SONY

AITRIOS

Vision and Sensing Application SDK Post Vision App Functional Specifications

Version 0.2.0

2023 - 1 - 30

TOC

1. Change history

Date	What/Why
2022/11/16	Initial draft
2023/01/30	Added post-processing debugging. Directory structure change. Updated the PDF build environment.

2. Terms/Abbreviations

Terms/Abbreviations	Meaning
Post Vision App	Post-processing (processing of Output Tensor, which is the output of the AI model)
PPL	A module that processes the output of the AI model(Output Tensor) of edge AI devices
Wasm	WebAssembly. Binary instruction format for virtual machines
FlatBuffers	Serialization library
WAMR-IDE	An integrated development environment that supports running and debugging WebAssembly applications
PPL parameter	Parameters used to initialize the PPL library
LLDB	Software debugger

3. Reference materials

- Reference/Related documents
  - WebAssembly
    - https://webassembly.org/
  - PPL Library API Specification for AITRIOS™ IMX500
    - https://developer.aitrios.sony-semicon.com/development-guides/documents/specifications/
  - FlatBuffers
    - https://google.github.io/flatbuffers/
  - WebAssembly Micro Runtime(WAMR)
    - https://github.com/bytecodealliance/wasm-micro-runtime/
  - LLDB
    - https://lldb.llvm.org/

4. Expected use case

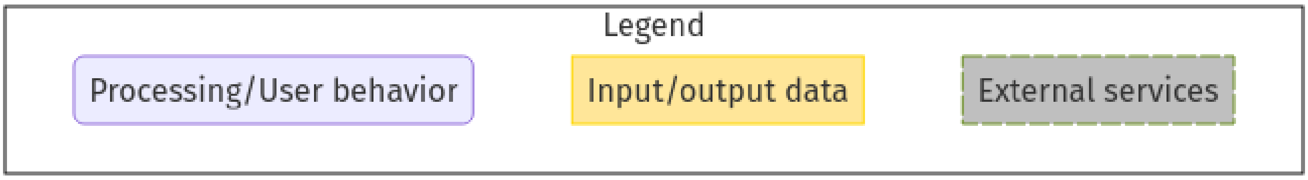
- You want to design and implement post-processing
- You want to convert post-processing code into a form that can be deployed to edge AI devices
- You want to debug post-processing code before deploying it to edge AI devices
- You want to debug post-processing code deployed on edge AI devices

