Arm® Ethos™-U55 NPU

Revision: r2p0

Technical reference manual

Non-Confidential

Issue 01

102420_0200_01_en

Arm® Ethos™-U55 NPU

Technical reference manual

Release Information

Document history

Issue	Date	Confidentiality	Change
0200-01	1 April 2021	Non-Confidential	First EAC release for r2p0.

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

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

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

This document includes terms that can be offensive. We will replace these terms in a future issue of this document.

If you find offensive terms in this document, please contact terms@arm.com.

Contents

1 Introduction

1.1 Product revision status

The r*x*p*y* identifier indicates the revision status of the product described in this manual, for example, r*1*p*2*, where:

r*x* Identifies the major revision of the product, for example, r1.

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

1.2 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™-U55 NPU.

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

Convention	Use
italic	Introduces 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` ***`bold`***	Denotes language keywords when used outside example code.
`monospace` <u>`underline`</u>	Denotes 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>`

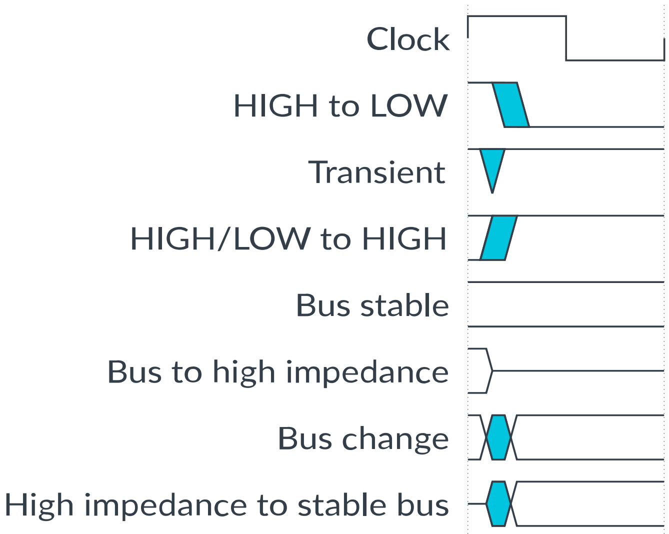
Convention	Use
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings, that are defined in the *Arm® Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.
Caution	This represents a recommendation which, if not followed, might lead to system failure or damage.
Warning	This represents a requirement for the system that, if not followed, might result in system failure or damage.
Danger	This represents a requirement for the system that, if not followed, will result in system failure or damage.
Note	This represents an important piece of information that needs your attention.
Tip	This represents a useful tip that might make it easier, better or faster to perform a task.
Remember	This is a reminder of something important that relates to the information you are reading.

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 1-1: 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.

1.4 Additional reading

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

Table 1-2: Arm Publications

Document name	Document ID	Licensee only
AMBA® AXI and ACE Protocol Specification AXI3, AXI4, AXI5, ACE, and ACE5	IHI 0022	No
AMBA® Low Power Interface Specification Arm® Q-Channel and P-Channel Interfaces	IHI 0068	No
Arm® Ethos™-U55 NPU Technical overview	101886	No
Arm® Ethos™-U NPU Application development overview	101888	No
Arm® Ethos™-U55 NPU Configuration and integration manual	101887	Yes
Arm® Ethos™-U NPU Functional model integration guide	101889	Yes

1.5 Feedback

Arm welcomes feedback on this product and its documentation.

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

Information about how to give feedback on the content.

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

- The title Arm® Ethos™-U55 NPU Technical reference manual.
- The number 102420_0200_01_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.

---

2 Neural processing unit introduction

This chapter introduces the processor.

2.1 Description of the neural processing unit

The *Neural Processing Unit* (NPU) improves the inference performance of neural networks. The NPU targets 8-bit and 16-bit integer quantized *Convolutional Neural Networks* (CNN) and *Recurrent Neural Networks* (RNN). The NPU supports 8-bit weights.

Arm delivers the hardware *Register Transfer Level* (RTL) of the NPU with an open-source driver and compiler. A neural network must be compiled offline 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 driver.

The driver programs the memory location of the command stream and other payloads into registers in the NPU. The *Central Control* (CC) processes the command stream.

The NPU includes a *Direct Memory Access* (DMA) controller that can read and write to external memory. 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) and *Output Feature Maps* (OFMs) and NPU-private intermediate data that is also held in system memory.

During runtime, *TensorFlow Lite* (TFL) loads the the `flatbuf` file, in which the Offline Compiler has created an Ethos™-U55 command stream for each custom operator. The driver gives a pointer to this command stream so that the NPU hardware can execute it. This means that the entire network can be a single operator that is run fully on the Ethos™-U55. The NPU reads the data (weights, commands, IFMs, OFMs, bias and scale) autonomously using the DMA.

The NPU uses a working buffer in SRAM for IFMs and OFMs in flight. The Offline Compiler decides the scheduling of this buffer and codes it into the command stream. The NPU uses the DMA to read and write autonomously to this work buffer. The location of the buffer is set at runtime through registers, meaning the coding in the command stream is relative, not absolute.

The external interfaces that the NPU implements are:

- Two Arm® AMBA® 5 AXI master interfaces that provide the DMA controller with access to external memory. One read/write master, M0, and one read-only master, M1. This means the NPU can present two sets of transactions at the same time. The command, weight, bias, and scale channels can be mapped to either AXI master.

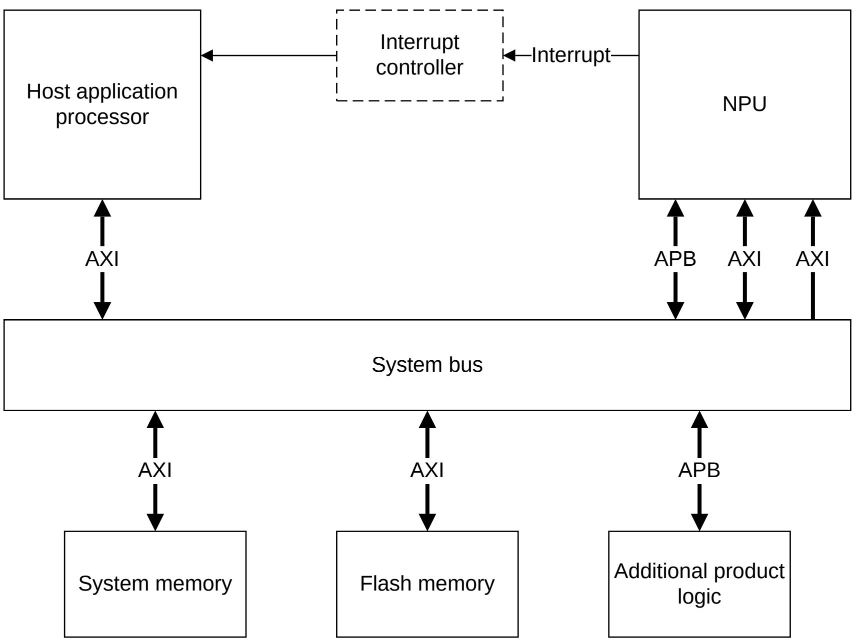
Note

The master interfaces are also AMBA® 4 AXI compatible.

- An Arm® AMBA® 4 APB slave interface with wake up signaling that allows the application processor to program the NPU.

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

Figure 2-1: Typical system configuration block diagram

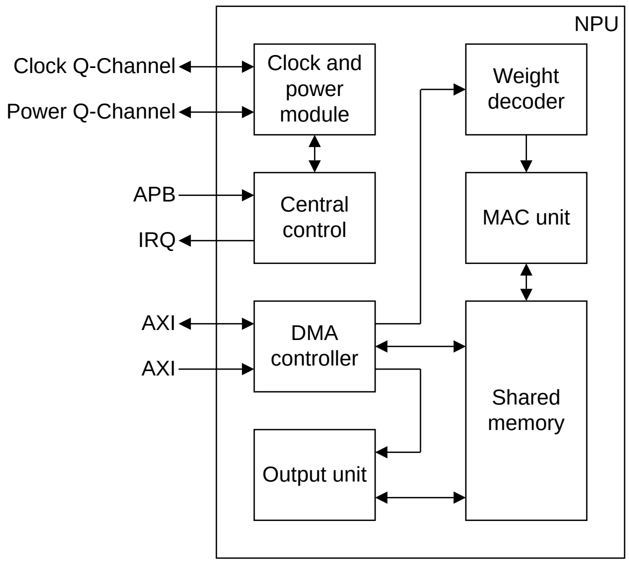


Note

Where the system has only SRAM and no flash memory, then the read-only master interface, M1, can be tied off.

The following figure shows the main components of the NPU.

Figure 2-2: Functional blocks diagram



2.1.1 Supported application programming interfaces

To program, test, and monitor the NPU, Arm deploys the open-source *TensorFlow Lite for Microcontrollers* (TFLµ) tool, which runs on an external host application processor. It uses the compiler offline to compile and optimize the neural network graph for the NPU. Its API generates a command stream for the NPU to process.

The compiler decides which parts of a network graph can be optimized and executed on the NPU. The NPU drivers manage the workloads that execute inferences on the NPU.

If the network maps exclusively to the NPU, then the power required by the external host application processor is negligible. If there is a requirement to process layers on the Cortex®-M core, then more performance is required.

2.1.2 Security support

The NPU supports TrustZone using security and privilege status on APB and AXI bus transactions.

The NPU also has a strict reset policy, where at any reset, all registers and memories in the NPU are cleared to prevent leakage between Secure and Non-secure processing.

The NPU security and privilege status can only be changed by software during soft reset and the host application processor cannot set the NPU to a higher security or privilege level than its own level. Also, the host application processor must have equal or higher security and privilege than the NPU for access through the APB.

To protect against illegal transactions, the AXI provides access permission signals.

The term AxPROT refers collectively to the **ARPROT** and **AWPROT** AXI signals. AxPROT contains both a security (AxPROT[1]) and a privilege (AxPROT[0]) level on the AXI masters M1 and M2.

Note

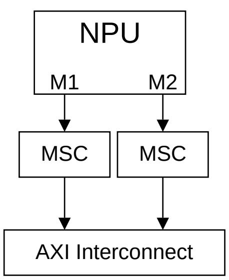
AxPROT also has an extra bit AxPROT[2] that is always LOW on the NPU. For further information about AxPROT, see the *AMBA® AXI and ACE Protocol Specification AXI3, AXI4, AXI5, ACE, and ACE5*.

Protection against unsecure access by the NPU on a system level is provided by more TrustZone components. The following figure shows the NPU integrated in a simplified example subsystem using the *Master Security Controller* (MSC) component. The MSC uses the NPU security and privilege level with a look-up table to determine if each bus transaction is allowed. When acting as an AXI master on allowed transaction, the configuration of each MSC determines the security and privilege status of that MSC.

Note

The MSC is only one example of a TrustZone component that interfaces with the NPU.

Figure 2-3: Simplified example subsystem



2.2 Interfaces

The NPU has several external interfaces.

The external interfaces are:

- Arm® AMBA® 4 APB slave with wake-up signaling.
- Two Arm® AMBA® 5 AXI masters:
  - A read/write master, M0.
  - A read-only master, M1.
- An interrupt.
- Two Q-Channels:
  - A Q-Channel for clock.
  - A Q-Channel for power.

- System configuration signals that determine the security level after boot.
- Clock.
- Reset.

2.3 Documentation

Arm Limited publishes documentation that describes the NPU, including this document.

Technical overview

The *Technical overview* (TO) describes the functionality of the NPU.

Technical reference manual

The *Technical reference manual* (TRM) describes the 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.

Application development overview

The *Application development overview* (ADO) describes the flow of data between an application and the NPU.

Configuration and integration manual

The *Configuration and integration manual* (CIM) describes the configuration and implementation of the NPU.

Functional model integration guide

The *Functional model integration guide* (FMIG) describes how to integrate the NPU functional model.

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

2.4 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.

2.5 Product revisions

Successive product revisions have differences in functionality.

r0p0

First release.

r0p1

No functional changes. Updates to the Q-Channel and internal reset to create glitch-free signals.

r1p0

Power improvements to the MAC unit focusing on glitch reduction and added support for the Power Q-Channel and ECC memories.

r2p0

Adds a new configuration option to allow the use of a custom DMA instance. This allows you to instantiate your own DMA and control up to four different internal data channels, *Input Feature Maps* (IFMs), *Output Feature Maps* (OFMs), weights, and bias and scale.

3 Functional description

This chapter describes the function and structure of the processor.

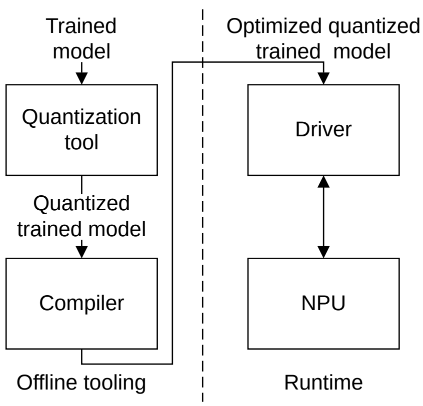
3.1 Control and data flow

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

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 3-1: The software stack of the NPU



The NPU uses offline tools to optimize the code. At runtime, the application processor 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™-U55 software. The compiler runs offline on the TFL flatbuffer; the compiler 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. This tool quantizes weights to 8-bit and activations to 8-bit or 16-bit values.
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.

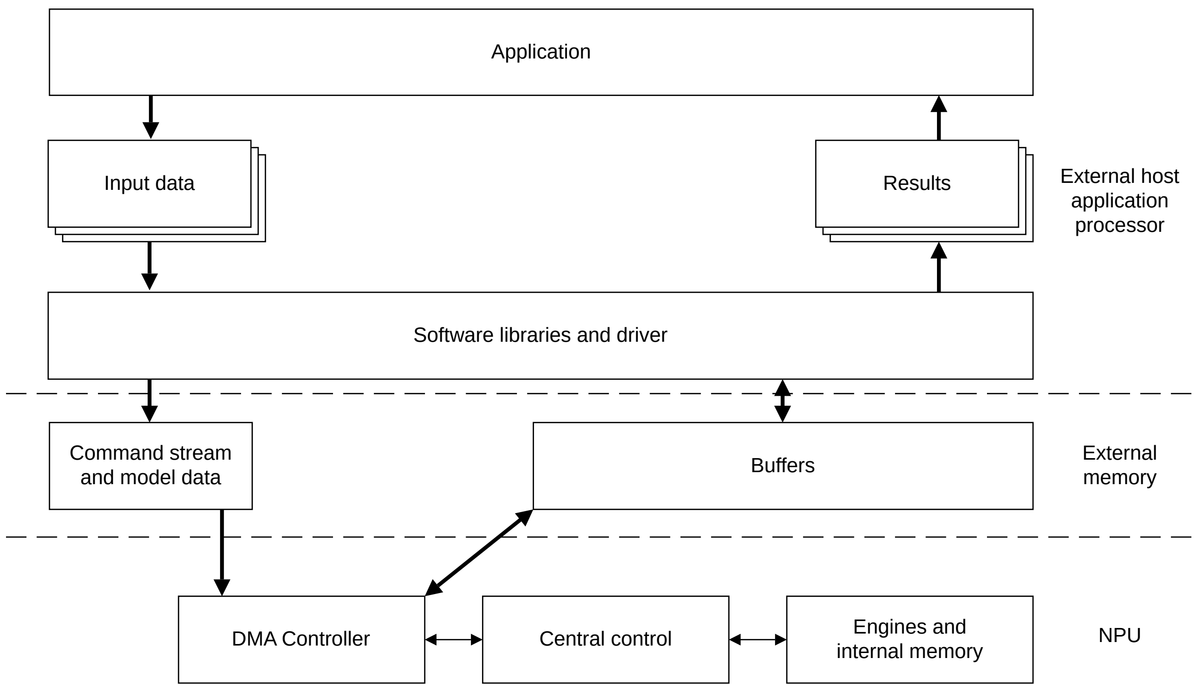
The following steps describe the runtime control and data flow:

1. The optimized model is placed in system memory, which is accessible by the NPU.
2. At runtime, the TFLµ tool reads the model and dispatches the operators.

3. The NPU reads the optimized model and runs the command stream that is included in it. The application processor 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 3-2: Control and data flow



3.1.1 Supported memory formats for feature maps

The NPU supports the industry-standard NHWC format of feature-map data.

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

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.
- Only tile 0 can be used, the address of tile 0 is `BASE`.

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-byte aligned.
- Tiles can be used.

3.2 Security and boot flow

The NPU can be set to different security and privilege modes during a reset. The host application processor 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 are complete before issuing the reset.

When the NPU is powered up after a hard reset, it reads the **PORPL** signal to set its privilege level:

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

When the NPU is powered up after a hard reset, it reads the **PORSL** signal to set its 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 may be used for memory protection at system-level.

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).

3.3 Functional blocks

The NPU consists of the *Central Control* (CC), a DMA controller, a MAC unit, an Output unit, and the interconnect fabric.

The following are descriptions of the units of the NPU:

- The CC receives tasks from the external host application processor. The CC queues and dispatches units of work to the DMA and engines.
- The DMA controller uses its two Arm® AMBA® 5 AXI master interfaces to move data between external memory and internal shared memory.

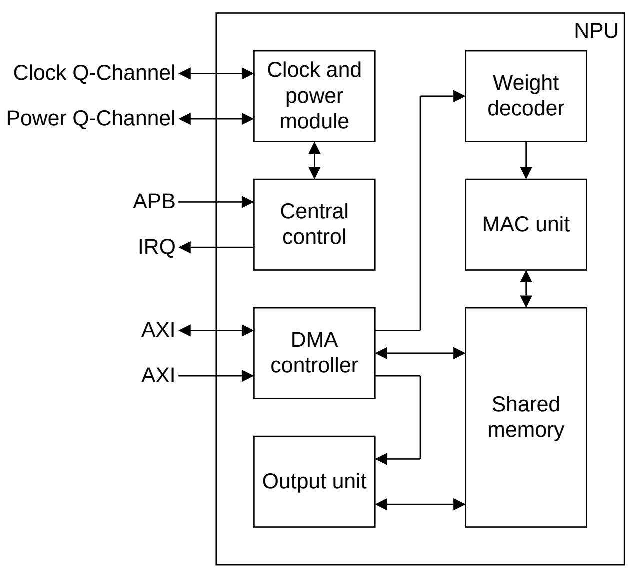
Note

The read-only AXI master, M1, can also be connected to SRAM, provided it has sufficient bandwidth, otherwise RAM stalls can occur. See also 2.1 Description of the neural processing unit on page 13.

- The MAC unit has various internal units for reading IFMs, performing dot products and accumulations.

The following figure shows the main components of the NPU.

Figure 3-3: Functional blocks diagram



3.3.1 External interfaces

The NPU uses two AMBA® 5 AXI master interfaces, an AMBA® 4 APB slave interface with wake-up signaling, an interrupt interface, two Q-Channel interfaces, clock, and reset to enable access to and from external components.

Two AMBA® 5 AXI master interfaces

These interfaces enable read- and write-access to external memory for the DMA controller.

The NPU has one read/write master, M0, and one read-only master, M1.

AMBA® 4 APB slave interface with wake-up signaling

Enables the device driver that runs on the external host application processor to access the control registers of the NPU.

Interrupt interface

Sends interrupt requests to the external host application processor, 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 be the same; however, a different clock ratio can be supported using the **CLKEN** signals.

3.3.2 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 application processor. The CC queues and dispatches units of work to the DMA controller, weight decoder, MAC unit, and Output unit. The DMA controller and MAC unit send events to the CC to signal the completion of work.

The CC contains multiple sets of operation settings to increase efficiency. This enables 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.

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 comprises a Command unit. This unit receives commands and parses them. Traversal tasks are passed to the traversal unit. Data dependencies can be coded into the NPU command stream by the Offline Compiler, so that data dependencies between commands are not broken. Measuring the data dependency is an NPU internal process.

Other commands can:

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

The CC implements an Arm® AMBA® 4 APB slave interface. This interface enables the application processor to control the NPU. This interface also enables performance measurements.

3.3.3 DMA controller

The DMA controller manages all transactions that use the Arm® AMBA® 5 AXI interfaces. It is also possible to add a custom DMA instance to drive a subset of the DMAs channels.

The channels that the DMA controller uses are:

Command channel

The NPU uses this channel to read the command stream, normally from external flash. The NPU moves the commands into CC. The application processor activates the command channel when it sets up the location and size of the Command queue. It sets up the Command queue by using the registers that are mapped to the AMBA® 4 APB.

IFM channel

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

The DMA controller considers the kernel stride, because this affects which bank or address the DMA controller requires to store activations.

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.

OFM channel

The NPU uses the OFM channel to write output feature maps from shared RAM to external RAM. Because the output is double-buffered in the shared RAM, the DMA requires an interface to synchronize with the output module to notify the DMA which buffer is empty or full.

For the last layer of a job, the output must be written out in NHWC format. This may require the DMA to pack the data, depending on the depth of the layer. Since this process reduces the bandwidth, this process is possible in a small register bank inside the DMA.

The traversal unit triggers the OFM channel once per output block for blocks that require transfer to external memory.

Weight channel

The weight channel transfers 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 the weight channel for blocks that require the transfer of weights.

The weight stream must be quantized to 8 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 8 bits or if it uses clustering and pruning techniques, it may also employ all three methods. Using lossless compression, an average of ~2 bits is possible in the final weight stream, especially if 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, non-volatile memory and store them in the SRAM. This might be performed in preparation for a layer which reads the weights multiple times. Having the weights in 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 Output 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 Output unit are written to the external SRAM. As the layers pass through the Output unit, activation functions can be fused.

Note

Only the mem2mem DMA channel is controllable directly by the command stream. The other channels are used to load or store data required by NPU operations. Write DMA channels must always use AXI port 0. Read DMA channels can use AXI port 0 or 1 according to which region is configured for the memory.

3.3.4 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.

3.3.4.1 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 shared buffer and DMA controller clock on, the CPM signals to the shared buffer and the DMA that the RAMs must be cleared.
5. Update the setting in the CPL, CSL register.

3.3.4.2 QLPI for clock

To enable high-level clock gating, the NPU exposes a Q-Channel slave port. This slave port 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® 4 APB, so that it keeps requesting a clock in stopped state.

3.3.4.3 QLPI for power

For high-level power gating, the NPU exposes a Q-Channel slave port. This slave port 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® 4 APB, so that it keeps requesting power in stopped state.

3.3.4.4 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.

3.3.5 Weight decoder

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

3.3.6 MAC unit

The MAC unit performs multiply-accumulate operations that are required for convolution, depth-wise pooling, vector products, and the max operation required for max pooling.

The MAC unit comprises:

- An IFM unit
- Dot product units
- An adder array.

3.3.6.1 IFM unit

The IFM unit inside the MAC unit reads the input feature maps from the shared SRAM and stores them in register slices. These slices are fed into the multipliers in the dot product units. The IFM unit also performs some extra services as part of other operations.

The IFM unit handles zero-padding around the outside edge of feature maps and the upscaling that deconvolution requires. Deconvolution upscaling uses nearest neighbor or zero insertion.

3.3.6.2 Dot product units

The MAC unit contains several dot product units. These dot product units perform the multiply-accumulate operations that are required for convolutions.

The dot product units contain a max operator that they use for max pooling.

3.3.6.3 Adder array

The adder array reads a set of accumulators from the shared RAM buffer and updates them with partial accumulations from the dot product units. The adder array then writes the result back.

Accuracy is maintained throughout this process. The internal accumulators retain precision so that the output is bit-exact to the software reference, in this case TFL.

The compiler selects the accumulator format in the shared buffer. This format can be:

- 32-bit two's complement
- 40-bit two's complement

You can also configure the compiler to use 16-bit floating-point format, which improves performance but impacts accuracy.

These formats are only used internally.

3.3.7 Output unit

The Output unit reads finished accumulators from the shared RAM and converts them into output activations. This process includes performing scaling for each OFM, adding the bias to values, and applying the activation function to each point.

Every layer is written to external SRAM, but the activation function and scaling are normally fused. There is no forwarding path from output to input inside the NPU. Although layers can be split into horizontal stripes and run in “cascade” to minimize the SRAM footprint. This means that the external SRAM footprint can be smaller than the largest layer.

The activation functions that the Output unit supports are:

- ReLU, ReLU1, ReLU6, and Leaky ReLU
- tanh
- sigmoid
- Configurable Lookup Table (LUT)
- None or bypass

The elementwise operations that the Output unit supports are:

- Elementwise ADD and SUB

- Elementwise Multiplication (MUL)
- Elementwise Min and Max
- Elementwise ABS
- Elementwise Shift Left (SHL) and Elementwise Shift Right (SHR)
- Elementwise Count-leading Zero (CLZ)

When the Output unit has computed output activations, it writes them back into the shared RAM. The output activations are buffered in the shared RAM where they wait for the DMA controller to send them to external memory.

3.3.7.1 Scaling unit

The Scaling unit in the Output unit performs scaling in convolutions and division in average pooling.

The number of scaling operations that are performed per clock depends on the configuration. The number of outputs per clock varies, depending on the operation.

