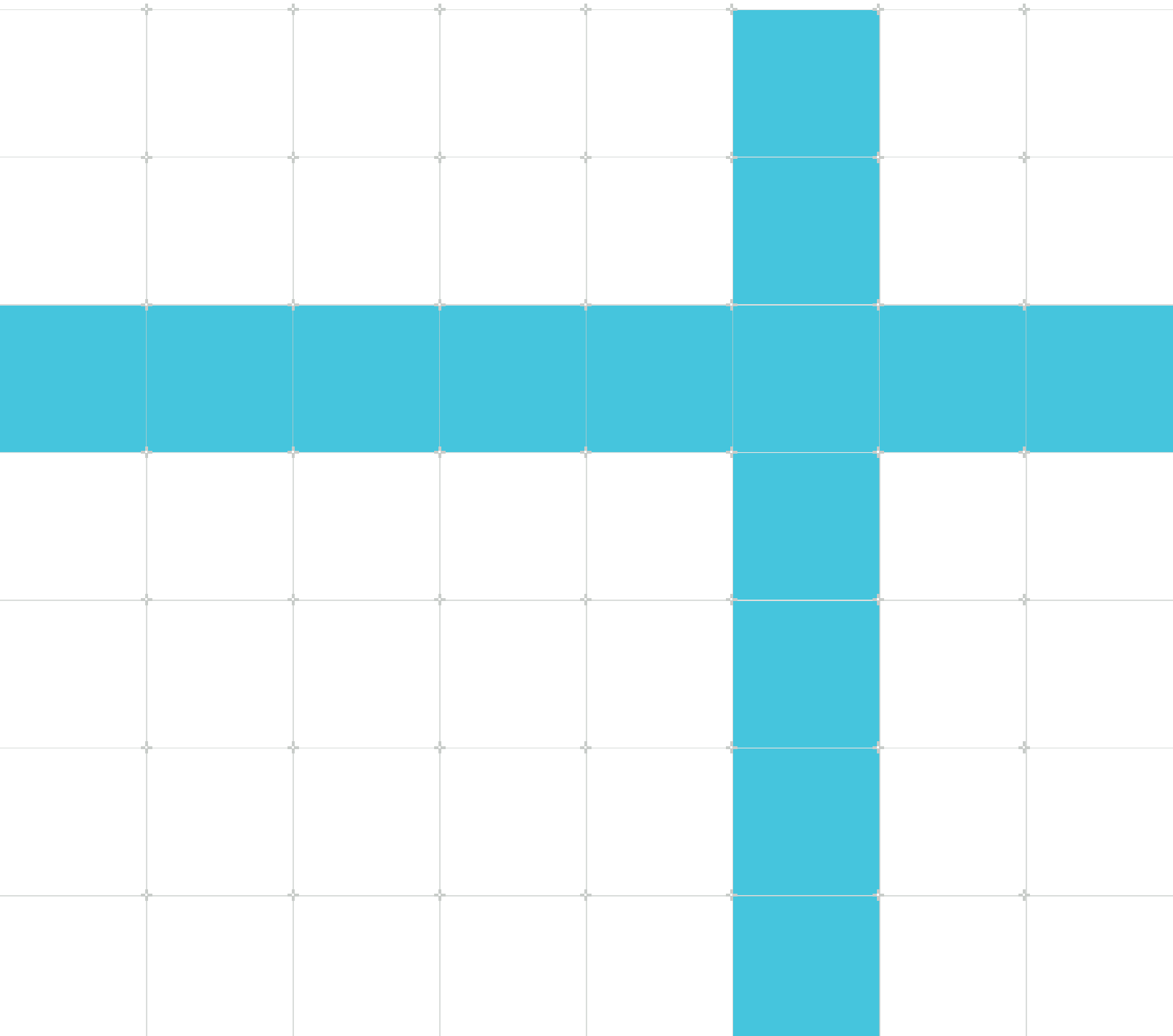
Arm® Ethos™-U85 NPU

Revision r0p0

Technical Reference Manual

Non-Confidential

Issue 05
102685_0000_05_en



Arm® Ethos™-U85 NPU Technical Reference Manual

This document (102685_0000_05_en) was issued on 2025-01-31. There might be a later issue at https://developer.arm.com/documentation/102685

The product revision is r0p0.

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

Start Reading

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

Intended audience

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

Inclusive language commitment

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

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Contents

1. Description of the Arm® Ethos™-U85 NPU

The Ethos™-U85 NPU is designed to accelerate Edge AI inference in both constrained Cortex®-M and Cortex®-A based systems. The NPU targets the *Tensor Operator Set Architecture* (TOSA) and TFLite integer quantized operations.

Note

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

The NPU accelerates, without host microcontroller or application processor fallback, any neural network that software tooling can lower to the TOSA integer profiles supported by the Arm®Ethos™-U software. For more information on the software tooling the NPU requires, see 1.1 Overview of supported software on page 12. For example, the neural networks the NPU can accelerate includes the following 8-bit and 16-bit integer quantized networks:

- Transformer networks
- *Convolutional Neural Networks* (CNN)
- *Recurrent Neural Networks* (RNN)

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

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

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

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

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

The external interfaces that the NPU implements are:

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

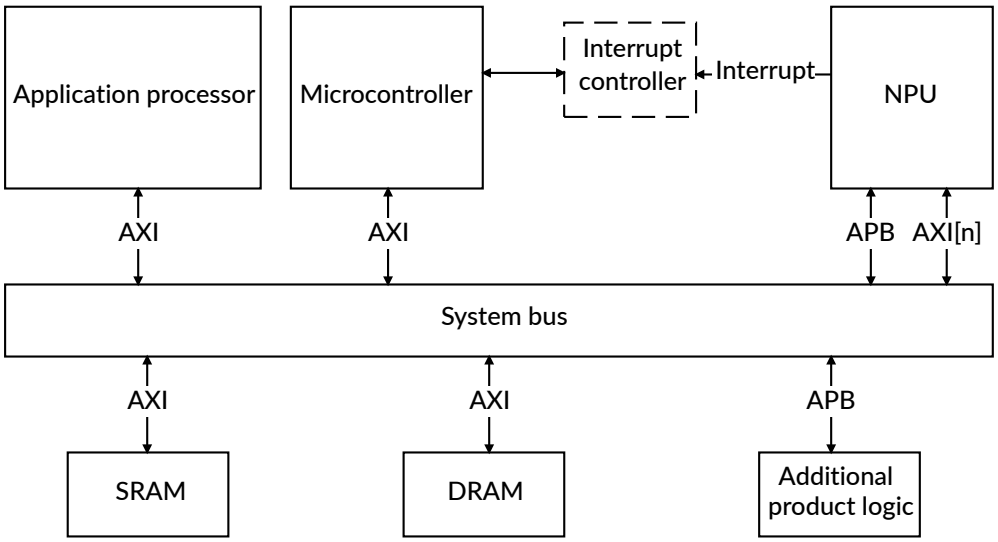
Note

The number of Arm® AMBA® 5 AXI manager interfaces depend on the NPU configuration. The manager interfaces are also AMBA® 4 AXI compatible.

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

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

Figure 1-1: Typical system configuration block diagram




Note

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

1.1 Overview of supported software

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

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

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

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

This manual does not cover the capabilities of the software. For more information about the related software, see the following project: https://gitlab.arm.com/artificial-intelligence/ethos-u/ethos-u.

1.2 Interfaces

The NPU has several external interfaces.

The external interfaces are:

- Arm® AMBA® 5 APB Completer with clock enable and wake-up signaling
- Two kinds of Arm® AMBA® 5 AXI AXI managers:
  - The AXI_SRAM for targeting system SRAM
  - The AXI_EXT for targeting DRAM or flash memory
- An interrupt
- Two Q-channel ports:
  - A Q-Channel for clock management

    - A Q-Channel for power management
- System configuration signals
- Clock
- Reset
- Warm reset interface
- MBIST interface
- Debug interface
- DFT interface

1.3 Documentation

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

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

Technical overview

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

Technical reference manual

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

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

Configuration and integration manual

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

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

The CIM is a confidential book only available to licensees.

1.4 Design process

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

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

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

Programming
The system programmer uses the following to develop the SoC:

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

1.5 Product revisions

Successive product revisions have differences in functionality.

r0p0
First release.

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

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

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

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

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

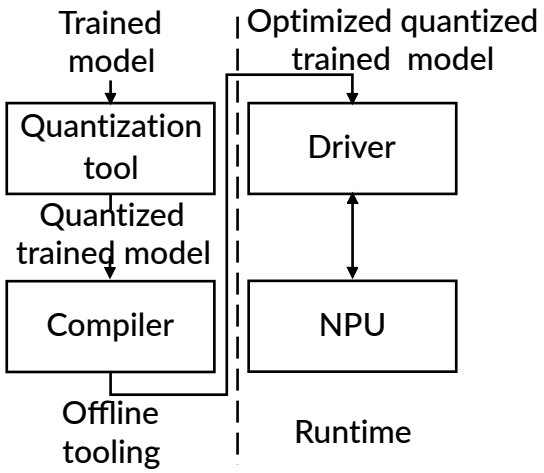
- *Central Control* (CC)
- DMA controller
- *Multiply-Accumulate* (MAC) unit
- *Weight Decoder* (WD)
- *Activation Output* (AO) unit

2.1 Control and data flow

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

The following figure shows the software stack for the NPU.

Figure 2-1: The offline software stack



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


Note

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

The following steps describe the offline tooling flow:

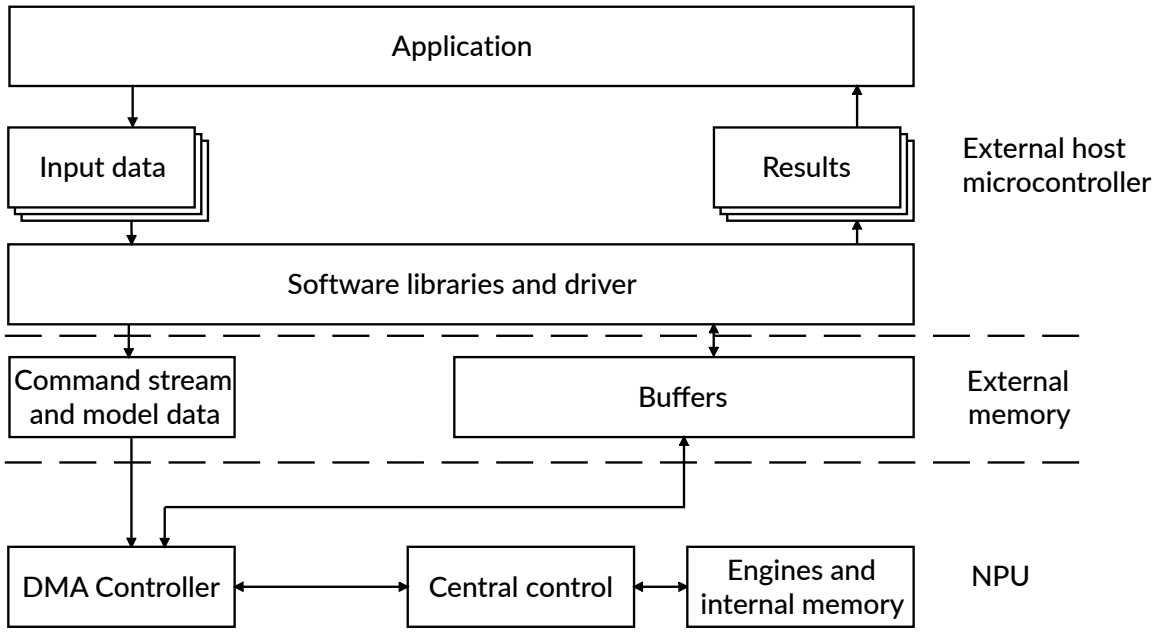
1. Pass your trained model through the quantization tool.
2. Pass the quantized model to the compiler. This tool optimizes the model for this NPU and outputs an optimized model that contains a command stream for the NPU.

The following steps describe the runtime control and data flow:

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

The following figure shows the control and data flow.

Figure 2-2: Control and data flow



2.1.1 Supported memory formats for feature maps

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

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

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

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

NHWC format

The NHWC format has the following properties:

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

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

NHCWB16 format

The NHCWB16 format has the following properties:

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

2.2 Security and boot flow

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

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

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

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

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

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

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

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

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

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

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


Note

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

2.3 Functional blocks

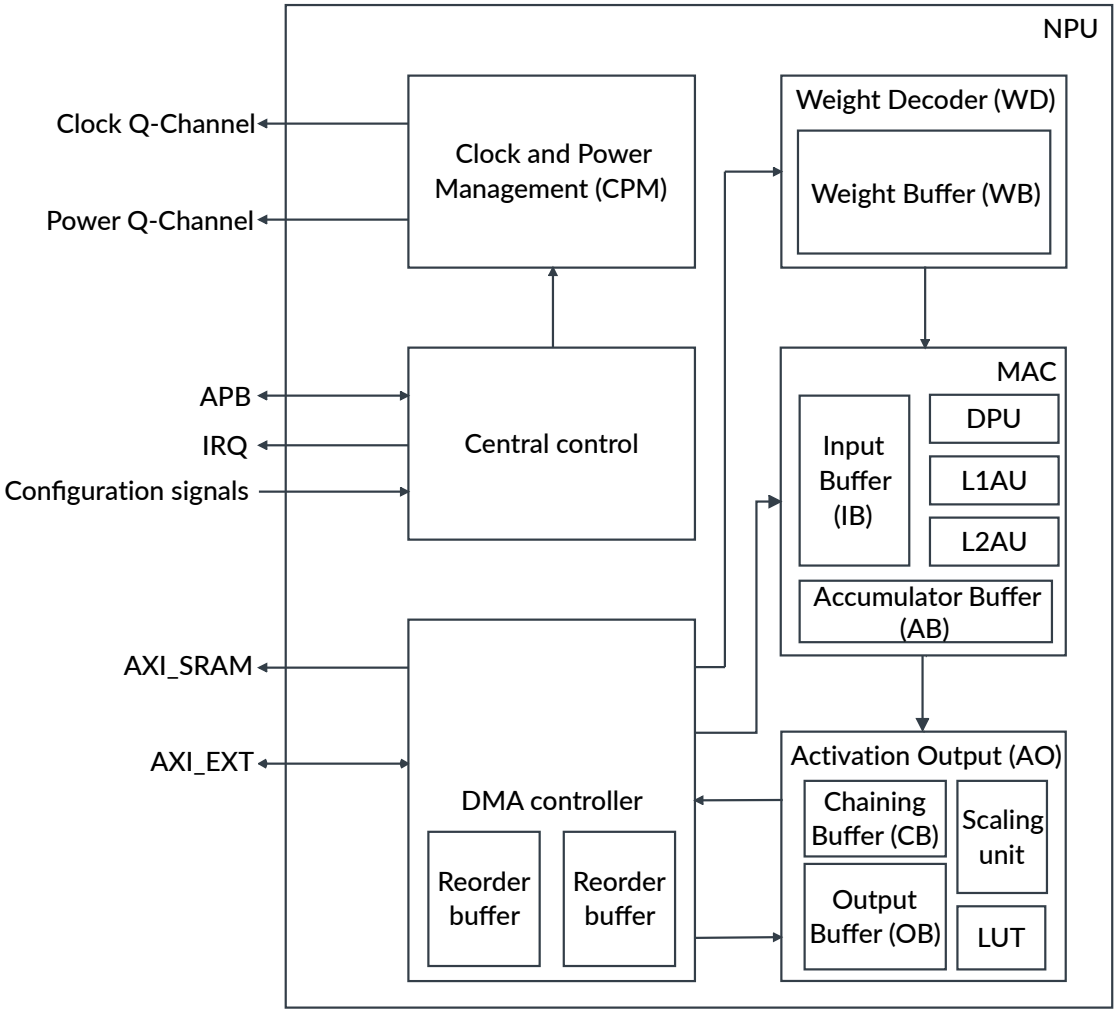
The NPU consists of the *Central Control* (CC), a DMA controller, a MAC unit, an output unit, and the *Weight Decoder* (WD).

The following are descriptions of the units of the NPU:

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

The following figure shows the main components of the NPU.

Figure 2-3: Functional blocks diagram



2.3.1 External interfaces

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

Two kinds of AMBA® 5 AXI manager interfaces

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

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

AMBA® 5 APB completer interface with wake-up signaling

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

Interrupt interface

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

Two Q-Channel interfaces

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

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

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

Clock and reset

The NPU has one clock and one reset signal.

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

Configuration signals

The following configuration signals determine the security level after boot:

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


Note

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

Warm reset signals

The following signals communicate a halt of the NPU:

- WRSTQREQ
- WRSTQACK

Debug signals

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

- DBGEN
- SPIDEN
- DBGHALTREQ
- DBGRESTARTREQ
- DBGHALTACK

Related information

Signal descriptions on page 344

2.3.2 Central control

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

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

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

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

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

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

Other commands can:

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

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

2.3.3 Configuration pins and AXI port striping

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

Striping

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

Striping control

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

CFGSRAMHASH0
: This pin enables the NPU to stripe across AXI_SRAM ports 0 and 1 when needed.

CFGSRAMHASH1
: This pin enables the NPU to stripe across AXI_SRAM ports 2 and 3 when present.

CFGEXTHASH0
: This pin enables the NPU to stripe across AXI_EXT ports when present.

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

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

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

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

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

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


Note

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

Configuration hash pins and corresponding striping

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

Note

For more information on striping, including a reference system that shows how an SoC uses striping, see the *Arm® Corstone™ Reference Systems Architecture Specification Ma2*.

Table 2-1: Configuration hash pins and corresponding striping

Ports	Striping	Formula	CFGSRAMHASH0 and CFGEXTHASH0	CFGSRAMHASH1
1	N/A	N/A	0x0000000000	0x0000000000
2	64B trivial	addr[6]	0x0000000040	0x0000000000
2	128B trivial	addr[7]	0x0000000080	0x0000000000
2	256B trivial	addr[8]	0x0000000100	0x0000000000
2	512B trivial	addr[9]	0x0000000200	0x0000000000
2	256B Fibonacci	addr[8] ^ addr[9] ^ addr[10] ^ addr[11] ^ addr[13] ^ addr[16] ^ addr[21] ^ addr[29]	0x0020212f00	0x0000000000
4	64B trivial	addr[7:6]	0x0000000040	0x0000000080
4	128B trivial	addr[8:7]	0x0000000080	0x0000000100
4	256B trivial	addr[9:8]	0x0000000100	0x0000000200
4	512B trivial	addr[10:9]	0x0000000200	0x0000000400
4	256B Fibonacci	sel[0] = addr[8] ^ addr[10] ^ addr[13] ^ addr[21] sel[1] = addr[9] ^ addr[11] ^ addr[16] ^ addr[29]	0x0000202500	0x0020010a00

Max outstanding transaction control for each AXI port type

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

The following are the configuration pins:

CFGSRAMCAP[31:0]

Configuration of capabilities for SRAM ports.

CFGEXTCAP[31:0]

Configuration of capabilities for EXT ports.

For more information on these configuration pins, see 3.3.1.20 CFG_SRAM_CAP register on page 70 and 3.3.1.21 CFG_EXT_CAP register on page 71.

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

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

2.3.3.1 AXI ports per NPU configuration

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

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

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

2.3.4 DMA controller

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

The channels that the DMA controller uses are:

Command channel

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

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

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

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

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

The NPU uses the IFMS channel to read input feature maps and move the data to the *Activation Output* (AO) unit. This stream is also used to move data to the WD when the NPU executes specific operators such as MATMUL. The channel supports the NHWC and NHCWB16 off-chip memory formats natively. Because the receiver must store activations from different x,y coordinates in different words, the DMA controller unpacks data which is stored in NHWC format. The DMA controller mainly uses the IFMS channel to read elementwise operations inputs.

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

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

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

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

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

Weight channel and fast weight channel

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

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

The weight stream must be quantized to eight bits or less by an offline tool. When passed through the offline compiler, weights are compressed losslessly and reordered into an NPU-specific weight stream. This process is effective, if the quantizer uses less than eight bits or if it uses clustering and pruning techniques. The quantizer can also employ all three methods. Using lossless compression on high sparsity weights, containing greater than 75% zeros can lead to compression below 3 bits per weight in the final weight stream.

mem2mem channel

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

The traversal unit triggers mem2mem operations on specific API commands.

Bias and scale channel

This channel streams data to the AO unit. The data that it transmits is the scale and bias necessary for the block that the NPU is processing. Layers that pass through the AO unit are

written to the on-chip SRAM. As the layers pass through the AO unit, activation functions can be fused.

2.3.5 Clock and power module

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

Clock and power module controlling reset

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

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

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

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

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

QLPI for clock

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

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

QLPI for power

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

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

Clock and power module clock gates

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

2.4 Weight decoder

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

The WD supports 2 weight stream formats for constant weights, the *Standard Weight Decoder* (SWD), and the *Fast Weight Decoder* (FWD).

The SWD supports lossless compression to reduce the size of the weight stream. Arm expects pruned or clustered weight streams to compress better than ones that are not pruned or clustered. The SWD supports all convolution and depth-wise convolution operations. A single SWD supports an external read bandwidth of up of 64 bits/cycle. Higher MAC configurations of the NPU include multiple weight decoders, given by the field `num_wd`, to increase the weight bandwidth. We recommend you use the SWD when weight size is important.

The FWD supports a higher read bandwidth of up to 256 bits/cycle. The FWD decoder supports convolution but not depth-wise convolution operations. The FWD decoder can gain in performance over the SWD when there is high memory bandwidth available and little weight reuse. For example, when a 1x1 convolution (a fully connected operator) accesses striped SRAM. The FWD provides a raw (uncompressed) 8-bit weight mode and table-based weight modes with 4 or 2 bits per weight. The table modes index tables of 16 or 4 8-bit weights, respectively. The FWD is typically not used for lower bandwidth memory, non-fully connected operations, or NPU configurations that contain 4 SWD.

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

2.5 MAC unit

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

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

The MAC unit comprises:

- An *Input Feature Map* (IFM) unit
- *Dot Product Units* (DPU)
- *Activation Input* (AI) unit

Related information

NPU_OP_CONV instruction on page 217
NPU_OP_DEPTHWISE instruction on page 219
NPU_OP_POOL instruction on page 229

2.5.1 Dot product units

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

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

2.5.2 Activation input unit

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

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

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

2.6 Activation output unit

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

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

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

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

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

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

The AO unit supports the following activation functions:

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

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

Related information

NPU_OP_ELEMENTWISE instruction on page 223
NPU_OP_RESIZE instruction on page 233

2.6.1 Scaling unit

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

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

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

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

2.6.2 Lookup table

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

The AO unit supports the following modes for LUTs:

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

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

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

2.6.3 Chaining buffer

The chaining buffer is an input buffer for elementwise operations.

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

2.6.4 Output Buffer

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

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

3. Programmers model

This section describes the registers, register map, and command stream instructions of the NPU.

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**.
- 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.
- The following scenarios return a bus error:
  - Nonprivileged access when the NPU is in privileged state
  - Non-secure access when the NPU is in secure state
  - Register writes during reset sequence
  - Subword write
  - Unaligned access

3.1 Register Interface

This section describes the NPU 32-bit APB register interface.

3.1.1 APB interface

The NPU registers are accessed using a 32-bit APB interface.

Power must be enabled before the interface is accessed. For more information on power, see Powering up. The Requester writing to the APB interface indicates one of four levels of privilege according to PPROT[0] and PPROT[1]:

- PPROT[0] is 0 for normal access and 1 for privileged access

- PPROT[1] is 0 for Secure access and 1 for Non-secure access

The following accesses always generate a bus error PSLVERR=1:

- Non-Privileged access when the NPU is in Privileged state (PPROT[0]=0 && PROT.active_CPL==1)
- Non-Secure access when the NPU is in Secure state (PPROT[1]==1 && PROT.active_CSL==0)
- A write smaller than a 32-bit word
- A read or write that is not 32-bit address aligned

The NPU is defined as being in a reset sequence when STATUS.reset_status==1. A reset sequence is triggered by power up or by software reset writing to RESET. If a read access occurs during a reset sequence then one of the following occurs:

- The NPU delays processing the read until the end of the reset sequence. The read is then processed as normal.
- The NPU processes a read to the STATUS register within the reset sequence. The read value has STATUS.reset_status==1.
- The NPU processes the read within the reset sequence to a register other than STATUS. Then the read returns the value zero (regardless of the actual register contents).

If a write access occurs during a reset sequence then one of the following occurs:

- The NPU delays processing the write until the end of the reset sequence. The write is then processed as normal.
- The NPU processes a write to the CMD register within the reset sequence. Then only bits clock_q_enable and power_q_enable are written. All other bits of the command register are ignored.
- The NPU processes a write to the RESET register within the reset sequence. The NPU updates the privilege level as for normal RESET writes. The NPU does not need to restart the reset sequence.
- The NPU raises a bus error PSLVERR=1 for writes within a reset sequence to registers other than CMD and RESET.

If any access causes a bus error, PSLVERR=1, then:

- For a read, the data returned is zero (PRDATA=0)
- For a write, the access has no effect

Accesses that are defined in the register description as UNPREDICTABLE can raise a bus error. These accesses do not always raise a bus error.

3.1.2 Powering up

There a few considerations when powering up the NPU.

If power Q-Channels are not supported, you must:

1. Assert nRESET
2. Enable NPU power
3. Deassert nRESET


Note

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

To ensure the NPU demands power, write to the CMD register. The field power_q_enable must be set to 0x0. Arm recommends setting clock_q_enable to 0x1 to enable high-level clock gating. Set all other fields in CMD to zero.

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 the first step and nRESET was not asserted. For more information about the soft reset, see Soft reset.

3.1.3 Soft reset

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.
2. Read the register field STATUS.reset_status until the value is 0x0. During this phase, no other APB accesses should be used.

3.1.4 Powering down

You must follow specific steps to power down the NPU.

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.

2. Write to the CMD register. The field power_q_enable must be set to 0x1 to permit power down.

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

3.2 host_map address map

This section shows the address map of the NPU control registers, as seen from the host micro-controller.

The base address that the NPU is mapped to is **IMPLEMENTATION SPECIFIC**, and differs from device to device.

The offsets in this address map are relative to the base address of the map.

This address map contains the following mappings:

Table 3-1: host_map address mapping

Offset	Size	Name	Content	Description
0x0000 - 0x007F	128B	BASE	Register page BASE	NPU base registers
0x0080 - 0x00FF	128B	BASE_POINTERS	Register page BASE_POINTERS	NPU base pointers
0x0100 - 0x017F	128B	reserved	Reserved	-
0x0180 - 0x02FF	384B	reserved	Reserved	-
0x0300 - 0x03FF	256B	reserved	Reserved	-
0x0400 - 0x093F	1344B	reserved	Reserved	-
0x0940 - 0x0BFF	704B	reserved	Reserved	-
0x0C00 - 0x0EFF	768B	reserved	Reserved	-
0x0F00 - 0x0FFF	256B	ID	Register page ID	NPU ID registers
0x1000 - 0x117F	384B	reserved	Reserved	-
0x1180 - 0x11FF	128B	PMU	Register page PMU	NPU PMU registers
0x1200 - 0x12FF	256B	reserved	Reserved	-
0x1300 - 0x13FF	256B	PMU_COUNTERS	Register page PMU_COUNTERS	NPU PMU counters
0x1400 - 0x1FFF	3KiB	reserved	Reserved	-

3.3 Register sets for NPU control

NPU control register sets.

Note

Attributes are marked with an asterisk * if child elements in the register page can override this attribute.

The following pages and subpages contain the NPU control control registers.

Table 3-2: NPU control register set summary

Name	Size	Description	Access
BASE	128-byte	The NPU control registers bank	*
BASE_POINTERS	128-byte	NPU register bank	*
ID	256-byte	NPU register bank	*
PMU	128-byte	Performance monitoring	*
PMU_COUNTERS	256-byte	Performance monitoring counters	*

3.3.1 BASE register summary

The NPU control registers bank.

Table 3-3: BASE register summary

Offset	Name	Type	Reset	Width	Description
0x0000	ID	ro	0x20007001	32-bit	ID register
0x0004	STATUS	ro	See individual bit resets	32-bit	Register describes the current operating status of the NPU
0x0008	CMD	rw	0x0000000C	32-bit	The command register. This register reads as last written command
0x000C	RESET	rw	0x00000000	32-bit	Request reset and new security mode
0x0010 - 0x0014	QBASE	rw	0x0000000000000000	64-bit	The base address of the command stream in bytes
0x0018	QREAD	ro	0x00000000	32-bit	The read offset in the command stream in bytes. Multiple of four in the range 0-16MB
0x001C	QCONFIG	rw	0x00000000	32-bit	The AXI configuration for the command stream in the range 0-3. Same encoding as for REGIONCFG
0x0020	QSIZE	rw	0x00000000	32-bit	The size of the command stream in bytes. Multiple of four in the range 0-16MB
0x0024	PROT	ro	0x00000000	32-bit	The protection level configured for the NPU when acting as an AXI Requester
0x0028	CONFIG	ro	See individual bit resets	32-bit	RTL configuration
0x002C	Reserved	-	-	32-bit	-
0x0030	COND_STATUS	ro	0x00000000	32-bit	Condition status of the NPU
0x0034	Reserved	-	-	32-bit	-
0x0038	POWER_CTRL	rw	0x00000000	32-bit	Power control register
0x003C	REGIONCFG	rw	0x00000000	32-bit	Specify which MEM_ATTR register applies to each region
0x0040	MEM_ATTR0	rw	0x00000000	32-bit	Memory attributes 0
0x0044	MEM_ATTR1	rw	0x00000000	32-bit	Memory attributes 1
0x0048	MEM_ATTR2	rw	0x00000000	32-bit	Memory attributes 2
0x004C	MEM_ATTR3	rw	0x00000000	32-bit	Memory attributes 3
0x0050	AXI_SRAM	rw	0x00000000	32-bit	The AXI configuration for SRAM ports
0x0054	AXI_EXT	rw	0x00000000	32-bit	The AXI configuration for EXT ports
0x0058 - 0x005C	Reserved	-	-	64-bit	-

Offset	Name	Type	Reset	Width	Description
0x0060	CFG_SRAM_CAP	ro	See individual bit resets	32-bit	The value of the CFGSRAMCAP pins, SRAM AXI ports cap
0x0064	CFG_EXT_CAP	ro	See individual bit resets	32-bit	The value of the CFGEXTCAP pins, EXT AXI ports cap
0x0068 - 0x006C	CFG_SRAM_HASH0	ro	See individual bit resets	64-bit	The value of the CFGSRAMHASH0 pins, SRAM AXI port select bit 0 hash
0x0070 - 0x0074	CFG_SRAM_HASH1	ro	See individual bit resets	64-bit	The value of the CFGSRAMHASH1 pins, SRAM AXI port select bit 1 hash
0x0078 - 0x007C	CFG_EXT_HASH0	ro	See individual bit resets	64-bit	The value of the CFGEXTHASH0 pins, EXT AXI port select bit 0 hash

3.3.1.1 ID register

ID register.

This register can be read in stopped or running state.

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

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0000

Type

ro

Reset value

0x20007001

Bit descriptions

Figure 3-1: ID bit assignments

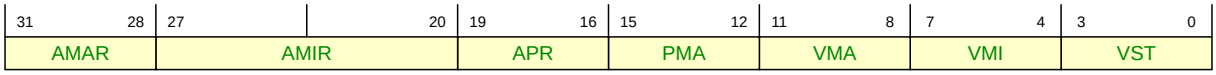


Table 3-4: BASE.ID bit descriptions

Bits	Name	Description	Reset
[3:0]	version_status (VST)	This value is the version of the product. This value is an unsigned integer. Its default value is 1 (**IMPLEMENTATION DEFINED**). **Access** ro	1 (**IMPLEMENTATION DEFINED**)
[7:4]	version_minor (VMI)	This value is the n for the P part of an RnPn release number. This value is an unsigned integer. Its default value is 0 (**IMPLEMENTATION DEFINED**). **Access** ro	0 (**IMPLEMENTATION DEFINED**)
[11:8]	version_major (VMA)	This value is the n for the R part of an RnPn release number. This value is an unsigned integer. Its default value is 0 (**IMPLEMENTATION DEFINED**). **Access** ro	0 (**IMPLEMENTATION DEFINED**)
[15:12]	product_major (PMA)	Product major ID number (unique per base product). This value is an unsigned integer. Its default value is 7 (**IMPLEMENTATION DEFINED**). **Access** ro	7 (**IMPLEMENTATION DEFINED**)
[19:16]	arch_patch_rev (APR)	This value is the patch number of the architecture version a.b. This value is an unsigned integer. Its default value is 0 (**IMPLEMENTATION DEFINED**). **Access** ro	0 (**IMPLEMENTATION DEFINED**)
[27:20]	arch_minor_rev (AMIR)	This value is the minor architecture version number, b in the architecture version a.b. This value is an unsigned integer. Its default value is 0 (**IMPLEMENTATION DEFINED**). **Access** ro	0 (**IMPLEMENTATION DEFINED**)
[31:28]	arch_major_rev (AMAR)	This value is the major architecture version number, a in the architecture version a.b. This value is an unsigned integer. Its default value is 2 (**IMPLEMENTATION DEFINED**). **Access** ro	2 (**IMPLEMENTATION DEFINED**)

Accessibility

Access to this register is restricted to mode ro.

3.3.1.2 STATUS register

Register describes the current operating status of the NPU.

This register can be read in stopped or running state.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0004

Type

ro

Reset value

See individual bit resets.

Bit descriptions

Figure 3-2: STATUS bit assignments

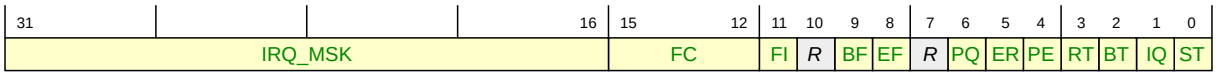


Table 3-5: BASE.STATUS bit descriptions

Bits	Name	Description	Reset
[0]	state (ST)	This value is an enumeration of type state_t. Its default value is stopped. This field can contain the following values: **0** stopped - The NPU is in stopped state **1** running - The NPU is in running state **Access** ro	stopped

Bits	Name	Description	Reset
[1]	irq_raised (IQ)	The raw (unmasked) IRQ status raised to the host. The IRQ is cleared using CMD.clear_irq. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[2]	bus_status (BT)	If a bus abort is detected, the NPU stops, sets this status bit and raises an IRQ to the host. The NPU does not process further commands or AXI transactions after the bus abort is detected. This fault bit can only be cleared by reset. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[3]	reset_status (RT)	When a reset is in progress only this STATUS register can be read. All other registers read as 0 and writes are ignored during a reset. This value is an unsigned integer. Its default value is 0x1. **Access** ro	0x1
[4]	cmd_parse_error (PE)	If a command stream parsing error is detected, the NPU stops, sets this status bit and raises an IRQ to the host. This error bit can only be cleared by reset. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[5]	cmd_end_reached (ER)	The command stream end is reached when QREAD = QSIZE. When commands are complete the NPU stops, sets this status bit and raises and IRQ to the host. Clear this bit by writing to QBASE or QSIZE when the NPU is in stopped state. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[6]	pmu_irq_raised (PQ)	This status bit is cleared using CMD.clear_irq. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[7]	Reserved	-	-

Bits	Name	Description	Reset
[8]	ecc_fault (EF)	An ECC fault is a detected but uncorrected error on internal RAMs. Corrected errors do not cause a fault. This fault is only available if ECC logic has been added to the internal RAMs. An ECC fault will cause the NPU to stop, set this status bit and then raise an IRQ to the host. This fault bit can only be cleared by reset. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[9]	branch_fault (BF)	A branch fault is a branch outside of the command stream offset range of 0 to QSIZE. An branch fault causes the NPU to stop, set this status bit and then raise an IRQ to the host. This fault bit can only be cleared by reset. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[10]	Reserved	-	-
[11]	faulting_interface (FI)	This value is an enumeration of type dma_fault_src_t. This field can contain the following values: **0** sram - SRAM **1** ext - External **Access** ro	-

Bits	Name	Description	Reset
[15:12]	faulting_channel (FC)	This value is an enumeration of type pmu_axi_channel_t. This field can contain the following values: **0x0** rd_cmd - Command stream read channel **0x1** rd_ifm - IFM read channel **0x2** rd_weights - Weights read channel **0x3** rd_scale_bias - Bias and scale read channel **0x4** rd_mem2mem - Memory to memory read channel **0x5** rd_ifm_stream - IFM to weights or elementwise **0x6** rd_mem2mem_idx - Memory to memory index channel **0x7** Reserved **0x8** wr_ofm - The OFM write channel **0x9** wr_mem2mem - The mem2mem write channel **0xA..0xF** Reserved **Access** ro	-
[31:16]	irq_history_mask (IRQ_MSK)	These bits are used for debug purposes. Each IRQ or event operation provides a 16-bit mask. This history mask contains the logical-or of these 16-bit masks. The bits can be cleared using CMD.clear_irq_history. This value is an unsigned integer. Its default value is 0x0000. **Access** ro	0x0000

Accessibility

Access to this register is restricted to mode ro.

3.3.1.3 CMD register

The command register. This register reads as last written command.

This register can be read or written in stopped or running state.

A read of this register returns the last written command.

If this register is written within a reset sequence then only the clock_q_enable and power_q_enable take effect. Other bits are ignored.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0008

Type

rw

Reset value

0x0000000C

Bit descriptions

Figure 3-3: CMD bit assignments

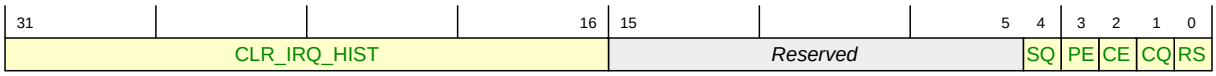


Table 3-6: BASE.CMD bit descriptions

Bits	Name	Description	Reset
[0]	transition_to_running_state (RS)	Writing 1 to this bit in stopped state causes the NPU to transition to running state and start executing the command stream. The address of the command stream is as follows: • If the NPU has previously transitioned from running to stopped state by execution of an NPU_OP_STOP operation and QBASE or QSIZE have not been written since that transition, then the NPU continues from the point of previous execution. That is the NPU executes next the command at address QBASE+QREAD. • Otherwise the NPU starts execution at the address QBASE Writing 1 to this bit in running state has no effect This value is an unsigned integer. Its default value is 0x0.	0x0
[1]	clear_irq (CQ)	Write 1 to clear the IRQ status in the STATUS register. Writing 0 has no effect. This value is an unsigned integer. Its default value is 0x0.	0x0

Bits	Name	Description	Reset
[2]	clock_q_enable (CE)	Write 1 to this bit to enable clock off using clock q-interface and enable the requester clock gate. This value is an unsigned integer. Its default value is 0x1.	0x1
[3]	power_q_enable (PE)	Write 1 to this bit to enable power off using power q-interface. This value is an unsigned integer. Its default value is 0x1.	0x1
[4]	stop_request (SQ)	Writing 1 to this bit in stopped state has no effect. Writing 1 to this bit in running state transitions the NPU to stopped state as follows: 1. If g_chain_length > 0 then the NPU continues to issue commands until g_chain_length = 0, which occurs after a command that writes to a non-chained OFM. 2. The NPU stops issuing new commands. 3. QREAD is set to the offset of the command that would have been issued next. 4. The NPU waits for all issued commands to complete. 5. The NPU enters stopped state. 6. The NPU issues an IRQ to the host. This value is an unsigned integer. Its default value is 0x0.	0x0
[15:5]	Reserved	-	-
[31:16]	clear_irq_history (CLR_IRQ_HIST)	When bit k is set then corresponding bit k of the status register is cleared. The corresponding bit is the IRQ history bit. This value is an unsigned integer. Its default value is 0x0000.	0x0000

Accessibility

Access to this register is restricted to mode rw.

3.3.1.4 RESET register

Request reset and new security mode.

This register can be read or written in stopped or running state.

A write to this register performs the following actions:

- Current operations are halted (in an AXI safe way).
- All register state is cleared apart from the pending privilege level set by this write.
- The NPU reset sequence is started, including internal RAM clear.
- The privilege level is set to the privilege level that is the lower of the host privilege level making this write (PPROT) and the pending privilege level set by this write. See PROT for further information.
- Power control is masked until the next write to the CMD register. Thus power_q_enable has no effect (power is kept on) until the next write to CMD.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x000C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-4: RESET bit assignments

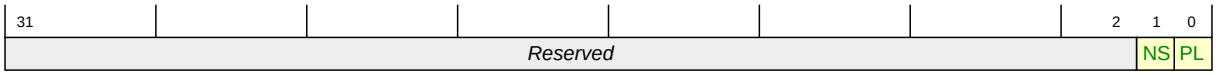


Table 3-7: BASE.RESET bit descriptions

Bits	Name	Description	Reset
[0]	pending_CPL (PL)	The current privilege level. This value is an enumeration of type privilege_level_t. Its default value is user. This field can contain the following values: **0** user - The NPU is configured for User-level access **1** privileged - The NPU is configured for Privileged-level access	user
[1]	pending_CSL (NS)	The current security level. This value is an enumeration of type security_level_t. Its default value is secure. This field can contain the following values: **0** secure - The security level of the NPU is configured as Secure **1** non_secure - The security level of the NPU is configured as Non-secure	secure
[31:2]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.5 QBASE register

The base address of the command stream in bytes.

The address must be four-byte aligned.

This register can only be accessed while the NPU is in stopped state. An access made in running state is UNPREDICTABLE.

The register reads or writes the command stream base address. Writing QBASE sets a new start address for execution the next time the NPU enters the run state.

Configurations

This register is available in all configurations.

Attributes

Width

64-bit

Address offset

`0x0010 - 0x0014`

Type

rw

Reset value

`0x0000000000000000`

Bit descriptions

Figure 3-5: QBASE bit assignments

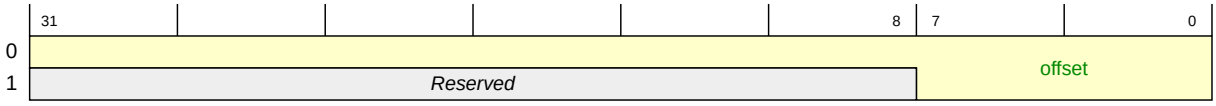


Table 3-8: BASE.QBASE bit descriptions

Bits	Name	Description	Reset
[39:0]	offset	This value is a pointer. Its default value is NULL. This pointer has the following extra properties: **Address space** virtual **Target type** byte	NULL
[63:40]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.6 QREAD register

The read offset in the command stream in bytes. Multiple of four in the range 0-16MB.

This register can be read in stopped or running state.

This register gives the current execution position in the command stream as an offset from QBASE. Specifically, commands in the command stream that occur in sequence before the command at address QBASE+QREAD are complete. Commands that are at address k for k>=QBASE+QREAD can have started execution but are not yet marked as completed.

At all times, 0 <= QREAD <= QSIZE. If QREAD==QSIZE then all commands in the stream are complete. If there is an error, QREAD does not necessarily point to the faulting command.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

`0x0018`

Type

ro

Reset value

`0x00000000`

Bit descriptions

Figure 3-6: QREAD bit assignments

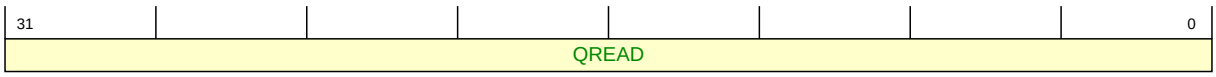


Table 3-9: BASE.QREAD bit descriptions

Bits	Name	Description	Reset
[31:0]	QREAD	The read offset of the current command under execution. This value is an unsigned integer. Its default value is 0x00000000. **Access** ro	0x00000000

Accessibility

Access to this register is restricted to mode ro.

3.3.1.7 QCONFIG register

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

This register can only be accessed while the NPU is in stopped state. An access made in running state is UNPREDICTABLE.

The register encodes the AXI configuration for access to the command stream.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x001C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-7: QCONFIG bit assignments

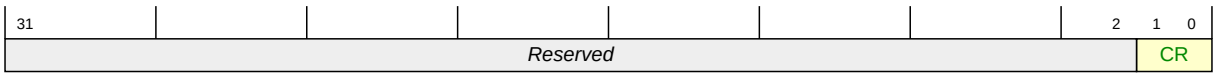


Table 3-10: BASE.QCONFIG bit descriptions

Bits	Name	Description	Reset
[1:0]	cmd_region0 (CR)	The command region configuration number. This value is an unsigned integer. Its default value is 0.	0
[31:2]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.8 QSIZE register

The size of the command stream in bytes. Multiple of four in the range 0-16MB.

This register can only be accessed while the NPU is in stopped state. An access made in running state is UNPREDICTABLE.

The NPU can read to the next multiple of 128 bytes beyond the end size of the command stream.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0020

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-8: QSIZE bit assignments

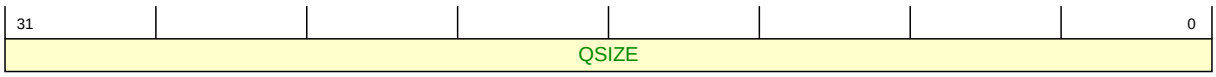


Table 3-11: BASE.QSIZE bit descriptions

Bits	Name	Description	Reset
[31:0]	QSIZE	The size of the next command stream to be executed by the NPU. This value is an unsigned integer. Its default value is 0x00000000.	0x00000000

Accessibility

Access to this register is restricted to mode rw.

3.3.1.9 PROT register

The protection level configured for the NPU when acting as an AXI Requester.

This register can be read while the NPU is in stopped or running state.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0024

Type

ro

Reset value

0x00000000

Bit descriptions

Figure 3-9: PROT bit assignments

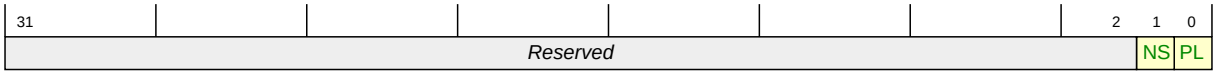


Table 3-12: BASE.PROT bit descriptions

Bits	Name	Description	Reset
[0]	active_CPL (PL)	This bit is used as AxPROT[0] when the NPU is an AXI Requester. After hard reset, this bit is set to Power On Reset Privilege Level (PORPL). After soft reset, this bit is set to pending_CPL, if PPROT[0]==1, otherwise the bit is set to 0. PPROT[0] is the value used by the APB Requester writing the RESET register that starts the reset. For this bit to be effective, there must be a system-level MPU built for the system. This MPU is not part of the NPU deliverables. This value is an enumeration of type privilege_level_t. Its default value is user. This field can contain the following values: **0** user - The NPU is configured for User-level access **1** privileged - The NPU is configured for Privileged-level access **Access** ro	user
[1]	active_CSL (NS)	This bit is used as AxPROT[1] when the NPU is an AXI Requester and set from Pending CSL after reset is complete. After hard reset, this bit is set to Power On Reset Security Level (PORSL), which allows for CPUs that do not support TrustZone. After soft reset, this bit is set to pending_CSL, if PPROT[1]==0, otherwise it is set to 1. PPROT[1] is the value used by the APB Requester writing the RESET register that starts the reset. For this bit to be effective, there must be a memory protection controller included. This memory protection controller is not part of the NPU deliverables. This value is an enumeration of type security_level_t. Its default value is secure. This field can contain the following values: **0** secure - The security level of the NPU is configured as Secure **1** non_secure - The security level of the NPU is configured as Non-secure **Access** ro	secure
[31:2]	Reserved	-	-

Accessibility

Access to this register is restricted to mode ro.

3.3.1.10 CONFIG register

RTL configuration.

This register can be read while the NPU is in stopped or running state.

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

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0028

Type

ro

Reset value

See individual bit resets.

Bit descriptions

Figure 3-10: CONFIG bit assignments

Table 3-13: BASE.CONFIG bit descriptions

Bits	Name	Description	Reset
[3:0]	macs_per_cc (MACS)	This value is an unsigned integer. Its default value is a configuration-specific value. The stored value is a modified version of the represented value, where the modifier applied is log2. The configurations and their values are: **ETHOSU85_128** 7 **ETHOSU85_256** 8 **ETHOSU85_512** 9 **ETHOSU85_1024** 10 **ETHOSU85_2048** 11 **Access** ro	A configuration-specific value

Bits	Name	Description	Reset
[7:4]	cmd_stream_version (CS_VER)	The command stream version accepted by this NPU. This value is an unsigned integer. Its default value is 1 (**IMPLEMENTATION DEFINED**). **Access** ro	1 (**IMPLEMENTATION DEFINED**)
[9:8]	num_axi_sram (AS)	The log2 of the number of AXI SRAM interfaces. This value is an unsigned integer. Its default value is a configuration-specific value. The stored value is a modified version of the represented value, where the modifier applied is log2. The configurations and their values are: **ETHOSU85_128** 1 **ETHOSU85_256** 1 **ETHOSU85_512** 1 **ETHOSU85_1024** 1 **ETHOSU85_2048** 2 **Access** ro	A configuration-specific value
[10]	num_axi_ext (AE)	The log2 of the number of on-chip SRAM memory interfaces. This value is an unsigned integer. Its default value is a configuration-specific value. The stored value is a modified version of the represented value, where the modifier applied is log2. The configurations and their values are: **ETHOSU85_128** 0 **ETHOSU85_256** 0 **ETHOSU85_512** 0 **ETHOSU85_1024** 1 **ETHOSU85_2048** 1 **Access** ro	A configuration-specific value
[11]	Reserved	-	-
[13:12]	num_wd (WD)	The log2 of the number of standard weight decoders. This value is an unsigned integer. Its default value is a configuration-specific value. The stored value is a modified version of the represented value, where the modifier applied is log2. The configurations and their values are: **ETHOSU85_128** 0 **ETHOSU85_256** 0 **ETHOSU85_512** 1 **ETHOSU85_1024** 2 **ETHOSU85_2048** 2 **Access** ro	A configuration-specific value
[26:14]	Reserved	-	-

Bits	Name	Description	Reset
[27]	custom_dma (CD)	The custom DMA configuration. This value is an enumeration of type custom_dma_t. Its default value is not_implemented (**IMPLEMENTATION DEFINED**). This field can contain the following values: **0** not_implemented - Custom DMA feature not implemented **1** implemented - Custom DMA feature implemented **Access** ro	not_implemented (**IMPLEMENTATION DEFINED**)
[31:28]	product	The product configuration. This value is an unsigned integer. Its default value is 2 (**IMPLEMENTATION DEFINED**). **Access** ro	2 (**IMPLEMENTATION DEFINED**)

Accessibility

Access to this register is restricted to mode ro.

3.3.1.11 COND_STATUS register

Condition status of the NPU.

This register can only be accessed while the NPU is in stopped state. An access made in running state is UNPREDICTABLE.

