Arm® Ethos™-U85 NPU

Revision r0p0

Technical Overview

Non-Confidential

Issue 02
102684_0000_02_en

Arm® Ethos™-U85 NPU Technical Overview

This document (102684_0000_02_en) was issued on 2024-06-05. There might be a later issue at http://developer.arm.com/documentation/102684

The product revision is r0p0.

See also: Proprietary Notice | Product and document information | Useful resources

Start Reading

If you prefer, you can skip to the start of the content.

Intended audience

This manual is for system designers, system integrators, and verification engineers who are designing a *System on Chip* (SoC) device that uses an Arm Ethos™-U85 NPU.

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.

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.

Contents

1. Description of the Arm® Ethos™-U85 Neural Processing Unit

The *Neural Processing Unit* (NPU) improves the inference performance of neural networks. The NPU is compliant with the *Tensor Operator Set Architecture* (TOSA) version 0.80.0 Base-Inference profile, level 8K with the exclusion of custom operators.

Note

For more information on the TOSA Base-Inference profile, see the TOSA specification.

The NPU accelerates, without host microcontroller or application processor fallback, any neural network that software tooling can lower to the TOSA version 0.80.0 level 8K. For more information on the software tooling the NPU requires, see 1.1 Supported application programming interfaces on page 7. For instance, the neural networks the NPU can accelerate includes the following 8-bit and 16-bit integer quantized networks:

- Transformer networks. The NPU targets networks such as the BERT_tiny and DEIT_Tiny networks.
- *Convolutional Neural Networks* (CNN)
- *Recurrent Neural Networks* (RNN)

Arm delivers the hardware *Register Transfer Level* (RTL) of the NPU with an open-source driver and compiler. A neural network must be compiled using the open-source compiler to produce a command stream. The application invokes the driver, which communicates with the NPU to tell it where the command stream is and initiates the network traversal. The command stream describes the steps necessary for the NPU to execute the operators compiled into the command stream autonomously. When complete, the NPU raises an IRQ to the host microcontroller or application processor.

The host microcontroller or application processor programs the memory location of the command stream and other payloads into registers in the NPU. The *Central Control* (CC) processes the command stream and executes the operations on different units in the NPU.

The NPU includes a *Direct Memory Access* (DMA) controller that can read from and write to on-chip SRAM and external memories such as DRAM and Flash. When the NPU performs inferences, the DMA controller reads the neural network description. This description contains:

- The command stream
- Network weights
- Bias information
- Scale information

The DMA controller also transfers the *Input Feature Maps* (IFMs), the *Output Feature Maps* (OFMs), and NPU-private intermediate data that is also held in system memory.

The external interfaces that the NPU implements are:

- Two kinds of Arm® AMBA® 5 AXI managers:
  - The AXI_SRAM for targeting on-chip SRAM.
  - The AXI_EXT for targeting DRAM or flash memory.

Note

The number of Arm® AMBA® 5 AXI manager interfaces depend on the NPU configuration.

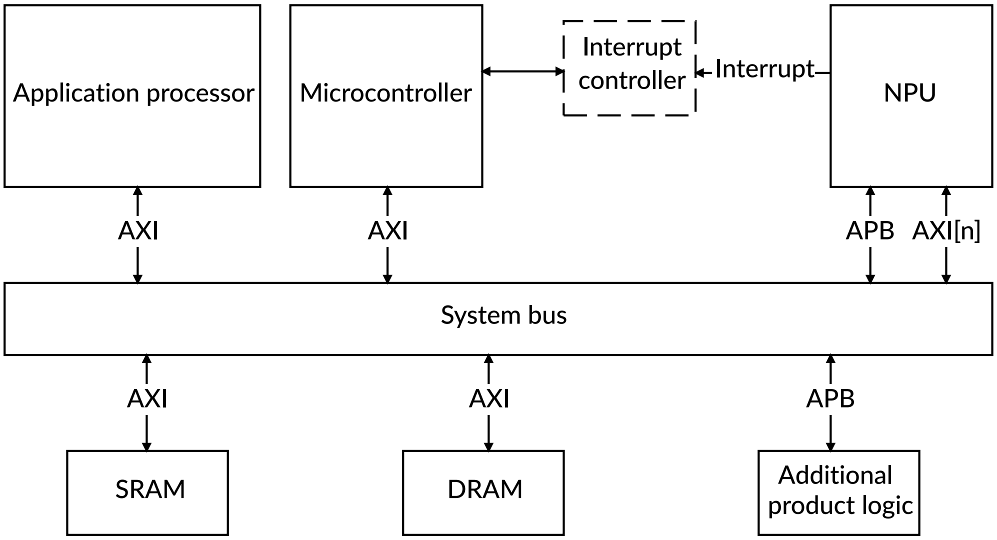
Note

The manager interfaces are also AMBA® 4 AXI compatible.

- An Arm® AMBA® 5 APB Completer interface with clock enable and wake-up signaling that allows the host microcontroller to program the NPU.

The following figure shows a typical system configuration block diagram for the NPU.

Figure 1-1: Typical system configuration block diagram



Note

There is more than one AXI interface between the system bus and the NPU.

1.1 Supported application programming interfaces

To program, test, and monitor the NPU, Arm deploys the open-source *TensorFlow Lite for Microcontrollers* (TFLµ) runtime, which runs on an external host microcontroller. The test program uses the Vela compiler offline to compile and optimize the neural network graph for the NPU. The Vela compiler generates a command stream and the TFLµ runtime sends the command stream to the NPU to process.

The compiler selects the TOSA compatible operators from the network graph that can be optimized and executed on the NPU. The NPU drivers manage the workloads that execute inferences on the NPU.

