Arm® Ethos™-U NPU

Version 5.0

Application development overview

arm

101888_0500_05_en

Arm® Ethos™-U NPU

Application development overview

Release Information

Document History

Issue	Date	Confidentiality	Change
0100-01	27 January 2020	Confidential	First release of version 1.0.
0200-02	27 March 2020	Confidential	First release of version 2.0.
0300-03	15 May 2020	Non-Confidential	First release of version 3.0.
0400-04	24 June 2020	Confidential	First release of version 4.0.
0500-05	23 October 2020	Non-Confidential	First release of version 5.0.

Non-Confidential Proprietary Notice

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

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.

Web Address

developer.arm.com

Contents

Arm® Ethos™-U NPU Application development overview

Preface

This preface introduces the *Arm® Ethos™-U NPU Application development overview*.

It contains the following:

About this book

This manual gives an overview of the flow of data between an application and the Arm® Ethos™-U NPU.

Intended audience

This manual is written for machine learning application developers who want to run their applications on the Ethos™-U NPU.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction

This chapter gives an overview of the Ethos-U Neural Processing Unit (NPU) and describes how this is incorporated into an embedded system.

Chapter 2 NPU software overview

This chapter gives an overview of the software components of the Ethos-U NPU. The chapter covers how the different components work together to send data to and from the Ethos-U NPU.

Chapter 3 Getting started

This chapter describes how to obtain the software components to begin Ethos-U NPU software development.

Appendix A Revisions

This appendix describes the technical changes between releases of this book.

Glossary

The Arm® Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the *Arm® Glossary* for more information.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace`

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace`

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`

Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`

Denotes language keywords when used outside example code.

SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional reading

This book contains information that is specific to this product. See the following documents for other relevant information.

Arm publications

Developer resources:

https://developer.arm.com/solutions/machine-learning-on-arm

Other publications

None.

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:

- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

If you have comments on content then send an e-mail to *errata@arm.com*. Give:

- The title *Arm Ethos-U NPU Application development overview*.
- The number 101888_0500_05_en.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Chapter 1
Introduction

This chapter gives an overview of the Ethos-U Neural Processing Unit (NPU) and describes how this is incorporated into an embedded system.

It contains the following sections:

1.1 Design configurability

The Ethos-U NPU is a small and power-efficient processor that is used to reduce both the inference time and memory requirements needed to run Machine Learning (ML) Neural Networks (NN).

The Ethos-U NPU is attached to a Cortex®-M series Central Processing Unit (CPU), which can be incorporated into an embedded system and connected to embedded or external memory using two Arm AMBA 5 AXI interfaces, M0 and M1.

- To optimize performance of the Ethos-U NPU, the AXI interface M0 should be connected to a high-speed, low-latency memory, such as SRAM. The memory is used for dynamic storage of runtime data during the inference of the neural network.
- The AXI interface M1 is used for memory transactions that tolerate lower bandwidth and higher latency. The AXI M1 interface can therefore be connected to memory that is slower or less burst efficient, for example flash or DRAM. The memory is used for the non-volatile storage of the runtime software stack (including the User Application) and the neural network definition (including weights).
- For the Ethos-U55 NPU, the AXI interface M1 is read-only. For the Ethos-U65 NPU, the AXI interface M1 is read/write.

The M0 and M1 ports typically connect to an interconnect, which allows the M0 and M1 AXI interfaces to access any memory. The Vela compiler schedules high bandwidth, low-latency memory transactions on the AXI interface M0, and all other transactions on the AXI interface M1.

The Ethos-U55 NPU and Ethos-U65 NPU are configurable to meet various performance points as outlined in the following tables.

Table 1-1 Ethos-U55 NPU configuration options

Configuration	Number of MACs per cycle	Internal memory	Performance @500MHz
256	256	48KB	256 GOP
128	128	24KB	128 GOP
64	64	16KB	64 GOP
32	32	16KB	32 GOP

Table 1-2 Ethos-U65 NPU configuration options

Configuration	Number of MACs per cycle	Internal memory	Performance @1GHz
512	512	96KB	1 TOP
256	256	48KB	512 GOP

1.2 Ethos-U system

The Ethos-U system is paired with a Cortex-M CPU. The system is highly configurable and can be built in many different ways.

The following figure shows a typical Ethos-U system.