A read of this register returns the status of conditions within the NPU that a following conditional branch can use. This register is read only from the APB interface.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0030

Type

ro

Reset value

0x00000000

Bit descriptions

Figure 3-11: COND_STATUS bit assignments

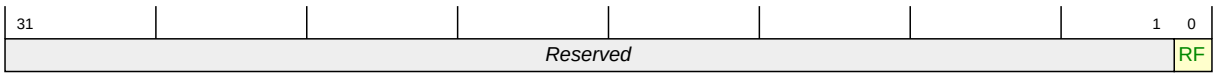


Table 3-14: BASE.COND_STATUS bit descriptions

Bits	Name	Description	Reset
[0]	result_flag (RF)	The tensor result flag. For OFM with a single element, this is bit 0 of the value. Otherwise UNPREDICTABLE. This value is an unsigned integer. Its default value is 0x0. **Access** ro	0x0
[31:1]	Reserved	-	-

Accessibility

Access to this register is restricted to mode ro.

3.3.1.12 POWER_CTRL register

Power control register.

This register can only be accessed while the NPU is in stopped state. An access made in running state is UNPREDICTABLE.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0038

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-12: POWER_CTRL bit assignments

Table 3-15: BASE.POWER_CTRL bit descriptions

Bits	Name	Description	Reset
[5:0]	mac_step_cycles	If the value is zero then power ramping is disabled. If the value is non-zero it specifies the number of cycles between each MAC unit step in ramp up or ramp down. This reduces rapid changes in power consumption by smoothing out the beginning and end of intensive arithmetic operations. The value is of the form 4*n where n is stored in the register. This value is an unsigned integer. Its default value is 0. The stored value is a modified version of the represented value, where the modifier applied is shr(2). **Access** rw	0
[31:6]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.13 REGIONCFG register

Specify which MEM_ATTR register applies to each region.

This register can be read in stopped or running state but only written in stopped state. A write made in running state is UNPREDICTABLE.

For each of the eight memory regions, there is a 2-bit value in the range 0-3. This value specifies which memory attribute register MEM_ATTR0 to MEM_ATTR3 applies to this region.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x003C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-13: REGIONCFG bit assignments

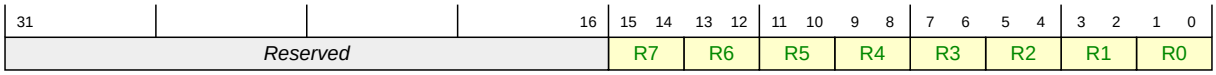


Table 3-16: BASE.REGIONCFG bit descriptions

Bits	Name	Description	Reset
[1:0]	region0 (R0)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[3:2]	region1 (R1)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[5:4]	region2 (R2)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[7:6]	region3 (R3)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[9:8]	region4 (R4)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[11:10]	region5 (R5)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[13:12]	region6 (R6)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[15:14]	region7 (R7)	Bits for Region(n) configuration. This value is an unsigned integer. Its default value is 0.	0
[31:16]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.14 MEM_ATTR0 register

Memory attributes 0.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0040

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-14: MEM_ATTR0 bit assignments

Table 3-17: BASE.MEM_ATTR0 bit descriptions

Bits	Name	Description	Reset
[1:0]	mem_domain (MEMD)	Memory domain. This value is an enumeration of type axi_mem_domain_t. Its default value is non_sharable. This field can contain the following values: **0x0** non_sharable - AxDomain=00 (non device memory only) **0x1** inner_sharable - AxDomain=01 (non device memory only) **0x2** outer_sharable - AxDomain=10 (non device memory only) **0x3** system - AxDomain=11 (normal and device memory only)	non_sharable

Bits	Name	Description	Reset
[2]	axi_port (AP)	The AXI port select. This value is an enumeration of type axi_port_t. Its default value is sram. This field can contain the following values: **0x0** sram - SRAM AXI port or ports selected **0x1** ext - EXT AXI port or ports selected	sram
[3]	Reserved	-	-
[7:4]	memtype	The Memtype used to encode AxCACHE signals. This value is an enumeration of type axi_mem_encoding_t. Its default value is device_non_bufferable. This field can contain the following values: **0x0** device_non_bufferable - ARCACHE=0000, AWCACHE=0000 **0x1** device_bufferable - ARCACHE=0001, AWCACHE=0001 **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	device_non_bufferable
[31:8]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.15 MEM_ATTR1 register

Memory attributes 1.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0044

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-15: MEM_ATTR1 bit assignments

Table 3-18: BASE.MEM_ATTR1 bit descriptions

Bits	Name	Description	Reset
[1:0]	mem_domain (MEMD)	Memory domain. This value is an enumeration of type axi_mem_domain_t. Its default value is non_sharable. This field can contain the following values: **0x0** non_sharable - AxDomain=00 (non device memory only) **0x1** inner_sharable - AxDomain=01 (non device memory only) **0x2** outer_sharable - AxDomain=10 (non device memory only) **0x3** system - AxDomain=11 (normal and device memory only)	non_sharable
[2]	axi_port (AP)	The AXI port select. This value is an enumeration of type axi_port_t. Its default value is sram. This field can contain the following values: **0x0** sram - SRAM AXI port or ports selected **0x1** ext - EXT AXI port or ports selected	sram
[3]	Reserved	-	-

Bits	Name	Description	Reset
[7:4]	memtype	The Memtype used to encode AxCACHE signals. This value is an enumeration of type axi_mem_encoding_t. Its default value is device_non_bufferable. This field can contain the following values: **0x0** device_non_bufferable - ARCACHE=0000, AWCACHE=0000 **0x1** device_bufferable - ARCACHE=0001, AWCACHE=0001 **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	device_non_bufferable
[31:8]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.16 MEM_ATTR2 register

Memory attributes 2.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0048

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-16: MEM_ATTR2 bit assignments

Table 3-19: BASE.MEM_ATTR2 bit descriptions

Bits	Name	Description	Reset
[1:0]	mem_domain (MEMD)	Memory domain. This value is an enumeration of type axi_mem_domain_t. Its default value is non_sharable. This field can contain the following values: **0x0** non_sharable - AxDomain=00 (non device memory only) **0x1** inner_sharable - AxDomain=01 (non device memory only) **0x2** outer_sharable - AxDomain=10 (non device memory only) **0x3** system - AxDomain=11 (normal and device memory only)	non_sharable
[2]	axi_port (AP)	The AXI port select. This value is an enumeration of type axi_port_t. Its default value is sram. This field can contain the following values: **0x0** sram - SRAM AXI port or ports selected **0x1** ext - EXT AXI port or ports selected	sram
[3]	Reserved	-	-

Bits	Name	Description	Reset
[7:4]	memtype	The Memtype used to encode AxCACHE signals. This value is an enumeration of type axi_mem_encoding_t. Its default value is device_non_bufferable. This field can contain the following values: **0x0** device_non_bufferable - ARCACHE=0000, AWCACHE=0000 **0x1** device_bufferable - ARCACHE=0001, AWCACHE=0001 **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	device_non_bufferable
[31:8]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.17 MEM_ATTR3 register

Memory attributes 3.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x004C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-17: MEM_ATTR3 bit assignments

Table 3-20: BASE.MEM_ATTR3 bit descriptions

Bits	Name	Description	Reset
[1:0]	mem_domain (MEMD)	Memory domain. This value is an enumeration of type axi_mem_domain_t. Its default value is non_sharable. This field can contain the following values: **0x0** non_sharable - AxDomain=00 (non device memory only) **0x1** inner_sharable - AxDomain=01 (non device memory only) **0x2** outer_sharable - AxDomain=10 (non device memory only) **0x3** system - AxDomain=11 (normal and device memory only)	non_sharable
[2]	axi_port (AP)	The AXI port select. This value is an enumeration of type axi_port_t. Its default value is sram. This field can contain the following values: **0x0** sram - SRAM AXI port or ports selected **0x1** ext - EXT AXI port or ports selected	sram
[3]	Reserved	-	-

Bits	Name	Description	Reset
[7:4]	memtype	The Memtype used to encode AxCACHE signals. This value is an enumeration of type axi_mem_encoding_t. Its default value is device_non_bufferable. This field can contain the following values: **0x0** device_non_bufferable - ARCACHE=0000, AWCACHE=0000 **0x1** device_bufferable - ARCACHE=0001, AWCACHE=0001 **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	device_non_bufferable
[31:8]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.18 AXI_SRAM register

The AXI configuration for SRAM ports.

The limits in this register apply to each SRAM port separately in the case of multiple SRAM AXI ports.

Table 3-21: SRAM ports (128b) 40-bit address configuration-specific ranges for max outstanding reads and max outstanding writes

NPU configurations (MACs/CC)	Number of ports	Max outstanding reads per port	Max outstanding writes per port
128	2	12	16
256	2	12	16
512	2	12	16
1024	2	12	16
2048	4	12	16

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0050

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-18: AXI_SRAM bit assignments

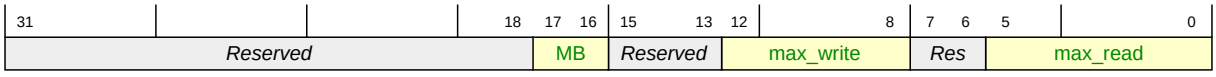


Table 3-22: BASE.AXI_SRAM bit descriptions

Bits	Name	Description	Reset
[5:0]	max_outstanding_read_m1 (max_read)	The range is 0 to 63. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.CFG registers and the NPU issue ability for the given data channel. See the initial table on configuration-specific ranges for the maximum number of issues for a given configuration. This value is an unsigned integer. Its default value is 0x00.	0x00
[7:6]	Reserved	-	-

Bits	Name	Description	Reset
[12:8]	max_outstanding_write_m1 (max_write)	The range is 0 to 31. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.CFG registers and the NPU issue ability for the given data channel. See the initial table on configuration-specific ranges for the maximum number of issues for a given configuration. This value is an unsigned integer. Its default value is 0x00.	0x00
[15:13]	Reserved	-	-
[17:16]	max_beats (MB)	Bursts are split at an alignment that is the minimum of this alignment and the alignment set by the corresponding configuration pins. This value is an enumeration of type max_beats_t. Its default value is B64. This field can contain the following values: **0** B64 - AXI bursts are split at a 64-byte aligned boundary **1** B128 - AXI bursts are split at a 128-byte aligned boundary **2** B256 - AXI bursts are split at a 256-byte aligned boundary **3** Reserved	B64
[31:18]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.19 AXI_EXT register

The AXI configuration for EXT ports.

The limits in this register apply to each EXT port separately in the case of multiple EXT AXI ports.

Table 3-23: EXT ports (128b) 40-bit address configuration-specific ranges for max outstanding reads and max outstanding writes

NPU configurations (MACs/CC)	Number of ports	Max outstanding reads per port	Max outstanding writes port
128	1	32	32
256	1	32	32
512	1	64	32
1024	2	64	32
2048	2	64	32

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0054

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-19: AXI_EXT bit assignments

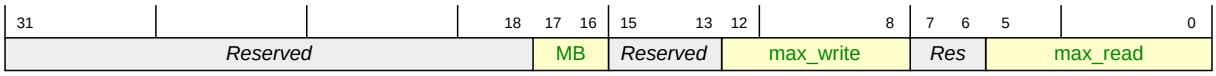


Table 3-24: BASE.AXI_EXT bit descriptions

Bits	Name	Description	Reset
[5:0]	max_outstanding_read_m1 (max_read)	The range is 0 to 63. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.CFG registers and the NPU issue ability for the given data channel. See the initial table on configuration-specific ranges for the maximum number of issues for a given configuration. This value is an unsigned integer. Its default value is 0x00.	0x00
[7:6]	Reserved	-	-
[12:8]	max_outstanding_write_m1 (max_write)	The range is 0 to 31. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.CFG registers and the NPU issue ability for the given data channel. See the initial table on configuration-specific ranges for the maximum number of issues for a given configuration. This value is an unsigned integer. Its default value is 0x00.	0x00
[15:13]	Reserved	-	-

Bits	Name	Description	Reset
[17:16]	max_beats (MB)	Bursts are split at an alignment that is the minimum of this alignment and the alignment set by the corresponding configuration pins. This value is an enumeration of type max_beats_t. Its default value is B64. This field can contain the following values: **0** B64 - AXI bursts are split at a 64-byte aligned boundary **1** B128 - AXI bursts are split at a 128-byte aligned boundary **2** B256 - AXI bursts are split at a 256-byte aligned boundary **3** Reserved	B64
[31:18]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.1.20 CFG_SRAM_CAP register

The value of the CFGSRAMCAP pins, SRAM AXI ports cap.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0060

Type

ro

Reset value

See individual bit resets.

Bit descriptions

Figure 3-20: CFG_SRAM_CAP bit assignments

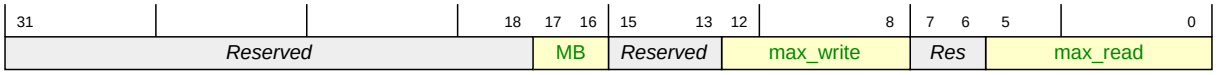


Table 3-25: BASE.CFG_SRAM_CAP bit descriptions

Bits	Name	Description	Reset
[5:0]	max_outstanding_read_m1 (max_read)	The range is 0 to 63. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.AXI registers and the NPU issue ability for the given data channel. This value is an unsigned integer.	-
[7:6]	Reserved	-	-
[12:8]	max_outstanding_write_m1 (max_write)	The range is 0 to 31. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.AXI registers and the NPU issue ability for the given data channel. This value is an unsigned integer.	-
[15:13]	Reserved	-	-
[17:16]	max_beats (MB)	Bursts are split at an alignment that is the minimum of this alignment and the alignment set by the corresponding AXI configuration register. This value is an enumeration of type max_beats_t. This field can contain the following values: **0** B64 - AXI bursts are split at a 64-byte aligned boundary **1** B128 - AXI bursts are split at a 128-byte aligned boundary **2** B256 - AXI bursts are split at a 256-byte aligned boundary **3** Reserved	-
[31:18]	Reserved	-	-

Accessibility

Access to this register is restricted to mode ro.

3.3.1.21 CFG_EXT_CAP register

The value of the CFGEXTCAP pins, EXT AXI ports cap.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0064

Type

ro

Reset value

See individual bit resets.

Bit descriptions

Figure 3-21: CFG_EXT_CAP bit assignments

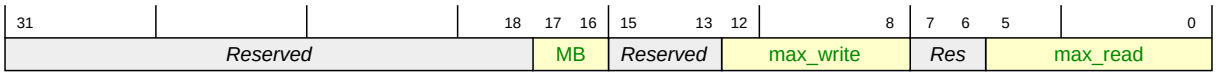


Table 3-26: BASE.CFG_EXT_CAP bit descriptions

Bits	Name	Description	Reset
[5:0]	max_outstanding_read_m1 (max_read)	The range is 0 to 63. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.AXI registers and the NPU issue ability for the given data channel. This value is an unsigned integer.	-
[7:6]	Reserved	-	-
[12:8]	max_outstanding_write_m1 (max_write)	The range is 0 to 31. The limit applies to each port separately in the case of multiple ports. This provides an upper limit and the limit may not be reached. For example, the number of outstanding accesses is also limited by the BASE.AXI registers and the NPU issue ability for the given data channel. This value is an unsigned integer.	-
[15:13]	Reserved	-	-

Bits	Name	Description	Reset
[17:16]	max_beats (MB)	Bursts are split at an alignment that is the minimum of this alignment and the alignment set by the corresponding AXI configuration register. This value is an enumeration of type max_beats_t. This field can contain the following values: **0** B64 - AXI bursts are split at a 64-byte aligned boundary **1** B128 - AXI bursts are split at a 128-byte aligned boundary **2** B256 - AXI bursts are split at a 256-byte aligned boundary **3** Reserved	-
[31:18]	Reserved	-	-

Accessibility

Access to this register is restricted to mode ro.

3.3.1.22 CFG_SRAM_HASH0 register

The value of the CFGSRAMHASH0 pins, SRAM AXI port select bit 0 hash.

The result of this hash function is used to define bit 0 of the SRAM AXI port select for configurations with 2 or 4 SRAM ports.

Configurations

This register is available in all configurations.

Attributes

Width

64-bit

Address offset

0x0068 - 0x006C

Type

ro

Reset value

See individual bit resets.

Bit descriptions

Figure 3-22: CFG_SRAM_HASH0 bit assignments

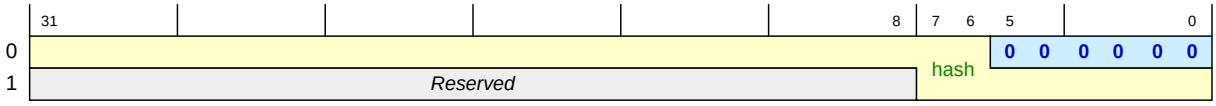


Table 3-27: BASE.CFG_SRAM_HASH0 bit descriptions

Bits	Name	Description	Reset
[5:0]	zero	These bits must be zero. This value is an unsigned integer. It must have the exact value 0b000000.	0b000000
[39:6]	hash	Hash function. This mask is combined with the address using a logical bitwise *AND* and then the resultant bits are all combined using an exclusive *OR*. The result is a single bit that is used for AXI port selection. This value is an unsigned integer.	-
[63:40]	Reserved	-	-

Accessibility

Access to this register is restricted to mode ro.

3.3.1.23 CFG_SRAM_HASH1 register

The value of the CFGSRAMHASH1 pins, SRAM AXI port select bit 1 hash.

The result of this hash function is used to define bit 1 of the SRAM AXI port select for configurations with 4 SRAM ports.

This register mirrors the values of the CFGSRAMHASH1 configuration pins.

CFGSRAMHASH1 = 0x0 for NUM_MACS = 128, 256 and 512

CFGSRAMHASH1 for NUM_MACS = 1024 and 2048 reflects the values on the configuration pins

Configurations

This register is available in all configurations.

Attributes

Width

64-bit

Address offset

0x0070 - 0x0074

Type

ro

Reset value

See individual bit resets.

Bit descriptions

Figure 3-23: CFG_SRAM_HASH1 bit assignments

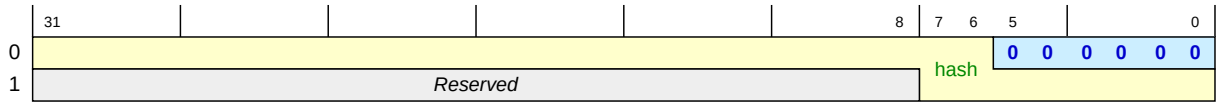


Table 3-28: BASE.CFG_SRAM_HASH1 bit descriptions

Bits	Name	Description	Reset
[5:0]	zero	These bits must be zero. This value is an unsigned integer. It must have the exact value 0b000000.	0b000000
[39:6]	hash	Hash function. This mask is combined with the address using a logical bitwise *AND* and then the resultant bits are all combined using an exclusive *OR*. The result is a single bit that is used for AXI port selection. This value is an unsigned integer.	-
[63:40]	Reserved	-	-

Accessibility

Access to this register is restricted to mode ro.

3.3.1.24 CFG_EXT_HASH0 register

The value of the CFGEXTHASH0 pins, EXT AXI port select bit 0 hash.

The result of this hash function is used to define bit 0 of the EXT AXI port select for configurations with 2 EXT ports.

Configurations

This register is available in all configurations.

Attributes

Width

64-bit

Address offset

0x0078 - 0x007C

Type

ro

Reset value

See individual bit resets.

Bit descriptions

Figure 3-24: CFG_EXT_HASH0 bit assignments

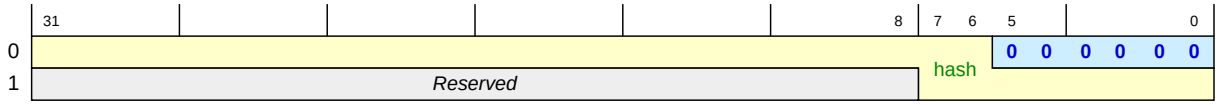


Table 3-29: BASE.CFG_EXT_HASH0 bit descriptions

Bits	Name	Description	Reset
[5:0]	zero	These bits must be zero. This value is an unsigned integer. It must have the exact value 0b000000.	0b000000
[39:6]	hash	Hash function. This mask is combined with the address using a logical bitwise *AND* and then the resultant bits are all combined using an exclusive *OR*. The result is a single bit that is used for AXI port selection. This value is an unsigned integer.	-
[63:40]	Reserved	-	-

Accessibility

Access to this register is restricted to mode ro.

3.3.2 BASE_POINTERS register summary

NPU register bank.

Table 3-30: BASE_POINTERS register summary

Offset	Name	Type	Reset	Width	Description
0x0000 - 0x003C	BASEP0 [0..7]	rw	0x0000000000000000	64-byte	AXI base address of the respective region number 0 - 7

Offset	Name	Type	Reset	Width	Description
0x0040 - 0x007C	Reserved	-	-	64-byte	-

3.3.2.1 BASEP_ARRAY[0..7] register array

AXI base address of the respective region number 0 - 7.

This is an array of similar registers, collectively referred to as BASEP_ARRAY[]. The definition of element 0 is as follows:

This base address is added to all address offsets in the command stream that access region 0. If the region contains data requiring A-byte alignment then the base address must be a multiple of A. The driver uses this register to set the run-time address of a region.

Configurations

This register is available in all configurations.

Attributes

Width

64-bit

Address offset

0x0000 - 0x0004

Type

rw

Reset value

0x0000000000000000

Bit descriptions

Figure 3-25: BASEP0 bit assignments

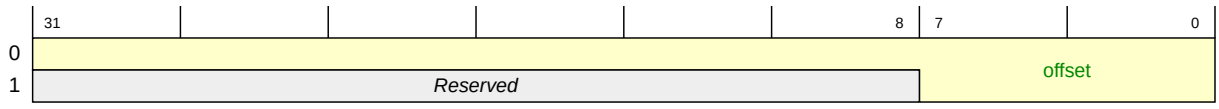


Table 3-31: BASE_POINTERS.BASEP_ARRAY bit descriptions

Bits	Name	Description	Reset
[39:0]	offset	This value is a pointer. Its default value is NULL. This pointer has the following extra properties: **Address space** virtual **Target type** byte	NULL
[63:40]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.3 ID register summary

NPU register bank.

Table 3-32: ID register summary

Offset	Name	Type	Reset	Width	Description
0x0000 - 0x00CC	Reserved	-	-	208-byte	-
0x00D0	PID4	ro	0x00000004	32-bit	Peripheral ID byte 4 (Arm=code 4)
0x00D4	PID5	ro	0x00000000	32-bit	Peripheral ID byte 5 (reserved)
0x00D8	PID6	ro	0x00000000	32-bit	Peripheral ID byte 6 (reserved)
0x00DC	PID7	ro	0x00000000	32-bit	Peripheral ID byte 7 (reserved)
0x00E0	PID0	ro	0x00000082	32-bit	Peripheral ID byte 0. This is bits[7:0] of the part number
0x00E4	PID1	ro	0x000000B5	32-bit	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]
0x00E8	PID2	ro	0x0000000B	32-bit	Peripheral ID byte 2. This is bits[6:4] of the Arm ID in bits[2:0], and bit 3 indicates format B
0x00EC	PID3	ro	0x00000000	32-bit	Peripheral ID byte 3
0x00F0	CID0	ro	0x0000000D	32-bit	Component ID byte 0
0x00F4	CID1	ro	0x000000F0	32-bit	Component ID byte 1
0x00F8	CID2	ro	0x00000005	32-bit	Component ID byte 2
0x00FC	CID3	ro	0x000000B1	32-bit	Component ID byte 3

3.3.3.1 PID4 register

Peripheral ID byte 4 (Arm=code 4).

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00D0

Type

ro

Reset value

0x00000004

Bit descriptions

Figure 3-26: PID4 bit assignments

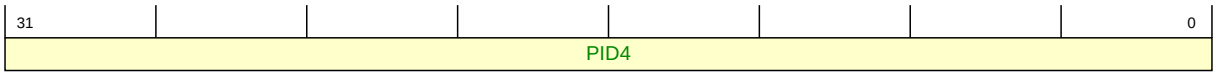


Table 3-33: ID.PID4 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID4	This value is an unsigned integer. Its default value is 0x00000004. **Access** ro	0x00000004

Accessibility

Access to this register is restricted to mode ro.

3.3.3.2 PID5 register

Peripheral ID byte 5 (reserved).

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00D4

Type

ro

Reset value

0x00000000

Bit descriptions

Figure 3-27: PID5 bit assignments

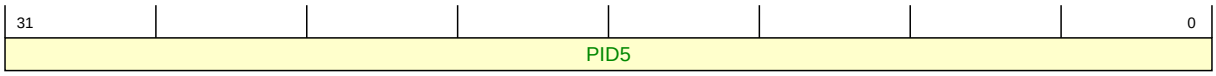


Table 3-34: ID.PID5 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID5	This value is an unsigned integer. Its default value is 0x00000000. **Access** ro	0x00000000

Accessibility

Access to this register is restricted to mode ro.

3.3.3.3 PID6 register

Peripheral ID byte 6 (reserved).

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00D8

Type

ro

Reset value

0x00000000

Bit descriptions

Figure 3-28: PID6 bit assignments

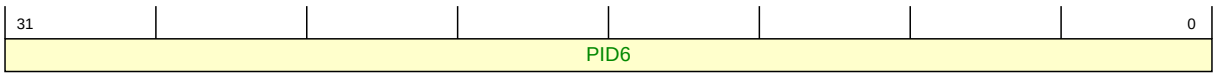


Table 3-35: ID.PID6 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID6	This value is an unsigned integer. Its default value is 0x00000000. **Access** ro	0x00000000

Accessibility

Access to this register is restricted to mode ro.

3.3.3.4 PID7 register

Peripheral ID byte 7 (reserved).

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00DC

Type

ro

Reset value

0x00000000

Bit descriptions

Figure 3-29: PID7 bit assignments

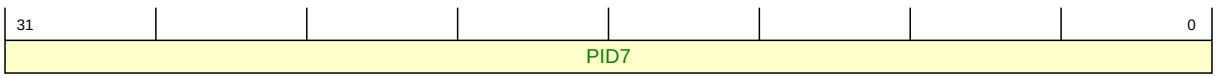


Table 3-36: ID.PID7 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID7	This value is an unsigned integer. Its default value is 0x00000000. **Access** ro	0x00000000

Accessibility

Access to this register is restricted to mode ro.

3.3.3.5 PID0 register

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

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00E0

Type

ro

Reset value

0x00000082

Bit descriptions

Figure 3-30: PID0 bit assignments

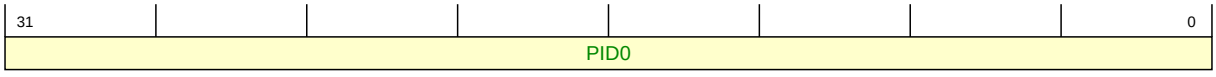


Table 3-37: ID.PID0 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID0	This value is an unsigned integer. Its default value is 0x00000082 (**IMPLEMENTATION DEFINED**). **Access** ro	0x00000082 (**IMPLEMENTATION DEFINED**)

Accessibility

Access to this register is restricted to mode ro.

3.3.3.6 PID1 register

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].

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00E4

Type

ro

Reset value

0x000000B5

Bit descriptions

Figure 3-31: PID1 bit assignments

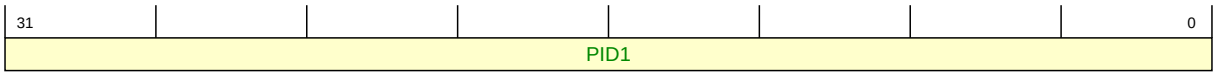


Table 3-38: ID.PID1 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID1	This value is an unsigned integer. Its default value is 0x000000B5. **Access** ro	0x000000B5

Accessibility

Access to this register is restricted to mode ro.

3.3.3.7 PID2 register

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

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00E8

Type

ro

Reset value

0x0000000B

Bit descriptions

Figure 3-32: PID2 bit assignments

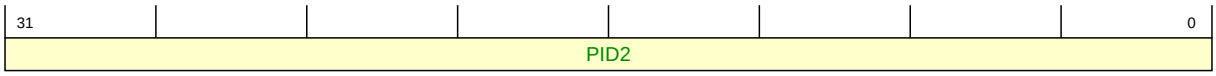


Table 3-39: ID.PID2 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID2	This value is an unsigned integer. Its default value is 0x0000000B. **Access** ro	0x0000000B

Accessibility

Access to this register is restricted to mode ro.

3.3.3.8 PID3 register

Peripheral ID byte 3.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00EC

Type

ro

Reset value

0x00000000

Bit descriptions

Figure 3-33: PID3 bit assignments

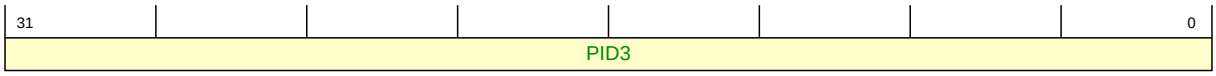


Table 3-40: ID.PID3 bit descriptions

Bits	Name	Description	Reset
[31:0]	PID3	This value is an unsigned integer. Its default value is 0x00000000. **Access** ro	0x00000000

Accessibility

Access to this register is restricted to mode ro.

3.3.3.9 CID0 register

Component ID byte 0.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00F0

Type

ro

Reset value

0x0000000D

Bit descriptions

Figure 3-34: CID0 bit assignments

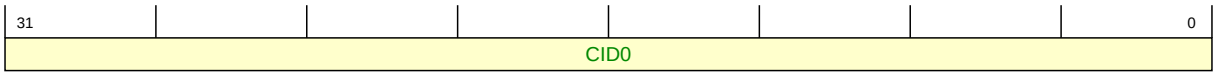


Table 3-41: ID.CID0 bit descriptions

Bits	Name	Description	Reset
[31:0]	CID0	This value is an unsigned integer. Its default value is 0x0000000D. **Access** ro	0x0000000D

Accessibility

Access to this register is restricted to mode ro.

3.3.3.10 CID1 register

Component ID byte 1.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00F4

Type

ro

Reset value

0x000000F0

Bit descriptions

Figure 3-35: CID1 bit assignments

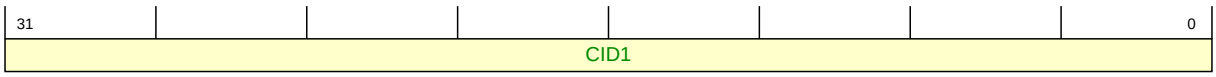


Table 3-42: ID.CID1 bit descriptions

Bits	Name	Description	Reset
[31:0]	CID1	This value is an unsigned integer. Its default value is 0x000000F0. **Access** ro	0x000000F0

Accessibility

Access to this register is restricted to mode ro.

3.3.3.11 CID2 register

Component ID byte 2.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

`0x00F8`

Type

ro

Reset value

`0x00000005`

Bit descriptions

Figure 3-36: CID2 bit assignments

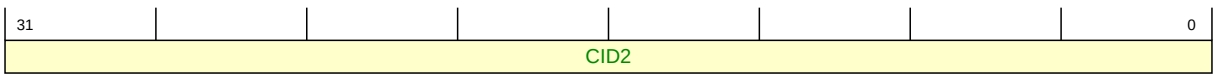


Table 3-43: ID.CID2 bit descriptions

Bits	Name	Description	Reset
[31:0]	CID2	This value is an unsigned integer. Its default value is 0x00000005. **Access** ro	0x00000005

Accessibility

Access to this register is restricted to mode ro.

3.3.3.12 CID3 register

Component ID byte 3.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x00FC

Type

ro

Reset value

0x000000B1

Bit descriptions

Figure 3-37: CID3 bit assignments

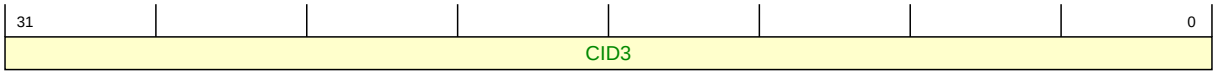


Table 3-44: ID.CID3 bit descriptions

Bits	Name	Description	Reset
[31:0]	CID3	This value is an unsigned integer. Its default value is 0x000000B1. **Access** ro	0x000000B1

Accessibility

Access to this register is restricted to mode ro.

3.3.4 PMU register summary

Performance monitoring.

Note

Register writes to the PMU, other than those that modify PMCR.cnt_en, are not guaranteed to take effect unless PMCR.cnt_en=1.

Note

Attributes are marked with an asterisk * if child elements in the register can override this attribute.

Table 3-45: PMU register summary

Offset	Name	Type	Reset	Width	Description
0x0000	PMCR	rw*	See individual bit resets	32-bit	PMU register control
0x0004	PMCNTENSET	rw	0x00000000	32-bit	Count enable set register
0x0008	PMCNTENCLR	rw	0x00000000	32-bit	Count enable clear register
0x000C	PMOVSSET	rw	0x00000000	32-bit	Overflow flag status set register
0x0010	PMOVSCLR	rw	0x00000000	32-bit	Overflow flag status clear register
0x0014	PMINTSET	rw	0x00000000	32-bit	Interrupt enable set register
0x0018	PMINTCLR	rw	0x00000000	32-bit	Interrupt enable clear register
0x001C	Reserved	-	-	32-bit	-
0x0020 - 0x0024	PMCCNTR	rw	0x0000000000000000	64-bit	Performance monitor cycle count register
0x0028	PMCCNTR_CFG	rw	0x00000000	32-bit	Set start/stop event on the cycle counter
0x002C	PMCAXI_CHAN	rw	0x00000000	32-bit	Set which AXI channel to monitor for latency measurements in PMU
0x0030	PMCLUT	rw	0x00000000	32-bit	Performance monitor control for lookup table
0x0034 - 0x007C	Reserved	-	-	76-byte	-

3.3.4.1 PMCR register

PMU register control.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0000

Type

rw*

Reset value

See individual bit resets.

Bit descriptions

Figure 3-38: PMCR bit assignments

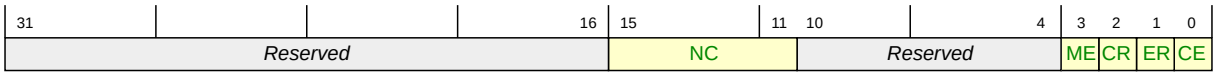


Table 3-46: PMU.PMCR bit descriptions

Bits	Name	Description	Reset
[0]	cnt_en (CE)	This value is an unsigned integer. Its default value is 0x0. **Access** rw	0x0
[1]	event_cnt_rst (ER)	This value is an unsigned integer. **Access** wo	-
[2]	cycle_cnt_rst (CR)	This value is an unsigned integer. **Access** wo	-
[3]	mask_en (ME)	This value is an unsigned integer. Its default value is 0x0.	0x0
[10:4]	Reserved	-	-
[15:11]	num_event_cnt (NC)	This value is an unsigned integer. Its default value is 8. **Access** ro	8
[31:16]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.4.2 PMCNTENSET register

Count enable set register.

Enables the dedicated cycle counter, PMCCNTR, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

PMCNTENSET is used with the PMCNTENCLR register. It is implemented in hardware with the same underlying state as PMCNTENCLR.

If counter n>0 is enabled then it counts the event configured by PMEVTYPER[n]. If counter 0 is enabled the event counted depends on PMCLUT.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0004

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-39: PMCNTENSET bit assignments

Table 3-47: PMU.PMCNTENSET bit descriptions

Bits	Name	Description	Reset
[0]	EVENT_CNT_0 (E0)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR0 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR0 event counter is enabled. When written, enables PMEVCNTR0. **Access** rw	false
[1]	EVENT_CNT_1 (E1)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR1 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR1 event counter is enabled. When written, enables PMEVCNTR1. **Access** rw	false
[2]	EVENT_CNT_2 (E2)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR2 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR2 event counter is enabled. When written, enables PMEVCNTR2. **Access** rw	false
[3]	EVENT_CNT_3 (E3)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR3 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR3 event counter is enabled. When written, enables PMEVCNTR3. **Access** rw	false

Bits	Name	Description	Reset
[4]	EVENT_CNT_4 (E4)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR4 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR4 event counter is enabled. When written, enables PMEVCNTR4. **Access** rw	false
[5]	EVENT_CNT_5 (E5)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR5 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR5 event counter is enabled. When written, enables PMEVCNTR5. **Access** rw	false
[6]	EVENT_CNT_6 (E6)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR6 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR6 event counter is enabled. When written, enables PMEVCNTR6. **Access** rw	false
[7]	EVENT_CNT_7 (E7)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR7 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR7 event counter is enabled. When written, enables PMEVCNTR7. **Access** rw	false
[30:8]	Reserved	-	-

Bits	Name	Description	Reset
[31]	CYCLE_CNT (CC)	Enables the dedicated cycle counter, PMCCNTR. This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means the cycle counter is disabled. When written, has no effect. **1** When read, means the cycle counter is enabled. When written, enables the cycle counter. **Access** rw	false

Accessibility

Access to this register is restricted to mode rw.

3.3.4.3 PMCNTENCLR register

Count enable clear register.

Disables the dedicated cycle counter, PMCCNTR, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

PMCNTENCLR is used with the PMCNTENSET register. It is implemented in hardware with the same underlying state as PMCNTENSET.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0008

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-40: PMCNTENCLR bit assignments

Table 3-48: PMU.PMCNTENCLR bit descriptions

Bits	Name	Description	Reset
[0]	EVENT_CNT_0 (E0)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR0 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR0 is enabled. When written, disables PMEVCNTR0. **Access** rw	false
[1]	EVENT_CNT_1 (E1)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR1 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR1 is enabled. When written, disables PMEVCNTR1. **Access** rw	false
[2]	EVENT_CNT_2 (E2)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR2 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR2 is enabled. When written, disables PMEVCNTR2. **Access** rw	false

Bits	Name	Description	Reset
[3]	EVENT_CNT_3 (E3)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR3 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR3 is enabled. When written, disables PMEVCNTR3. **Access** rw	false
[4]	EVENT_CNT_4 (E4)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR4 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR4 is enabled. When written, disables PMEVCNTR4. **Access** rw	false
[5]	EVENT_CNT_5 (E5)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR5 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR5 is enabled. When written, disables PMEVCNTR5. **Access** rw	false
[6]	EVENT_CNT_6 (E6)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR6 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR6 is enabled. When written, disables PMEVCNTR6. **Access** rw	false

Bits	Name	Description	Reset
[7]	EVENT_CNT_7 (E7)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR7 is disabled. When written, has no effect. **1** When read, means that PMEVCNTR7 is enabled. When written, disables PMEVCNTR7. **Access** rw	false
[30:8]	Reserved	-	-
[31]	CYCLE_CNT (CC)	Disables the dedicated cycle counter, PMCCNTR. This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means the cycle counter is disabled. When written, has no effect. **1** When read, means the cycle counter is enabled. When written, disables the cycle counter. **Access** rw	false

Accessibility

Access to this register is restricted to mode rw.

3.3.4.4 PMOVSSET register

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 PMEVCNTR<n>.

PMOVSSET is used with the PMOVSCLR register. It is implemented in hardware with the same underlying state as PMOVSCLR.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x000C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-41: PMOVSSET bit assignments

Table 3-49: PMU.PMOVSSET bit descriptions

Bits	Name	Description	Reset
[0]	EVENT_CNT_0_OVF (E0)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR0 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR0 has overflowed. When written, sets the PMEVCNTR0 overflow bit to 1. **Access** rw	false
[1]	EVENT_CNT_1_OVF (E1)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR1 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR1 has overflowed. When written, sets the PMEVCNTR1 overflow bit to 1. **Access** rw	false

Bits	Name	Description	Reset
[2]	EVENT_CNT_2_OVF (E2)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR2 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR2 has overflowed. When written, sets the PMEVCNTR2 overflow bit to 1. **Access** rw	false
[3]	EVENT_CNT_3_OVF (E3)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR3 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR3 has overflowed. When written, sets the PMEVCNTR3 overflow bit to 1. **Access** rw	false
[4]	EVENT_CNT_4_OVF (E4)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR4 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR4 has overflowed. When written, sets the PMEVCNTR4 overflow bit to 1. **Access** rw	false
[5]	EVENT_CNT_5_OVF (E5)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR5 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR5 has overflowed. When written, sets the PMEVCNTR5 overflow bit to 1. **Access** rw	false

Bits	Name	Description	Reset
[6]	EVENT_CNT_6_OVF (E6)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR6 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR6 has overflowed. When written, sets the PMEVCNTR6 overflow bit to 1. **Access** rw	false
[7]	EVENT_CNT_7_OVF (E7)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR7 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR7 has overflowed. When written, sets the PMEVCNTR7 overflow bit to 1. **Access** rw	false
[30:8]	Reserved	-	-
[31]	CYCLE_CNT_OVF (CC)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means the cycle counter has not overflowed. When written, has no effect. **1** When read, means the cycle counter has overflowed. When written, sets the overflow bit to 1. **Access** rw	false

Accessibility

Access to this register is restricted to mode rw.

3.3.4.5 PMOVSCLR register

Overflow flag status clear register.

Contains the state of the overflow bit for the dedicated cycle counter, PMCCNTR, and each of the implemented event counters PMEVCNTR<n>. Writing to this register clears these bits.

PMOVSCLR is used in conjunction with the PMOVSSET register. It is implemented in hardware with the same underlying state as PMOVSSET.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0010

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-42: PMOVSCLR bit assignments

Table 3-50: PMU.PMOVSCLR bit descriptions

Bits	Name	Description	Reset
[0]	EVENT_CNT_0_OVF (E0)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR0 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR0 has overflowed. When written, clears the PMEVCNTR0 overflow bit to 0. **Access** rw	false

Bits	Name	Description	Reset
[1]	EVENT_CNT_1_OVF (E1)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR1 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR1 has overflowed. When written, clears the PMEVCNTR1 overflow bit to 0. **Access** rw	false
[2]	EVENT_CNT_2_OVF (E2)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR2 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR2 has overflowed. When written, clears the PMEVCNTR2 overflow bit to 0. **Access** rw	false
[3]	EVENT_CNT_3_OVF (E3)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR3 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR3 has overflowed. When written, clears the PMEVCNTR3 overflow bit to 0. **Access** rw	false
[4]	EVENT_CNT_4_OVF (E4)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR4 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR4 has overflowed. When written, clears the PMEVCNTR4 overflow bit to 0. **Access** rw	false

Bits	Name	Description	Reset
[5]	EVENT_CNT_5_OVF (E5)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR5 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR5 has overflowed. When written, clears the PMEVCNTR5 overflow bit to 0. **Access** rw	false
[6]	EVENT_CNT_6_OVF (E6)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR6 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR6 has overflowed. When written, clears the PMEVCNTR6 overflow bit to 0. **Access** rw	false
[7]	EVENT_CNT_7_OVF (E7)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that PMEVCNTR7 has not overflowed. When written, has no effect. **1** When read, means that PMEVCNTR7 has overflowed. When written, clears the PMEVCNTR7 overflow bit to 0. **Access** rw	false
[30:8]	Reserved	-	-
[31]	CYCLE_CNT_OVF (CC)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means the cycle counter has not overflowed. When written, has no effect. **1** When read, means the cycle counter has overflowed. When written, clears the overflow bit to 0. **Access** rw	false

Accessibility

Access to this register is restricted to mode rw.

3.3.4.6 PMINTSET register

Interrupt enable set register.

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

PMINTSET is used with the PMINTCLR register. It is implemented in hardware with the same underlying state as PMINTCLR.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0014

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-43: PMINTSET bit assignments

Table 3-51: PMU.PMINTSET bit descriptions

Bits	Name	Description	Reset
[0]	EVENT_CNT_0_INT (E0)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR0 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR0 event counter interrupt request is enabled. When written, enables the PMEVCNTR0 interrupt request. **Access** rw	false
[1]	EVENT_CNT_1_INT (E1)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR1 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR1 event counter interrupt request is enabled. When written, enables the PMEVCNTR1 interrupt request. **Access** rw	false
[2]	EVENT_CNT_2_INT (E2)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR2 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR2 event counter interrupt request is enabled. When written, enables the PMEVCNTR2 interrupt request. **Access** rw	false

Bits	Name	Description	Reset
[3]	EVENT_CNT_3_INT (E3)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR3 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR3 event counter interrupt request is enabled. When written, enables the PMEVCNTR3 interrupt request. **Access** rw	false
[4]	EVENT_CNT_4_INT (E4)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR4 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR4 event counter interrupt request is enabled. When written, enables the PMEVCNTR4 interrupt request. **Access** rw	false
[5]	EVENT_CNT_5_INT (E5)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR5 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR5 event counter interrupt request is enabled. When written, enables the PMEVCNTR5 interrupt request. **Access** rw	false
[6]	EVENT_CNT_6_INT (E6)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR6 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR6 event counter interrupt request is enabled. When written, enables the PMEVCNTR6 interrupt request. **Access** rw	false

Bits	Name	Description	Reset
[7]	EVENT_CNT_7_INT (E7)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR7 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR7 event counter interrupt request is enabled. When written, enables the PMEVCNTR7 interrupt request. **Access** rw	false
[30:8]	Reserved	-	-
[31]	CYCLE_CNT_INT (CC)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. **1** When read, means the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request. **Access** rw	false

Accessibility

Access to this register is restricted to mode rw.

3.3.4.7 PMINTCLR register

Interrupt enable clear register.

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

PMINTCLR is used with the PMINTSET register. It is implemented in hardware with the same underlying state as PMINTSET.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0018

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-44: PMINTCLR bit assignments

Table 3-52: PMU.PMINTCLR bit descriptions

Bits	Name	Description	Reset
[0]	EVENT_CNT_0_INT (E0)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR0 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR0 event counter interrupt request is enabled. When written, disables the PMEVCNTR0 interrupt request. **Access** rw	false
[1]	EVENT_CNT_1_INT (E1)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR1 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR1 event counter interrupt request is enabled. When written, disables the PMEVCNTR1 interrupt request. **Access** rw	false

Bits	Name	Description	Reset
[2]	EVENT_CNT_2_INT (E2)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR2 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR2 event counter interrupt request is enabled. When written, disables the PMEVCNTR2 interrupt request. **Access** rw	false
[3]	EVENT_CNT_3_INT (E3)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR3 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR3 event counter interrupt request is enabled. When written, disables the PMEVCNTR3 interrupt request. **Access** rw	false
[4]	EVENT_CNT_4_INT (E4)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR4 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR4 event counter interrupt request is enabled. When written, disables the PMEVCNTR4 interrupt request. **Access** rw	false
[5]	EVENT_CNT_5_INT (E5)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR5 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR5 event counter interrupt request is enabled. When written, disables the PMEVCNTR5 interrupt request. **Access** rw	false

Bits	Name	Description	Reset
[6]	EVENT_CNT_6_INT (E6)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR6 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR6 event counter interrupt request is enabled. When written, disables the PMEVCNTR6 interrupt request. **Access** rw	false
[7]	EVENT_CNT_7_INT (E7)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means that the PMEVCNTR7 event counter interrupt request is disabled. When written, has no effect. **1** When read, means that the PMEVCNTR7 event counter interrupt request is enabled. When written, disables the PMEVCNTR7 interrupt request. **Access** rw	false
[30:8]	Reserved	-	-
[31]	CYCLE_CNT_INT (CC)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. **1** When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request. **Access** rw	false

Accessibility

Access to this register is restricted to mode rw.

3.3.4.8 PMCCNTR register

Performance monitor cycle count register.

Holds the value of the dedicated cycle counter, PMCCNTR.

Configurations

This register is available in all configurations.

Attributes

Width

64-bit

Address offset

0x0020 - 0x0024

Type

rw

Reset value

0x0000000000000000

Bit descriptions

Figure 3-45: PMCCNTR bit assignments

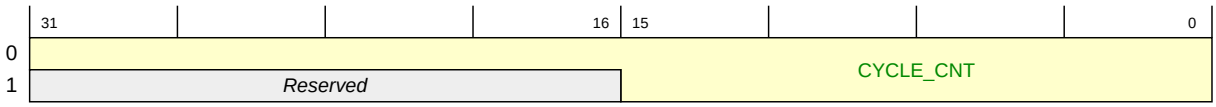


Table 3-53: PMU.PMCCNTR bit descriptions

Bits	Name	Description	Reset
[47:0]	CYCLE_CNT	This value is an unsigned integer. Its default value is 0x000000000000. **Access** rw	0x000000000000
[63:48]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.4.9 PMCCNTR_CFG register

Set start/stop event on the cycle counter.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0028

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-46: PMCCNTR_CFG bit assignments

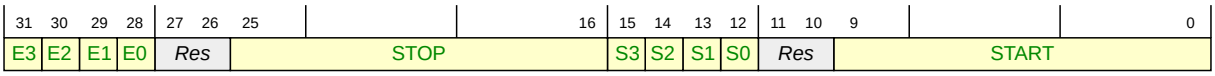


Table 3-54: PMU.PMCCNTR_CFG bit descriptions

Bits	Name	Description	Reset
[9:0]	CYCLE_CNT_CFG_START (START)	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table CYCLE_CNT_CFG_START	no_event
[11:10]	Reserved	-	-
[12]	S0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Bits	Name	Description	Reset
[13]	S1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[14]	S2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[15]	S3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[25:16]	CYCLE_CNT_CFG_STOP (STOP)	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table CYCLE_CNT_CFG_STOP	no_event
[27:26]	Reserved	-	-

Bits	Name	Description	Reset
[28]	E0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[29]	E1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[30]	E2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31]	E3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Table 3-55: Field CYCLE_CNT_CFG_STOP values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles

Value	Name	Description
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1

Value	Name	Description
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3

Value	Name	Description
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1
672..1023	Reserved	-

Table 3-56: Field CYCLE_CNT_CFG_START values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state

Value	Name	Description
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-

Value	Name	Description
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1

Value	Name	Description
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0

Value	Name	Description
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.4.10 PMCAXI_CHAN register

Set which AXI channel to monitor for latency measurements in PMU.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x002C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-47: PMCAXI_CHAN bit assignments

Table 3-57: PMU.PMCAXI_CHAN bit descriptions

Bits	Name	Description	Reset
[3:0]	CH_SEL (CH)	This value is an enumeration of type pmu_axi_channel_t. Its default value is rd_cmd. This field can contain the following values: **0x0** rd_cmd - Command stream read channel **0x1** rd_ifm - IFM read channel **0x2** rd_weights - Weights read channel **0x3** rd_scale_bias - Bias and scale read channel **0x4** rd_mem2mem - Memory to memory read channel **0x5** rd_ifm_stream - IFM to weights or elementwise **0x6** rd_mem2mem_idx - Memory to memory index channel **0x7** Reserved **0x8** wr_ofm - The OFM write channel **0x9** wr_mem2mem - The mem2mem write channel **0xA..0xF** Reserved	rd_cmd
[7:4]	Reserved	-	-

Bits	Name	Description	Reset
[8]	AXI_SEL (AS)	This value is an enumeration of type axi_port_t. Its default value is sram. This field can contain the following values: **0x0** sram - SRAM AXI port or ports selected **0x1** ext - EXT AXI port or ports selected	sram
[9]	Reserved	-	-
[10]	BW_CH_SEL_EN (BE)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** AXI bw events measured for all channels **1** AXI bw events measured for channel specified by CH_SEL **Access** rw	false
[31:11]	Reserved	-	-

Accessibility

Access to this register is restricted to mode rw.