5. Functional overview/Algorithm

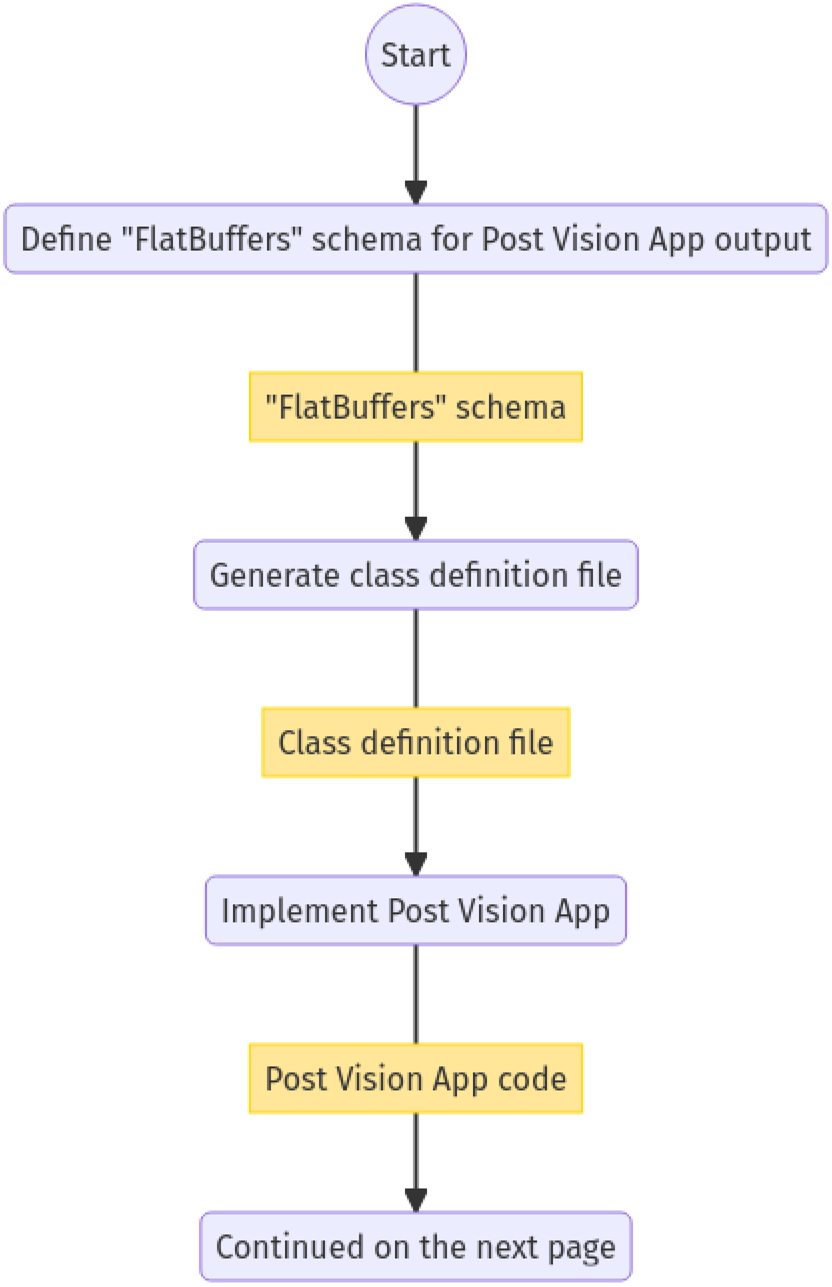
Functional overview

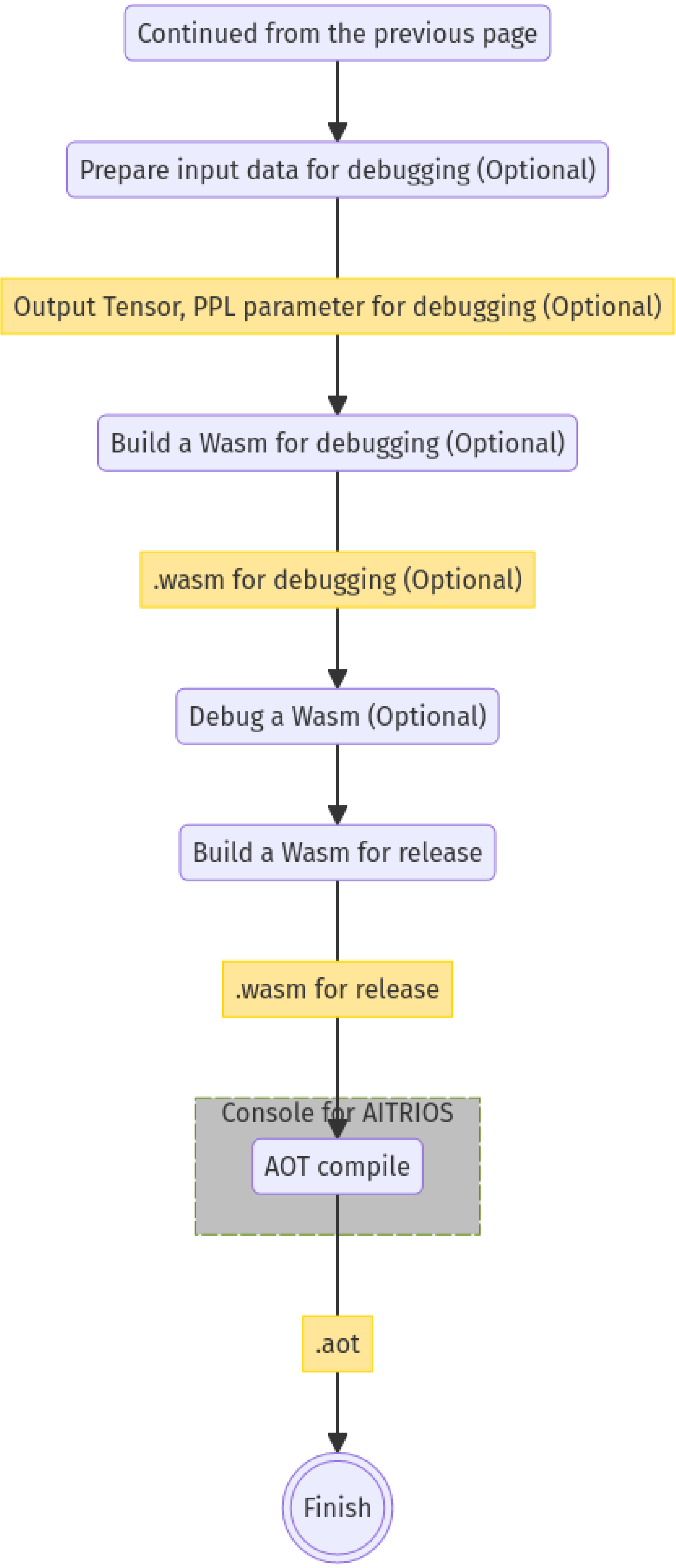
- Users can design and implement Post Vision App in C or C++
- Users can serialize Post Vision App output with "FlatBuffers"
  - Users can generate class definition files from "FlatBuffers" schema files
- Users can build a Post Vision App implemented in C or C++ to a Wasm
- Users can take advantage of the Post Vision App sample environment, including code for debugging
- Users can build a Wasm with Post Vision App sample code
- Users can build a Post Vision App to a Wasm for debugging, and debug on SDK environment using test app