3.3.7.2 ReLU and Leaky ReLU

Rectified Linear Unit (ReLU) operations are typically performed after scaling and bias addition.

The number of ReLU operations that are done in parallel is the same as the number of parallel operations that the Scaling unit performs.

Leaky ReLU (LReLU) is a variant, a nonzero ReLU with a small positive gradient that targets negative values, unlike standard ReLU functions. Leaky ReLU implements Leaky ReLU as long as the input and output quantization scale are the same. The most recent TensorFlow Lite allows the quantization scale to differ. In that case, we recommend using the LUT for 8-bit activations and element wise operators for 16-bit activations.

3.3.7.3 tanh, sigmoid, and LUT

The Output unit supports tanh and sigmoid functions using a hardwired table combined with bilinear interpolation. The same table is used for both functions, because they are mathematically related.

The Output unit can perform one tanh or sigmoid function per cycle.

There is also a *Configurable Lookup Table* (LUT) that can be used for any point-wise activation or function. For 8-bit activations, the LUT holds up to 256 8-bit values that are directly mapped from IFM to OFM. The LUT size increases to 512 for 16-bit values; however, the outputs are interpolated, bilinear values.

The LUT can be configured by setting up a mem2mem transfer. For more information, refer to 3.3.3 DMA controller on page 24.

3.3.7.4 Elementwise operations

The Output unit supports a number of elementwise operations on activations.

CLZ

For 32-bit input it is possible to elementwise count the number of leading zeros.

SHR and SHL

Activations can elementwise be shifted left or right. The operation uses two input, one for the operand to shift and a second for the shift amount.

4 Programmers model

This chapter describes a register and register map of the NPU.

4.1 Register characteristics

The registers in the NPU have common characteristics.

The following are the characteristics of the registers in the NPU:

- Register addresses are shown as offsets from the base address.
- Registers are 32-bit wide words.
- Register reads and writes use word accesses only.
- Register halfword and byte reads are **UNDEFINED**.
- Register halfword and byte writes are **UNPREDICTABLE**.
- Every access to the registers is compared with the *Current active Privilege Level* (CPL) and the active *Current Non-Secure level* (CNS) of the PROT register:
    - Register access is permitted if (PPROT[0]>=CPL && PPROT[1]<=CNS). Otherwise the register access is not permitted.
    - A read access that is not permitted, either due to privilege or being a write-only register, returns the value zero.
    - A write-access that is not permitted, either due to privilege or being a read-only register, is ignored.

4.2 Register page BASE

The NPU control registers bank.

Table 4-1: BASE registers

Address	Link	Usage	Access	Default
`0x00000000`	4.2.1 Register ID on page 32	ID register	Read-only	0x10104201
`0x00000004`	4.2.2 Register STATUS on page 34	Register describes the current operating status of the NPU	Read-only	0x00000008
`0x00000008`	4.2.3 Register CMD on page 36	Command register, reads as last written command	Read/write	0x0000000C
`0x0000000C`	4.2.4 Register RESET on page 37	Request Reset and new security mode	Read/write	0x00000000
`0x00000010`	4.2.5 Register QBASE0 on page 38	Base address of Command-queue bits[31:0]. The address is 4-byte-aligned	Read/write	0x00000000
`0x00000014`	4.2.6 Register QBASE1 on page 38	Address extension bits[47:32] for queue base	Read/write	0x00000000

Address	Link	Usage	Access	Default
0x00000018	4.2.7 Register QREAD on page 38	Read offset in the command stream in bytes. Multiples of 4 in the range 0-16 MB	Read-only	0x00000000
0x0000001C	4.2.8 Register QCONFIG on page 39	AXI configuration for the command stream in the range 0-3. Same encoding as for REGIONCFG	Read/write	0x00000000
0x00000020	4.2.9 Register QSIZE on page 39	Size of the command stream in bytes. Multiples of 4 in the range 0-16 MB	Read/write	0x00000000
0x00000024	4.2.10 Register PROT on page 39	Protection level configured for the NPU when acting as an AXI master	Read-only	0x00000000
0x00000028	4.2.11 Register CONFIG on page 40	RTL configuration	Read-only	Implementation defined
0x0000002C	4.2.12 Register LOCK on page 42	Lock register. This register is designed for driver use and does not affect NPU functionality	Read/write	0x00000000
0x0000003C	4.2.13 Register REGIONCFG on page 42	Base pointer configuration. Bits[2*k+1:2*k] give the memory type for REGION[k]	Read/write	0x00000000
0x00000040	4.2.14 Register AXI_LIMIT0 on page 46	AXI limits for port 0 counter 0	Read/write	0x00000000
0x00000044	4.2.15 Register AXI_LIMIT1 on page 47	AXI limits for port 0 counter 1	Read/write	0x00000000
0x00000048	4.2.16 Register AXI_LIMIT2 on page 48	AXI limits for port 1 counter 2	Read/write	0x00000000
0x0000004C	4.2.17 Register AXI_LIMIT3 on page 49	AXI limits for port 1 counter 3	Read/write	0x00000000

4.2.1 Register ID

The ID register.

The default value of this RO register describes the product version. Please refer to the individual fields for information.

Table 4-2: Register BASE.ID layout

Bits	Link	Name	Usage	Default
[31:28]	arch_major_rev	arch_major_rev	This is the major architecture version number, a in the architecture version a.b	1 (implementation defined)
[27:20]	arch_minor_rev	arch_minor_rev	This is the minor architecture version number, b in the architecture version a.b	1 (implementation defined)
[19:16]	arch_patch_rev	arch_patch_rev	This is the patch number of the architecture version a.b	0 (implementation defined)
[15:12]	product_major	product_major	This is the X-part of the ML00X product number	4 (implementation defined)
[11:8]	version_major	version_major	This is the n for the R-part of an RnPn release number	0x2
[7:4]	version_minor	version_minor	This is the n for the P-part of an RnPn release number	0x0
[3:0]	version_status	version_status	This is the version of the product	1 (implementation defined)

Field arch_major_rev

This is the major architecture version number, a in the architecture version a.b.

arch_major_rev is stored in bits[31:28] and is a 4-bit unsigned integer. Its default value is 1 (implementation defined).

Field arch_minor_rev

This is the minor architecture version number, b in the architecture version a.b.

arch_minor_rev is stored in bits[27:20] and is an 8-bit unsigned integer. Its default value is 0 (implementation defined).

Field arch_patch_rev

This is the patch number of the architecture version a.b.

arch_patch_rev is stored in bits[19:16] and is a 4-bit unsigned integer. Its default value is 6 (implementation defined).

Field product_major

This is the X-part of the ML00X product number.

product_major is stored in bits[15:12] and is a 4-bit unsigned integer. Its default value is 4 (implementation defined).

Field version_major

This is the n for the R-part of an RnPn.

version_major is stored in bits[11:8] and is a 4-bit unsigned integer. Its default value is 0x1.

Field version_minor

This is the n for the P-part of an RnPn.

version_minor is stored in bits[7:4] and is a 4-bit unsigned integer. Its default value is 0x0.

Field version_status

VERSION_STATUS is stored in bits [3:0] and is a 4-bit enumeration.

It contains the status of the NPU release. This status starts at 0 and increases by one for each release.

The named values indicated here are valid for one particular implementation only.

The field can contain the following values:

Table 4-3: VERSION_STATUS values

Value	Name	Meaning
0	BET	Beta

Value	Name	Meaning
1	EAC	Early access
2	REL	REL

4.2.2 Register STATUS

This register describes the current operating status of the NPU.

Table 4-4: Register BASE.STATUS layout

Bits	Link	Name	Usage	Default
[31:16]	irq_history_mask	irq_history_mask	IRQ History mask	0x0
[15:12]	faulting_channel	faulting_channel	Faulting channel on a bus abort. Read: 0=Cmd, 1=IFM, 2=Weights, 3=Scale +Bias, 4=Mem2Mem; Write: 8=OFM, 9=Mem2Mem	0x0
[11]	faulting_interface	faulting_interface	Faulting interface on bus abort. 0=AXI-M0, 1=AXI-M1	0x0
[10:9]	Reserved	-	-	-
[8]	ecc_fault	ecc_fault	ECC state for internal RAMs: 0=no fault, 1=ECC fault signalled. Can only be cleared by reset.	0x0
[7]	Reserved	-	-	-
[6]	pmu_irq_raised	pmu_irq_raised	0=No PMU IRQ, 1=PMU IRQ raised. Cleared by using command register bit 1	0x0
[5]	cmd_end_reached	cmd_end_reached	0=Not reached, 1=Reached. Cleared by writing QBASE or QSIZE when the NPU is in stopped state.	0x0
[4]	cmd_parse_error	cmd_parse_error	0=No error, 1=Command-stream parsing error detected. Can only be cleared by a reset.	0x0
[3]	reset_status	reset_status	Reset is ongoing and only this register can be read (other registers read as 0 and writes are ignored). A value of 0 means the NPU is not being reset and can be accessed as normal.	0x1
[2]	bus_status	bus_status	0=OK, 1=Bus abort detected and processing halted (the NPU has reached IDLE state and does not start to process any more commands/AXI transactions). Can only be cleared by a reset.	0x0
[1]	irq_raised	irq_raised	Raw IRQ status: 0 = IRQ not raised, 1 = IRQ raised. IRQ is cleared using command register bit 1.	0x0
[0]	state	state	NPU state; 0 = Stopped, 1 = Running	stopped

Field irq_history_mask

IRQ History mask.

irq_history_mask is stored in bits[31:16] and is a 16-bit unsigned integer. Its default value is 0x0.

This is used for debug purposes. Each IRQ or Event operation provides a 16-bit mask which is logically ORed into these bits. The bits can be cleared using the command register.

Field faulting_channel

Faulting channel on a bus abort. Read: 0=Cmd, 1=IFM, 2=Weights, 3=Scale+Bias, 4=Mem2Mem; Write: 8=OFM, 9=Mem2Mem.

faulting_channel is stored in bits[15:12] and is a 4-bit unsigned integer. Its default value is 0x0.

Field faulting_interface

Faulting interface on bus abort. 0=AXI-M0, 1=AXI-M1.

faulting_interface is stored in bit[11] and is a 1-bit unsigned integer. Its default value is 0x0.

Field ecc_fault

ECC state for internal RAMs: 0=no fault, 1=ECC fault signalled. Can only be cleared by reset.

ecc_fault is stored in bit[8] and is a 1-bit unsigned integer. Its default value is 0x0.

Field wd_fault

Weight decoder state: 0=no fault, 1=weight decoder decompression fault. Can only be cleared by reset.

wd_fault is stored in bit[7] and is a 1-bit unsigned integer. Its default value is 0x0.

Field pmu_irq_raised

0=No PMU IRQ, 1=PMU IRQ raised. Cleared by using command register bit 1.

pmu_irq_raised is stored in bit[6] and is a 1-bit unsigned integer. Its default value is 0x0.

Field cmd_end_reached

0=Not reached, 1=Reached. Cleared by writing QBASE or QSIZE when the NPU is in stopped state.

cmd_end_reached is stored in bit[5] and is a 1-bit unsigned integer. Its default value is 0x0.

Field cmd_parse_error

0=No error 1=Command stream parsing error detected. Can only be cleared by a reset.

cmd_parse_error is stored in bit[4] and is a 1-bit unsigned integer. Its default value is 0x0.

Field reset_status

Reset is ongoing and only this register can be read (other registers read as 0 and writes are ignored). A value of 0 means the NPU is not being reset and can be accessed as normal.

reset_status is stored in bit[3] and is a 1-bit unsigned integer. Its default value is 0x1.

Field bus_status

0=OK, 1=Bus abort detected and processing halted (the NPU has reached IDLE state and does not start to process any more commands/AXI transactions). Can only be cleared by a reset.

bus_status is stored in bit[2] and is a 1-bit unsigned integer. Its default value is 0x0.

Field irq_raised

Raw IRQ status: 0 = IRQ not raised, 1 = IRQ raised. IRQ is cleared using command register bit 1.

irq_raised is stored in bit[1] and is a 1-bit unsigned integer. Its default value is 0x0.

Field state

NPU state, 0 = Stopped, 1 = Running.

state is stored in bit[0] and is a 1-bit enumeration. Its default value is stopped.

The field can contain the following values:

Table 4-5: Field state values

Value	Name	Meaning
0 (default)	stopped	The NPU is in Stopped state.
1	running	The NPU is in Running state.

4.2.3 Register CMD

The Command register, reads as last written command.

Table 4-6: Register BASE.CMD layout

Bits	Link	Name	Usage	Default
[31:16]	clear_irq_history	clear_irq_history	Clears the IRQ history mask	0x0
[15:4]	Reserved	-	-	-
[3]	power_q_enable	power_q_enable	Write 1 to this bit to enable power off using the Power Q-interface	0x1
[2]	clock_q_enable	clock_q_enable	Write 1 to this bit to enable clock off using the Clock Q-interface and enable the main clock gate	0x1
[1]	clear_irq	clear_irq	Write 1 to clear the IRQ status in the STATUS register. Writing 0 has no effect	0x0
[0]	transition_to_running_state	transition_to_running_state	Write 1 to transition the NPU to running state. Writing 0 has no effect	0x0

Field clear_irq_history

Clears the IRQ history mask.

clear_irq_history is stored in bits[31:16] and is a 16-bit unsigned integer. Its default value is 0x0.

When bit k is set, then the corresponding bit k of the STATUS register (IRQ history) is cleared.

Field power_q_enable

Write 1 to this bit to enable power off using the Power Q-interface.

power_q_enable is stored in bit[3] and is a 1-bit unsigned integer. Its default value is 0x1.

Field clock_q_enable

Write 1 to this bit to enable clock off using the Clock Q-interface and enable the main clock gate.

clock_q_enable is stored in bit[2] and is a 1-bit unsigned integer. Its default value is 0x1.

Field clear_irq

Write 1 to clear the IRQ status in the STATUS register. Writing 0 has no effect.

clear_irq is stored in bit[1] and is a 1-bit unsigned integer. Its default value is 0x0.

Field transition_to_running_state

Write 1 to transition the NPU to running state. Writing 0 has no effect.

transition_to_running_state is stored in bit[0] and is a 1-bit unsigned integer. Its default value is 0x0.

4.2.4 Register RESET

Request Reset and new security mode.

If this register is written to by a permitted master, then the NPU is reset (clearing all internal RAMs) and the reset register value is updated. (Otherwise the write to this register is ignored and the NPU is not reset.)

The value written to this register sets the privilege level used by the NPU when the NPU acts as an AXI master. The host is permitted to set any level of privilege less than or equal to the host privilege level.

Table 4-7: Register BASE.RESET layout

Bits	Link	Name	Usage	Default
[31:2]	Reserved	-		
[1]	pending_CSL	pending_CSL	Current security level: 0=Secure, 1=Non secure	secure
[0]	pending_CPL	pending_CPL	Current privilege level: 0=User, 1=Privileged	user

Field pending_CSL

Current security level 0=Secure, 1=Non secure.

pending_CSL is stored in bit[1] and is a 1-bit enumeration. Its default value is secure.

The field can contain the following values:

Table 4-8: Field pending_CSL values

Value	Name	Meaning
0 (default)	secure	The NPU's security level is configured as Secure.
1	non_secure	The NPU's security level is configured as Non-Secure.

Field pending_CPL

Current privilege level: 0=User, 1=Privileged.

pending_CPL is stored in bit[0] and is a 1-bit enumeration. Its default value is user.

The field can contain the following values:

Table 4-9: Field pending_CPL values

Value	Name	Meaning
0 (default)	user	The NPU is configured for User level access.
1	privileged	The NPU is configured for Privileged level access.

4.2.5 Register QBASE0

The Base address of Command-queue bits[31:0]. The address is 4-byte aligned.

Table 4-10: Register BASE.QBASE0 layout

Bits	Link	Name	Usage	Default
[31:0]	QBASE0	QBASE0	The 4-byte-aligned lower bytes of the base address value for the command stream	0x00000000

Field QBASE0

The 4-byte-aligned lower bytes of the base address value for the command stream.

QBASE0 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00000000.

4.2.6 Register QBASE1

Address extension bits[47:32] for queue base.

Table 4-11: Register BASE.QBASE1 layout

Bits	Link	Name	Usage	Default
[31:0]	QBASE1	QBASE1	The 4-byte-aligned upper bytes of the base address value for the command stream	0x00000000

Field QBASE1

The 4-byte-aligned upper bytes of the base address value for the command stream.

QBASE1 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00000000.

4.2.7 Register QREAD

The Read offset in the command stream in bytes. Multiples of 4 in the range 0-16 MB.

Table 4-12: Register BASE.QREAD layout

Bits	Link	Name	Usage	Default
[31:0]	QREAD	QREAD	The read offset of the current command under execution	0x00000000

Field QREAD

The read offset of the current command under execution.

QREAD is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00000000.

4.2.8 Register QCONFIG

The AXI configuration for the command stream in the range 0-3. Same encoding as for REGIONCFG.

Table 4-13: Register BASE.QCONFIG layout

Bits	Link	Name	Usage	Default
[31:0]	QCONFIG	QCONFIG	AXI configuration for the command stream in the range 0-3	0x00000000

Field QCONFIG

AXI configuration for the command stream in the range 0-3.

QCONFIG is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00000000.

4.2.9 Register QSIZE

Size of the command stream in bytes. Multiples of 4 in the range 0-16 MB.

Table 4-14: Register BASE.QSIZE layout

Bits	Link	Name	Usage	Default
[31:0]	QSIZE	QSIZE	Size of the next command stream to be executed by the NPU	0x00000000

Field QSIZE

Size of the next command stream to be executed by the NPU.

QSIZE is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00000000.

4.2.10 Register PROT

The Protection level configured for the NPU when acting as an AXI master.

Table 4-15: Register BASE.PROT layout

Bits	Link	Name	Usage	Default
[31:2]	Reserved	-	-	-
[1]	active_CSL	active_CSL	Current security level: 0=Secure, 1=Non-secure	Dependent on PORSL
[0]	active_CPL	active_CPL	Current privilege level: 0=User, 1=Privileged	Dependent on PORPL

Field active_CSL

Current security level 0=Secure, 1=Non-secure.

active_CSL is stored in bit[1] and is a 1-bit enumeration. Its default value is dependent on PORSL .

This is used as AxPROT[1] when the NPU is a master and set from Pending CSL after the reset is complete.

- After a hard reset, this is set to Power-on-reset security level (PORSL), which allows for CPUs that do not support TrustZone.
- After a soft reset, this is set to pending_CSL, if PPROT[1]==0, otherwise it is set to 1. For this to be effective, there must be a memory-protection controller included in the system (not part of Ethos™-U55).

The field can contain the following values:

Table 4-16: Field active_CSL values

Value	Name	Meaning
0 (default)	secure	The NPU security level is configured as Secure.
1	non_secure	The NPU security level is configured as Non-Secure.

Field active_CPL

Current privilege level 0=User, 1=Privileged.

active_CPL is stored in bit[0] and is a 1-bit enumeration. Its default value is dependent on PORPL.

This is used as AxPROT[0] when the NPU is a master.

- After hard reset, this is set to Power-on-reset privilege level (PORPL).
- After soft reset, this is set to pending_CPL, if PPROT[0]==1, otherwise it is set to 0. For this to be effective, there must be a system-level MPU built for the system (not part of the Ethos™-U55 deliverables).

The field can contain the following values:

Table 4-17: Field active_CPL values

Value	Name	Meaning
0 (default)	user	The NPU is configured for User-level access.
1	privileged	The NPU is configured for Privileged-level access.

4.2.11 Register CONFIG

The RTL configuration register.

The default value of this RO register describes the NPU configuration. Please refer to the individual fields for information.

Table 4-18: Register BASE.CONFIG layout

Bits	Link	Name	Usage	Default
[31:28]	product	product	Product configuration	0 (implementation defined)
[27]	custom_dma	custom_dma	Custom DMA configuration	-
[26:16]	Reserved	-	-	-
[15:8]	shram_size	shram_size	Size in KB of SHRAM in the range 8-48.	-
[7:4]	cmd_stream_version	cmd_stream_version	Command-stream version accepted by this NPU.	0x0
[3:0]	macs_per_cc	macs_per_cc	The log2(macs/clock cycle). Valid encoding range is 5-8 for 32-256 MACs/clock cycle.	-

Field product

Product configuration.

product is stored in bits[31:28] and is a 4-bit unsigned integer. Its default value is 0 (implementation defined).

Field custom_dma

Custom DMA configuration.

custom_dma is stored in bit[27] and is a 1-bit enumeration of type config_custom_dma.

The field can contain the following values:

Table 4-19: Field custom_dma values

Value	Name	Meaning
0	custom_dma_not_implemented	Custom DMA feature not implemented.
1	custom_dma_implemented	Custom DMA feature implemented.

Field shram_size

Size in KB of SHRAM in the range 8-48.

shram_size is stored in bits[15:8] and is an 8-bit enumeration.

The field can contain the following values:

Table 4-20: Field shram_size values

Value	Name	Meaning
0x30	SHRAM_48kB	The available SHRAM is 48 kBytes.
0x18	SHRAM_24kB	The available SHRAM is 24 kBytes.
0x10	SHRAM_16kB	The available SHRAM is 16 kBytes.

Field cmd_stream_version

Command-stream version accepted by this NPU.

cmd_stream_version is stored in bits[7:4] and is a 4-bit unsigned integer. Its default value is 0x0.

Field macs_per_cc

The log2(macs/clock cycle). Valid encoding range is 5-8 for 32-256 MACs/clock cycle (each MAC is an 8-bit x 8-bit MAC).

macs_per_cc is stored in bits[3:0] and is a 4-bit enumeration.

The field can contain the following values:

Table 4-21: Field macs_per_cc values

Value	Name	Meaning
0x5	Macs_per_cc_is_5	The number of MACs per clock cycle is 2^5.
0x6	Macs_per_cc_is_6	The number of MACs per clock cycle is 2^6.
0x7	Macs_per_cc_is_7	The number of MACs per clock cycle is 2^7.
0x8	Macs_per_cc_is_8	The number of MACs per clock cycle is 2^8.

4.2.12 Register LOCK

The Lock register. This register is designed for driver use and does not affect NPU functionality.

This register holds a 32-bit value which is cleared to 0 on a reset. The register has special write semantics. Suppose the current register value is "c" and the newly written register value is "w":

If (c==0 or w==0), then the register is updated to the newly written value w.

Otherwise the write is ignored and the value remains unchanged.

- To try to claim the lock, write a nonzero ID value and read back to see if the value was accepted.
- To release the lock (that contains your nonzero ID value), write the value 0 to the lock register.

Table 4-22: Register BASE.LOCK layout

Bits	Link	Name	Usage	Default
[31:0]	LOCK	LOCK	32-bit value for the LOCK configuration	0x00000000

Field LOCK

32-bit value for the LOCK configuration.

LOCK is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00000000.

4.2.13 Register REGIONCFG

Region memory type configuration. Bits[2*k+1:2*k] give the memory type for REGION[k].

Table 4-23: Register BASE.REGIONCFG layout

Bits	Link	Name	Usage	Default
[31:16]	Reserved	-	-	-
[15:14]	region7	region7	Bits for the Region7 configuration	axi0_outstanding_counter0
[13:12]	region6	region6	Bits for the Region6 configuration	axi0_outstanding_counter0
[11:10]	region5	region5	Bits for the Region5 configuration	axi0_outstanding_counter0
[9:8]	region4	region4	Bits for the Region4 configuration	axi0_outstanding_counter0
[7:6]	region3	region3	Bits for the Region3 configuration	axi0_outstanding_counter0
[5:4]	region2	region2	Bits for the Region2 configuration	axi0_outstanding_counter0
[3:2]	region1	region1	Bits for the Region1 configuration	axi0_outstanding_counter0
[1:0]	region0	region0	Bits for the Region0 configuration	axi0_outstanding_counter0

Field region7

Bits for the Region7 configuration.

region7 is stored in bits[15:14] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-24: Field region7 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

Field region6

Bits for the Region6 configuration.

region6 is stored in bits[13:12] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-25: Field region6 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.

Value	Name	Meaning
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

Field region5

Bits for the Region5 configuration.

region5 is stored in bits[11:10] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-26: Field region5 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

Field region4

Bits for the Region4 configuration.

region4 is stored in bits[9:8] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-27: Field region4 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

Field region3

Bits for the Region3 configuration.

region3 is stored in bits[7:6] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-28: Field region3 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.

Value	Name	Meaning
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

Field region2

Bits for the Region2 configuration.

region2 is stored in bits[5:4] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-29: Field region2 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

Field region1

Bits for the Region1 configuration.

region1 is stored in bits[3:2] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-30: Field region1 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

Field region0

Bits for the Region0 configuration.

region0 is stored in bits[1:0] and is a 2-bit enumeration. Its default value is axi0_outstanding_counter0.

The field can contain the following values:

Table 4-31: Field region0 values

Value	Name	Meaning
0 (default)	axi0_outstanding_counter0	AXI0 port, outstanding counter 0. AXI limits set by the AXI_LIMIT0 register.