3.3.4.11 PMCLUT register

Performance monitor control for lookup table.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0030

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-48: PMCLUT bit assignments

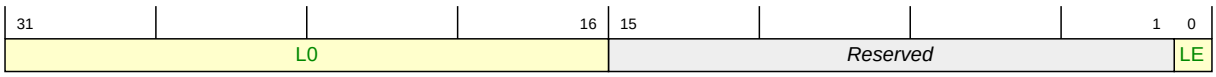


Table 3-58: PMU.PMCLUT bit descriptions

Bits	Name	Description	Reset
[0]	PCM_LUT_EN_0 (LE)	This value is a Boolean flag. Its default value is false. This field can contain the following values: **0** Lookup table is not enabled **1** Lookup table is enabled for event counter 0 **Access** rw	false
[15:1]	Reserved	-	-
[31:16]	PMC_LUT_0 (L0)	This register changes the event counted by event counter 0. If lookup table is enabled and event counter 0 counting is enabled then event counter 0 counts a compound event. The compound event is ((PMC_LUT_0 >> k) & 1)==1 where k = 8*event[3] + 4*event[2] + 2*event[1] + event[0] and event[n] is the event configured by PMEVTYPER[n]. Note that event[n] is included in k even if PMU counter n is counting is disabled. Note that if event n counts multiple times in a single cycle, event[n] is taken to be 1 for the calculation of this compound event. This value is an unsigned integer. Its default value is 0x0000. **Access** rw	0x0000

Accessibility

Access to this register is restricted to mode rw.

3.3.5 PMU_COUNTERS register summary

Performance monitoring counters.

Table 3-59: PMU_COUNTERS register summary

Offset	Name	Type	Reset	Width	Description
0x0000	PMEVCNTR0	rw	0x00000000	32-bit	Performance monitor event 0 count register
0x0004	PMEVCNTR1	rw	0x00000000	32-bit	Performance monitor event 1 count register
0x0008	PMEVCNTR2	rw	0x00000000	32-bit	Performance monitor event 2 count register
0x000C	PMEVCNTR3	rw	0x00000000	32-bit	Performance monitor event 3 count register

Offset	Name	Type	Reset	Width	Description
0x0010	PMEVCNTR4	rw	0x00000000	32-bit	Performance monitor event 4 count register
0x0014	PMEVCNTR5	rw	0x00000000	32-bit	Performance monitor event 5 count register
0x0018	PMEVCNTR6	rw	0x00000000	32-bit	Performance monitor event 6 count register
0x001C	PMEVCNTR7	rw	0x00000000	32-bit	Performance monitor event 7 count register
0x0020 - 0x007C	Reserved	-	-	96-byte	-
0x0080	PMEVTYPER0	rw	0x00000000	32-bit	Performance monitor event type register 0
0x0084	PMEVTYPER1	rw	0x00000000	32-bit	Performance monitor event type register 1
0x0088	PMEVTYPER2	rw	0x00000000	32-bit	Performance monitor event type register 2
0x008C	PMEVTYPER3	rw	0x00000000	32-bit	Performance monitor event type register 3
0x0090	PMEVTYPER4	rw	0x00000000	32-bit	Performance monitor event type register 4
0x0094	PMEVTYPER5	rw	0x00000000	32-bit	Performance monitor event type register 5
0x0098	PMEVTYPER6	rw	0x00000000	32-bit	Performance monitor event type register 6
0x009C	PMEVTYPER7	rw	0x00000000	32-bit	Performance monitor event type register 7
0x00A0 - 0x00FC	Reserved	-	-	96-byte	-

3.3.5.1 PMEVCNTR0 register

Performance monitor event 0 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0000

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-49: PMEVCNTR0 bit assignments

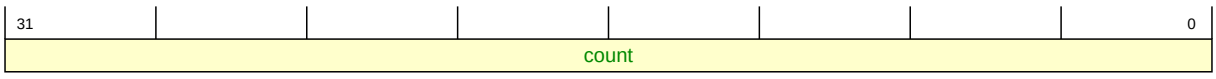


Table 3-60: PMU_COUNTERS.PMEVCNTR0 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.2 PMEVCNTR1 register

Performance monitor event 1 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0004

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-50: PMEVCNTR1 bit assignments

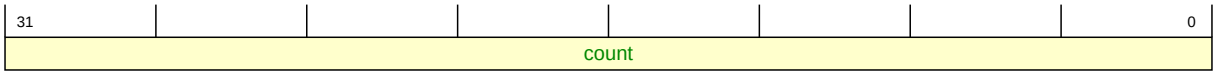


Table 3-61: PMU_COUNTERS.PMEVCNTR1 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.3 PMEVCNTR2 register

Performance monitor event 2 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0008

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-51: PMEVCNTR2 bit assignments

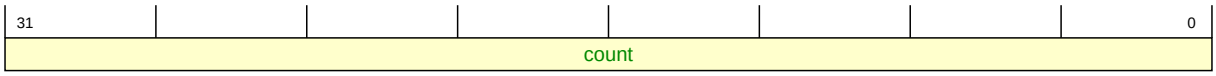


Table 3-62: PMU_COUNTERS.PMEVCNTR2 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.4 PMEVCNTR3 register

Performance monitor event 3 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x000C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-52: PMEVCNTR3 bit assignments

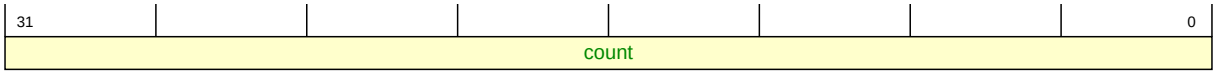


Table 3-63: PMU_COUNTERS.PMEVCNTR3 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.5 PMEVCNTR4 register

Performance monitor event 4 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0010

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-53: PMEVCNTR4 bit assignments

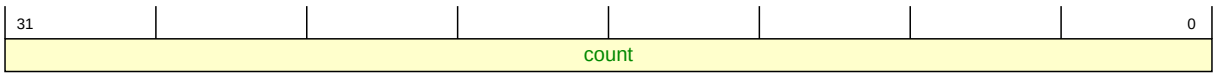


Table 3-64: PMU_COUNTERS.PMEVCNTR4 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.6 PMEVCNTR5 register

Performance monitor event 5 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0014

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-54: PMEVCNTR5 bit assignments

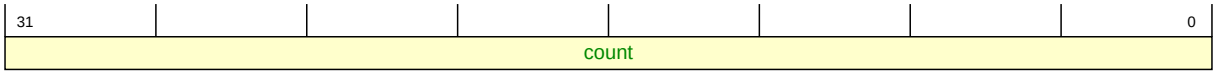


Table 3-65: PMU_COUNTERS.PMEVCNTR5 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.7 PMEVCNTR6 register

Performance monitor event 6 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0018

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-55: PMEVCNTR6 bit assignments

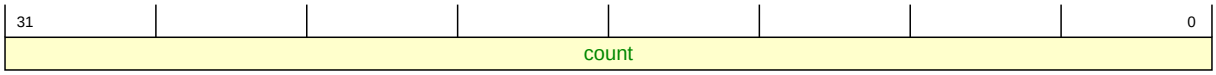


Table 3-66: PMU_COUNTERS.PMEVCNTR6 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.8 PMEVCNTR7 register

Performance monitor event 7 count register.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x001C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-56: PMEVCNTR7 bit assignments

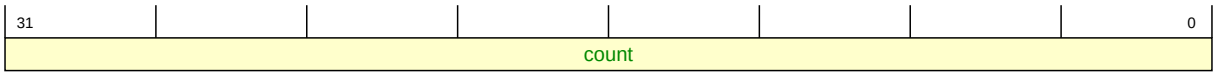


Table 3-67: PMU_COUNTERS.PMEVCNTR7 bit descriptions

Bits	Name	Description	Reset
[31:0]	count	This value is an unsigned integer. Its default value is 0.	0

Accessibility

Access to this register is restricted to mode rw.

3.3.5.9 PMEVTYPER0 register

Performance monitor event type register 0.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0080

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-57: PMEVTYPER0 bit assignments

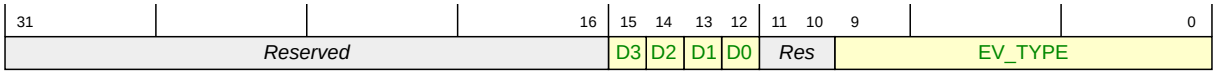


Table 3-68: PMU_COUNTERS.PMEVTYPER0 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Bits	Name	Description	Reset
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-69: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports

Value	Name	Description
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports

Value	Name	Description
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2

Value	Name	Description
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1

Value	Name	Description
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.5.10 PMEVTYPER1 register

Performance monitor event type register 1.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0084

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-58: PMEVTYPER1 bit assignments

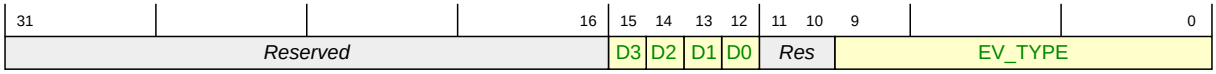


Table 3-70: PMU_COUNTERS.PMEVTYPER1 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Bits	Name	Description	Reset
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-71: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-

Value	Name	Description
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0

Value	Name	Description
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3

Value	Name	Description
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1

Value	Name	Description
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.5.11 PMEVTYPER2 register

Performance monitor event type register 2.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0088

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-59: PMEVTYPER2 bit assignments

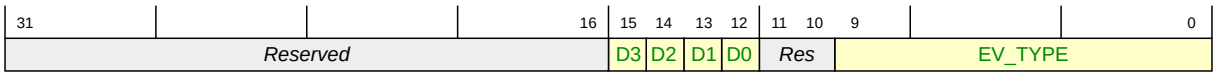


Table 3-72: PMU_COUNTERS.PMEVTYPER2 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-

Bits	Name	Description	Reset
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-73: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-

Value	Name	Description
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected

Value	Name	Description
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-

Value	Name	Description
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0

Value	Name	Description
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.5.12 PMEVTYPER3 register

Performance monitor event type register 3.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x008C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-60: PMEVTYPER3 bit assignments

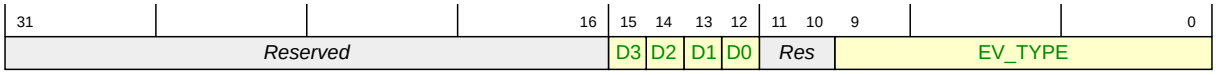


Table 3-74: PMU_COUNTERS.PMEVTYPER3 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Bits	Name	Description	Reset
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-75: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports

Value	Name	Description
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports

Value	Name	Description
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2

Value	Name	Description
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1

Value	Name	Description
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.5.13 PMEVTYPER4 register

Performance monitor event type register 4.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0090

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-61: PMEVTYPER4 bit assignments

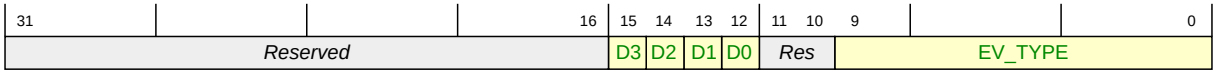


Table 3-76: PMU_COUNTERS.PMEVTYPER4 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Bits	Name	Description	Reset
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-77: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-

Value	Name	Description
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0

Value	Name	Description
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3

Value	Name	Description
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1

Value	Name	Description
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.5.14 PMEVTYPER5 register

Performance monitor event type register 5.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0094

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-62: PMEVTYPER5 bit assignments

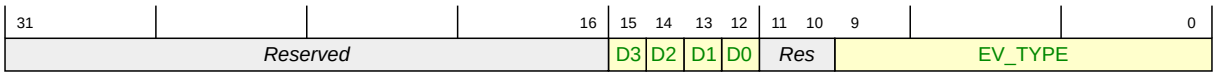


Table 3-78: PMU_COUNTERS.PMEVTYPER5 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-

Bits	Name	Description	Reset
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-79: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-

Value	Name	Description
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected

Value	Name	Description
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-

Value	Name	Description
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0

Value	Name	Description
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.5.15 PMEVTYPER6 register

Performance monitor event type register 6.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x0098

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-63: PMEVTYPER6 bit assignments

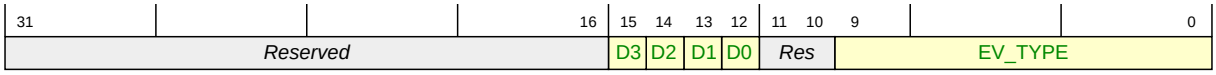


Table 3-80: PMU_COUNTERS.PMEVTYPER6 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Bits	Name	Description	Reset
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-81: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports

Value	Name	Description
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports

Value	Name	Description
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2

Value	Name	Description
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1

Value	Name	Description
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.3.5.16 PMEVTYPER7 register

Performance monitor event type register 7.

Configurations

This register is available in all configurations.

Attributes

Width

32-bit

Address offset

0x009C

Type

rw

Reset value

0x00000000

Bit descriptions

Figure 3-64: PMEVTYPER7 bit assignments

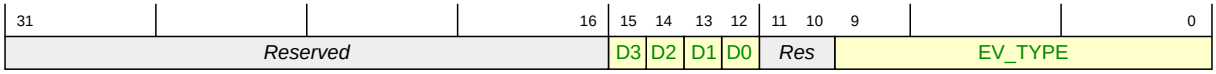


Table 3-82: PMU_COUNTERS.PMEVTYPER7 bit descriptions

Bits	Name	Description	Reset
[9:0]	EV_TYPE	This value is an enumeration of type pmu_event_t. Its default value is no_event. This field can contain the values shown in table EV_TYPE **Access** rw	no_event
[11:10]	Reserved	-	-
[12]	D0	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[13]	D1	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[14]	D2	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable

Bits	Name	Description	Reset
[15]	D3	This value is an enumeration of type pmu_port_disable_t. Its default value is enable. This field can contain the following values: **0x0** enable - PMU counting enabled for this AXI port **0x1** disable - PMU counting disabled for this AXI port **Access** rw	enable
[31:16]	Reserved	-	-

Table 3-83: Field EV_TYPE values

Value	Name	Description
0	no_event	No event. Event never occurs. No event counted
1..16	Reserved	-
17	cycle	Elapsed cycle. Event occurs every cycle. Used for counter debug
18..31	Reserved	-
32	npu_idle	NPU in stopped state
33..34	Reserved	-
35	npu_active	NPU in running state
36..47	Reserved	-
48	mac_active	MAC is doing block traversal. Valid blk_cmd and not stalled
49..50	Reserved	-
51	mac_dpu_active	At least one dot product unit is active, meaning the unit has at least one valid weight and activation pair
52..63	Reserved	-
64	ao_active	AO is doing block traversal of accumulation or chaining buffer. Valid blk_cmd and not stalled
65..79	Reserved	-
80	wd_active	WD is decoding weight stream. Valid blk_cmd and not stalled
81	wd_stalled	WD stalled on lack of weight stream data or lack of free WD buffer space
82..127	Reserved	-
128	sram_rd_trans_accepted	Read transfer (arready & arvalid) sum over SRAM AXI ports
129	sram_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over SRAM AXI ports
130	sram_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over SRAM AXI ports
131	sram_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over SRAM AXI ports
132	sram_wr_trans_accepted	Write transfers (awready & awvalid) sum over SRAM AXI ports
133..134	Reserved	-
135	sram_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over SRAM AXI ports
136	sram_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over SRAM AXI ports
137	sram_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over SRAM AXI ports
138..139	Reserved	-

Value	Name	Description
140	sram_enabled_cycles	Memory clock cycle (aclken_input), sum over SRAM AXI ports
141	Reserved	-
142	sram_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over SRAM AXI ports
143	sram_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over SRAM AXI ports
144..159	Reserved	-
160	axi_latency_any	Any latency - measures the total number of transactions for the specified channel
161	axi_latency_32	Latency was >= 32 cycles
162	axi_latency_64	Latency was >= 64 cycles
163	axi_latency_128	Latency was >= 128 cycles
164	axi_latency_256	Latency was >= 256 cycles
165	axi_latency_512	Latency was >= 512 cycles
166	axi_latency_1024	Latency was >= 1024 cycles
167..175	Reserved	-
176	ecc_dma	ECC event in DMA controller RAM. This event is either corrected or uncorrected
177	ecc_mac_ib	ECC event in MAC input buffer RAM. This event is either corrected or uncorrected
178	ecc_mac_ab	ECC event in MAC AB RAM. This event is either corrected or uncorrected
179	ecc_ao_cb	ECC event in AO CB RAM. This event is either corrected or uncorrected
180	ecc_ao_ob	ECC event in AO Output Buffer (OB) RAM. This event is either corrected or uncorrected
181	ecc_ao_lut	ECC event in AO lookup table RAM. This event is either corrected or uncorrected
182..383	Reserved	-
384	ext_rd_trans_accepted	Read transfer (arready & arvalid) sum over EXT AXI ports
385	ext_rd_trans_completed	Read complete (rready & rvalid & rlast) sum over EXT AXI ports
386	ext_rd_data_beat_received	Read bandwidth (rready & rvalid) sum over EXT AXI ports
387	ext_rd_tran_req_stalled	Read stalls (arvalid & ~arready) sum over EXT AXI ports
388	ext_wr_trans_accepted	Write transfers (awready & awvalid) sum over EXT AXI ports
389..390	Reserved	-
391	ext_wr_data_beat_written	Write bandwidth (wvalid & wready) sum over EXT AXI ports
392	ext_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) sum over EXT AXI ports
393	ext_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) sum over EXT AXI ports
394..395	Reserved	-
396	ext_enabled_cycles	Memory clock cycle (aclken_input), sum over EXT AXI ports
397	Reserved	-
398	ext_rd_stall_limit	Read stalls due to max outstanding transactions limit, sum over EXT AXI ports
399	ext_wr_stall_limit	Write stalls due to max outstanding transactions limit, sum over EXT AXI ports
400..511	Reserved	-
512	sram0_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 0
513	sram0_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 0
514	sram0_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 0
515	sram0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 0
516	sram0_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 0

Value	Name	Description
517..518	Reserved	-
519	sram0_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 0
520	sram0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 0
521	sram0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 0
522..523	Reserved	-
524	sram0_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 0
525	Reserved	-
526	sram0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 0
527	sram0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 0
528	sram1_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 1
529	sram1_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 1
530	sram1_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 1
531	sram1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 1
532	sram1_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 1
533..534	Reserved	-
535	sram1_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 1
536	sram1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 1
537	sram1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 1
538..539	Reserved	-
540	sram1_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 1
541	Reserved	-
542	sram1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 1
543	sram1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 1
544	sram2_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 2
545	sram2_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 2
546	sram2_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 2
547	sram2_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 2
548	sram2_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 2
549..550	Reserved	-
551	sram2_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 2
552	sram2_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 2
553	sram2_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 2
554..555	Reserved	-
556	sram2_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 2
557	Reserved	-
558	sram2_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 2
559	sram2_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 2
560	sram3_rd_trans_accepted	Read transfer (arready & arvalid) for SRAM AXI port 3
561	sram3_rd_trans_completed	Read complete (rready & rvalid & rlast) for SRAM AXI port 3
562	sram3_rd_data_beat_received	Read bandwidth (rready & rvalid) for SRAM AXI port 3

Value	Name	Description
563	sram3_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for SRAM AXI port 3
564	sram3_wr_trans_accepted	Write transfers (awready & awvalid) for SRAM AXI port 3
565..566	Reserved	-
567	sram3_wr_data_beat_written	Write bandwidth (wvalid & wready) for SRAM AXI port 3
568	sram3_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for SRAM AXI port 3
569	sram3_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for SRAM AXI port 3
570..571	Reserved	-
572	sram3_enabled_cycles	Memory clock cycle (aclken_input), for SRAM AXI port 3
573	Reserved	-
574	sram3_rd_stall_limit	Read stalls due to max outstanding transactions limit, for SRAM AXI port 3
575	sram3_wr_stall_limit	Write stalls due to max outstanding transactions limit, for SRAM AXI port 3
576..639	Reserved	-
640	ext0_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 0
641	ext0_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 0
642	ext0_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 0
643	ext0_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 0
644	ext0_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 0
645..646	Reserved	-
647	ext0_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 0
648	ext0_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 0
649	ext0_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 0
650..651	Reserved	-
652	ext0_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 0
653	Reserved	-
654	ext0_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 0
655	ext0_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 0
656	ext1_rd_trans_accepted	Read transfer (arready & arvalid) for EXT AXI port 1
657	ext1_rd_trans_completed	Read complete (rready & rvalid & rlast) for EXT AXI port 1
658	ext1_rd_data_beat_received	Read bandwidth (rready & rvalid) for EXT AXI port 1
659	ext1_rd_tran_req_stalled	Read stalls (arvalid & ~arready) for EXT AXI port 1
660	ext1_wr_trans_accepted	Write transfers (awready & awvalid) for EXT AXI port 1
661..662	Reserved	-
663	ext1_wr_data_beat_written	Write bandwidth (wvalid & wready) for EXT AXI port 1
664	ext1_wr_tran_req_stalled	write transfer stalled by memory (awvalid & ~awready) for EXT AXI port 1
665	ext1_wr_data_beat_stalled	Write beat stalled by memory (wvalid & ~wready) for EXT AXI port 1
666..667	Reserved	-
668	ext1_enabled_cycles	Memory clock cycle (aclken_input), for EXT AXI port 1
669	Reserved	-
670	ext1_rd_stall_limit	Read stalls due to max outstanding transactions limit, for EXT AXI port 1
671	ext1_wr_stall_limit	Write stalls due to max outstanding transactions limit, for EXT AXI port 1

Value	Name	Description
672..1023	Reserved	-

Accessibility

Access to this register is restricted to mode rw.

3.4 Access states for NPU control

This section provides a description of the access types.

This section gives a description of the different access states this document refers to.

Table 3-84: Access states for NPU control

Name	Description
ro	Read-only
rw	Read/write
wo	Write-only

3.4.1 ro access state

Read-only.

This access type is used for read-only registers.

Table 3-85: Permissions for access state ro

Requester	Read	Write
permitted_requester	allow	ignore
non_permitted_requester	zero	ignore

3.4.2 rw access state

Read/write.

This access type is used for read/write registers.

Table 3-86: Permissions for access state rw

Requester	Read	Write
permitted_requester	allow	allow
non_permitted_requester	zero	ignore

3.4.3 wo access state

Write-only.

This access type is used for write-only registers.

Table 3-87: Permissions for access state wo

Requester	Read	Write
permitted_requester	zero	allow
non_permitted_requester	zero	ignore

3.5 Command stream architecture

This section gives an overview of the format of the NPU command stream.

3.5.1 Command stream

The Ethos-U instruction stream is organized in a conventional manner as a series of commands to be executed.

Each job of work consists of a single command stream that executes on the NPU and signals when the job is complete.

The command stream is relocatable, meaning that the stream can be compiled offline in advance. The actual address the NPU accesses can be set at run-time using the region mechanism.

3.5.2 NPU Execution states

The NPU can be in one of two Execution states: STOPPED or RUNNING.

The current state of the NPU can be read from the STATUS register field state.

- In STOPPED state, the NPU does not execute the command stream.
- In RUNNING state, the NPU executes the command stream until it reaches a condition that causes it to stop. For more information, see the following on transitioning from STOPPED to RUNNING.

Transition from STOPPED to RUNNING state occurs when the user writes to the CMD register field transition_to_running_state.

Transition from RUNNING state to STOPPED state occurs under any of the following conditions:

- The NPU is reset.
- The NPU executes a NPU_OP_STOP command in the Command queue and all previous commands are complete.

- The NPU runs out of commands to execute. In other words, the NPU reaches the QSIZE limit and all commands are complete.
- The NPU encounters an error. The error is signaled in the STATUS register.

Transition to stopped state, other than on reset, raises an IRQ to the host when the transition is complete. The stop transition is complete only when all commands in the stream before the stop point are complete. The interface state is signaled as RUNNING in the status register until the transition to STOPPED is complete.

3.5.3 Command stream encoding

Each Ethos-U command consists of an integer number of 32-bit words, starting at a 32-bit aligned address.

The first 32-bit word determines the length of the command which can be between one or two words (four bytes or eight bytes).

There are two forms of commands and they are as follows:

- CMD0 consists of a 32-bit header word with no data following
- CMD1 consists of a 32-bit header word and a 32-bit payload word (payload_1)

The command header words are encoded as follows:

Table 3-88: Command stream formats

Command type	b[31:16]	b[15:14]	b[13:10]	b[9:0]	Following data
CMD0	payload_0	00	0000	opcode	None
CMD1	payload_0	01	0000	opcode	Single 32-bit word payload_1

All other command encoding values are reserved. Their behavior is UNPREDICTABLE.

3.5.4 Unpredictable behavior

In this specification, the term UNPREDICTABLE means behavior that cannot be relied on.

UNPREDICTABLE behavior can lead to unexpected output data from the NPU or the NPU accessing the wrong addresses but the following rules must always be met:

- UNPREDICTABLE behavior must not introduce security holes. Specifically, UNPREDICTABLE behavior must not change the state of the RESET or PROT registers.
- All AXI accesses must meet the AXI protocol and be made with the protection level set by the PROT register.
- UNPREDICTABLE behavior must not cause the NPU or its interfaces to hang. The NPU must respond to the APB interface and must always be able to be reset by writing to the RESET register.

You must construct the command stream and input tensor data so that the NPU does not perform operations that are UNPREDICTABLE as defined by this architecture specification. If this condition is not met, the result of the inference is UNPREDICTABLE and cannot be relied on.

3.5.5 Command types

There are two types of NPU commands.

There are NPU commands that set state and NPU commands that start operations.

The following is a description of the two types of commands:

- Commands that set state. The names of these commands have the prefix NPU_SET. They configure state values or registers for use by following operation commands. When set, the value can be used in any of the following operations, until a new NPU_SET explicitly writes the value or the value implicitly changes as a side effect of chaining. For details of this behavior, see the State Setting Commands section.
- Commands that start operations. The names of these commands have the prefix NPU_OP. These commands issue a new operation to the NPU based on the configured state. The command captures the value of the state registers at the point of issue. Therefore, NPU_SET commands following an NPU_OP command do not affect the behavior of the previous NPU_OP, even if the operation is not yet complete. If a future operation requires the results of the current operation, the operations completion rules must be considered. For details of this behavior, see section Command issue and completion

3.5.5.1 State Setting Commands

This section describes the rules on the order of state settings commands in the command stream.

State settings commands have the prefix NPU_SET. State setting commands must appear in the command stream before operation commands that use them. Let OP2 be a kernel operation in the command stream as defined in Command issue and completion. OP2 is said to be part of a chain if any of the IFM, IFM2, or OFM access a chaining buffer.

If OP2 is part of a chain then define OP1 to be the preceding operation in the command stream, or start of the command stream if there is none. The following rules apply:

- If OP2 uses IFM as an input then the following instructions must be present between OP1 and OP2. If OP2 does not use IFM as an input they must not be present in that interval:
  - SET_IFM_PRECISION
  - SET_IFM_ZERO_POINT

  Furthermore if OP2 uses IFM and any of the following commands then they must occur between SET_IFM_PRECISION and OP2:
  - SET_IFM_BASE0, SET_IFM_BASE1, SET_IFM_BASE2, SET_IFM_BASE3
  - SET_IFM_WIDTH0_M1, SET_IFM_HEIGHT0_M1, SET_IFM_HEIGHT1_M1

    - SET_IFM_STRIDE_X, SET_IFM_STRIDE_Y, SET_IFM_STRIDE_C
    - SET_IFM_REGION
- If OP2 uses IFM2 as an input then the following instructions must be present between OP1 and OP2. If OP2 does not use IFM2 as an input they must not be present in that interval:
    - SET_IFM2_PRECISION
    - SET_IFM2_ZERO_POINT

    Furthermore if OP2 uses IFM2 and any of the following commands then they must occur between SET_IFM2_PRECISION and OP2:
    - SET_IFM2_BASE0, SET_IFM2_BASE1, SET_IFM2_BASE2, SET_IFM2_BASE3
    - SET_IFM2_WIDTH0_M1, SET_IFM2_HEIGHT0_M1, SET_IFM2_HEIGHT1_M1
    - SET_IFM2_STRIDE_X, SET_IFM2_STRIDE_Y, SET_IFM2_STRIDE_C
    - SET_IFM2_REGION

If OP2 is not part of a chain then define OP1 to be the preceding operation in the command stream that is part of a chain, or the start of the command stream if there is none. The following rules apply:

- If OP2 uses IFM as an input then the following instructions must be present between OP1 and OP2.
    - SET_IFM_PRECISION
    - SET_IFM_ZERO_POINT

    Furthermore if OP2 uses IFM and any of the following commands then they must occur between OP1 and OP2:
    - SET_IFM_BASE0, SET_IFM_BASE1, SET_IFM_BASE2, SET_IFM_BASE3
    - SET_IFM_WIDTH0_M1, SET_IFM_HEIGHT0_M1, SET_IFM_HEIGHT1_M1
    - SET_IFM_STRIDE_X, SET_IFM_STRIDE_Y, SET_IFM_STRIDE_C
    - SET_IFM_REGION
- If OP2 uses IFM2 as an input then the following instructions must be present between OP1 and OP2.
    - SET_IFM2_PRECISION
    - SET_IFM2_ZERO_POINT

    Furthermore if OP2 uses IFM2 and any of the following commands then they must occur between OP1 and OP2:
    - SET_IFM2_BASE0, SET_IFM2_BASE1, SET_IFM2_BASE2, SET_IFM2_BASE3
    - SET_IFM2_WIDTH0_M1, SET_IFM2_HEIGHT0_M1, SET_IFM2_HEIGHT1_M1
    - SET_IFM2_STRIDE_X, SET_IFM2_STRIDE_Y, SET_IFM2_STRIDE_C
    - SET_IFM2_REGION

3.5.5.2 Command issue and completion

This section describes when commands can issue and when a command is complete.

A command is blocking if following commands cannot be issued until the command is completed.

The following table lists, for each type of command, when the command is blocking and when the command is determined to be completed. In this table, NPU_OP_<KERNEL> is any operation where KERNEL is one of CONV, DEPTHWISE, ELEMENTWISE, POOL , RESIZE .

Table 3-89: Command completion

Command type	Blocking	Completion criteria
NPU_OP_STOP	Yes	NPU has entered STOPPED state and raised an IRQ
NPU_OP_IRQ	No	NPU had raised an IRQ
NPU_OP_<KERNEL>	No	NPU has calculated the result and the memory system has accepted all result writes.
NPU_OP_DMA_START	Yes	NPU accepts the DMA instruction into the internal DMA queue (there is space in the queue). The DMA transfer is not required to have completed or started for this instruction to be considered complete.
NPU_OP_DMA_WAIT	Yes	The DMA wait condition has been satisfied. This command acts as a barrier until the wait condition is true.
NPU_OP_KERNEL_WAIT	Yes	The KERNEL wait condition has been satisfied. This command acts as a barrier until the wait condition is true.
NPU_OP_PMU_MASK	Yes	NPU has updated the PMU mask.
NPU_OP_BRANCH	Yes when NPU_OP_BRANCH.branch_cond = rf_true	NPU has computed the address of the next command the NPU executes.
NPU_SET_<REGISTER>	Yes	NPU has updated the register state.

3.6 Pseudocode

This section defines pseudocode used by command stream operations.

3.6.1 Unpredictable pseudocode

Operations are not always valid or predictable.

The pseudocode defines the functionality of each operation. If an operation is only valid or predictable for a certain range of values, an `assert()` statement is added to define the valid range. If an assertion fails, it follows that the behavior is **UNPREDICTABLE** as defined in Unpredictable behavior.

assert

```
void assert(bool condition) {
    if (!condition) {
```

```
        UNPREDICTABLE();    // The NPU behavior is unpredictable
    }
}
```

3.6.2 Data types

There are several data types used within the pseudocode.

The following data types are used within the pseudocode:

- integer - unbounded integer with a used range that is calculated from the context
- uint8_t - unsigned eight-bit integer in the range 0 to 255
- int8_t - signed eight-bit integer in the range -128 to +127
- int16_t - signed 16-bit integer in the range -32768 to +32767
- int32_t - signed 32-bit integer in the range (-1<<31) to (1<<31)-1
- int48_t - signed 48-bit integer in the range (-1<<47) to (1<<47)-1
- int64_t - signed 64-bit integer in the range (-1<<63) to (1<<63)-1
- int - signed 32-bit integer in the range (-1<<31) to (1<<31)-1
- addr_t - a 40-bit address

align_up

Return value aligned up to the next multiple of alignment.

```
int align_up(int value, int alignment) {
  int r = value % alignment;
  assert(r >= 0);
  return value + ((r != 0) ? (alignment - r) : 0);
}
```

number_of_bits

The following code gives the number of bits used by a type.

```
int number_of_bits(activation_precision_t precision) {
    int bits = 0;
    switch (precision) {
        case b8 : bits=8;  break;
        case b16: bits=16; break;
        case b32: bits=32; break;
        case b64: bits=64; break;
    }
    assert(bits!=0);
    return bits;
}
```

zero_extend

The following code zero extends from a given bit position.

```
int64_t zero_extend(int64_t value, int n) {
```

```
    int64_t range = (int64_t)1 << n;
    value = value & (range-1);   // lower n bits
    return value;
}
```

sign_extend

The following code sign extends from a given bit position.

```
int64_t sign_extend(int64_t value, int n) {
    int64_t range = (int64_t)1 << n;
    value = value & (range-1);   // lower n bits
    if (value >= (range/2)) {
        value = value - range;      // sign extend negative values
    }
    return value;
}
```

truncate

The following code truncates the result to a given number of bits.

```
 int64_t truncate(int64_t value, int n, activation_type_t type) {
    if (type == unsigned) {
        value = zero_extend(value, n);
    } else {
        value = sign_extend(value, n);
    }
    return value;
}
```

3.6.3 Memory access functions

This section defines functions for accessing or moving memory.

Memory addresses generated by the command stream are offsets within a given region. At run-time, the driver specifies the address of the region. The following functions illustrate how regions are accessed.

memory_read

```
<type> memory_read<type>(int region, addr_t addr) {
    assert(0 <= region && region < 8);
    addr += (addr_t)BASEP[region];
    return read_little_endian<type>(addr);
}
```

memory_write

```
void memory_write<type>(int region, addr_t addr, <type> value) {
    if (region==-1) {
        // Write to the internal lookup table
        addr += (addr_t)(&lookup_table_storage);
    } else {
        assert(0 <= region && region < 8);
        addr += (addr_t)BASEP[region];
    }
    return write_little_endian<type>(addr, value);
```

```
}
```

copy_bytes

The following function is the copy function used by memory to memory DMA.

```
void copy_bytes(addr_t dst_addr, int dst_region, addr_t src_addr, int src_region,
 addr_t length)
{
    if (region == -1) { // internal memory
        assert((dst_addr & 15)==0);     // 16 byte aligned address required
        assert((length   & 15)==0);     // 16 byte aligned length required
    }
    for (int i=0; i < length; i++) {
        int8_t v = memory_read<int8_t>(src_region, src_addr + i);
        memory_write<int8_t>(dest_region, dst_addr + i);
    }
}
```

3.6.4 Chaining functions

This section defines functions for chaining command stream operations.

The following global state is held across chaining functions:

```
// Chaining state:
#define NUM_CHAINING_FIFO 3
bool    g_fifo_valid[NUM_CHAINING_FIFO];
integer g_fifo_head[NUM_CHAINING_FIFO];
integer g_fifo_tail[NUM_CHAINING_FIFO];
FIFO    *g_fifo_data[NUM_CHAINING_FIFO];
int     g_num_ifm=0;
int     g_chain_length=0;
```

microblock_depth

The following code returns the microblock depth in output channels depending on the NPU configuration CONFIG.macs_per_cc.

```
int microblock_depth(void) {
    microblock_t microblock = NPU_SET_ACC_FORMAT.microblock;
    int depth;
    switch (microblock) {
        case u1x1: depth = macs_per_cc/8; break;
        case u1x2: depth = macs_per_cc/16; break;
        case u1x4: depth = macs_per_cc/32; break;
        case u2x2: depth = macs_per_cc/32; break;
        case u2x4: depth = macs_per_cc/64; break;
        case u4x4: depth = macs_per_cc/128; break;
    }
    return depth;
}
```

add_operation_to_chain

The following code adds an operation to a chain. This function is called once for each kernel operation in the command stream.

```
void add_operation_to_chain(
    bool use_ifm,  // true if the operation uses ifm input
    bool use_ifm2, // true if the operation uses ifm2 input
    bool append    // true if the operation can be appended to a chain
)
{
    assert(g_chain_length == 0 || append);
    if (g_chain_length == 0) {
        // Start of a chain. This chain must be independent of previous chains.
        g_num_ifm = 0;
        for (int i = 0; i < NUM_CHAINING_FIFO; i++) {
            g_fifo_valid[i] = false;
        }
        ();
    }
    g_chain_length++;
    assert(g_chain_length <= 4);
    if (use_ifm) {
        if (NPU_SET_IFM_PRECISION.activation_storage == chained) {
            assert(NPU_SET_IFM_REGION.region < NUM_CHAINING_FIFO);
            assert(g_fifo_valid[NPU_SET_IFM_REGION.region]);
        } else {
            g_num_ifm++;
        }
    }
    if (use_ifm2) {
        if (NPU_SET_IFM2_PRECISION.activation_storage == chained) {
            assert(NPU_SET_IFM2_REGION.region < NUM_CHAINING_FIFO);
            assert(g_fifo_valid[NPU_SET_IFM2_REGION.region]);
            if (use_ifm && NPU_SET_IFM_PRECISION.activation_storage == chained) {
                // Cannot read two inputs in parallel from the same chaining buffer
                assert(NPU_SET_IFM_REGION.region != NPU_SET_IFM2_REGION.region);
            }
        } else {
            g_num_ifm++;
        }
    }
    assert(g_num_ifm <= 3);
    if (NPU_SET_OFM_PRECISION.activation_storage == chained) {
        assert(NPU_SET_OFM_REGION.region < NUM_CHAINING_FIFO);
        g_fifo_valid[NPU_SET_OFM_REGION.region] = true;
    } else {
        // This is the end of the chain
        g_chain_length = 0;
    }
}
```

write_chaining_fifo

The following code writes to a chaining FIFO.

```
void write_chaining_fifo(int fifo_number, int64_t value) {
    // Note that FIFO_SIZE and order of filling/draining FIFO is
    // implementation dependent but functionally must match an
    // unbounded FIFO.
    assert(0 <= fifo_number && fifo_number <= 2);
    head = g_fifo_head[fifo_number];
    tail = g_fifo_tail[fifo_number];
    data = g_fifo_data[fifo_number];
    data[head % FIFO_SIZE] = value;
    head = head + 1;
```

```
    assert( head - tail <= FIFO_SIZE );
    g_fifo_head[fifo_number] = head;
}
```

read_chaining_fifo

The following code reads from a chaining FIFO.

```
int64_t read_chaining_fifo(int fifo_number) {
    // Note that FIFO_SIZE and order of filling/draining FIFO is
    // implementation dependent but functionally must match an
    // unbounded FIFO.
    assert(0 <= fifo_number && fifo_number <= 2);
    head = g_fifo_head[fifo_number];
    tail = g_fifo_tail[fifo_number];
    data = g_fifo_data[fifo_number];
    value = data[tail % FIFO_SIZE];
    tail = tail + 1;
    assert( head >= tail );
    g_fifo_tail[fifo_number] = tail;
    return value;
}
```

check_broadcast_permitted

The following code checks if broadcast is permitted. Broadcast is not permitted from a chaining FIFO.

```
void check_broadcast_permitted(broadcast_mode_t broadcast_mode, activation_storage_t
 activation_storage) {
    if (broadcast_mode != none) {
        assert(activation_storage != chained);
    }
}
```

3.6.5 Tensor access functions

This section defines functions for reading or writing a tensor element at a given tensor location.

element_address_in_region

The following code calculates the byte address of an element within a given address region.

```
addr_t element_address_in_region(
    addr_t base[4],             // base address of tensor tiles in the region
    addr_t stride_y,            // y stride in bytes
    addr_t stride_x,            // x stride in bytes
    addr_t stride_c,            // c stride in bytes
    addr_t width0,              // tile 0 width
    addr_t height[2],           // tile 0 and 1 height
    activation_storage_t   storage,     // storage tiling mode
    activation_precision_t precision,   // number of bits of precision
    activation_format_t    format,      // NHWC or brick
    int y,                      // y co-ordinate
    int x,                      // x co-orindate
    int c                       // channel co-ordinate
)
{
    int tile = 0;
    assert(storage != none);
```

```
    switch (storage) {
        case tile2x2:
            if (x >= width0)         { x -= width0;       tile += 1; }
            if (y >= height[tile])   { y -= height[tile]; tile += 2; }
            break;
        case tile3x1:
            if (y >= height[1])      { y -= height[1];    tile = 2; }
            else if (y >= height[0]) { y -= height[0];    tile = 1; }
            break;
    }
    addr_t addr = base[tile] + y*stride_y;
    element_size = number_of_bits(precision)/8;
    switch (format) {
        case nhwc:
            assert((base[tile] % element_size)==0);
            assert((stride_x   % element_size)==0);
            assert((stride_y   % element_size)==0);
            addr += x*stride_x + c;
            break;
        case nhcwb16:
            assert((base[tile] % 16)==0);
            assert((stride_c   % (16*element_size))==0);
            assert((stride_y   % (16*element_size))==0);
            addr += (c>>4)*stride_c + (16*x + (c&15))*element_size;
            break;
    }
    return addr;
}
```

read_ifm

The following code does an IFM tensor read, including zero point subtract.

```
int32_t read_ifm(int y, int x, int c) {
    activation_precision_t precision = NPU_SET_IFM_PRECISION.activation_precision;
    activation_type_t      type      = NPU_SET_IFM_PRECISION.activation_type;
    activation_format_t    format    = NPU_SET_IFM_PRECISION.activation_format;
    activation_storage_t   storage   = NPU_SET_IFM_PRECISION.activation_storage;
    int                    region    = NPU_SET_IFM_REGION.region;
    ifm_upscale_mode_t     upscale   = NPU_SET_IFM_UPSCALE.mode;

    // check for 2x2 upscale of this IFM
    if (upscale != none) {
        assert(read_kernel_x_stride()==1);
        assert(read_kernel_y_stride()==1);
        assert(storage != activation_storage_chained);
        if (upscale == zeros && ((x & 1)==1 || (y & 1)==1)) {
            return 0;
        }
        x = x >> 1;
        y = y >> 1;
    }

    int32_t value;
    if (storage == chained) {
        value = read_chaining_fifo(region);
    } else {
        addr_t base[4]   = { IFM_BASE0, IFM_BASE1, IFM_BASE2, IFM_BASE3}
        addr_t height[2] = { IFM_HEIGHT0, IFM_HEIGHT1 }
        addr_t addr = element_address_in_region(
            base, IFM_STRIDE_Y, IFM_STRIDE_X, IFM_STRIDE_C,
            IFM_WIDTH0, height, storage, precision, format, y, x, c);
        value = memory_read(IFM_REGION, addr, precision, type);
    }

    int32_t zero_point;
    if (type == unsigned) {
        zero_point = (uint16_t)NPU_SET_IFM_ZERO_POINT.zero_point;
```

```
        assert((precision == b8 && zero_point < 256) ||
               (precision == b16 && zero_point == 32768) ||
               zero_point == 0);
    } else {
        zero_point = (int16_t)NPU_SET_IFM_ZERO_POINT.zero_point;
        assert((precision == b8 && -128 <= zero_point && zero_point < 128) ||
               zero_point == 0);
    }
    return value - zero_point;
}
```

read_ifm2

The following code does an IFM2 tensor read including zero point subtract.

```
integer read_ifm2(int y, int x, int c, bool read_accumulator=false) {
    activation_precision_t precision;
    activation_type_t      type;
    if (read_accumulator) {
        if (NPU_SET_ACC_FORMAT.acc_format == i48) {
            precision == b64;
        } else {
            precision == b32;
        }
        type      = signed;
        assert(NPU_SET_IFM2_PRECISION.activation_precision == precision);
        assert(NPU_SET_IFM2_PRECISION.activation_type == type);
    } else {
        precision = NPU_SET_IFM2_PRECISION.activation_precision;
        type      = NPU_SET_IFM2_PRECISION.activation_type;
        assert(NPU_SET_IFM_PRECISION.activation_precision == precision);
        assert(NPU_SET_IFM_PRECISION.activation_type == type);
    }
    activation_format_t    format    = NPU_SET_IFM2_PRECISION.activation_format;
    activation_storage_t   storage   = NPU_SET_IFM2_PRECISION.activation_storage;
    int                    region    = NPU_SET_IFM2_REGION.region;

    if (storage == chained) {
        value = read_chaining_fifo(region);
    } else {
        addr_t base[4]   = { IFM2_BASE0, IFM2_BASE1, IFM2_BASE2, IFM2_BASE3}
        addr_t height[2] = { IFM2_HEIGHT0, IFM2_HEIGHT1 }
        addr_t addr = element_address_in_region(
            base, IFM2_STRIDE_Y, IFM2_STRIDE_X, IFM2_STRIDE_C,
            IFM_WIDTH0, height, storage, precision, format, y, x, c);
        value = memory_read(region, addr, precision, type);
    }

    int32_t zero_point;
    if (type == unsigned) {
        zero_point = (uint16_t)NPU_SET_IFM2_ZERO_POINT.zero_point;
        assert((precision == b8 && zero_point < 256) ||
               (precision == b16 && zero_point == 32768) ||
               zero_point == 0);
    } else {
        zero_point = (int16_t)NPU_SET_IFM2_ZERO_POINT.zero_point;
        assert((precision == b8 && -128 <= zero_point && zero_point < 128) ||
               zero_point == 0);
    }
    return value - zero_point;
}
```

write_ofm

The following code does an OFM tensor write, excluding zero point add.

```
typedef enum {
    allowed_perms_rtrans,          // restricted transposes only
    allowed_perms_all,             // any supported permutation
} allowed_perms_t;

void write_ofm(
    int y, int x, int c,           // (y,x,c) co-ordinate in OFM
    int64_t value,                 // Value to write
    allowed_perms_t allowed_perms  // Permutations permitted
)
{
    activation_precision_t precision = NPU_SET_OFM_PRECISION.activation_precision;
    activation_type_t      type      = NPU_SET_OFM_PRECISION.activation_type;
    activation_format_t    format    = NPU_SET_OFM_PRECISION.activation_format;
    activation_storage_t   storage   = NPU_SET_OFM_PRECISION.activation_storage;
    activation_reverse_t   reverse   = NPU_SET_OFM_PRECISION.activation_reverse;
    activation_transpose_t transpose = NPU_SET_OFM_PRECISION.activation_transpose;
    int region         = NPU_SET_OFM_REGION.region;
    int ofm_height     = NPU_SET_OFM_HEIGHT_M1.height_m1 + 1;
    int ofm_width      = NPU_SET_OFM_WIDTH_M1.width_m1 + 1;
    int ofm_depth      = NPU_SET_OFM_DEPTH_M1.depth_m1 + 1;
    int ofm_blk_height = NPU_SET_OFM_BLK_HEIGHT_M1.height_m1 + 1;
    int ofm_blk_width  = NPU_SET_OFM_BLK_WIDTH_M1.width_m1 + 1;
    int ofm_blk_depth  = NPU_SET_OFM_BLK_DEPTH_M1.depth_m1 + 1;

    addr_t base[4]   = { OFM_BASE0, OFM_BASE1, OFM_BASE2, OFM_BASE3}
    addr_t height[2] = { OFM_HEIGHT0, OFM_HEIGHT1 }
    assert(0 <= y < ofm_height);
    assert(0 <= x < ofm_width);
    assert(0 <= c < ofm_depth);

    // Truncate the value to the OFM type. This ensures that chaining gets
    // the same value as non-chained OFM write followed by IFM read.
    value = truncate(value, number_of_bits(precision), type);

    // Record bottom bit of single value tensor for conditional branch
    if (ofm_width == 1 && ofm_height == 1 && ofm_depth == 1) {
        COND_STATUS.result_flag = (value & 1);
    } else {
        COND_STATUS.result_flag = UNPREDICTABLE();
    }

    if (storage == chained) {
        assert(reverse   == none);
        assert(transpose == HWC);
        write_chaining_fifo(region, value);
        return;
    }
    if (storage == none) {
        return;
    }

    // Apply reverse (not permitted with transpose or chaining)
    if (reverse != none) {
        assert(transpose == HWC);
        assert(allowed_perms == allowed_perms_all);
        assert(precision != b64);
        switch (reverse) {
            case H: y = ofm_height - 1 - y; break;
            case W: x = ofm_width  - 1 - x; break;
            case C:
                mb_depth = max(16, microblock_depth());
                if ((ofm_depth % mb_depth) != 0) {
                    assert(format == nhwc);
                    assert(ofm_blk_depth <= 16);
```

```
                }
                c = ofm_depth - 1 - c; break;
        }
    }

    // For NHCWB16 output the block dimension mapping to channel must be a multiple
 of 16
    if (format == nhcwb16) {
        if (transpose == HCW || transpose == CHW) {
            assert(ofm_blk_width % 16 == 0);
        }
        if (transpose == CWH || transpose == WCH) {
            assert(ofm_blk_height % 16 == 0);
        }
    }
    // Apply transpose.
    // The following transpose operations are always permitted on OFM write:
    // - HWC (no operation)
    // - HCW (2D transpose swapping C and W)
    // - CHW (useful for NHWC to NCHW conversion)
    // Other permutations must be permitted by allowed_perms.
    assert(transpose == HWC ||
           transpose == HCW ||
           transpose == CHW ||
           allowed_perms == allowed_perms_all);
    // Transpose not supported for 64-bit output
    assert(transpose == HWC || precision != b64);
    const int MAX_BLOCK_HEIGHT = 128;
    const int MAX_BLOCK_WIDTH  = 128;
    const int MAX_BLOCK_DEPTH  = 1024;
    switch (transpose) {
        case HWC:
            break;
        case WHC:
            assert(ofm_blk_width  <= MAX_BLOCK_HEIGHT);
            assert(ofm_blk_height <= MAX_BLOCK_WIDTH);
            t=y, y=x, x=t; break;
        case HCW:
            assert(ofm_blk_depth  <= MAX_BLOCK_WIDTH);
            assert(ofm_blk_width  <= MAX_BLOCK_DEPTH);
            t=x, x=c, c=t; break;
        case CWH:
            assert(ofm_blk_depth  <= MAX_BLOCK_HEIGHT);
            assert(ofm_blk_height <= MAX_BLOCK_DEPTH);
            t=y, y=c, c=t; break;
        case CHW:
            assert(ofm_blk_depth  <= MAX_BLOCK_HEIGHT);
            assert(ofm_blk_height <= MAX_BLOCK_WIDTH);
            assert(ofm_blk_width  <= MAX_BLOCK_DEPTH);
            t=y; y=c; c=x; x=t; break;
        case WCH:
            assert(ofm_blk_width  <= MAX_BLOCK_HEIGHT);
            assert(ofm_blk_depth  <= MAX_BLOCK_WIDTH);
            assert(ofm_blk_height <= MAX_BLOCK_DEPTH);
            t=y; y=x; x=c; c=t; break;
    }
    addr_t addr = element_address_in_region(
        base, OFM_STRIDE_Y, OFM_STRIDE_X, OFM_STRIDE_C,
        OFM_WIDTH0, height, storage, precision, format, y, x, c);
    memory_write(region, addr, precision, value);
}
```

read_weight

Read weight value from the weight stream given the output channel, kernel position and input channel.