Post Vision App creation flow



Flow


Start
Define "FlatBuffers" schema for Post Vision App output
"FlatBuffers" schema
Generate class definition file
Class definition file
Implement Post Vision App
Post Vision App code
Continued on the next page


Continued from the previous page
Prepare input data for debugging (Optional)
Output Tensor, PPL parameter for debugging (Optional)
Build a Wasm for debugging (Optional)
.wasm for debugging (Optional)
Debug a Wasm (Optional)
Build a Wasm for release
.wasm for release
Console for AITRIOS
AOT compile
.aot
Finish

Wasm files created in the SDK environment are AOT compiled in Console for AITRIOS and converted into a form that can be deployed to edge AI devices. (You can’t do that in a debug build)

Build features

Provides the following build features:

- Builds a Wasm for release
  Generates a Wasm file (.wasm) for deployment to edge AI devices
  - Generates a Wasm file (.wasm) from Post Vision App code (.c, or .cpp)
    - Object files (.o) are generated as intermediate files during the Wasm build process
- Builds a Wasm for debugging
  Generates a Wasm file (.wasm) to debug code before deploying to edge AI devices
  - Generates a Wasm file (.wasm) from Post Vision App code (.c, or .cpp)
    - Object files (.o) are generated as intermediate files during the Wasm build process

Debugging features

Debugging feature using test app

- The following Wasm debugging features are available through the LLDB and WAMR-IDE libraries and VS Code UI:
  - Specify breakpoint
  - Step execution (Step In, Step Out, Step Over)
  - Specify watch expression
  - Check variable
  - Check call stack
  - Check logs on terminal
- Provides a test app as a driver to invoke the processing of Wasm files
  - You can specify parameters to input into a Wasm, such as Output Tensor, PPL parameter, when running the test app

Does not support project management feature of WAMR-IDE

What you can do with sample code for debugging

Provides sample code for debugging.
Sample code for debugging stores debugging information in Post Vision App output.
Stores debugging information where Post Vision App stores its processed inferences.

By using this feature, you can get error information for unexpected behavior during running inference. For example, when you deploy a Post Vision App to an edge AI device and the inference results are not available.

- Example of running inference in Console for AITRIOS

The following is an example of inference results:

```
{
  "DeviceID": "xxxxx",
  "ModelID": "xxxxx",
  "Image": true,
  "Inferences": [
    {
      "T": "xxxxx",
      "O": "AQAAAA=="
    }
  ],
  "id": "xxxxx",
  "_rid": "xxxxx",
  "_self": "xxxxx",
  "_etag": "xxxxx",
  "_attachments": "xxxxx",
  "_ts": 0
}
```

Debugging information is stored in `"O"` of `"Inferences"` instead of inference results.
The `"AQAAAA=="` in `"O"` is Base64 encoded.