The NPU is optimized for running neural operations and supports a wide variety of neural operations. This optimization allows higher energy efficiency for running a neural workload when compared to running the same workload on the host microcontroller.

Neural networks models are typically developed in a framework such as TensorFlow. For deployment, the network must be quantized and compiled into a command stream suitable for executing on the NPU. A runtime executing on a host microcontroller schedules command streams on the NPU. Any TFLµ operators that do not map to the NPU, the external host microcontroller executes in software.

This manual does not cover the capabilities of software. See the following project for further information on related software: https://review.mlplatform.org/plugins/gitiles/ml/ethos-u/ethos-u.

1.2 Interfaces

The NPU has several external interfaces.

The external interfaces are:

- Arm® AMBA® 5 APB Completer with clock enable and wake-up signaling
- Two kinds of Arm® AMBA® 5 AXI AXI managers:
  - The AXI_SRAM for targeting system SRAM
  - The AXI_EXT for targeting DRAM or flash memory
- An interrupt
- Two Q-channel ports:
  - A Q-Channel for clock management
  - A Q-Channel for power management
- System configuration signals
- Clock
- Reset
- Warm reset interface
- MBIST interface
- Debug interface
- DFT interface

1.3 Documentation

The documentation provides instructions and reference material for configuration, implementation, and integration.

The documentation in the Ethos™-U85 product package includes:

Technical overview

The *Technical Overview* (TO) briefly describes the functionality of the processor.

Technical reference manual

The *Technical Reference Manual* (TRM) describes the full functionality and the effects of functional options on the behavior of the processor. It is required at all stages of the design flow. Design flow choices can mean that some behavior that the TRM describes is not relevant. If you are programming the processor, obtain additional information from:

- The implementer to determine the build configuration of the implementation.
- The integrator to determine the pin configuration of the device that you are using.

Configuration and integration manual

The *Configuration and Integration Manual* (CIM) describes:

- The available build configuration options and related issues in selecting them.
- How to configure the *Register Transfer Level* (RTL) source files with the build configuration options.
- How to integrate RAM arrays.
- How to run test vectors.
- The processes to sign off on the configured design.
- How to connect and test a device in a *System on Chip* (SoC) design or to other IP.

The CIM and TRM are confidential books only available to licensees.

1.4 Design process

The NPU is delivered as synthesizable RTL. Before it can be used in a product, it must go through the design process.

Implementation

The implementer configures and synthesizes the RTL to produce a hard macrocell.

Integration

The integrator connects the configured design into an SoC, including a memory system and peripherals.

Programming

The system programmer uses the following to develop the SoC:

- The software to configure and initialize the NPU.
- The application software and the SoC tests.

1.5 Product revisions

Successive product revisions have differences in functionality.

r0p0

First release.

2. Functional description of the Arm® Ethos™-U85 NPU

The NPU is composed of different individual subcomponents and handles the flow of data in specific ways.

The software stack manages the control and data flows between the application software running on an external host microcontroller and individual subcomponents of the NPU.

The NPU can also be set to different security and privilege modes.

The NPU consists of the following subcomponents which perform specific functions:

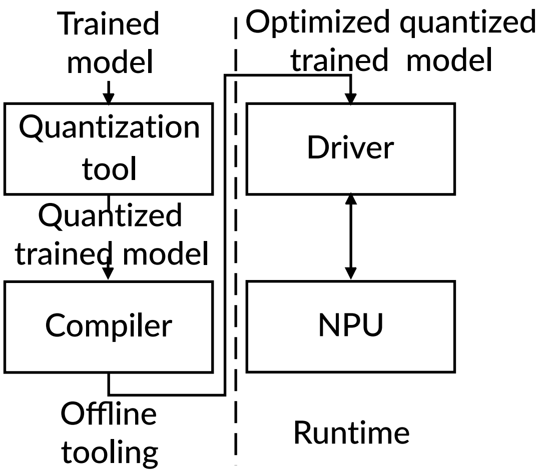
- *Central Control* (CC)
- DMA controller
- *Multiply-Accumulate* (MAC) unit
- *Weight Decoder* (WD)
- *Activation Output* (AO) unit

2.1 Control and data flow

The components of the software stack communicate with each other to handle the control and data flow between the neural network application and the NPU.

The following figure shows the software stack for the NPU.

Figure 2-1: The offline software stack



The NPU uses offline tools to optimize the code. At runtime, the microcontroller passes this optimized trained model to the NPU.

Note

Quantization is managed through the TensorFlow workflows and is not a specific component delivered with the Ethos™-U85 software. The compiler runs offline on the TFL flatbuffer and has knowledge about which operators the NPU supports.

The following steps describe the offline tooling flow:

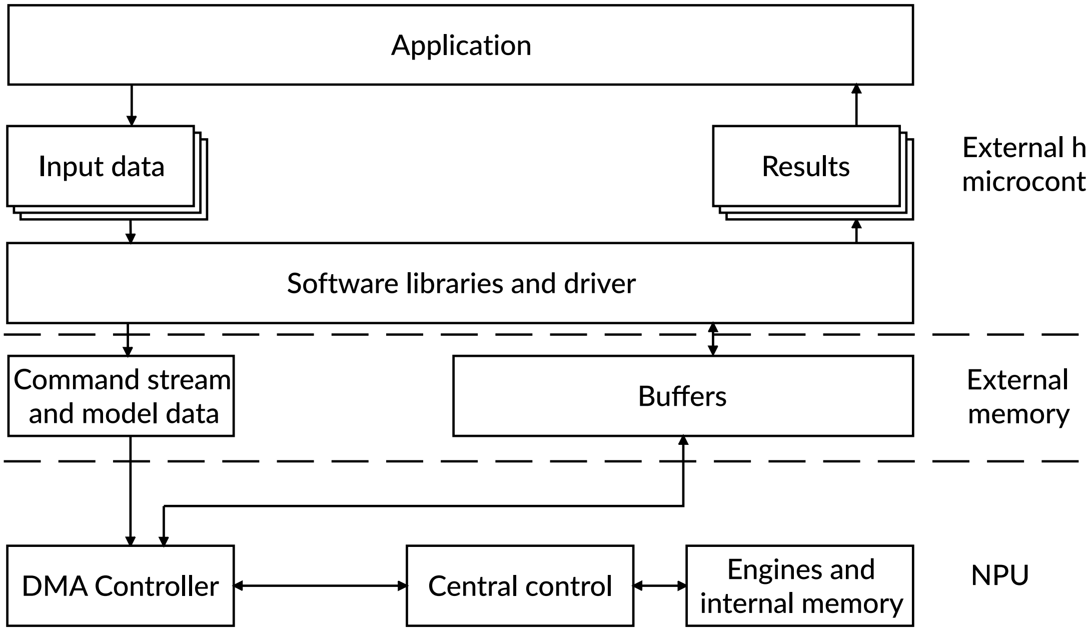
1. Pass your trained model through the quantization tool.
2. Pass the quantized model to the compiler. This tool optimizes the model for this NPU and outputs an optimized model that contains a command stream for the NPU.

The following steps describe the runtime control and data flow:

1. The optimized model is placed in system memory, which is accessible by the NPU.
2. Runtime software executes the model, dispatching operations to the NPU where there is a compiled command stream.
3. The NPU reads the optimized model and runs the command stream that is included in it. The microcontroller runs any parts that the NPU cannot execute.
4. When the inference is complete, the result is placed in the memory location that the driver specifies.

The following figure shows the control and data flow.

Figure 2-2: Control and data flow


Application
Input data
Results
External host microcontroller
Software libraries and driver
Command stream and model data
Buffers
External memory
DMA Controller
Central control
Engines and internal memory
NPU

2.1.1 Supported memory formats for feature maps

The NPU supports the industry-standard NHWC and NCHW formats of feature-map data.

The NPU supports the NCHW format by using internal transpose operations which convert the feature map data to either the NHWC or NHCWB16 format.

NHWC is used as an input and output format by the NPU for communication with TensorFlow lite.

When the NPU processes multiple layers, it reformats NHWC-formatted feature maps into an internal NHCWB16 format when reading in data. The NPU also performs the reverse transformation on the final output layer.

NHWC format

The NHWC format has the following properties:

- H (height), W (width), and C (channels) data.
- The size of each element (ElemSize) is 1-byte or 2-bytes.
- Only a single batch is supported (N=1).
- The address of an element y, x, c is `(BASE+y*STRIDE_Y+x*STRIDE_X+c*ElemSize)`.

- The values BASE, STRIDE_Y, and STRIDE_X must be aligned in element size.
- Tiles can be used.

NHCWB16 format

The NHCWB16 format has the following properties:

- A block format consisting of 16 channels per block.
- Only a single batch is supported (N=1).
- The address of an element y, x, c is (BASE+y*STRIDE_Y+(c/16)*STRIDE_C + (x*16 + (c %16))*ElemSize).
- The values BASE, STRIDE_Y, and STRIDE_C must be 16*ElemSize byte aligned, where ElemSize is the size of an element in bytes.
- Tiles can be used.

2.2 Security and boot flow

The NPU can be set to different security and privilege modes during a reset. The host microcontroller cannot reset the NPU to a higher security level than its current level.

At any reset, all registers and memories in the NPU are cleared to prevent leakage between states.

When a soft reset is requested, the NPU ensures that all AMBA® 5 AXI transactions complete before issuing the internal reset.

After powerup and release of hard reset, the NPU reads PORPL during the following clock period. PORPL then sets the NPU privilege level.

- LOW indicates user mode.
- HIGH indicates privileged mode.

After powerup and release of hard reset, the NPU reads PORSL during the following clock period. PORSL then sets the NPU security level.

- LOW indicates Secure mode.
- HIGH indicates Non-secure mode.

When the NPU is accessed, it uses the PPROT signal to check if the access is permitted. The NPU security and privilege level that is used on the AXI ports are the ARPROT/AWPROT signals. The ARPROT/AWPROT signals can be used for memory protection at system-level.

The following is a typical powerup procedure assuming QLPI is enabled:

1. The power controller detects a signal to power up the NPU. Typically, this signal is a PWRQACTIVE input to the power controller which has detected an APB access to the NPU. However, there can be other mechanisms to initiate power up.
2. The power controller applies power to NPU domain.

3. The power controller removes isolation on NPU outputs.
4. nRESET to the NPU is deasserted.
5. The power controller requests a move to Q_RUN state by deasserting PWRQREQn.
6. The NPU asserts CLKQACTIVE to indicate that clock is needed.
7. The clock controller initiates a move to Q_RUN state by raising CLKQREQn. After this point, the NPU starts clearing its internal RAMs.
8. The NPU completes the move to Q_RUN state by deasserting PWRQACCEPTn.

Note

The NPU assumes that the software on the host that has permission to access it is trusted. You must ensure that the system provides suitable protection from memory tampering. For example, by protecting the flash.