For depth-wise convolutions, `ic == 0` as there is no reduction over input channels. In this case *oc* is the input and output channel.

```
int9_t read_weight(int oc, int ky, int kx, int ic) {
    // Parse weight[] as defined in Weight stream
    return weight[oc, ky, kx, ic];
}
```

read_bias

The following code reads the bias value from the scale and bias stream.

```
int48_t read_bias(int oc) {
    // Parse bias[] as defined in Scale and bias stream
    return bias[oc];
}
```

read_scale

The following code reads the shift and scale value from the scale and bias stream.

```
int32_t read_scale(int oc, int *shift) {
    // Parse scale[] and shift[] as defined in Scale and bias stream
    *shift = shift[oc];
    return scale[oc];
}
```

read_accumulator

The following code reads the accumulator.

```
int48_t read_accumulator(int oy, int ox, int oc, int32_t reset_value) {
    int48_t acc;
    switch (ACC_FORMAT.acc_input) {
        case reset:
            acc = reset_value;
            break;
        case keep:
            ofm_block_height = NPU_SET_OFM_BLK_HEIGHT_M1.height_m1 + 1;
            ofm_block_width  = NPU_SET_OFM_BLK_WIDTH_M1.width_m1 + 1;
            ofm_block_depth  = NPU_SET_OFM_BLK_DEPTH_M1.depth_m1 + 1;
            assert(oy < ofm_block_height);
            assert(ox < ofm_block_width);
            assert(oc < ofm_block_depth);
            acc = read_accumulator_storage(oy, ox, oc);
            break;
        case ifm2:
            acc = read_ifm2(oy, ox, oc, true);
            break;
    }
    return acc;
}
```

write_accumulator

The following code writes to the accumulator.

```
void write_accumulator(
    int oy, int ox, int oc,         // location
```

```
    int48_t acc,                    // value
    rescale_mode_t rescale_mode     // scaling mode
)
{
    if (NPU_SET_ACC_FORMAT.acc_output == enable) {
        acc = accumulate(acc, read_bias(oc));
        int64_t v = rescale(acc, oc, rescale_mode);
        v = activation(v);
        write_ofm(oy, ox, oc, v, allowed_perms_all);
    } else {
        ofm_block_height = NPU_SET_OFM_BLK_HEIGHT_M1.height_m1 + 1;
        ofm_block_width  = NPU_SET_OFM_BLK_WIDTH_M1.width_m1 + 1;
        ofm_block_depth  = NPU_SET_OFM_BLK_DEPTH_M1.depth_m1 + 1;
        assert(oy < ofm_block_height);
        assert(ox < ofm_block_width);
        assert(oc < ofm_block_depth);
        write_accumulator_storage(oy, ox, oc, acc);
    }
}
```

3.6.6 Arithmetic functions

This section defines functions for arithmetic processing of tensor elements and scalars.

clamp

The following code is for clamp result range.

```
integer clamp(integer acc, integer acc_min, integer acc_max) {
    if (acc < acc_min) {
        acc = acc_min;
    } else if (acc > acc_max) {
        acc = acc_max;
    }
    return acc;
}
```

accumulate

The following code accumulates the integer value onto accumulator.

```
integer accumulate(integer acc, integer value) {
    acc = acc + value;
    acc_format = NPU_SET_ACC_FORMAT.acc_format;

    // Accumulator wraps at the configured precision
    switch (acc_format) {
        case i32: acc = sign_extend(acc, 32); break;
        case i48: acc = sign_extend(acc, 48); break;
    }

    return acc;
}
```

rounded_shift_right

The following code shifts right with rounding, according to rounding mode.

```
int64_t rounded_shift_right(int64_t acc, int shift, round_mode_t mode, int dbl_rnd)
 {
```

```
    assert(shift>=0 && shift <= 63);
    assert(dbl_rnd>=0 && dbl_rnd <= 30);
    if (shift != 0) {
        integer R = 1 << (shift-1);   // natural round
        integer D = (shift > 31-dbl_rnd) ? (1<<(30-dbl_rnd)) : 0; // double round
        switch (mode) {
            case double_symmetric:
                R = (acc >= 0) ? (R + D) : (R - D); break;
            case double_asymmetric:
                R = R + D; break;
            case truncate_to_zero:
                R = (acc>=0) ? 0 : (1 << shift)-1; break;
            case truncate_to_lower:
                R = 0; break;
            case natural: break;
            case symmetric:
                R = (acc>=0) ? R : R-1; break;
        }
        acc = (acc + R) >> shift;   // arithmetic right shift
    }
    return acc;
}
```

rescale

The following code outputs the scale of the accumulator.

```
typedef enum {
    rescale_mode_none,       // no rescale permitted
    rescale_mode_normal,     // global or per-channel scale and shift
    rescale_mode_global,     // global only scale and shift permitted
    rescale_mode_shift_only, // global only shift permitted
    rescale_mode_avpool=64   // start of average pool scales
} rescale_mode_t;

int64_t rescale(int48_t acc, int oc, rescale_mode_t rescale_mode) {
    int32_t scale;
    int64_t result;
    uint6_t shift;

    if (rescale_mode == rescale_mode_none) {
        return acc;
    }
    if (rescale_mode >= rescale_avpool) {
        int d = 1 + (rescale_mode - rescale_avpool);  // need to scale by 1/d
        int k = 32 - clz(d - 1); // (1<<k)/2 < d <= (1<<k)
        int64_t numerator = (((int64_t)1 << 30) + 1) << k;
        scale = numerator / d; // (1<<30) <= scale < (1<<31)
        shift = 30 + k;
    } else if (NPU_SET_OFM_PRECISION.scale_mode == per_channel) {
        // per channel scaling
        assert(scale_mode == rescale_mode_normal);
        assert(oc >= 0);
        scale = read_scale(oc, &shift);
    } else {
        // per tensor (global) scaling
        scale = NPU_SET_OFM_SCALE.scale;
        shift = NPU_SET_OFM_SCALE.shift;
    }
    assert(scale >= 0);
    if (NPU_SET_ACC_FORMAT.acc_format == i32) {
        assert(acc == sign_extend(acc, 32));
    } else {
        assert(acc   == sign_extend(acc, 48));
        assert(scale == sign_extend(scale, 16));
    }
    if (rescale_mode != rescale_mode_shift_only)
        result = (int64_t)acc * scale;
```

```
    } else {
        result = acc;
    }
    mode = NPU_SET_OFM_SCALE.round_mode;
    dbl_rnd = NPU_SET_OFM_SCALE.dbl_rnd;
    result = rounded_shift_right(result, shift, mode, dbl_rnd);
    return result;
}
```

scale_a

The following code outputs the scale of the first input IFM.

```
int32_t scale_a(int32_t a) {
    activation_precision_t precision = NPU_SET_IFM_PRECISION.activation_precision;
    activation_type_t      act_type  = NPU_SET_IFM_PRECISION.activation_type;
    if (precision == b8 || (precision == b16 && act_type == signed))
    {
        int32_t scale = NPU_SET_IFM_SCALE.scale;
        uint6_t shift = NPU_SET_IFM_SCALE.shift;
        int dbl_rnd   = NPU_SET_IFM_SCALE.dbl_rnd;
        round_mode_ifm_t round_mode = NPU_SET_IFM_SCALE.round_mode;
        assert(a == sign_extend(a, 16));
        assert(scale >= 0);
        int48_t result = a * scale;
        a = rounded_shift_right(result, shift, round_mode, dbl_rnd);
    }
    return a;
}
```

scale_b

The following code outputs the scale of the second input IFM2.

```
int32_t scale_b(int32_t b) {
    activation_precision_t precision = NPU_SET_IFM_PRECISION.activation_precision;
    activation_type_t      act_type  = NPU_SET_IFM_PRECISION.activation_type;
    if (precision == b8 || (precision == b16 && act_type == signed))
    {
        int32_t scale = NPU_SET_IFM2_SCALE.scale;
        uint6_t shift = NPU_SET_IFM2_SCALE.shift;
        int dbl_rnd   = NPU_SET_IFM2_SCALE.dbl_rnd;
        round_mode_ifm_t round_mode = NPU_SET_IFM2_SCALE.round_mode;
        assert(b == sign_extend(b, 16));
        assert(scale >= 0);
        int48_t result = b * scale;
        b = rounded_shift_right(result, shift, round_mode, dbl_rnd);
    }
    return b;
}
```

clz

The following is the count leading zeros function. The function returns the number of leading zeros in the range 0 to n for an n-bit input value.

```
unsigned integer clz(unsigned integer a, integer n) {
    while (a!=0) {
        n--;
        a = (unsigned)a>>1;
    }
    return n;
```

```
}
```

3.6.7 activation

The following code applies an activation function.

```
int64_t activation(int64_t acc64) {
    activation_precision_t precision = NPU_SET_OFM_PRECISION.activation_precision;
    activation_type_t act_type       = NPU_SET_OFM_PRECISION.activation_type;
    if (precision == b64) {
        // 64-bit output does not apply any activation functions
        return acc64;
    }

    int17_t zero_point, act_min, act_max;
    if (act_type == unsigned) {
        zero_point = (uint16_t)(NPU_SET_OFM_ZERO_POINT.zero_point);
        act_min    = (uint16_t)(NPU_SET_ACTIVATION_MIN.clip_boundary);
        act_max    = (uint16_t)(NPU_SET_ACTIVATION_MAX.clip_boundary);
        assert(zero_point < 256 || zero_point == 32768);
    } else {
        zero_point = (int16_t)(NPU_SET_OFM_ZERO_POINT.zero_point);
        act_min    = (int16_t)(NPU_SET_ACTIVATION_MIN.clip_boundary);
        act_max    = (int16_t)(NPU_SET_ACTIVATION_MAX.clip_boundary);
        assert(-128 <= zero_point || zero_point < 128);
    }

    // Add zero point and clamp range
    int32_t acc = (int32_t)(acc64 + zero_point);
    if (NPU_SET_ACTIVATION.activation_function == lut_tanh) {
        acc = clamp(acc, -0x10000, 0xffff);
    } else if (NPU_SET_ACTIVATION.activation_function == lut_sigmoid) {
        acc = clamp(acc, -0x1f000, 0x1efff);
    } else if (NPU_SET_ACTIVATION.activation_clip_range == b16) {
        acc = clamp(acc, act_min, act_max);
    }

    // Lookup table base activation
    int table = NPU_SET_ACTIVATION.table;  // table number
    switch (NPU_SET_ACTIVATION.activation_function) {
        case lut_none:
            break;
        case lut_u8_u8:
            acc = ((uint8_t*)lookup_table_storage)[(table << 8) + (acc & 0xFF)];
            break;
        case lut_s8_s8:
            acc = ((int8_t*)lookup_table_storage)[(table << 8) + ((acc+128) &
0xFF)];
            break;
        case lut_s8_s16:
            assert(table < 4);
            acc = ((int16_t*)lookup_table_storage)[(table << 8) + ((acc+128) &
0xFF)];
            break;
        case lut_s8_s32:
            assert(table < 2);
            acc = ((int32_t*)lookup_table_storage)[(table << 8) + ((acc+128) &
0xFF)];
            break;
        case lut_s16_s16:
            assert(table == 0);
            index = 256 + (acc>>7);
            base  = ((int16_t *)lookup_table_storage)[2*index+0];  // base and slope
can be looked
            slope = ((int16_t *)lookup_table_storage)[2*index+1];  // up with a
single 32-bit access
```

```
            acc   = ((base << 7) + (acc & 0x7F)*slope + 64)>>7;
            acc   = (int16_t)acc;  // truncate to 16-bit
            break;
        case lut_s16_s32:
            assert(table == 0);
            index = 256 + (acc>>7);
            base  = ((int16_t *)lookup_table_storage)[2*index+0];  // base and slope
 can be looked
            slope = ((int16_t *)lookup_table_storage)[2*index+1];  // up with a
 single 32-bit access
            acc   = (base << 7) + (acc & 0x7F)*slope;
            break;
        case lut_tanh:
            assert(table == 0);
            index = 256 + (acc>>8);
            base  = ((int16_t *)lookup_table_storage)[2*index+0];  // base and slope
 can be looked
            slope = ((int16_t *)lookup_table_storage)[2*index+1];  // up with a
 single 32-bit access
            round = (acc >= 0) ?  0x80 : 0x7F;
            acc   = ((base << 8) + (acc & 0xFF)*slope + round)>>8;
            break;
        case lut_sigmoid:
            assert(table == 0);
            index = 256 + (acc>>9);
            base  = ((int16_t *)lookup_table_storage)[2*index+0];  // base and slope
 can be looked
            slope = ((int16_t *)lookup_table_storage)[2*index+1];  // up with a
 single 32-bit access
            round = (acc >= 0) ? 0x1000200 : 0x10001FF; // includes input bias of
 1<<15
            acc   = ((base << 9) + (acc & 0x1FF)*slope + round)>>10;
            break;
        }
    }
    return acc;
}
```

3.6.8 Error handling

The following code raises an error.

raise_error

```
void raise_error(void) {
      safely_complete_external_axi_transactions();
      STATUS.state = stopped;
      raise_irq();
}
```

3.6.9 Command stream state setting (SET commands)

The following code sets the state register from the command stream.

set_register

```
void set_register(void) {
      // This pseduo-function sets the value of the register with
      // the name of the command to the value of the parameters set
      // in the command.
```

```
}
```

3.7 Scale and bias stream

This section describes the scale and bias stream format and order.

3.7.1 Scale and bias stream format

The scale and bias values are grouped into channel sets.

The scale and bias stream encodes scaling multiplier, shift and bias values for each output channel. The stream is made up of a number of channel sets. The number of channels in a set is defined to be the microblock depth rounded up to the next multiple of 16:

channels_in_set

```
channels_in_set = align_up(microblock_depth(), 16);
```

The section Scale and bias ordering defines the ordering of channel sets within a stream.

The section Scale and bias set encoding defines how each set is encoded as bits.

3.7.2 Scale and bias ordering

The ordering of channel sets within a stream.

If NPU_SET_OFM_PRECISION.activation_reverse != C then the order matches the output channel order.

If NPU_SET_OFM_PRECISION.activation_reverse == C then the order of channel sets is reversed and so matches the output channel order. However, within a channel set, the order matches the MAC output order and so is not reversed. If the number of OFM channels is greater than or equal to 16 then channel reverse is only permitted when the number of output channels is a multiple of the channel set size. Thus all channel sets are full in this case. Note also that reverse is not permitted in combination with transpose.

The following code shows how to parse the scale and bias stream where *oc* is the original output channel order before reverse or transpose:

parse_scale_bias_channel_stream

```
parse_scale_bias_channel_stream(bitstream) {
    int oc_size = NPU_SET_OFM_DEPTH_M1.depth_m1 + 1;
    if (NPU_SET_OFM_PRECISION.activation_reverse == C) {
        assert(oc_size <= 16 || oc_size % channels_in_set == 0);
        for (int oc = align_up(oc_size, channels_in_set); oc > 0; ) {
            oc -= channels_in_set
            parse_scale_bias_channel_set(bitstream, oc);
```

```
        }
    } else {
        for (int oc = 0; oc < oc_size; ) {
            parse_scale_bias_channel_set(bitstream, oc);
            oc += channels_in_set;
        }
    }
}
```

3.7.3 Scale and bias set encoding

A set encodes scale, shift, and bias for a number of channels.

Each scale and bias set encodes channels_in_set number of channels. If this is larger than the available number of output channels then unused symbols are encoded as zero.

A scale and bias channel set is encoded as a bit-stream with least significant bit first. This is the same order as in the weight stream. See Bit order convention.

Each scale and bias channel record consists of 10 bytes (80 bits). The byte-stream format is described by the following parsing pseudocode. This reads a channel set starting at channel number *oc*.

parse_scale_bias_channel_set

```
parse_scale_bias_channel_set(bitstream, int oc) {
    for (i = 0; i < channels_in_set; i++) {
        if (NPU_SET_ACC_FORMAT.acc_format == i32)) {
            SYMBOL_UINT(bitstream, bias [oc + i], 32);
            SYMBOL_UINT(bitstream, scale[oc + i], 31);
            SYMBOL_RESERVED(bitstream,         1);
        } else { // 48-bit accumulator
            SYMBOL_UINT(bitstream, bias [oc + i], 48);
            SYMBOL_UINT(bitstream, scale[oc + i], 15);
            SYMBOL_RESERVED(bitstream,         1);
        }
        SYMBOL_UINT(bitstream, shift[oc + i],  6);
        SYMBOL_RESERVED(bitstream,        10);
    }
}
```

3.8 Tiling

This section describes how feature maps can be split into tiles.

The NPU supports spatial tiling in which a feature map of dimensions `[H,W,C]` can be stored in memory as up to four feature submaps. These submaps have dimension `[H[t],W[t],C]` where t=0,1,2,3 is the tile number. The NPU accesses the appropriate tile depending on the `[Y,X]` co-ordinates accessed within the tensor. Tiles allow for efficient construction of rolling buffers the compiler uses in memory optimizations.

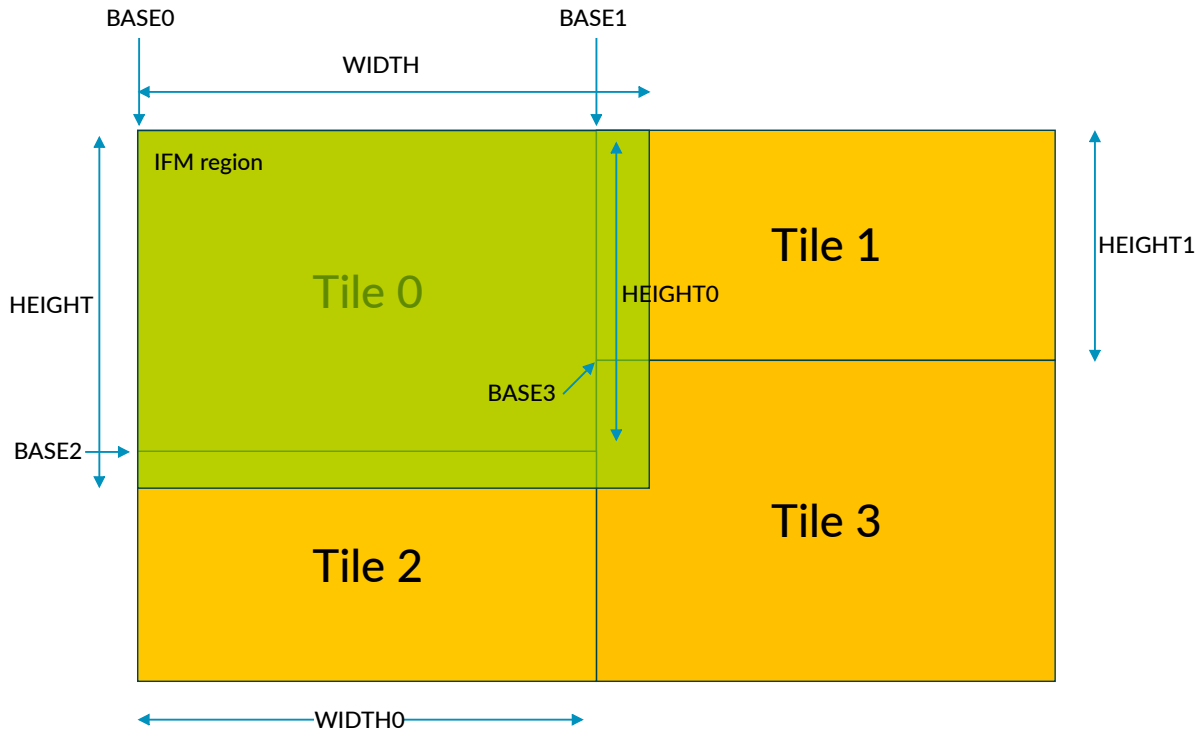
Tiles are available for NHWC and NHCWB16 formats. Tiles are available in a 2x2 and 3x1 arrangement.

3.8.1 Tiling 2x2

The NPU supports 2x2 Input Feature Map (IFM) tiling.

The following figure shows how registers specify the tiles of an IFM feature map for a 2x2 tiling. Similar tiling is defined for IFM2 and OFM feature maps. The IFM feature map accessed has size HEIGHT x WIDTH.

Figure 3-65: IFM 2x2 tiling

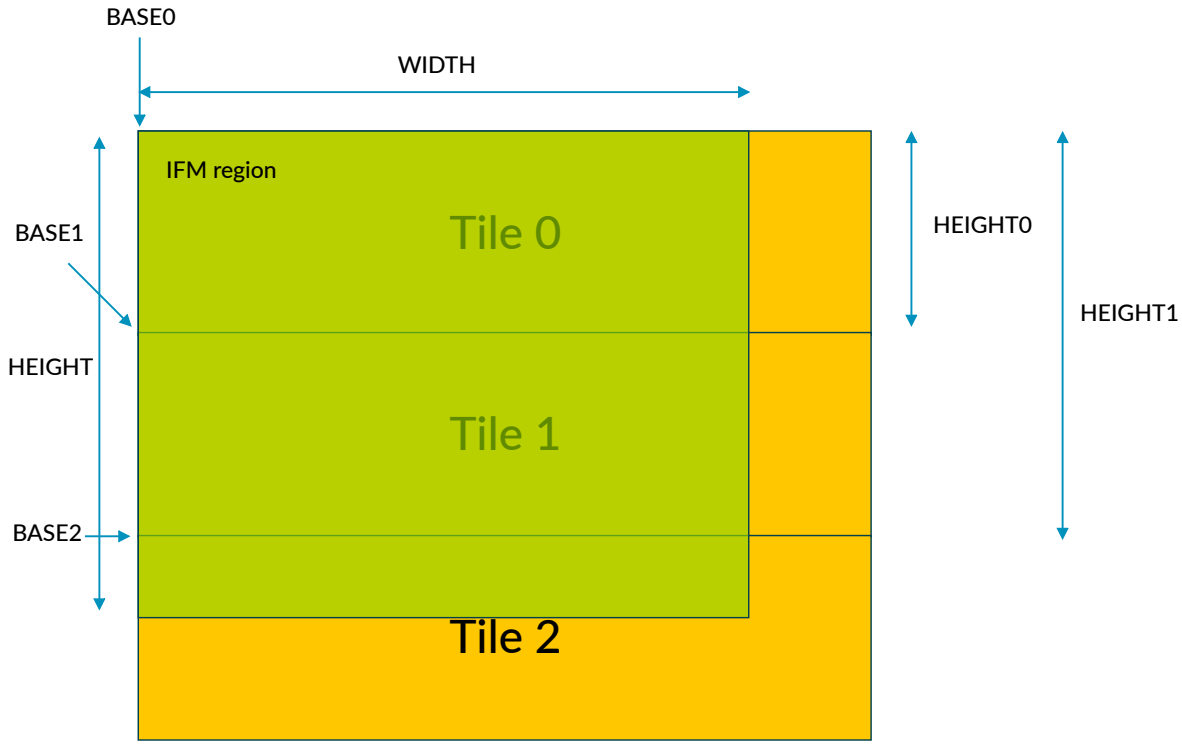


3.8.2 Tiling 3x1

The NPU supports 3x1 Input Feature Map (IFM) tiling.

The following figure shows how registers specify the tiles of an IFM feature map for a 3x1 tiling. Similar tiling is defined for IFM2 and OFM feature maps. The IFM feature map accessed has size HEIGHT x WIDTH.

Figure 3-66: IFM 3x1 tiling



3.9 Weight stream

This section describes the weight bit-stream format and weight order.

3.9.1 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 the lossless compressed format described in this section.

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 kept during the slice.

Ethos-U command stream operations NPU_OP_CONV and NPU_OP_DEPTHWISE that read in a weight stream require that the weights come in a certain order to function properly. The weights are not only reordered, padding is also inserted to align to full weight blocks that the weight decoder works on. Padding is done by inserting weights that are zero into the weight stream.

Therefore, unless the operation aligns perfectly to the internal work blocks, the uncompressed weight stream is always larger than the original weights.

3.9.2 Weight ordering

This section defines the order of weights within the weight stream and the following sections define the bitstream encoding of those weights.

The pseudocode in this section maps a 4D weight tensor into one or more 1D weight stream arrays. The number of weight streams is configuration-dependent and is set as num_wd. num_wd and the depthwise specific padding parameters, dw8_pad_k and dw16_pad_k, are specified in the following table. The original weights have dimensions: weight[OFM channel][kernel height][kernel width][IFM channel]. The output is the different streams: weight_stream[0], ... , weight_stream[num_wd-1]. The number of elements (including padding) in each weight stream is also calculated in the code: wcnt[0], ... , wcnt[num_wd-1].

Table 3-90: Number of weight streams and depthwise kernel padding.

macs_per_cc	num_wd	dw8_pad_k	dw16_pad_k
128	1	0	2
256	1	0	2
512	2	2	2
1024	4	6	6
2048	4	6	6

```
for (i = 0; i < num_wd; i++) {
    wcnt[i] = 0;
}

depthwise = (operation == NPU_OP_DEPTHWISE);
ofm_block_depth  = NPU_SET_OFM_BLK_DEPTH_M1.depth_m1 + 1;
ofm_depth = NPU_SET_OFM_DEPTH_M1.depth_m1 + 1;
ifm_depth = NPU_SET_IFM_DEPTH_M1.depth_m1 + 1;
decomp_size = NPU_SET_KERNEL_STRIDE.decomposition == d8x8 ? 8 : 4;
dilation_y = NPU_SET_KERNEL_STRIDE.dilation_y; // 0 or 1
dilation_x = NPU_SET_KERNEL_STRIDE.dilation_x; // 0 or 1
kernel_height = NPU_SET_KERNEL_HEIGHT_M1.height_m1 + 1;
kernel_width  = NPU_SET_KERNEL_WIDTH_M1.width_m1 + 1;
partkernel = (NPU_SET_KERNEL_STRIDE.weight_order == part_kernel_first) || depthwise;
sparsity = (NPU_SET_WEIGHT_FORMAT.weight_sparsity == sparse_2_4);
ifm16bit = (NPU_SET_IFM_PRECISION.activation_precision == b16);
dw_pad_k = ifm16bit ? dw16_pad_k : dw8_pad_k;
microblock = NPU_SET_ACC_FORMAT.microblock;
mb_i16only = (microblock_depth() == 16 && (microblock == u1x1 || microblock ==
 u1x2));
// mb_i16only is only permitted for 16-bit IFM with 1x1 kernel
assert(!mb_i16only || (ifm16bit && kernel_height == 1 && kernel_width == 1));
if (partkernel) {
    ifm_block_depth = 16;  // unused in the case of depthwise
} else if (!sparsity && mb_i16only) {
    ifm_block_depth = 32;
} else if (!sparsity || ifm16bit) {
    ifm_block_depth = 64;
} else { // sparsity and 8-bit IFM
    ifm_block_depth = 128;
}
```

```
wblk_k = partkernel ? (!ifm16bit && !sparsity && !depthwise ? 5 : 10) : 1;
wblk_o = microblock_depth();
wblk_i = depthwise ? 1 : (partkernel && ifm16bit && !sparsity ? 8 :
 ifm_block_depth);

ifm_loop_start = 0;
ifm_loop_end = depthwise ? 1 : ifm_depth; // disable loop for depthwise by setting
 depth=1
ifm_loop_inc = -ifm_block_depth;

for (oblk_z = 0; oblk_z < ofm_depth; oblk_z += ofm_block_depth) {
    ifm_loop_inc = -ifm_loop_inc;
    for (iblk_z = ifm_loop_start; iblk_z < ifm_loop_end && iblk_z >= 0; iblk_z +=
 ifm_loop_inc) {
        ifm_loop_start = iblk_z;
        kernel_width_undilated = (kernel_width+(1<<dilation_x)-1) >> dilation_x;
        kernel_height_undilated = (kernel_height+(1<<dilation_y)-1) >> dilation_y;
        decomp_w = decomp_size >> dilation_x;
        decomp_h = decomp_size >> dilation_y;
        for (subk_x = 0; subk_x < kernel_width_undilated; subk_x += decomp_w) {
            for (subk_y = 0; subk_y < kernel_height_undilated; subk_y += decomp_h) {
                subk_w = min(kernel_width_undilated - kernel_x, decomp_w);
                subk_h = min(kernel_height_undilated - kernel_y, decomp_h);
                iblk_depth = depthwise ? 1 : min(ifm_block_depth, ifm_depth -
 ifm_blk_z);
                for (wblk_iz = 0; wblk_iz < iblk_depth; wblk_iz += wblk_i) {
                    oblk_depth = min(ofm_block_depth, ofm_depth - oblk_z);
                    for (ublk_z = 0; ublk_z < oblk_depth; ublk_z += ublk_depth) {
                        if (partkernel) {
                            partkernel_order(weight, weight_stream, wcnt, subk_x,
 subk_y,
                                             oblk_z + ublk_z, iblk_z, subk_w,
 subk_h,
                                             wblk_k, wblk_o, wblk_i, ofm_depth,
 ifm_depth,
                                             depthwise, dw_pad_k, sparsity);
                        } else {
                            depthfirst_order(weight, weight_stream, wcnt, subk_x,
 subk_y,
                                             oblk_z + ublk_z, iblk_z, subk_w,
 subk_h,
                                             wblk_o, wblk_i, ofm_depth, ifm_depth,
                                             sparsity);
                        }
                    }
                }
            }
        }
    }
}


void partkernel_order(weight, weight_stream, wcnt, kernel_x, kernel_y, oblk_z,
 iblk_z,
                      kw, kh, wblk_k, wblk_o, wblk_i,
                      ofm_depth, ifm_depth, depthwise, dw_pad_k, sparsity)
{
    stride_y = 2*NPU_SET_KERNEL_STRIDE.stride_y_msb
 + NPU_SET_KERNEL_STRIDE.stride_y_lsb + 1;
    stride_x = 2*NPU_SET_KERNEL_STRIDE.stride_x_msb
 + NPU_SET_KERNEL_STRIDE.stride_x_lsb + 1;

    subk_num = subk_w * subk_h;
    subk_num = ((subk_num + wblk_k - 1) / wblk_k) * wblk_k;

    for (sy = 0; sy < stride_y; sy++) {
        for (sx = 0; sx < stride_x; sx++) {
            strided_kh = (kh + stride_y-1 - sy) / stride_y;
            for (ky = 0; ky < strided_kh; ky++) {
                strided_kw = (kw + stride_x-1 - sx) / stride_x;
                for (kx = 0; kx < strided_kw; kx++) {
```

```
                    x = (ky%2 == 0) ? kx : (strided_kw - 1 - kx);
                    subk_num--;
                    i_sz = depthwise ? 1 : (sparsity ? 16 : 8);
                    for (i = 0; i < wblk_i; i += i_sz) {
                        for (o = 0; o < wblk_o; o++) {
                            ws = (o>>2) & (num_wd - 1);
                            for (ii = 0; ii < i_sz; ii++) {
                                ifm_z = iblk_z + i + ii;
                                ofm_z = oblk_z + o;
                                xx = kernel_x + sx + x * stride_x;
                                y = kernel_y + sy + ky * stride_y;
                                if (ofm_z < ofm_depth && ifm_z < ifm_depth) {
                                    weight_stream[ws][wcnt[ws]++] = weight[ofm_z][y]
[xx][ifm_z];
                                } else {
                                    weight_stream[ws][wcnt[ws]++] = 0;
                                }
                            }
                        }
                    }
                    if (depthwise && subk_num % wblk_k == 0) {
                        for (ws = 0; ws < num_wd; ws++) {
                            for (i = 0; i < dw_pad_k*wblk_o/num_wd; i++) {
                                weight_stream[ws][wcnt[ws]++] = 0;
                            }
                        }
                    }
                }
            }
        }
    }
    if (subk_num > 0) {
        for (ws = 0; ws < num_wd; ws++) {
            for (i = 0; i < (subk_num + (depthwise ? dw_pad_k : 0))*wblk_i*wblk_o/
num_wd; i++) {
                weight_stream[ws][wcnt[ws]++] = 0;
            }
        }
    }
}

void depthfirst_order(weight, weight_stream, wcnt, kernel_x, kernel_y, oblk_z,
 iblk_z,
                      kw, kh, wblk_o, wblk_i, ofm_depth, ifm_depth, sparsity)
{
    i_sz = sparsity ? 16 : 8;  // number of ifm channels process every cycle
    for (ky = 0; ky < kh; ky++) {
        for (kx = 0; kx < strided_kw; kx++) {
            x = (ky%2 == 0) ? kx : (kw - 1 - kx);
            for (i = 0; i < wblk_i; i += i_sz) {
                for (o = 0; o < wblk_o; o++) {
                    ws = (o>>2) & (num_wd - 1);
                    for (ii = 0; ii < i_sz; ii++) {
                        ifm_z = iblk_z + i + ii;
                        ofm_z = oblk_z + o;
                        xx = kernel_x + x;
                        y = kernel_y + ky;
                        if (ofm_z < ofm_depth && ifm_z < ifm_depth) {
                            weight_stream[ws][wcnt[ws]++] = weight[ofm_z][y][xx]
[ifm_z];
                        } else {
                            weight_stream[ws][wcnt[ws]++] = 0;
                        }
                    }
                }
            }
        }
    }
}
```

If the Fast Weight Decoder (FWD) is used, all the different weight streams are merged into one fwd_weight_stream in the following way. If the NPU is a configuration that only uses one weight stream, fwd_weight_stream becomes the same as weight_stream[0].

```
if (NPU_SET_WEIGHT_FORMAT.weight_format == fwd) {
    fwd_wcnt = 0;
    wblk_size = wblk_k*wblk_o*wblk_i / num_wd;
    for (i = 0; i < wcnt[0]; i += wblk_size) {
        for (ws = 0; ws < num_wd; ws++) {
            for (j = 0; j < wblk_size; j++) {
                fwd_weight_stream[fwd_wcnt++] = weight_stream[ws][i+j];
            }
        }
    }
}
```

3.9.3 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 `0x55`, 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`.

The following functions are used in the parsing code to decode bit packed symbols.

SYMBOL_UINT

```
SYMBOL_UINT(bitstream_t *b, &value, int n) {
    value = 0;
    for (i = 0; i< n; i++) {
        value |= read_bit(bitstream) << i;
    }
}
```

SYMBOL_SINT

```
SYMBOL_UINT(bitstream_t *b, &value, int n) {
    value = 0;
    for (i = 0; i< n; i++) {
        value |= read_bit(bitstream) << i;
    }
    value = sign_extend(value, n);
}
```

SYMBOL_RESERVED

```
SYMBOL_RESERVED(bitstream_t *b, int n) {
    for (i = 0; i< n; i++) {
        assert(read_bit(bitstream) == 0);
    }
}
```

3.9.4 Syntax of a weight stream

A weight stream consists of a sequence of weight slices.

Each weight slice is prefixed by a *zdiv* field that specifies the coding mode used for the slice.

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.

parse_weight_stream

```
parse_weight_stream(weight_stream_t *weight_stream) {
    bool end_of_stream = false;
    while (!end_of_stream) {
        // The zdiv symbol specifies the coding mode:
        // 0-3 selects alternating mode with zero-run compression using a GRC
        //     divisor of (1 << zdiv)
        // 6   selects that zero-run compression is not used
        // 7   specifies end of stream
        // other values are reserved
        SYMBOL_UINT(weight_stream, zdiv, 3);
        if (zdiv == 7) {
            // align to 128 bits using padding with '1'
            while (!aligned(weight_stream[ws], 128)) {
                alignment_bit(weight_stream[ws]);
            }
            end_of_stream = true;
        } else {
            parse_weight_slice(weight_stream);
        }
    }
}
```

3.9.5 Syntax of a weight slice

A weight slice consists of a slice header followed by one or more data chunks.

The *zdiv* field before slice header indicates to the weight decoder when to switch the coding mode. Using an extended header, the slice header can optionally be used to reload the palette (lookup table).

The following table shows the symbols used in a weight slice. The example code shows the weight slice syntax.

Table 3-91: Weight slice symbols

Symbol	Meaning
`slicelen`	The number of weights in this slice in the range 1–32768. In alternating mode, this is the number of nonzero weights.
`wdiv`	Weight GRC divisor. 0-5 specifies the weight index GRC divisor to be `1<<wdiv` . 7 specifies uncompressed mode.
`wtrunc`	If this symbol is set, the weight GRC unary length is truncated to 2.

Symbol	Meaning
`newpal`	If this symbol is one, a new palette mode is configured. If this symbol is zero, `dirofs`, `palsize`, `palbits`, and palette[i] keep the values from the previous slice. This symbol must be set for the first slice in a stream.
`dirofs`	Direct mode offset.
`palsize`, palette_size	Indicates the number of entries in the palette. A value of 0 means direct mode where the palette is not used.
`palbits`, palette_bits	If the palette is used (palette_size>0), palette_bits indicates the precision in bits of each palette entry. In direct mode (palette_size==0), palette_bits indicates the precision used in uncompressed mode.
palette[i]	Weight value for palette entry with index *i*. The weight value is stored in sign-magnitude format. Bit `palette[i][0]` specifies the sign. Bits `palette[i][palette_bits-1:1]` specify the absolute level. The weight value is calculated with the following formula: `weight_value = (palette[i] & 1) ? -(palette[i]>>1) : +(palette[i]>>1);`

parse_weight_slice

```
parse_weight_slice(weight_stream_t *weight_stream) {
    SYMBOL_UINT(weight_stream, slicelen_m1, 15);
    slicelen = slicelen_m1 + 1;
    SYMBOL_UINT(weight_stream, wdiv, 3);
    SYMBOL_UINT(weight_stream, wtrunc, 1);
    SYMBOL_UINT(weight_stream, newpal, 1);
    if (newpal) {
        SYMBOL_UINT(weight_stream, dirofs, 5);
        SYMBOL_UINT(weight_stream, palsize, 5);
        SYMBOL_UINT(weight_stream, palbits, 3);
        palette_size = (palsize == 0) ? 0 : palsize + 1;
        palette_bits = palbits + 2;
        for (i = 0; i < palette_size; i++) {
            SYMBOL_UINT(weight_stream, palette[i], palette_bits);
        }
    }
    while (!end_of_slice()) {
        parse_weight_chunk();
    }
}
```

3.9.6 Palette mode and direct coding mode

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

If the weight index is less than palette_size, then the weight index is used as an index into the palette. The weight value is found in the palette entry for that index. If the weight index is greater than or equal to the palette_size, the weight value is calculated directly from the weight index using the following formula. If palette_size==0, this indicates direct mode where the palette is not used.

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

3.9.7 Golomb-Rice weight index coding mode

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

This weight index coding mode is signaled by *wdiv* in the range of 0–5.

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

The quotient *wq* must be less than or equal to 31. If *wtrunc* is set, then *wq* must be less than or equal to two. It is the responsibility of the encoder to select the *wdiv* parameter so that this condition is true. 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 3-92: 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 3-93: 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

3.9.8 Uncompressed weight index coding mode

In uncompressed coding, the weight_index is coded directly as an unsigned binary integer.

If *wdiv*=7, this indicates uncompressed coding.

The number of bits used, uncompressed_bits, is derived from the palette size when the palette is non-empty. If the palette is empty, 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.

3.9.9 Alternating coding mode (zero-run coding)

Alternating coding mode is beneficial if weights with a value of zero are frequent in the weight stream.

If `zdiv < 4`, alternating mode is enabled.

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. Therefore, `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
```

We then code the sequence as follows:

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


Note

Zero weight values must not be encoded using weight indices in alternating mode except in the case of a slice where all weights are zero and *newpal* = 1. In this case the first weight index may encode a zero weight.

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. In particular, the initial zero run is coded 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, assume we have three slices and all of them are coded using alternating mode and newpal is set for slice one and slice three.

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

We then code the following.

```
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)
```

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

3.9.10 Syntax of a weight chunk

Weight chunks follow the slice header and encode the weight indices.

In alternating mode, the chunk encodes the zero runs that belong to the slice in addition to the weight indices. Each chunk encodes 0–12 weight indices and 0–12 zero-run values. These values are generally not the same number. The reason for this difference is that the number of values depends on the quotient values, that is, the unary lengths. If the unary lengths are long, 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 because this number depends on the weight and `zrun` values.

In alternating mode, a type of flow control is used to ensure the number of coded weight indices and `zrun` values are roughly the same. This rough equivalence 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 eight, only `zrun` values are included in the chunk to allow `zrun` to catch up. Similarly, if the balance is less than zero, only weight indices are included in the chunk. If the balance is between zero and seven, both weights and `zrun` values are included in the chunk.

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

The chunk bitstream syntax and parsing process are described in the following example code. The output of the process is the weight_indices and the `zruns` arrays.

parse_weight_chunk

```
parse_weight_chunk(weight_stream_t *weight_stream) {
  w_cnt = slicelen;
  z_cnt = slicelen + 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, balance ensures the rate of weight
    // indices and zrun values 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) {
      SYMBOL_UINT(weight_stream, wunary0, 12);
    }
    if (z_enable) {
      SYMBOL_UINT(weight_stream, 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) {
          wunary1_len++;
        }
      }
      SYMBOL_UINT(weight_stream, 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) {
        SYMBOL_UINT(weight_stream, wremain, wremain_bits);
        weight_indices[wr_i++] += wremain;
      }
    }
    if (prev_z_enable) {
      while (zr_i < prev_zq_i) {
        SYMBOL_UINT(weight_stream, 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);
}
```

3.9.11 FWD Weight stream structure

The Fast Weight Decoder (FWD) uses a different weight stream format from the weight stream described previously. The stream consists of a header followed by weights packed at a fixed number of bits per weight.

The following code shows the syntax of the 256-bit header.

```
parse_fwd_header(weight_stream_t *weight_stream) {
    SYMBOL_UINT(weight_stream, raw_mode_flag, 1);
    SYMBOL_UINT(weight_stream, small_lut_flag, 1);
    SYMBOL_UINT(weight_stream, zero_reserved, 102);
    // Zero point in range -128 to +127 for 8-bit weights
    SYMBOL_SINT(weight_stream, zero_point, 8);
    // Remaining 18 bytes are a table of 16 x 9-bit weight values
    for (i = 0; i < 16; i++) {
        // each weight is encoded as an 8-bit magnitude and 1-bit sign
        SYMBOL_UINT(weight_stream, lut_sign[i], 1);  // first (LSB) is sign
        SYMBOL_UINT(weight_stream, lut_abs [i], 8);  // magnitude
    }
}
```

The FWD header is followed by one or more 256-bit words, each word containing 32, 64, or 128 weights according to the flags as follows:

```
parse_fwd_word(weight_stream_t *weight_stream) {
    if (raw_mode_flag == 1) {
        // 8 bits per weight
        for (int i = 0; i < 32; i++) {
            // raw weight in the range -128 to +127
            SYMBOL_SINT(weight_stream, value, 8);
            // subtract zero point (from header) for range -255 to +255
            value = value - zero_point
            weight_abs[i]  = (value < 0) ? -value : value;
            weight_sign[i] = (value < 0) ?      1 :     0;
        }
    } else if (small_lut_flag == 0) {
        // 4 bits per weight index
```

```
        for (int i = 0; i < 64; i++) {
            SYMBOL_UINT(weight_stream, index, 4);
            weight_abs[i]  = lut_abs [index];
            weight_sign[i] = lut_sign[index];
        }
    } else { // small_lut_flag = 1
        // 2 bits per weight index
        for (int i = 0; i < 128; i++) {
            SYMBOL_UINT(weight_stream, index, 2);
            weight_abs[i]  = lut_abs [index];
            weight_sign[i] = lut_sign[index];
        }
    }
}
```

3.10 Instruction index for cmd0 stream

This section describes 32-bit NPU commands.

Commands in this section form part of the NPU command stream and execute on the NPU. Each command in this section consists of a single 32-bit word. Bits [31:16] of the word encode a 16-bit parameter, payload_0. Commands are 32-bit aligned in memory.

Table 3-94: List of instructions in the cmd0 pipe

Name	Description
NPU_OP_CONV	2D convolution
NPU_OP_DEPTHWISE	Depth-wise 2D convolution
NPU_OP_DMA_START	Queue new DMA for the given channel
NPU_OP_DMA_WAIT	Wait for the mem2mem DMA channel to have k or fewer active descriptors outstanding
NPU_OP_ELEMENTWISE	Elementwise operation
NPU_OP_IRQ	Raises an IRQ to the host
NPU_OP_KERNEL_WAIT	Wait for kernel operations to complete
NPU_OP_PMU_MASK	Enable or disable PMU counting (debug feature only)
NPU_OP_POOL	Pooling
NPU_OP_RESIZE	Resize operation
NPU_OP_STOP	Signal the end of command stream
NPU_SET_ACC_FORMAT	Accumulator format
NPU_SET_ACTIVATION	Activation function and clip range
NPU_SET_ACTIVATION_MAX	Upper bound clip
NPU_SET_ACTIVATION_MIN	Lower bound clip
NPU_SET_BLOCKDEP	Block dependency offset for kernel operations
NPU_SET_DMA0_DST_REGION	DMA0 destination region
NPU_SET_DMA0_IDX_REGION	DMA0 index region
NPU_SET_DMA0_SIZE0	Size of second dimension for 2D/3D transfers
NPU_SET_DMA0_SIZE1	Size of third dimension for 3D transfers
NPU_SET_DMA0_SRC_REGION	DMA0 source region
NPU_SET_IFM2_BROADCAST	IFM2 broadcast configuration

Name	Description
NPU_SET_IFM2_HEIGHT0_M1	IFM2 Tile 0 height
NPU_SET_IFM2_HEIGHT1_M1	IFM2 Tile 1 height
NPU_SET_IFM2_PRECISION	IFM2 Precision
NPU_SET_IFM2_REGION	Index n for IFM2 access
NPU_SET_IFM2_WIDTH0_M1	IFM2 Tile 0 and Tile 2 width
NPU_SET_IFM2_ZERO_POINT	IFM2 zero point
NPU_SET_IFM_BROADCAST	IFM broadcast configuration
NPU_SET_IFM_DEPTH_M1	Number of input channels for convolution
NPU_SET_IFM_HEIGHT0_M1	IFM Tile 0 height
NPU_SET_IFM_HEIGHT1_M1	IFM Tile 1 height
NPU_SET_IFM_PAD_BOTTOM	IFM bottom pad
NPU_SET_IFM_PAD_LEFT	IFM left pad
NPU_SET_IFM_PAD_RIGHT	IFM right pad
NPU_SET_IFM_PAD_TOP	IFM top pad
NPU_SET_IFM_PRECISION	IFM Precision
NPU_SET_IFM_REGION	Index n for IFM access
NPU_SET_IFM_UPSCALE	IFM upscale mode
NPU_SET_IFM_WIDTH0_M1	IFM Tile 0 and Tile 2 width
NPU_SET_IFM_ZERO_POINT	IFM zero point
NPU_SET_KERNEL_HEIGHT_M1	Kernel height
NPU_SET_KERNEL_STRIDE	Kernel stride
NPU_SET_KERNEL_WIDTH_M1	Kernel width
NPU_SET_OFM_BLK_DEPTH_M1	OFM block depth
NPU_SET_OFM_BLK_HEIGHT_M1	OFM block height
NPU_SET_OFM_BLK_WIDTH_M1	OFM block width
NPU_SET_OFM_DEPTH_M1	Output feature map depth
NPU_SET_OFM_HEIGHT0_M1	OFM Tile 0 height
NPU_SET_OFM_HEIGHT1_M1	OFM Tile 1 height
NPU_SET_OFM_HEIGHT_M1	Output feature map height
NPU_SET_OFM_PRECISION	OFM Precision
NPU_SET_OFM_REGION	Index n for OFM access
NPU_SET_OFM_WIDTH0_M1	OFM Tile 0 and tile 2 width
NPU_SET_OFM_WIDTH_M1	Output feature map width
NPU_SET_OFM_ZERO_POINT	OFM zero point
NPU_SET_RESIZE_X_OFFSET	Set resize offset X
NPU_SET_RESIZE_X_SCALE_N_M1	Set resize scale X numerator
NPU_SET_RESIZE_Y_OFFSET	Set resize offset Y
NPU_SET_RESIZE_Y_SCALE_N_M1	Set resize scale Y numerator
NPU_SET_SCALE_REGION	Index n for scale stream access
NPU_SET_WEIGHT_FORMAT	Set weight stream format

Name	Description
NPU_SET_WEIGHT_REGION	Index n for weight stream access

3.10.1 Instruction encoding index for cmd0 stream

This section describes 32-bit NPU commands.

Commands in this section form part of the NPU command stream and execute on the NPU. Each command in this section consists of a single 32-bit word. Bits [31:16] of the word encode a 16-bit parameter, payload_0. Commands are 32-bit aligned in memory.

Figure 3-67: Instruction encoding index for cmd0 stream

Instructions	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
NPU_OP_STOP	MASK				0 0	Reserved		0 0	0 0	0 0	0 0	0 0
NPU_OP_IRQ	MASK				0 0	Reserved		0 0	0 0	0 0	0 0	0 1
NPU_OP_CONV	Reserved			WI	0 0	Reserved		0 0	0 0	0 0	0 0	1 0
NPU_OP_DEPTHWISE	Reserved				0 0	Reserved		0 0	0 0	0 0	0 0	1 1
NPU_OP_POOL	Reserved			mode	0 0	Reserved		0 0	0 0	0 0	0 1	0 1
NPU_OP_ELEMENTWISE	Reserved		mode		0 0	Reserved		0 0	0 0	0 0	1 1	1 0
NPU_OP_RESIZE	Reserved			RM	0 0	Reserved		0 0	0 0	0 0	1 1	1 1
NPU_OP_DMA_START	Reserved				0 0	Reserved		0 0	0 0	0 1	0 0	0 0
NPU_OP_DMA_WAIT	Reserved			k	0 0	Reserved		0 0	0 0	0 1	0 0	0 1
NPU_OP_KERNEL_WAIT	Reserved			n	0 0	Reserved		0 0	0 0	0 1	0 0	1 0
NPU_OP_PMU_MASK	Reserved			EN	0 0	Reserved		0 0	0 0	0 1	0 0	1 1
NPU_SET_IFM_PAD_TOP	Reserved		pad		0 0	Reserved		0 1	0 0	0 0	0 0	0 0
NPU_SET_IFM_PAD_LEFT	Reserved		pad		0 0	Reserved		0 1	0 0	0 0	0 0	0 1
NPU_SET_IFM_PAD_RIGHT	Reserved	pad			0 0	Reserved		0 1	0 0	0 0	0 0	1 0
NPU_SET_IFM_PAD_BOTTOM	Reserved	pad			0 0	Reserved		0 1	0 0	0 0	0 0	1 1
NPU_SET_IFM_DEPTH_M1	depth_m1				0 0	Reserved		0 1	0 0	0 0	0 1	0 0
NPU_SET_IFM_PRECISION	ST	Reserved	AF	Res AP R AT	0 0	Reserved		0 1	0 0	0 0	0 1	0 1
NPU_SET_IFM_UPSCALE	Reserved			MD	0 0	Reserved		0 1	0 0	0 0	0 1	1 1
NPU_SET_IFM_BROADCAST	Reserved			BM	0 0	Reserved		0 1	0 0	0 0	1 0	0 0
NPU_SET_IFM_ZERO_POINT	ZP				0 0	Reserved		0 1	0 0	0 0	1 0	0 1
NPU_SET_IFM_WIDTH0_M1	width_m1				0 0	Reserved		0 1	0 0	0 0	1 0	1 0
NPU_SET_IFM_HEIGHT0_M1	height_m1				0 0	Reserved		0 1	0 0	0 0	1 0	1 1
NPU_SET_IFM_HEIGHT1_M1	height_m1				0 0	Reserved		0 1	0 0	0 0	1 1	0 0
NPU_SET_IFM_REGION	Reserved			region	0 0	Reserved		0 1	0 0	0 0	1 1	1 1
NPU_SET_OFM_WIDTH_M1	width_m1				0 0	Reserved		0 1	0 0	0 1	0 0	0 1
NPU_SET_OFM_HEIGHT_M1	height_m1				0 0	Reserved		0 1	0 0	0 1	0 0	1 0
NPU_SET_OFM_DEPTH_M1	depth_m1				0 0	Reserved		0 1	0 0	0 1	0 0	1 1
NPU_SET_OFM_PRECISION	ST	TR	RV	SM AF	Reserved	AP AT		0 0	Reserved		0 1	0 0
NPU_SET_OFM_BLK_WIDTH_M1	Reserved		width_m1		0 0	Reserved		0 1	0 0	0 1	0 1	0 1
NPU_SET_OFM_BLK_HEIGHT_M1	Reserved		height_m1		0 0	Reserved		0 1	0 0	0 1	0 1	1 0
NPU_SET_OFM_BLK_DEPTH_M1	Reserved	depth_m1			0 0	Reserved		0 1	0 0	0 1	0 1	1 1
NPU_SET_OFM_ZERO_POINT	ZP				0 0	Reserved		0 1	0 0	0 1	1 0	0 0
NPU_SET_OFM_WIDTH0_M1	width_m1				0 0	Reserved		0 1	0 0	0 1	1 0	1 0
NPU_SET_OFM_HEIGHT0_M1	height_m1				0 0	Reserved		0 1	0 0	0 1	1 0	1 1
NPU_SET_OFM_HEIGHT1_M1	height_m1				0 0	Reserved		0 1	0 0	0 1	1 1	0 0
NPU_SET_OFM_REGION	Reserved			region	0 0	Reserved		0 1	0 0	0 1	1 1	1 1
NPU_SET_KERNEL_WIDTH_M1	width_m1				0 0	Reserved		0 1	0 0	1 0	0 0	0 0
NPU_SET_KERNEL_HEIGHT_M1	height_m1				0 0	Reserved		0 1	0 0	1 0	0 0	0 1
NPU_SET_KERNEL_STRIDE	Reserved	YM	Res	XM DE DY DX WO YL XL	0 0	Reserved		0 1	0 0	1 0	0 0	1 0
NPU_SET_ACC_FORMAT	Reserved	MB	R	AO AI Res AF	0 0	Reserved		0 1	0 0	1 0	0 1	0 0
NPU_SET_ACTIVATION	Reserved	AC	Reserved	table AF	0 0	Reserved		0 1	0 0	1 0	0 1	0 1
NPU_SET_ACTIVATION_MIN	CB				0 0	Reserved		0 1	0 0	1 0	0 1	1 0
NPU_SET_ACTIVATION_MAX	CB				0 0	Reserved		0 1	0 0	1 0	0 1	1 1
NPU_SET_WEIGHT_REGION	Reserved			region	0 0	Reserved		0 1	0 0	1 0	1 0	0 0
NPU_SET_SCALE_REGION	Reserved			region	0 0	Reserved		0 1	0 0	1 0	1 0	0 1
NPU_SET_RESIZE_X_SCALE_N_M1	Reserved	resize_x_scale_n_m1			0 0	Reserved		0 1	0 0	1 0	1 0	1 0
NPU_SET_RESIZE_Y_SCALE_N_M1	Reserved	resize_y_scale_n_m1			0 0	Reserved		0 1	0 0	1 0	1 0	1 1
NPU_SET_RESIZE_X_OFFSET	Reserved	resize_x_offset			0 0	Reserved		0 1	0 0	1 0	1 1	0 0
NPU_SET_RESIZE_Y_OFFSET	Reserved	resize_y_offset			0 0	Reserved		0 1	0 0	1 0	1 1	0 1
NPU_SET_WEIGHT_FORMAT	Reserved			WS Reserved WF	0 0	Reserved		0 1	0 0	1 0	1 1	1 0
NPU_SET_BLOCKDEP	Reserved			BD	0 0	Reserved		0 1	0 0	1 0	1 1	1 1
NPU_SET_DMA0_DST_REGION	Reserved	IM	Res	RM Reserved region	0 0	Reserved		0 1	0 0	1 1	0 0	0 1
NPU_SET_DMA0_SIZE0	size				0 0	Reserved		0 1	0 0	1 1	0 0	1 0
NPU_SET_DMA0_SIZE1	size				0 0	Reserved		0 1	0 0	1 1	0 0	1 1
NPU_SET_DMA0_IDX_REGION	Reserved			region	0 0	Reserved		0 1	0 0	1 1	0 1	0 0
NPU_SET_IFM2_BROADCAST	Reserved			BM	0 0	Reserved		0 1	1 0	0 0	0 0	0 0
NPU_SET_IFM2_PRECISION	ST	Reserved	AF	Res AP R AT	0 0	Reserved		0 1	1 0	0 0	0 1	0 1
NPU_SET_IFM2_ZERO_POINT	ZP				0 0	Reserved		0 1	1 0	0 0	1 0	0 1
NPU_SET_IFM2_WIDTH0_M1	width_m1				0 0	Reserved		0 1	1 0	0 0	1 0	1 0
NPU_SET_IFM2_HEIGHT0_M1	height_m1				0 0	Reserved		0 1	1 0	0 0	1 0	1 1
NPU_SET_IFM2_HEIGHT1_M1	height_m1				0 0	Reserved		0 1	1 0	0 0	1 1	0 0
NPU_SET_IFM2_REGION	Reserved			region	0 0	Reserved		0 1	1 0	0 0	1 1	1 1
NPU_SET_DMA0_SRC_REGION	Reserved	IM	SM	0 Reserved region	0 0	Reserved		0 1	0 0	1 1	0 0	0 0

3.10.2 NPU_OP_CONV instruction

2D convolution.

Starts a 2D convolution stripe operation.

Execution