Value	Name	Meaning
1	axi0_outstanding_counter1	AXI0 port, outstanding counter 1. AXI limits set by the AXI_LIMIT1 register.
2	axi1_outstanding_counter2	AXI1 port, outstanding counter 2. AXI limits set by the AXI_LIMIT2 register.
3	axi1_outstanding_counter3	AXI1 port, outstanding counter 3. AXI limits set by the AXI_LIMIT3 register.

4.2.14 Register AXI_LIMIT0

The AXI limits for port 0 counter 0.

Table 4-32: Register BASE.AXI_LIMIT0 layout

Bits	Link	Name	Usage	Default
[31:24]	max_outstanding_write_m1	max_outstanding_write_m1	Maximum number of outstanding AXI write transactions - 1 in range 0-15	0x00
[23:16]	max_outstanding_read_m1	max_outstanding_read_m1	Maximum number of outstanding AXI read transactions - 1 in range 0-31	0x00
[15:8]	Reserved	-	-	-
[7:4]	memtype	memtype	Memtype	-
[3:2]	Reserved	-	-	-
[1:0]	max_beats	max_beats	Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved	0x0

Field max_outstanding_write_m1

Maximum number of outstanding AXI write transactions - 1 in range 0-15.

max_outstanding_write_m1 is stored in bits[31:24] and is an 8-bit unsigned integer. Its default value is 0x00.

Field max_outstanding_read_m1

Maximum number of outstanding AXI read transactions - 1 in range 0-31.

max_outstanding_read_m1 is stored in bits[23:16] and is an 8-bit unsigned integer. Its default value is 0x00.

Field memtype

Memtype is used to encode AxCACHE signals.

BASE.AXI_LIMIT0.memtype is stored in bits[7:4] and is a 4-bit enumeration of type axi_mem_encodign_type. Its default value is Device_Non_Bufferable.

The field can contain the following values:

Table 4-33: Field memtype values

Value	Name	Meaning
0x0 (default)	Device_Non_Bufferable	ARCACHE=0000, AWCACHE=0000
0x1	Device_Bufferable	ARCACHE=0001, AWCACHE=0001

Value	Name	Meaning
0x2	Normal_Non_cacheable_Non_bufferable	ARCACHE=0010, AWCACHE=0010
0x3	Normal_Non_cacheable_Bufferable	ARCACHE=0011, AWCACHE=0011
0x4	Write_through_No_allocate	ARCACHE=1010, AWCACHE=0110
0x5	Write_through_Read_allocate	ARCACHE=1110, AWCACHE=0110
0x6	Write_through_Write_allocate	ARCACHE=1010, AWCACHE=1110
0x7	Write_through_Read_and_Write_allocate	ARCACHE=1110, AWCACHE=1110
0x8	Write_back_No_allocate	ARCACHE=1011, AWCACHE=0111
0x9	Write_back_Read_allocate	ARCACHE=1111, AWCACHE=0111
0xA	Write_back_Write_allocate	ARCACHE=1011, AWCACHE=1111
0xB	Write_back_Read_and_Write_allocate	ARCACHE=1111, AWCACHE=1111
0xC - 0xF	Reserved_12_15	Reserved

Field max_beats

Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved.

max_beats is stored in bits[1:0] and is a 2-bit unsigned integer. Its default value is 0x0.

4.2.15 Register AXI_LIMIT1

The AXI limits for port 0 counter 1.

Table 4-34: Register BASE.AXI_LIMIT1 layout

Bits	Link	Name	Usage	Default
[31:24]	max_outstanding_write_m1	max_outstanding_write_m1	Maximum number of outstanding AXI write transactions - 1 in range 0-15	0x00
[23:16]	max_outstanding_read_m1	max_outstanding_read_m1	Maximum number of outstanding AXI read transactions - 1 in range 0-31	0x00
[15:8]	Reserved	-	-	-
[7:4]	memtype	memtype	Memtype	-
[3:2]	Reserved	-	-	-
[1:0]	max_beats	max_beats	Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved	0x0

Field max_outstanding_write_m1

Maximum number of outstanding AXI write transactions - 1 in range 0-15.

max_outstanding_write_m1 is stored in bits[31:24] and is an 8-bit unsigned integer. Its default value is 0x00.

Field max_outstanding_read_m1

Maximum number of outstanding AXI read transactions - 1 in range 0-31.

max_outstanding_read_m1 is stored in bits[23:16] and is an 8-bit unsigned integer. Its default value is 0x00.

Field memtype

Memtype.

memtype is stored in bits[7:4] and is a 4-bit unsigned integer.

Field max_beats

Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved.

max_beats is stored in bits[1:0] and is a 2-bit unsigned integer. Its default value is 0x0.

4.2.16 Register AXI_LIMIT2

The AXI limits for port 1 counter 2.

Table 4-35: Register BASE.AXI_LIMIT2 layout

Bits	Link	Name	Usage	Default
[31:24]	max_outstanding_write_m1	max_outstanding_write_m1	Maximum number of outstanding AXI write transactions - 1 in range 0-15	0x00
[23:16]	max_outstanding_read_m1	max_outstanding_read_m1	Maximum number of outstanding AXI read transactions - 1 in range 0-31	0x00
[15:8]	Reserved	-	-	-
[7:4]	memtype	memtype	Memtype	-
[3:2]	Reserved	-	-	-
[1:0]	max_beats	max_beats	Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved	0x0

Field max_outstanding_write_m1

Maximum number of outstanding AXI write transactions - 1 in range 0-15.

max_outstanding_write_m1 is stored in bits[31:24] and is an 8-bit unsigned integer. Its default value is 0x00.

Field max_outstanding_read_m1

Maximum number of outstanding AXI read transactions - 1 in range 0-31.

max_outstanding_read_m1 is stored in bits[23:16] and is an 8-bit unsigned integer. Its default value is 0x00.

Field memtype

Memtype.

memtype is stored in bits[7:4] and is a 4-bit unsigned integer.

Field max_beats

Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved.

max_beats is stored in bits[1:0] and is a 2-bit unsigned integer. Its default value is 0x0.

4.2.17 Register AXI_LIMIT3

The AXI limits for port 1 counter 3.

Table 4-36: Register BASE.AXI_LIMIT3 layout

Bits	Link	Name	Usage	Default
[31:24]	max_outstanding_write_m1	max_outstanding_write_m1	Maximum number of outstanding AXI write transactions - 1 in range 0-15	0x00
[23:16]	max_outstanding_read_m1	max_outstanding_read_m1	Maximum number of outstanding AXI read transactions - 1 in range 0-31	0x00
[15:8]	Reserved	-	-	-
[7:4]	memtype	memtype	Memtype	-
[3:2]	Reserved	-	-	-
[1:0]	max_beats	max_beats	Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved	0x0

Field max_outstanding_write_m1

Maximum number of outstanding AXI write transactions - 1 in range 0-15.

max_outstanding_write_m1 is stored in bits[31:24] and is an 8-bit unsigned integer. Its default value is 0x00.

Field max_outstanding_read_m1

Maximum number of outstanding AXI read transactions - 1 in range 0-31.

max_outstanding_read_m1 is stored in bits[23:16] and is an 8-bit unsigned integer. Its default value is 0x00.

Field memtype

Memtype.

memtype is stored in bits[7:4] and is a 4-bit unsigned integer.

Field max_beats

Burst-split alignment: 0=64 bytes, 1=128 bytes, 2=256 bytes, 3=reserved.

max_beats is stored in bits[1:0] and is a 2-bit unsigned integer. Its default value is 0x0.

4.3 Register page BASE_POINTERS

The NPU base-pointer registers bank.

Table 4-37: BASE_POINTERS registers

Address	Link	Usage	Access	Default
0x00000080	4.3.1 Register BASEP0 on page 50	Lower 32 bits of the Base pointer for region index 0	Read/write	0x00000000
0x00000084	4.3.2 Register BASEP1 on page 50	Upper 32 bits of the Base pointer for region index 0	Read/write	0x00000000
0x00000088	4.3.3 Register BASEP2 on page 51	Lower 32 bits of the Base pointer for region index 1	Read/write	0x00000000
0x0000008C	4.3.4 Register BASEP3 on page 51	Upper 32 bits of the Base pointer for region index 1	Read/write	0x00000000
0x00000090	4.3.5 Register BASEP4 on page 51	Lower 32 bits of the Base pointer for region index 2	Read/write	0x00000000
0x00000094	4.3.6 Register BASEP5 on page 52	Upper 32 bits of the Base pointer for region index 2	Read/write	0x00000000
0x00000098	4.3.7 Register BASEP6 on page 52	Lower 32 bits of the Base pointer for region index 3	Read/write	0x00000000
0x0000009C	4.3.8 Register BASEP7 on page 52	Upper 32 bits of the Base pointer for region index 3	Read/write	0x00000000
0x000000A0	4.3.9 Register BASEP8 on page 53	Lower 32 bits of the Base pointer for region index 4	Read/write	0x00000000
0x000000A4	4.3.10 Register BASEP9 on page 53	Upper 32 bits of the Base pointer for region index 4	Read/write	0x00000000
0x000000A8	4.3.11 Register BASEP10 on page 53	Lower 32 bits of the Base pointer for region index 5	Read/write	0x00000000
0x000000AC	4.3.12 Register BASEP11 on page 54	Upper 32 bits of the Base pointer for region index 5	Read/write	0x00000000
0x000000B0	4.3.13 Register BASEP12 on page 54	Lower 32 bits of the Base pointer for region index 6	Read/write	0x00000000
0x000000B4	4.3.14 Register BASEP13 on page 54	Upper 32 bits of the Base pointer for region index 6	Read/write	0x00000000
0x000000B8	4.3.15 Register BASEP14 on page 55	Lower 32 bits of the Base pointer for region index 7	Read/write	0x00000000
0x000000BC	4.3.16 Register BASEP15 on page 55	Upper 32 bits of the Base pointer for region index 7	Read/write	0x00000000

4.3.1 Register BASEP0

Lower 32 bits of the Base pointer for region index 0.

Table 4-38: Register BASE_POINTERS.BASEP0 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.2 Register BASEP1

Upper 32 bits of the Base pointer for region index 0.

Table 4-39: Register BASE_POINTERS.BASEP1 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.3 Register BASEP2

Lower 32 bits of the Base pointer for region index 1.

Table 4-40: Register BASE_POINTERS.BASEP2 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.4 Register BASEP3

Upper 32 bits of the Base pointer for region index 1.

Table 4-41: Register BASE_POINTERS.BASEP3 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.5 Register BASEP4

Lower 32 bits of the Base pointer for region index 2.

Table 4-42: Register BASE_POINTERS.BASEP4 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.6 Register BASEP5

Upper 32 bits of the Base pointer for region index 2.

Table 4-43: Register BASE_POINTERS.BASEP5 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.7 Register BASEP6

Lower 32 bits of the Base pointer for region index 3.

Table 4-44: Register BASE_POINTERS.BASEP6 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.8 Register BASEP7

Upper 32 bits of the Base pointer for region index 3.

Table 4-45: Register BASE_POINTERS.BASEP7 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.9 Register BASEP8

Lower 32 bits of the Base pointer for region index 4.

Table 4-46: Register BASE_POINTERS.BASEP8 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.10 Register BASEP9

Upper 32 bits of the Base pointer for region index 4.

Table 4-47: Register BASE_POINTERS.BASEP9 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.11 Register BASEP10

Lower 32 bits of the Base pointer for region index 5.

Table 4-48: Register BASE_POINTERS.BASEP10 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.12 Register BASEP11

Upper 32 bits of the Base pointer for region index 5.

Table 4-49: Register BASE_POINTERS.BASEP11 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.13 Register BASEP12

Lower 32 bits of the Base pointer for region index 6.

Table 4-50: Register BASE_POINTERS.BASEP12 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.14 Register BASEP13

Upper 32 bits of the Base pointer for region index 6.

Table 4-51: Register BASE_POINTERS.BASEP13 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.15 Register BASEP14

Lower 32 bits of the Base pointer for region index 7.

Table 4-52: Register BASE_POINTERS.BASEP14 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The low word of the 64-bit address

Field addr_word

The low word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.3.16 Register BASEP15

Upper 32 bits of the Base pointer for region index 7.

Table 4-53: Register BASE_POINTERS.BASEP15 layout

Bits	Link	Name	Usage
[31:0]	addr_word	addr_word	The high word of the 64-bit address

Field addr_word

The high word of the 64-bit address.

addr_word is stored in bits[31:0] and is a 32-bit unsigned integer.

4.4 Register page ID

The NPU ID-bytes register bank.

Table 4-54: ID registers

Address	Link	Usage	Access	Default
0x00000FD0	4.4.1 Register PID4 on page 56	Peripheral ID byte 4 (Arm®=code 4)	Read-only	0x00000004
0x00000FD4	4.4.2 Register PID5 on page 56	Peripheral ID byte 5 (reserved)	Read-only	0x00000000
0x00000FD8	4.4.3 Register PID6 on page 57	Peripheral ID byte 6 (reserved)	Read-only	0x00000000
0x00000FDC	4.4.4 Register PID7 on page 57	Peripheral ID byte 7 (reserved)	Read-only	0x00000000
0x00000FE0	4.4.5 Register PID0 on page 57	Peripheral ID byte 0. This is bits[7:0] of the part number.	Read-only	0x00000080
0x00000FE4	4.4.6 Register PID1 on page 58	Peripheral ID byte 1. This is bits[11:8] of the part number in bits[3:0], and bits[3:0] of the Arm® ID in bits[7:4].	Read-only	0x000000B5
0x00000FE8	4.4.7 Register PID2 on page 58	Peripheral ID byte 2. This is bits[6:4] of the Arm® ID in bits[2:0], and bit 3 indicates format B.	Read-only	0x0000000B
0x00000FEC	4.4.8 Register PID3 on page 58	Peripheral ID byte 3.	Read-only	0x00000000
0x00000FF0	4.4.9 Register CID0 on page 59	Component ID byte 0.	Read-only	0x0000000D
0x00000FF4	4.4.10 Register CID1 on page 59	Component ID byte 1.	Read-only	0x000000F0
0x00000FF8	4.4.11 Register CID2 on page 59	Component ID byte 2.	Read-only	0x00000005
0x00000FFC	4.4.12 Register CID3 on page 60	Component ID byte 3.	Read-only	0x000000B1

4.4.1 Register PID4

Peripheral ID byte 4 (Arm®=code 4).

Table 4-55: Register ID.PID4 layout

Bits	Link	Name	Usage	Default
[31:0]	PID4	PID4	Byte 4 of the Peripheral ID (Lower 8 bits valid)	0x04

Field PID4

Byte 4 of the Peripheral ID (Lower 8 bits valid).

PID4 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x04.

4.4.2 Register PID5

Peripheral ID byte 5 (reserved).

Table 4-56: Register ID.PID5 layout

Bits	Link	Name	Usage	Default
[31:0]	PID5	PID5	Byte 5 of the Peripheral ID (Lower 8 bits valid)	0x00

Field PID5

Byte 5 of the Peripheral ID (Lower 8 bits valid).

PID5 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00.

4.4.3 Register PID6

Peripheral ID byte 6 (reserved).

Table 4-57: Register ID.PID6 layout

Bits	Link	Name	Usage	Default
[31:0]	PID6	PID6	Byte 6 of the Peripheral ID (Lower 8 bits valid)	0x00

Field PID6

Byte 6 of the Peripheral ID (Lower 8 bits valid).

PID6 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00.

4.4.4 Register PID7

Peripheral ID byte 7 (reserved).

Table 4-58: Register ID.PID7 layout

Bits	Link	Name	Usage	Default
[31:0]	PID7	PID7	Byte 7 of the Peripheral ID (Lower 8 bits valid)	0x00

Field PID7

Byte 7 of the Peripheral ID (Lower 8 bits valid).

PID7 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00.

4.4.5 Register PID0

Peripheral ID byte 0. This is bits[7:0] of the part number.

Table 4-59: Register ID.PID0 layout

Bits	Link	Name	Usage	Default
[31:0]	PID0	PID0	Byte 0 of the Peripheral ID (Lower 8 bits valid)	0x80

Field PID0

Byte 0 of the Peripheral ID (Lower 8 bits valid).

PID0 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x80.

4.4.6 Register PID1

Peripheral ID byte 1. This is bits[11:8] of the part number in bits[3:0] and bits[3:0] of the Arm® ID in bits[7:4].

Table 4-60: Register ID.PID1 layout

Bits	Link	Name	Usage	Default
[31:0]	PID1	PID1	Byte 1 of the Peripheral ID (Lower 8 bits valid)	0xB5

Field PID1

Byte 1 of the Peripheral ID (Lower 8 bits valid).

PID1 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0xB5.

4.4.7 Register PID2

Peripheral ID byte 2. This is bits[6:4] of the Arm® ID in bits[2:0] and bit 3 indicates format B.

Table 4-61: Register ID.PID2 layout

Bits	Link	Name	Usage	Default
[31:0]	PID2	PID2	Byte 2 of the Peripheral ID (Lower 8 bits valid)	0x0B

Field PID2

Byte 2 of the Peripheral ID (Lower 8 bits valid).

PID2 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x0B.

4.4.8 Register PID3

Peripheral ID byte 3.

Table 4-62: Register ID.PID3 layout

Bits	Link	Name	Usage	Default
[31:0]	PID3	PID3	Byte 1 of the Peripheral ID (Lower 8 bits valid)	0x0

Field PID3

Byte 1 of the Peripheral ID (Lower 8 bits valid).

PID3 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x0.

4.4.9 Register CID0

Component ID byte 0.

Table 4-63: Register ID.CID0 layout

Bits	Link	Name	Usage	Default
[31:0]	CID0	CID0	Byte 0 of the Component ID (Lower 8 bits valid)	0x0D

Field CID0

Byte 0 of the Component ID (Lower 8 bits valid).

CID0 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x0D.

4.4.10 Register CID1

Component ID byte 1.

Table 4-64: Register ID.CID1 layout

Bits	Link	Name	Usage	Default
[31:0]	CID1	CID1	Byte 1 of the Component ID (Lower 8 bits valid)	0xF0

Field CID1

Byte 1 of the Component ID (Lower 8 bits valid).

CID1 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0xF0.

4.4.11 Register CID2

Component ID byte 2.

Table 4-65: Register ID.CID2 layout

Bits	Link	Name	Usage	Default
[31:0]	CID2	CID2	Byte 2 of the Component ID (Lower 8 bits valid)	0x05

Field CID2

Byte 2 of the Component ID (Lower 8 bits valid).

CID2 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x05.

4.4.12 Register CID3

Component ID byte 3.

Table 4-66: Register ID.CID3 layout

Bits	Link	Name	Usage	Default
[31:0]	CID3	CID3	Byte 3 of the Component ID (Lower 8 bits valid)	0xB1

Field CID3

Byte 3 of the Component ID (Lower 8 bits valid).

CID3 is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0xB1.

4.5 Register page PMU

The Performance Monitoring Unit (PMU) control registers.

The PMU consists of a 48-bit cycle counter that can be enabled or disabled, reset, and read through APB. Also, there are programmable event counters controlled through APB.

The PMU has four event counters that log AXI-related events to monitor system performance. It can be configured to generate an interrupt on counter overflow. There is also an option to control the PMU through a command-stream operation.

Note

The PMU uses the NPU clock after the top-level clock gate to count cycles. To get non-gated clock cycles, the NPU clock must be forced. To force the NPU clock gate, set bit[2] of the CMD register to LOW to disable clock-off through the QLPI interface and the main clock gate.

Table 4-67: PMU registers

Address	Link	Usage	Access	Default
0x0180	4.5.1 Register PMCR on page 61	PMU master control register	Read/ write	0x00002000
0x0184	4.5.2 Register PMCNTENSET on page 62	Count-enable set register	Read/ write	0x00000000
0x0188	4.5.3 Register PMCNTENCLR on page 64	Count-enable clear register	Read/ write	0x00000000
0x018C	4.5.6 Register PMOVSSET on page 68	Overflow-flag status set register	Read/ write	0x00000000
0x0190	4.5.7 Register PMOVSCLR on page 70	Overflow-flag status clear register	Read/ write	0x00000000
0x0194	4.5.8 Register PMINTSET on page 71	Interrupt-enable set register	Read/ write	0x00000000
0x0198	4.5.9 Register PMINTCLR on page 73	Interrupt-enable clear register	Read/ write	0x00000000
0x01A0	4.5.10 Register PMCCNTR_LO on page 75	Performance-monitor cycle count low register	Read/ write	0x00000000
0x01A4	4.5.11 Register PMCCNTR_HI on page 76	Performance-monitor cycle count high register	Read/ write	0x00000000
0x01AC	4.5.12 Register PMCAXI_CHAN on page 76	Set which AXI channel monitor	Read/ write	0x00000000
0x0300	4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66	Performance-monitor event counters	Read/ write	0x00000000
0x0380	4.5.5 PMU_EVTYPER0 ... PMU_EVTYPER3 on page 66	Performance-monitor event-type control counters	Read/ write	0x00000000

4.5.1 Register PMCR

The PMCR register is the master control register of the PMU.

Table 4-68: Register PMU.PMCR layout

Bits	Link	Name	Usage	Default
[31:16]	Reserved	-	-	-
[15:11]	num_event_cnt	num_event_cnt	Number of event counters available for performance measurement	0x04
[10:4]	Reserved	-	-	-
[3]	mask_en	mask_en	PMU can be enabled/disabled by command stream operation NPU_OP_PMU_MASK	0x0
[2]	cycle_cnt_rst	cycle_cnt_rst	Reset cycle counter	0
[1]	event_cnt_rst	event_cnt_rst	Reset event counter	0
[0]	cnt_en	cnt_en	Enable counter	0x0

Field num_event_cnt

Number of event counters available for performance management.

num_event_cnt is stored in bits[15:11] and is a 5-bit unsigned integer. Its default value is 0x04.

The number of available event counters is hard-coded to four.

Field mask_en

PMU can be enabled/disabled by command stream operation NPU_OP_PMU_MASK.

mask_en is stored in bit[3] and is a 1-bit unsigned integer. Its default value is 0x0.

Note that field cnt_en must be enabled for the PMU to be active.

Field cycle_cnt_rst

Reset cycle counter.

cycle_cnt_rst is located in bit[2] and is a 1-bit unsigned integer. Its default value is 0.

Writing a 1 to this register resets the cycle counter. If the cycle counter is active, it will continue counting after reset. This register bit always reads a 0.

Field event_cnt_rst

Reset event counter.

event_cnt_rst is located in bit[1] and is a 1-bit unsigned integer. Its default value is 0.

Writing a 1 to this field resets all event counters. If any counter is active, it will continue counting after reset. This register bit always reads a 0.

Field cnt_en

Enable counter.

cnt_en is stored in bit[0] and is a 1-bit unsigned integer. Its default value is 0x0.

This is the master switch. When the switch is disabled, the PMU is always off.

4.5.2 Register PMCNTENSET

Count-enable set registers to activate the counters.

This register enables the dedicated cycle counter, PMCCNTR, and any implemented event counters PMU_EVCNTRn.

4.5.2 Register PMCNTENSET on page 62 is used together with the 4.5.3 Register PMCNTENCLR on page 64 register. It is implemented in hardware with the same underlying state as the 4.5.3 Register PMCNTENCLR on page 64.

Writing to this register enables the counters as follows: writing 1 to bit[31] enables the cycle counter and writing 1 to bit[0-3] enables event counter 0-3, respectively.

Reading from 4.5.2 Register PMCNTENSET on page 62 or 4.5.3 Register PMCNTENCLR on page 64 gives the same value, which is the enable status of the counters.

Table 4-69: Register PMU.PMCNTENSET layout

Bits	Link	Name	Usage	Default
[31]	CYCLE_CNT	CYCLE_CNT	PMCCNTR enable bit	0
[30:4]	Reserved	-	-	-
[3]	EVENT_CNT_3	EVENT_CNT_3	Event-counter enable bit for PMU_EVCNTR3	0
[2]	EVENT_CNT_2	EVENT_CNT_2	Event-counter enable bit for PMU_EVCNTR2	0
[1]	EVENT_CNT_1	EVENT_CNT_1	Event-counter enable bit for PMU_EVCNTR1	0
[0]	EVENT_CNT_0	EVENT_CNT_0	Event-counter enable bit for PMU_EVCNTR0	0

Field CYCLE_CNT

PMCCNTR enable bit.

CYCLE_CNT is stored in bit[31] and is a 1-bit flag. Its default value is 0.

Enables the dedicated cycle counter, PMCCNTR.

Table 4-70: Field CYCLE_CNT values

Value	Meaning
0 (default)	When read, it means the cycle counter is disabled. When written, it has no effect.
1	When read, it means the cycle counter is enabled. When written, it enables the cycle counter.

Field EVENT_CNT_3

Event-counter enable bit for PMU_EVCNTR3.

EVENT_CNT_3 is stored in bit[3] and is a 1-bit flag. Its default value is 0.

Table 4-71: Field EVENT_CNT_3 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event counter is enabled. When written, it enables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

Field EVENT_CNT_2

Event-counter enable bit for PMU_EVCNTR2.

EVENT_CNT_2 is stored in bit[2] and is a 1-bit flag. Its default value is 0.