6. User interface specifications

How to start each function

1. Launch the SDK environment and preview the `README.md` in the top directory
2. Jump to the `README.md` in the `tutorials` directory from the hyperlink in the SDK environment top directory
3. Jump to the `4_prepare_application` directory from the hyperlink in the `README.md` in the `tutorials` directory
4. Jump to each feature from each file in the `4_prepare_application` directory

Design and implement a Post Vision App

1. Follow the procedures in the `README.md` to create the "FlatBuffers" schema file for Post Vision App output
2. Follow the procedures in the `README.md` to open a terminal from the VS Code UI and run the command to generate a header file of class definitions from a schema file
   - Class definition header file is generated on the same level as the schema file
3. Implement a Post Vision App
   - Implement in C or C++
   - Implement source files either by creating a new one or modifying the provided sample code for the Post Vision App
   - Implement using the class definition file generated by the "2."
   - Implement *Post Vision App interface* using the Post Vision App’s sample code
   - You can optionally install the OSS and external libraries needed to design your Post Vision App and incorporate them into your Post Vision App

> ℹ This SDK does not guarantee the installation or use of OSS or external libraries, which users may use at their discretion.

Generate a Wasm file for debugging from Post Vision App code

> ℹ Follow this procedure only when using the debugging feature.

1. Follow the procedures in the `README.md` to modify the Makefile for the file location and filename of the Post Vision App code
2. Follow the procedures in the `README.md` to open a terminal from the VS Code UI and run the command to build a Wasm for debugging
    - A Docker image is created for the debug environment, including a Wasm build for debugging, on the Dev Container, and a debug directory is created on the same level as the `Makefile`, and the .wasm file is stored in that directory

Edit input parameters for debugging a Wasm file

> Follow this procedure only when using the debugging feature.

1. Follow the procedures in the `README.md` to modify the input parameters, such ad Output Tensor, PPL parameter, for test

Debug a Wasm file

> Follow this procedure only when using the debugging feature.

1. Follow the procedures in the `README.md` to debug and check the logs in the terminal of VS Code UI, or open the Wasm source code in VS Code UI and specify breakpoint to check stack etc.

Generate a Wasm file from Post Vision App code

1. Follow the procedures in the `README.md` to modify the Makefile for the file location and filename of the Post Vision App code
2. Follow the procedures in the `README.md` to open a terminal from the VS Code UI and run the command to remove build a Wasm
    - A Docker image for the environment to build a Wasm are created on the Dev Container, and a release directory is created on the same level as the `Makefile`, and the .wasm file is stored in that directory

Remove build generation files

1. Follow the procedures in the `README.md` to open a terminal from the VS Code UI and run the command to remove build generation files
    - All files generated by the Wasm build (object files, Wasm files) are removed from the Dev Container. See *Builds a Wasm for release* and *Builds a Wasm for debugging* for builds.

Remove build generation files and the Docker image for environment to build a Wasm

1. Follow the procedures in the `README.md` to open a terminal from the VS Code UI, and run the command to remove build generation files and the Docker image for environment to build a Wasm
    - All files generated by the Wasm build (object files, Wasm files) and Docker images for the Wasm build environment are removed from the Dev Container. See *Builds a Wasm for release* and *Builds a Wasm for debugging* for builds.

The following error will be returned if the command to remove files generated by Wasm build is given with arguments other than those listed in the `README.md`:

```
ERROR: '<argument>' is unexpected argument.
Please see the document.
```

7. Post Vision App interface

When you design a Post Vision App, you need to implement a set of functions in the Post Vision App interface. Example implementations of these functions are provided in the sample code. See PPL Library API Specification for AITRIOS™ IMX500 in the separate document for details.

8. Target performances/Impact on performances

- Usability
    - When the SDK environment is built, users can generate class definition file for "FlatBuffers", build a Wasm, and debug a Wasm without any additional installation steps

9. Assumption/Restriction

- Supports only Post Vision App code implemented in C or C++ for Wasm builds

10. Remarks

- Check the following version information for the tools needed to develop Post Vision App that comes with the SDK
    - "FlatBuffers": Described in the `README.md` in the `4_prepare_application` directory
    - Other tools: Described in the `Dockerfile` in the `4_prepare_application` directory

11. Unconfirmed items

- None