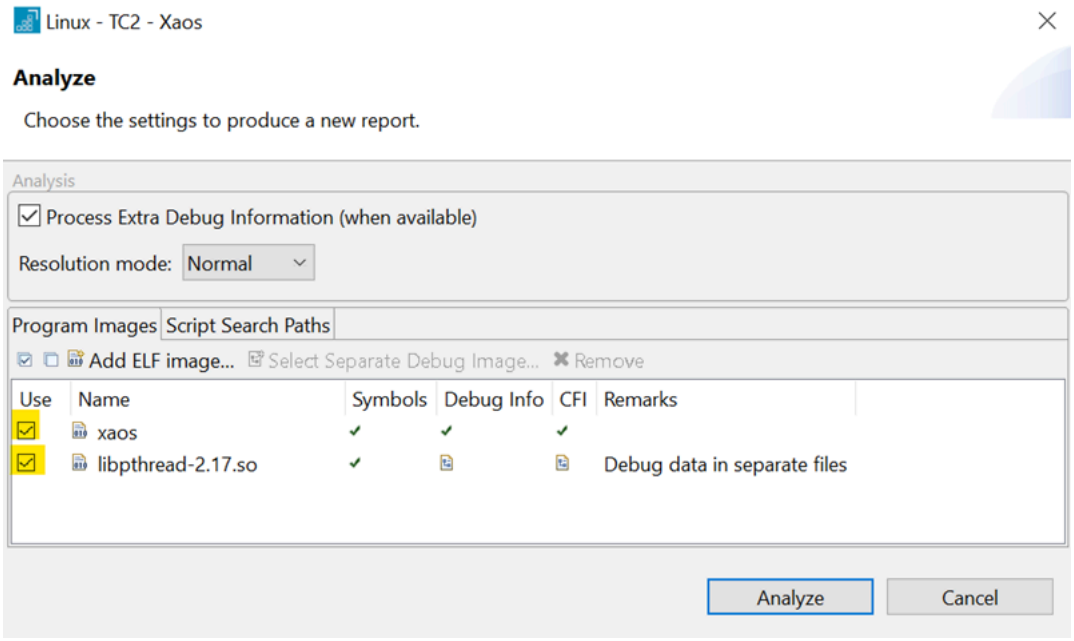



## Why do I have to re-analyze the report?

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.

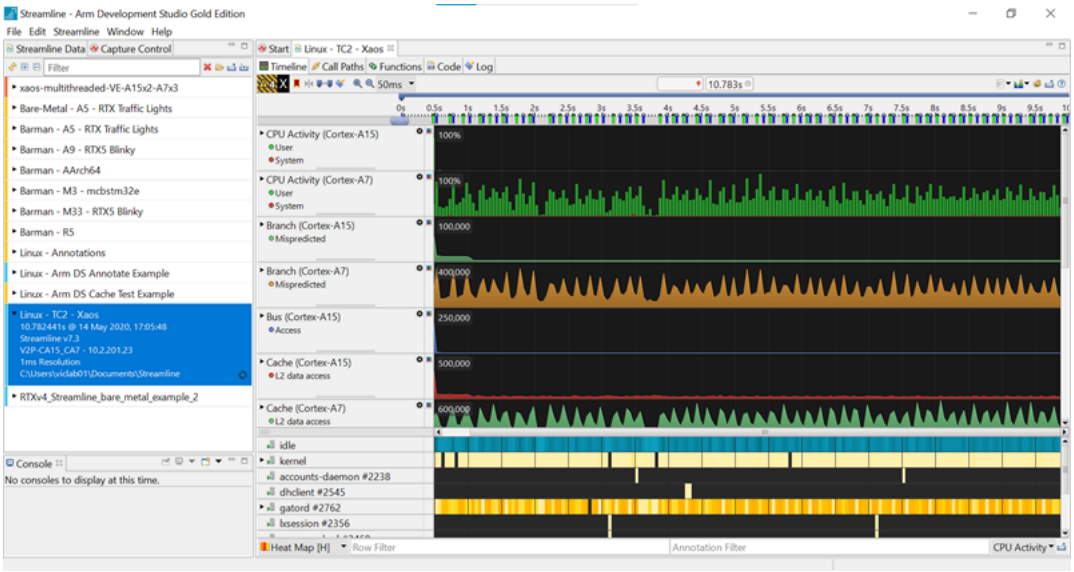
**Figure 4-2: Streamline Data view analyze window**



# 5. Exploring the example 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.

**Figure 5-1: Streamline data view blue bar**




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.