2.3 Functional blocks

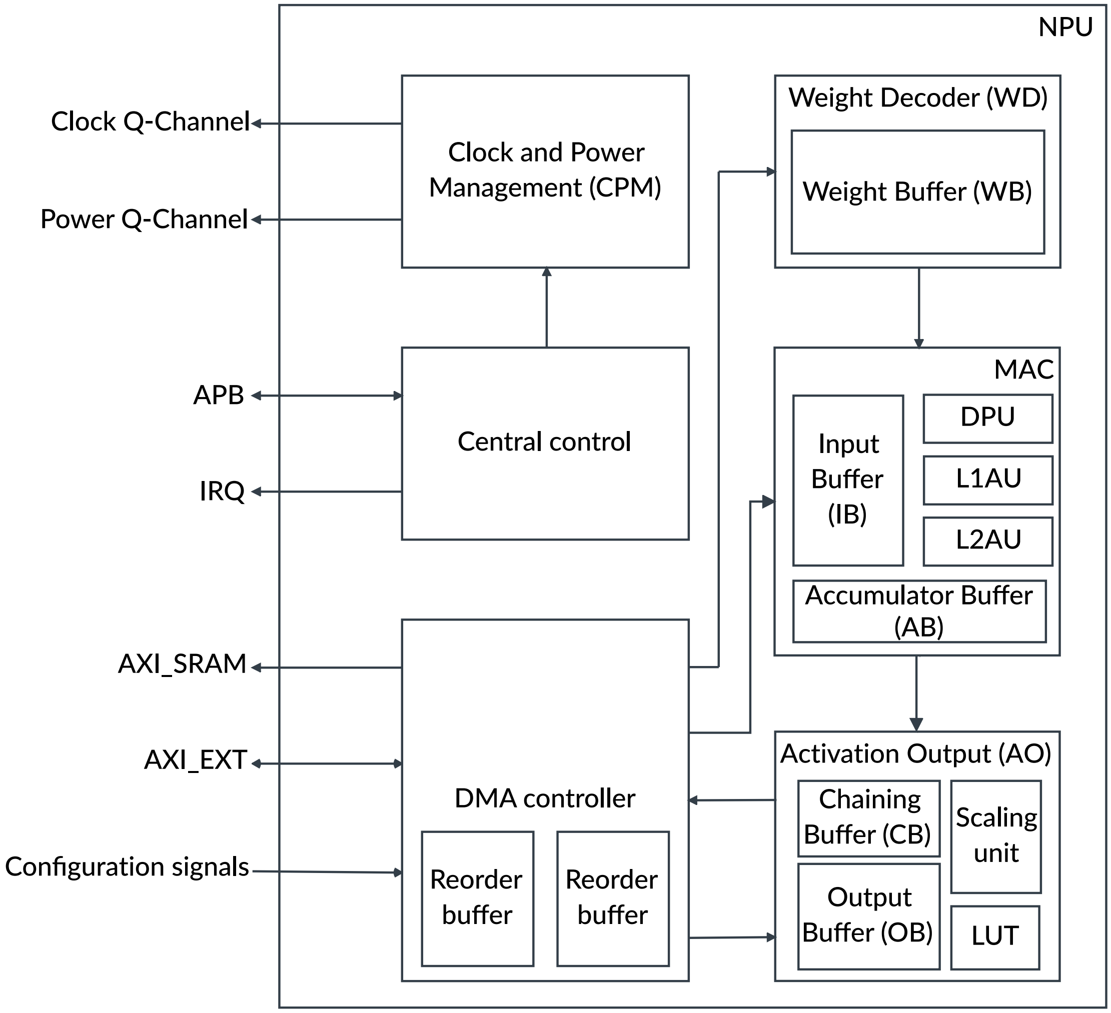
The NPU consists of the *Central Control* (CC), a DMA controller, a MAC unit, an output unit, and the *Weight Decoder* (WD).

The following are descriptions of the units of the NPU:

- The CC receives tasks from the external host microcontroller. The CC queues and dispatches units of work to the DMA and other units inside the NPU.
- The DMA controller uses its two kinds of Arm® AMBA® 5 AXI manager interfaces to move data between off-chip memory and on-chip SRAM and internal memory. The DMA also fetches data from external and on-chip memory to feed the MAC, WD and the output unit. It also writes out data to on-chip SRAM or external memory..
- The MAC unit has various internal units for reading IFMs from the input buffer of the NPU, performing dot products and accumulations.

The following figure shows the main components of the NPU.

Figure 2-3: Functional blocks diagram



2.3.1 External interfaces

The NPU uses two kinds of AMBA® 5 AXI manager interfaces, an AMBA® 5 APB completer interface with wake-up signaling, an interrupt interface, two Q-Channel interfaces, clock, and reset to enable access to and from external components.

Two kinds of AMBA® 5 AXI manager interfaces

The two kinds of interfaces are AXI_EXT and AXI_SRAM. The AXI_EXT interface is designed to allow the DMA controller to access off-chip memory such as DRAM and flash memory. The AXI_SRAM is designed to allow the DMA controller to access on-chip SRAM.

The number of AXI_EXT and AXI_SRAM interfaces varies based on the NPU configuration.

AMBA® 5 APB completer interface with wake-up signaling

This interface enables the device driver that runs on the external host microcontroller to access the control registers of the NPU.

Interrupt interface

This interface sends interrupt requests to the external host microcontroller, usually to signal a completed job.

Two Q-Channel interfaces

These interfaces enable communication with an external clock controller and power controller. This communication enables the system to automatically disable the clock of the NPU or disable the power to it. The clock is otherwise free-running. The NPU does not quiesce while executing a task, and usually does not quiesce if there are any tasks in a job queue.

The NPU software stack partly manages the activity the Q-Channel reports on.

You can configure the NPU to request underclocking or powering down when it is idle. This underclocking can be when the command queue is empty or when the NPU is waiting to be restarted after being stopped.

Clock and reset

The NPU has one clock and one reset signal.

Arm® recommends that the AXI and NPU clock is the same. However, a different clock ratio can be supported using the CLKEN signals.