Table 4-72: Field EVENT_CNT_2 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.

Value	Meaning
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event counter is enabled. When written, it enables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

Field EVENT_CNT_1

Event-counter enable bit for PMU_EVCNTR1.

EVENT_CNT_1 is stored in bit[1] and is a 1-bit flag. Its default value is 0.

Table 4-73: Field EVENT_CNT_1 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event counter is enabled. When written, it enables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

Field EVENT_CNT_0

Event-counter enable bit for PMU_EVCNTR0.

EVENT_CNT_0 is stored in bit[0] and is a 1-bit flag. Its default value is 0.

Table 4-74: Field EVENT_CNT_0 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event counter is enabled. When written, it enables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

4.5.3 Register PMCNTENCLR

Count-enable clear registers to disable the counters.

This register disables the dedicated cycle counter, PMCCNTR, and any implemented event counters PMU_EVCNTRn.

4.5.3 Register PMCNTENCLR on page 64 is used together with the 4.5.2 Register PMCNTENSET on page 62 register. It is implemented in hardware with the same underlying state as 4.5.2 Register PMCNTENSET on page 62.

Writing to this register disables the counters as follows: writing 1 to bit[31] disables the cycle counter and writing 1 to bit[0-3] disables event counter 0-3, respectively.

Reading from 4.5.2 Register PMCNTENSET on page 62 or 4.5.3 Register PMCNTENCLR on page 64 gives the same value, which is the enable status of the counters.

Table 4-75: Register PMU.PMCNTENCLR layout

Bits	Link	Name	Usage	Default
[31]	CYCLE_CNT	CYCLE_CNT	PMCCNTR disable bit	0
[30:4]	Reserved	-	-	-
[3]	EVENT_CNT_3	EVENT_CNT_3	Event-counter disable bit for PMU_EVCNTR3	0
[2]	EVENT_CNT_2	EVENT_CNT_2	Event-counter disable bit for PMU_EVCNTR2	0
[1]	EVENT_CNT_1	EVENT_CNT_1	Event-counter disable bit for PMU_EVCNTR1	0
[0]	EVENT_CNT_0	EVENT_CNT_0	Event-counter disable bit for PMU_EVCNTR0	0

Field CYCLE_CNT

PMCCNTR disable bit.

CYCLE_CNT is stored in bit[31] and is a 1-bit flag. Its default value is 0.

Disables the dedicated cycle counter, PMCCNTR.

Table 4-76: Field CYCLE_CNT values

Value	Meaning
0 (default)	When read, it means the cycle counter is disabled. When written, it has no effect.
1	When read, it means the cycle counter is enabled. When written, it disables the cycle counter.

Field EVENT_CNT_3

Event-counter disable bit for PMU_EVCNTR3.

EVENT_CNT_3 is stored in bit[3] and is a 1-bit flag. Its default value is 0.

Table 4-77: Field EVENT_CNT_3 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is enabled. When written, it disables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

Field EVENT_CNT_2

Event-counter disable bit for PMU_EVCNTR2.

EVENT_CNT_2 is stored in bit[2] and is a 1-bit flag. Its default value is 0.

Table 4-78: Field EVENT_CNT_2 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is enabled. When written, it disables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

Field EVENT_CNT_1

Event-counter disable bit for PMU_EVCNTR1.

EVENT_CNT_1 is stored in bit[1] and is a 1-bit flag. Its default value is 0.

Table 4-79: Field EVENT_CNT_1 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is enabled. When written, it disables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

Field EVENT_CNT_0

Event-counter disable bit for PMU_EVCNTR0.

EVENT_CNT_0 is stored in bit[0] and is a 1-bit flag. Its default value is 0.

Table 4-80: Field EVENT_CNT_0 values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is disabled. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 is enabled. When written, it disables 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66.

4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3

Performance-monitor event counters.

PMU_EVCNTR[k]: these are the four 32-bit performance counters (k=0-3).

4.5.5 PMU_EVTYPER0 ... PMU_EVTYPER3

The performance-monitor event-type counters controlling the respective event counters.

PMU_EVTYPER0 ... PMU_EVTYPER3 are the events that are connected to performance counters 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66, where PMU_EVTYPER[k] controls performance counter PMU_EVCNTR[k].

An event is selected using a 10-bit word from the following table.

Field EV_TYPE

Event type.

EV_TYPE is stored is a 10-bit enumeration. Its default value is no_event.

The field can contain the following values:

Table 4-81: Field EV_TYPE values

Value	Name	Meaning
0x00 (default)	no_event	No event counted (the event never occurs)
0x11	Cycle	Event occurs every cycle.
0x20	NPU idle	NPU in stopped state
0x23	NPU running	NPU in running state
0x30	MAC: ACTIVE (8 or 16 bit)	MAC is doing block traversal. Valid blk_cmd and not stalled.
0x31	MAC: ACTIVE 8-bit	MAC is doing 8-bit block traversal. Valid blk_cmd and not stalled
0x32	MAC: ACTIVE 16-bit	MAC is doing 16-bit block traversal. Valid blk_cmd and not stalled
0x40	AO: ACTIVE (8-bit or 16-bit)	AO is doing block traversal of ACC or IB. Valid blk_cmd and not stalled
0x41	AO: ACTIVE 8-bit	AO is doing 8-bit block traversal of ACC or IB. Valid blk_cmd and not stalled
0x42	AO: ACTIVE 16-bit	AO is doing 16-bit block traversal of ACC or IB. Valid blk_cmd and not stalled
0x50	WD: ACTIVE	WD is decoding weight stream. Valid ofd_cmd and not stalled.
0x80	axi0_rd_trans_accepted	AXI-0 read transfer accepted, arready & arvalid (number of read transfers)
0x81	-	-
0x82	axi0_rd_data_beat_received	AXI-0, rready & rvalid (read bandwidth)
0x83	axi0_rd_tran_req_stalled	AXI-0, arvalid & ~arready (read stalls due memory system)
0x84	axi0_wr_trans_accepted	AXI0, awready & awvalid (number write transfers)
0x85-0x86	-	-
0x87	axi0_wr_data_beat_written	AXI-0, wvalid wready (write bandwidth)
0x88	axi0_wr_tran_req_stalled	AXI-0, awvalid & ~awready (write transfer stalls due to memory system)
0x89	axi0_wr_data_beat_stalled	AXI-0, wvalid & ~wready (write beat stalls due to memory system)
0x8A-0x8B	-	-
0x8C	axi0_enabled_cycles	AXI-0, aclken_input (memory system frequency)
0x8D	-	-
0x8E	axi0_rd_stall_limit	AXI-0, check if read stalled due to AXI counter limit reached
0x8F	axi0_wr_stall_limit	AXI-0, check if write stalled due to AXI counter limit reached
0xA0	axi_latency_any	Any latency; measures the total number of transactions for the specified ID and interface
0xA1	axi_latency_32	Latency was ≥ 32 cycles
0xA2	axi_latency_64	Latency was ≥ 64 cycles
0xA3	axi_latency_128	Latency was ≥ 128 cycles
0xA4	axi_latency_256	Latency was ≥ 256 cycles
0xA5	axi_latency_512	Latency was ≥ 512 cycles
0xA6	axi_latency_1024	Latency was ≥ 1024 cycles
0xB0	DMA ECC event	DMA RAM error (corrected or uncorrected)
0xB1	SB ECC event	SB RAM error (corrected or uncorrected)
0x180	axi1_rd_trans_accepted	AXI-1 read transfer accepted, arready & arvalid (number of read transfers)
0x181	-	-
0x182	axi1_rd_data_beat_received	AXI-1, rready & rvalid (read bandwidth)
0x183	axi1_rd_tran_req_stalled	AXI-1, arvalid & ~arready (read stalls due memory system)

Value	Name	Meaning
0x184	axi1_wr_trans_accepted	AXI-1, awready & awvalid (number write transfers)
0x185-0x186	-	-
0x187	axi1_wr_data_beat_written	AXI-1, wvalid & wready (write bandwidth)
0x188	axi1_wr_tran_req_stalled	AXI-1, awvalid & ~awready (write transfer stalls due to memory system)
0x189	axi1_wr_data_beat_stalled	AXI-1, wvalid & ~wready (write beat stalls due to memory system)
0x18A-0x18B	-	-
0x18C	axi1_enabled_cycles	AXI-1, aclken_input (memory system frequency)
0x18D	-	-
0x18E	axi1_rd_stall_limit	AXI-1, check if read stalled due to AXI counter limit reached
0x18F	axi1_wr_stall_limit	AXI-1, check if write stalled due to AXI counter limit reached

4.5.6 Register PMOVSSET

The overflow-flag status set register.

Sets the state of the overflow bit for the dedicated cycle counter, PMCCNTR, and each of the implemented event counters PMU_EVCNTRn.

4.5.6 Register PMOVSSET on page 68 is used together with the 4.5.7 Register PMOVSCLR on page 70 register. It is implemented in hardware with the same underlying state as 4.5.7 Register PMOVSCLR on page 70.

This register sets the overflow bit as follows: writing 1 to bit[31] sets the overflow bit for the cycle counter and writing 1 to bit[0-3] sets the overflow bit for event counter[0-3]. This register is not written to in normal operation.

Table 4-82: Register PMU.PMOVSSET layout

Bits	Link	Name	Usage	Default
[31]	CYCLE_CNT_OVF	CYCLE_CNT_OVF	PMCCNTR overflow set bit	0
[30:4]	Reserved	-	-	-
[3]	EVENT_CNT_3_OVF	EVENT_CNT_3_OVF	Event-counter overflow set bit for PMU_EVCNTR3	0
[2]	EVENT_CNT_2_OVF	EVENT_CNT_2_OVF	Event-counter overflow set bit for PMU_EVCNTR2	0
[1]	EVENT_CNT_1_OVF	EVENT_CNT_1_OVF	Event-counter overflow set bit for PMU_EVCNTR1	0
[0]	EVENT_CNT_0_OVF	EVENT_CNT_0_OVF	Event-counter overflow set bit for PMU_EVCNTR0	0

Field CYCLE_CNT_OVF

PMCCNTR overflow set bit.

CYCLE_CNT_OVF is stored in bit[31] and is a 1-bit flag. Its default value is 0.

Table 4-83: Field CYCLE_CNT_OVF values

Value	Meaning
0 (default)	When read, it means the cycle counter has not overflowed. When written, it has no effect.
1	When read, it means the cycle counter has overflowed. When written, it sets the overflow bit to 1.

Field EVENT_CNT_3_OVF

Event-counter overflow set bit for PMU_EVCNTR3.

EVENT_CNT_3_OVF is stored in bit[3] and is a 1-bit flag. Its default value is 0.

Table 4-84: Field EVENT_CNT_3_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it sets the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 1.

Field EVENT_CNT_2_OVF

Event-counter overflow set bit for PMU_EVCNTR2.

EVENT_CNT_2_OVF is stored in bit[2] and is a 1-bit flag. Its default value is 0.

Table 4-85: Field EVENT_CNT_2_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it sets the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 1.

Field EVENT_CNT_1_OVF

Event-counter overflow set bit for PMU_EVCNTR1.

EVENT_CNT_1_OVF is stored in bit[1] and is a 1-bit flag. Its default value is 0.

Table 4-86: Field EVENT_CNT_1_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it sets the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 1.

Field EVENT_CNT_0_OVF

Event-counter overflow set bit for PMU_EVCNTR0.

EVENT_CNT_0_OVF is stored in bit[0] and is a 1-bit flag. Its default value is 0.

Table 4-87: Field EVENT_CNT_0_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it sets the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 1.

4.5.7 Register PMOVSCLR

The overflow-flag status clear register.

Contains the status of the overflow bit for the dedicated cycle counter, PMCCNTR, and each of the implemented event counters PMU_EVCNTRn.

4.5.7 Register PMOVSCLR on page 70 is used together with the 4.5.6 Register PMOVSSET on page 68 register. It is implemented in hardware with the same underlying state as 4.5.6 Register PMOVSSET on page 68.

Writing to this register clears overflows as follows: writing a 1 to bit[31] clears overflow for the cycle counter and writing 1 to bit[0-3] clears overflow from event counter 0-3, respectively.

Reading from this register gives the overflow status.

Table 4-88: Register PMU.PMOVSCLR layout

Bits	Link	Name	Usage	Default
[31]	CYCLE_CNT_OVF	CYCLE_CNT_OVF	PMCCNTR overflow clear bit	0
[30:4]	Reserved	-	-	-
[3]	EVENT_CNT_3_OVF	EVENT_CNT_3_OVF	Event-counter overflow clear bit for PMU_EVCNTR3	0
[2]	EVENT_CNT_2_OVF	EVENT_CNT_2_OVF	Event-counter overflow clear bit for PMU_EVCNTR2	0
[1]	EVENT_CNT_1_OVF	EVENT_CNT_1_OVF	Event-counter overflow clear bit for PMU_EVCNTR1	0
[0]	EVENT_CNT_0_OVF	EVENT_CNT_0_OVF	Event-counter overflow clear bit for PMU_EVCNTR0	0

Field CYCLE_CNT_OVF

PMCCNTR overflow clear bit.

CYCLE_CNT_OVF is stored in bit[31] and is a 1-bit flag. Its default value is 0.

Table 4-89: Field CYCLE_CNT_OVF values

Value	Meaning
0 (default)	When read, it means the cycle counter has not overflowed. When written, it has no effect.
1	When read, it means the cycle counter has overflowed. When written, it clears the overflow bit to 0.

Field EVENT_CNT_3_OVF

Event-counter overflow clear bit for PMU_EVCNTR3.

EVENT_CNT_3_OVF is stored in bit[3] and is a 1-bit flag. Its default value is 0.

Table 4-90: Field EVENT_CNT_3_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it clears the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 0.

Field EVENT_CNT_2_OVF

Event-counter overflow clear bit for PMU_EVCNTR2.

EVENT_CNT_2_OVF is stored in bit[2] and is a 1-bit flag. Its default value is 0.

Table 4-91: Field EVENT_CNT_2_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it clears the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 0.

Field EVENT_CNT_1_OVF

Event-counter overflow clear bit for PMU_EVCNTR1.

EVENT_CNT_1_OVF is stored in bit[1] and is a 1-bit flag. Its default value is 0.

Table 4-92: Field EVENT_CNT_1_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it clears the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 0.

Field EVENT_CNT_0_OVF

Event-counter overflow clear bit for PMU_EVCNTR0.

EVENT_CNT_0_OVF is stored in bit[0] and is a 1-bit flag. Its default value is 0.

Table 4-93: Field EVENT_CNT_0_OVF values

Value	Meaning
0 (default)	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has not overflowed. When written, it has no effect.
1	When read, it means that 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 has overflowed. When written, it clears the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 overflow bit to 0.

4.5.8 Register PMINTSET

The interrupt-enable set register.

Enables the generation of interrupt requests on overflows from the dedicated cycle counter, PMCCNTR, and the event counters PMU_EVCNTRn. Reading the register shows which overflow interrupt requests are enabled.

4.5.8 Register PMINTSET on page 71 is used together with the 4.5.9 Register PMINTCLR on page 73 register. It is implemented in hardware with the same underlying state as 4.5.9 Register PMINTCLR on page 73.

Writing to this register enables overflow interrupt detection as follows: writing a 1 to bit[31] enables overflow interrupts from the cycle counter and writing a 1 to bit[0-3] enables overflow interrupts from event counter 0-3, respectively.

Reading from 4.5.8 Register PMINTSET on page 71 or 4.5.9 Register PMINTCLR on page 73 gives the same value, which is the overflow enable status of the counters.

Table 4-94: Register PMU.PMINTSET layout

Bits	Link	Name	Usage	Default
[31]	CYCLE_CNT_INT	CYCLE_CNT_INT	PMCCNTR overflow interrupt-request enable bit	0
[30:4]	Reserved	-	-	-
[3]	EVENT_CNT_3_INT	EVENT_CNT_3_INT	Event-counter overflow interrupt-request enable bit for PMU_EVCNTR3	0
[2]	EVENT_CNT_2_INT	EVENT_CNT_2_INT	Event-counter overflow interrupt-request enable bit for PMU_EVCNTR2	0
[1]	EVENT_CNT_1_INT	EVENT_CNT_1_INT	Event-counter overflow interrupt-request enable bit for PMU_EVCNTR1	0
[0]	EVENT_CNT_0_INT	EVENT_CNT_0_INT	Event-counter overflow interrupt-request enable bit for PMU_EVCNTR0	0

Field CYCLE_CNT_INT

PMCCNTR overflow interrupt-request enable bit.

CYCLE_CNT_INT is stored in bit[31] and is a 1-bit flag. Its default value is 0.

Table 4-95: Field CYCLE_CNT_INT values

Value	Meaning
0 (default)	When read, it means the cycle-counter overflow interrupt request is disabled. When written, it has no effect.
1	When read, it means the cycle-counter overflow interrupt request is enabled. When written, it enables the cycle count overflow interrupt request.

Field EVENT_CNT_3_INT

Event-counter overflow interrupt-request enable bit for PMU_EVCNTR3.

EVENT_CNT_3_INT is stored in bit[3] and is a 1-bit flag. Its default value is 0.

Table 4-96: Field EVENT_CNT_3_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is enabled. When written, it enables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

Field EVENT_CNT_2_INT

Event-counter overflow interrupt-request enable bit for PMU_EVCNTR2.

EVENT_CNT_2_INT is stored in bit[2] and is a 1-bit flag. Its default value is 0.

Table 4-97: Field EVENT_CNT_2_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is enabled. When written, it enables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

Field EVENT_CNT_1_INT

Event-counter overflow interrupt-request enable bit for PMU_EVCNTR1.

EVENT_CNT_1_INT is stored in bit[1] and is a 1-bit flag. Its default value is 0.

Table 4-98: Field EVENT_CNT_1_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is enabled. When written, it enables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

Field EVENT_CNT_0_INT

Event-counter overflow interrupt-request enable bit for PMU_EVCNTR0.

EVENT_CNT_0_INT is stored in bit[0] and is a 1-bit flag. Its default value is 0.

Table 4-99: Field EVENT_CNT_0_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is enabled. When written, it enables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

4.5.9 Register PMINTCLR

The interrupt-enable clear register.

Disables the generation of interrupt requests on overflows from the dedicated cycle counter, PMCCNTR, and the event counters PMU_EVCNTRn. Reading the register shows which overflow interrupt requests are enabled.

4.5.9 Register PMINTCLR on page 73 is used together with the 4.5.8 Register PMINTSET on page 71 register. It is implemented in hardware with the same underlying state as 4.5.8 Register PMINTSET on page 71.

Writing to this register disables overflow interrupt detection as follows: writing a 1 to bit[31] disables overflow interrupts from the cycle counter and writing a 1 to bit[0-3] disables overflow interrupts from event counter 0-3, respectively.

Reading from 4.5.8 Register PMINTSET on page 71 or 4.5.9 Register PMINTCLR on page 73 gives the same value, which is the overflow enable status of the counters.

Table 4-100: Register PMU.PMINTCLR layout

Bits	Link	Name	Usage	Default
[31]	CYCLE_CNT_INT	CYCLE_CNT_INT	PMCCNTR overflow interrupt-request disable bit	0
[30:4]	Reserved	-	-	-
[3]	EVENT_CNT_3_INT	EVENT_CNT_3_INT	Event-counter overflow interrupt-request disable bit for PMU_EVCNTR3	0
[2]	EVENT_CNT_2_INT	EVENT_CNT_2_INT	Event-counter overflow interrupt-request disable bit for PMU_EVCNTR2	0
[1]	EVENT_CNT_1_INT	EVENT_CNT_1_INT	Event-counter overflow interrupt-request disable bit for PMU_EVCNTR1	0
[0]	EVENT_CNT_0_INT	EVENT_CNT_0_INT	Event-counter overflow interrupt-request disable bit for PMU_EVCNTR0	0

Field CYCLE_CNT_INT

PMCCNTR overflow interrupt-request disable bit.

CYCLE_CNT_INT is stored in bit[31] and is a 1-bit flag. Its default value is 0.

Table 4-101: Field CYCLE_CNT_INT values

Value	Meaning
0 (default)	When read, it means the cycle-counter overflow interrupt-request is disabled. When written, it has no effect.
1	When read, it means the cycle-counter overflow interrupt-request is enabled. When written, it disables the cycle count overflow interrupt request.

Field EVENT_CNT_3_INT

Event-counter overflow interrupt-request disable bit for PMU_EVCNTR3.

EVENT_CNT_3_INT is stored in bit[3] and is a 1-bit flag. Its default value is 0.

Table 4-102: Field EVENT_CNT_3_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event counter interrupt request is enabled. When written, it disables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

Field EVENT_CNT_2_INT

Event-counter overflow interrupt-request disable bit for PMU_EVCNTR2.

EVENT_CNT_2_INT is stored in bit[2] and is a 1-bit flag. Its default value is 0.

Table 4-103: Field EVENT_CNT_2_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is enabled. When written, it disables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

Field EVENT_CNT_1_INT

Event-counter overflow interrupt-request disable bit for PMU_EVCNTR1.

EVENT_CNT_1_INT is stored in bit[1] and is a 1-bit flag. Its default value is 0.

Table 4-104: Field EVENT_CNT_1_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is enabled. When written, it disables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

Field EVENT_CNT_0_INT

Event-counter overflow interrupt-request disable bit for PMU_EVCNTR0.

EVENT_CNT_0_INT is stored in bit[0] and is a 1-bit flag. Its default value is 0.

Table 4-105: Field EVENT_CNT_0_INT values

Value	Meaning
0 (default)	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is disabled. When written, it has no effect.
1	When read, it means that the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 event-counter interrupt request is enabled. When written, it disables the 4.5.4 PMU_EVCNTR0 ... PMU_EVCNTR3 on page 66 interrupt request.

4.5.10 Register PMCCNTR_LO

Performance-monitor cycle count low register.

This represents the lower 32 bits of the dedicated 48-bit cycle counter, PMCCNTR.

Table 4-106: Register PMU.PMCCNTR_LO layout

Bits	Link	Name	Usage	Default
[31:0]	CYCLE_CNT_LO	CYCLE_CNT_LO	Cycle count low	0x00000000

Field CYCLE_CNT_LO

Cycle count low.

CYCLE_CNT_LO is stored in bits[31:0] and is a 32-bit unsigned integer. Its default value is 0x00000000.

4.5.11 Register PMCCNTR_HI

Performance-monitor cycle count high register.

This represents the higher 16 bits of the dedicated 48-bit cycle counter, PMCCNTR.

Table 4-107: Register PMU.PMCCNTR_HI layout

Bits	Link	Name	Usage	Default
[31:16]	Reserved	-	-	-
[15:0]	CYCLE_CNT_HI	CYCLE_CNT_HI	Cycle count high	0x0000

Field CYCLE_CNT_HI

Cycle count high.

CYCLE_CNT_HI is stored in bits[15:0] and is a 16-bit unsigned integer. Its default value is 0x0000.

4.5.12 Register PMCAXI_CHAN

Set which AXI channel to monitor.

Monitors for AXI bandwidth (bw) events (0x80-0x89, 0x180-0x189) and AXI latency events (0xA0-0xA6).

Table 4-108: Register PMU.PMCAXI_CHAN layout

Bits	Link	Name	Usage	Default
[31:11]	Reserved	-	-	-
[10]	BW_CH_SEL_EN	BW_CH_SEL_EN	Enable bandwidth channel selector: 0=AXI bw events measured for all channels, 1=AXI bw events measured for channel specified by CH_SEL	0x000000

Bits	Link	Name	Usage	Default
[9:8]	AXI_CNT_SEL	AXI_CNT_SEL	Select AXI counter to monitor for latency measurements (0=AXI0 counter0, 1=AXI0 counter1, 2=AXI1 counter 2, 3=AXI1 counter3)	0x000000
[7:4]	Reserved	-	-	-
[3:0]	CH_SEL	CH_SEL	Specify the type of traffic for bandwidth or latency measurements (Read: 0=command traffic, 1=IFM traffic, 2=Weight traffic, 3=Scale+Bias, 4=Mem2Mem traffic - read direction; Write: 8=OFM traffic, 9=Mem2Mem traffic - write direction)	0x0

Field BW_CH_SEL_EN

Enable bandwidth channel selector: 0=AXI bw events measured for all channels, 1=AXI bw events measured for channel specified by CH_SEL.

BW_CH_SEL_EN is stored in bit[10] and is a 1-bit unsigned integer. Its default value is 0x000000.

Field AXI_CNT_SEL

Select AXI counter to monitor for latency measurements (0=AXI0 counter0, 1=AXI0 counter1, 2=AXI1 counter 2, 3=AXI1 counter3).

AXI_CNT_SEL is stored in bits[9:8] and is a 2-bit unsigned integer. Its default value is 0x000000.

A maximum of two separate outstanding transaction queues can be connected to each AXI interface. The counters are used to express the maximum number of outstanding jobs per queue.

Field CH_SEL

Specify the type of traffic for bandwidth or latency measurements (Read: 0=command traffic, 1=IFM traffic, 2=Weight traffic, 3=Scale+Bias, 4=Mem2Mem traffic - Read direction; Write: 8=OFM traffic, 9=Mem2Mem traffic - Write direction).