```
// Pseudocode for 2D Convolution
dilated_height = NPU_SET_KERNEL_HEIGHT_M1.height_m1 + 1;  //
 (kernel_height-1)*dilation_y + 1
dilated_width  = NPU_SET_KERNEL_WIDTH_M1.width_m1 + 1;    // (kernel_width
 -1)*dilation_x + 1
ofm_height = NPU_SET_OFM_HEIGHT_M1.height_m1 + 1;
ofm_width  = NPU_SET_OFM_WIDTH_M1.width_m1 + 1;
ofm_depth = NPU_SET_OFM_DEPTH_M1.depth_m1 + 1
ifm_depth = NPU_SET_IFM_DEPTH_M1.depth_m1 + 1;
// strides are in the range 1 to 3
stride_y = 2*NPU_SET_KERNEL_STRIDE.stride_y_msb + NPU_SET_KERNEL_STRIDE.stride_y_lsb
 + 1;
stride_x = 2*NPU_SET_KERNEL_STRIDE.stride_x_msb + NPU_SET_KERNEL_STRIDE.stride_x_lsb
 + 1;
dilation_y = NPU_SET_KERNEL_STRIDE.dilation_y + 1; // dilation of 1 or 2
dilation_x = NPU_SET_KERNEL_STRIDE.dilation_x + 1; // dilation of 1 or 2
pad_y = NPU_SET_IFM_PAD_TOP.pad;
pad_x = NPU_SET_IFM_PAD_LEFT.pad;
pad_height = pad_y + NPU_SET_IFM_PAD_BOTTOM.pad;
pad_width  = pad_x + NPU_SET_IFM_PAD_RIGHT.pad;
ifm_height = (ofm_height-1)*stride_y + dilated_height - pad_height;
ifm_width  = (ofm_width -1)*stride_x + dilated_width  - pad_width;

use_ifm2 = (weights_ifm2 || NPU_SET_ACC_FORMAT.acc_input == ifm2);
add_operation_to_chain(true, use_ifm2, false);

assert(ifm_height >= 1 && ifm_width >= 1);      // check pad range
assert(dilated_height*dilated_width <= 64*64);  // check kernel range
assert(dilated_height <= 64 || dilated_width <= 64);
activation_precision_t precision = NPU_SET_IFM_PRECISION.activation_precision;
assert(precision == b8 ||
       (precision == b16 && NPU_SET_IFM_PRECISION.activation_type == signed));

if (weights_ifm2) {
    assert(dilated_width == 1 && dilated_height == 1);
    assert(ofm_height == 1);
    assert(pad_x == 0 && pad_y == 0);
    assert(dilation_x == 1 && dilation_y == 1);
    assert(stride_x == 1 && stride_y == 1);
    assert(NPU_SET_IFM_PRECISION.activation_precision
 == NPU_SET_IFM2_PRECISION.activation_precision);
    assert(NPU_SET_ACC_FORMAT.acc_input != ifm2);
    assert(NPU_SET_WEIGHT_FORMAT.weight_sparsity == none);
    assert(NPU_SET_KERNEL_STRIDE.weight_order == depth_first);
}
for (oy = 0; oy < ofm_height; oy++) {
    for (ox = 0; ox < ofm_width; ox++) {
        for (oc = 0; oc < ofm_depth; oc++) {
            int48_t acc = read_accumulator(oy, ox, oc, 0);
            for (ky = 0; ky*dilation_y < dilated_height; ky++) {
                for (kx = 0; kx*dilation_x < dilated_width; kx++) {
                    for (ic = 0; ic < ifm_depth; ic++) {
                        iy = -pad_y + oy*stride_y + ky*dilation_y;
                        ix = -pad_x + ox*stride_x + kx*dilation_x;
                        if (iy < 0 || iy >= ifm_height || ix < 0 || ix >= ifm_width)
 {
                            v = 0;
                        } else {
                            v = read_ifm(iy, ix, ic);
                        }
                        if (weights_ifm2) {
                            w = read_ifm2(0, ic, oc);
                        } else {
                            w = read_weight(oc,ky,kx,ic);
                        }
                        acc = accumulate(acc, v * w);
                    }
                }
```

```
            }
            write_accumulator(oy, ox, oc, acc, rescale_mode_normal);
        }
    }
}
```

Encoding

Figure 3-68: NPU_OP_CONV bit assignments

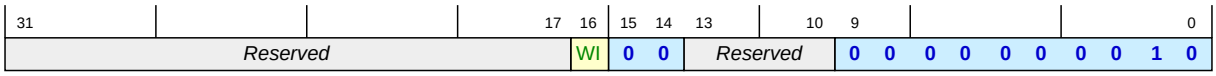


Table 3-95: Instruction NPU_OP_CONV encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_conv.	npu_op_conv
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	weights_ifm2 (WI)	This value is an unsigned integer.	-
[31:17]	Reserved	-	-

3.10.3 NPU_OP_DEPTHWISE instruction

Depth-wise 2D convolution.

Starts a depth-wise 2D convolution operation.

Execution

```
// Depthwise convolution
dilated_height = NPU_SET_KERNEL_HEIGHT_M1.height_m1 + 1;  //
 (kernel_height-1)*dilation_y + 1
dilated_width  = NPU_SET_KERNEL_WIDTH_M1.width_m1 + 1;    // (kernel_width
 -1)*dilation_x + 1
ofm_height = NPU_SET_OFM_HEIGHT_M1.height_m1 + 1;
ofm_width  = NPU_SET_OFM_WIDTH_M1.width_m1 + 1;
ofm_depth = NPU_SET_OFM_DEPTH_M1.depth_m1 + 1
// strides are in the range 1 to 3
stride_y = 2*NPU_SET_KERNEL_STRIDE.stride_y_msb + NPU_SET_KERNEL_STRIDE.stride_y_lsb
 + 1;
stride_x = 2*NPU_SET_KERNEL_STRIDE.stride_x_msb + NPU_SET_KERNEL_STRIDE.stride_x_lsb
 + 1;
dilation_y = NPU_SET_KERNEL_STRIDE.dilation_y + 1; // dilation of 1 or 2
dilation_x = NPU_SET_KERNEL_STRIDE.dilation_x + 1; // dilation of 1 or 2
pad_y = NPU_SET_IFM_PAD_TOP.pad;
pad_x = NPU_SET_IFM_PAD_LEFT.pad;
pad_height = pad_y + NPU_SET_IFM_PAD_BOTTOM.pad;
pad_width  = pad_x + NPU_SET_IFM_PAD_RIGHT.pad;
ifm_height = (ofm_height-1)*stride_y + dilated_height - pad_height;
ifm_width  = (ofm_width -1)*stride_x + dilated_width  - pad_width;
```

```
use_ifm2 = (NPU_SET_ACC_FORMAT.acc_input == ifm2);
add_operation_to_chain(true, use_ifm2, false);

assert(ifm_height >= 1 && ifm_width >= 1);      // check pad range
assert(dilated_height*dilated_width <= 64*64);  // check kernel range
assert(dilated_height <= 64 || dilated_width <= 64);
activation_precision_t precision = NPU_SET_IFM_PRECISION.activation_precision;
assert(precision == b8 ||
       (precision == b16 && NPU_SET_IFM_PRECISION.activation_type == signed));
assert(NPU_SET_WEIGHT_FORMAT.weight_format != fwd);
assert(NPU_SET_WEIGHT_FORMAT.weight_sparsity == none);

for (oy = 0; oy < ofm_height; oy++) {
    for (ox = 0; ox < ofm_width; ox++) {
        for (oc = 0; oc < ofm_depth; oc++) {
            int48_t acc = read_accumulator(oy, ox, oc, 0);
            for (ky = 0; ky*dilation_y < dilated_height; ky++) {
                for (kx = 0; kx*dilation_x < dilated_width; kx++) {
                    iy = -pad_y + oy*stride_y + ky*dilation_y;
                    ix = -pad_x + ox*stride_x + kx*dilation_x;
                    if (iy < 0 || iy >= ifm_height || ix < 0 || ix >= ifm_width) {
                        v = 0;
                    } else {
                        v = read_ifm(iy, ix, oc);
                    }
                    w = read_weight(oc, ky, kx, 0);
                    acc = accumulate(acc, v * w);
                }
            }
            write_accumulator(oy, ox, oc, acc, rescale_mode_normal);
        }
    }
}
```

Encoding

Figure 3-69: NPU_OP_DEPTHWISE bit assignments

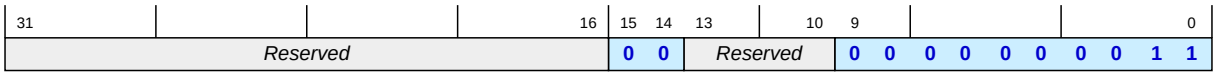


Table 3-96: Instruction NPU_OP_DEPTHWISE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_depthwise.	npu_op_depthwise
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	Reserved	-	-

3.10.4 NPU_OP_DMA_START instruction

Queue new DMA for the given channel.

This command blocks until the DMA channel can accept a new descriptor.

This command is viewed as complete once the DMA has been queued and does not need to wait for the DMA to complete. This command is different to other NPU_OP commands that must have their final results written to memory before they are considered complete.

Execution

```
while (!dma_free_descriptor(0)) {
  wait();   // wait for DMA0 to accept new operation
}

/* Code below is functional description of the memory copy operation */
src_region      = NPU_SET_DMA0_SRC_REGION.region;
src_region_mode = NPU_SET_DMA0_SRC_REGION.region_mode;
src_stride_mode = NPU_SET_DMA0_SRC_REGION.stride_mode;
src_idx_mode    = NPU_SET_DMA0_SRC_REGION.idx_mode;
dst_region      = NPU_SET_DMA0_DST_REGION.region;
dst_region_mode = NPU_SET_DMA0_DST_REGION.region_mode;
dst_idx_mode    = NPU_SET_DMA0_DST_REGION.idx_mode;
length          = (NPU_SET_DMA0_LEN.addr_hi<<32) + NPU_SET_DMA0_LEN.addr_lo;
width           = NPU_SET_DMA0_SIZE0.size;
height          = NPU_SET_DMA0_SIZE1.size;
src_p           = (NPU_SET_DMA0_SRC.addr_hi<<32) + NPU_SET_DMA0_SRC.addr_lo;
dst_p           = (NPU_SET_DMA0_DST.addr_hi<<32) + NPU_SET_DMA0_DST.addr_lo;
idx_p           = (NPU_SET_DMA0_IDX.addr_hi<<32) + NPU_SET_DMA0_IDX.addr_lo;
idx_region      = NPU_SET_DMA0_IDX_REGION.region;;
idx_max         = NPU_SET_DMA0_IDX_MAX.idx_max;
src_stride_0    = (NPU_SET_DMA0_SRC_STRIDE0.addr_hi<<32)
 + NPU_SET_DMA0_SRC_STRIDE0.addr_lo;
src_stride_1    = (NPU_SET_DMA0_SRC_STRIDE1.addr_hi<<32)
 + NPU_SET_DMA0_SRC_STRIDE1.addr_lo;
dst_stride_0    = (NPU_SET_DMA0_DST_STRIDE0.addr_hi<<32)
 + NPU_SET_DMA0_DST_STRIDE0.addr_lo;
dst_stride_1    = (NPU_SET_DMA0_DST_STRIDE1.addr_hi<<32)
 + NPU_SET_DMA0_DST_STRIDE1.addr_lo;
idx_skip_1      = (NPU_SET_DMA0_IDX_SKIP1.addr_hi<<32)
 + NPU_SET_DMA0_IDX_SKIP1.addr_lo;

// Check that gather and scatter are not active at the same time
assert(src_region_mode == external);
assert(!(src_idx_mode == enabled && dst_idx_mode == enabled));
if (dst_region_mode == internal) {
     dst_region = -1; // signal internal address as offset into LUT_RAM
}
if (src_stride_mode == d1) {
    // 1D strding (the same mode used for SRC and DST)
    // src_stride_1, dst_stride_1 and idx_skip_1 values are not used
    // src_stride_0 or dst_stride_0 is used only in src or dst index mode
    width = 1;  // width is forced to one in this case
}
if (src_stride_mode != d3) {
    // 1D or 2D strding (the same mode used for SRC and DST)
    // src_stride_1, dst_stride_1 and idx_skip_1 values are not used
    height = 1;  // height is forced to one in this case
}

for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        bool copy_valid = true;
        int32_t src_idx = x;
        int32_t dst_idx = x;
```

```
        if (src_idx_mode == enabled || dst_idx_mode == enabled) {
            int32_t idx = load_int32(idx_p, idx_region);
            idx_p += 4
            if (idx < 0 || idx > idx_max) {
                copy_valid = false;
            }
            if (src_idx_mode == enabled) {
                src_idx = idx;
            } else {
                dst_idx = idx;
            }
        }
        if (copy_valid) {
            src_addr = src_p + src_idx * src_stride_0;
            dst_addr = dst_p + dst_idx * dst_stride_0;
            /* Copy bytes from src_addr to dst_addr */
            copy_bytes(dst_addr, dst_region, src_addr, src_region, length);
        }
    }
    src_p += src_stride_1;
    dst_p += dst_stride_1;
    if (src_idx_mode == enabled || dst_idx_mode == enabled) {
        idx_p += idx_skip_1;
    }
}
```

Encoding

Figure 3-70: NPU_OP_DMA_START bit assignments

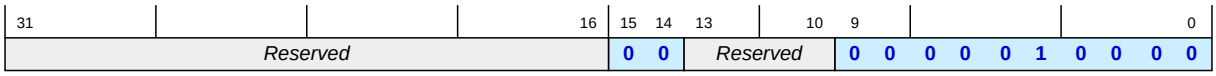


Table 3-97: Instruction NPU_OP_DMA_START encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_dma_start.	npu_op_dma_start
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	Reserved	-	-

3.10.5 NPU_OP_DMA_WAIT instruction

Wait for the mem2mem DMA channel to have k or fewer active descriptors outstanding.

Wait for the mem2mem DMA channel to have k or fewer outstanding descriptors. A descriptor is outstanding when it has been issued but is not complete. A descriptor is complete if the data is written to memory and can be read by the next command.

There are four mem2mem DMA descriptors and so 0 <= k <= 3.

Execution

```
/* Wait for mem2mem DMA completion */
while (number_outstanding_mem2mem_descriptos() > k) {
    /* wait for mem2mem DMA descriptor to complete */
}
```

Encoding

Figure 3-71: NPU_OP_DMA_WAIT bit assignments

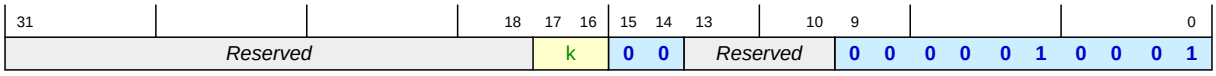


Table 3-98: Instruction NPU_OP_DMA_WAIT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_dma_wait.	npu_op_dma_wait
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[17:16]	k	This value is an unsigned integer.	-
[31:18]	Reserved	-	-

3.10.6 NPU_OP_ELEMENTWISE instruction

Elementwise operation.

Starts an elementwise stripe operation.

Execution

```
// psuedocode for elementwise operations
ofm_height = NPU_SET_OFM_HEIGHT_M1.height_m1 + 1;
ofm_width  = NPU_SET_OFM_WIDTH_M1.width_m1 + 1;
ofm_depth = NPU_SET_OFM_DEPTH_M1.depth_m1 + 1
assert(NPU_SET_IFM_PRECISION.activation_precision
 == NPU_SET_IFM2_PRECISION.activation_precision);

unary = (elementwise_mode == lrelu) ||
        (elementwise_mode == abs) ||
        (elementwise_mode == clz) ||
        (elementwise_mode == not);
use_ifm2 = !(unary || NPU_SET_IFM2_BROADCAST.broadcast_mode == scalar);
add_operation_to_chain(true, use_ifm2, true);

activation_precision_t precision = NPU_SET_IFM_PRECISION.activation_precision;
activation_type_t act_type = NPU_SET_IFM_PRECISION.activation_type;
assert(precision == b8 || precision == b16 ||
       (precision == b32 && act_type == signed));
assert(NPU_SET_OFM_PRECISION.activation_reverse == none);
assert(NPU_SET_OFM_PRECISION.activation_transpose == HWC ||
```

```
       NPU_SET_OFM_PRECISION.activation_transpose == HCW ||
       NPU_SET_OFM_PRECISION.activation_transpose == CHW;
assert(NPU_SET_IFM_UPSCALE.mode == none);

for (oy = 0; oy < ofm_height; oy++) {
    for (ox = 0; ox < ofm_width; ox++) {
        for (oc = 0; oc < ofm_depth; oc++) {
            int32_t acc;
            if (NPU_SET_IFM_BROADCAST.broadcast_mode == scalar) {
                assert(NPU_SET_IFM_PRECISION.activation_storage == none);
                a = NPU_SET_OP_SCALAR.scalar;
            } else {

  check_broadcast_permitted(NPU_SET_IFM_BROADCAST.broadcast_mode, NPU_SET_IFM_PRECISION.activatio
                iy = (NPU_SET_IFM_BROADCAST.broadcast_mode & h) ? 0 : oy;
                ix = (NPU_SET_IFM_BROADCAST.broadcast_mode & w) ? 0 : ox;
                ic = (NPU_SET_IFM_BROADCAST.broadcast_mode & c) ? 0 : oc;
                a = read_ifm(iy,ix,ic);
            }
            if (!unary) {
                if (NPU_SET_IFM2_BROADCAST.broadcast_mode == scalar) {
                    assert(NPU_SET_IFM_BROADCAST.broadcast_mode != scalar)
                    assert(NPU_SET_IFM2_PRECISION.activation_storage == none);
                    b = NPU_SET_OP_SCALAR.scalar;
                } else {

  check_broadcast_permitted(NPU_SET_IFM2_BROADCAST.broadcast_mode, NPU_SET_IFM2_PRECISION.activat
                    iy = (NPU_SET_IFM2_BROADCAST.broadcast_mode & h) ? 0 : oy;
                    ix = (NPU_SET_IFM2_BROADCAST.broadcast_mode & w) ? 0 : ox;
                    ic = (NPU_SET_IFM2_BROADCAST.broadcast_mode & c) ? 0 : oc;
                    b = read_ifm2(iy,ix,ic);
                }
            }
            rescale_mode = rescale_mode_global;
            switch (elementwise_mode) {
                case mul:
                    // uint16 elementwise multiply must have a zero point of 32768
                    int32_t zero_point = NPU_SET_IFM_ZERO_POINT.zero_point;
                    int32_t zero_point2 = NPU_SET_IFM2_ZERO_POINT.zero_point;
                    assert(precision == b8 || act_type == signed ||
                           (zero_point == 32768 && zero_point2 == 32768));
                    acc = a * b;
                    if (!(precision == b8 || precision == b16)) {
                        rescale_mode = rescale_mode_shift_only;
                    }
                    break;
                case add:
                    acc = scale_a(a) + scale_b(b); break;
                case sub:
                    acc = scale_a(a) - scale_b(b); break;
                case min:
                    acc = min(scale_a(a), scale_b(b)); break;
                case max:
                    acc = max(scale_a(a), scale_b(b)); break;
                case lrelu:
                    acc = (a >= 0) ? scale_a(a) : scale_b(a); break;
                case abs:
                    acc = abs(scale_a(a)); break;
                case clz:
                    acc = clz(scale_a(a), 32); break;
                case shr:
                    // arithmetic shift right with configurable round
                    acc = scale_a(a);
                    shift = scale_b(b) & 31;
                    mode = NPU_SET_OFM_SCALE.round_mode;
                    dbl_rnd = NPU_SET_OFM_SCALE.dbl_rnd;
                    acc = rounded_shift_right(acc, shift, mode, dbl_rnd);
                    rescale_mode = rescale_mode_none; break;
                case shl:
                    // left shift
                    acc = scale_a(a) << (scale_b(b) & 31);
```

```
                    acc = acc & ACTIVATION_MASK;
                    rescale_mode = rescale_mode_none; break;
                case lsr:
                    // logical shift right (no round)
                    acc = scale_a(a);
                    shift = scale_b(b) & 31;
                    switch (NPU_SET_IFM_PRECISION.activation_precision) {
                        case b8: acc=acc & 0xFF; break;
                        case b16: acc=acc & 0xFFFF; break;
                    }
                    acc = (uint32_t)acc >> shift;
                    rescale_mode = rescale_mode_none; break;
                case div:
                    assert(NPU_SET_IFM_SCALE.scale == 1 && NPU_SET_IFM_SCALE.shift
 == 0);
                    assert(NPU_SET_IFM_ZERO_POINT.zero_point == 0);
                    assert(NPU_SET_IFM2_SCALE.scale == 1 && NPU_SET_IFM2_SCALE.shift
 == 0);
                    assert(NPU_SET_IFM2_ZERO_POINT.zero_point == 0);
                    assert(b!=0);
                    assert(!(b==-1 && a==-0x80000000));
                    acc = a/b;  // truncate towards zero integer divide
                    assert(NPU_SET_OFM_SCALE.scale == 1 && NPU_SET_OFM_SCALE.shift
 == 0);
                    assert(NPU_SET_OFM_ZERO_POINT.zero_point == 0);
                    break;
                case cmp_eq:
                    acc = (scale_a(a) == scale_b(b)) ? -1 : 0; break;
                case cmp_ne:
                    acc = (scale_a(a) != scale_b(b)) ? -1 : 0; break;
                case cmp_ge:
                    acc = (scale_a(a) >= scale_b(b)) ? -1 : 0; break;
                case cmp_gt:
                    acc = (scale_a(a) >  scale_b(b)) ? -1 : 0; break;
                case not:
                    acc = ~ scale_a(a); break;
                case and:
                    acc = scale_a(a) & scale_b(b); break;
                case or:
                    acc = scale_a(a) | scale_b(b); break;
                case xor:
                    acc = scale_a(a) ^ scale_b(b); break;
                case and_not:
                    acc = scale_a(a) & ( ~ scale_b(b) ); break;
            }
            // output scaling, per-channel scale is not permitted
            acc = rescale(acc, -1, rescale_mode);
            acc = activation(acc);
            write_ofm(oy, ox, oc, acc, allowed_perms_rtrans);
        }
    }
}
```

Encoding

Figure 3-72: NPU_OP_ELEMENTWISE bit assignments

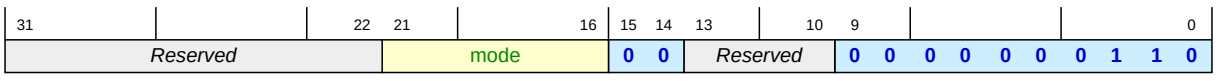


Table 3-99: Instruction NPU_OP_ELEMENTWISE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_elementwise.	npu_op_elementwise
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[21:16]	elementwise_mode (mode)	This value is an enumeration of type elementwise_mode_t. This field can contain the values shown in table elementwise_mode	-
[31:22]	Reserved	-	-

Table 3-100: Field elementwise_mode values

Value	Name	Description
0b000000	mul	Elementwise multiplication
0b000001	add	Elementwise addition
0b000010	sub	Elementwise subtraction
0b000011	min	Elementwise minimum
0b000100	max	Elementwise maximum
0b000101	lrelu	Elementwise Leaky ReLU
0b000110	abs	Elementwise absolute value
0b000111	clz	Elementwise count leading zeros
0b001000	shr	Elementwise arithmetic shift right
0b001001	shl	Elementwise shift left
0b001010	lsr	Elementwise logical shift right
0b001011	div	Elementwise divide
0b001100..0b001111	Reserved	-
0b010000	cmp_eq	Elementwise compare equal
0b010001	cmp_ne	Elementwise compare not equal
0b010010	cmp_ge	Elementwise compare greater or equal
0b010011	cmp_gt	Elementwise compare greater than
0b010100..0b100000	Reserved	-
0b100001	and	Elementwise bitwise AND
0b100010	or	Elementwise bitwise OR
0b100011	xor	Elementwise bitwise XOR
0b100100	not	Elementwise bitwise NOT
0b100101..0b101001	Reserved	-
0b101010	and_not	Elementwise bitwise AND_NOT
0b101011..0b111111	Reserved	-

3.10.7 NPU_OP_IRQ instruction

Raises an IRQ to the host.

Raises an IRQ. The NPU signals IRQ to the host when all previous commands are complete. Commands after the IRQ are permitted to start before the IRQ is raised.

This command is UNPREDICTABLE if used within a chain.

Execution

```
/* Raise an IRQ once all preceding commands are complete */
assert(g_chain_length == 0);
while (!preceding_commands_complete()) {
   // continue running - following commands may run
}
STATUS.irq_raised = 1;
STATUS.irq_history_mask |= mask;
raise_host_irq();
```

Encoding

Figure 3-73: NPU_OP_IRQ bit assignments

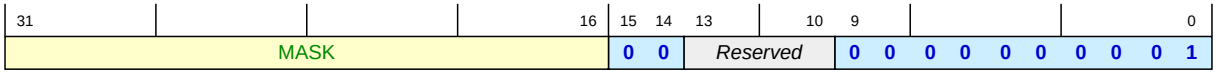


Table 3-101: Instruction NPU_OP_IRQ encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_irq.	npu_op_irq
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	mask (MASK)	Encodes a 16-bit mask that will eventually be or'ed into irq_history_mask This value is a bitset. The nth bit in the bitset (n=0-15) has the following meaning: **0** Bit mask low **1** Bit mask high	-

3.10.8 NPU_OP_KERNEL_WAIT instruction

Wait for kernel operations to complete.

Wait for n or fewer kernel operations to be remaining before starting the next command. A kernel operation is remaining if it has been issued but is not yet complete. A kernel operation is complete if all result data has been written to memory. This command is typically placed before an NPU_OP_DMA_START command to prevent the DMA from writing to memory that a previous kernel operation is still reading.

A kernel operation is one of CONV, DEPTHWISE, ELEMENTWISE, POOL, RESIZE together with operations in the same chain. Thus a chain counts as a single outstanding operation.

- A value of n=0 indicates that this command waits for all previous kernel operations be complete.
- A value of n=1 indicates that this command waits for all but the previous kernel operation (which may be a chain) to be complete. Multiple commands may be outstanding in the case of a chain.

In this version of the architecture, when a command starts there are at most two kernel operations remaining.

Execution

```
/* Wait for kernel operation completion */
/* Note: A chain counts as a single kernel operation. */
while(number_of_remaining_kernel_operations() > n) {
    /* wait for kernel operation to complete */
}
```

Encoding

Figure 3-74: NPU_OP_KERNEL_WAIT bit assignments

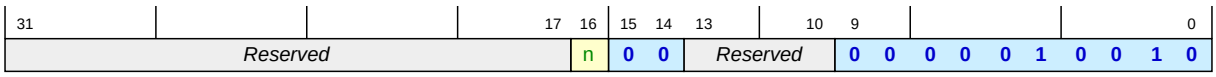


Table 3-102: Instruction NPU_OP_KERNEL_WAIT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_kernel_wait.	npu_op_kernel_wait
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	n	This value is an unsigned integer.	-
[31:17]	Reserved	-	-

3.10.9 NPU_OP_PMU_MASK instruction

Enable or disable PMU counting (debug feature only).

Enable or disable PMU counting (debug feature only).

Execution

```
if (enable == true) {
    enable_pmu_counters()
} else {
    disable_pmu_counters()
}
```

Encoding

Figure 3-75: NPU_OP_PMU_MASK bit assignments

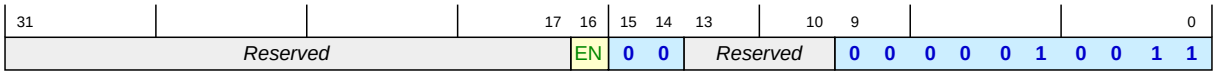


Table 3-103: Instruction NPU_OP_PMU_MASK encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_pmu_mask.	npu_op_pmu_mask
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	enable (EN)	This value is an unsigned integer.	-
[31:17]	Reserved	-	-

3.10.10 NPU_OP_POOL instruction

Pooling.

Starts a pooling operation.

Note that pooling_mode == none cannot be chained.

Execution

```
// pseudocode for 2D pooling
kernel_height = NPU_SET_KERNEL_HEIGHT_M1.height_m1 + 1;  // non dilated height
kernel_width  = NPU_SET_KERNEL_WIDTH_M1.width_m1 + 1;    // non dilated width
ofm_height = NPU_SET_OFM_HEIGHT_M1.height_m1 + 1;
ofm_width  = NPU_SET_OFM_WIDTH_M1.width_m1 + 1;
ofm_depth = NPU_SET_OFM_DEPTH_M1.depth_m1 + 1;
```

```
// strides are in the range 1 to 3
stride_y = 2*NPU_SET_KERNEL_STRIDE.stride_y_msb + NPU_SET_KERNEL_STRIDE.stride_y_lsb
 + 1;
stride_x = 2*NPU_SET_KERNEL_STRIDE.stride_x_msb + NPU_SET_KERNEL_STRIDE.stride_x_lsb
 + 1;
pad_y = NPU_SET_IFM_PAD_TOP.pad;
pad_x = NPU_SET_IFM_PAD_LEFT.pad;
pad_height = pad_y + NPU_SET_IFM_PAD_BOTTOM.pad;
pad_width  = pad_x + NPU_SET_IFM_PAD_RIGHT.pad;
ifm_height = (ofm_height-1)*stride_y + kernel_height - pad_height;
ifm_width  = (ofm_width -1)*stride_x + kernel_width  - pad_width;
assert(NPU_SET_KERNEL_STRIDE.dilation_y + 1 == 1);
assert(NPU_SET_KERNEL_STRIDE.dilation_x + 1 == 1);

use_ifm  = (pooling_mode != none);
use_ifm2 = (NPU_SET_ACC_FORMAT.acc_input == ifm2);
add_operation_to_chain(use_ifm, use_ifm2, false);
if (pooling_mode == none) {
    // A pooling mode of none cannot be chained
    assert(NPU_SET_OFM_PRECISION.activation_storage != chained);
}
assert(ifm_height >= 1 && ifm_width >= 1); // check pad range

// Check IFM precision
activation_precision_t precision = NPU_SET_IFM_PRECISION.activation_precision;
assert(precision == b8 ||
       (precision == b16 && NPU_SET_IFM_PRECISION.activation_type == signed) ||
       (precision == b32 && NPU_SET_IFM_PRECISION.activation_type == signed));

// Check IFM precision and kernel size for each mode
switch (pooling_mode) {
    case max:
    case min:
        assert(kernel_height*kernel_width <= 65536);
        assert(kernel_height <= 256 || kernel_width <= 256);
        assert(NPU_SET_IFM_UPSCALE.mode != zeros);
        break;
    case average:
        assert(kernel_height <= 8 && kernel_width <= 8);
        assert(max(NPU_SET_IFM_PAD_TOP.pad,  NPU_SET_IFM_PAD_BOTTOM.pad) <
 kernel_height);
        assert(max(NPU_SET_IFM_PAD_LEFT.pad, NPU_SET_IFM_PAD_RIGHT.pad ) <
 kernel_width);
        assert(NPU_SET_KERNEL_STRIDE.decomposition == d8x8);
        assert(NPU_SET_IFM_PRECISION.activation_precision != b32);
        break;
    case reduce_sum:
        assert(kernel_height == 1 && kernel_width == 1);
        assert(stride_x == 1 && stride_y == 1);
        assert(NPU_SET_IFM_UPSCALE.mode == none);
        assert(NPU_SET_KERNEL_STRIDE.weight_order == depth_first);
        assert(NPU_SET_IFM_ZERO_POINT.zero_point == 0);
        break;
    case sum:
        assert(NPU_SET_IFM_PRECISION.activation_precision != b32);
        assert(kernel_height <= 65536);
        assert(kernel_width  <= 65536);
        break;
    case argmax_x:
        assert(NPU_SET_IFM_PRECISION.activation_precision != b32);
        assert(NPU_SET_KERNEL_STRIDE.decomposition == d8x8);
        assert(kernel_height == 1);
        assert(kernel_width <= 65536);
        assert(stride_x == 1 && stride_y == 1);
        assert(ofm_width == 1);
        assert(NPU_SET_IFM_UPSCALE.mode == none);
        break;
    case argmax_y:
        assert(NPU_SET_IFM_PRECISION.activation_precision != b32);
        assert(NPU_SET_KERNEL_STRIDE.decomposition == d8x8);
        assert(kernel_height <= 65536);
```

```
        assert(kernel_width == 1);
        assert(ofm_height == 1);
        assert(stride_x == 1 && stride_y == 1);
        assert(NPU_SET_IFM_UPSCALE.mode == none);
        break;
}

// Accumulator format must be 32-bit except for sum, reduce_sum or null pooling
 modes
assert(pooling_mode == sum ||
       pooling_mode == reduce_sum ||
       pooling_mode == none ||
       NPU_SET_ACC_FORMAT.acc_format == i32);

for (oy = 0; oy < ofm_height; oy++) {
    for (ox = 0; ox < ofm_width; ox++) {
        if (pooling_mode == reduce_sum) {
            // Reduce sum in channel direction
            assert(ofm_depth == 1);
            int ifm_depth = NPU_SET_IFM_DEPTH_M1.depth_m1 + 1;
            int48_t acc = read_accumulator(oy, ox, 0, 0);
            for (c=0; c < ifm_depth; c++) {
                v = read_ifm(oy, ox, c);
                acc = accumulate(acc, v);
            }
            write_accumulator(oy, ox, 0, acc, rescale_mode_normal);
        } else {
            // Depthwise pooling operations
            for (c = 0; c < ofm_depth; c++) {
                int32_t identity = 0;
                if (pooling_mode == max ||
                    pooling_mode == argmax_x ||
                    pooling_mode == argmax_y) {
                    identity = -(1<<31);
                } else if (pooling_mode == min) {
                    identity = (1<<31)-1;
                }
                int48_t acc = read_accumulator(oy, ox, c, identity);
                int count = 0;      // only used by average pool
                for (ky = 0; ky < kernel_height; ky++) {
                    for (kx = 0; kx < kernel_width; kx++) {
                        iy = -pad_y + oy*stride_y + ky;
                        ix = -pad_x + ox*stride_x + kx;
                        bool pad = (iy < 0 || iy >= ifm_height || ix < 0 || ix >=
 ifm_width);
                        if  (!pad) count++;         // count valid inputs
                        // Note that the effective pad value is the identity
                        // value which is not 0 for maximum and minimum pooling
                        int32_t v = identity;
                        if (use_ifm && !pad) {
                            v = read_ifm(iy, ix, c);
                        }
                        switch (pooling_mode) {
                            case max:
                                acc = max(acc, v); break;
                            case average:  // fall through
                            case sum:
                                acc = accumulate(acc, v); break;
                            case none:
                                // accumulator values is not altered
                                // but keeps read_accumulator() value;
                                break;
                            case min:
                                acc = min(acc, v); break;
                            case argmax_x:
                                if (v > sign_extend(acc, 16)) {
                                    acc = v + 65536*kx; // new max and position
                                }
                                break;
                            case argmax_y:
                                if (v > sign_extend(acc, 16)) {
```

```
                                    acc = v + 65536*ky; // new max and position
                                }
                                break;
                            }
                        }
                    }
                    rescale_mode = rescale_mode_normal;
                    if (pooling_mode == average) {
                        rescale_mode = rescale_mode_avpool + (count-1);
                    }
                    write_accumulator(oy, ox, oc, acc, rescale_mode);
                }
            }
        }
    }
```

Encoding

Figure 3-76: NPU_OP_POOL bit assignments

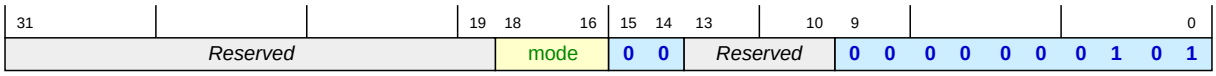


Table 3-104: Instruction NPU_OP_POOL encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_pool.	npu_op_pool
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[18:16]	pooling_mode (mode)	This value is an enumeration of type pooling_mode_t. This field can contain the following values: **0b000** max - Max pooling **0b001** average - Average pooling up to 8x8, inbuilt scale only **0b010** reduce_sum - Sum reduction **0b011** sum - Sum or average pooling, per-channel, or global scale **0b100** none - Null pooling **0b101** min - Min pooling **0b110** argmax_x - Argmax pooling - return x index of maximum **0b111** argmax_y - Argmax pooling - return y index of maximum	-
[31:19]	Reserved	-	-

3.10.11 NPU_OP_RESIZE instruction

Resize operation.

Starts a tensor resize operation.


Note

To implement a resize with nearest sample position use the replicate mode with X and Y *offset* values increased by *scale_n*/2

Execution

```
// psuedocode for resize operations
uint16_t ofm_height_m1 = NPU_SET_OFM_HEIGHT_M1.height_m1;
uint16_t ofm_width_m1  = NPU_SET_OFM_WIDTH_M1.width_m1;
uint16_t ofm_depth_m1  = NPU_SET_OFM_DEPTH_M1.depth_m1;
uint16_t ifm_height_m1 = NPU_SET_KERNEL_HEIGHT_M1.height_m1;
uint16_t ifm_width_m1  = NPU_SET_KERNEL_WIDTH_M1.width_m1;
uint12_t scale_y_n     = NPU_SET_RESIZE_Y_SCALE_N_M1.resize_y_scale_n_m1 + 1;
uint12_t scale_x_n     = NPU_SET_RESIZE_X_SCALE_N_M1.resize_x_scale_n_m1 + 1;
int12_t  y_off         = NPU_SET_RESIZE_Y_OFFSET.resize_y_offset;
int12_t  x_off         = NPU_SET_RESIZE_X_OFFSET.resize_x_offset;
uint11_t step_y_mod    = NPU_SET_RESIZE_Y_STEP.one_step_mod;
uint11_t step_x_mod    = NPU_SET_RESIZE_X_STEP.one_step_mod;
uint4_t  step_y_int    = NPU_SET_RESIZE_Y_STEP.one_step_int;
uint4_t  step_x_int    = NPU_SET_RESIZE_X_STEP.one_step_int;
```

```
assert(-scale_y_n <= y_off && y_off < scale_y_n);
assert(-scale_x_n <= x_off && x_off < scale_x_n);
assert(step_y_mod < scale_y_n);
assert(step_x_mod < scale_x_n);

// chaining and transpose not permitted
assert(g_chain_length==0);
assert(NPU_SET_OFM_PRECISION.activation_storage != chained);
assert(NPU_SET_OFM_PRECISION.activation_transpose == HWC);
();

assert(NPU_SET_IFM_UPSCALE.mode == none);
assert(NPU_SET_IFM_ZERO_POINT.zero_point == 0);
assert(NPU_SET_IFM2_ZERO_POINT.zero_point == 0);

if (NPU_SET_IFM_PRECISION.activation_precision == b8) {
    // uint8 or int8 input tensor
    acc_t = int31_t;
} else { // must be int16 input tensor
    assert(NPU_SET_IFM_PRECISION.activation_precision == b16)
    assert(NPU_SET_IFM_PRECISION.activation_type == signed)
    acc_t = int39_t;
}

int16_t  y_int = (y_off < 0) ? -1 : 0;
uint12_t y_mod = (y_off < 0) ? (y_off + scale_y_n) : y_off;
for (oy = 0; oy <= ofm_height_m1; oy++) {
    uint16_t iy0 = max(y_int, 0);
    uint16_t iy1 = min(y_int + 1, ifm_height_m1);
    assert(iy0 <= ifm_height_m1);
    int16_t  x_int = (x_off < 0) ? -1 : 0;
    uint12_t x_mod = (x_off < 0) ? (x_off + scale_x_n) : x_off;
    for (ox = 0; ox <= ofm_width_m1; ox++) {
        uint16_t ix0 = max(x_int, 0);
        uint16_t ix1 = min(x_int + 1, ifm_width_m1);
        assert(ix0 <= ifm_width_m1);
        assert(ox < ofm_width_m1 || ix1 == ifm_width_m1);
        for (oc = 0; oc <= ofm_depth_m1; oc++) {
            acc_t acc;
            v0 = read_ifm(iy0, ix0, oc);
            if (resize_mode == bilinear) {
                v1 = read_ifm(iy0, ix1, oc);
                v2 = read_ifm(iy1, ix0, oc);
                v3 = read_ifm(iy1, ix1, oc);
                acc = (scale_y_n - y_mod)*((scale_x_n - x_mod)*v0 + x_mod*v1) +
                                  y_mod *((scale_x_n - x_mod)*v2 + x_mod*v3);
            } else if (resize_mode == nearest) {
                uint16_t ix = (2*x_mod < scale_x_n) ? ix0 : ix1;
                uint16_t iy = (2*y_mod < scale_y_n) ? iy0 : iy1;
                acc = read_ifm(iy, ix, oc);
            } else { // resize_mode == replicate
                acc = v0;
            }
            // Rescale, only the right shift is supported and shift
            // must be 16 or more
            int64_t acc2 = (int64_t)acc << 16;
            assert(NPU_SET_OFM_SCALE.shift >= 16);
            acc2 = rescale(acc2, -1, rescale_mode_shift_only);
            acc2 = activation(acc2);
            write_ofm(oy, ox, oc, acc2);
        }
        x_mod += step_x_mod;
        x_int += step_x_int;
        if (x_mod >= scale_x_n) {
            x_mod -= scale_x_n;
            x_int++;
        }
    }
    y_mod += step_y_mod;
    y_int += step_y_int;
    if (y_mod >= scale_y_n) {
```

```
            y_mod -= scale_y_n;
            y_int++;
        }
    }
```

Encoding

Figure 3-77: NPU_OP_RESIZE bit assignments

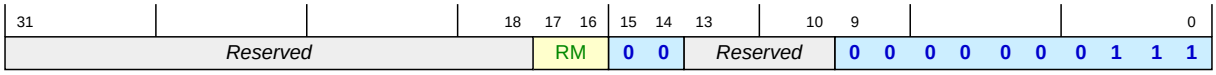


Table 3-105: Instruction NPU_OP_RESIZE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_resize.	npu_op_resize
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[17:16]	resize_mode (RM)	This value is an enumeration of type resize_mode_t. This field can contain the following values: **0b00** bilinear - Resize with bilinear interpolation **0b01** replicate - Resize with replicate of the last integer position **0b10** nearest - Resize from nearest integer position **0b11** Reserved	-
[31:18]	Reserved	-	-

3.10.12 NPU_OP_STOP instruction

Signal the end of command stream.

Complete any outstanding operations and then transition to STOPPED state. The NPU signals IRQ to the host when the transition is complete.

This command is UNPREDICTABLE if used within a chain.

Execution

```
/* Stop processing once all preceding commands are complete */
assert(g_chain_length == 0);
while (!preceding_commands_complete()) {
   // continue running - following commands may not start
```

```
}
STATUS.state = stopped;
STATUS.irq_raised = 1;
STATUS.irq_history_mask |= mask;
raise_host_irq();
```

Encoding

Figure 3-78: NPU_OP_STOP bit assignments

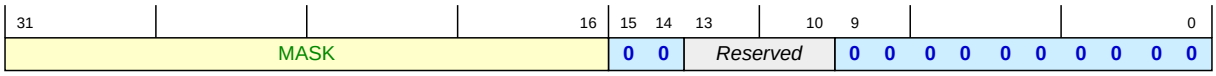


Table 3-106: Instruction NPU_OP_STOP encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_op_stop.	npu_op_stop
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	mask (MASK)	Encodes a 16-bit mask that will eventually be or'ed into irq_history_mask This value is a bitset. The *n*th bit in the bitset (*n*=0-15) has the following meaning: **0** Bit mask low **1** Bit mask high	-

3.10.13 NPU_SET_ACC_FORMAT instruction

Accumulator format.

Set Accumulator format and microblock size.

This command can only be used for the first operation of a chain and must not be used within a chain.

Execution