Configuration signals

The following configuration signals determine the security level after boot:

- PORPL
- PORSL
- CFGSRAMCAP[31:0]
- CFGEXTCAP[31:0]
- CFGSRAMHASH0[39:0]
- CFGSRAMHASH1[39:0]
- CFGEXTHASH0[39:0]

Note

The signals CFGSRAMHASH1[39:0] and CFGEXTHASH0[39:0] are not present in all configurations of the NPU.

Warm reset signals

The following signals communicate a halt of the NPU:

- WRSTQREQ
- WRSTQACK

Debug signals

The following signals are used to halt and restart the NPU during a debug session:

- DBGEN
- SPIDEN
- DBGHALTREQ
- DBGRESTARTREQ
- DBGHALTACK

2.3.2 Central control

The *Central Control* (CC) is the main control unit inside the NPU. The CC controls how the NPU processes neural networks, maintains synchronization, and handles data dependencies.

The CC receives tasks from the external host microcontroller. The CC queues and dispatches units of work to the DMA controller, *Weight Decoder* (WD), MAC unit, and *Activation Output* (AO) unit. The DMA controller and MAC unit send events to the CC to signal the completion of work.

The CC comprises a command unit. This unit receives commands and parses them. Traversal tasks are passed to the traversal unit. The offline compiler can code data dependencies into the NPU command stream, so that data dependencies between commands are not broken. Measuring the data dependency is an NPU internal process.

The CC comprises a traversal unit. The CC instructs this unit to handle commands that require traversal. This unit breaks commands down into smaller commands, performs synchronization as they execute, and implements the different data flows the NPU requires.

The CC contains multiple sets of operation settings to increase efficiency. Multiple sets enable the CC to set up the next piece of work while the current one is being processed.

After completing scheduling, dispatching, or processing work, the CC checks for any events that have been triggered. If there are no new events, the CC requests underclocking or powerdown, depending on the configuration.

Other commands can:

- Trigger interrupts.
- Cause the NPU to wait for a data dependency to clear.
- Set up internal registers with information relating to the next execution step.

The CC implements an Arm® AMBA® 5 APB completer interface. This interface enables the microcontroller to control the NPU. This interface also enables performance measurements.

2.3.3 Configuration pins and AXI port striping

Using certain configuration pins to change how the NPU uses the AXI ports to access external memory can improve performance.

Striping

Striping allows the NPU to access the external memory in parallel, enabling higher throughput and leading to better inferences per second performance. To enable striping, the values on the hash pins must be tied off from the NPU top level to values corresponding to the striping boundary. The minimum striping boundary is 64 bytes.

Striping control

There are configuration hash pins that control striping. The different pins enable striping in the following ways:

CFGSRAMHASH0

This pin enables the NPU to stripe across AXI_SRAM ports 0 and 1 when needed.

CFGSRAMHASH1

This pin enables the NPU to stripe across AXI_SRAM ports 2 and 3 when present.

CFGEXTHASH0

This pin enables the NPU to stripe across AXI_EXT ports when present.

The following is the formula for generating a 2-bit hash:

```
sel[0] = ^(CFGSRAMHASH0[39:6] & addr[39:6])
sel[1] = ^(CFGSRAMHASH1[39:6] & addr[39:6])
```

The 2-bit hash values `{sel[1]` and `sel[0]}` are used to select between ports SRAM0 and SRAM1 or SRAM2 and SRAM3.

For DRAM, we have at most 2 ports, so the following is the formula:

```
sel = ^(CFGEXTHASH0[39:6] & addr[39:6])
```

When 2 SRAM ports are present or connected, CFGSRAMHASH1 must be set to 0, so only ports 0 and 1 get selected. To use 1 SRAM port, both CFGSRAMHASH0 and CFGSRAMHASH1 must be set to 0.

Note

Setting the hash pins to 0 disables one of the AXI ports from being used. For example if CFGSRAMHASH0 is set to 0x0, the AXI_SRAM port 1 is not used. If both CFGSRAMHASH0 and CFGSRAMHASH1 are set to 0, AXI_SRAM port 1 and AXI_SRAM port 3 are unused.

Configuration hash pins and corresponding striping

The following table shows example values for the configuration hash pins and the corresponding striping implemented.

Table 2-1: Configuration hash pins and corresponding striping

Ports	Striping	Formula	CFGSRAMHASH0 and CFGEXTHASH0	CFGSRAMHASH1
1	N/A	N/A	0x0000000000	0x0000000000
2	64B trivial	addr[6]	0x0000000040	0x0000000000
2	128B trivial	addr[7]	0x0000000080	0x0000000000
2	256B trivial	addr[8]	0x0000000100	0x0000000000
2	512B trivial	addr[9]	0x0000000200	0x0000000000
2	256B Fibonacci	addr[8] ^ addr[9] ^ addr[10] ^ addr[11] ^ addr[13] ^ addr[16] ^ addr[21] ^ addr[29]	0x0020212f00	0x0000000000
4	64B trivial	addr[7:6]	0x0000000040	0x0000000080
4	128B trivial	addr[8:7]	0x0000000080	0x0000000100
4	256B trivial	addr[9:8]	0x0000000100	0x0000000200
4	512B trivial	addr[10:9]	0x0000000200	0x0000000400

4	256B Fibonacci	sel[0] = addr[8] ^ addr[10] ^ addr[13] ^ addr[21]
		sel[1] = addr[9] ^ addr[11] ^ addr[16] ^ addr[29]
			0x0000202500	0x0020010a00

Max outstanding transaction control for each AXI port type

The NPU also has configuration pins that allow the integrator to control the max outstanding transactions that each AXI port type can issue.

The following are the configuration pins:

CFGSRAMCAP[31:0]

Configuration of capabilities for SRAM ports.

CFGEXTCAP[31:0]

Configuration of capabilities for EXT ports.