CH_SEL is stored in bits[3:0] and is a 4-bit unsigned integer. Its default value is 0x0.

4.6 Command stream

The application processor uses a command stream to issue tasks to the NPU. The command stream is made from 16-bit commands.

There are two command formats, `cmd0` and `cmd1`. `cmd0` is a 32-bit command with no data item. `cmd1` is a 32-bit command followed by a single 32-bit data item. In the command stream, these commands must be aligned to start on a 32-bit boundary.

Bits[15:0] determine the command name. Bits[31:16] are the command parameter which the command uses.

The NPU processes commands in the order they are received.

The following table lists the command formats and their differences.

Table 4-109: Command stream formats

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bits 9-0	Data
0	0	0	0	0	0	cmd0	No data item
0	1	0	0	0	0	cmd1	32-bit data item payload after the command

Note

All unused combinations of bits[0:15] are reserved.

The following is an example command stream for a Conv2D network with input tensor 8x8x16, weight tensor 16x2x2x16, stride 2x2, and output tensor 4x4x16. The following example applies to a configuration with 256 MAC units.

```
Code:    Command:                          Param: Payload:
0x0130 cmd0.NPU_SET_DMA0_SRC_REGION          0    -
0x4030 cmd1.NPU_SET_DMA0_SRC                 0    0x00000000 (0)
0x0131 cmd0.NPU_SET_DMA0_DST_REGION          1    -
0x4031 cmd1.NPU_SET_DMA0_DST                 0    0x00000400 (1024)
0x4032 cmd1.NPU_SET_DMA0_LEN                 0    0x000002e0 (736)
0x0010 cmd0.NPU_OP_DMA_START                 0    -
0x0116 cmd0.NPU_SET_OFM_BLK_HEIGHT_M1        3    -
0x0115 cmd0.NPU_SET_OFM_BLK_WIDTH_M1         3    -
0x0117 cmd0.NPU_SET_OFM_BLK_DEPTH_M1        15    -
0x010d cmd0.NPU_SET_IFM_IB_END              10    -
0x012d cmd0.NPU_SET_AB_START                30    -
0x0124 cmd0.NPU_SET_ACC_FORMAT               0    -
0x0107 cmd0.NPU_SET_IFM_UPSCALE              0    -
0x0100 cmd0.NPU_SET_IFM_PAD_TOP              0    -
0x0101 cmd0.NPU_SET_IFM_PAD_LEFT             0    -
0x0103 cmd0.NPU_SET_IFM_PAD_BOTTOM           0    -
0x0102 cmd0.NPU_SET_IFM_PAD_RIGHT            0    -
0x0121 cmd0.NPU_SET_KERNEL_HEIGHT_M1         1    -
0x0120 cmd0.NPU_SET_KERNEL_WIDTH_M1          1    -
0x0122 cmd0.NPU_SET_KERNEL_STRIDE            7    -
0x4020 cmd1.NPU_SET_WEIGHT_BASE              0    0x00000400 (1024)
0x4021 cmd1.NPU_SET_WEIGHT_LENGTH            0    0x000002e0 (736)
0x0128 cmd0.NPU_SET_WEIGHT_REGION            1    -
0x4022 cmd1.NPU_SET_SCALE_BASE               0    0x000002e0 (736)
0x4023 cmd1.NPU_SET_SCALE_LENGTH             0    0x000000a0 (160)
0x0129 cmd0.NPU_SET_SCALE_REGION             0    -
0x0125 cmd0.NPU_SET_ACTIVATION               0    -
0x0126 cmd0.NPU_SET_ACTIVATION_MIN           0    -
0x0127 cmd0.NPU_SET_ACTIVATION_MAX         255    -
0x0112 cmd0.NPU_SET_OFM_HEIGHT_M1            3    -
0x0111 cmd0.NPU_SET_OFM_WIDTH_M1             3    -
0x0113 cmd0.NPU_SET_OFM_DEPTH_M1            15    -
0x0104 cmd0.NPU_SET_IFM_DEPTH_M1            15    -
0x0109 cmd0.NPU_SET_IFM_ZERO_POINT         128    -
0x010b cmd0.NPU_SET_IFM_HEIGHT0_M1           7    -
0x010c cmd0.NPU_SET_IFM_HEIGHT1_M1           7    -
0x010a cmd0.NPU_SET_IFM_WIDTH0_M1            7    -
0x010f cmd0.NPU_SET_IFM_REGION               1    -
0x4000 cmd1.NPU_SET_IFM_BASE0                0    0x00000000 (0)
0x4001 cmd1.NPU_SET_IFM_BASE1                0    0x00000000 (0)
0x4002 cmd1.NPU_SET_IFM_BASE2                0    0x00000000 (0)
0x4003 cmd1.NPU_SET_IFM_BASE3                0    0x00000000 (0)
0x4006 cmd1.NPU_SET_IFM_STRIDE_C             0    0x00000001 (1)
0x4004 cmd1.NPU_SET_IFM_STRIDE_X             0    0x00000010 (16)
0x4005 cmd1.NPU_SET_IFM_STRIDE_Y             0    0x00000080 (128)
0x0118 cmd0.NPU_SET_OFM_ZERO_POINT         128    -
0x011b cmd0.NPU_SET_OFM_HEIGHT0_M1           3    -
```

```
0x011c cmd0.NPU_SET_OFM_HEIGHT1_M1         3   -
0x011a cmd0.NPU_SET_OFM_WIDTH0_M1          3   -
0x011f cmd0.NPU_SET_OFM_REGION             1   -
0x4010 cmd1.NPU_SET_OFM_BASE0              0   0x000006e0 (1760)
0x4011 cmd1.NPU_SET_OFM_BASE1              0   0x00000000 (0)
0x4012 cmd1.NPU_SET_OFM_BASE2              0   0x00000000 (0)
0x4013 cmd1.NPU_SET_OFM_BASE3              0   0x00000000 (0)
0x4016 cmd1.NPU_SET_OFM_STRIDE_C           0   0x00000001 (1)
0x4014 cmd1.NPU_SET_OFM_STRIDE_X           0   0x00000010 (16)
0x4015 cmd1.NPU_SET_OFM_STRIDE_Y           0   0x00000040 (64)
0x0114 cmd0.NPU_SET_OFM_PRECISION          0   -
0x0105 cmd0.NPU_SET_IFM_PRECISION          0   -
0x0011 cmd0.NPU_OP_DMA_WAIT                0   -
0x012f cmd0.NPU_SET_BLOCKDEP               3   -
0x0002 cmd0.NPU_OP_CONV                    0   -
0x0000 cmd0.NPU_OP_STOP                65535   -
```

The following is an example command stream for a MaxPool2D with 2x2 kernel and 8x8x16 tensor. The following example applies to a configuration with 256 MAC units.

```
Code:    Command:                         Param: Payload:
0x0116 cmd0.NPU_SET_OFM_BLK_HEIGHT_M1      7   -
0x0115 cmd0.NPU_SET_OFM_BLK_WIDTH_M1       7   -
0x0117 cmd0.NPU_SET_OFM_BLK_DEPTH_M1      15   -
0x010d cmd0.NPU_SET_IFM_IB_END            10   -
0x012d cmd0.NPU_SET_AB_START              30   -
0x0124 cmd0.NPU_SET_ACC_FORMAT             0   -
0x0107 cmd0.NPU_SET_IFM_UPSCALE            0   -
0x0100 cmd0.NPU_SET_IFM_PAD_TOP            0   -
0x0101 cmd0.NPU_SET_IFM_PAD_LEFT           0   -
0x0103 cmd0.NPU_SET_IFM_PAD_BOTTOM         1   -
0x0102 cmd0.NPU_SET_IFM_PAD_RIGHT          1   -
0x0121 cmd0.NPU_SET_KERNEL_HEIGHT_M1       1   -
0x0120 cmd0.NPU_SET_KERNEL_WIDTH_M1        1   -
0x0122 cmd0.NPU_SET_KERNEL_STRIDE          0   -
0x0125 cmd0.NPU_SET_ACTIVATION             0   -
0x0126 cmd0.NPU_SET_ACTIVATION_MIN         0   -
0x0127 cmd0.NPU_SET_ACTIVATION_MAX       255   -
0x0112 cmd0.NPU_SET_OFM_HEIGHT_M1          7   -
0x0111 cmd0.NPU_SET_OFM_WIDTH_M1           7   -
0x0113 cmd0.NPU_SET_OFM_DEPTH_M1          15   -
0x0104 cmd0.NPU_SET_IFM_DEPTH_M1          15   -
0x0109 cmd0.NPU_SET_IFM_ZERO_POINT       128   -
0x010b cmd0.NPU_SET_IFM_HEIGHT0_M1         7   -
0x010c cmd0.NPU_SET_IFM_HEIGHT1_M1         7   -
0x010a cmd0.NPU_SET_IFM_WIDTH0_M1          7   -
0x010f cmd0.NPU_SET_IFM_REGION             1   -
0x4000 cmd1.NPU_SET_IFM_BASE0              0   0x00000000 (0)
0x4001 cmd1.NPU_SET_IFM_BASE1              0   0x00000000 (0)
0x4002 cmd1.NPU_SET_IFM_BASE2              0   0x00000000 (0)
0x4003 cmd1.NPU_SET_IFM_BASE3              0   0x00000000 (0)
0x4006 cmd1.NPU_SET_IFM_STRIDE_C           0   0x00000001 (1)
0x4004 cmd1.NPU_SET_IFM_STRIDE_X           0   0x00000010 (16)
0x4005 cmd1.NPU_SET_IFM_STRIDE_Y           0   0x00000080 (128)
0x0118 cmd0.NPU_SET_OFM_ZERO_POINT       128   -
0x011b cmd0.NPU_SET_OFM_HEIGHT0_M1         7   -
0x011c cmd0.NPU_SET_OFM_HEIGHT1_M1         7   -
0x011a cmd0.NPU_SET_OFM_WIDTH0_M1          7   -
0x011f cmd0.NPU_SET_OFM_REGION             1   -
0x4010 cmd1.NPU_SET_OFM_BASE0              0   0x00000400 (1024)
0x4011 cmd1.NPU_SET_OFM_BASE1              0   0x00000000 (0)
0x4012 cmd1.NPU_SET_OFM_BASE2              0   0x00000000 (0)
0x4013 cmd1.NPU_SET_OFM_BASE3              0   0x00000000 (0)
0x4016 cmd1.NPU_SET_OFM_STRIDE_C           0   0x00000001 (1)
0x4014 cmd1.NPU_SET_OFM_STRIDE_X           0   0x00000010 (16)
0x4015 cmd1.NPU_SET_OFM_STRIDE_Y           0   0x00000080 (128)
0x0114 cmd0.NPU_SET_OFM_PRECISION          0   -
0x0105 cmd0.NPU_SET_IFM_PRECISION          0   -
```

```
0x012f cmd0.NPU_SET_BLOCKDEP          3   -
0x0005 cmd0.NPU_OP_POOL               0   -
0x0000 cmd0.NPU_OP_STOP           65535   -
```

4.6.1 Non-blocking command types

Commands can be non-blocking, which means that later commands can start before they are completed.

The following table lists the non-blocking command types and the criteria that must be met for the command to complete.

Table 4-110: Non-blocking command types

Command	Completion criteria
NPU_OP_IRQ	An **IRQ** is raised.
NPU_OP_<KERNEL> <KERNEL> can be: • CONV for convolution operations • DEPTHWISECONV for depth-wise convolution operations • POOL for pooling operations • ELEMENTWISE for elementwise operations	The resulting tensor is calculated and written to memory.

4.6.2 Blocking command types

Commands can be blocking, which means that later commands cannot start before these commands are completed.

The following table lists the blocking command types and the criteria that must be met for the command to complete.

Table 4-111: Non-blocking command types

Command	Completion criteria
NPU_SET_<STATE>	The value is written to the appropriate internal state. This value is applied to all following kernel operations, until a new command overwrites it. New values must not affect operations that are already in progress.
NPU_OP_STOP	The NPU enters a stopped state.
NPU_OP_DMA_START	The Direct Memory Access (DMA) instruction is accepted into the internal DMA queue. The DMA instruction does not need to complete.
NPU_OP_<CONDITION>_WAIT	The wait condition is satisfied.

4.6.3 Command dependency requirements

When an operation is started, the NPU must know all the input data for it to be valid. If the NPU does not know all the input data, then the behavior is **UNPREDICTABLE**.

The NPU_OP_SET_BLOCKDEP command sets the block dependency between NPU kernel operations.

The NPU_OP_DMA_WAIT command causes the NPU to wait for certain results from previously started DMA operations to be completed and written to memory. During this wait, the NPU does not add later commands to the Command queue.

4.6.4 cmd0 commands

`cmd0` commands have bits[15:10] = 0. `cmd0` bits[9:0] indicate the command. `cmd0` commands do not take additional data.

Use these commands to:

- Perform an action, for example, raising an **IRQ** or starting an operation.
- Set a state based on the 16-bit parameter value.

The following table lists the `cmd0` commands and their actions.

Table 4-112: cmd0 operations

cmd0	Enumerator	Parameter	Function
`0x000`	NPU_OP_STOP	mask	(1) Set BASE_STATUS |= (mask<<16). (2) Move to stopped state. (3) Raise IRQ to host (regardless of mask value). At the point the IRQ is raised, the NPU is stopped and all operations complete up to and including the STOP operation. Operations after the STOP may have been buffered in the Command queue, but are not started (so no input or weight data is read).
`0x001`	NPU_OP_IRQ	mask	(1) Set BASE_STATUS |= (mask<<16). (2) Remain in run state. (3) Raise IRQ to host (regardless of mask value). At the point the IRQ is raised, all operations are complete up to and including the IRQ operation. Operations after the IRQ may have been started (or even completed). At most, only one IRQ operation can be placed between an NPU_OP_<KERNEL> command and the following NPU_OP_<KERNEL> command (see 4.6.1 Non-blocking command types on page 80 for the definition of KERNEL).
`0x002`	NPU_OP_CONV	0	Start stripe with all-layer convolution or deconvolution.

cmd0	Enumerator	Parameter	Function
`0x003`	NPU_OP_DEPTHWISE	0	Start stripe width depth-wise convolution or deconvolution operation.
`0x004`	-	-	-
`0x005`	NPU_OP_POOL	mode	Start stripe with pooling operation. mode: 0=MaxPool, 1=Average pool, 2=REDUCE_SUM.
`0x006`	NPU_OP_ELEMENTWISE	mode	Start stripe with elementwise operation between two IFMs. mode: 0=Mul, 1=Add, 2=Sub, 3=Min, 4=Max, 5=LReLU, 6=ABS, 7=CLZ, 8=SHR, 9=SHL.
`0x007`	-	-	-
`0x010`	NPU_OP_DMA_START	16*channel	Queue new DMA for the given channel. The NPU contains one user channel. Therefore, channel=0. This command blocks until the DMA channel can accept a new descriptor. This command is viewed as complete when the DMA has been queued and does not need to wait for the DMA to complete. (This is different to other NPU_OP commands that must have their final results written to memory before they are considered complete.)
`0x011`	NPU_OP_DMA_WAIT	16*channel + k	Wait for the DMA channel to have k or fewer active descriptors outstanding. The NPU contains one user channel. Therefore, channel=0. The NPU contains one descriptor per channel. Therefore, k=0 and the command waits for the single DMA to be complete. Descriptors are not outstanding if they have completed, which means that data written to memory and can be read by the next command.
`0x012`	NPU_OP_KERNEL_WAIT	*n*=0-3	Wait for *n* or fewer kernel operations to be remaining (that is, not complete) before starting the next command. A kernel operation is Conv, Depthwise, Pool, Elementwise. This command is typically placed before an NPU_OP_DMA_START command to prevent the DMA from starting until a previous kernel operation reading the memory has completed.
`0x013`	NPU_OP_PMU_MASK	0 or 1	Enable or disable PMU counting (for debug purposes only)
`0x100`	NPU_SET_IFM_PAD_TOP	0-127	IFM top pad. Padding is applied after upscale, if ifm_upscale_mode!=none.
`0x101`	NPU_SET_IFM_PAD_LEFT	0-127	IFM left pad. Padding is applied after upscale, if ifm_upscale_mode!=none.
`0x102`	NPU_SET_IFM_PAD_RIGHT	0-128	IFM right pad. Padding is applied after upscale, if ifm_upscale_mode!=none.
`0x103`	NPU_SET_IFM_PAD_BOTTOM	0-128	IFM bottom pad. Padding is applied after upscale if ifm_upscale_mode!=none.

cmd0	Enumerator	Parameter	Function
`0x104`	NPU_SET_IFM_DEPTH_M1	0-65535	Number of input channels for convolution -1.
`0x105`	NPU_SET_IFM_PRECISION	bitfield	b0 = activation type 0=unsigned, 1=signed b1 = reserved for weight size b[3:2] = activation precision 0=8 bit, 1=16 bit, 2=32 bit (only available for certain operations) b[7:6] = IFM format select 0=NHWC or 1=NHCWB16 b[9:8] = IFM scale mode for elementwise ADD and SUB: 0=16-bit OPA/OPB scale, 1=32-bit OPA scale applied to OPA, 2=32-bit OPA scale applied to OPB b[15:14] = IFM round mode: 0=double rounding, 2=round to nearest with 0.5 round to +infinity
`0x106`	-	-	-
`0x107`	NPU_SET_IFM_UPSCALE	0, 1, 2	b[1:0] = ifm_upscale_mode (0=none, 1=2x2 insert nearest, 2=2x2 insert zeros)
`0x108`	-	-	-
`0x109`	NPU_SET_IFM_ZERO_POINT	int16 or uint16	IFM zero-point offset. Encoded as int16, if activation is signed or uint16, if activation is unsigned. Must be zero for 32-bit IFM and for CLZ operation. Must be a valid activation value.
`0x10A`	NPU_SET_IFM_WIDTH0_M1	0-65535	IFM Tile 0 and tile 2 (width-1)
`0x10B`	NPU_SET_IFM_HEIGHT0_M1	0-65535	IFM Tile 0 (height-1)
`0x10C`	NPU_SET_IFM_HEIGHT1_M1	0-65535	IFM Tile 1 (height-1)
`0x10D`	NPU_SET_IFM_IB_END	0-48	End of IB0,IB1 buffers in the SHRAM in KB units. Multiples of 2.
`0x10E`	-	-	-
`0x10F`	NPU_SET_IFM_REGION	0-7	Index *n* for IFM access: Region[*n*] is added to all IFM addresses.
`0x110`	-	-	-
`0x111`	NPU_SET_OFM_WIDTH_M1	0-65535	OFM width-1 (for the stripe to process)
`0x112`	NPU_SET_OFM_HEIGHT_M1	0-65535	OFM height-1 (for the stripe to process)
`0x113`	NPU_SET_OFM_DEPTH_M1	0-65535	OFM depth-1 for convolution

cmd0	Enumerator	Parameter	Function
0x114	NPU_SET_OFM_PRECISION	bitfield	b0 = activation type 0=unsigned, 1=signed b[2:1] = activation precision type 0=8 bit, 1=16 bit, 2=32 bit (only available for certain operations) b[7:6] = OFM format select 0=NHWC or 1=NHCWB16 b[8] = scaling, 0=Per channel scale/bias, 1=Global scale (SET_OFM_SCALE), no bias b[15:14] = rounding mode, 0=double rounding, 1=truncate towards zero, 2=Natural rounding
0x115	NPU_SET_OFM_BLK_WIDTH_M1	0-31	OFM_BLOCK_WIDTH-1 (see 4.9 Block based operation on page 110)
0x116	NPU_SET_OFM_BLK_HEIGHT_M1	0-31	OFM_BLOCK_HEIGHT-1 (see 4.9 Block based operation on page 110)
0x117	NPU_SET_OFM_BLK_DEPTH_M1	3-127	OFM_BLOCK_DEPTH-1 (see 4.9 Block based operation on page 110)
0x118	NPU_SET_OFM_ZERO_POINT	int16 or uint16	OFM zero-point offset. Encoded as int16, if activation is signed or uint16, if activation is unsigned. Must be a valid activation value given by ACTIVATION[15:12]. Must be 0 for 32-bit activation range of for CLZ. **Note:** This can be nonzero, if OFM is 32 bit but ACTIVATION[15:12] range is 8 bit.
0x119	-	-	-
0x11A	NPU_SET_OFM_WIDTH0_M1	0-65535	OFM Tile 0 and tile 2 (width-1)
0x11B	NPU_SET_OFM_HEIGHT0_M1	0-65535	OFM Tile 0 (height-1)
0x11C	NPU_SET_OFM_HEIGHT1_M1	0-65535	OFM Tile 1 (height-1)
0x11D	-	-	-
0x11E	-	-	-
0x11F	NPU_SET_OFM_REGION	0-7	Index *n* for OFM access: Region[*n*] is added to all OFM addresses
0x120	NPU_SET_KERNEL_WIDTH_M1	0-65535	Set (dilated_kernel_width-1) = (kernel_width-1)*kernel_x_dilation
0x121	NPU_SET_KERNEL_HEIGHT_M1	0-65535	Set (dilated_kernel_height-1) = (kernel_height-1)*kernel_y_dilation

cmd0	Enumerator	Parameter	Function
`0x122`	NPU_SET_KERNEL_STRIDE	bitfield	b0 = (kernel_x_stride – 1)&1 (x stride low bit) b1 = (kernel_y_stride – 1)&1 (y stride low bit) b2 = kernel_weight_order (0=depth-first weight order, 1=part kernel-first weight order) b3 = kernel_x_dilation - 1 (0=no x dilation, 1=x dilation of x2) b4 = kernel_y_dilation -1 (0=no y dilation, 1=y dilation of x2) b5 = 0 for kernel_split_size=8, 1 for kernel_split_size=4 (8x8 or 4x4 kernel decomposition) b[8:6] = (kernel_x_stride-1) >> 1 (stride extension bits – supported stride range is 1 to 3) b[11:9] = (kernel_y_stride-1)>>1 (stride extension bits – supported stride range is 1 to 3)
`0x123`	-	-	-
`0x124`	NPU_SET_ACC_FORMAT	0-3	Sets the accumulator format: 0=32-bit integer, 1=40-bit integer, 2=s5.10 floating point
`0x125`	NPU_SET_ACTIVATION	0, 3 ,4, 0x10+n	0=none/ReLU, 3=tanh, 4=sigmoid; 0x10+*n* for 0<=*n*<8 indicates a LUT operation starting at address *n**256 bytes in the last 2KB page of the SHRAM b[15:12] = Activation clip range (before table lookup, set to '0' if table lookup is not used). 0=OFM precision, 2=force to uint8 3=force to int8, 5=force to int16
`0x126`	NPU_SET_ACTIVATION_MIN	int16 or uint16	Lower bound clip for OFM activations – range is the OFM type range
`0x127`	NPU_SET_ACTIVATION_MAX	int16 or uint16	Upper bound clip for OFM activations – range is the OFM type range
`0x128`	NPU_SET_WEIGHT_REGION	0-7	Index *n* for weight access: Region[*n*] is added to all Weight stream offsets
`0x129`	NPU_SET_SCALE_REGION	0-7	Index *n* for scale access: Region[*n*] is added to all scale stream offsets
`0x12A`	-	-	-
`0x12B`	-	-	-
`0x12C`	-	-	-
`0x12D`	NPU_SET_AB_START	0-48	Start of ACC0,ACC1 buffers in the SHRAM in KB units. Multiples of 2.
`0x12E`	-	-	-
`0x12F`	NPU_SET_BLOCKDEP	0-3	Set block number of blocks-dependency between kernel operations.

cmd0	Enumerator	Parameter	Function
0x130	NPU_SET_DMA0_SRC_REGION	Bitmap	If Bit[8]=0, Bit[7:0] = Region number in the range 0<=*n*<8 of SRC offset Bit[8] = must be 0 for external Bit[10:9] = stride mode 0=1D
0x131	NPU_SET_DMA0_DST_REGION	Bitmap	If Bit[8]=0, Bit[7:0] = Region number in the range 0<=*n*<8 of DST offset If Bit[8]=1, Bit[7:0] = Core mask to write to (bit k set for core k=0,1) Bit[8] = select external/internal=0/1 Bit[10:9] = stride mode 0=1D
0x132-0x17F	-	-	-
0x180	NPU_SET_IFM2_BROADCAST	bitfield	b0 = broadcast H dimension (if set, then any accesses to IFM2 sets y=0 and IFM2 height=1) b1 = broadcast W dimension (if set, then any accesses to IFM2 sets x=0 and IFM2 width=1) b2 = broadcast C dimension (if set, then any accesses to IFM2 sets c=0 and IFM2 depth=1) b6 = operand order 0=IFM2 is second operand B, 1=IFM2 is first operand A. b7 = broadcast constant given by NPU_SET_IFM2_SCALAR and so ignore b0-b2
0x181	NPU_SET_IFM2_SCALAR	int16 or uint16	IFM2 scalar value at range IFM_PRECISION. The scalar is encoded with IFM2_ZERO_POINT. Values are encoded as signed or unsigned 16-bit values depending on whether IFM2_PRECISION is signed or unsigned.
0x182-0x184	-	-	-
0x185	NPU_SET_IFM_PRECISION	bitfield	b[0] = activation type 0=unsigned, 1=signed – MUST MATCH IFM b[3:2] = activation precision 0=8 bit, 1=16 bit, 2=32 bit – MUST MATCH IFM b[7:6] = IFM2 format, select 0=NHWC or 1=NHCWB16
0x186-0x188	-	-	-
0x189	NPU_SET_IFM2_ZERO_POINT	int16 or uint16	IFM2 zero-point offset. Encoded as int16, if activation is signed or uint16, if activation is unsigned. Must be zero for 32-bit IFM. Must be a valid activation value.
0x18A	NPU_SET_IFM2_WIDTH0_M1	0-65535	IFM2 Tile 0 and tile 2 (width-1)
0x18B	NPU_SET_IFM2_HEIGHT0_M1	0-65535	IFM2 Tile 0 (height-1)

cmd0	Enumerator	Parameter	Function
0x18C	NPU_SET_IFM2_HEIGHT1_M1	0-65535	IFM2 Tile 1 (height-1)
0x18D	NPU_SET_IFM2_IB_START	0-48	Start of IB0, IB1 buffers for IFM2 in SHRAM. In KB units, multiples of 2.
0x18E	-	-	-
0x18F	NPU_SET_IFM2_REGION	0-7	Index *n* for IFM2 access: Region[*n*] is added to all IFM2 addresses

4.6.5 cmd1 commands

`cmd1` commands have bits[15:14] = 1. `cmd1` bits[9:0] indicate the command. `cmd1` commands take a payload data item of 32 bits in addition to the 16-bit parameter field.