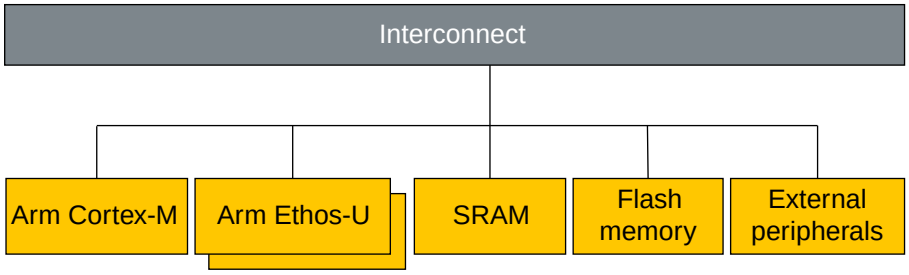

Figure 1-1 Ethos-U system

Cortex-M

The Cortex-M series CPU is the application processor that controls one, or multiple Ethos-U NPUs. You can specify your preferred Cortex-M series CPU, but the recommended CPUs are:

- Cortex-M4
- Cortex-M7
- Cortex-M33
- Cortex-M55

You also have access to the source code meaning you can use the Cortex-M series CPU for tasks other than machine learning.

Ethos-U NPU

Either an Ethos-U55 NPU or an Ethos-U65 NPU can be paired with the Cortex-M CPU, but the Ethos-U65 NPU has been designed to optimize data transfer between the slower memory and the fast memory cache.

SRAM

The input feature map (IFM) data and the output feature map (OFM) data is stored in SRAM. You can specify your preferred amount of SRAM, but optimal performance is obtained when the network is placed fully in SRAM. If the network cannot be placed fully in SRAM, only the temporary data is stored in SRAM.

Flash memory

The weights and biases are stored in flash memory, DRAM, or SRAM.

External peripherals

Controllers for external peripherals, such as a microphone or camera, can be added.

1.3 Ethos-U subsystem

The Ethos-U subsystem can connect to a Linux host and various other operating systems.

The following figure shows a typical Ethos-U subsystem.

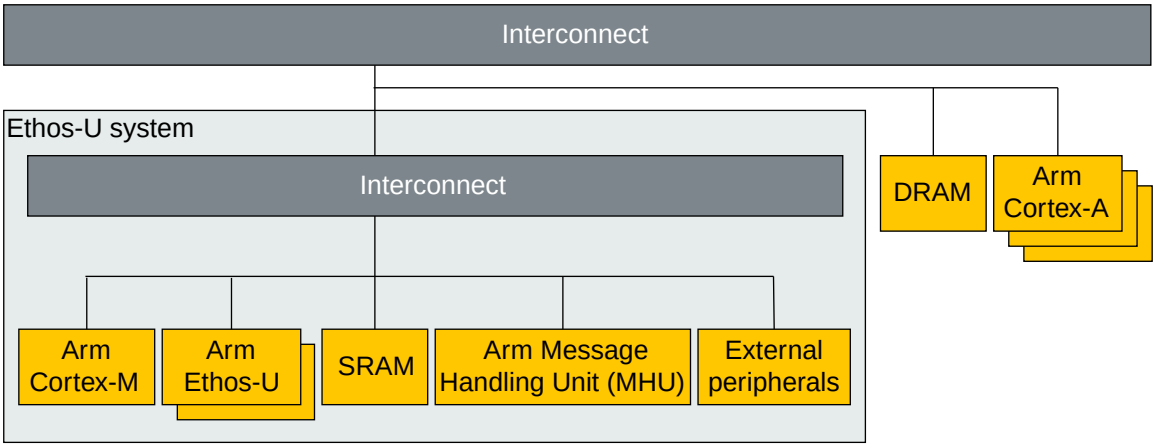

Figure 1-2 Ethos-U subsystem

Ethos-U NPU

Either an Ethos-U55 NPU or an Ethos-U65 NPU can be paired with the Cortex-M CPU. The Ethos-U65 NPU has been designed to optimize data transfer between the slower memory and the fast memory cache and is therefore the recommended NPU.

Message Handling Unit

Any type of mailbox, similar to the Arm Message Handling Unit (MHU), can be used.

Note

For an example of using an MHU, see the *Arm CoreLink SSE-200 Subsystem for Embedded Technical Reference Manual* at *https://developer.arm.com/documentation/101104/0200*.

DRAM

Weights, biases, and the input feature map (IFM) and the output feature map (OFM) data are stored in slower, high latency memory like DRAM.

Cortex-A

The Cortex-A series CPU only communicates with the Cortex-M series CPU. The Cortex-A series CPU has no direct contact with the Ethos-U NPU. Communication between the CPUs is based on a memory interface in DRAM and the MHU doorbell.