```
/* Set accumulator format and microblock size */
    assert(g_chain_length==0);      // cannot be used within a chain
    set_register();                 // set register contents
```

Encoding

Figure 3-79: NPU_SET_ACC_FORMAT bit assignments

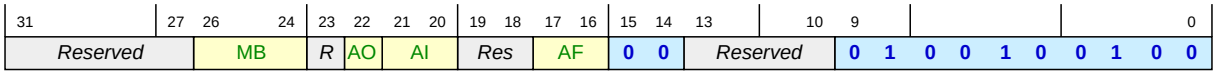


Table 3-107: Instruction NPU_SET_ACC_FORMAT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_acc_format.	npu_set_acc_format
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[17:16]	acc_format (AF)	This value is an enumeration of type acc_format_t. This field can contain the following values: **0b00** i32 - 32-bit integer **0b01** i48 - 48-bit integer **0b10..0b11** Reserved	-
[19:18]	Reserved	-	-
[21:20]	acc_input (AI)	This value is an enumeration of type acc_input_t. This field can contain the following values: **0b00** reset - Reset accumulator before the operation (reset value depends on the operation) **0b01** keep - Accumulator on to value from previous operation **0b10** ifm2 - Restore the accumulator from the IFM2 input stream **0b11** Reserved	-
[22]	acc_output (AO)	This value is an enumeration of type acc_output_t. This field can contain the following values: **0b0** enable - Output accumulator result for scaling and storage **0b1** disable - Do not output accumulator, keep for next operation	-
[23]	Reserved	-	-

Bits	Name	Description	Reset
[26:24]	microblock (MB)	The available microblock size depends on the NPU configuration CONFIG.macs_per_cc and the operation. This value is an enumeration of type microblock_t. This field can contain the following values: **0b000** u1x1 - Microblock H=1 W=1 C=macs_per_cc/8, macs_per_cc=128 only **0b001** u1x2 - Microblock H=1 W=2 C=macs_per_cc/16, macs_per_cc=128 or 256 **0b010** u1x4 - Microblock H=1 W=4 C=macs_per_cc/32, macs_per_cc>=256 **0b011** u2x2 - Microblock H=2 W=2 C=macs_per_cc/32, macs_per_cc>=256 **0b100** u2x4 - Microblock H=2 W=4 C=macs_per_cc/64, macs_per_cc=1024 only **0b101** u4x4 - Microblock H=4 W=4 C=macs_per_cc/128, macs_per_cc=2048 only **0b110..0b111** Reserved	-
[31:27]	Reserved	-	-

3.10.14 NPU_SET_ACTIVATION instruction

Activation function and clip range.

Set activation function and clip range.

Execution

```
set_register()
```

Encoding

Figure 3-80: NPU_SET_ACTIVATION bit assignments

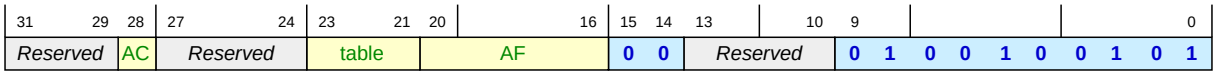


Table 3-108: Instruction NPU_SET_ACTIVATION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_activation.	npu_set_activation

Bits	Name	Description	Reset
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[20:16]	activation_function (AF)	This value is an enumeration of type activation_function_t. This field can contain the following values: **0b00000** lut_none - No additional activation **0b00001** lut_u8_u8 - Lookup table u8->u8 **0b00010..0b00011** Reserved **0b00100** lut_s8_s8 - Lookup table s8->s8 **0b00101** lut_s8_s16 - Lookup table s8->s16 **0b00110** Reserved **0b00111** lut_s8_s32 - Lookup table s8->s32 **0b01000** lut_s16_s16 - Lookup table s16->s16 interpolate **0b01001** lut_s16_s32 - Lookup table s16->16.7 interpolate **0b01010** lut_tanh - Lookup table s18->tanh lookup **0b01011** lut_sigmoid - Lookup table s18->sigmoid lookup **0b01100..0b11111** Reserved	-
[23:21]	table	This value is an unsigned integer.	-
[27:24]	Reserved	-	-
[28]	activation_clip_range (AC)	This value is an enumeration of type activation_clip_range_t. This field can contain the following values: **0b0** b16 - 16-bit activation min and max applied **0b1** none - No activation clip applied	-
[31:29]	Reserved	-	-

3.10.15 NPU_SET_ACTIVATION_MAX instruction

Upper bound clip.

Set upper bound clip for OFM activations – range is given by activation clip range

Execution

```
set_register()
```

Encoding

Figure 3-81: NPU_SET_ACTIVATION_MAX bit assignments

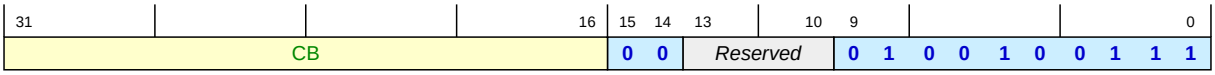


Table 3-109: Instruction NPU_SET_ACTIVATION_MAX encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_activation_max.	npu_set_activation_max
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	clip_boundary (CB)	Clip boundary for OFM activations – range is given by the activation clip range This value is an untyped value.	-

3.10.16 NPU_SET_ACTIVATION_MIN instruction

Lower bound clip.

Set lower bound clip for OFM activations – range is given by activation clip range

Execution

```
set_register()
```

Encoding

Figure 3-82: NPU_SET_ACTIVATION_MIN bit assignments

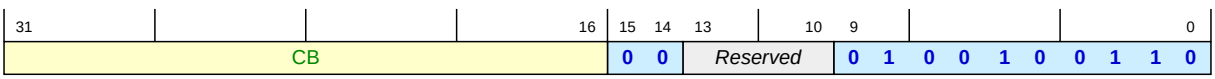


Table 3-110: Instruction NPU_SET_ACTIVATION_MIN encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_activation_min.	npu_set_activation_min
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	clip_boundary (CB)	Clip boundary for OFM activations – range is given by the activation clip range This value is an untyped value.	-

3.10.17 NPU_SET_BLOCKDEP instruction

Block dependency offset for kernel operations.

Execution

```
set_register()
```

Encoding

Figure 3-83: NPU_SET_BLOCKDEP bit assignments

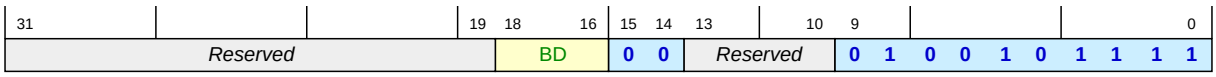


Table 3-111: Instruction NPU_SET_BLOCKDEP encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_blockdep.	npu_set_blockdep
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[18:16]	blockdep (BD)	This value is an unsigned integer.	-
[31:19]	Reserved	-	-

3.10.18 NPU_SET_DMA0_DST_REGION instruction

DMA0 destination region.

Set DMA0 destination region configuration.

The destination striding mode is set to the same as the source striding mode NPU_SET_DMA0_SRC_REGION.stride_mode.

Execution

```
set_register()
```

Encoding

Figure 3-84: NPU_SET_DMA0_DST_REGION bit assignments

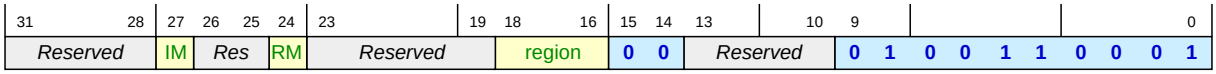


Table 3-112: Instruction NPU_SET_DMA0_DST_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_dma0_dst_region.	npu_set_dma0_dst_region
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[18:16]	region	If region_mode == external, this value gives the memory region number to use. See BASEP_ARRAY[] for the AXI base address of each memory region. See REGIONCFG for the AXI memory attributes of each memory region. If region_mode == internal, this value must be zero. This value is an unsigned integer.	-
[23:19]	Reserved	-	-
[24]	region_mode (RM)	This value is an enumeration of type dma_region_mode_t. This field can contain the following values: **0b0** external - DMA end point is in external memory **0b1** internal - DMA end point is in internal LUT RAM	-
[26:25]	Reserved	-	-

Bits	Name	Description	Reset
[27]	idx_mode (IM)	If index mode is set on a DMA source, this mode signals a gather operation with the index stream providing the gather locations. If index mode is set on a DMA destination, this mode signals a scatter operation with the index stream providing the gather locations. Behavior is UNPREDICTABLE if index mode is set for both source and destination. This value is an enumeration of type dma_idx_mode_t. This field can contain the following values: **0b0** disabled - DMA index mode is disabled **0b1** enabled - DMA index mode is enabled	-
[31:28]	Reserved	-	-

3.10.19 NPU_SET_DMA0_IDX_REGION instruction

DMA0 index region.

Set DMA0 index region configuration.

This register must be set if either SRC or DST uses indexing mode.

Execution

```
set_register()
```

Encoding

Figure 3-85: NPU_SET_DMA0_IDX_REGION bit assignments

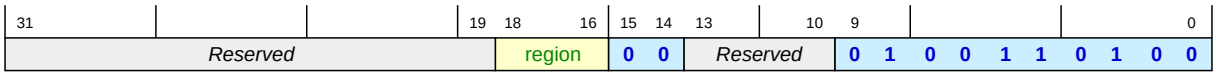


Table 3-113: Instruction NPU_SET_DMA0_IDX_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_dma0_idx_region.	npu_set_dma0_idx_region
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[18:16]	region	The memory region number to use for external accesses. See BASEP_ARRAY[] for the AXI base address of each memory region. See REGIONCFG for the AXI memory attributes of each memory region. This value is an unsigned integer.	-
[31:19]	Reserved	-	-

3.10.20 NPU_SET_DMA0_SIZE0 instruction

Size of second dimension for 2D/3D transfers.

Set size of second dimension for 2D/3D transfers.

This register must be set if either SRC or DST uses 2D/3D mode.

Execution

```
set_register()
```

Encoding

Figure 3-86: NPU_SET_DMA0_SIZE0 bit assignments

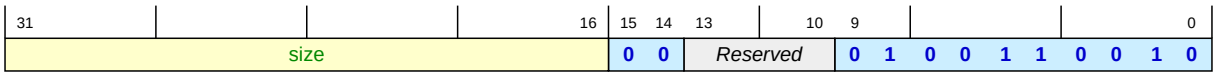


Table 3-114: Instruction NPU_SET_DMA0_SIZE0 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_dma0_size0.	npu_set_dma0_size0
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	size	This value is an unsigned integer.	-

3.10.21 NPU_SET_DMA0_SIZE1 instruction

Size of third dimension for 3D transfers.

Set size of third dimension for 3D transfers.

This register must be set if either SRC or DST uses 3D mode.

Execution

```
set_register()
```

Encoding

Figure 3-87: NPU_SET_DMA0_SIZE1 bit assignments

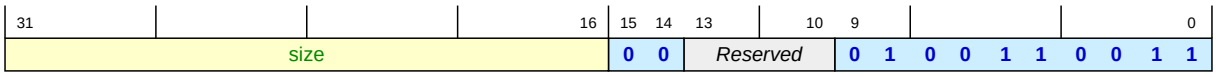


Table 3-115: Instruction NPU_SET_DMA0_SIZE1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_dma0_size1.	npu_set_dma0_size1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	size	This value is an unsigned integer.	-

3.10.22 NPU_SET_DMA0_SRC_REGION instruction

DMA0 source region.

Set DMA0 source region configuration.

Execution

```
set_register()
```

Encoding

Figure 3-88: NPU_SET_DMA0_SRC_REGION bit assignments

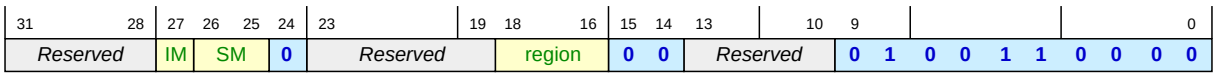


Table 3-116: Instruction NPU_SET_DMA0_SRC_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_dma0_src_region.	npu_set_dma0_src_region
[13:10]	Reserved	-	-

Bits	Name	Description	Reset
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[18:16]	region	The memory region number to use for external accesses. See BASEP_ARRAY[] for the AXI base address of each memory region. See REGIONCFG for the AXI memory attributes of each memory region. This value is an unsigned integer.	-
[23:19]	Reserved	-	-
[24]	region_mode (RM)	This value is an enumeration of type dma_region_mode_t. It must have the exact value external.	external
[26:25]	stride_mode (SM)	This value sets the stride mode for both source and destination DMA accesses. This value is an enumeration of type dma_stride_mode_t. This field can contain the following values: **0b00** d1 - 1D **0b01** d2 - 2D **0b10** d3 - 3D **0b11** Reserved	-
[27]	idx_mode (IM)	If index mode is set on a DMA source, this mode signals a gather operation with the index stream providing the gather locations. If index mode is set on a DMA destination, this mode signals a scatter operation with the index stream providing the gather locations. Behavior is UNPREDICTABLE if index mode is set for both source and destination. This value is an enumeration of type dma_idx_mode_t. This field can contain the following values: **0b0** disabled - DMA index mode is disabled **0b1** enabled - DMA index mode is enabled	-
[31:28]	Reserved	-	-

3.10.23 NPU_SET_IFM2_BROADCAST instruction

IFM2 broadcast configuration.

Set IFM2 broadcast configuration.

Execution

```
set_register()
```

Encoding

Figure 3-89: NPU_SET_IFM2_BROADCAST bit assignments

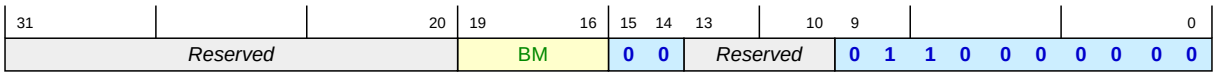


Table 3-117: Instruction NPU_SET_IFM2_BROADCAST encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm2_broadcast.	npu_set_ifm2_broadcast
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[19:16]	broadcast_mode (BM)	When not using broadcast_mode_scalar, accesses to IFM2 sets corresponding axes to 0 and corresponding IFM1 H/W/C to 1. If broadcast_mode_scalar is set for IFM2 then it must not be set for IFM1. This value is an enumeration of type broadcast_mode_t. This field can contain the following values: **0b0000** none - Broadcast disabled in C, W, and H **0b0001** h - Broadcast enabled on H **0b0010** w - Broadcast enabled on W **0b0011** hw - Broadcast enabled on H and W **0b0100** c - Broadcast enabled on C **0b0101** ch - Broadcast enabled on C and H **0b0110** cw - Broadcast enabled on C and W **0b0111** cwh - Broadcast enabled on C, W, and H **0b1000** scalar - Broadcast constant. Use constant given by NPU_SET_OP_SCALAR **0b1001..0b1111** Reserved	-
[31:20]	Reserved	-	-

3.10.24 NPU_SET_IFM2_HEIGHT0_M1 instruction

IFM2 Tile 0 height.

This value represents the height for IFM2 Tile 0.

The value stored is the height-1.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-90: NPU_SET_IFM2_HEIGHT0_M1 bit assignments

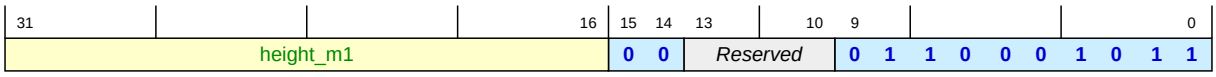


Table 3-118: Instruction NPU_SET_IFM2_HEIGHT0_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm2_height0_m1.	npu_set_ifm2_height0_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.25 NPU_SET_IFM2_HEIGHT1_M1 instruction

IFM2 Tile 1 height.

This value represents the height for IFM2 Tile 1.

The value stored is the height-1.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-91: NPU_SET_IFM2_HEIGHT1_M1 bit assignments

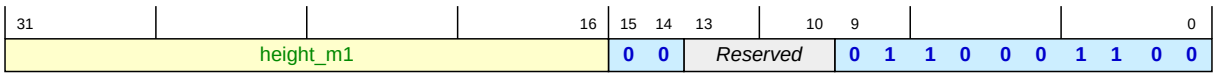


Table 3-119: Instruction NPU_SET_IFM2_HEIGHT1_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm2_height1_m1.	npu_set_ifm2_height1_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.26 NPU_SET_IFM2_PRECISION instruction

IFM2 Precision.

Set IFM2 precision information and storage format.

SET_IFM2_PRECISION must be set before any operation that uses IFM2 as defined in State Setting Commands.

IFM2 precision and type must match IFM precision and type for elementwise operations.

Execution

```
set_register()
```

Encoding

Figure 3-92: NPU_SET_IFM2_PRECISION bit assignments

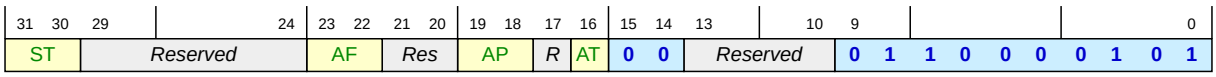


Table 3-120: Instruction NPU_SET_IFM2_PRECISION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm2_precision.	npu_set_ifm2_precision
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	activation_type (AT)	This value is an enumeration of type activation_type_t. This field can contain the following values: **0b0** unsigned - Unsigned **0b1** signed - Signed	-
[17]	Reserved	-	-
[19:18]	activation_precision (AP)	This value is an enumeration of type activation_precision_t. This field can contain the following values: **0b00** b8 - 8-bit **0b01** b16 - 16-bit **0b10** b32 - 32-bit **0b11** b64 - 64-bit	-
[21:20]	Reserved	-	-
[23:22]	activation_format (AF)	This value is an enumeration of type activation_format_t. This field can contain the following values: **0b00** nhwc - NHWC packed format **0b01** nhcwb16 - NHCWB16 brick format with 16 channels **0b10..0b11** Reserved	-
[29:24]	Reserved	-	-

Bits	Name	Description	Reset
[31:30]	activation_storage (ST)	If NPU_SET_IFM2_BROADCAST.broadcast_mode == scalar then this field must be set to none. This value is an enumeration of type activation_storage_t. This field can contain the following values: **0b00** tile2x2 - Stored in memory, 2 row x 2 col tiling **0b01** tile3x1 - Stored in memory, 3 row x 1 col tiling **0b10** chained - Stored in internal chaining buffer **0b11** none - Disable storage	-

3.10.27 NPU_SET_IFM2_REGION instruction

Index n for IFM2 access.

If the tensor is stored in memory, then n specifies the region number in the range 0 - 7.

If the tensor is stored in a chaining buffer, then n specified the chaining buffer number in the range 0 - 2.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-93: NPU_SET_IFM2_REGION bit assignments

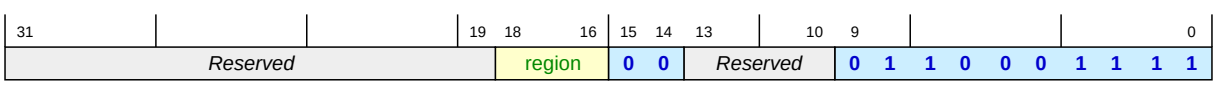


Table 3-121: Instruction NPU_SET_IFM2_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm2_region.	npu_set_ifm2_region
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[18:16]	region	The memory region number to use for external accesses. See BASEP_ARRAY[] for the AXI base address of each memory region. See REGIONCFG for the AXI memory attributes of each memory region. This value is an unsigned integer.	-
[31:19]	Reserved	-	-

3.10.28 NPU_SET_IFM2_WIDTH0_M1 instruction

IFM2 Tile 0 and Tile 2 width.

This value represents the width for IFM2 Tile 0 and Tile 2.

The value stored is the width-1.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-94: NPU_SET_IFM2_WIDTH0_M1 bit assignments

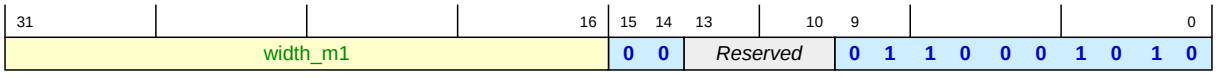


Table 3-122: Instruction NPU_SET_IFM2_WIDTH0_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm2_width0_m1.	npu_set_ifm2_width0_m1

Bits	Name	Description	Reset
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	width_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.29 NPU_SET_IFM2_ZERO_POINT instruction

IFM2 zero point.

Set IFM2 zero point.

For int8 IFM2, this is in the range -128 to +127 encoded in an int16 container.

For uint8 IFM2, this is in the range 0 to +255 encoded in an int16 container.

For uint16 IFM2, this is 0 or 32768 encoded in a uint16 container.

Otherwise, the zero point must be zero.

SET_IFM2_ZERO_POINT must be set before any operation that uses IFM2 as defined in State Setting Commands.

Execution

```
set_register()
```

Encoding

Figure 3-95: NPU_SET_IFM2_ZERO_POINT bit assignments

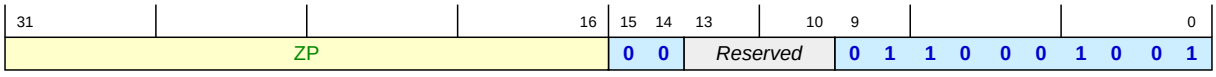


Table 3-123: Instruction NPU_SET_IFM2_ZERO_POINT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm2_zero_point.	npu_set_ifm2_zero_point
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	zero_point (ZP)	Zero point offset. This value is an untyped value.	-

3.10.30 NPU_SET_IFM_BROADCAST instruction

IFM broadcast configuration.

Set IFM broadcast configuration.

Execution

```
set_register()
```

Encoding

Figure 3-96: NPU_SET_IFM_BROADCAST bit assignments

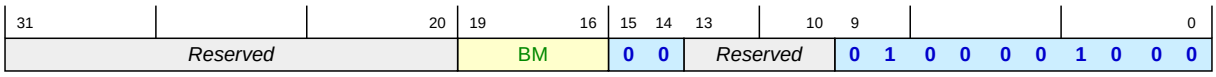


Table 3-124: Instruction NPU_SET_IFM_BROADCAST encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_broadcast.	npu_set_ifm_broadcast
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[19:16]	broadcast_mode (BM)	When not using broadcast_mode_scalar, accesses to IFM1 sets corresponding axes to 0 and corresponding IFM1 H/W/C to 1. This value is an enumeration of type broadcast_mode_t. This field can contain the following values: **0b0000** none - Broadcast disabled in C, W, and H **0b0001** h - Broadcast enabled on H **0b0010** w - Broadcast enabled on W **0b0011** hw - Broadcast enabled on H and W **0b0100** c - Broadcast enabled on C **0b0101** ch - Broadcast enabled on C and H **0b0110** cw - Broadcast enabled on C and W **0b0111** cwh - Broadcast enabled on C, W, and H **0b1000** scalar - Broadcast constant. Use constant given by NPU_SET_OP_SCALAR **0b1001..0b1111** Reserved	-
[31:20]	Reserved	-	-

3.10.31 NPU_SET_IFM_DEPTH_M1 instruction

Number of input channels for convolution.

This value represents the number of input channels for a convolution.

The value stored is the number of input channels-1.

Execution

```
set_register()
```

Encoding

Figure 3-97: NPU_SET_IFM_DEPTH_M1 bit assignments

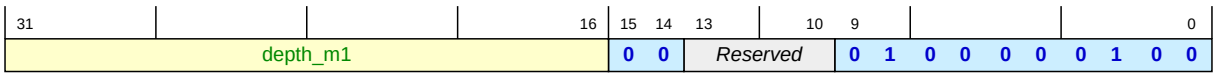


Table 3-125: Instruction NPU_SET_IFM_DEPTH_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_depth_m1.	npu_set_ifm_depth_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	depth_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.32 NPU_SET_IFM_HEIGHT0_M1 instruction

IFM Tile 0 height.

This value represents the height for IFM Tile 0.

The value stored is the height-1.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-98: NPU_SET_IFM_HEIGHT0_M1 bit assignments

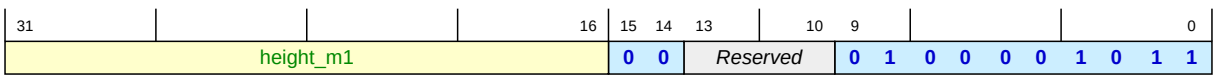


Table 3-126: Instruction NPU_SET_IFM_HEIGHT0_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_height0_m1.	npu_set_ifm_height0_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.33 NPU_SET_IFM_HEIGHT1_M1 instruction

IFM Tile 1 height.

This value represents the height for IFM Tile 1.

The value stored is the height-1.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-99: NPU_SET_IFM_HEIGHT1_M1 bit assignments

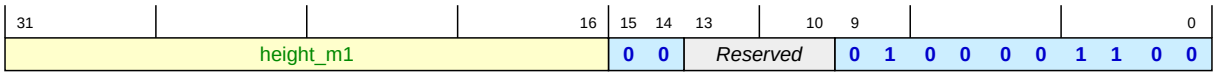


Table 3-127: Instruction NPU_SET_IFM_HEIGHT1_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_height1_m1.	npu_set_ifm_height1_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.34 NPU_SET_IFM_PAD_BOTTOM instruction

IFM bottom pad.

Set IFM bottom pad. Pad is applied after upscale if ifm_upscale_mode!=none.

Execution

```
set_register()
```

Encoding

Figure 3-100: NPU_SET_IFM_PAD_BOTTOM bit assignments

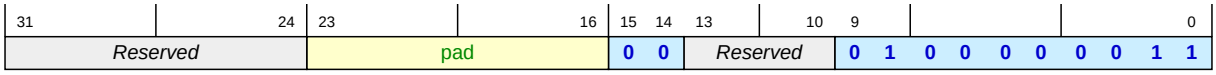


Table 3-128: Instruction NPU_SET_IFM_PAD_BOTTOM encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_pad_bottom.	npu_set_ifm_pad_bottom
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[23:16]	pad	This value is an unsigned integer.	-
[31:24]	Reserved	-	-

3.10.35 NPU_SET_IFM_PAD_LEFT instruction

IFM left pad.

Set IFM left pad. Pad is applied after upscale if ifm_upscale_mode!=none.

Execution

```
set_register()
```

Encoding

Figure 3-101: NPU_SET_IFM_PAD_LEFT bit assignments

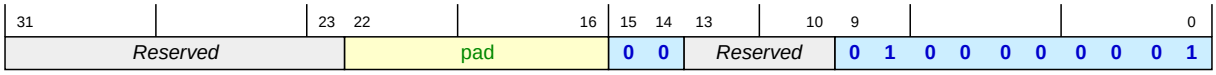


Table 3-129: Instruction NPU_SET_IFM_PAD_LEFT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_pad_left.	npu_set_ifm_pad_left
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[22:16]	pad	This value is an unsigned integer.	-
[31:23]	Reserved	-	-

3.10.36 NPU_SET_IFM_PAD_RIGHT instruction

IFM right pad.

Set IFM right pad. Pad is applied after upscale if ifm_upscale_mode!=none.

Execution

```
set_register()
```

Encoding

Figure 3-102: NPU_SET_IFM_PAD_RIGHT bit assignments

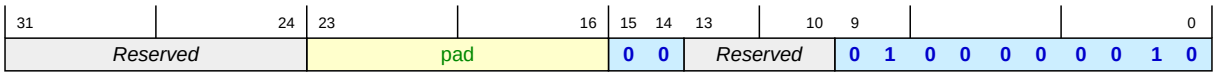


Table 3-130: Instruction NPU_SET_IFM_PAD_RIGHT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_pad_right.	npu_set_ifm_pad_right
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[23:16]	pad	This value is an unsigned integer.	-
[31:24]	Reserved	-	-

3.10.37 NPU_SET_IFM_PAD_TOP instruction

IFM top pad.

Set IFM top pad. Pad is applied after upscale if ifm_upscale_mode!=none.

Execution

```
set_register()
```

Encoding

Figure 3-103: NPU_SET_IFM_PAD_TOP bit assignments

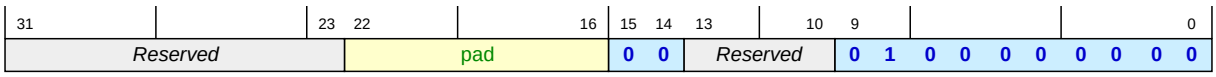


Table 3-131: Instruction NPU_SET_IFM_PAD_TOP encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_pad_top.	npu_set_ifm_pad_top
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[22:16]	pad	This value is an unsigned integer.	-
[31:23]	Reserved	-	-

3.10.38 NPU_SET_IFM_PRECISION instruction

IFM Precision.

Set IFM precision information and storage format.

SET_IFM_PRECISION must be set before any operation that uses IFM as defined in State Setting Commands.

Execution

```
set_register()
```

Encoding

Figure 3-104: NPU_SET_IFM_PRECISION bit assignments

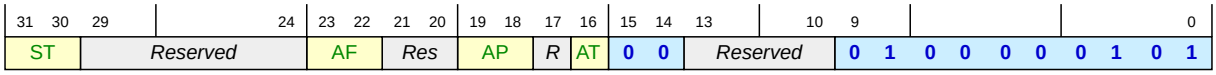


Table 3-132: Instruction NPU_SET_IFM_PRECISION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_precision.	npu_set_ifm_precision
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	activation_type (AT)	This value is an enumeration of type activation_type_t. This field can contain the following values: **0b0** unsigned - Unsigned **0b1** signed - Signed	-
[17]	Reserved	-	-
[19:18]	activation_precision (AP)	This value is an enumeration of type activation_precision_t. This field can contain the following values: **0b00** b8 - 8-bit **0b01** b16 - 16-bit **0b10** b32 - 32-bit **0b11** b64 - 64-bit	-
[21:20]	Reserved	-	-
[23:22]	activation_format (AF)	This value is an enumeration of type activation_format_t. This field can contain the following values: **0b00** nhwc - NHWC packed format **0b01** nhcwb16 - NHCWB16 brick format with 16 channels **0b10..0b11** Reserved	-

Bits	Name	Description	Reset
[29:24]	Reserved	-	-
[31:30]	activation_storage (ST)	If NPU_SET_IFM_BROADCAST.broadcast_mode == scalar then this field must be set to none. This value is an enumeration of type activation_storage_t. This field can contain the following values: **0b00** tile2x2 - Stored in memory, 2 row x 2 col tiling **0b01** tile3x1 - Stored in memory, 3 row x 1 col tiling **0b10** chained - Stored in internal chaining buffer **0b11** none - Disable storage	-

3.10.39 NPU_SET_IFM_REGION instruction

Index n for IFM access.

If the tensor is stored in memory, then n specifies the region number in the range 0 - 7.

If the tensor is stored in a chaining buffer, then n specified the chaining buffer number in the range 0 - 2.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-105: NPU_SET_IFM_REGION bit assignments

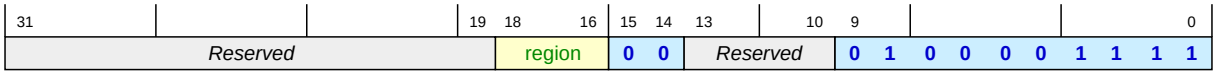


Table 3-133: Instruction NPU_SET_IFM_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_region.	npu_set_ifm_region
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[18:16]	region	The memory region number to use for external accesses. See BASEP_ARRAY[] for the AXI base address of each memory region. See REGIONCFG for the AXI memory attributes of each memory region. This value is an unsigned integer.	-
[31:19]	Reserved	-	-

3.10.40 NPU_SET_IFM_UPSCALE instruction

IFM upscale mode.

Set IFM upscale mode.

Execution

```
set_register()
```

Encoding

Figure 3-106: NPU_SET_IFM_UPSCALE bit assignments

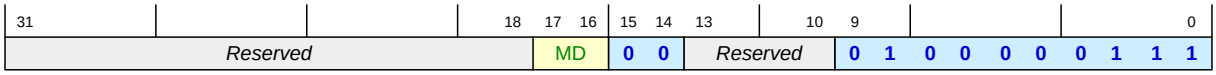


Table 3-134: Instruction NPU_SET_IFM_UPSCALE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_upscale.	npu_set_ifm_upscale
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[17:16]	mode (MD)	This value is an enumeration of type ifm_upscale_mode_t. This field can contain the following values: **0b00** none - None **0b01** nearest - 2x2 insert nearest **0b10** zeros - 2x2 insert zeros **0b11** Reserved	-
[31:18]	Reserved	-	-

3.10.41 NPU_SET_IFM_WIDTH0_M1 instruction

IFM Tile 0 and Tile 2 width.

This value represents the width for IFM Tile 0 and Tile 2.

The value stored is the width-1.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-107: NPU_SET_IFM_WIDTH0_M1 bit assignments

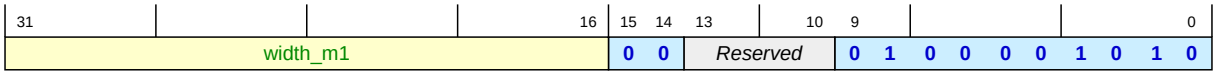


Table 3-135: Instruction NPU_SET_IFM_WIDTH0_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_width0_m1.	npu_set_ifm_width0_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	width_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.42 NPU_SET_IFM_ZERO_POINT instruction

IFM zero point.

Set IFM zero point.

For int8 IFM, this is in the range -128 to +127 encoded in an int16 container.

For uint8 IFM, this is in the range 0 to +255 encoded in an int16 container.

For uint16 IFM, this is 0 or 32768 encoded in a uint16 container.

Otherwise, the zero point must be zero.

SET_IFM_ZERO_POINT must be set before any operation that uses IFM as defined in State Setting Commands.

For reduce_sum operations, the zero-point must be zero.

Execution

```
set_register()
```

Encoding

Figure 3-108: NPU_SET_IFM_ZERO_POINT bit assignments

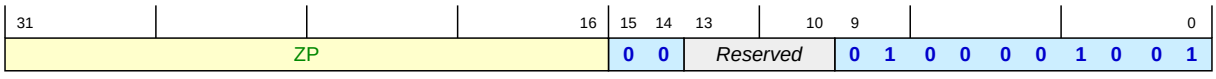


Table 3-136: Instruction NPU_SET_IFM_ZERO_POINT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ifm_zero_point.	npu_set_ifm_zero_point
[13:10]	Reserved	-	-

Bits	Name	Description	Reset
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	zero_point (ZP)	Zero point offset. This value is an untyped value.	-

3.10.43 NPU_SET_KERNEL_HEIGHT_M1 instruction

Kernel height.

For convolution or pooling operations this value stores (dilated_height-1) = (kernel_height-1)*kernel_y_dilation.

For resize operations this value stores (ifm_height-1).

Execution

```
set_register()
```

Encoding

Figure 3-109: NPU_SET_KERNEL_HEIGHT_M1 bit assignments

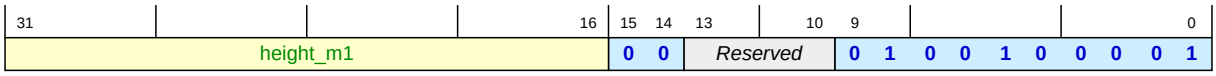


Table 3-137: Instruction NPU_SET_KERNEL_HEIGHT_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_kernel_height_m1.	npu_set_kernel_height_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.44 NPU_SET_KERNEL_STRIDE instruction

Kernel stride.

Set kernel stride and dilation

Execution

```
set_register()
```

Encoding

Figure 3-110: NPU_SET_KERNEL_STRIDE bit assignments

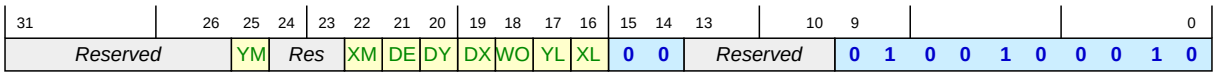


Table 3-138: Instruction NPU_SET_KERNEL_STRIDE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_kernel_stride.	npu_set_kernel_stride
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	stride_x_lsb (XL)	This value is an unsigned integer.	-
[17]	stride_y_lsb (YL)	This value is an unsigned integer.	-
[18]	weight_order (WO)	This value is an enumeration of type weight_order_t. This field can contain the following values: **0b0** depth_first - Depth first weight order **0b1** part_kernel_first - Part kernel first weight order	-
[19]	dilation_x (DX)	This value is an enumeration of type kernel_dilation_t. This field can contain the following values: **0b0** none - No dilation **0b1** x2 - Dilation of x2	-
[20]	dilation_y (DY)	This value is an enumeration of type kernel_dilation_t. This field can contain the following values: **0b0** none - No dilation **0b1** x2 - Dilation of x2	-

Bits	Name	Description	Reset
[21]	decomposition (DE)	This value is an enumeration of type kernel_decomposition_t. This field can contain the following values: **0b0** d8x8 - 8x8 decomposition mode **0b1** d4x4 - 4x4 decomposition mode	-
[22]	stride_x_msb (XM)	This value is an unsigned integer.	-
[24:23]	Reserved	-	-
[25]	stride_y_msb (YM)	This value is an unsigned integer.	-
[31:26]	Reserved	-	-

3.10.45 NPU_SET_KERNEL_WIDTH_M1 instruction

Kernel width.

For convolution or pooling operations this value stores (dilated_width-1) = (kernel_width-1)*kernel_x_dilation.

For resize operations this value stores (ifm_width-1). This register must be clipped to the IFM width read for the resize. If,

- The horizontal scaling ratio is scale_x_n/scale_x_d
- The horizontal scaling offset is offset_x
- The OFM width is ofm_width

Then the IFM width read is calculated as `ifm_width_read = ((ofm_width-1)*scale_x_d + offset_x)/scale_x_n + 2` .

For resize this register must be set to `min(ifm_width - 1, ifm_width_read - 1)` .

Execution

```
set_register()
```

Encoding

Figure 3-111: NPU_SET_KERNEL_WIDTH_M1 bit assignments

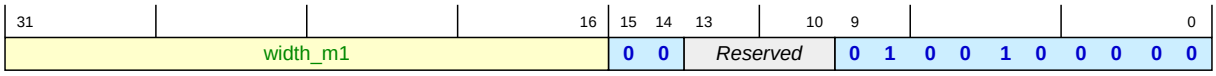


Table 3-139: Instruction NPU_SET_KERNEL_WIDTH_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_kernel_width_m1.	npu_set_kernel_width_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	width_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.46 NPU_SET_OFM_BLK_DEPTH_M1 instruction

OFM block depth.

This value represents the OFM block depth.

The value stored is the depth-1.

If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the OFM is transposed on write. The value encoded here is the shape before transposition.

If NPU_SET_OFM_PRECISION.activation_transpose transposes the channel dimension to width then depth <= 128. If NPU_SET_OFM_PRECISION.activation_transpose transposes the channel dimension to height then depth <= 128.

Execution

```
set_register()
```

Encoding

Figure 3-112: NPU_SET_OFM_BLK_DEPTH_M1 bit assignments

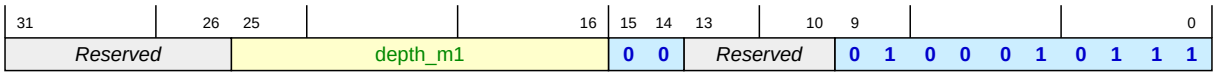


Table 3-140: Instruction NPU_SET_OFM_BLK_DEPTH_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_blk_depth_m1.	npu_set_ofm_blk_depth_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[25:16]	depth_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-
[31:26]	Reserved	-	-

3.10.47 NPU_SET_OFM_BLK_HEIGHT_M1 instruction

OFM block height.

This value represents the OFM block height.

The value stored is the height-1.

If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the OFM is transposed on write. The value encoded here is the shape before transposition.

Execution

```
set_register()
```

Encoding

Figure 3-113: NPU_SET_OFM_BLK_HEIGHT_M1 bit assignments

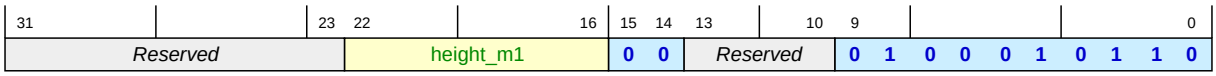


Table 3-141: Instruction NPU_SET_OFM_BLK_HEIGHT_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_blk_height_m1.	npu_set_ofm_blk_height_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[22:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-
[31:23]	Reserved	-	-

3.10.48 NPU_SET_OFM_BLK_WIDTH_M1 instruction

OFM block width.

This value represents the OFM block width.

The value stored is the width-1.

If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the OFM is transposed on write. The value encoded here is the shape before transposition.

Execution

```
set_register()
```

Encoding

Figure 3-114: NPU_SET_OFM_BLK_WIDTH_M1 bit assignments

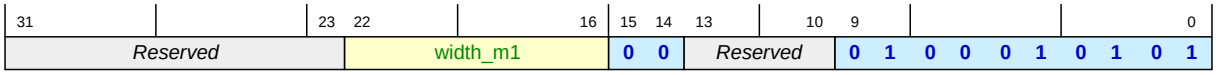


Table 3-142: Instruction NPU_SET_OFM_BLK_WIDTH_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_blk_width_m1.	npu_set_ofm_blk_width_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[22:16]	width_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-
[31:23]	Reserved	-	-

3.10.49 NPU_SET_OFM_DEPTH_M1 instruction

Output feature map depth.

This value represents the output feature map depth.

The value stored is the depth-1.

If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the OFM is transposed on write. The value encoded here is the shape before transposition.

Execution

```
set_register()
```

Encoding

Figure 3-115: NPU_SET_OFM_DEPTH_M1 bit assignments

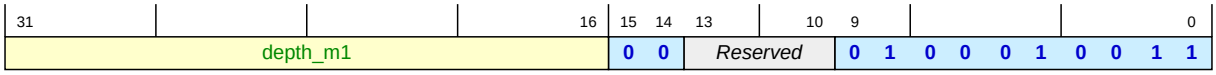


Table 3-143: Instruction NPU_SET_OFM_DEPTH_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_depth_m1.	npu_set_ofm_depth_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	depth_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.50 NPU_SET_OFM_HEIGHT0_M1 instruction

OFM Tile 0 height.

This value represents the OFM Tile 0 height.

The value stored is the height-1.

This shape refers to the tensor stored in memory. If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the axes referred to are after transposition.

Execution

```
set_register()
```

Encoding

Figure 3-116: NPU_SET_OFM_HEIGHT0_M1 bit assignments

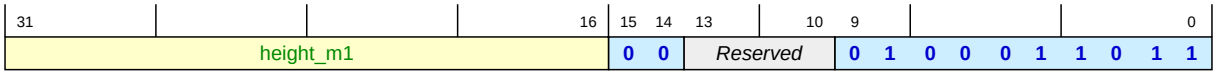


Table 3-144: Instruction NPU_SET_OFM_HEIGHT0_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_height0_m1.	npu_set_ofm_height0_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.51 NPU_SET_OFM_HEIGHT1_M1 instruction

OFM Tile 1 height.

This value represents the OFM Tile 1 height.

The value stored is the height-1.

This shape refers to the tensor stored in memory. If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the axes referred to are after transposition.

Execution

```
set_register()
```

Encoding

Figure 3-117: NPU_SET_OFM_HEIGHT1_M1 bit assignments

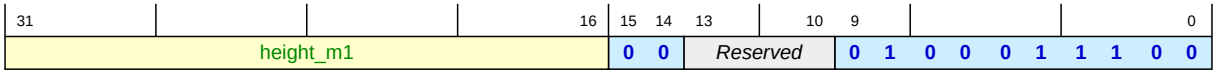


Table 3-145: Instruction NPU_SET_OFM_HEIGHT1_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_height1_m1.	npu_set_ofm_height1_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.52 NPU_SET_OFM_HEIGHT_M1 instruction

Output feature map height.

This value represents the output feature map height.

The value stored is the height-1.

If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the OFM is transposed on write. The value encoded here is the shape before transposition.

Execution

```
set_register()
```

Encoding

Figure 3-118: NPU_SET_OFM_HEIGHT_M1 bit assignments

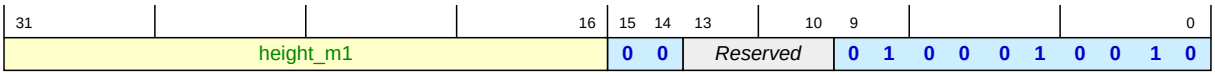


Table 3-146: Instruction NPU_SET_OFM_HEIGHT_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_height_m1.	npu_set_ofm_height_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	height_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.53 NPU_SET_OFM_PRECISION instruction

OFM Precision.

Set OFM precision information and storage format.

Execution

```
set_register()
```

Encoding

Figure 3-119: NPU_SET_OFM_PRECISION bit assignments

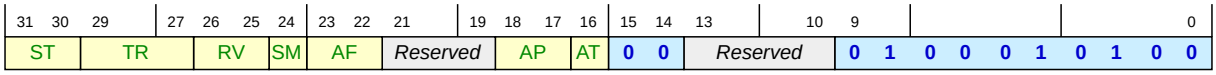


Table 3-147: Instruction NPU_SET_OFM_PRECISION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_precision.	npu_set_ofm_precision
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	activation_type (AT)	This value is an enumeration of type activation_type_t. This field can contain the following values: **0b0** unsigned - Unsigned **0b1** signed - Signed	-
[18:17]	activation_precision (AP)	This value is an enumeration of type activation_precision_t. This field can contain the following values: **0b00** b8 - 8-bit **0b01** b16 - 16-bit **0b10** b32 - 32-bit **0b11** b64 - 64-bit	-
[21:19]	Reserved	-	-
[23:22]	activation_format (AF)	This value is an enumeration of type activation_format_t. This field can contain the following values: **0b00** nhwc - NHWC packed format **0b01** nhcwb16 - NHCWB16 brick format with 16 channels **0b10..0b11** Reserved	-

Bits	Name	Description	Reset
[24]	scale_mode (SM)	This value is an enumeration of type ofm_scale_mode_t. This field can contain the following values: **0b0** per_channel - Per channel scale/bias **0b1** global - Global scale (SET_OFM_SCALE), no bias	-
[26:25]	activation_reverse (RV)	This value is an enumeration of type activation_reverse_t. This field can contain the following values: **0b00** none - No reverse **0b01** H - Reverse Y axis **0b10** W - Reverse X axis **0b11** C - Reverse C axis	-
[29:27]	activation_transpose (TR)	This value is an enumeration of type activation_transpose_t. This field can contain the following values: **0b000** HWC - No transpose **0b001** WHC - Swap Y and X dimensions **0b010** HCW - Swap X and C dimensions **0b011** WCH - Swap X,C then Y,C **0b100..0b101** Reserved **0b110** CHW - Swap Y,C then X,C **0b111** CWH - Swap Y and C dimensions	-

Bits	Name	Description	Reset
[31:30]	activation_storage (ST)	This value is an enumeration of type activation_storage_t. This field can contain the following values: **0b00** tile2x2 - Stored in memory, 2 row x 2 col tiling **0b01** tile3x1 - Stored in memory, 3 row x 1 col tiling **0b10** chained - Stored in internal chaining buffer **0b11** none - Disable storage	-

3.10.54 NPU_SET_OFM_REGION instruction

Index n for OFM access.

If the tensor is stored in memory then n specifies the region number in the range 0 to 7.

If the tensor is stored in a chaining buffer then n specified the chaining buffer number in the range 0 to 2.

Execution

```
set_register()
```

Encoding

Figure 3-120: NPU_SET_OFM_REGION bit assignments

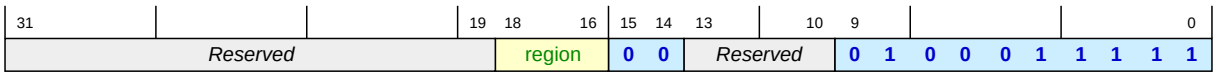


Table 3-148: Instruction NPU_SET_OFM_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_region.	npu_set_ofm_region
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl

Bits	Name	Description	Reset
[18:16]	region	The memory region number to use for external accesses. See BASEP_ARRAY[] for the AXI base address of each memory region. See REGIONCFG for the AXI memory attributes of each memory region. This value is an unsigned integer.	-
[31:19]	Reserved	-	-

3.10.55 NPU_SET_OFM_WIDTH0_M1 instruction

OFM Tile 0 and tile 2 width.

This value represents the OFM Tile 0 and Tile 2 width.

The value stored is the width-1.

This shape refers to the tensor stored in memory. If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the axes referred to are after transposition.

Execution

```
set_register()
```

Encoding

Figure 3-121: NPU_SET_OFM_WIDTH0_M1 bit assignments

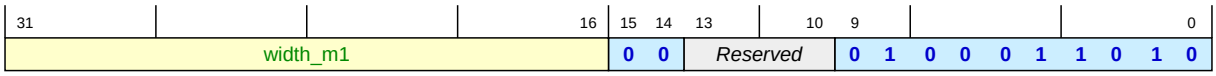


Table 3-149: Instruction NPU_SET_OFM_WIDTH0_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_width0_m1.	npu_set_ofm_width0_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	width_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.56 NPU_SET_OFM_WIDTH_M1 instruction

Output feature map width.

This value represents the output feature map width.

The value stored is the width-1.

If NPU_SET_OFM_PRECISION.activation_transpose != HWC then the OFM is transposed on write. The value encoded here is the shape before transposition.

Execution

```
set_register()
```

Encoding

Figure 3-122: NPU_SET_OFM_WIDTH_M1 bit assignments

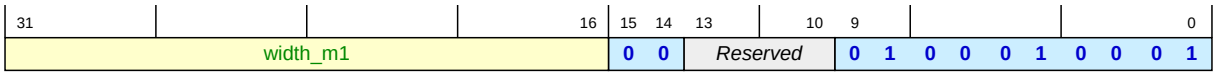


Table 3-150: Instruction NPU_SET_OFM_WIDTH_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_width_m1.	npu_set_ofm_width_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	width_m1	This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-

3.10.57 NPU_SET_OFM_ZERO_POINT instruction

OFM zero point.

Set OFM zero point.

For NPU_SET_OFM_PRECISION.activation_type == unsigned the zero point is encoded as uint16 and must be 0-255 or 32768.

For NPU_SET_OFM_PRECISION.activation_type == signed the zero point is encoded as int16 and must be -128 to +127.

Execution

```
set_register()
```

Encoding

Figure 3-123: NPU_SET_OFM_ZERO_POINT bit assignments

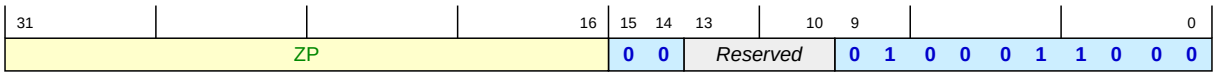


Table 3-151: Instruction NPU_SET_OFM_ZERO_POINT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_ofm_zero_point.	npu_set_ofm_zero_point
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[31:16]	zero_point (ZP)	Zero point offset. This value is an untyped value.	-

3.10.58 NPU_SET_RESIZE_X_OFFSET instruction

Set resize offset X.

The scale applied to this axis is defined as *scale_n*/*scale_d*. For a scale up this value is greater than one and for a scale down the value is less than one. The step between each sampling position in the IFM is *scale_d*/*scale_n*.

Output coordinate *v* maps to input sampling coordinate *u* = (*v* * *scale_d* + *offset*)/*scale_n*.

This instruction sets the *offset* value for X scaling.

Execution