About the Parameter field

Where payload items in the following table give an address offset, stride, or data length, the value is in bytes.

The following table lists the `cmd1` commands and their functionality.

Note

Address extension bits are supported, but for those ending with '_32' the Ethos™-U55 NPU supports only 32-bit addresses.

Table 4-113: cmd1 operations

cmd1	Enumerator	Parameter	Payload data
0x000	NPU_SET_IFM_BASE0	extu_47_32	IFM tile0 byte offset (top-left tile) from IFM_REGION start
0x001	NPU_SET_IFM_BASE1	extu_47_32	IFM tile1 byte offset (top-right tile) from IFM_REGION start
0x002	NPU_SET_IFM_BASE2	extu_47_32	IFM tile2 byte offset (bottom-left tile) from IFM_REGION start
0x003	NPU_SET_IFM_BASE3	extu_47_32	IFM tile3 byte offset (bottom-right tile) from IFM_REGION start
0x004	NPU_SET_IFM_STRIDE_X	exts_47_32	IFM byte stride between horizontal values
0x005	NPU_SET_IFM_STRIDE_Y	exts_47_32	IFM byte stride between vertical values
0x006	NPU_SET_IFM_STRIDE_C	exts_47_32	IFM byte stride between channel blocks (of 16 bytes each block)
0x007-0x00F	-	-	-
0x010	NPU_SET_OFM_BASE0	extu_47_32	OFM tile0 byte offset (top-left tile) from OFM_REGION
0x011	NPU_SET_OFM_BASE1	extu_47_32	OFM tile1 byte offset (top-right tile) from OFM_REGION
0x012	NPU_SET_OFM_BASE2	extu_47_32	OFM tile2 byte offset (bottom-left tile) from OFM_REGION
0x013	NPU_SET_OFM_BASE3	extu_47_32	OFM tile3 byte offset (bottom-right tile) from OFM_REGION
0x014	NPU_SET_OFM_STRIDE_X	exts_47_32	OFM byte stride between horizontal values

cmd1	Enumerator	Parameter	Payload data
0x015	NPU_SET_OFM_STRIDE_Y	exts_47_32	OFM byte stride between vertical values
0x016	NPU_SET_OFM_STRIDE_C	exts_47_32	OFM byte stride between channel blocks (of 16 bytes each block)
0x017-0x01F	-	-	-
0x020	NPU_SET_WEIGHT_BASE	extu_47_32	Weight stream byte offset in WEIGHT_REGION
0x021	NPU_SET_WEIGHT_LENGTH	0	Weight stream byte length (unsigned 32 bits)
0x022	NPU_SET_SCALE_BASE	extu_47_32	Scale and bias stream input byte offset from SCALE_REGION
0x023	NPU_SET_SCALE_LENGTH	0	Scale and bias stream input byte length (unsigned 20 bits)
0x024	NPU_SET_OFM_SCALE	shift (6-bit unsigned)	Unsigned scale (32 bit). Used by average pool with pad=0, elementwise MUL, ADD, SUB, ABS. **Note:** For 32-bit operations scale is not applied but shift is.
0x025	NPU_SET_OPA_SCALE	shift (6-bit unsigned)	Unsigned input scale. The format depends on the IFM_PRECISION register: • If IFM scale mode is 0, then shift is ignored and scale is 16 bit. • If IFM scale mode is 1 or 2, then shift is 6 bit and scale is 32 bit or 16 bit, respectively.
0x026	NPU_SET_OPB_SCALE	Reserved	Unsigned input scale. The format depends on the IFM_PRECISION register: • If IFM scale mode is 0, then scale is 16 bit. • If IFM scale mode is 1 or 2, then this register is not used.
0x027-0x02F	-	-	-
0x030	NPU_SET_DMA0_SRC	extu_47_32	DMA user channel 0 source byte offset from DMA0_SRC_REGION
0x031	NPU_SET_DMA0_DST	extu_47_32	DMA user channel 0 destination byte offset from DMA0_DST_REGION
0x032	NPU_SET_DMA0_LEN	extu_47_32	DMA user channel 0 transfer length in bytes for ID mode.
0x033	NPU_SET_DMA0_SKIP0	extu_47_32	Byte distance to skip after each inner (1D) transfer (2D/3D mode), any alignment
0x034	NPU_SET_DMA0_SKIP1	extu_47_32	Byte distance to skip after each 2D transfer (3D mode), any alignment
0x035-0x03F	-	-	-
0x080	NPU_SET_IFM2_BASE0	extu_47_32	IFM2 tile0 byte offset (top-left tile) from IFM2_REGION start
0x081	NPU_SET_IFM2_BASE1	extu_47_32	IFM2 tile1 byte offset (top-right tile) from IFM2_REGION start
0x082	NPU_SET_IFM2_BASE2	extu_47_32	IFM2 tile2 byte offset (bottom-left tile) from IFM2_REGION start
0x083	NPU_SET_IFM2_BASE3	extu_47_32	IFM2 tile3 byte offset (bottom-right tile) from IFM2_REGION start
0x084	NPU_SET_IFM2_STRIDE_X	exts_47_32	IFM2 byte stride between horizontal values
0x085	NPU_SET_IFM2_STRIDE_Y	exts_47_32	IFM2 byte stride between vertical values

cmd1	Enumerator	Parameter	Payload data
0x086	NPU_SET_IFM2_STRIDE_C	exts_47_32	IFM2 byte stride between channel blocks (of 16 bytes per block)
0x087-0x08F	-	-	-
0x090	NPU_SET_WEIGHT1_BASE	extu_47_32	Weight stream byte offset in WEIGHT_REGION
0x091	NPU_SET_WEIGHT1_LENGTH	0	Weight stream byte length (unsigned 32 bits)
0x092	NPU_SET_SCALE1_BASE	extu_47_32	Scale and bias stream input byte offset from SCALE_REGION
0x093	NPU_SET_SCALE1_LENGTH	0	Scale and bias stream input byte length (unsigned 20 bits)
0x094-0x09F	-	-	-

4.7 Weight stream format

The weight stream format encodes a sequence of signed weight values in the range -255 to +255. The weights are stored in a lossless compressed format.

The compression encodes sequences of zeros efficiently. Nonzero weight values are compressed using Golomb-Rice coding and a configurable lookup table. The weight stream is made from several bitstream slices, a slice header, and some Variable Length Coded (VLC) symbols. The VLC symbols are grouped into chunks. For each slice, the compression parameters are specified in the slice header and then kept for the duration of the slice.

4.7.1 Bit order convention

In the weight stream, all bits are stored in ascending bit number order. The LSB is therefore the first bit read in a byte.

Syntax elements are stored with the LSB first. Therefore, writing `0b10010` or `0x12`, then `0b1011` or `0xB`, then `0b1010101` or `0xAB`, stores `0b10101011 01110010` from MSB to LSB. Therefore, the content of the first byte is `0b01110010` or `0x72`, and the content of the second byte is `0b10101011` or `0xAB`.

4.7.2 Weight stream structure and slice header syntax

The slice header indicates to the NPU when to switch coding mode. Using an extended header, the slice header can optionally be used to reload the palette (lookup table).

The encoder decides the frequency of slice headers. A higher frequency is a trade-off between improving the compression ratio when switching coding mode and the cost of inserting a header. Adding a slice header also affects the decoding throughput, particularly when a header signals a reload of the palette.

The following figure shows an example weight stream payload.

Figure 4-1: Example weight stream payload

Header	Palette	Chunks	Header	Chunks

The following example specifies the high-level weight bitstream structure and the slice header syntax. The number of bits used in the bitstream is listed next to each symbol.

```
weight_stream() {                                   // -
  while( !end_of_stream() ) {                       // -
    zdiv                                            // 3 bit
  if (zdiv == 7) {                                  // -
    while (!byte_aligned() )                        // -
   bytealign                                        // 1 bit
      } else {                                      // -
    slice_header()                                  // -
       chunks()                                     // -
      }                                             // -
  }                                                 // -
  assert( word_aligned() )                          // -
}                                                   // -

slice_header() {                                    // -
  slicelen                                          // 15 bits
  slice_length = slicelen + 1                       // -
  wdiv                                              // 3 bits
  wtrunc                                            // 1 bit
  newpal                                            // 1 bit
  if (newpal) {                                     // -
    dirofs                                          // 5 bits
    palsize                                         // 5 bits
    palbits                                         // 3 bits
    palette_size = palsize==0 ? 0 : palsize + 1     // -
    palette_bits = palbits + 2                      // -
    for (i = 0; i <palette_size; i++)               // -
   palette[I]                                       // pallette_bits
  }                                                 // -
}                                                   // -
```

Note

The `byte_aligned()` function returns true if the current bit position is on a byte boundary, otherwise the return value is false. Similarly, the `word_aligned()` function returns true if the current bit position is on a 128-bit boundary. The `end_of_stream()` function returns true if the weight stream has reached the end.

The following table lists the symbols in this bitstream and their meanings.

Table 4-114: Bitstream symbols

Symbol	Valid values	Meaning	
`zdiv`	0-3, 6, or 7	**0-3**	Specifies the zero-run GRC divisor to be 1<<zdiv.
		6	Indicates that zero run compression is not used for the slice.
		7	Indicates that several bits follow to byte-align the next syntax element. This is used at the end of the weight stream to make sure the weight stream has a length that is a multiple of 128-bit words.
`bytealign`	1	Padding used to align the end of the stream. Must have the value 1.	

Symbol	Valid values	Meaning
`slicelen`, `slice_length`	All	Number of weights in this slice. In alternate mode, this is the number of nonzero weights.
`wdiv`	0-5 or 7	Weight GRC divisor. Possible values are: **0-5** Specifies the weight index GRC divisor to be 1<<wdiv. **7** Uncompressed mode
`wtrunc`	All	If this is set, the weight GRC unary length is truncated to 2.
`newpal`	All	If this is set, a new palette mode is configured. If this is not set, then `dirofs`, `palsize`, `palbits`, and `palette[i]` keep the values from the previous slice. This must be set for the first slice in a stream.
`dirofs`	All	Direct mode offset. For more information about direct mode, see 4.7.3.1 Palette mode and direct mode on page 91.
`palsize`, `palette_size`	All	Indicates the number of entries in the palette. A value of 0 means direct mode where the palette is not used.
`palbits`, `palette_bits`	All	If the palette is used (`palette_size>0`), then `palette_bits` indicates the precision in bits of each palette entry. In direct mode (`palette_size==0`), then `palette_bits` indicates the precision used in uncompressed mode.
`palette[i]`	All	Weight value for palette entry with index i. The weight value is stored in sign-magnitude format. The LSB of `palette[i]` is the sign and the remainder of the bits (bit palette_bits-1 down to bit 1) indicate the absolute level. The weight value is calculated with the following formula: weight_value = palette[i] & 1 ? -(palette[i]>>1) : (palette[i]>>1)

4.7.3 Coding modes

There are a few different coding modes.

4.7.3.1 Palette mode and direct mode

The weight stream encodes compressed weight indices. A weight index is a 9-bit unsigned integer in the range 0 to 511 which represents different weight values.

If the weight index is less than `palette_size`, the weight index is used as an index into the palette, and the weight value is found in the palette entry for that index. Otherwise (if the weight index is greater than or equal to the palette_size), the weight value is calculated directly from the weight index using a formula as indicated below. The first mode is called palette mode and the latter mode is called direct mode.

```
if ( weight_index < palette_size )
 tmp = palette[weight_index]
else
 tmp = weight_index - palette_size + dirofs
weight_value = tmp&1 ? -(tmp>>1) : +(   tmp>>1)
```

4.7.3.2 Weight index coding

Weight indices are either Golomb-Rice coded or uncompressed as indicated by `wdiv`.

4.7.3.2.1 Golomb-Rice coding

In Golomb-Rice coding, the weight index is represented as a quotient and a remainder.

Golomb-Rice coding is represented as follows:

```
wq = weight_index >> wdiv
wr = weight_index & ((1<<wdiv)-1)
```

The quotient `wq` must be less than or equal to 31. If `wtruncis` is set, then `wqmust` be less than or equal to 2. It is the responsibility of the encoder to select the `wdiv` parameter so that this is the case. The quotient is unary coded in the bitstream and the remainder is stored as an unsigned binary in `wdiv` bits. Unary coding is a variable length coding where numbers are coded as zero-terminated strings of ones as follows:

Table 4-115: Example of unary coding structure

wq	Unary coding
0	0
1	10
2	110
3	1110
...	...
31	11111111111111111111111111111110

If truncated coding (`wtrunc`) is set, the coding is as follows:

Table 4-116: Truncated unary coding

wq	Unary coding
0	0
1	10
2	11

The unary part is coded in the `wunary0` and `wunary1` syntax elements and the remainder is encoded in the `wremain` syntax element as described later.

4.7.3.2.2 Uncompressed coding

If `wdiv` indicates uncompressed coding, the `weight_index` is coded directly as an unsigned binary integer.

The number of bits used, `uncompressed_bits`, is derived from the palette size when the palette is non-empty. If the palette is empty, then `palette_bits` is repurposed to indicate the uncompressed precision. This behavior is summarized in the following formula:

```
uncompressed_bits = palette_size>0 ? ceil( log2(palette_size) ) :
palette_bits
```

The uncompressed weight index is coded in the `wremain` syntax element as described later.

4.7.3.3 Alternating mode (zero-run coding)

If `zdiv<4`, alternating mode is enabled. This mode is beneficial if weights with a value of 0 are frequent in the weight stream.

Let `n` be the number of nonzero weight values and let the array `weight_values` (of length `n`) be the sequence of nonzero weight values. Let the array `zruns` (of length n+1) be the sequence of zero run lengths between the nonzero weights (`zruns[0]` is the initial zero run length and `zruns[n]` is the ending zero run length). For example, consider the following weight sequence:

```
0, 5, 6, 0, 0, 0, 7, 0
```

You then code the following:

```
n = 3
weight_values = {5, 6, 7}
zruns = {1, 0, 3, 1}
```

From the prior code, the original weight sequence can be reconstructed.

The weights values and the `zrun` values are potentially coded in multiple slices. The initial zero run is only coded for slices with `newpal` set (and in particular for the first slice in the weight stream, since the first slice must have `newpal` set). A slice is only allowed to change between alternating and non-alternating coding if `newpal` is set. So, a slice that does not set `newpal` must be of the same kind (alternating or non-alternating) as the previous slice.

The following formulas give the number of coded weights values and the number of `zrun` values in a slice.

```
n_weight_values = slice_length
n_zruns = slice_length + newpal
```

For example, say we have 3 slices and all of them are coded using alternating mode and assume that `newpal` is set for slice 1 and slice 3

```
0, 5, 6, 0, 0, 0, 7, 0, 8, 9, 10, 0, 11, 12, 13, 14, 15
<-- slice 1 ---> <-- slice 2 ----> <-- slice 3 ----->
```

then we code

```
Slice 1 (newpal=1):
 slice_length = 2
 weight_values = {5, 6 }
 zruns = {1, 0, 3 }
Slice 2 (newpal=0):
 slice_length = 4
 weight_values = {7, 8, 9, 10 }
 zruns = {1, 0, 0, 1 }
Slice 3 (newpal=1):
 slice_length = 5
 weight_values = {11, 12, 13, 14, 15 }
 zruns = {0, 0, 0, 0, 0, 0 }
```

The nonzero weight values are coded using the direct and palette modes described in previous sections. The zero run values are Golomb-Rice coded using `1<<zdiv` as divisor, so, each zero run value is represented as a quotient and a remainder as follows:

```
zq = zrun >> zdiv
zr = zrun & ((1<<zdiv)-1)
```

Note that unlike for `wq`, there is no upper bound for `zq`. That means zero runs of arbitrary length can be coded.

4.7.4 Chunk syntax

There is a specific chunk syntax structure.

After the slice header follows some chunks to encode the weight indices and, if alternating mode, the zero runs that belong to the slice. Each chunk encodes from 0-12 weight indices and from 0-12 zero run values. These values are generally not the same number. The reason for this is that the number of values depends on the quotient values, that is, the unary lengths. If the unary lengths are long, then fewer values fit in the chunk compared to if the unary length is short.

The number of chunks in the slice is not known in advance since this number depends on the weight and `zrun` values.

In alternating mode, a sort of flow control is used to make sure the number of coded weight indices and `zrun` values are roughly the same. This is achieved by tracking a balance (number of weight indices minus number of `zrun` values so far). If the balance is greater than or equal to 8, then only `zrun` values are included in the chunk (so that `zrun` can catch up). Similarly, if the balance is less than 0, then only weight indices are included in the chunk. If the balance is between 0 and 7, then both weights and `zrun` values are included in the chunk.

The Golomb-Rice remainders are pipelined to the chunk after the chunk containing the corresponding quotient values.

The chunk bitstream syntax and parsing process are described below. The output of the process is the `weight_indices` and the `zruns` arrays.

```
chunks() {                                                          // -
 w_cnt = slice_length                                               // -
 z_cnt = slice_length + new_pal                                     // -
 zunary_len = zdiv<3 ? 12 : 8                                       // -
 alternating_mode = zdiv<4                                          // -
 uncompressed_mode = wdiv==7                                        // -
 wremain_bits = uncompressed_mode ? uncompressed_bits : wdiv        // -
 uncompressed_per_chunk = uncompressed_bits<=5 ? 12 : 8             // -
 wq = 0                                                             // -
 wq_i = 0                                                           // -
 wr_i = 0                                                           // -
 zq = 0                                                             // -
 zq_i = 0                                                           // -
 zr_i = 0                                                           // -
 prev_w_enable=0                                                    // -
 prev_z_enable=0                                                    // -
  do {                                                              // -
     // In alternating mode, make sure the rate of weight indices
     // and zrun are kept about the same.                           // -
     balance = wq_i - zq_i                                          // -
     w_enable = (balance<8 || !alternating_mode) && wq_i < w_cnt    // -
     z_enable = balance>=0 && alternating_mode && zq_i < z_cnt      // -
     if (w_enable && !uncompressed_mode)                            // -
       wunary0                                                      // 12 bits
     if (z_enable) {                                                // -
       zunary                                                       // zunary_len
       for(i=0; i<zunary_bits; i++) {                               // -
         if ( (zunary>>i)&1) ) {                                    // -
            zq++                                                    // -
         } else {                                                   // -
            zruns[zq_i++] = zq<<zdiv                                // -
            zq=0                                                    // -
       }                                                            // -
     }                                                              // -
    }                                                               // -
     if (w_enable && !uncompressed_mode) {                          // -
       wunary1_len = 0                                              // -
       for(i=0; i<12; i++)                                          // -
          if ( (wunary0>>i)&1 || wtrunc )                           // -
            wunary1_len++                                           // -
       wunary1                                                      // wunary1_len
       for(i=0, j=0; i<12 && wq_i<w_cnt; i++) {                     // -
          c=0                                                       // -
          if ( (wunary0>>i)&1 ) {                                   // -
            c = 1 + ((wunary1>>j)&1)                                // -
            j++                                                     // -
          }                                                         // -
          wq+=c                                                     // -
          if (c<2 || wtrunc) {                                      // -
            assert(wq<32)                                           // -
            weight_indices[wq_i++] = wq<<wdiv                       // -
            wq=0                                                    // -
          }                                                         // -
        }                                                           // -
      }                                                             // -
       if (w_enable && uncompressed_mode) {                         // -
          for(i=0; i<uncompressed_per_chunk && wq_i<w_cnt; i++) {   // -
            weight_indices[wq_i++] = 0                              // -
         }                                                          // -
       }                                                            // -
       // Remainders corresponding to the quotients
```

```
        // in the previous chunk                                      // -
        if (prev_w_enable) {                                          // -
           while( wr_i < prev_wq_i ) {                                // -
               wremain                                                // wremain_bits
               weight_indices[wr_i++] += wremain                      // -
          }                                                           // -
        }                                                             // -
        if (prev_z_enable) {                                          // -
          while( zr_i < prev_zq_i ) {                                 // -
               zremain                                                // zdiv
               zruns[zr_i++] += zremain                               // -
          }                                                           // -
        }                                                             // -
        prev_w_enable = w_enable                                      // -
        prev_wq_i = wq_i                                              // -
        prev_z_enable = z_enable                                      // -
        prev_zq_i = zq_i                                              // -
    } while( prev_w_enable || prev_z_enable )                         // -
```

4.7.5 Weight blocks and ordering

The Ethos™-U55 NPU must get weights in a certain order to function correctly. This process is described in this section.

The weights are also compressed, as described in section 4.7 Weight stream format on page 89. This section describes how the 1D array, that is the input to the weight encoder, is ordered.

Overview

The weights are not only reordered, padding is also inserted to align to full weight blocks that the weight decoder works on. Here padding is done by inserting weights that are 0 into the weight stream. Therefore, unless the stripe dimensions align perfectly to the internal work blocks of the NPU, the uncompressed weight stream is larger than the original weights.

The ordering is described below in pseudocode as nested loops. It is divided into depth-wise convolution, normal convolution with depth-first order, and normal convolution with part-kernel-first order, although they are in most ways similar where, for example, depth-wise with only some exception using the same order as part-kernel-first convolution, but removing the loops used to traverse ifm depth.