Chapter 2
NPU software overview

This chapter gives an overview of the software components of the Ethos-U NPU. The chapter covers how the different components work together to send data to and from the Ethos-U NPU.

It also provides a description of the software tooling and the runtime software stack that is used by an embedded application which runs on the Ethos-U NPU.

It contains the following sections:

2.1 NPU software components

The Ethos-U NPU software comprises both offline software tooling and the online Cortex-M runtime software stack.

Vela software tooling is used to compile (optimize and convert) a User Application's NN model. The input model is a fully trained and quantized TensorFlow Lite for Microcontrollers (TFLμ) model. The output is an optimized model that is able to run optimally on an Ethos-U NPU embedded system. All software tooling is done on a desktop PC or similar device.

The runtime software stack provides all the software that will execute on the Cortex-M series CPU. This includes the User Application, which uses the TFLμ library to execute parts of the optimized model (command stream) on the Ethos-U NPU.

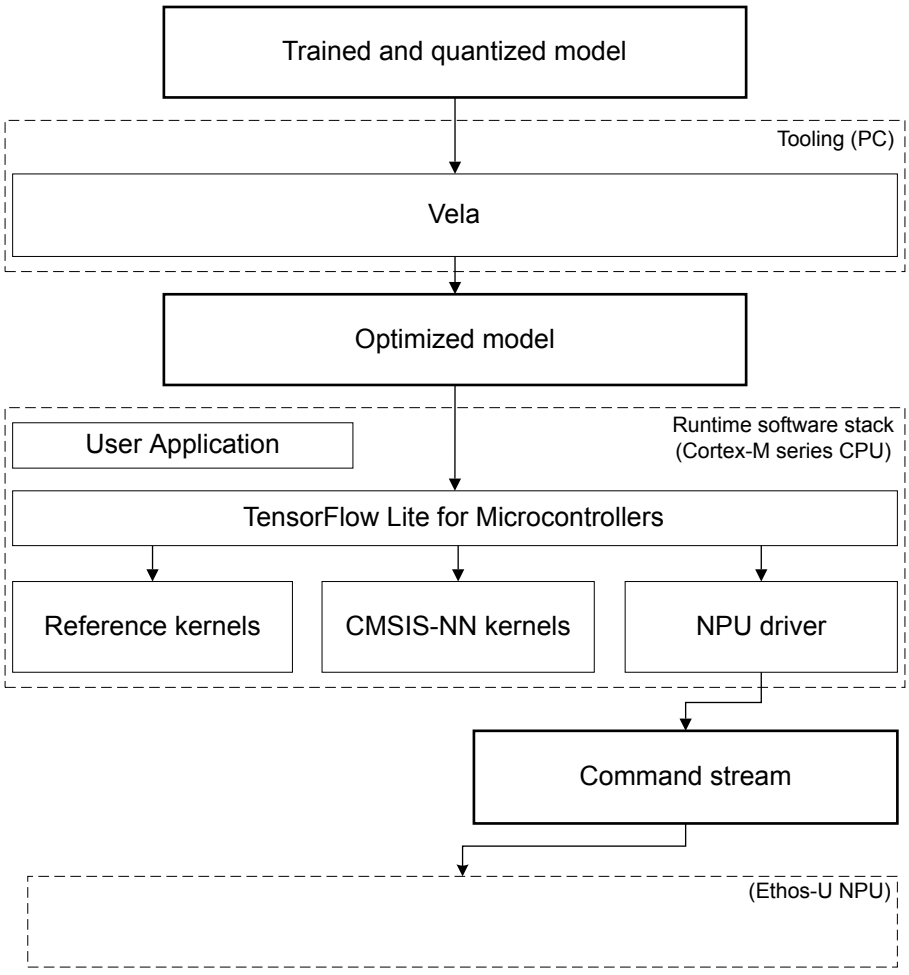

Figure 2-1 NPU software overview

2.2 NPU software tooling

Software tooling consists of the following optimization tools, all of which run in a PC environment.

This section contains the following subsection:

2.2.1 The Vela compiler

This tool is used to compile a TFLμ model into an optimized version that can run on the Ethos-U NPU.

The optimized model contains TensorFlow Lite custom operators (supported operators) for those parts of the model that can be accelerated by the Ethos-U NPU. Parts of the model that cannot be accelerated are left unchanged and will instead run on the Cortex-M series CPU using an appropriate kernel.

Vela trials a number of different compilation strategies and applies a cost function to each one. It then chooses the optimal execution schedule for each supported operator or group of operators.