The NPU uses the minimum value set by these ports and the preceding software programmable registers to program its internal DMA to set the max outstanding read and writes of the DMA unit and the max allowed burst length.

For example, if the CFGSRAMCAP[5:0] port is set to 0x3 and the APB register BASE_CFG_SRAM_CAP[5:0] is set to 0x7, the DMA only issues a maximum of 4 outstanding read transactions on its AXI_SRAM type ports. This limit is applied per port.

2.3.3.1 AXI ports per NPU configuration

The following table shows the number of AXI ports in each configuration of the NPU.

Table 2-2: Number of AXI ports in each configuration of the NPU

NPU Configuration (MACs/CC)	Number of SRAM ports (128b) 40-bit Address	Number of EXT ports (128b) 40-bit Address
128	2	1
256	2	1
512	2	1
1024	2	2
2048	4	2

2.3.4 DMA controller

The DMA controller manages all transactions that use the two kinds of Arm® AMBA® 5 AXI interfaces.

The channels that the DMA controller uses are:

Command channel

The NPU uses this channel to read the command stream, normally from on-chip SRAM. The NPU moves the commands into *Central Control* (CC). The microcontroller activates the command channel when it sets up the location and size of the command queue. The microcontroller sets up the command queue by using the registers that are mapped to the AMBA® 5 APB.

***Input Feature Map* (IFM) channel**

The NPU uses the IFM channel to read input feature maps and stores them in the *Input Buffer* (IB). Because the IB must store activations from different x,y coordinates in different words, the DMA controller unpacks data which is stored in NHWC format. This data might require extra internal buffering, but only for the initial layer of a job. Internal layers can use a more efficient format.

When the DMA controller is in vector-product mode, it supports fetching multiple batches.

The IFM channel is triggered once per block for blocks that require input feature maps.

***Input Feature Map Stream* (IFMS) channel**

The NPU uses the IFMS channel to read input feature maps and move the data to the *Activation Output* (AO) unit. This stream is also used to move data to the WD when the NPU executes specific operators such as MATMUL. The channel supports the NHWC and

NHCWB16 off-chip memory formats natively. Because the receiver must store activations from different x,y coordinates in different words, the DMA controller unpacks data which is stored in NHWC format. The DMA controller mainly uses the IFMS channel to read elementwise operations inputs.

The IFMS channel is triggered once for unary elementwise operators or twice for binary elementwise operators per work unit of the AO unit.

***Output Feature Map* (OFM) channel**

The NPU uses the OFM channel to write output feature maps from the *Output Buffer* (OB) to any memory mapped to its AXI interfaces. This memory can be the on-chip SRAM using the AXI_SRAM interface or off-chip DRAM using the AXI_EXT interface.

The OFM channel supports NHWC and NHCWB16 memory formats natively and can support NCHW by using transpose operations.

The traversal unit triggers the OFM channel once per output block for blocks that require transfer to any memory mapped to the two kinds of AXI ports of the NPU.

Weight channel and fast weight channel

The weight and fast weight channels transfer compressed weights from external memory to the weight decoder. The DMA controller uses a read buffer to hide bus latency from the weight decoder and to enable the DMA to handle data arriving out of order.

The traversal unit triggers these channels for blocks that require the transfer of weights.

The weight stream must be quantized to eight bits or less by an offline tool. When passed through the offline compiler, weights are compressed losslessly and reordered into an NPU-specific weight stream. This process is effective, if the quantizer uses less than eight bits or if it uses clustering and pruning techniques. The quantizer can also employ all three methods. Using lossless compression, an average of approximately two bits per weight is possible in the final weight stream, especially when the weight stream has many zeros.

mem2mem channel

The NPU uses this channel to stream general data from memory to memory. The main purpose of this channel is to read weights from slow, external memory such as DRAM or flash and store them in the on-chip SRAM. This read might be performed in preparation for a layer which reads the weights multiple times. Having the weights in on-chip SRAM saves power and improves performance compared to reads to non-volatile memory.

The traversal unit triggers mem2mem operations on specific API commands.

Bias and scale channel

This channel streams data to the AO unit. The data that it transmits is the scale and bias necessary for the block that the NPU is processing. Layers that pass through the AO unit are written to the on-chip SRAM. As the layers pass through the AO unit, activation functions can be fused.

2.3.5 Clock and power module

The *Clock and Power Module* (CPM) handles hard and soft resets, contains registers for the current security settings, the main clock gate, and the QLPI interface.

Clock and power module controlling reset

The nRESET input signal triggers a hard reset. When the APB RESET register is written to, a soft reset is triggered, as long as Write-Access is permitted. The APB-PPROT and CPL, CSL register values determine whether a write is permitted.

Register access to APB RESET is permitted, if (PPROT[0]>=CPL && PPROT[1]<=CNS). Otherwise the register access is not permitted.

At any reset, all registers and memories in the NPU are cleared to prevent leakage between Security states. The CPM triggers all soft resets. Hard resets must come from an external reset controller.

Both hard and soft resets use a similar procedure, which is:

1. If the reset is a soft reset:
    a. With the DMA controller clock on, signal to the DMA that a soft reset is initiated.
    b. Wait for the DMA to acknowledge the reset request.
2. With the internal NPU clock off, activate the system reset within two clock cycles.
3. Deactivate the system reset.
4. With the internal memory and DMA controller clock on, the CPM signals to the internal memory and the DMA that the RAMs must be cleared.
5. Update the setting in the CPL, CSL register.

QLPI for clock

To enable high-level clock gating, the NPU exposes the Q-Channel. The Q-Channel enables the system to automatically disable the clock of the NPU, that is free-running except during reset.

If the entire NPU is in stopped state, it indicates when the clock can be turned off. You can configure the NPU registers using the AMBA® 5 APB, so that it keeps requesting a clock in stopped state.