Depth-wise convolution

Table 4-117: Depth-wise convolution weight ordering

Inputs/outputs	Description	Range
	Input	
weights	3D array of 9-bit signed weights in 2's complement Dimensions: [ofm-z][ifm_z][kernel_x][kernel_y]	[-255..255]
	Stripe-dependent input	
ofm_depth	Number of ofm channels	[1..65536]
ofm_block_depth	Number of ofm channels per block	[1..128]

Inputs/outputs	Description	Range
kernel_width	Kernel width (before dilation)	[1..65536]
kernel_height	Kernel height (before dilation)	[1..65536]
kernel_x_dilation	Kernel x dilation by 2 enabled	Boolean
kernel_y_dilation	Kernel y dilation by 2 enabled	Boolean
kernel_split_size	Kernel decomposition size	[4,8]
	Configuration-dependent input	
ublk_depth	Microblock depth	[4,8]
	Output	
weight_stream	1D array of 9-bit signed weights	[-255..255]

Example code of weight ordering for depth-wise convolution.

```
decomp_w = kernel_split_size
if (kernel_x_dilation)
   decomp_w = decomp_w / 2
decomp_h = kernel_split_size
if (kernel_y_dilation)
   decomp_h = decomp_h / 2
w_idx = 0
for ( blk_z = 0; blk_z < ofm_depth; blk_z += ofm_block_depth )
 for ( kernel_x = 0; kernel_x < kernel_width; kernel_x += decomp_h )
  for ( kernel_y = 0; kernel_y < kernel_height; kernel_y += decomp_w )
   subkernel_w = min(kernel_width - kernel_x, decomp_w)
   subkernel_h = min(kernel_height - kernel_y, decomp_h)
   subkernel_size = ((subkernel_w * subkernel_h + 3) / 4) * 4
   blk_d = min(ofm_block_depth, ofm_depth - blk_z)
   for ( ublk_z = 0; ublk_z < blk_d; ublk_z += ublk_depth )
    for ( kernel_i = 0; kernel_i < subkernel_size; kernel_i++ )
     subkernel_x = kernel_i % subkernel_width
     subkernel_y = kernel_i / subkernel_width
     for ( z = 0; z < ublk_depth; z++ )
      kx = kernel_x + subkernel_x
      ky = kernel_y + subkernel_y
      ofm_z = blk_z + ublk_z + z
      padding = False
      if ( subkernel_y = subkernel_height ||
       ofm_z = ofm_depth )
      weight_stream[w_idx++] = 0
      else
      weight_stream[w_idx++] = weights[ofm_z][ky][kx]
```

Convolution - depth-first weight order

Table 4-118: Depth-first weight ordering

Inputs/outputs	Description	Range
	Input	
weights	4D array of 9-bit signed weights in 2's complement Dimensions: [ofm-z][ifm_z][kernel_x][kernel_y]	[-255..255]
	Stripe-dependent input	
ofm_depth	Number of ofm channels	[1..65536]

Inputs/outputs	Description	Range
ofm_block_depth	Number of ofm channels per block	[1..128]
kernel_width	Kernel width (before dilation)	[1..65536]
kernel_height	Kernel height (before dilation)	[1..65536]
kernel_x_dilation	Kernel x dilation by 2 enabled	Boolean
kernel_y_dilation	Kernel y dilation by 2 enabled	Boolean
kernel_split_size	Kernel decomposition size	[4,8]
ifm_depth	Number of IFM channels	[1..65536]
ifm_bitdepth	Bit depth for IFM elements	[8,16]
	Configuration-dependent input	
ublk_depth	Microblock depth	[4,8]
	Output	
weight_stream	1D array of 9-bit signed weights in 2's complement	[-255..255]

Example code for depth-first weight ordering.

```
decomp_w = kernel_split_size
if (kernel_x_dilation)
   decomp_w = decomp_w / 2
decomp_h = kernel_split_size
if (kernel_y_dilation)
   decomp_h = decomp_h / 2
ifm_block_depth = 32
if ( ifm_bitdepth == 16 )
   ifm_block_depth = 16
w_idx = 0
for ( blk_z = 0; blk_z < ofm_depth; blk_z += ofm_block_depth )
  for ( iblk_z = 0; iblk_z < ifm_depth; iblk_z += ifm_block_depth )
 for ( kernel_x = 0; kernel_x < kernel_width; kernel_x += decomp_h )
   for ( kernel_y = 0; kernel_y < kernel_height; kernel_y += decomp_w )
  subkernel_width = min(kernel_width - kernel_x, decomp_w)
  subkernel_height = min(kernel_height - kernel_y, decomp_h)
  subkernel_size = subkernel_width * subkernel_height
  blk_d = min(ofm_block_depth, ofm_depth - blk_z)
  for ( ublk_z = 0; ublk_z < blk_d; ublk_z += ublk_depth )
    for ( kernel_i = 0; kernel_i < subkernel_size; kernel_i++ )
   subkernel_x = kernel_i % subkernel_width
   subkernel_y = kernel_i / subkernel_width
   for ( iublk_z = 0; iublk_z < ifm_block_depth; iublk_z += 8 )
     for ( z = 0; z < ublk_depth; z++ )
    for ( iz = 0; iz < 8; iz++ )
      kx = kernel_x + subkernel_x
      ky = kernel_y + subkernel_y
      ifm_z = iblk_z + iublk_z + iz
      ofm_z = blk_z + ublk_z + z
      if ( ifm_z >= ifm_depth ||
       ofm_z >= ofm_depth )
     weight_stream[w_idx++] = 0
      else
     weight_stream[w_idx++] = weights[ofm_z][ifm_z][ky][kx]
```

Convolution - part-kernel-first weight order

Table 4-119: Part-kernel-first weight ordering

Inputs/outputs	Description	Range
	Input	

Inputs/outputs	Description	Range
weights	4D array of 9-bit signed weights in 2's complement Dimensions: [ofm-z][ifm_z][kernel_x][kernel_y]	[-255..255]
Stripe-dependent input		
ofm_depth	Number of ofm channels	[1..65536]
ofm_block_depth	Number of ofm channels per block	[1..128]
kernel_width	Kernel width (before dilation)	[1..65536]
kernel_height	Kernel height (before dilation)	[1..65536]
kernel_x_dilation	Kernel x dilation by 2 enabled	Boolean
kernel_y_dilation	Kernel y dilation by 2 enabled	Boolean
kernel_split_size	Kernel decomposition size	[4,8]
ifm_depth	Number of IFM channels	[1..65536]
ifm_bitdepth	Bit depth for IFM elements	[8,16]
Configuration-dependent input		
ublk_depth	Microblock depth	[4,8]
Output		
weight_stream	1D array of 9-bit signed weights in 2's complement	[-255..255]

Example code for part-kernel-first weight ordering.

```
decomp_w = kernel_split_size
if (kernel_x_dilation)
   decomp_w = decomp_w / 2
decomp_h = kernel_split_size
if (kernel_y_dilation)
   decomp_h = decomp_h / 2
ifm_block_depth = 32
if ( ifm_bitdepth == 16 )
   ifm_block_depth = 16
w_idx = 0
for ( blk_z = 0; blk_z < ofm_depth; blk_z += ofm_block_depth )
  for ( iblk_z = 0; iblk_z < ifm_depth; iblk_z += ifm_block_depth )
 for ( kernel_x = 0; kernel_x < kernel_width; kernel_x += decomp_h )
   for ( kernel_y = 0; kernel_y < kernel_height; kernel_y += decomp_w )
  subkernel_width = min(kernel_width - kernel_x, decomp_w)
  subkernel_height = min(kernel_height - kernel_y, decomp_h)
  subkernel_size = subkernel_width * subkernel_height
  if ( ifm_bitdepth == 16 )
    subkernel_size = ((subkernel_size + 1) / 2) * 2
    if ( ifm_bitdepth == 8 )
   subkernel_size = ((subkernel_size + 3) / 4) * 4
   iblk_d = min(16, ifm_depth - iblk_z)
   for ( iublk_z = 0; iublk_z < iblk_d; iublk_z += 8 )
     blk_d = min(ofm_block_depth, ofm_depth - blk_z)
     for ( ublk_z = 0; ublk_z < blk_d; ublk_z += ublk_depth )
    for ( kernel_i = 0; kernel_i < subkernel_size; kernel_i++ )
      subkernel_x = kernel_i % subkernel_width
      subkernel_y = kernel_i / subkernel_width
     for ( z = 0; z < ublk_depth; z++ )
       for ( iz = 0; iz < 8; iz++ )
      kx = kernel_x + subkernel_x
      ky = kernel_y + subkernel_y
      ifm_z = iblk_z + iublk_z + iz
```

```
       ofm_z = blk_z + ublk_z + z
       if ( subkernel_y = subkernel_height ||
        ifm_z = ifm_depth ||
        ofm_z = ofm_depth )
         weight_stream[w_idx++] = 0
       else
         weight_stream[w_idx++] = weights[ofm_z][ifm_z][ky][kx]
```

4.8 Operators and performance

This section provides information on supported data types, operators, and operations, and details the convolution and elementwise performance of the Ethos™-U55 NPU.

4.8.1 Supported data types and operators

The NPU design process supports the following data types and operators to enable a range of operations. The command-stream generator can construct additional operators.

Data types

The following data types and formats are supported.

Table 4-120: Supported data types

Data type	Range / values
Supported activation and weight combinations	Unsigned 8-bit activations with unsigned 8-bit weights. These allow unsigned zero point of range 0-255 on both activations and weights on a per-tensor basis.
	Signed 8-bit activations with signed 8-bit weights. These allow signed zero point of range -128 to +127 on activations per tensor, but not zero point on weights (weights are symmetric).
	Signed 16-bit activations with signed 8-bit weights. Both activations and weights are symmetric (zero point is not supported).
Output-channel bias-and-scale activations	8-bit activations per output-channel bias and scale 16-bit activations per output-channel bias and scale
Accumulator formats	32-bit accumulators, 40-bit accumulators, 16-bit floating point (s5.10) accumulators
Bit sizes	8x16-bit operations run at half the speed of 8x8-bit operations.
Tensor dimensions	Tensor height range 1–65536. Tensor width range 1–65536. Tensor depth range 1–65536.

Note

The zero-point data type and range must match the corresponding weight or activation data type and range. For example:

- For int8_t activations, the zero_point is also int8_t and both are in the range [-128, 127]. The minimum value of the range activation-zero_point is -128 - (+127) = -255 and the maximum value is +127 - (-128) = +255.
- For uint8_t activations, the zero_point is also uint8_t and both are in the range [0,255]. The minimum value of the range activation-zero_point is 0 - (+255) = -255 and the maximum value is +255 - (0) = +255.

Note

The tensor size is limited by available memory; therefore, tensor dimensions cannot all have maximum values at the same time.

Operators

The command-stream generator can combine features of the NPU to create the following additional operators.

Table 4-121: Command-stream generated operators

Operator	Construction
Concat	The Concatenation operator is constructed by using strides to lay out tensors.
ExpandDims	The ExpandDims operator does not move data for packed NHWC, but adds a '1' dimension.
GRU	The Gated Recurrent Unit (GRU) operation is constructed from vector products and point-wise MUL, ADD, and SUB.
Identity	Identity can be realized as a 1x1 average pool with a 1x1 stride. This can be useful for rearranging data.
Logistic	This is a different name for sigmoid activation, both are `1/(1+exp(-x))`.
LSTM	The Long Short-Term Memory (LSTM) operation is constructed from vector products and point-wise MUL, ADD, and SUB.
Pack	Same as Stack (see below).
Reshape	The Reshape operator does not move data for packed NHWC, but reinterprets the dimensions.
Split	The Split operator is the inverse of Concatenate and can be constructed by using strides to extract a subtensor.
Squeeze	The Squeeze operator does not move data for packed NHWC, but removes a '1' dimension.
Stack	The Stack operator is constructed by using strides. For example, `stack NxHWC` tensors to obtain one NHWC tensor.
Unpack	Same as Unstack (see below).
Unstack	The inverse of Stack. This can be constructed by using strides to extract the lower dimension subtensors.
Resize_Bilinear	For a bilinear x2 upscale, this can be achieved by performing a nearest-neighbor upscale combined with a 2x2 average pool.
BatchRenorm	Average pool 1x1 with per-channel scale and bias to rescale data at inference time with fixed scaling only.
StridedSlice, 1-strides only	StridedSlice with strides of 1 extracts a subtensor and can be implemented in NHWC format. (StridedSlice with strides not equal to 1 are not supported.)

4.8.2 Operations

The following tables provide details of parameters that enable a number of convolution, depth-wise convolution, pooling, vector-product, elementwise, and reduction operations.

Convolution operations

A convolution has a weight matrix of size `HxWxICxOC`

where

HxWxIC

is the size of the convolution kernel,

IC

the number of input channels, and

OC

the number of convolutions to apply (= number of output channels).

Table 4-122: Convolution operations

Parameter	Range / values
Kernels	1 <= kernel_x*kernel_y <= 64*64 1 <= kernel_y <= 64 (kernel limit applies after any kernel dilation) The sum of absolute weights must not exceed 127*65536.
Precision	Weight types: {int8, uint8} {IFM types} → {OFM types} supported combinations: {uint8, int8, int16} → {uint8, int8, int16, int32}, any pairing
Stride	1 <= stride_x <= 3 1 <= stride_y <= 3
Kernel dilation	1x1, 1x2, 2x1, 2x2
Input upscale	None, 2x2 (nearest neighbor, insert zeros). A 2x2 upscale must use a stride of 1x1.
Input padding	0-31 top/left, 0-32 bottom/right
Fused activation	Available activations for {activation type}: {int8, uint8, int16}: None, ReLU, ReLUX, tanh, sigmoid, LUT {int32}: None (linear output only) If LUT is not used, the activation and OFM type must match.
Weight order	Depth-first order, part-kernel-first order (either order can be used for any IFM depth)
Scaling	Per output-channel scale and bias parameters
Accumulators	fp(s5.10), int32, int40

Note

The restrictions in the range / values column allow 2D convolutions of size up to 64x64 and 1D convolutions of size up to 1x4096. The condition on the sum of absolute weights ensures that a 32-bit accumulator does not overflow for 8-bit activation values and a 40-bit accumulator does not overflow for 16-bit activation values.

Depth-wise convolution operations

Depth-wise convolutions have a matrix of `HxWxC`, where the kernel of size `HxW` is applied to each channel independently. Only one kernel is applied to each layer (`depth_multiplier=1`).

Table 4-123: Depth-wise convolution operations

Parameter	Range / values
Kernels	1 <= kernel_x*kernel_y <= 64*64 1 <= kernel_y <= 64 (kernel limit applies after any kernel dilation) The sum of absolute weights must not exceed 127*65536.
Precision	Weight types: {int8, uint8} {IFM types} → {OFM types} supported combinations: {uint8, int8, int16} → {uint8, int8, int16, int32}, any pairing
Stride	1 <= stride_x <= 3 1 <= stride_y <= 3
Dilation	1x1, 1x2, 2x1, 2x2
Input scale	None, 2x2 (nearest neighbor, insert zeros). A 2x2 upscale must use a stride of 1x1.
Input padding	0-31 top/left, 0-32 bottom/right
Fused activation	Available activations for {activation type}: {int8, uint8, int16}: None, ReLU, ReLUX, tanh, sigmoid, LUT {int32}: None (linear output only) If LUT is not used, the activation and OFM type must match.
Depth multiplier	1
Scaling	Per output-channel scale and bias parameters
Accumulators	fp(s5.10), int32, int40

Note

The restrictions in the range / values column allow 2D convolutions of size up to 64x64 and 1D convolutions of size up to 1x4096. The condition on the sum of absolute weights ensures that a 32-bit accumulator does not overflow for 8-bit activation values and a 40-bit accumulator does not overflow for 16-bit activation values.

Pooling operations

Pooling operations are applied independently to each channel.

Table 4-124: Pooling operations

Parameter	Range / format
Kernels	Average pool with padding (for example SAME padding): 1 <= kernel_x <= 8 1 <= kernel_y <= 8 Average pool without padding and max pool, any padding: 1 <= kernel_x*kernel_y <= 256*256 1 <= kernel_y <= 256
Precision	Average pool without padding (VALID type): {IFM types} → {OFM types} supported combinations: {uint8, int8, int16} → {uint8, int8, int16} (any pairing) Average pool with padding or max pool. OFM type must equal IFM type. Supported types: {int8, uint8, int16}
Stride	1 <= stride_x <= 3 1 <= stride_y <= 3
Input upscale	Average pool: none, 2x2 nearest neighbor OR 2x2 insert zeros. Max pool: none, 2x2 nearest neighbor (only for 2x2 mode). A 2x2 upscale must use a stride of 1x1.
Input padding	Average pool: 0-3 top/left, 0-4 bottom/right Max pool: 0-127 top/left, 0-128 bottom/right
Fused activation	Available activations for {activation type}: {int8, uint8, int16}: None, ReLU, ReLUX, tanh, sigmoid, LUT If LUT is not used, the activation and OFM type must match.
Scaling	Average pool with padding or Max pool has no scaling. Average pool with pad=0 has selectable per-channel scale and bias or global scale.
Accumulators	All pooling: int32 Average pool with no padding: int32, int40

Vector-product operations

The kernel for a (fully connected) vector product is `1x1xIC`, where IC is the number of input channels. Multiple output vector products with the same weights can be executed in batches of up to eight.

Vector product is implemented as a convolution 2D with a 1x1 kernel size.

Table 4-125: Vector-product operations

Parameter	Range / format
Kernels	1x1x1 to 1x1x64K vector product
Precision	Weight types: {int8, uint8} {IFM types} → {OFM types} supported combinations: {uint8, int8, int16} → {uint8,int8,int16,int32}, any pairing
Fused activation	Available activations for {activation type}: {int8, uint8, int16}: None, ReLU, ReLUX, tanh, sigmoid, LUT {int32}: None (linear output only) If LUT is not used, the activation and OFM type must match.
Scaling	Per output-channel scale and bias parameters
Accumulators	int32, int40

Elementwise operations

The following operations include both unary element-wise (or point-wise) and binary elementwise operations, which support two IFMs to produce one OFM.

Table 4-126: Elementwise operations

Parameter	Range / format
Kernels	Binary operations: Multiply, Add, Subtract, Minimum, Maximum, SHR, SHL. Unary operations: ABS, Leaky ReLU, CLZ.
Precision	Multiply, Add, Subtract {IFM}→{OFM}: {uint8, int8, int16 int32} → {uint8, int8, int16, int32}, any pairing Minimum, Maximum, LReLU, ABS: IFM and OFM must be of the same type, one of: {int8, uint8, int16} SHR {IFM}→{OFM}: {int32}→{int8, uint8, int32}, any pairing CLZ and SHL: {int32}→{int32} only
Broadcast (for binary tensor operations)	Operand IFM2 can be one of the following: (a) A scalar constant broadcast to all elements for 8-bit or 16-bit IFM (scalar constant is not supported for a 32-bit IFM). (b) A tensor whose dimensions are either 1 or match IFM1. If (b), any dimension that is broadcast to match the dimension of IFM1.

Parameter	Range / format
Operand order	Selectable if IFM2 is the first or second operand (A or B).
Fused activation	Available activations for {activation type}: {int8, uint8, int16}: None, ReLU, ReLUX, tanh, sigmoid, LUT {int32}: None (linear output only) If LUT is not used, the activation and OFM type must match.
Input scaling	For ADD and SUB (only) the following input scales are supported when neither the IFM nor the activation type is 32-bit: 1. 16-bit input scale on elementwise ADD and SUB operands. 2. 32-bit input scale applied to only input (fixed shift for the other).
Output scaling	Global 32-bit output scale on elementwise MUL, ADD, SUB, ABS, LReLU, ABS, SHR. Leaky ReLU scales only negative inputs.

Reduction operations

The following operations the supported reduction operations for REDUCE_SUM, which reduce the channel dimension from an HWC tensor to an HW1 tensor.

Table 4-127: Reduction operations

Parameter	Range / format
Kernels	REDUCE_SUM
Precision	Supported {IFM types} → {OFM types}: {uint8, int8, int16, int32} → {int32} (any pairing)
Input upscale	1x1 only
Input padding	None
Fused activation	Available activations for {activation type}: {int8, uint8, int16}: None, ReLU, ReLUX, tanh, sigmoid, LUT {int32}: None (linear output only). If LUT is not used, the activation and OFM type must match.
Scaling	Global 32-bit scale.
Accumulators	int32, int40

4.8.3 Convolution performance

The following tables detail the convolution performance of the Ethos™-U55 NPU by configuration.

The convolution performance for the different configurations of the NPU depends on the operation used, such as the kernel height (kh), kernel width (kw). In addition, it also depends on the dimensions of the tensors being processed.

Note

The purpose of these tables is to explain the architectural limitations of the MAC utilization of different convolutional operations. If layers are broken into small jobs, there may be more overhead at top level.

For shallow 1x1 convolutions, where IFM depth is <64 or OFM depth is <16, the overall performance is limited by the output and memory bandwidth.

Convolution performance of the Ethos™-U55$_{256}$

In the following tables k, h, w, d, and n should be integers to achieve the MACs per cycle as specified for the operation. For any non-integer values, the hardware effectively rounds up this value and the extra MACs computed as a consequence are lost.

Note

Cells marked "WB" denote weight-bound values. The actual performance of weight-bound layers depends on the number of weights that can be compressed by the weight decoder per cycle. This number is affected by the compression ratio and the bandwidth of the memory available for the weights. (The capacity of the weight decoder itself is unaffected.)

Table 4-128: Convolution performance for 8-bit activations

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV2D (depth first)	kh*kw=1*k	2*h	2*w	8*d	32*n	256
CONV2D (kernel first)	kh*kw=4*k	2*h	2*w	8*d	8*n	256
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	8*d	32*n	256
CONV1D (kernel first)	kh=1 kw=4*k	1	4*w	8*d	8*n	256
Fully connected	kh=1 kw=1	1	1	8*d	32*n	WB
DepthwiseConv2D	kh*kw=4*k	2*h	2*w	8*d	8*n	32
DepthwiseConv1D	kh=1 kw=4*k	1	4*w	8*d	8*n	32

Table 4-129: Convolution performance for 16-bit activations

16-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV2D (depth first)	kh*kw=1*k	2*h	2*w	8*d	16*n	128
CONV2D (kernel first)	kh*kw=2*k	2*h	2*w	8*d	8*n	128
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	8*d	16*n	128
CONV1D (kernel first)	kh=1 kw=2*k	1	4*w	8*d	8*n	128
Fully connected	kh=1 kw=1	1	1	8*d	16*n	WB
DepthwiseConv2D	kh*kw=4*k	2*h	2*w	8*d	8*n	16
DepthwiseConv1D	kh=1 kw=4*k	1	4*w	8*d	8*n	16

Convolution performance of the Ethos™-U55$_{128}$

Table 4-130: Convolution performance for 8-bit activations

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV (depth first)	kh*kw=1*k	h	2*w	8*d	32*n	128
CONV (kernel first)	kh*kw=4*k	h	2*w	8*d	8*n	128
Fully connected	kh=1 kw=1	1	1	8*d	32*n	WB
DepthwiseConv	kh*kw=4*k	h	2*w	8*d	8*n	16

Table 4-131: Convolution performance for 16-bit activations

16-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV (depth first)	kh*kw=1*k	h	2*w	8*d	16*n	64
CONV (kernel first)	kh*kw=2*k	h	2*w	8*d	8*n	64
Fully connected	kh=1 kw=1	1	1	8*d	16*n	WB
DepthwiseConv	kh*kw=4*k	h	2*w	8*d	8*n	8

Convolution performance of the Ethos™-U55$_{64}$

Table 4-132: Convolution performance for 8-bit activations

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV (depth first)	kh*kw=1*k	h	w	8*d	32*n	64
CONV (kernel first)	kh*kw=4*k	h	w	8*d	8*n	64
Fully connected	kh=1 kw=1	1	1	8*d	32*n	WB
DepthwiseConv	kh*kw=4*k	h	w	8*d	8*n	8

Table 4-133: Convolution performance for 16-bit activations

16-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV (depth first)	kh*kw=1*k	h	w	8*d	16*n	32
CONV (kernel first)	kh*kw=2*k	h	w	8*d	8*n	32
Fully connected	kh=1 kw=1	1	1	8*d	16*n	WB
DepthwiseConv	kh*kw=4*k	h	w	8*d	8*n	4

Convolution performance of the Ethos™-U55$_{32}$

Table 4-134: Convolution performance for 8-bit activations

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV (depth first)	kh*kw=1*k	h	w	4*d	32*n	32
CONV (kernel first)	kh*kw=4*k	h	w	4*d	8*n	32

8-bit activation						
Operation	Kernel size	OFM height	OFM width	OFM depth	IFM depth	MACs per cycle
Fully connected	kh=1 kw=1	1	1	4*d	32*n	WB
DepthwiseConv	kh*kw=4*k	h	w	4*d	8*n	4

Table 4-135: Convolution performance for 16-bit activations

16-bit activation						
Operation	Kernel size	OFM height	OFM width	OFM depth	IFM depth	MACs per cycle
CONV (depth first)	kh*kw=1*k	h	w	4*d	16*n	16
CONV (kernel first)	kh*kw=2*k	h	w	4*d	8*n	16
Fully connected	kh=1 kw=1	1	1	4*d	16*n	WB
DepthwiseConv	kh*kw=4*k	h	w	4*d	8*n	2

4.8.4 Elementwise performance

The following tables detail the elementwise performance of the Ethos™-U55 NPU by configuration.

The performance of elementwise operations depends on the configuration of the NPU, as well as which operation is performed as shown in the following tables. Note that some operations are bound by the bandwidth required to read and write the operations to external SRAM.

Elementwise performance of the Ethos™-U55 NPU

Table 4-136: Operations per cycle for 8-bit activations

Ethos™-U55 configuration	LReLU, ABS	MIN, MAX	MUL	Simple ADD, SUB	Advanced ADD, SUB	LUT, tanh, sigmoid
32	0.5	0.5	0.33	0.25	0.17	0.5
64	1	1	0.67	0.5	0.33	1
128	2	2	1.33	1	0.67	1
256	4	2.67[1]	2.67	2	1.33	1

Table 4-137: Operations per cycle for 16-bit activations

Ethos™-U55 configuration	LReLU, ABS	MIN, MAX	MUL	Simple ADD, SUB	Advanced ADD, SUB	LUT, tanh, sigmoid
32	0.5	0.5	0.33	0.25	0.17	0.5
64	1	1	0.67	0.5	0.33	1
128	2	1.33[1]	1.33	1	0.67	1
256	2[1]	1.33[1]	1.33[1]	1.33[1]	1.33	1

Table 4-138: Operations per cycle for int32

Ethos™-U55 configuration	CLZ	SHR, SHL
32	0.5	0.33

[1] This value is memory-bound.

Ethos™-U55 configuration	CLZ	SHR, SHL
64	1	0.67
128	1[1]	0.67[1]
256	1[1]	0.67[1]

4.9 Block based operation

Due to limited internal storage, the NPU must break down an operation into smaller jobs.

The stripe is divided into one or more blocks and jobs scheduled by the hardware are processed one block at a time. The size of each block is specified in the command stream, each block follows the restrictions described in this section. If the block is not a multiple of the stripe size, the hardware runs partial blocks at the edge of the stripe.

Output feature map

The NPU generates the *Output Feature Map* (OFM) of an operation in blocks which repeat in z, x, y order over the OFM. The size of each OFM block must not exceed the size of the available *SHared RAM* (SHRAM).