```
set_register()
```

Encoding

Figure 3-124: NPU_SET_RESIZE_X_OFFSET bit assignments

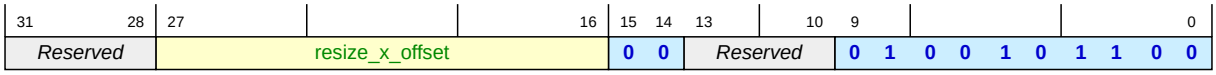


Table 3-152: Instruction NPU_SET_RESIZE_X_OFFSET encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_resize_x_offset.	npu_set_resize_x_offset
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[27:16]	resize_x_offset	This value is a signed integer.	-
[31:28]	Reserved	-	-

3.10.59 NPU_SET_RESIZE_X_SCALE_N_M1 instruction

Set resize scale X numerator.

The scale applied to this axis is defined as *scale_n*/*scale_d*. For a scale up this value is greater than one and for a scale down the value is less than one. The step between each sampling position in the IFM is *scale_d*/*scale_n*.

Output coordinate *v* maps to input sampling coordinate *u* = (*v* * *scale_d* + *offset*)/*scale_n*.

This instruction sets the *scale_n* value for X scaling. The input step rate scale_d/scale_n is encoded in NPU_SET_RESIZE_X_STEP.

Execution

```
set_register()
```

Encoding

Figure 3-125: NPU_SET_RESIZE_X_SCALE_N_M1 bit assignments

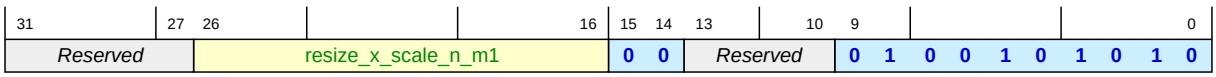


Table 3-153: Instruction NPU_SET_RESIZE_X_SCALE_N_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_resize_x_scale_n_m1.	npu_set_resize_x_scale_n_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[26:16]	resize_x_scale_n_m1	A numerator value in range 1 to 2048 is stored with 1 subtracted, as 0 to 2047. This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-
[31:27]	Reserved	-	-

3.10.60 NPU_SET_RESIZE_Y_OFFSET instruction

Set resize offset Y.

The scale applied to this axis is defined as *scale_n*/*scale_d*. For a scale up this value is greater than one and for a scale down the value is less than one. The step between each sampling position in the IFM is *scale_d*/*scale_n*.

Output coordinate *v* maps to input sampling coordinate *u* = (*v* * *scale_d* + *offset*)/*scale_n*.

This instruction sets the *offset* value for Y scaling.

Execution

```
set_register()
```

Encoding

Figure 3-126: NPU_SET_RESIZE_Y_OFFSET bit assignments

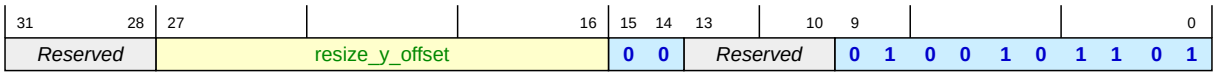


Table 3-154: Instruction NPU_SET_RESIZE_Y_OFFSET encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_resize_y_offset.	npu_set_resize_y_offset
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[27:16]	resize_y_offset	This value is a signed integer.	-

Bits	Name	Description	Reset
[31:28]	Reserved	-	-

3.10.61 NPU_SET_RESIZE_Y_SCALE_N_M1 instruction

Set resize scale Y numerator.

The scale applied to this axis is defined as *scale_n*/*scale_d*. For a scale up this value is greater than one and for a scale down the value is less than one. The step between each sampling position in the IFM is *scale_d*/*scale_n*.

Output coordinate *v* maps to input sampling coordinate *u* = (*v* * *scale_d* + *offset*)/*scale_n*.

This instruction sets the *scale_n* value for Y scaling. The input step rate scale_d/scale_n is encoded in NPU_SET_RESIZE_Y_STEP.

Execution

```
set_register()
```

Encoding

Figure 3-127: NPU_SET_RESIZE_Y_SCALE_N_M1 bit assignments

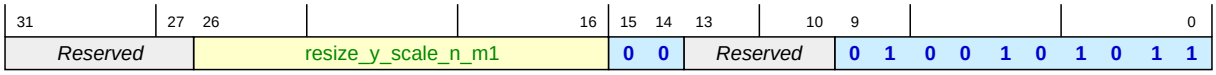


Table 3-155: Instruction NPU_SET_RESIZE_Y_SCALE_N_M1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_resize_y_scale_n_m1.	npu_set_resize_y_scale_n_m1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[26:16]	resize_y_scale_n_m1	A numerator value in range 1 to 2048 is stored with 1 subtracted, as 0 to 2047. This value is an unsigned integer. The stored value is a modified version of the represented value, where the modifier applied is minus(1).	-
[31:27]	Reserved	-	-

3.10.62 NPU_SET_SCALE_REGION instruction

Index n for scale stream access.

Set Index n for scale stream access: Region[n] is added to all scale stream addresses.

Execution

```
set_register()
```

Encoding

Figure 3-128: NPU_SET_SCALE_REGION bit assignments

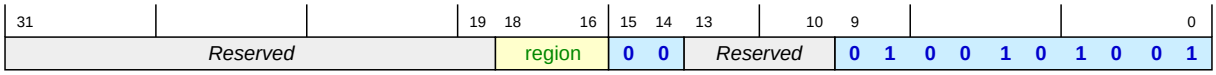


Table 3-156: Instruction NPU_SET_SCALE_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_scale_region.	npu_set_scale_region
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[18:16]	region	This value is an unsigned integer.	-
[31:19]	Reserved	-	-

3.10.63 NPU_SET_WEIGHT_FORMAT instruction

Set weight stream format.

Set weight stream format for convolution operations.

Execution

```
set_register()
```

Encoding

Figure 3-129: NPU_SET_WEIGHT_FORMAT bit assignments

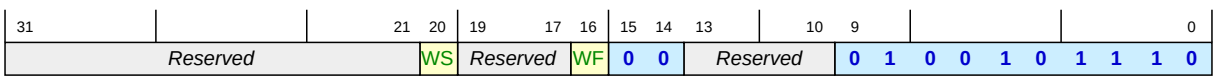


Table 3-157: Instruction NPU_SET_WEIGHT_FORMAT encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_weight_format.	npu_set_weight_format
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[16]	weight_format (WF)	This value is an enumeration of type weight_format_t. This field can contain the following values: **0b0** swd - Standard weight decoder stream(s) Standard weight decoder of num_wd stream(s)" **0b1** fwd - Fast weight decoder, single stream	-
[19:17]	Reserved	-	-
[20]	weight_sparsity (WS)	This value is an enumeration of type weight_sparsity_t. This field can contain the following values: **0b0** none - No sparsity assumed The weight stream values must be in the range -255 to +255. **0b1** sparse_2_4 - 2 weights zero out of each 4 in input channel direction The weight stream values must be in the range -127 to +127. In the input channel dimension, 2 out of each group of 4 weights must be zero.	-
[31:21]	Reserved	-	-

3.10.64 NPU_SET_WEIGHT_REGION instruction

Index n for weight stream access.

Set Index n for weight stream access: Region[n] is added to all weight stream addresses.

Execution

```
set_register()
```

Encoding

Figure 3-130: NPU_SET_WEIGHT_REGION bit assignments

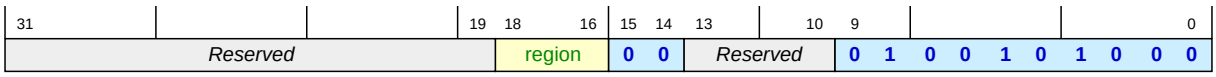


Table 3-158: Instruction NPU_SET_WEIGHT_REGION encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd0_opcode_t. It must have the exact value npu_set_weight_region.	npu_set_weight_region
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd0_ctrl.	cmd0_ctrl
[18:16]	region	This value is an unsigned integer.	-
[31:19]	Reserved	-	-

3.11 Instruction index for cmd1 stream

This section describes 64-bit NPU commands.

Commands in this section form part of the NPU command stream and execute on the NPU. Each command in this section consists of two 32-bit words forming a 64-bit command. The first 32-bit word (bits [31:0] of the command) encodes parameter information payload_0 in bits [31:16]. The second 32-bit word (bits [63:32] of the command) encodes a 32-bit parameter payload_1. Commands are 32-bit aligned in memory.

Table 3-159: List of instructions in the cmd1 pipe

Name	Description
NPU_OP_BRANCH	Branch to new location
NPU_SET_DMA0_DST	DMA user channel 0 destination byte offset from DMA0_DST_REGION
NPU_SET_DMA0_DST_STRIDE0	Destination byte stride after each 1D transfer
NPU_SET_DMA0_DST_STRIDE1	Destination byte stride after 2D transfer
NPU_SET_DMA0_IDX	DMA channel 0 index array address
NPU_SET_DMA0_IDX_MAX	DMA channel 0 index maximum value
NPU_SET_DMA0_IDX_SKIP1	Index byte distance to skip in index after each 2D transfer
NPU_SET_DMA0_LEN	DMA user channel 0 transfer length in bytes for each 1D transfer
NPU_SET_DMA0_SRC	DMA user channel 0 source byte offset from DMA0_SRC_REGION
NPU_SET_DMA0_SRC_STRIDE0	Source byte stride after each 1D transfer
NPU_SET_DMA0_SRC_STRIDE1	Source byte stride after each 2D transfer
NPU_SET_IFM2_BASE0	IFM2 Tile 0 address
NPU_SET_IFM2_BASE1	IFM2 Tile 1 address
NPU_SET_IFM2_BASE2	IFM2 Tile 2 address
NPU_SET_IFM2_BASE3	IFM2 Tile 3 address
NPU_SET_IFM2_SCALE	IFM2 input scale
NPU_SET_IFM2_STRIDE_C	IFM2 byte stride between channel blocks
NPU_SET_IFM2_STRIDE_X	IFM2 byte stride between horizontal values
NPU_SET_IFM2_STRIDE_Y	IFM2 byte stride between vertical values
NPU_SET_IFM_BASE0	IFM Tile 0 address

Name	Description
NPU_SET_IFM_BASE1	IFM Tile 1 address
NPU_SET_IFM_BASE2	IFM Tile 2 address
NPU_SET_IFM_BASE3	IFM Tile 3 address
NPU_SET_IFM_SCALE	IFM input scale
NPU_SET_IFM_STRIDE_C	IFM byte stride between channel blocks
NPU_SET_IFM_STRIDE_X	IFM byte stride between horizontal values
NPU_SET_IFM_STRIDE_Y	IFM byte stride between vertical values
NPU_SET_OFM_BASE0	OFM Tile 0 address
NPU_SET_OFM_BASE1	OFM Tile 1 address
NPU_SET_OFM_BASE2	OFM Tile 2 address
NPU_SET_OFM_BASE3	OFM Tile 3 address
NPU_SET_OFM_SCALE	OFM scale
NPU_SET_OFM_STRIDE_C	OFM byte stride between channel blocks
NPU_SET_OFM_STRIDE_X	OFM byte stride between horizontal values
NPU_SET_OFM_STRIDE_Y	OFM byte stride between vertical values
NPU_SET_OP_SCALAR	Operation scalar value
NPU_SET_RESIZE_X_STEP	Resize X axis step parameters
NPU_SET_RESIZE_Y_STEP	Resize Y axis step parameters
NPU_SET_SCALE_BASE	Scale and bias stream input byte offset from SCALE_REGION
NPU_SET_SCALE_LENGTH	Scale and bias stream input byte length
NPU_SET_WEIGHT1_BASE	Weight stream byte offset in WEIGHT_REGION for weight decoder 1
NPU_SET_WEIGHT1_LENGTH	Weight stream byte length for weight decoder 1
NPU_SET_WEIGHT2_BASE	Weight stream byte offset in WEIGHT_REGION for weight decoder 2
NPU_SET_WEIGHT2_LENGTH	Weight stream byte length for weight decoder 2
NPU_SET_WEIGHT3_BASE	Weight stream byte offset in WEIGHT_REGION for weight decoder 3
NPU_SET_WEIGHT3_LENGTH	Weight stream byte length for weight decoder 3
NPU_SET_WEIGHT_BASE	Weight stream byte offset in WEIGHT_REGION
NPU_SET_WEIGHT_LENGTH	Weight stream byte length

3.11.1 Instruction encoding index for cmd1 stream

This section describes 64-bit NPU commands.

Commands in this section form part of the NPU command stream and execute on the NPU. Each command in this section consists of two 32-bit words forming a 64-bit command. The first 32-bit word (bits [31:0] of the command) encodes parameter information payload_0 in bits [31:16]. The second 32-bit word (bits [63:32] of the command) encodes a 32-bit parameter payload_1. Commands are 32-bit aligned in memory.

Figure 3-131: Instruction encoding index for cmd1 stream

Instructions	63 62 ... 59 58 ... 52	51 ... 48	47 ... 43	42 ... 32	31 30	29 28	27 26 ... 24	23 22 21 20	19 ... 17 16	15 14	13 ... 10	9 ... 0
NPU_SET_IFM_BASE0	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 0 0 0 0 0
NPU_SET_IFM_BASE1	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 0 0 0 0 1
NPU_SET_IFM_BASE2	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 0 0 0 1 0
NPU_SET_IFM_BASE3	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 0 0 0 1 1
NPU_SET_IFM_STRIDE_X	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 0 0 1 0 0
NPU_SET_IFM_STRIDE_Y	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 0 0 1 0 1
NPU_SET_IFM_STRIDE_C	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 0 0 1 1 0
NPU_SET_OFM_BASE0	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 1 0 0 0 0
NPU_SET_OFM_BASE1	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 1 0 0 0 1
NPU_SET_OFM_BASE2	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 1 0 0 1 0
NPU_SET_OFM_BASE3	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 1 0 0 1 1
NPU_SET_OFM_STRIDE_X	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 1 0 1 0 0
NPU_SET_OFM_STRIDE_Y	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 1 0 1 0 1
NPU_SET_OFM_STRIDE_C	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 0 1 0 1 1 0
NPU_SET_WEIGHT_BASE	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 0 0 0 0 0
NPU_SET_WEIGHT_LENGTH	length				Reserved					0 1	Reserved	0 0 0 0 1 0 0 0 0 1
NPU_SET_SCALE_BASE	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 0 0 0 1 0
NPU_SET_SCALE_LENGTH	Reserved		length		Reserved					0 1	Reserved	0 0 0 0 1 0 0 0 1 1
NPU_SET_OFM_SCALE	R	scale			RM		Res	dbl_rnd	shift	0 1	Reserved	0 0 0 0 1 0 0 1 0 0
NPU_SET_IFM_SCALE	R	scale			Res	RM	Res	dbl_rnd	shift	0 1	Reserved	0 0 0 0 1 0 0 1 0 1
NPU_SET_IFM2_SCALE	R	scale			Res	RM	Res	dbl_rnd	shift	0 1	Reserved	0 0 0 0 1 0 0 1 1 0
NPU_SET_OP_SCALAR	scalar				Reserved					0 1	Reserved	0 0 0 0 1 0 0 1 1 1
NPU_SET_DMA0_SRC	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 0 0 0
NPU_SET_DMA0_DST	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 0 0 1
NPU_SET_DMA0_LEN	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 0 1 0
NPU_SET_DMA0_SRC_STRIDE0	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 0 1 1
NPU_SET_DMA0_SRC_STRIDE1	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 1 0 0
NPU_SET_DMA0_DST_STRIDE0	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 1 0 1
NPU_SET_DMA0_DST_STRIDE1	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 1 1 0
NPU_SET_DMA0_IDX	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 0 1 1 1
NPU_SET_DMA0_IDX_MAX	R	idx_max			Reserved					0 1	Reserved	0 0 0 0 1 1 1 0 0 0
NPU_SET_DMA0_IDX_SKIP1	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 0 0 1 1 1 0 0 1
NPU_SET_IFM2_BASE0	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 0 0 0 0 0
NPU_SET_IFM2_BASE1	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 0 0 0 0 1
NPU_SET_IFM2_BASE2	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 0 0 0 1 0
NPU_SET_IFM2_BASE3	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 0 0 0 1 1
NPU_SET_IFM2_STRIDE_X	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 0 0 1 0 0
NPU_SET_IFM2_STRIDE_Y	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 0 0 1 0 1
NPU_SET_IFM2_STRIDE_C	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 0 0 1 1 0
NPU_SET_WEIGHT1_BASE	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 1 0 0 0 0
NPU_SET_WEIGHT1_LENGTH	length				Reserved					0 1	Reserved	0 0 1 0 0 1 0 0 0 1
NPU_SET_WEIGHT2_BASE	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 1 0 0 1 0
NPU_SET_WEIGHT2_LENGTH	length				Reserved					0 1	Reserved	0 0 1 0 0 1 0 0 1 1
NPU_SET_WEIGHT3_BASE	addr_lo				Reserved			addr_hi		0 1	Reserved	0 0 1 0 0 1 0 1 0 0
NPU_SET_WEIGHT3_LENGTH	length				Reserved					0 1	Reserved	0 0 1 0 0 1 0 1 0 1
NPU_SET_RESIZE_X_STEP	Reserved	blk_step_mod	Reserved	ne_step_mod	R	blk_step_int			one_step_int	0 1	Reserved	0 0 1 0 0 1 0 1 1 0
NPU_SET_RESIZE_Y_STEP	Reserved	blk_step_mod	Reserved	ne_step_mod	R	blk_step_int			one_step_int	0 1	Reserved	0 0 1 0 0 1 0 1 1 1
NPU_OP_BRANCH	branch_target				Reserved				CC	0 1	Reserved	0 1 0 0 0 0 0 0 0 0

3.11.2 NPU_OP_BRANCH instruction

Branch to new location.

Branch the command stream conditionally or unconditionally

Execution

```
// Command stream branch
if (branch_cond == rf_true)
{
    // Wait for branch condition to become available
    NPU_OP_KERNEL_WAIT(n=0);
}
// Check branch target is valid, even if branch is not taken
assert((branch_target & 3)==0)
// Although the valid range of QREAD is 0 <= QREAD <= QSIZE,
// the branch offset must branch to a valid command to fetch and so
// be less than QSIZE (and is automatically >=0 as it is unsigned).
if (branch_target >= QSIZE.QSIZE) {
    STATUS.branch_fault = 1;
    raise_error();
}
if (branch_cond == always ||
    (branch_cond == rf_true && COND_STATUS.result_flag==1))
{
    // Take the branch
    QREAD = branch_target;
}
```

Encoding

Figure 3-132: NPU_OP_BRANCH bit assignments

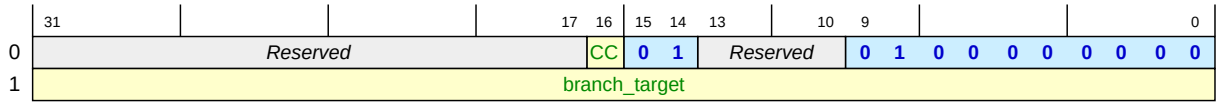


Table 3-160: Instruction NPU_OP_BRANCH encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_op_branch.	npu_op_branch
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[16]	branch_cond (CC)	This value is an enumeration of type branch_cond_t. This field can contain the following values: **0b0** always - Always **0b1** rf_true - Result flag is true	-
[31:17]	Reserved	-	-
[63:32]	branch_target	This value is an unsigned integer.	-

3.11.3 NPU_SET_DMA0_DST instruction

DMA user channel 0 destination byte offset from DMA0_DST_REGION.

Set DMA user channel 0 destination byte offset from DMA0_DST_REGION.

Any byte alignment is permitted for an external destination. Internal destination must be 16 byte aligned.

Execution

```
set_register()
```

Encoding

Figure 3-133: NPU_SET_DMA0_DST bit assignments

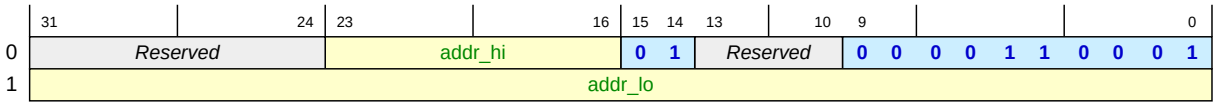


Table 3-161: Instruction NPU_SET_DMA0_DST encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_dst.	npu_set_dma0_dst
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.4 NPU_SET_DMA0_DST_STRIDE0 instruction

Destination byte stride after each 1D transfer.

Set byte stride between the starting destination address of each inner (1D) transfer (2D/3D mode) (any alignment).

This register must be set if either SRC or DST uses 2D/3D mode or DST indexing is used.

This value is signed 2's complement and can be negative, zero or positive.

Execution

```
set_register()
```

Encoding

Figure 3-134: NPU_SET_DMA0_DST_STRIDE0 bit assignments

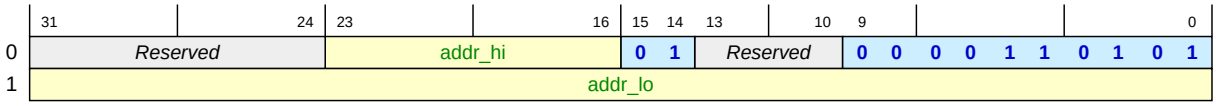


Table 3-162: Instruction NPU_SET_DMA0_DST_STRIDE0 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_dst_stride0.	npu_set_dma0_dst_stride0
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is a signed integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.5 NPU_SET_DMA0_DST_STRIDE1 instruction

Destination byte stride after 2D transfer.

Set byte stride between the starting destination address of each 2D transfer (3D mode) (any alignment).

This register must be set if either SRC or DST uses 3D mode.

This value is signed 2's complement and can be negative, zero or positive.

Execution

```
set_register()
```

Encoding

Figure 3-135: NPU_SET_DMA0_DST_STRIDE1 bit assignments

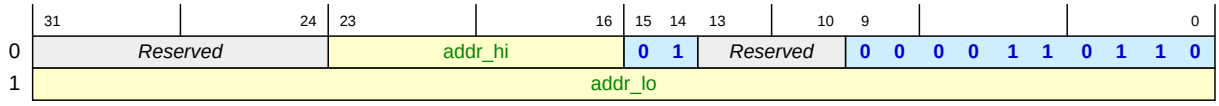


Table 3-163: Instruction NPU_SET_DMA0_DST_STRIDE1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_dst_stride1.	npu_set_dma0_dst_stride1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is a signed integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.6 NPU_SET_DMA0_IDX instruction

DMA channel 0 index array address.

Set the base address of DMA0 index channel for gather/scatter operations, 4 byte aligned.

This register must be set if either SRC or DST uses indexing mode.

If DST uses indexing, a scatter operation, then the indices must be distinct.

Execution

```
set_register()
```

Encoding

Figure 3-136: NPU_SET_DMA0_IDX bit assignments

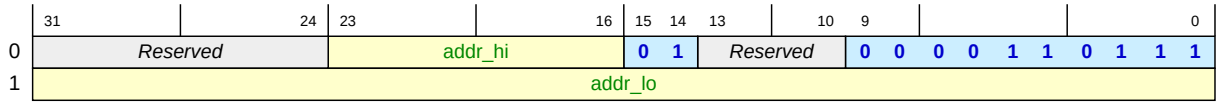


Table 3-164: Instruction NPU_SET_DMA0_IDX encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_idx.	npu_set_dma0_idx
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.7 NPU_SET_DMA0_IDX_MAX instruction

DMA channel 0 index maximum value.

Set maximum permitted value for the index operations.

This register must be set if either SRC or DST uses index mode.

Execution

```
set_register()
```

Encoding

Figure 3-137: NPU_SET_DMA0_IDX_MAX bit assignments

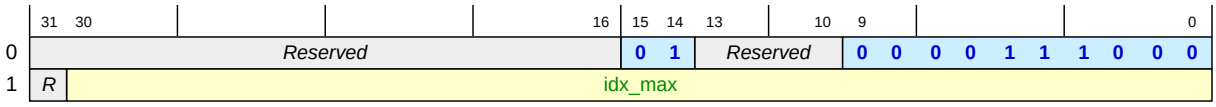


Table 3-165: Instruction NPU_SET_DMA0_IDX_MAX encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_idx_max.	npu_set_dma0_idx_max
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[31:16]	Reserved	-	-
[62:32]	idx_max	This value is an unsigned integer.	-
[63]	Reserved	-	-

3.11.8 NPU_SET_DMA0_IDX_SKIP1 instruction

Index byte distance to skip in index after each 2D transfer.

Set byte distance to skip in index after each 2D transfer, 4 byte aligned.

This register must be set if either SRC or DST uses 3D mode and one of SRC or DST uses indexing mode.

This value is signed 2's complement and can be negative, zero or positive.

Execution

```
set_register()
```

Encoding

Figure 3-138: NPU_SET_DMA0_IDX_SKIP1 bit assignments

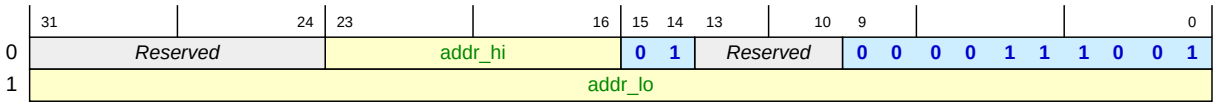


Table 3-166: Instruction NPU_SET_DMA0_IDX_SKIP1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_idx_skip1.	npu_set_dma0_idx_skip1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is a signed integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.9 NPU_SET_DMA0_LEN instruction

DMA user channel 0 transfer length in bytes for each 1D transfer.

Set DMA user channel 0 transfer length in bytes for each 1D transfer.

This is the size in bytes of the innermost (1D) transfer and the total transfer size for a 3D transfer is DMA0_LEN*DMA0_SIZE0*DMA_SIZE1. The length must be greater than zero. Any byte alignment

is permitted for an external destination. For an internal destination the length must be a multiple of 16.

Execution

```
set_register()
```

Encoding

Figure 3-139: NPU_SET_DMA0_LEN bit assignments

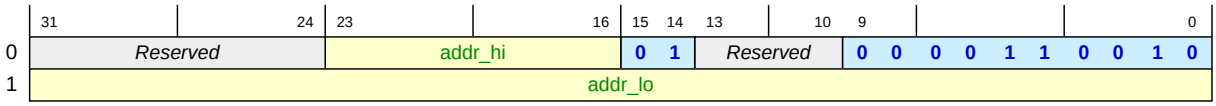


Table 3-167: Instruction NPU_SET_DMA0_LEN encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_len.	npu_set_dma0_len
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.10 NPU_SET_DMA0_SRC instruction

DMA user channel 0 source byte offset from DMA0_SRC_REGION.

Set DMA user channel 0 source byte offset from DMA0_SRC_REGION.

Any byte alignment is permitted for an external source. Internal source is not permitted.

Execution

```
set_register()
```

Encoding

Figure 3-140: NPU_SET_DMA0_SRC bit assignments

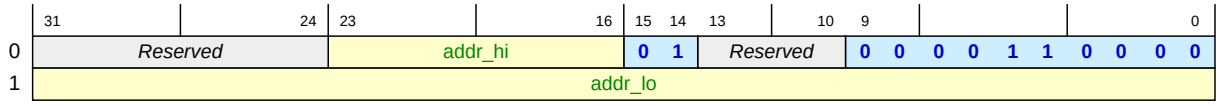


Table 3-168: Instruction NPU_SET_DMA0_SRC encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_src.	npu_set_dma0_src
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.11 NPU_SET_DMA0_SRC_STRIDE0 instruction

Source byte stride after each 1D transfer.

Set byte stride between the starting source address of each inner (1D) transfer (2D/3D mode) (any alignment).

This register must be set if either SRC or DST uses 2D/3D mode or SRC indexing is used.

This value is signed 2's complement and can be negative, zero or positive.

Execution

```
set_register()
```

Encoding

Figure 3-141: NPU_SET_DMA0_SRC_STRIDE0 bit assignments

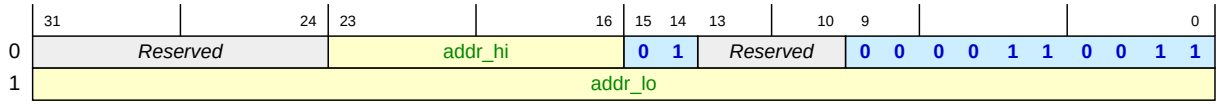


Table 3-169: Instruction NPU_SET_DMA0_SRC_STRIDE0 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_src_stride0.	npu_set_dma0_src_stride0
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is a signed integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.12 NPU_SET_DMA0_SRC_STRIDE1 instruction

Source byte stride after each 2D transfer.

Set byte stride between the starting source address of each 2D transfer (3D mode) (any alignment).

This register must be set if either SRC or DST uses 3D mode.

This value is signed 2's complement and can be negative, zero or positive.

Execution

```
set_register()
```

Encoding

Figure 3-142: NPU_SET_DMA0_SRC_STRIDE1 bit assignments

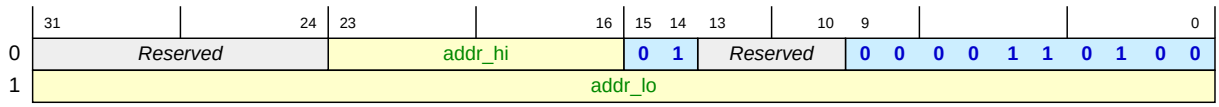


Table 3-170: Instruction NPU_SET_DMA0_SRC_STRIDE1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_dma0_src_stride1.	npu_set_dma0_src_stride1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is a signed integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.13 NPU_SET_IFM2_BASE0 instruction

IFM2 Tile 0 address.

Set IFM2 tile 0 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-143: NPU_SET_IFM2_BASE0 bit assignments

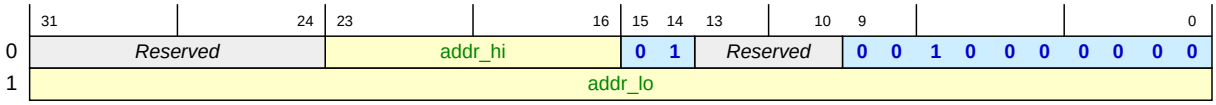


Table 3-171: Instruction NPU_SET_IFM2_BASE0 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_base0.	npu_set_ifm2_base0
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.14 NPU_SET_IFM2_BASE1 instruction

IFM2 Tile 1 address.

Set IFM2 tile 1 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-144: NPU_SET_IFM2_BASE1 bit assignments

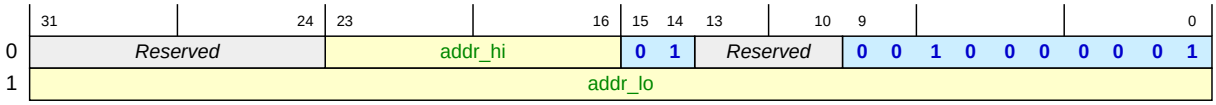


Table 3-172: Instruction NPU_SET_IFM2_BASE1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_base1.	npu_set_ifm2_base1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.15 NPU_SET_IFM2_BASE2 instruction

IFM2 Tile 2 address.

Set IFM2 tile 2 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-145: NPU_SET_IFM2_BASE2 bit assignments

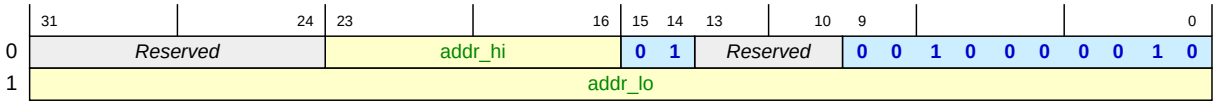


Table 3-173: Instruction NPU_SET_IFM2_BASE2 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_base2.	npu_set_ifm2_base2
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.16 NPU_SET_IFM2_BASE3 instruction

IFM2 Tile 3 address.

Set IFM2 tile 3 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-146: NPU_SET_IFM2_BASE3 bit assignments

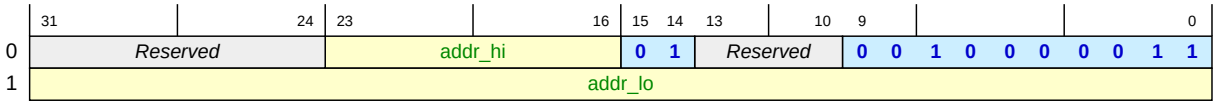


Table 3-174: Instruction NPU_SET_IFM2_BASE3 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_base3.	npu_set_ifm2_base3
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.17 NPU_SET_IFM2_SCALE instruction

IFM2 input scale.

Set input scale and rounding mode for IFM2 input.

Execution

```
set_register()
```

Encoding

Figure 3-147: NPU_SET_IFM2_SCALE bit assignments

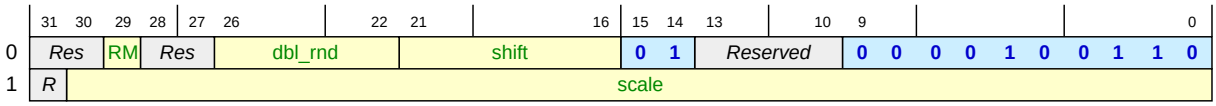


Table 3-175: Instruction NPU_SET_IFM2_SCALE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_scale.	npu_set_ifm2_scale
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[21:16]	shift	This value is an unsigned integer.	-
[26:22]	dbl_rnd	This value is an unsigned integer.	-
[28:27]	Reserved	-	-

Bits	Name	Description	Reset
[29]	round_mode (RM)	This value is an enumeration of type round_mode_ifm_t. This field can contain the following values: **0b0** double_symmetric - Double rounding symmetric about 0 **0b1** natural - Round to nearest with 0.5 round to +infinity	-
[31:30]	Reserved	-	-
[62:32]	scale	This value is an unsigned integer.	-
[63]	Reserved	-	-

3.11.18 NPU_SET_IFM2_STRIDE_C instruction

IFM2 byte stride between channel blocks.

Byte stride between WB16 blocks in NHCWB16 format. Each block has size `width*(16*element_size)` bytes.

This stride must be a positive multiple of (16*element_size) bytes for NHCWB16 activation format, or is ignored for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-148: NPU_SET_IFM2_STRIDE_C bit assignments

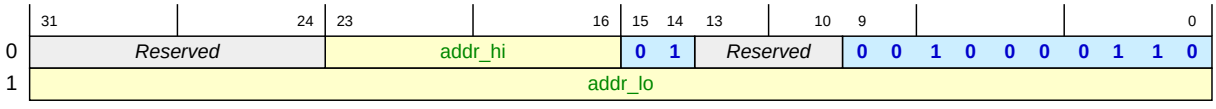


Table 3-176: Instruction NPU_SET_IFM2_STRIDE_C encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_stride_c.	npu_set_ifm2_stride_c
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.19 NPU_SET_IFM2_STRIDE_X instruction

IFM2 byte stride between horizontal values.

This stride is ignored for NHCWB16 activation format, and must be a positive multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-149: NPU_SET_IFM2_STRIDE_X bit assignments

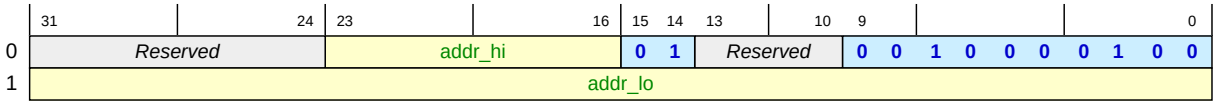


Table 3-177: Instruction NPU_SET_IFM2_STRIDE_X encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_stride_x.	npu_set_ifm2_stride_x
[13:10]	Reserved	-	-

Bits	Name	Description	Reset
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.20 NPU_SET_IFM2_STRIDE_Y instruction

IFM2 byte stride between vertical values.

This stride must be a positive multiple of (16*element_size) bytes for NHCWB16 activation format, or a positive multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM2_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-150: NPU_SET_IFM2_STRIDE_Y bit assignments

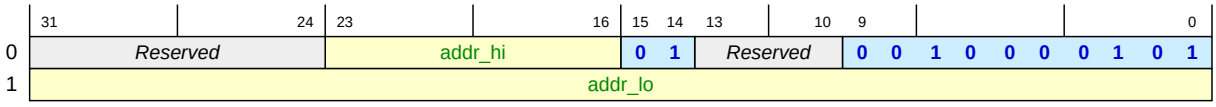


Table 3-178: Instruction NPU_SET_IFM2_STRIDE_Y encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm2_stride_y.	npu_set_ifm2_stride_y
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-

Bits	Name	Description	Reset
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.21 NPU_SET_IFM_BASE0 instruction

IFM Tile 0 address.

Set IFM tile 0 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-151: NPU_SET_IFM_BASE0 bit assignments

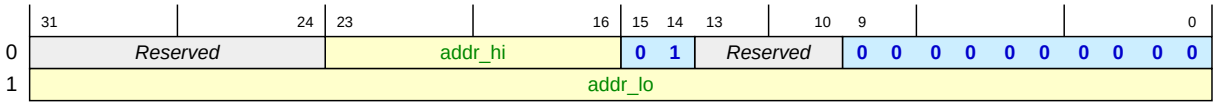


Table 3-179: Instruction NPU_SET_IFM_BASE0 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_base0.	npu_set_ifm_base0
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.22 NPU_SET_IFM_BASE1 instruction

IFM Tile 1 address.

Set IFM tile 1 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-152: NPU_SET_IFM_BASE1 bit assignments

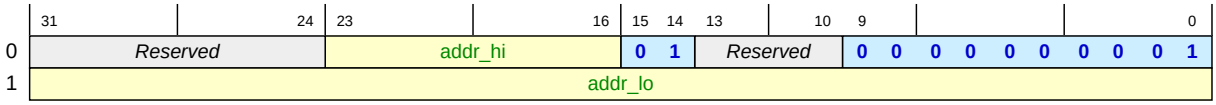


Table 3-180: Instruction NPU_SET_IFM_BASE1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_base1.	npu_set_ifm_base1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.23 NPU_SET_IFM_BASE2 instruction

IFM Tile 2 address.

Set IFM tile 2 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-153: NPU_SET_IFM_BASE2 bit assignments

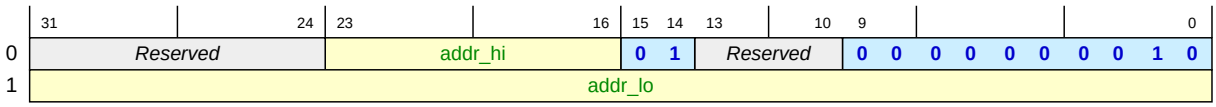


Table 3-181: Instruction NPU_SET_IFM_BASE2 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_base2.	npu_set_ifm_base2
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.24 NPU_SET_IFM_BASE3 instruction

IFM Tile 3 address.

Set IFM tile 3 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-154: NPU_SET_IFM_BASE3 bit assignments

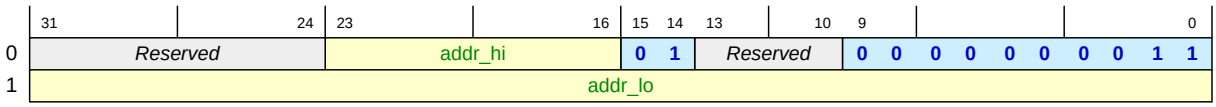


Table 3-182: Instruction NPU_SET_IFM_BASE3 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_base3.	npu_set_ifm_base3
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.25 NPU_SET_IFM_SCALE instruction

IFM input scale.

Set input scale and rounding mode for IFM input.

Execution

```
set_register()
```

Encoding

Figure 3-155: NPU_SET_IFM_SCALE bit assignments

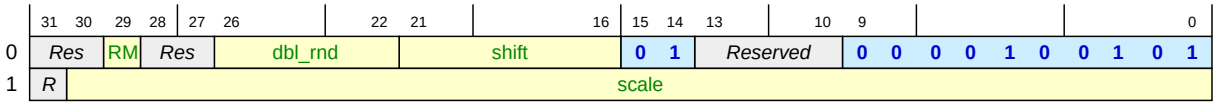


Table 3-183: Instruction NPU_SET_IFM_SCALE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_scale.	npu_set_ifm_scale
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[21:16]	shift	This value is an unsigned integer.	-
[26:22]	dbl_rnd	This value is an unsigned integer.	-
[28:27]	Reserved	-	-
[29]	round_mode (RM)	This value is an enumeration of type round_mode_ifm_t. This field can contain the following values: **0b0** double_symmetric - Double rounding symmetric about 0 **0b1** natural - Round to nearest with 0.5 round to +infinity	-
[31:30]	Reserved	-	-
[62:32]	scale	This value is an unsigned integer.	-
[63]	Reserved	-	-

3.11.26 NPU_SET_IFM_STRIDE_C instruction

IFM byte stride between channel blocks.

Byte stride between WB16 blocks in NHCWB16 format. Each block has size `width*(16*element_size)` bytes.

This stride must be a positive multiple of (16*element_size) bytes for NHCWB16 activation format, or is ignored for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-156: NPU_SET_IFM_STRIDE_C bit assignments

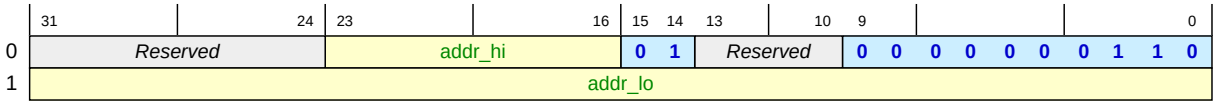


Table 3-184: Instruction NPU_SET_IFM_STRIDE_C encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_stride_c.	npu_set_ifm_stride_c
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.27 NPU_SET_IFM_STRIDE_X instruction

IFM byte stride between horizontal values.

This stride is ignored for NHCWB16 activation format, and must be a positive multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-157: NPU_SET_IFM_STRIDE_X bit assignments

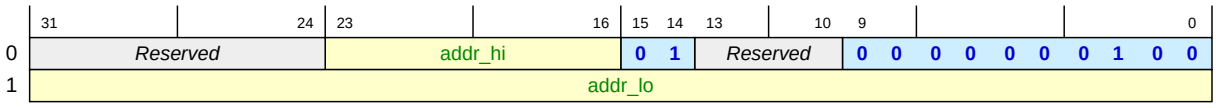


Table 3-185: Instruction NPU_SET_IFM_STRIDE_X encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_stride_x.	npu_set_ifm_stride_x
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.28 NPU_SET_IFM_STRIDE_Y instruction

IFM byte stride between vertical values.

This stride must be a positive multiple of (16*element_size) bytes for NHCWB16 activation format, or a positive multiple of the element size for NHWC format.

The following rules must be observed where OP2 is the next NPU_OP_KERNEL operation following this command.

(a) There must not be an NPU_SET_IFM_PRECISION operation between this operation and OP2.

(b) If OP2 uses this register then the command must be present unless the value has been set in a previous non-chained operation and all intervening operations are not chained.

Execution

```
set_register()
```

Encoding

Figure 3-158: NPU_SET_IFM_STRIDE_Y bit assignments

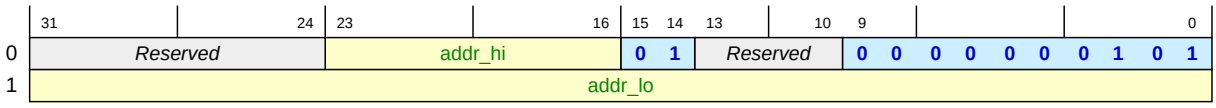


Table 3-186: Instruction NPU_SET_IFM_STRIDE_Y encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ifm_stride_y.	npu_set_ifm_stride_y
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.29 NPU_SET_OFM_BASE0 instruction

OFM Tile 0 address.

Set OFM tile 0 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

Execution

```
set_register()
```

Encoding

Figure 3-159: NPU_SET_OFM_BASE0 bit assignments

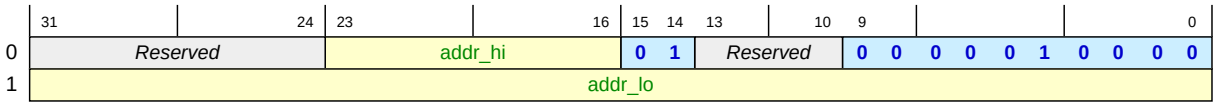


Table 3-187: Instruction NPU_SET_OFM_BASE0 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_base0.	npu_set_ofm_base0
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.30 NPU_SET_OFM_BASE1 instruction

OFM Tile 1 address.

Set OFM tile 1 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

Execution

```
set_register()
```

Encoding

Figure 3-160: NPU_SET_OFM_BASE1 bit assignments

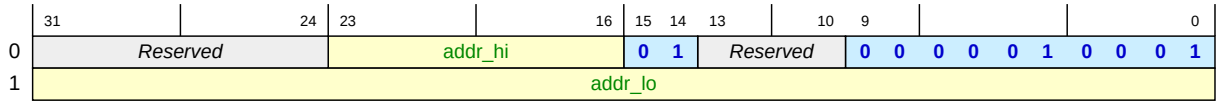


Table 3-188: Instruction NPU_SET_OFM_BASE1 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_base1.	npu_set_ofm_base1
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.31 NPU_SET_OFM_BASE2 instruction

OFM Tile 2 address.

Set OFM tile 2 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

Execution

```
set_register()
```

Encoding

Figure 3-161: NPU_SET_OFM_BASE2 bit assignments

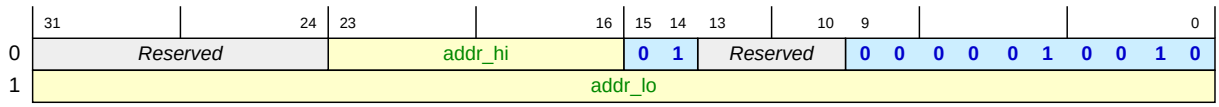


Table 3-189: Instruction NPU_SET_OFM_BASE2 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_base2.	npu_set_ofm_base2
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.32 NPU_SET_OFM_BASE3 instruction

OFM Tile 3 address.

Set OFM tile 3 address offset.

This offset must be a multiple of 16 bytes for NHCWB16 activation format, or a multiple of the element size for NHWC format.

Execution

```
set_register()
```

Encoding

Figure 3-162: NPU_SET_OFM_BASE3 bit assignments

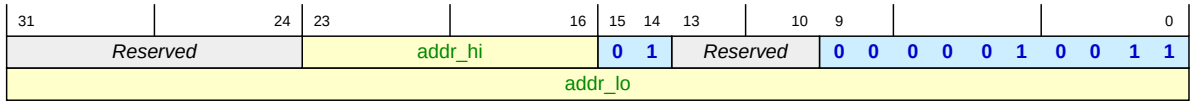


Table 3-190: Instruction NPU_SET_OFM_BASE3 encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_base3.	npu_set_ofm_base3
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.33 NPU_SET_OFM_SCALE instruction

OFM scale.

Set the output (post operation) scaling factor and rounding mode.

Execution

```
set_register()
```

Encoding

Figure 3-163: NPU_SET_OFM_SCALE bit assignments

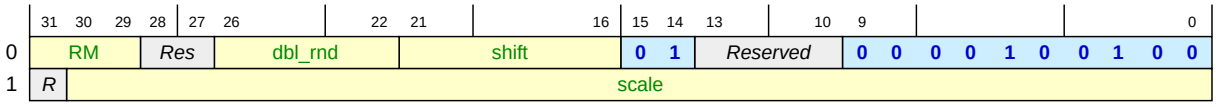


Table 3-191: Instruction NPU_SET_OFM_SCALE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_scale.	npu_set_ofm_scale
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[21:16]	shift	This value is an unsigned integer.	-
[26:22]	dbl_rnd	This value is an unsigned integer.	-
[28:27]	Reserved	-	-

Bits	Name	Description	Reset
[31:29]	round_mode (RM)	This value is an enumeration of type round_mode_ofm_t. This field can contain the following values: **0b000** double_symmetric - Double rounding symmetric about 0 **0b001** natural - Round to nearest with 0.5 round to +infinity **0b010** double_asymmetric - Double rounding asymmetric **0b011** symmetric - Round to nearest with 0.5 round away from 0 **0b100** truncate_to_zero - Truncate towards 0 **0b101** truncate_to_lower - Truncate towards -infinity **0b110..0b111** Reserved	-
[62:32]	scale	This value is an unsigned integer.	-
[63]	Reserved	-	-

3.11.34 NPU_SET_OFM_STRIDE_C instruction

OFM byte stride between channel blocks.

Byte stride between WB16 blocks in NHCWB16 format. Each block has size `width*(16*element_size)` bytes.

This stride must be a positive multiple of (16*element_size) bytes for NHCWB16 activation format, or is ignored for NHWC format.

Execution

```
set_register()
```

Encoding

Figure 3-164: NPU_SET_OFM_STRIDE_C bit assignments

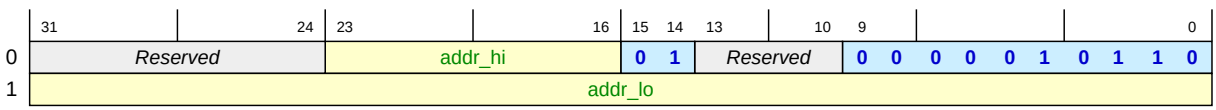


Table 3-192: Instruction NPU_SET_OFM_STRIDE_C encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_stride_c.	npu_set_ofm_stride_c
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.35 NPU_SET_OFM_STRIDE_X instruction

OFM byte stride between horizontal values.

This stride is ignored for NHCWB16 activation format, and must be a positive multiple of the element size for NHWC format.

Execution

```
set_register()
```

Encoding

Figure 3-165: NPU_SET_OFM_STRIDE_X bit assignments

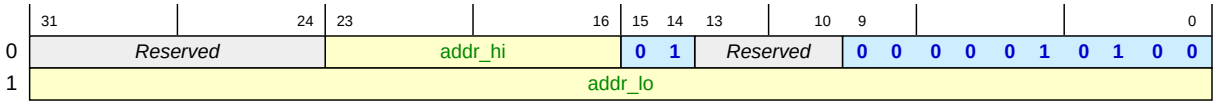


Table 3-193: Instruction NPU_SET_OFM_STRIDE_X encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_stride_x.	npu_set_ofm_stride_x
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.36 NPU_SET_OFM_STRIDE_Y instruction

OFM byte stride between vertical values.

This stride must be a positive multiple of (16*element_size) bytes for NHCWB16 activation format, or a positive multiple of the element size for NHWC format.

Execution

```
set_register()
```

Encoding

Figure 3-166: NPU_SET_OFM_STRIDE_Y bit assignments

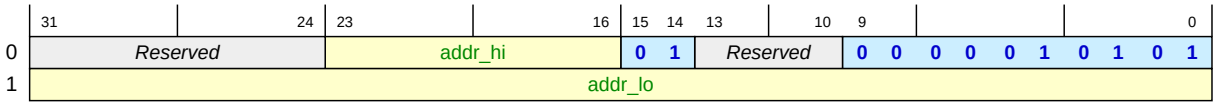


Table 3-194: Instruction NPU_SET_OFM_STRIDE_Y encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_ofm_stride_y.	npu_set_ofm_stride_y
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.37 NPU_SET_OP_SCALAR instruction

Operation scalar value.

Set scalar value to be used instead if IFM for scalar broadcast operations

Execution