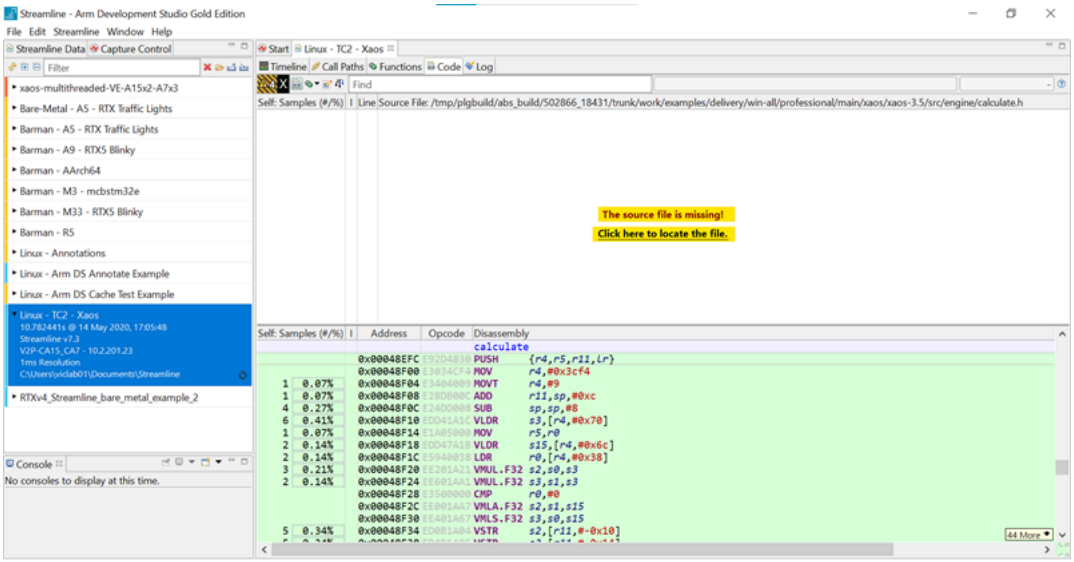
**I see numerous `<unknown code in "x">` in the Call Paths and Functions tabs:**

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.

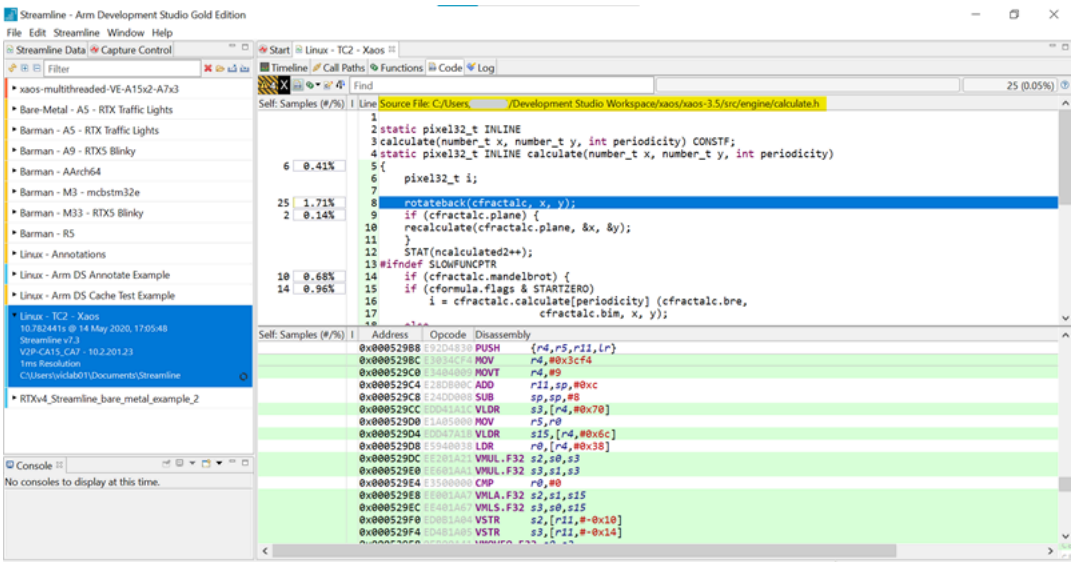
**Figure 6-1: A description of the image for screen readers**




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:

**Figure 6-2: A description of the image for screen readers**

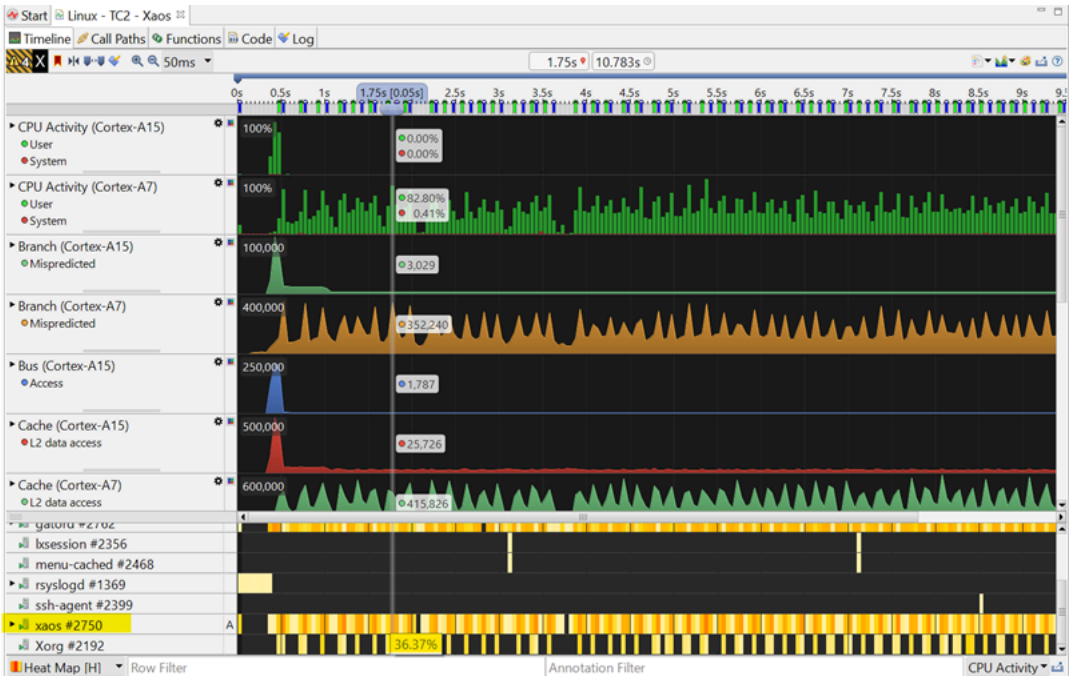


# 7. Example - How to analyze the Streamline capture

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.

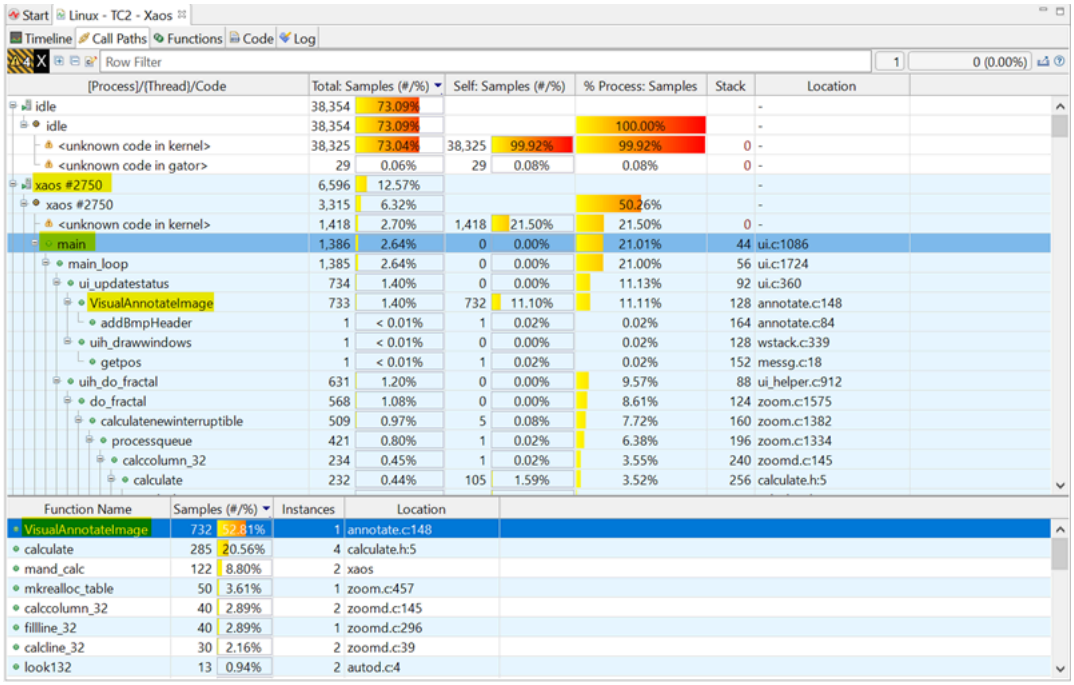
In the heat map below, you can see the CPU activity for the different processes. At 1.75s, you notice that the `xaos #2750` process is in orange color, using 36.37% of the CPU Activity.