QLPI for power

For high-level power gating, the NPU exposes the Q-Channel. The Q-Channel permits the system to automatically disable the power of the NPU.

If the entire NPU is in stopped state, it indicates when power can be turned off. You can configure the NPU using the AMBA® 5 APB, so that it keeps requesting power in stopped state.

Clock and power module clock gates

The CPM contains one main clock gate. Other clock gating is performed inside each of the blocks, which the CPM can override. These clock gates are explicitly instantiated, with the CPM clock gate preceding the block level clock gates.

2.4 Weight decoder

The *Weight Decoder* (WD) reads the weight stream from the DMA controller. The WD decompresses and stores this stream in a double-buffered register, ready for the MAC unit to consume it.

The WD supports two weight stream formats, the standard decoder and the *Fast Weight Decoder* (FWD). The standard decoder performs higher compression and the FWD has higher bandwidth.

The WD also contains a tensor core that is targeted for use in MATMUL operations where two *Input Feature Maps* (IFM) are multiplied. The tensor core reformats one of the IFMs the MATMUL operation needs into a format the MAC unit desires.

2.5 MAC unit

The MAC unit performs multiply-accumulate operations that are required for convolution, depth-wise convolution, and vector products. The MAC unit also supports pooling modes such as MAX, MIN, AVERAGE, REDUCE_SUM, and many more.

The MAC unit does accumulation on two levels. L1 accumulation happens close to the DPUs every clock cycle and the MAC unit stores the results in the L1 *Accumulator Unit* (AU) registers. L2 accumulation usually happens every eight clock cycles and the MAC unit stores the results in the accumulator buffer.

The NPU supports a zero-weight optimization scheme referred to as 2:4 ratio sparsity. In this scheme, the weights of a network are pre-pruned so two out of four weights in the input channel are guaranteed to be zero. In this way, the MAC unit can double its performance because the calculations corresponding to the pre-pruned zeros can be omitted. Therefore, the *Dot Product Units* (DPUs) in the MAC unit can process twice the depth of the *Input Feature Map* (IFM) per clock cycle in comparison to the default usage.

The MAC unit comprises:

- An *Input Feature Map* (IFM) unit
- *Dot Product Units* (DPU)
- *Activation Input* (AI) unit
- L1 *Accumulator Unit* (AU)
- L2 *Accumulator Unit* (AU)

2.5.1 Dot product units

The MAC unit contains a configurable number of *Dot Product Units* (DPUs). These DPUs perform the multiply-accumulate operations that are required for convolutions.

The DPUs also contains logic to support several modes of pooling.

2.5.2 Activation input unit

The *Activation Input* (AI) unit manages the input buffer where the *Input Feature Map* (IFM) data is temporarily stored during operations.

The DMA module sends IFM data to the input buffer inside the AI unit. The DPUs inside the MAC unit consume this data while performing calculations.

The AI unit also handles zero-padding around feature maps and the replication or zero-insertion upscaling for deconvolution.

2.5.3 L1 Accumulator Unit

The L1 *Accumulator Unit* (AU) performs level one of accumulation. The number of L1 accumulators is the same as the number of DPUs. Every clock cycle the L1 accumulators receive input data from the DPUs.

Each L1 accumulator can store two values, one that the DPU updates, and one that the L2 *Accumulator Unit* (AU) reads.

2.5.4 L2 Accumulator Unit

The L2 *Accumulator Unit* (AU) performs level two of accumulation. The accumulators in this unit are updated at fixed intervals with input data from the L1 AU. The L2 AU reads and writes data to the *Accumulator Buffer* (AB).

The L2 AU also contains logic to support several modes of pooling.

2.6 Activation output unit

The *Activation Output* (AO) unit scales accumulators the MAC unit generates, scales elementwise operations, performs elementwise operation arithmetic, performs the operations the Resize operator needs, and applies activation functions.

The AO unit reads accumulators from the *Accumulator Buffer* (AB) for convolution and vector product operators. Accumulators can remain as 32-bit or 64-bit raw format. For the more common use case, the AO unit scales operators to the 8-bit or 16-bit range of the OFM.

Accumulator scaling consists of per-output-channel bias, scale multiplier, and down-scaling shift. The DMA module provides the bias and scale information as a stream of data. Accumulator scaling use cases include the averaging in Average Pooling, the scaling of Reduce Sum output to implement the Mean operator, and batch normalization. Average Pooling with pad mode SAME uses a hardwired look-up table of scaling information to accelerate kernel sizes up to 8 x 8.

The AO unit supports reverse and transpose of data. These operations are not standalone operations but fused to existing AO unit operations such as RESCALE.

The AO unit supports a wide range of unary and binary elementwise operations. The AO unit also supports independent broadcast and scaling of both input tensors for binary operations. An input tensor can also be specified as a single element scalar in the command stream.

The AO unit also supports chaining of operations, where up to three external IFMs can be used to perform up to four elementwise operations. There are three buffers for intermediate chaining data that the AO unit writes to and reads from.

The AO unit supports the following activation functions:

- ReLU
- Leaky ReLU
- A programmable LUT as a generic activation function

ReLU and LUT can be applied directly to the output of any AO operation. Leaky ReLU is a separate elementwise AO operation. As with any elementwise operation, it can be chained onto previous AO output using the chaining mechanism. The chaining buffer buffers the activation data output before streaming the data to the DMA module.

2.6.1 Scaling unit

The scaling unit implements most of the activation data processing within the activation output unit.

The unit consists of dedicated scaling units for *Input Feature Maps* (IFMs) and *Output Feature Maps* (OFMs). The NPU configuration determines the number of scaling units which dictates the number of output elements that the NPU can process in parallel. In addition to supporting the scaling defined by the TOSA Rescale operator, the scaling unit includes several extra rounding modes. These rounding modes extend the possible numeric precisions to allow the possibility of bit-exact results for multiple frameworks or future operators.