```
set_register()
```

Encoding

Figure 3-167: NPU_SET_OP_SCALAR bit assignments

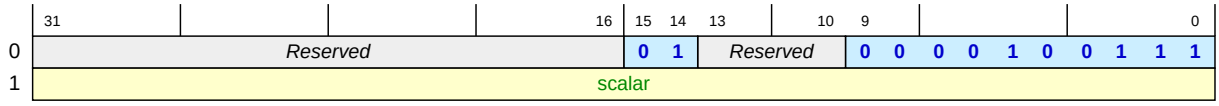


Table 3-195: Instruction NPU_SET_OP_SCALAR encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_op_scalar.	npu_set_op_scalar
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[31:16]	Reserved	-	-
[63:32]	scalar	This value is a signed integer.	-

3.11.38 NPU_SET_RESIZE_X_STEP instruction

Resize X axis step parameters.

The scale applied to this axis is defined as *scale_n*/*scale_d*. For a scale up this value is greater than one and for a scale down the value is less than one. The step between each sampling position in the IFM is *scale_d*/*scale_n*.

Output coordinate *v* maps to input sampling coordinate *u* = (*v* * *scale_d* + *offset*)/*scale_n*.

This command stores calculated step values for a step of one OFM X unit and (ofm_block_size_x - 1) OFM units

Execution

```
set_register()
```

Encoding

Figure 3-168: NPU_SET_RESIZE_X_STEP bit assignments

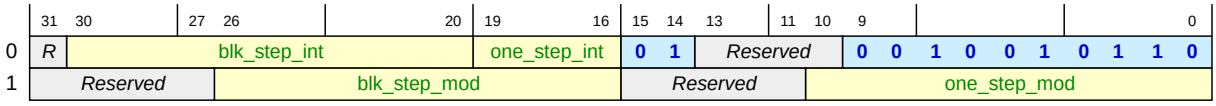


Table 3-196: Instruction NPU_SET_RESIZE_X_STEP encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_resize_x.	npu_set_resize_x
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[19:16]	one_step_int	This value is an unsigned integer.	-
[30:20]	blk_step_int	This value is an unsigned integer.	-
[31]	Reserved	-	-
[42:32]	one_step_mod	This value is an unsigned integer.	-
[47:43]	Reserved	-	-
[58:48]	blk_step_mod	This value is an unsigned integer.	-
[63:59]	Reserved	-	-

3.11.39 NPU_SET_RESIZE_Y_STEP instruction

Resize Y axis step parameters.

The scale applied to this axis is defined as $scale\_n/scale\_d$. For a scale up this value is greater than one and for a scale down the value is less than one. The step between each sampling position in the IFM is $scale\_d/scale\_n$.

Output coordinate v maps to input sampling coordinate $u = (v * scale\_d + offset)/scale\_n$.

This command stores calculated step values for a step of one OFM Y unit and (ofm_block_size_y - 1) OFM units

Execution

```
set_register()
```

Encoding

Figure 3-169: NPU_SET_RESIZE_Y_STEP bit assignments

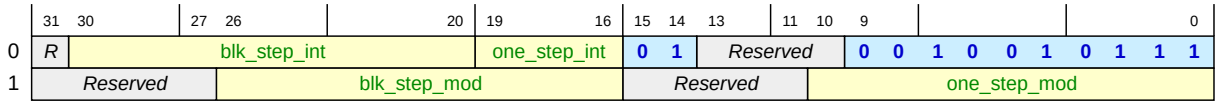


Table 3-197: Instruction NPU_SET_RESIZE_Y_STEP encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_resize_y.	npu_set_resize_y
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[19:16]	one_step_int	This value is an unsigned integer.	-
[30:20]	blk_step_int	This value is an unsigned integer.	-
[31]	Reserved	-	-
[42:32]	one_step_mod	This value is an unsigned integer.	-
[47:43]	Reserved	-	-
[58:48]	blk_step_mod	This value is an unsigned integer.	-
[63:59]	Reserved	-	-

3.11.40 NPU_SET_SCALE_BASE instruction

Scale and bias stream input byte offset from SCALE_REGION.

Set scale and bias stream input byte offset from SCALE_REGION.

Execution

```
set_register()
```

Encoding

Figure 3-170: NPU_SET_SCALE_BASE bit assignments

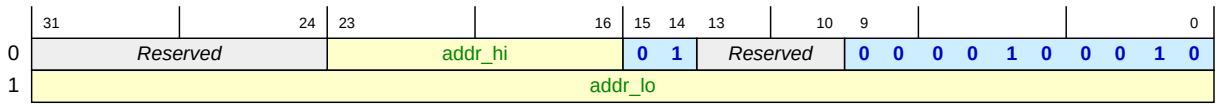


Table 3-198: Instruction NPU_SET_SCALE_BASE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_scale_base.	npu_set_scale_base
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.41 NPU_SET_SCALE_LENGTH instruction

Scale and bias stream input byte length.

Set scale and bias stream input byte length.

This offset must be 16 byte (128-bit) aligned.

Execution

```
set_register()
```

Encoding

Figure 3-171: NPU_SET_SCALE_LENGTH bit assignments

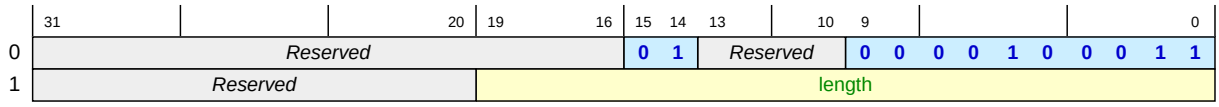


Table 3-199: Instruction NPU_SET_SCALE_LENGTH encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_scale_length.	npu_set_scale_length
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[31:16]	Reserved	-	-
[51:32]	length	This value is an unsigned integer.	-
[63:52]	Reserved	-	-

3.11.42 NPU_SET_WEIGHT1_BASE instruction

Weight stream byte offset in WEIGHT_REGION for weight decoder 1.

This offset must be 16 byte (128-bit) aligned.

Execution

```
set_register()
```

Encoding

Figure 3-172: NPU_SET_WEIGHT1_BASE bit assignments

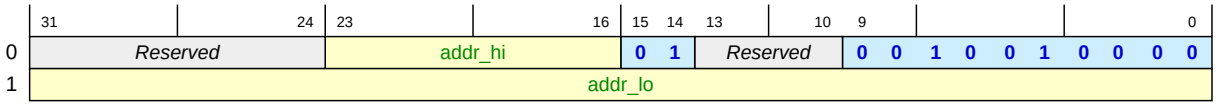


Table 3-200: Instruction NPU_SET_WEIGHT1_BASE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight1_base.	npu_set_weight1_base
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.43 NPU_SET_WEIGHT1_LENGTH instruction

Weight stream byte length for weight decoder 1.

Set weight stream byte length for weight decoder 1.

This length must be a multiple of 16 bytes.

Execution

```
set_register()
```

Encoding

Figure 3-173: NPU_SET_WEIGHT1_LENGTH bit assignments

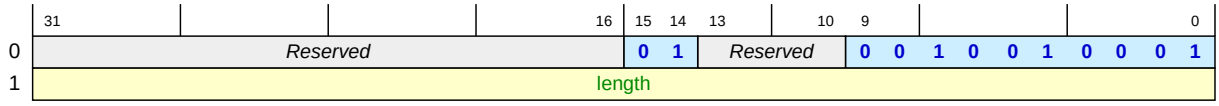


Table 3-201: Instruction NPU_SET_WEIGHT1_LENGTH encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight1_length.	npu_set_weight1_length
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[31:16]	Reserved	-	-
[63:32]	length	This value is an unsigned integer.	-

3.11.44 NPU_SET_WEIGHT2_BASE instruction

Weight stream byte offset in WEIGHT_REGION for weight decoder 2.

Weight stream byte offset in WEIGHT_REGION for weight decoder 2.

This offset must be 16 byte (128-bit) aligned.

Execution

```
set_register()
```

Encoding

Figure 3-174: NPU_SET_WEIGHT2_BASE bit assignments

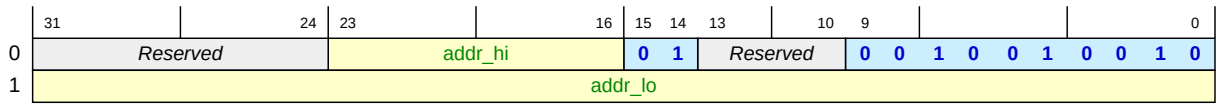


Table 3-202: Instruction NPU_SET_WEIGHT2_BASE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight2_base.	npu_set_weight2_base
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.45 NPU_SET_WEIGHT2_LENGTH instruction

Weight stream byte length for weight decoder 2.

Set weight stream byte length for weight decoder 2.

This length must be a multiple of 16 bytes.

Execution

```
set_register()
```

Encoding

Figure 3-175: NPU_SET_WEIGHT2_LENGTH bit assignments

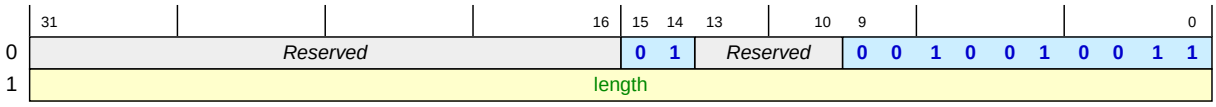


Table 3-203: Instruction NPU_SET_WEIGHT2_LENGTH encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight2_length.	npu_set_weight2_length
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[31:16]	Reserved	-	-
[63:32]	length	This value is an unsigned integer.	-

3.11.46 NPU_SET_WEIGHT3_BASE instruction

Weight stream byte offset in WEIGHT_REGION for weight decoder 3.

This offset must be 16 byte (128-bit) aligned.

Execution

```
set_register()
```

Encoding

Figure 3-176: NPU_SET_WEIGHT3_BASE bit assignments

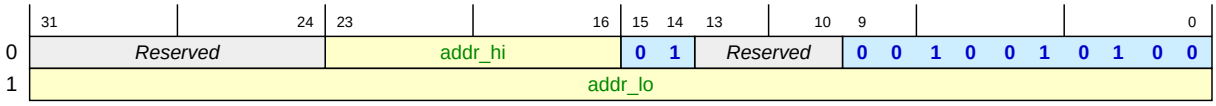


Table 3-204: Instruction NPU_SET_WEIGHT3_BASE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight3_base.	npu_set_weight3_base
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.47 NPU_SET_WEIGHT3_LENGTH instruction

Weight stream byte length for weight decoder 3.

Set Weight stream byte length for weight decoder 3.

This length must be a multiple of 16 bytes.

Execution

```
set_register()
```

Encoding

Figure 3-177: NPU_SET_WEIGHT3_LENGTH bit assignments

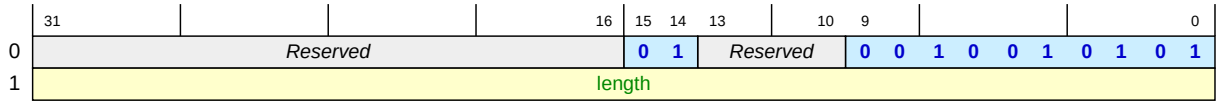


Table 3-205: Instruction NPU_SET_WEIGHT3_LENGTH encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight3_length.	npu_set_weight3_length
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[31:16]	Reserved	-	-
[63:32]	length	This value is an unsigned integer.	-

3.11.48 NPU_SET_WEIGHT_BASE instruction

Weight stream byte offset in WEIGHT_REGION.

Set weight stream byte offset in WEIGHT_REGION.

This offset must be 16 byte (128-bit) aligned.

Execution

```
set_register()
```

Encoding

Figure 3-178: NPU_SET_WEIGHT_BASE bit assignments

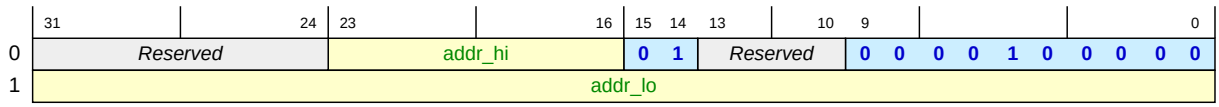


Table 3-206: Instruction NPU_SET_WEIGHT_BASE encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight_base.	npu_set_weight_base
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[23:16]	addr_hi	This value is an unsigned integer.	-
[31:24]	Reserved	-	-
[63:32]	addr_lo	This value is an unsigned integer.	-

3.11.49 NPU_SET_WEIGHT_LENGTH instruction

Weight stream byte length.

Set weight stream byte length.

This length must be a multiple of 16 bytes for standard weight decoder streams. This length must be a multiple of 32 bytes for fast weight decoder streams.

Execution

```
set_register()
```

Encoding

Figure 3-179: NPU_SET_WEIGHT_LENGTH bit assignments

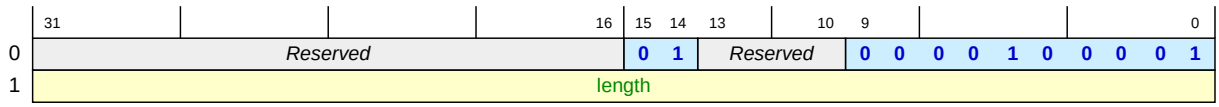


Table 3-207: Instruction NPU_SET_WEIGHT_LENGTH encoding layout

Bits	Name	Description	Reset
[9:0]	opcode (OP)	This value is an enumeration of type cmd1_opcode_t. It must have the exact value npu_set_weight_length.	npu_set_weight_length
[13:10]	Reserved	-	-
[15:14]	control (CTRL)	This value is an enumeration of type cmd_ctrl_t. It must have the exact value cmd1_ctrl.	cmd1_ctrl
[31:16]	Reserved	-	-
[63:32]	length	This value is an unsigned integer.	-

4. Operators and performance

This section provides information on supported data types, operators, and operations, and details the convolution and elementwise performance of the Ethos™-U85 NPU.

For the full list of supported operators and their constraints, see the Vela Compiler: https://gitlab.arm.com/artificial-intelligence/ethos-u/ethos-u-vela

4.1 Convolution performance

The following tables detail the convolution performance of the Ethos™-U85 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™-U85$_{128}$

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.

The definition of the k_rnd function: k_rnd(k,s,r) = floor(k/s)*s + max(k%s,r)


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.)


Note

With sparsity enabled, all MACs per cycle numbers with * effectively doubles as half of the multiplies for the convolutions are skipped in the hardware. This skipping can be done because half of the weights are guaranteed to be 0.

Table 4-1: 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	1*h	2*w	8*d	no sparsity: 32*n sparsity: 64*n	128*
CONV2D (kernel first) ifm_depth<=8, no sparsity	k_rnd(kh*kw,5,4)=1*k	1*h	2*w	8*d	8	128
CONV2D (kernel first) ifm_depth>8	no sparsity: k_rnd(kh*kw,5,2)=1*k sparsity: k_rnd(kh*kw,10,4)=1*k	1*h	2*w	8*d	16*n	128*
CONV1D (depth first)	kh=1 kw=1*k	1	2*w	8*d	no sparsity: 32*n sparsity: 64*n	128*
CONV1D (kernel first) ifm_depth<=8, no sparsity	kh=1 k_rnd(kw,5,4)=1*k	1	2*w	8*d	8	128
CONV1D (kernel first) ifm_depth>8	no sparsity: kh=1 k_rnd(kw,5,2)=1*k sparsity: kh=1 k_rnd(kw,10,4)=1*k	1	2*w	8*d	16*n	128*
Fully connected	kh=1 kw=1	1	1	8*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	1*h	1*w	16*d	16*n	16
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	1*w	16*d	16*n	16

Table 4-2: 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	1*h	2*w	8*d	no sparsity: 32*n sparsity: 64*n	64*
CONV2D (kernel first)	k_rnd(kh*kw,10,4)=1*k	1*h	2*w	8*d	no sparsity: 8*n sparsity: 16*n	64*
CONV1D (depth first)	kh=1 kw=1*k	1	2*w	8*d	no sparsity: 32*n sparsity: 64*n	64*
CONV1D (kernel first)	kh=1 k_rnd(kw,10,4)=1*k	1	2*w	8*d	no sparsity: 8*n sparsity: 16*n	64*
Fully connected	kh=1 kw=1	1	1	16*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	1*h	2*w	8*d	8*n	8
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	2*w	8*d	8*n	8

Convolution performance of the Ethos™-U85$_{256}$

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.

The definition of the k_rnd function: k_rnd(k,s,r) = floor(k/s)*s + max(k%s,r)

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.)

Note

With sparsity enabled, all MACs per cycle numbers with * effectively doubles as half of the multiplies for the convolutions are skipped in the hardware. This skipping can be done because half of the weights are guaranteed to be 0.

Table 4-3: 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	no sparsity: 32*n sparsity: 64*n	256*
CONV2D (kernel first) ifm_depth<=8, no sparsity	k_rnd(kh*kw,5,4)=1*k	2*h	2*w	8*d	8	256
CONV2D (kernel first) ifm_depth>8	no sparsity: k_rnd(kh*kw,5,2)=1*k sparsity: k_rnd(kh*kw,10,4)=1*k	2*h	2*w	8*d	16*n	256*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	8*d	no sparsity: 32*n sparsity: 64*n	256*
CONV1D (kernel first) ifm_depth<=8, no sparsity	kh=1 k_rnd(kw,5,4)=1*k	1	4*w	8*d	8	256
CONV1D (kernel first) ifm_depth>8	no sparsity: kh=1 k_rnd(kw,5,2)=1*k sparsity: kh=1 k_rnd(kw,10,4)=1*k	1	4*w	8*d	16*n	256*
Fully connected	kh=1 kw=1	1	1	8*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	1*h	2*w	16*d	16*n	32
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	2*w	16*d	16*n	32

Table 4-4: 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	no sparsity: 32*n sparsity: 64*n	128*
CONV2D (kernel first)	k_rnd(kh*kw,10,4)=1*k	2*h	2*w	8*d	no sparsity: 8*n sparsity: 16*n	128*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	8*d	no sparsity: 32*n sparsity: 64*n	128*
CONV1D (kernel first)	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	8*d	no sparsity: 8*n sparsity: 16*n	128*

16-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
Fully connected	kh=1 kw=1	1	1	16*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	2*h	2*w	8*d	8*n	16
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	8*d	8*n	16

Convolution performance of the Ethos™-U85$_{512}$

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 this value up and the extra MACs computed as a consequence are lost.

The definition of the k_rnd function: k_rnd(k,s,r) = floor(k/s)*s + max(k%s,r)

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.)

Note

With sparsity enabled, all MACs per cycle numbers with * effectively doubles as half of the multiplies for the convolutions are skipped in the hardware. This skipping can be done because half of the weights are guaranteed to be 0.

Table 4-5: 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	16*d	no sparsity: 32*n sparsity: 64*n	512*
CONV2D (kernel first) ifm_depth<=8, no sparsity	k_rnd(kh*kw,5,4)=1*k	2*h	2*w	16*d	8	512
CONV2D (kernel first) ifm_depth>8	no sparsity: k_rnd(kh*kw,5,2)=1*k sparsity: k_rnd(kh*kw,10,4)=1*k	2*h	2*w	16*d	16*n	512*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	16*d	no sparsity: 32*n sparsity: 64*n	512*
CONV1D (kernel first) ifm_depth<=8, no sparsity	kh=1 k_rnd(kw,5,4)=1*k	1	4*w	16*d	8	512
CONV1D (kernel first) ifm_depth>8	no sparsity: kh=1 k_rnd(kw,5,2)=1*k sparsity: kh=1 k_rnd(kw,10,4)=1*k	1	4*w	16*d	16*n	512*
Fully connected	kh=1 kw=1	1	1	16*d	no sparsity: 32*n sparsity: 64*n	WB

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	2*h	2*w	16*d	16*n	64
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	16*d	16*n	64

Table 4-6: 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	16*d	no sparsity: 32*n sparsity: 64*n	256*
CONV2D (kernel first)	k_rnd(kh*kw,10,4)=1*k	2*h	2*w	16*d	no sparsity: 8*n sparsity: 16*n	256*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	16*d	no sparsity: 32*n sparsity: 64*n	256*
CONV1D (kernel first)	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	16*d	no sparsity: 8*n sparsity: 16*n	256*
Fully connected	kh=1 kw=1	1	1	16*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	2*h	2*w	8*d	8*n	16
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	8*d	8*n	16

Convolution performance of the Ethos™-U85$_{1024}$

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 this value up and the extra MACs computed as a consequence are lost.

The definition of the k_rnd function: k_rnd(k,s,r) = floor(k/s)*s + max(k%s,r)


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.)


Note

With sparsity enabled, all MACs per cycle numbers with * effectively doubles as half of the multiplies for the convolutions are skipped in the hardware. This skipping can be done because half of the weights are guaranteed to be 0.

Table 4-7: Convolution performance for 8-bit activations

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV2D deep (depth first)	kh*kw=1*k	2*h	2*w	32*d	no sparsity: 32*n sparsity: 64*n	1024*
CONV2D deep (kernel first) ifm_depth<=8, no sparsity	k_rnd(kh*kw,5,4)=1*k	2*h	2*w	32*d	8	1024
CONV2D deep (kernel first) ifm_depth>8	no sparsity: k_rnd(kh*kw,5,2)=1*k sparsity: k_rnd(kh*kw,10,4)=1*k	2*h	2*w	32*d	16*n	1024*
CONV2D shallow (depth first)	kh*kw=1*k	2*h	4*w	16*d	no sparsity: 32*n sparsity: 64*n	1024*
CONV2D shallow (kernel first) ifm_depth<=8, no sparsity	k_rnd(kh*kw,5,4)=1*k	2*h	4*w	16*d	8	1024
CONV2D shallow (kernel first) ifm_depth>8	no sparsity: k_rnd(kh*kw,5,2)=1*k sparsity: k_rnd(kh*kw,10,4)=1*k	2*h	4*w	16*d	16*n	1024*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	32*d	no sparsity: 32*n sparsity: 64*n	1024*
CONV1D (kernel first) ifm_depth<=8, no sparsity	kh=1 k_rnd(kw,5,4)=1*k	1	4*w	32*d	8	1024
CONV1D (kernel first) ifm_depth>8	no sparsity: kh=1 k_rnd(kw,5,2)=1*k sparsity: kh=1 k_rnd(kw,10,4)=1*k	1	4*w	32*d	16*n	1024*
Fully connected	kh=1 kw=1	1	1	32*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	2*h	4*w	16*d	16*n	128
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	16*d	16*n	64

Table 4-8: Convolution performance for 16-bit activations

16-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV2D deep (depth first)	kh*kw=1*k	2*h	2*w	32*d	no sparsity: 32*n sparsity: 64*n	512*
CONV2D deep (kernel first)	k_rnd(kh*kw,10,4)=1*k	2*h	2*w	32*d	no sparsity: 8*n sparsity: 16*n	512*
CONV2D shallow (depth first)	kh*kw=1*k	2*h	4*w	16*d	no sparsity: 32*n sparsity: 64*n	512*
CONV2D shallow (kernel first)	k_rnd(kh*kw,10,4)=1*k	2*h	4*w	16*d	no sparsity: 8*n sparsity: 16*n	512*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	16*d	no sparsity: 32*n sparsity: 64*n	512*
CONV1D (kernel first)	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	16*d	no sparsity: 8*n sparsity: 16*n	512*
Fully connected	kh=1 kw=1	1	1	16*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	2*h	4*w	8*d	8*n	64

16-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	8*d	8*n	32

Convolution performance of the Ethos™-U85$_{2048}$

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 this value up and the extra MACs computed as a consequence are lost.

The definition of the k_rnd function: k_rnd(k,s,r) = floor(k/s)*s + max(k%s,r)

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.)

Note

With sparsity enabled, all MACs per cycle numbers with * effectively doubles as half of the multiplies for the convolutions are skipped in the hardware. This skipping can be done because half of the weights are guaranteed to be 0.

Table 4-9: Convolution performance for 8-bit activations

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV2D deep (depth first)	kh*kw=1*k	2*h	2*w	64*d	no sparsity: 32*n sparsity: 64*n	2048*
CONV2D deep (kernel first) ifm_depth<=8, no sparsity	k_rnd(kh*kw,5,4)=1*k	2*h	2*w	64*d	8	2048
CONV2D deep (kernel first) ifm_depth>8	no sparsity: k_rnd(kh*kw,5,2)=1*k sparsity: k_rnd(kh*kw,10,4)=1*k	2*h	2*w	64*d	16*n	2048*
CONV2D shallow (depth first)	kh*kw=1*k	4*h	4*w	16*d	no sparsity: 32*n sparsity: 64*n	2048*
CONV2D shallow (kernel first) ifm_depth<=8, no sparsity	k_rnd(kh*kw,5,4)=1*k	4*h	4*w	16*d	8	2048
CONV2D shallow (kernel first) ifm_depth>8	no sparsity: k_rnd(kh*kw,5,2)=1*k sparsity: k_rnd(kh*kw,10,4)=1*k	4*h	4*w	16*d	16*n	2048*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	64*d	no sparsity: 32*n sparsity: 64*n	2048*
CONV1D (kernel first) ifm_depth<=8, no sparsity	kh=1 k_rnd(kw,5,4)=1*k	1	4*w	64*d	8	2048

8-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV1D (kernel first) ifm_depth>8	no sparsity: kh=1 k_rnd(kw,5,2)=1*k sparsity: kh=1 k_rnd(kw,10,4)=1*k	1	4*w	64*d	16*n	2048*
Fully connected	kh=1 kw=1	1	1	64*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	4*h	4*w	16*d	16*n	256
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	16*d	16*n	64

Table 4-10: Convolution performance for 16-bit activations

16-bit activation						
Operation	**Kernel size**	**OFM height**	**OFM width**	**OFM depth**	**IFM depth**	**MACs per cycle**
CONV2D deep (depth first)	kh*kw=1*k	2*h	2*w	32*d	no sparsity: 32*n sparsity: 64*n	1024*
CONV2D deep (kernel first)	k_rnd(kh*kw,10,4)=1*k	2*h	2*w	32*d	no sparsity: 8*n sparsity: 16*n	1024*
CONV2D shallow (depth first)	kh*kw=1*k	2*h	4*w	16*d	no sparsity: 32*n sparsity: 64*n	1024*
CONV2D shallow (kernel first)	k_rnd(kh*kw,10,4)=1*k	2*h	4*w	16*d	no sparsity: 8*n sparsity: 16*n	1024*
CONV1D (depth first)	kh=1 kw=1*k	1	4*w	16*d	no sparsity: 32*n sparsity: 64*n	1024*
CONV1D (kernel first)	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	16*d	no sparsity: 8*n sparsity: 16*n	1024*
Fully connected	kh=1 kw=1	1	1	16*d	no sparsity: 32*n sparsity: 64*n	WB
DepthwiseConv2D	k_rnd(kh*kw,10,4)=1*k	4*h	4*w	8*d	8*n	128
DepthwiseConv1D	kh=1 k_rnd(kw,10,4)=1*k	1	4*w	8*d	8*n	32

Matrix multiply performance

Matrix multiply is implemented as a convolution with a kernel size of 1x1 and a tensor height of 1. The width and channel dimensions form the 2D matrix rows and columns. In this form, a 1x1 kernel convolution performs the multiply of a matrix A by the transpose of a second matrix B.

4.2 AO operations performance

The performance for *Multiply-Accumulate* (MAC) operations and Performance for elementwise operations tables list the throughput per *Scaling Unit* (SU) for different combinations of input and

output types. Multiply this number with NUM_SU to get the number of processed elements per clock cycle.

The table shows the performance for all modes except for OP_RESIZE_BILINEAR, OP_RESIZE_NEAREST and OP_DIV which are described separately:

- The Input row list inputs from *Accumulator Buffer* (AB) or *Chaining Buffer* (CB) and the associated bit width.
- The Output row list outputs to *Output Buffer* (OB) or CB and the associated bit width.

If the performance is lower than 1 then a reason is listed within parenthesis.

The following reasons exist:

- *Chaining Buffer* (CB): The CB performance is halved in case of 32 bit CB input and/or 32 bit CB output. There is not enough CB RAM bandwidth to read two 32 bit inputs per cycle. And for 32 bit CB writes the throughput is halved as a compromise to reduce CB RAM slot size.
- Activation output data processing pipeline (P): The data processing pipeline is in general 32 bits wide. This means that wider words will be processed at half throughput.
- *Direct Memory Access* (DMA): In some cases the available DMA bandwidth on the AO_DMA_OFM_IF is a limiting factor.

Table 4-11: Performance of AO for operations with inputs from MAC

	NUM_SU	MAC ops AB->OB	MAC ops AB->OB	MAC Ops AB->OB	MAC Ops AB->OB	MAC ops AB->CB	MAC ops AB->CB
Input		AB_32 AB_48	AB_32 AB_48	AB_32 AB_48	AB_32 AB_48	AB_32 AB_48	AB_32 AB_48
Output		OB_8	OB_16	OB_32	OB_64	CB_8 CB_16	CB_32
128	2	1	1	1	1/2 (P)	1	1/2 (CB)
256	4	1	1	1	1/2 (P, DMA)	1	1/2 (CB)
512	8	1	1	1/2 (DMA)	1/4 (DMA)	1	1/2 (CB)
1024/2048	16/32	1	1/2 (DMA)	1/4 (DMA)	1/8 (DMA)	1	1/2 (CB)

Table 4-12: Performance for elementwise operations

	Input	Output	128	256	512	1024 2048
NUM_SU	-	-	2	4	8	16/ 32

	Input	Output	128	256	512	1024 2048
Elemwise CB->CB	CB_8 CB_16	CB_8 CB_16	1	1	1	1
Elemwise CB->CB	CB_8 CB_16 CB_32	CB_32	1/2 (CB)	1/2 (CB)	1/2 (CB)	1/2 (CB)
Elemwise CB->CB	CB_32	CB_8 CB_16 CB_32	1/2 (CB)	1/2 (CB)	1/2 (CB)	1/2 (CB)
Elemwise CB->OB	CB_8 CB_16	OB_8	1	1	1	1
Elemwise CB->OB	CB_8 CB_16	OB_16	1	1	1	1/2 (DMA)
Elemwise CB->OB	CB_8 CB_16	OB_32	1	1	1/2 (DMA)	1/4 (DMA)
Elemwise CB->OB	CB_8 CB_16	OB_64	1/2 (P)	1/2 (P, DMA)	1/4 (DMA)	1/8 (DMA)
Elemwise CB->OB Unary	CB_32	OB_8 OB_16	1	1	1	1/2 (DMA)
Elemwise CB->OB Binary	CB_32	OB_8 OB_16	1/2 (CB)	1/2 (CB)	1/2 (CB)	1/2 (CB, DMA)
Elemwise CB->OB Unary	CB_32	OB_32	1	1	1/2 (DMA)	1/4 (DMA)
Elemwise CB->OB Binary	CB_32	OB_32	1/2 (CB)	1/2 (CB)	1/2 (CB, DMA)	1/4 (DMA)

	Input	Output	128	256	512	1024 2048
Elemwise CB->OB Unary	CB_32	OB_64	1/2 (P)	1/2 (P, DMA)	1/4 (DMA)	1/8 (DMA)
Elemwise CB->OB Binary	CB_32	OB_64	1/2 (P, CB)	1/2 (P, CB, DMA)	1/4 (DMA)	1/8 (DMA)

The OP_DIV operator can take 68 cycles per output sample.

There are general rules that affects all processing:

- A slow block affects all following blocks as long as it is in the processing chain.
- Partial SBs might be used at block edges.
- A bubble might be inserted when reading from the AB due to a grant delay of maximum one cycle.

AO_DMA_OFM_IF table shows the limitations which are taken into account in Performance for elementwise operations table and RESIZE performance table:

- The column "OFM_DATA_W" shows the bandwidth per cycle in number of bits.
- The column "OFM 64 BW" shows the required bandwidth per cycle in number of bits in case of 64 bit OFM.
- The column "OFM 64 ratio" shows available bandwidth divided by required bandwidth in case of 64 bit OFM.
- The column "OFM 32 ratio" shows available bandwidth divided by required bandwidth in case of 32 bit OFM.
- The column "OFM 16 ratio" shows available bandwidth divided by required bandwidth in case of 16 bit OFM.

Table 4-13: AO_DMA_OFM_IF

Configuration	NUM_SU	OFM_DATA_W	OFM 64 BW	OFM 64 ratio	OFM 32 ratio	OFM 16 ratio
Ethosu85-128	2	128	128	1	2	4
Ethosu85-256	4	128	256	0.5	1	2
Ethosu85-512	8	128	512	0.25	0.5	1
Ethosu85-1024	16	128	1024	0.125	0.25	0.5
Ethosu85-2048	32	256	2048	0.125	0.25	0.5

4.2.1 Transpose

The performance in case of transpose for Ethos™-U85-2048 where width and height are transposed is maximum 1/2 since half of the SUs are disabled. These modes are WHC, CWH and WCH.

4.2.2 Reverse

The performance in case of REVERSE_W for Ethos™-U85-2048 is maximum 1/2 since half of the SUs are disabled.

4.2.3 Resize

RESIZE performance table shows a summary of the RESIZE performance. Note that the performance in Performance for elementwise operations table must be taken into account for RESIZE. The lowest value of Performance for elementwise operations table and RESIZE performance table gives the performance.

Table 4-14: RESIZE performance

Configuration	Input bit width	Performance per SU
Ethos™-U85-128/256/512	8	One OFM element per clock cycle
Ethos™-U85-1024	8	Upscale factor >= 2: 1 OFM element per clock cycle. Upscale factor < 2: 1/2 OFM element per clock cycle.
Ethos™-U85-2048	8	Upscale factor >= 4: 1 OFM element per clock cycle. Upscale factor < 4: 1/2 OFM element per clock cycle.
All	16	One OFM element every other clock cycle

5. Signal descriptions

This chapter describes the signals for the processor.

5.1 Clock and reset signals

The NPU has one clock signal and two reset signals.

Signal definitions

Table 5-1: Clock and reset signals

Signal	Direction	Description	Connection information	Clock domain
CLK	Input	Clock input	To clock source	-
nRESET	Input	Reset. This signal is active-LOW.	To reset controller	Asynchronous
nMBISTRESET	Input	Reset for MBIST logic. This signal is active-LOW.	To reset controller	Asynchronous

5.2 Configuration signals

The system configuration signals determine the security level of the NPU after boot.

Signal definitions

Table 5-2: Configuration signals

Signal	Direction	Description	Connection information	Clock domain
PORPL	Input	The *Power On Reset Privilege Level* (PORPL) This signal sets the privilege level of the NPU after hard reset. 0 = User, 1 = Privileged	To configuration controller, or tied to constant.	CLK
PORSL	Input	The *Power On Reset Security Level* (PORSL) This signal sets the security level of the NPU after hard reset. 0 = Secure, 1 = Non-secure	To configuration controller, or tied to constant.	CLK
DBGEN	Input	This signal is the debug enable. 0 = Disabled, 1 = Enabled	To configuration controller, or tied to constant.	CLK
SPIDEN	Input	This signal is the debug enable for Secure state 0 = Disabled, 1 = Enabled	To configuration controller, or tied to constant.	CLK
CFGSRAMCAP[31:0]	Input	The configuration of capabilities for SRAM ports. This signal is sampled with soft and hard reset.	To configuration controller, or tied to constant.	CLK

Signal	Direction	Description	Connection information	Clock domain
CFGEXTCAP[31:0]	Input	The configuration of capabilities for EXT ports. Sampled with soft and hard reset.	To configuration controller, or tied to constant.	CLK
CFGSRAMHASH0[39:0]	Input	The configuration of hash function for selecting among SRAM ports This signal is Sampled with soft and hard reset.	To configuration controller, or tied to constant.	CLK
CFGSRAMHASH1[39:0]	Input	The configuration of hash function for selecting among SRAM ports. This signal is sampled with soft and hard reset. This signal is only present when the SRAM2 and SRAM3 ports are present.	To configuration controller, or tied to constant.	CLK
CFGEXTHASH0[39:0]	Input	The configuration of hash function for selecting among EXT ports. This signal is sampled with soft and hard reset. This signal is only present when the EXT1 port is present.	To configuration controller, or tied to constant.	CLK

5.3 Interrupt and event signals

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

Signal definitions

Table 5-3: Interrupt and event signals

Signal	Direction	Description	Connection information	Clock domain
IRQ	Output	This signal is the interrupt request. This signal is level triggered.	To interrupt controller	CLK

5.4 Power management signals

The NPU has several signals for power management that must be connected to a system level power controller.

Signal definitions

Table 5-4: Power management signals

Signal	Direction	Description	Connection information	Clock domain
PWRQACTIVE	Output	This signal indicates that the NPU requires power.	To power controller	Asynchronous

Signal	Direction	Description	Connection information	Clock domain
PWRQREQn	Input	This signal indicates the power controller wants to power down. This signal is active-LOW.	To power controller	Asynchronous
PWRQACCEPTn	Output	This signal indicates the NPU accepts the power controller request. This signal is active-LOW.	To power controller	Asynchronous
PWRQDENY	Output	This signal indicates the NPU denies power controller request.	To power controller	Asynchronous

5.5 Clock management signals

The NPU has several signals for clock management that must be connected to a system level clock controller.

Signal definitions

Table 5-5: Clock management signals

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

5.6 Warm reset halt signals

The NPU has a few signals for warm reset halt that must be connected to a system control unit.

Signal definitions

Table 5-6: Warm reset halt signals

Signal	Direction	Description	Connection information	Clock domain
WRSTQREQ	Input	This signal indicates a request to halt the NPU. This signal is active-HIGH.	To system control unit	Asynchronous
WRSTQACK	Output	This signal acknowledges halting the NPU. This signal is active-HIGH.	To system control unit	Asynchronous

5.7 Cross trigger interface signals

The NPU has a few *Cross Trigger Interface* (CTI) signals that must be connected to a debug unit.

Signal definitions

Table 5-7: *Cross Trigger Interface* (CTI) signals

Signal	Direction	Description	Connection information	Clock domain
DBGHALTREQ	Input	This signal indicates the debug halt request. This signal is level-triggered and active-HIGH.	To debug unit	Asynchronous
DBGRESTARTREQ	Input	This signal indicates the debug restart request. This signal is pulse triggered and active-HIGH.	To debug unit	CLK
DBGHALTACK	Output	This signal is the pulsed halt acknowledge. This signal is active-HIGH.	To debug unit	Asynchronous

Also, there are the additional related signals, DBGEN and SPIDEN, listed in the Configuration signals topic.

5.8 AMBA® 5 AXI manager signals

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

Depending on the configuration, there can be up to six AXI manager ports named SRAM0, SRAM1, SRAM2, SRAM3, EXT0, and EXT1.

The notation <X> must be replaced with SRAM0, SRAM1, SRAM2, SRAM3, EXT0, or EXT1 in the following tables in this section.

5.8.1 Wake-up and clock enable signals

The NPU has a few wake-up and clock enable signals that must be connected to the AXI interconnect.

Signal definitions

Table 5-8: Wake-up and clock enable signals

Signal	Direction	Description	Connection information	Clock domain
AWAKEUP<X>	Output	This signal is the pending activity indicator.	To AXI interconnect	ACLK<X>
ACLKEN<X>	Input	This signal is the clock enable. The inputs are sampled when high and outputs held stable when low.	To AXI interconnect	CLK

5.8.2 Write address channel signals

The NPU has several write address channel signals that must be connected to the AXI interconnect.

Signal definitions

Table 5-9: Write address channel signals

Signal	Direction	Description	Connection information	Clock domain
AWVALID<X>	Output	This signal indicates the write address is valid.	To AXI interconnect	ACLK<X>
AWID<X>[5:0]	Output	This signal is the write address ID.	To AXI interconnect	ACLK<X>
AWADDR<X>[39:0]	Output	This signal is the write address.	To AXI interconnect	ACLK<X>
AWLEN<X>[7:0]	Output	This signal is the write burst length.	To AXI interconnect	ACLK<X>
AWSIZE<X>[2:0]	Output	This signal is the write burst size.	To AXI interconnect	ACLK<X>
AWBURST<X>[1:0]	Output	This signal is the write burst type.	To AXI interconnect	ACLK<X>
AWCACHE<X>[3:0]	Output	This signal is the write outer cache type.	To AXI interconnect	ACLK<X>
AWDOMAIN<X>[1:0]	Output	This signal is the write Shareability domain.	To AXI interconnect	ACLK<X>
AWPROT<X>[2:0]	Output	This signal is the write protection type.	To AXI interconnect	ACLK<X>
AWREADY<X>	Input	This signal indicates the write address is ready.	To AXI interconnect	ACLK<X>

5.8.3 Write data channel signals

The NPU has several write data channel signals that must be connected to the AXI interconnect.

Signal definitions

Table 5-10: Write data channel signals

Signal	Direction	Description	Connection information	Clock domain
WVALID<X>	Output	This signal indicates the write data is valid.	To AXI interconnect	ACLK<X>
WDATA<X>[127:0]	Output	This signal indicates the write data.	To AXI interconnect	ACLK<X>
WSTRB<X>[7:0]	Output	This signal is the write byte lane strobes.	To AXI interconnect	ACLK<X>
WLAST<X>	Output	This signal is the write data last transfer indication.	To AXI interconnect	ACLK<X>>
WREADY<X>	Input	This signal indicates the write data is ready.	To AXI interconnect	ACLK<X>

5.8.4 Write response channel signals

The NPU has a several write response channel signals that must be connected to the AXI interconnect.

Signal definitions

Table 5-11: Write response channel signals

Signal	Direction	Description	Connection information	Clock domain
BVALID<X>	Input	This signal indicates the write response is valid.	To AXI interconnect	ACLK<X>

Signal	Direction	Description	Connection information	Clock domain
BID<X>[5:0]	Input	This signal is the write response ID.	To AXI interconnect	ACLK<X>
BRESP<X>[1:0]	Input	This signal is the write response.	To AXI interconnect	ACLK<X>
BREADY<X>	Output	This signal indicates the write response is ready.	To AXI interconnect	ACLK<X>

5.8.5 Read address channel signals

The NPU has several read address channel signals that must be connected to the AXI interconnect.

Signal definitions

Table 5-12: Read address channel signals

Signal	Direction	Description	Connection information	Clock domain
ARVALID<X>	Output	This signal indicates the read address is valid.	To AXI interconnect	ACLK<X>
ARID<X>[5:0]	Output	Read address ID. ARIDSRAM0-ARIDSRAM3 have the same signal width as ARIDEXT0. As a result, the MSBs are zero.	To AXI interconnect	ACLK<X>
ARADDR<X>[39:0]	Output	This signal indicates the read address.	To AXI interconnect	ACLK<X>
ARLEN<X>[7:0]	Output	This signal indicates the read burst length.	To AXI interconnect	ACLK<X>
ARSIZE<X>[2:0]	Output	This signal indicates the read burst size.	To AXI interconnect	ACLK<X>
ARBURST<X>[1:0]	Output	This signal indicates the read burst type.	To AXI interconnect	ACLK<X>
ARCACHE<X>[3:0]	Output	This signal indicates the read cache type.	To AXI interconnect	ACLK<X>
ARDOMAIN<X>[1:0]	Output	This signal indicates the read Shareability domain.	To AXI interconnect	ACLK<X>
ARPROT<X>[2:0]	Output	This signal indicates the read protection type	To AXI interconnect	ACLK<X>
ARREADY<X>	Input	This signal indicates the read address is ready.	To AXI interconnect	ACLK<X>

5.8.6 Read data channel signals

The NPU has several read address channel signals that must be connected to the AXI interconnect.

Signal definitions

Table 5-13: Read data channel signals

Signal	Direction	Description	Connection information	Clock domain
RVALID<X>	Input	This signal indicates the read data is valid.	To AXI interconnect	ACLK<X>
RID<X>[5:0]	Input	This signal is the read data ID.	To AXI interconnect	ACLK<X>
RDATA<X>[127:0]	Input	This signal indicates the read data.	To AXI interconnect	ACLK<X>

Signal	Direction	Description	Connection information	Clock domain
RRESP<X>[1:0]	Input	This signal indicates the read data response.	To AXI interconnect	ACLK<X>
RLAST<X>	Input	This signal indicates the read data last transfer indication.	To AXI interconnect	ACLK<X>
RREADY<X>	Output	This signal indicates the read data is ready.	To AXI interconnect	ACLK<X>

5.9 AMBA 5 APB Completer interface signals

The APB port implements an AMBA® 5 APB Completer interface, with the addition of a PCLKEN signal.


Note

The NPU samples PWAKEUP and PCLKEN on any CLK edge. For this reason, there must not be a multi-cycle path applied to these signals. These signals are different from other signals in the AMBA® 5 APB Completer interface.

Signal definitions

Table 5-14: AMBA® 5 APB Completer interface signals

Signal	Direction	Description	Connection information	Clock domain
PWAKEUP	Input	This signal is wake-up. The signal is input to an OR-gate driving CLKQACTIVE.	To APB interconnect	CLK
PCLKEN	Input	This signal is the clock enable. The inputs of the signal are sampled when high and outputs held stable when low.	To APB interconnect	CLK
PSEL	Input	This signal is a transfer request.	To APB interconnect	PCLK
PENABLE	Input	This signal indicates the second and subsequent cycles of an APB transfer.	To APB interconnect	PCLK
PPROT[2:0]	Input	This signal is the transfer privilege and security level. PPROT[2] is an indicator for data or instruction. The NPU does not use PPROT[2].	To APB interconnect	PCLK
PWRITE	Input	This signal indicates the write transfer.	To APB interconnect	PCLK
PADDR[12:0]	Input	This signal indicates the transfer address.	To APB interconnect	PCLK
PWDATA[31:0]	Input	This signal indicates the write data.	To APB interconnect	PCLK
PSTRB[3:0]	Input	This signal indicates the write data byte strobes.	To APB interconnect	PCLK
PREADY	Output	This signal indicates the subordinate is ready.	To APB interconnect	PCLK
PSLVERR	Output	This signal is the subordinate error response.	To APB interconnect	PCLK
PRDATA[31:0]	Output	This signal is the subordinate read data.	To APB interconnect	PCLK

5.10 DFT and MBIST signals

The NPU has a few DFT and MBIST signals that must be connected to the DFT controller.

Signal definitions

Table 5-15: DFT and MBIST signals

Signal	Direction	Description	Connection information	Clock domain
DFTCGEN	Input	This signal forces on the clock gates during scan shift.	To DFT controller	CLK
DFTRSTDISABLE[1:0]	Input	This signal disables the internal synchronized reset during scan shift.	To DFT controller	CLK
DFTRAMHOLD	Input	This signal disables the RAM chip select during scan shift.	To DFT controller	CLK
MBISTREQ	Input	This signal is the MBIST test request.	To DFT controller	CLK

6. 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 microcontroller. 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.

7. Boot flow information

This chapter 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 3.3.1.3 CMD register on page 42.

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 the following.

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 3.3.1.4 RESET register on page 44.
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, other reads are meaningless because they will return a 0. APB writes during a

soft reset are also not advised because they cause a bus error. CMD and RESET writes are ignored during a soft reset.

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 3.3.1.3 CMD register on page 42.

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.

Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

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

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

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

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

export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

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

Product status

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

Product completeness status

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

Product revision status

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

r *(value)* Identifies the major revision of the product, for example, r1.
p *(value)* Identifies the minor revision or modification status of the product, for example, p2.

Revision history

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

Document release information

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

Document history

Issue	Date	Confidentiality	Change
0000-05	31 January 2025	Non-Confidential	First release for r0p0
0000-04	14 June 2024	Confidential	Second early access release for r0p0
0000-03	5 June 2024	Confidential	First early access release for r0p0
0000-02	5 April 2024	Confidential	First Beta release for r0p0
0000-01	12 January 2024	Confidential	First DEV release for r0p0

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in Document release information on page 357.

Table 2: Differences between issue 0000-04 and 0000-05

Change	Location
First Non-Confidential release for r0p0	-
Improved the description of the NPU, its supported networks, DMA controller, and supported software.	1. Description of the Arm Ethos-U85 NPU on page 10
Improved the note about trusted software running on the host.	2.2 Security and boot flow on page 18
Moved configuration signals in the Functional blocks diagram to properly reflect where the signals go	2.3 Functional blocks on page 19
Added a note on getting information for how an SoC uses striping	2.3.3 Configuration pins and AXI port striping on page 22
Clarified the description of the IFM channel, weight channel, and fast weight channel.	2.3.4 DMA controller on page 25
Added information on when to use the Standard Weight Decoder (SWD) and the Fast Weight Decoder (FWD)	2.4 Weight decoder on page 28
Clarified the pooling modes the MAC unit supports and	2.5 MAC unit on page 28
Update to the description of the RESET register	3.3.1.4 RESET register on page 44
Update to the mac_per_cc (MACS) bit description in the BASE.CONFIG bit descriptions table	3.3.1.10 CONFIG register on page 52
Updated the NPU_OP_CONV, NPU_OP_DEPTHWISE, NPU_OP_DMA_START, NPU_OP_DMA_WAIT, NPU_OP_ELEMENTWISE, NPU_OP_PMU_MASK, and NPU_OP_RESIZE pseudocode and the NPU_OP_POOL pseudocode and description. Also updated the description of microblock (MB) in the Instruction NPU_SET_ACC_FORMAT encoding layout table.	3.10 Instruction index for cmd0 stream on page 214
Added a link to the full list of supported operators	4. Operators and performance on page 332
Adds a description of how matrix multiply is implemented	4.1 Convolution performance on page 332

Table 3: Differences between issue 0000-03 and 0000-04

Change	Location
Second Confidential early access release for r0p0	-

Table 4: Differences between issue 0000-02 and 0000-03

Change	Location
First Confidential early access release for r0p0	-

Table 5: Differences between issue 0000-01 and 0000-02

Change	Location
First Confidential Beta release for r0p0	-

Table 6: Issue 0000-01

Change	Location
First Confidential DEV release for r0p0	-

Conventions

The following subsections describe conventions used in Arm documents.

Glossary

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

See the Arm Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

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

We recommend the following. If you do not follow these recommendations your system might not work.

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or of harming yourself.

Note
This information is important and needs your attention.

Tip
This information might help you perform a task in an easier, better, or faster way.

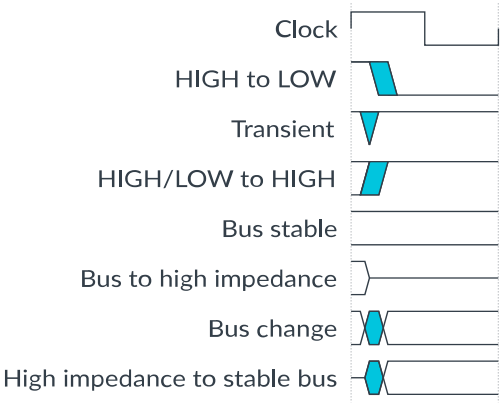
Remember
This information reminds you of something important relating to the current content.

Timing diagrams

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

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

Figure 1: Key to timing diagram conventions



Signals

The signal conventions are:

Signal level

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

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

Lowercase n

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

Useful resources

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

Access to Arm documents depends on their confidentiality:

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

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

Arm architecture and specifications	Document ID	Confidentiality
AMBA® AXI Protocol Specification	IHI 0022	Non-Confidential
AMBA® Low Power Interface Specification	IHI 0068	Non-Confidential
Arm® Corstone™ Reference Systems Architecture Specification Ma2	107610	Non-Confidential
TOSA specification	N/A	Non-Confidential