**Figure 7-1: Heat map showing the xaos #2750 process using 36.37% of CPU Activity**



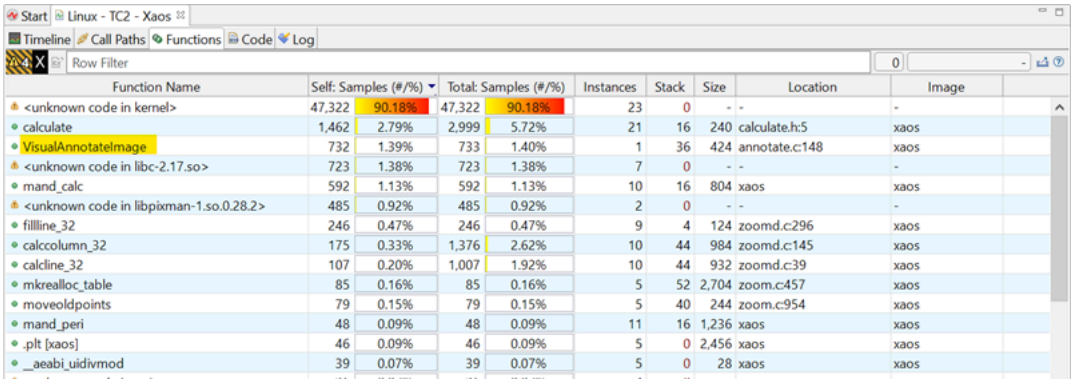

Go to the Call Paths view selecting the Call Paths tab. The `main` code is contained within the `xaos #2750` process and thread. We want to see which functions in `main` are using more resources. If you click `main`, you can see that more than half of the samples (52.81%) in main correspond to the function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

**Figure 7-2: Call Paths view**




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.

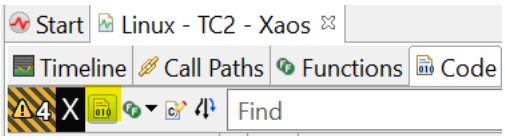
**Figure 7-3: Functions view**




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.

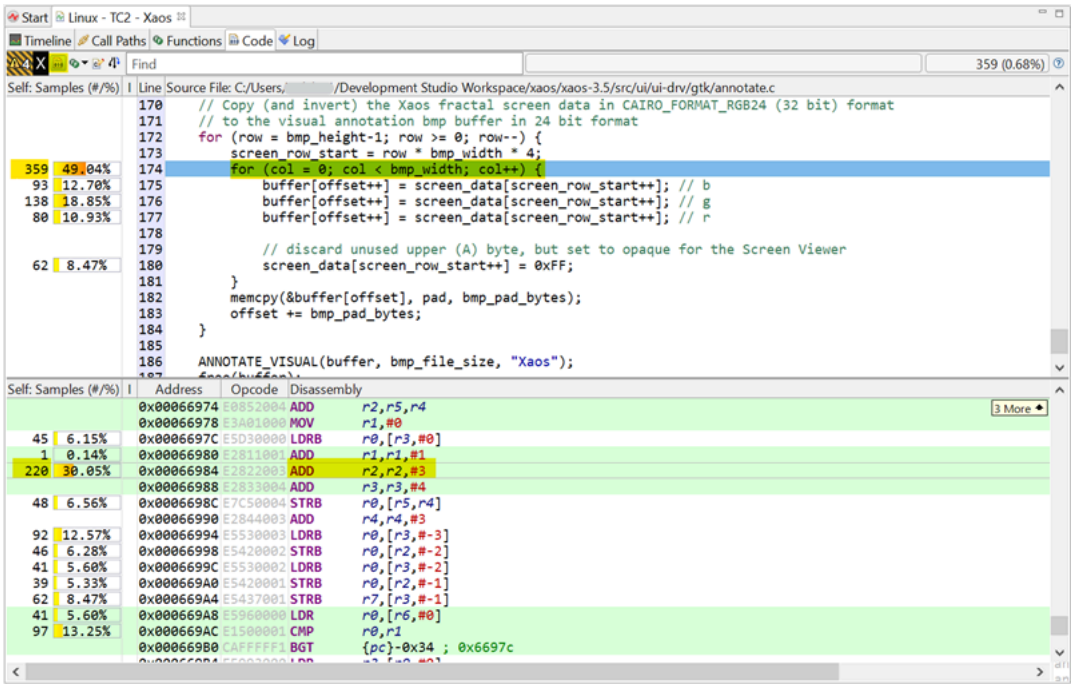
In the Code view, you see the code for the `VisualAnnotateImage` function, which is included in `annotate.c`. Select the highlighted icon in the following image to also show the assembly code:

**Figure 7-4: Code view tab**




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.

**Figure 7-5: Highlight of the for loop on line 174 in code view**



# 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.