Memory optimization

The Vela compiler also performs various memory optimizations to reduce both the permanent (for example flash) and runtime (for example SRAM) memory requirements.

One such technique for permanent storage is the compression of all the weights in the model.

Another technique is cascading, which addresses the runtime memory usage. Cascading reduces the maximum memory requirement by splitting the feature maps (FM) of a group of consecutively supported operators into stripes. A stripe can be either the full or partial width of the FM. And it can be the full or partial height of the FM. Each stripe in turn is then run through all the operators in the group.

The parts of the model that can be optimized and accelerated are grouped and converted into TensorFlow Lite custom operators. The operators are then compiled into a command stream that can be executed by the Ethos-U NPU.

Finally, the optimized model is written out as a TFLμ model and a Performance Estimation report is generated that provides statistics, such as memory usage and inference time.

The compiler includes numerous configuration options that allow you to specify various aspects of the embedded system configuration (for example the Ethos-U NPU configuration, memory types, and memory sizes). There are also options to control the types of optimization that are performed during the compilation process.

2.3 NPU runtime software stack

The runtime software stack consists of components that interact with each other in specific ways.

User Application

The User Application runs the required functions and makes calls to the TensorFlow Lite for Microcontrollers (TFLμ) library when it performs an inference of the model.

TensorFlow Lite for Microcontrollers

The TFLμ framework is compiled into a C++ library that contains a copy of the optimized model along with versions of Reference and CMSIS-NN kernels. This library is then used by the User Application to perform inferences.

During an inference, the model is parsed one operator at a time and the corresponding kernels are executed. The exception to this is when it encounters a TensorFlow Lite Custom operator. In this case, the library sends the operator and associated tensor data to the Ethos-U NPU driver instead.

Reference kernels

Contains a set of kernels for all operators in the TFLμ framework.

CMSIS-NN

The CMSIS-NN contains highly optimized and performant kernels that accelerate a subset of operators in the TFLμ framework.

NPU driver

The Ethos-U NPU driver handles the communication between the TFLμ framework and the Ethos-U NPU to process a custom operator. When the Ethos-U NPU has completed its processing, it signals back to the driver, which in turn informs the TFLμ library.

2.4 Linux driver stack

The Linux driver stack is provided as an example of how a rich operating system like Linux can dispatch inferences to an Ethos-U subsystem.

The source code is provided. In accordance with the license, you can modify and further develop the source code.

The following figure describes the software components that are required for Linux to dispatch networks and inferences to the Cortex-M series CPU.