OFM scaling supports per-output-channel bias, scale multiplier, and right shift. A per-tensor scaling mode is also available as an alternative to per-output-channel scaling.

IFM scaling supports per-tensor scaling only. The right shift is performed with a per-tensor programmable rounding mode for both IFM and OFM.

2.6.2 Lookup table

The *Activation Output* (AO) unit supports lookup through a programmable table, where the activation acts as an index in the *LookUp Table* (LUT). This LUT can be used as a generic activation function or to reduce a series of layers of pointwise operations into a single fused lookup.

The AO unit supports the following modes for LUTs:

- Direct 8-bit to 8-bit lookup from 8 tables of 256 bytes each
- Direct signed 8-bit to signed 16-bit lookup from 4 tables of 512 bytes each
- Direct signed 8-bit to signed 32-bit lookup from 2 tables of 1024 bytes each
- Piecewise linear approximation for signed 16-bit to signed 16-bit

The table entries for piecewise linear approximation consists of a 16-bit base and a 16-bit slope. The lookup is made from the high nine bits of the 16-bit activation so a single 2048 bytes table of 512 32-bit entries is available. The interpolation is made from the low seven bits of the 16-bit activation.

Although the direct modes enable bit-exact support for any activation or pointwise function, the linear approximation available for 16-bit activations likely diverges from a bit-exact representation.

2.6.3 Chaining buffer

The chaining buffer is an input buffer for elementwise operations.

The chaining buffer supports chaining of operators, allowing an output buffer for one operation to be reused as input buffer for a following operation. The chaining of operators reduces the external AXI traffic and reduces power consumption. The chaining of operators can improve performance. The chaining of operators can also reduce the on-chip SRAM area.

2.6.4 Output Buffer

The *Output Buffer* (OB) is a buffer for *Output Feature Map* (OFM) data.

The *Activation Output* (AO) unit buffers OFM data before transferring the data to the DMA controller.

Proprietary Notice

This document is protected by copyright and other related rights and the use 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 Limited ("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 for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. 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, any patents, copyrights, trade secrets, trademarks, or other rights.

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.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in the Arm documents.

Product status

All products and Services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

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

Product revision status

This product is r0p0, which indicates the revision status of the product described in this manual, where:

r ***(value)*** Identifies the major revision of the product, for example, r1.

p ***(value)*** Identifies the minor revision or modification status of the product, for example, p2.

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
0000-02	5 June 2024	Non-Confidential	First EAC release for r0p0
0000-01	5 April 2024	Non-Confidential	First Beta release for r0p0

The first table is for the first release. Then, each table compares the new issue of the manual with the last released issue of the manual. Issue numbers match the revision history in Document release information on page 30.

Table 2: Issue 0000-01

Change	Location
First Non-confidential Beta release for r0p0	-

Table 3: Differences between issue 0000-01 and 0000-02

Change	Location
Updates the TOSA version	1. Description of the Arm Ethos-U85 Neural Processing Unit on page 5
Clarifies the description from TFLµ tool to TFLµ runtime	1.1 Supported application programming interfaces on page 7
Clarifies how the DMA controller moves data	2.3 Functional blocks on page 15
Updates the name of one of the pins and clarifies the wording of the description of using 1 SRAM port.	2.3.3 Configuration pins and AXI port striping on page 18
Updates the description of the *Output Feature Map* (OFM) channel	2.3.4 DMA controller on page 21
Updates the hard and resets procedure	2.3.5 Clock and power module on page 22
Updates the weight decoder description	2.4 Weight decoder on page 24

Conventions

The following subsections describe conventions used in Arm documents.

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: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

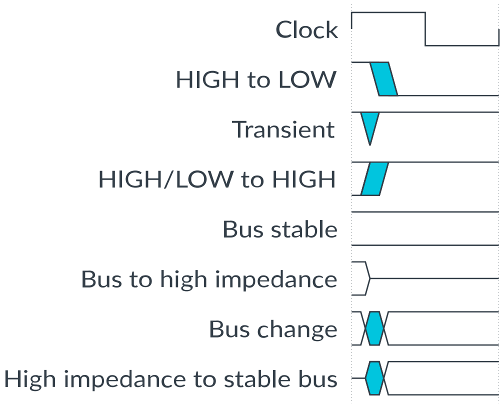
Caution	We recommend the following. If you do not follow these recommendations your system might not work.
Warning	Your system requires the following. If you do not follow these requirements your system will not work.
Danger	You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.
Note	This information is important and needs your attention.
Tip	This information might help you perform a task in an easier, better, or faster way.
Remember	This information reminds you of something important relating to the current content.

Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

Figure 1: Key to timing diagram conventions



Signals

The signal conventions are:

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name, n denotes an active-LOW signal.

Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Access to Arm documents depends on their confidentiality:

- Non-Confidential documents are available at developer.arm.com/documentation. Each document link in the following tables goes to the online version of the document.
- Confidential documents are available to licensees only through the product package.

Arm product resources	Document ID	Confidentiality
Arm® Ethos™-U85 NPU Configuration and Integration Manual	102683	Confidential
Arm® Ethos™-U85 NPU iBEP user guide	PJDOC-1505342170-539487	Confidential
ML Developers Guide for Cortex-M Processors and Ethos-U NPU	109267	Non-Confidential

Arm architecture and specifications	Document ID	Confidentiality
AMBA® AXI Protocol Specification	IHI 0022	Non-Confidential
AMBA® Low Power Interface Specification	IHI 0068	Non-Confidential
TOSA specification	N/A	Non-Confidential