Each block is configured in the command stream according to the following restrictions:

- OFM_BLOCK_WIDTH must be in the range 1-64 and a multiple of the MIN_BLOCK_WIDTH.
- OFM_BLOCK_HEIGHT must be in the range 1-32 and a multiple of the MIN_BLOCK_HEIGHT.
- OFM_BLOCK_DEPTH must be in the range 1-128 and a multiple of MIN_BLOCK_DEPTH.
- If OFM_BLOCK_DEPTH is not a multiple of 16, then OFM_DEPTH <= OFM_BLOCK_DEPTH.

The minimum block sizes are listed in the following table.

Table 4-139: Minimum block sizes

Configuration	MIN_BLOCK_HEIGHT	MIN_BLOCK_WIDTH	MIN_BLOCK_DEPTH
32	1	1	4
64	1	1	8
128	1	2	8
256	2	2	8

Input feature map

To generate an OFM block, the NPU reads one or more *Input Feature Map* (IFM) blocks. An upper limit on the size of an IFM block is derived from the OFM block size and the operation being performed, as listed in the following table.

Note

The size of the IFM and OFM blocks must not exceed the size of the available SHRAM. For more information about the size of the available SHRAM, see 4.9.1 Internal shared RAM on page 112.

Table 4-140: IFM block size limit

Dimension	OFM block size and operation
IFM_BLOCK_HEIGHT	ALIGN_HEIGHT(min(ifm_get_height(OFM_BLOCK_HEIGHT, min(kernel_split_size, dilated_kernel_height)), ifm_get_height(OFM_HEIGHT, dilated_kernel_height - PAD_TOP - PAD_BOTTOM))
IFM_BLOCK_WIDTH	ALIGN_WIDTH(min(ifm_get_width(OFM_BLOCK_WIDTH, min(kernel_split_size, dilated_kernel_width)), ifm_get_width(OFM_WIDTH, dilated_kernel_width - PAD_LEFT - PAD_RIGHT))
IFM_MEMBLK_DEPTH	OFM_BLK_DEPTH for a depth-wise convolution, max or average pooling and elementwise operations ALIGN(min(32, IFM_DEPTH), 8) for conv2d, fully connected or reduce_sum with 8-bit activations and kernel_weight_order=0 ALIGN(min(16, IFM_DEPTH), 8) for conv2d, fully connected or reduce_sum with 8-bit activation and kernel_weight_order=1 ALIGN(min(16, IFM_DEPTH), 4) for conv2d, fully connected or reduce_sum with 16-bit activation tensor ALIGN(min(8, IFM_DEPTH), 2) for reduce_sum with 32-bit activation

The definitions used in the preceding table are:

- ALIGN(x, n) = (int)ceil(x/(float)n)*n = (x + (n-1)) &~ (n-1)
- ALIGN_HEIGHT(h) = ALIGN(h, MIN_BLOCK_HEIGHT)
- ALIGN_WIDTH(w) = ALIGN(w, MIN_BLOCK_WIDTH)
- ifm_get_height(ofm_height, border_height) = (int)ceil(((ofm_height-1)*kernel_y_stride + border_height)/(float)upscaling_factor_y)
- ifm_get_width(ofm_width, border_width) = (int)ceil(((ofm_width-1)*kernel_x_stride + border_width)/(float)upscaling_factor_x)
- dilated_kernel_height = (kernel_height-1)*kernel_y_dilation+1, dilated_kernel_width = (kernel_width-1)*kernel_x_dilation+1
- upscaling_factor_x = upscaling_factor_y = (ifm_upscale_mode!=0 ? 2 : 1)

Block dependency

The output of one operation is the input of the following operation. The NPU breaks down the output and input operations into blocks, creating dependencies between each block. The dependency between blocks is specified in the command stream, which ensures the hardware writes the input data before the input data is read. Correctly setting the block dependency allows the hardware to run two operations back to back more efficiently without having to flush the hardware pipeline.

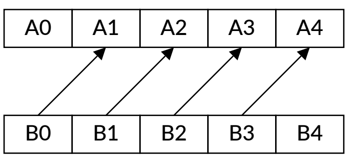
Each block operation reads an IFM block and updates or completes an OFM block. The order of block operations is:

- For depth-wise convolution, pooling, or elementwise operations, the block operations iterate over the IFM and OFM blocks at the same position in z, x, y order (depth, horizontal, then vertical). The IFM block position matches the OFM block position.
- For convolution-2D, the block operations iterate over the OFM blocks in z, x, y order and for each OFM block, the IFM block iterates over the IFM in z order. Each separate IFM block for the same OFM block counts as a separate block operation.

NPU_SET_BLOCKDEP takes a block offset k as a parameter. The block dependency guarantees that IFM block read n in the kernel does not start until all OFM block writes of the previous kernel operation, except max(k-n, 0), are complete and written to memory.

The following figure shows an example with two stripes, each of five blocks A0-A4 and B0-B4. The B operation is applied to the output of the A operation but due to the filter margin, the block B(k) read depends on the A(k+1) write as indicated by the arrows.

Figure 4-2: Example blocks



The example shows blocks issued in normal order A0, A1, A2, A3, A4, B0, B1, B2, B3, B4, but B0 is not permitted to start until A1 is complete and written to memory. Similarly, B1 is not permitted to start until A2 is complete. This sequence continues until A4 is complete, and B3 is then permitted to start.

An example of how the dependency is expressed in the command stream is:

- NPU_OP_A issues operation A
- NPU_SET_BLOCKDEP #3 expresses the B->A dependency as three block operations
- NPU_OP_B issues operation B

4.9.1 Internal shared RAM

The NPU has internal *SHared RAM* (SHRAM) that stores data.

SHRAM purpose and buffers

The purposes of the SHRAM are:

- To store data that the NPU is processing, for example *Input Feature Map* (IFM) blocks, accumulators, or *Lookup Table* (LUT) definitions which allow for data reuse.
- To store data being transferred to or from external memory by the *Direct Memory Access* (DMA) controller which absorbs memory read or write latency.

The following table lists the buffers that are placed within SHRAM.

Table 4-141: SHRAM buffers

Buffer	Buffer entries	Buffer contents
IFM	IB0, IB1	Double buffered input block buffers that must be the size in bytes of at least IFM_BLOCK_HEIGHT * IFM_BLOCK_WIDTH * ALIGN(IFM_MEMBLK_DEPTH*IFM_BYTEWIDTH,8).
IFM2	IB0, IB1	Double buffered IFM2 input block buffers that must be the size of at least IFM2_BLOCK_HEIGHT * IFM2_BLOCK_WIDTH * ALIGN(IFM2_MEMBLK_DEPTH*IFM_BYTEWIDTH,8). Where IFM2 dimensions are equal to the IFM dimension, or are set to one if the dimension is broadcast.
Accumulator	ACC0, ACC1	Double buffered accumulator and output block buffers that must be the size in bytes of at least min(OFM_HEIGHT, OFM_BLOCK_HEIGHT) * min(OFM_WIDTH, OFM_BLOCK_WIDTH) * ALIGN(OFM_BLK_DEPTH,8) * ACC_BYTEWIDTH.
Output	OB0, OB1	Scale output to OFM streaming buffer. The size of OB0 and OB1 is fixed at 1KB.
LUT	Tables	A single 2KB buffer that, if used, must be in the last 2KB of the SHRAM.

SHRAM format

The SHRAM is divided into 1KB units and each buffer is a whole number of kilobytes.

The following table lists the SHRAM layout for non-elementwise operations. For the 128 and 256 configurations of the NPU, or if a LUT is used, the value of t is set to one. For the 32 and 64 configurations of the NPU, a LUT is not used and the value of t is set to zero.

Table 4-142: Non-elementwise operations

Bank address (KB)	Bank +0KB	Bank +1KB	Notes	
0	OB0	OB1	Output data buffer	-
2	IB0	IB1	IFM data buffer	The IFM data buffer is allocated from bank address 2 to IFM_IB_END-2 in steps of 2KB.
4	IB0	IB1	IFM data buffer	
...	...	...	...	
IFM_IB_END-2	IB0	IB1	IFM data buffer	
IFM_IB_END +0	-	-	Not used for the current block	Unallocated bank addresses can be zero or greater.
AB_START+0	ACC0	ACC0	Accumulator buffer 0	The accumulator buffer is allocated from bank address AB_START+0 to SB_SIZE-2-2t in steps of 2KB.
AB_START+2	ACC1	ACC1	Accumulator buffer 1	
AB_START+4	ACC0	ACC0	Accumulator buffer 0	
...	...	...	...	
SB_SIZE-2-2t	ACC1	ACC1	Accumulator buffer 1	

The following table lists the SHRAM layout for elementwise operations AB_START=SB_SIZE.

Table 4-143: Elementwise operations

Bank address (KB)	Bank +0KB	Bank +1KB	Notes	
0	OB0	OB1	Output data buffer	-

Bank address (KB)	Bank +0KB	Bank +1KB	Notes	
2	IB0	IB1	IFM data buffer	The IFM data buffer is allocated from bank address 2 to IFM2_IB_START-2 in steps of 2KB.
4	IB0	IB1	IFM data buffer	
...	...	...	...	
IFM2_IB_START-2	IB0	IB1	IFM data buffer	
IFM2_IB_START +0	IB0	IB1	IFM2 data buffer	The IFM2 data buffer is allocated from bank address IFM2_IB_START+0 to IFM_IB_END-2 in steps of 2KB.
...	...	...	...	
IFM_IB_END-2	IB0	IB1	IFM2 data buffer	
IFM_IB_END+0	-	-	Not used for the current block	IFM_IB_END+0 to SB_SIZE-2-2t are unallocated bank addresses.
...	...	...	...	
SB_SIZE-2-2t	-	-	Not used for the current block	

Buffer restrictions

The following table lists the restrictions on IB_END, IFM2_IB_START, and AB_START. The table also lists the total RAM size, SB_SIZE for each NPU configuration. The values n, m, and k are positive integers determining the size of the IFM, IFM2, and accumulator buffers respectfully.

Table 4-144: Buffer restrictions

NPU configuration (MAC/cycle)	Elementwise non-scaler		Other operations	AB_START values in KB				SB_SIZE in KB
	IFM2_IB_START	IB_END	IB_END (KB)	16-bit accumulator	32-bit accumulator	40-bit accumulator	Elementwise	
32	2+2*n	2+2*(n +m)	2+2*n	16-2*t-4*k	16-2*t-4*k	16-2*t-4*k	16-2*t	16
64	2+2*n	2+2*(n +m)	2+2*n	16-2*t-4*k	16-2*t-4*k	16-2*t-8*k	16-2*t	16
128	2+4*n	2+4*(n +m)	2+4*n	22-4*k	22-8*k	22-12*k	22	24
256	2+8*n	2+8*(n +m)	2+8*n	46-8*k	46-16*k	46-20*k	46	48

The values must satisfy IFM2_IB_START <= IB_END <= AB_START. The input and accumulator buffer regions must be large enough to hold the configured block size. If the activation LUT is used, the value of t is set to one by NPU_SET_ACTIVATION in the command stream. If the activation LUT is not used, the value of t is set to zero.

If t is set to one, then the activation LUT is always in the final 2KB of the SHRAM.

Buffer reconfiguration

The SHRAM can be reconfigured between stripes and operations. The hardware uses IB_END and AB_START to ensure that data is not overwritten. The host processor must be aware that if

IB_END of the current operation is larger than AB_START of the previous operation, a pipeline delay occurs.

Note

Because the accumulators are not required for elementwise operations, set AB_START to SB_SIZE.

Appendix A Signal descriptions

This appendix describes the signals for the processor.

A.1 Clock and reset signals

The processor has one clock signal and two reset signals.

The following table lists the clock and reset signals.

Table A-1: Clock and reset signals

Signal	Direction	Description
CLK	Input	Clock input
nRESET	Input	The reset. This signal is an asynchronous, active-LOW signal.
nMBISTRESET	Input	The reset is used to prepare the IP for MBIST mode. This signal is an asynchronous, active-LOW signal.
PORPL	Input	The Power-On-Reset Privilege Level (PORPL). This signal sets the privilege level of the NPU after a hard reset. LOW means User level. HIGH means Privileged level.
PORSL	Input	The Power-On-Reset Security Level (PORSL). This signal sets the security level of the NPU after a hard reset. LOW means Secure. HIGH means Non-secure.

Related information

AMBA® 5 AXI master signals on page 117
DFT and MBIST signals on page 121

A.2 Interrupt signals

The processor has an interrupt signal which you must connect to an interrupt controller.

The following table lists the interrupt signals.

Table A-2: Interrupt signals

Signal	Direction	Edge or level trigger
IRQ	Output	Level triggered when HIGH.

A.3 Power management signals

The processor has several signals for power management.

The following table lists the clock Q-Channel signals.

Table A-3: Clock Q-Channel signals

Signal	Direction	Description
CLKQACTIVE	Output	This signal indicates that the NPU requires **CLK** to be active.
CLKQREQn	Input	This signal indicates that the clock controller wants to gate the clock. This signal is active-LOW.
CLKQACCEPTn	Output	This signal indicates that the NPU accepts the clock controller request. This signal is active-LOW.
CLKQDENY	Output	This signal indicates that the NPU denies the clock controller request.

The following table lists the power Q-Channel signals.

Table A-4: Power Q-Channel signals

Signal	Direction	Description
PWRQACTIVE	Output	This signal indicates that the NPU requires power.
PWRQREQn	Input	This signal indicates that the power controller wants to power down the NPU. This signal is active-LOW.
PWRQACCEPTn	Output	This signal indicates that the NPU accepts the power controller request. This signal is active-LOW.
PWRQDENY	Output	This signal indicates that the NPU denies the power controller request.

A.4 AMBA® 5 AXI master signals

The master port implements a subset of AMBA® 5 AXI which is compatible with AMBA® 4 AXI, with the addition of **ACLKEN** and **AWAKEUP** signals.

M0 wake-up signal and clock enable signals

The following table lists the wake-up and clock enable signals for master 0.

Table A-5: M0 wake-up and clock enable signals

Signal	Direction	Description
AWAKEUPM0	Output	This signal indicates if there is pending activity.
ACLKENM0	Input	This signal is the clock enable. Inputs are sampled when this signal is HIGH and outputs are held stable when this signal is LOW.

M0 write address channel signals

The following table lists the write address channel signals for master 0.

Table A-6: M0 write address channel signals

Signal	Direction	Description
AWVALIDM0	Output	This signal indicates that the write address is valid.
AWIDM0[5:0]	Output	This signal indicates the write address ID.
AWADDRM0[31:0]	Output	This signal indicates the write address.
AWLENM0[7:0]	Output	This signal indicates the write burst length.
AWSIZEM0[2:0]	Output	This signal indicates the write burst size.
AWBURSTM0[1:0]	Output	This signal indicates the write burst type.
AWCACHEM0[3:0]	Output	This signal indicates the write cache type.
AWPROTM0[2:0]	Output	This signal indicates the write protection type.
AWREADYM0	Input	This signal indicates that the write address is ready.

M0 write data channel signals

The following table lists the write data channel signals for master 0.

Table A-7: M0 write data channel signals

Signal	Direction	Description
WVALIDM0	Output	This signal indicates that the write data is valid.
WDATAM0[63:0]	Output	This signal indicates the write data.
WSTRBM0[7:0]	Output	This signal indicates the write byte lane strobes.
WLASTM0	Output	This signal is the write data last transfer indicator.
WREADYM0	Input	This signal indicates that the write data is ready.

M0 write response channel signals

The following table lists the write response channel signals for master 0.

Table A-8: M0 write response channel signals

Signal	Direction	Description
BVALIDM0	Input	This signal indicates that the write response is valid.
BIDM0[5:0]	Input	This signal indicates the write response ID.
BRESPM0[1:0]	Input	This signal indicates the write response.
BREADYM0	Output	This signal indicates that the write response is ready.

M0 read address channel signals

The following table lists the read address channels signals for master 0.

Table A-9: M0 read address channel signals

Signal	Direction	Description
ARVALIDM0	Output	This signal indicates that the read address is valid.
ARIDM0[5:0]	Output	This signal indicates the read address ID.
ARADDRM0[31:0]	Output	This signal indicates the read address.
ARLENM0[7:0]	Output	This signal indicates the read burst length.

Signal	Direction	Description
ARSIZEM0[2:0]	Output	This signal indicates the read burst size.
ARBURSTM0[1:0]	Output	This signal indicates the read burst type.
ARCACHEM0[3:0]	Output	This signal indicates the read cache type.
ARPROTM0[2:0]	Output	This signal indicates the read protection type.
ARREADYM0	Input	This signal indicates that the read address is ready.

The DMA uses different ARID values to fetch data from external memories. The following table lists the ARIDM0 values that correspond to each stream used by the DMA.

Table A-10: ARIDM0

ARID values	Channel
0-3	Command stream
4-29	IFM stream
30-55	Weight stream
56-59	Bias stream
60-63	M2M stream

M1 wake-up signal and clock enable signals

The following table lists the wake-up and clock enable signals for master 1.

Table A-11: M1 wake-up and clock enable signals

Signal	Direction	Description
AWAKEUPM1	Output	This signal indicates if there is pending activity.
ACLKENM1	Input	This signal is the clock enable. Inputs are sampled when this signal is HIGH and outputs are held stable when this signal is LOW.

M1 read address channel signals

The following table lists the read address channels signals for master 1.

Table A-12: M1 read address channel signals

Signal	Direction	Description
ARVALIDM1	Output	This signal indicates that the read address is valid.
ARIDM1[5:0]	Output	This signal indicates the read address ID.
ARADDRM1[31:0]	Output	This signal indicates the read address.
ARLENM1[7:0]	Output	This signal indicates the read burst length.
ARSIZEM1[2:0]	Output	This signal indicates the read burst size.
ARBURSTM1[1:0]	Output	This signal indicates the read burst type.
ARCACHEM1[3:0]	Output	This signal indicates the read cache type.
ARPROTM1[2:0]	Output	This signal indicates the read protection type.
ARREADYM1	Input	This signal indicates that the read address is ready.

The DMA uses different ARID values to fetch data from external memories. The following tables list the ARIDM1 values that correspond to each stream used by the DMA.

Table A-13: ARIDM1

ARID values	Channel
0-3	Command stream
4-29	IFM stream
30-55	Weight stream
56-59	Bias stream
60-63	M2M stream

M1 read data channel signals

The following table lists the read data channel signals for master 1.

Table A-14: M1 read data channel signals

Signal	Direction	Description
RVALIDM1	Input	This signal indicates that the read data is valid.
RIDM1[5:0]	Input	This signal indicates the read data ID.
RDATAM1[63:0]	Input	This signal indicates the read data.
RRESPM1[1:0]	Input	This signal indicates the read data response.
RLASTM1	Input	This signal is the read data last transfer indicator.
RREADYM1	Output	This signal indicates that the read data is ready.

Related information

Clock and reset signals on page 116
DFT and MBIST signals on page 121

A.5 AMBA® 4 APB slave signals

The slave port implements AMBA® 4 APB, with the addition of **PCLKEN** and **PWAKEUP** signals.

The following table lists the AMBA® 4 APB slave signals.

Table A-15: AMBA® 4 APB signals

Signal	Direction	Description
PWAKEUP	Input	This signal indicates if there is pending activity. This signal is input into an OR-gate that drives **CLKQACTIVE**.
PCLKEN	Input	This signal is the clock enable, Inputs are sampled when this signal is HIGH and outputs are held stable when this signal is LOW.
PSEL	Input	This signal indicates a transfer request.
PENABLE	Input	This signal indicates the second and later cycles of an AMBA® 4 APB transfer.
PPROT[2:0]	Input	This signal indicates the transfer privilege and security level. **PPROT**[2] is an indicator for data or instruction and is not used by the NPU.

Signal	Direction	Description
PWRITE	Input	This signal indicates a write transfer.
PADDR[11:0]	Input	This signal indicates the transfer address.
PWDATA[31:0]	Input	This signal indicates the write data.
PSTRB[3:0]	Input	This signal indicates the write data byte strobes.
PREADY	Output	This signal indicates that the slave is ready.
PSLVERR	Output	This signal indicates the slave error response.
PRDATA[31:0]	Output	This signal indicates the slave read data.

A.6 DFT and MBIST signals

The NPU has several DFT and MBIST signals that you must connect.

The following table lists the DFT and MBIST signals.

Table A-16: DFT and MBIST signals

Signal	Direction	Description
DFTCGEN	Input	This signal forces the clock gates on during scan shift.
DFTRSTDISABLE[1:0]	Input	This signal disables the internal synchronized reset during scan shift.
DFTRAMHOLD	Input	This signal disables the RAM chip select during scan shift.
MBISTREQ	Input	This signal is the MBIST test request.
nMBISTRESET	Input	This signal is the MBIST reset for the whole NPU. This active-LOW signal overrides the system resets when the **MBISTREQ** signal is asserted.

Related information

AMBA® 5 AXI master signals on page 117
Clock and reset signals on page 116

Appendix B General neural network concepts

This appendix describes the various concepts Arm uses to describe the NPU.

B.1 General neural network concepts

Arm uses various concepts to describe the NPU.

The following list describes how Arm uses these architectural concepts in this document:

Feature map

A feature map is a 3D array of elements. Feature maps are the data that the layers of a neural network consume and produce. The NPU works with 8-bit or 16-bit integer elements. For example, the initial input to an image recognition network might be a three channel feature map. In this example, the channels correspond to the red, green, and blue color planes of an image. Each element contains an RGB value. Therefore, the feature maps for the first layer describe the image.

Note

Integer elements can also be described as activation values to distinguish them from weight values.

Layer

A neural network (NN) is composed of several layers; the input to one layer is the output from a prior layer. The NPU is designed to process the layer of a network without requiring interaction from the host application processor. There are various types of layers, with CNNs named due to their large usage of convolutional layers.

NHWC and NCHW

NHWC and NCHW are standard memory formats of feature maps. Each letter in the NHWC and NCHW memory formats represents an axis of the feature map. The order of the letters represents the sequence of data when stored in memory. The letters of the memory formats represent:

N

Number of batches.

H

Height.

W

Width.

C

Channels.

NHWC is the standard format for the TensorFlow Lite stack used by the NPU.

Weights, kernels, and filters

Weights, kernels, and filters are all related concepts. A filter is an operation on a signal. A kernel is a linear function that is used within a convolution as a filter. A kernel can be represented as a matrix. A weight is an individual element of this matrix.

Appendix C Boot flow information

This appendix describes the various boot flows for the NPU.

C.1 Boot flow information

This appendix describes the software interactions needed to boot up the NPU, perform a soft reset of the NPU, and power down the NPU.

Boot flow

At system start-up, the NPU is normally powered down. You must do the following before the NPU can be used:

1. If Power Q-Channels are not supported, you must:
   a. Assert **nRESET**.
   b. Enable NPU power.
   c. Deassert **nRESET**.

Note

The software interface to control the deassertion of the **nRESET** signal and NPU power is platform-dependent.

---

2. Perform a write to the CMD register.

Note

To ensure the NPU demands power, set the field `power_q_enable` to 0x0.

Setting the field `clock_q_enable` to 0x0 ensures the NPU demands that the clock is running. Setting the field `clock_q_enable` to 0x1 enables automatic high-level clock gating. Arm recommends setting field `clock_q_enable` to 0x1.

Set all other fields in the CMD register to 0x0. For more information about the CMD register, see 4.2.3 Register CMD on page 36.

---

3. To ensure the NPU is now in a known state, Arm recommends doing a soft reset. A soft reset negates the risk that power was on before step 1 and **nRESET** was not asserted. For more information about the soft reset, see Soft reset flow on page 124.

Soft reset flow

A soft reset is used for setting the NPU in a known state and to update the NPU security status. Do the following to perform a soft reset of the NPU:

1. To trigger a soft reset, write to the RESET register. For more information about setting the fields `pending_CSL` and `pending_CPL`, see 4.2.4 Register RESET on page 37.

2. Read the STATUS register until the field `reset_status` no longer yields the value 0x1.

Note: The value 0x1 indicates a soft reset phase is in progress. During this phase, no other APB accesses are allowed.

3. Write the CMD register. If the Power Q-Channel is used, set the field `power_q_enable` to 0x0 to keep power enabled.

Note: Setting the field `clock_q_enable` to 0x0 ensures the NPU demands that the clock is running. Setting the field `clock_q_enable` to 0x1 enables automatic high-level clock gating. Arm recommends setting field `clock_q_enable` to 0x1.

Powering down flow

Do the following to power down the NPU:

1. Acknowledge any pending interrupts by writing register CMD.

Note: All interrupts must be cleared for power down to occur.

For more information about the CMD register, see 4.2.3 Register CMD on page 36.

2. Write to the CMD register.

Note: The field `power_q_enable` must be set to 0x1 to permit power down.

3. After the preceding sequence of register writes, the powering down starts by the NPU handshaking with the power controller.

Related information

Security and boot flow on page 21

Appendix D Revisions

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

Related information

Product revisions on page 18

D.1 Revisions

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

Table D-1: First EAC release for r2p0

Change	Location	Affects
First release	-	-