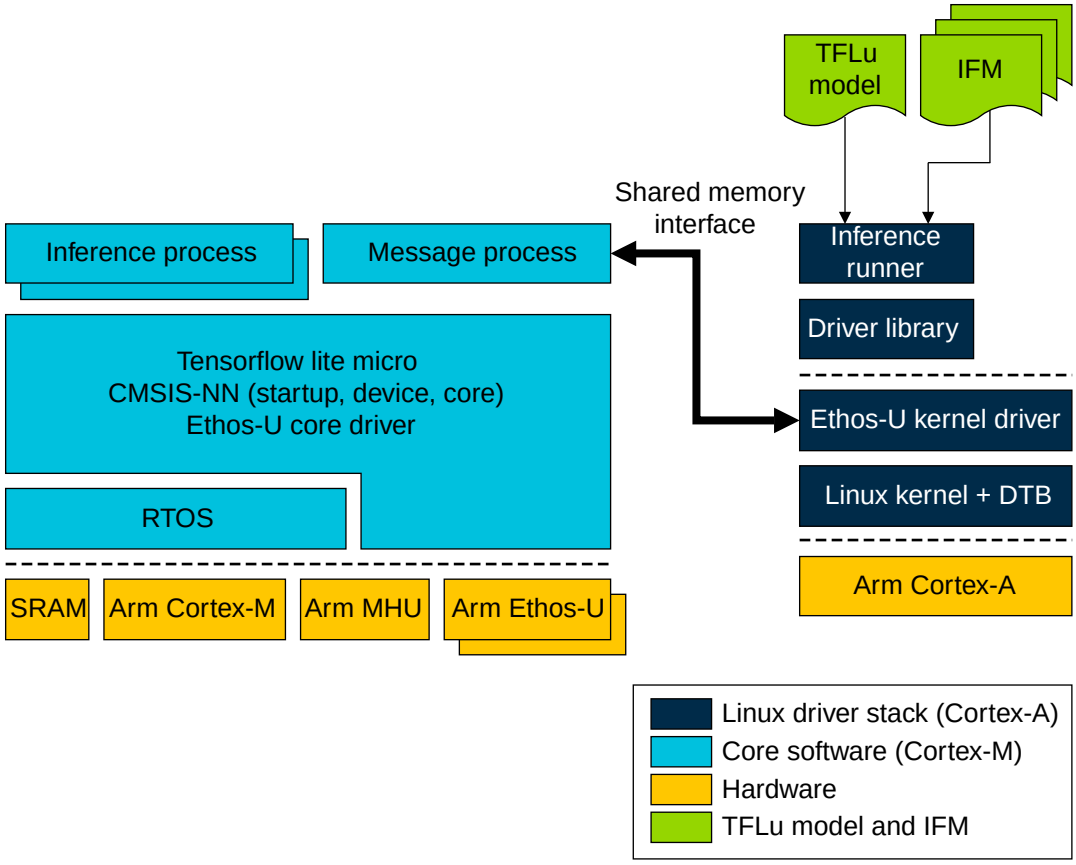

Figure 2-2 Driver stack

Inference runner

The inference runner is a test application that runs inferences on the Ethos-U driver stack. The inference runner inputs a TFLμ model that was optimized by Vela and an input file containing input feature map (IFM) data. The output from the inference runner is an output feature map (OFM) file.

Driver library

The driver library is a thin C++ interface around the kernel-user API (UAPI) header file that the kernel driver exports to user space. The driver library enables user space applications to detect NPU capabilities, create buffers, register networks, and run inferences.

Kernel driver

The kernel driver is the bridge between user space and the Ethos-U subsystem. It presents a UAPI that allows a user space application to run inferences. The inference request from user space is forwarded to the Ethos-U subsystem that runs the inference.

Linux kernel and DTB

Any vanilla Linux kernel can be used. The Debug and Trace Bus (DTB) entry for the Ethos-U subsystem is documented in the kernel driver.

2.5 Use cases

There are two major use cases for the subsystem.

Linux dispatches inferences

1. Linux allocates DRAM memory for the network, input feature map (IFM), and the output feature map (OFM).
2. An inference request is sent from Linux to the Ethos-U subsystem.
3. The Ethos-U subsystem executes inference and returns an inference response.

This use case is implemented and has been verified.

Ethos™-U running without Linux

1. The Ethos-U subsystem is capturing IFMs and running inferences without the help of Linux. Linux is busy, in sleep mode, or even powered down.
2. The Ethos-U subsystem captures an IFM (audio, video, or sensor data) and runs inference.
3. When the Ethos-U subsystem detects something of interest, Linux is notified.

A possible use for this use case would be an AI speaker scanning audio for a particular word or a camera scanning faces to trigger an unlock event.

This use case is not implemented in the Linux driver stack. You would have to implement this use case.

Chapter 3
Getting started

This chapter describes how to obtain the software components to begin Ethos-U NPU software development.

It contains the following section:

3.1 Accessing software components

Arm makes use of open-source components to allow you to develop the Ethos-U NPU software.

To access the Ethos-U NPU open-source software, tools, documentation, and additional instructions, go to *https://review.mlplatform.org/plugins/gitiles/ml/ethos-u/ethos-u*.

Appendix A
Revisions

This appendix describes the technical changes between releases of this book.

It contains the following section:

A.1 Revisions

This appendix describes the technical changes between releases of this book.

Table A-1 First release of version 1.0

Change	Location	Affects
First development release	-	-

Table A-2 First release of version 2.0

Change	Location	Affects
Clarified offline and runtime handling and optimizing of models.	*2.1 NPU software components* on page 2-14	All
Changed versioning to reflect standard software releases.	-	-

Table A-3 First release of version 3.0

Change	Location	Affects
Added information about the AXI interfaces.	*1.1 Design configurability* on page 1-10	All
Added the getting started chapter.	*Chapter 3 Getting started* on page 3-20	All

Table A-4 First release of version 4.0

Change	Location	Affects
Added Ethos-U65 NPU information.	*1.1 Design configurability* on page 1-10	All

Table A-5 First release of version 5.0

Change	Location	Affects
Updated the M0 and M1 memory information. Updated the Ethos-U65 NPU configuration information.	*1.1 Design configurability* on page 1-10	All
Added new sections.	*1.2 Ethos-U system* on page 1-11	All
	1.3 Ethos-U subsystem on page 1-12	
	2.4 Linux driver stack on page 2-17	
	2.5 Use cases on page 2-19	