nVidia Hardware Documentation

Release git

Marcin Kościelnicki

March 02, 2016

Contents

Contents:

CHAPTER 1

Notational conventions

Contents

1.1 Introduction

Semantics of many operations are described in pseudocode. Here are some often used primitives.

1.2 Bit operations

In many places, the GPUs allow specifying arbitrary X-input boolean or bitwise operations, where X is 2, 3, or 4. They are described by a `2**X`-bit mask selecting the bit combinations for which the output should be true. For example, 2-input operation 0x4 (0b0100) is `~v1 & v2`: only bit 2 (`0b10`) is set, so the only input combination (0, 1) results in a true output. Likewise, 3-input operation 0xaa (0b10101010) is simply a passthrough of first input: the bits set in the mask are 1, 3, 5, 7 (`0b001, 0b011, 0b101, 0b111`), which corresponds exactly to the input combinations which have the first input equal to 1.

The exact semantics of such operations are:

```
# single-bit version
def bitop_single(op, *inputs):
    # first, construct mask bit index from the inputs
    bitidx = 0
    for idx, input in enumerate(inputs):
        if input:
            bitidx |= 1 << idx
    # second, the result is the given bit of the mask
    return op >> bitidx & 1

def bitop(op, *inputs):
    max_len = max(input.bit_length() for input in inputs)
    res = 0
```

```
    # perform bitop_single operation on each bit (+ 1 for sign bit)
    for x in range(max_len + 1):
        res |= bitop_single(op, *(input >> x & 1 for input in inputs)) << x
    # all bits starting from max_len will be identical - just what sext does
    return sext(res, max_len)
```

As further example, the 2-input operations on `a`, `b` are:

- `0x0`: always 0
- `0x1`: `~a & ~b`
- `0x2`: `a & ~b`
- `0x3`: `~b`
- `0x4`: `~a & b`
- `0x5`: `~a`
- `0x6`: `a ^ b`
- `0x7`: `~a | ~b`
- `0x8`: `a & b`
- `0x9`: `~a ^ b`
- `0xa`: `a`
- `0xb`: `a | ~b`
- `0xc`: `b`
- `0xd`: `~a | b`
- `0xe`: `a | b`
- `0xf`: always 1

For further enlightenment, you can search for GDI raster operations, which correspond to 3-input bit operations.

1.3 Sign extension

An often used primitive is sign extension from a given bit. This operation is known as `sext` after xtensa instruction of the same name and is formally defined as follows:

```
def sext(val, bit):
    # mask with all bits up from #bit set
    mask = -1 << bit
    if val & 1 << bit:
        # sign bit set, negative, set all upper bits
        return val | mask
    else:
        # sign bit not set, positive, clear all upper bits
        return val & ~mask
```

1.4 Bitfield extraction

Another often used primitive is bitfield extraction. Extracting an unsigned bitfield of length `l` starting at position `s` in `val` is denoted by `extr(val, s, l)`, and signed one by `extrs(val, s, l)`:

```
def extr(val, s, l):
    return val >> s & ((1 << l) - 1)

def extrs(val, s, l):
    return sext(extrs(val, s, l), l - 1)
```

CHAPTER 2

nVidia hardware documentation

Contents:

2.1 nVidia GPU introduction

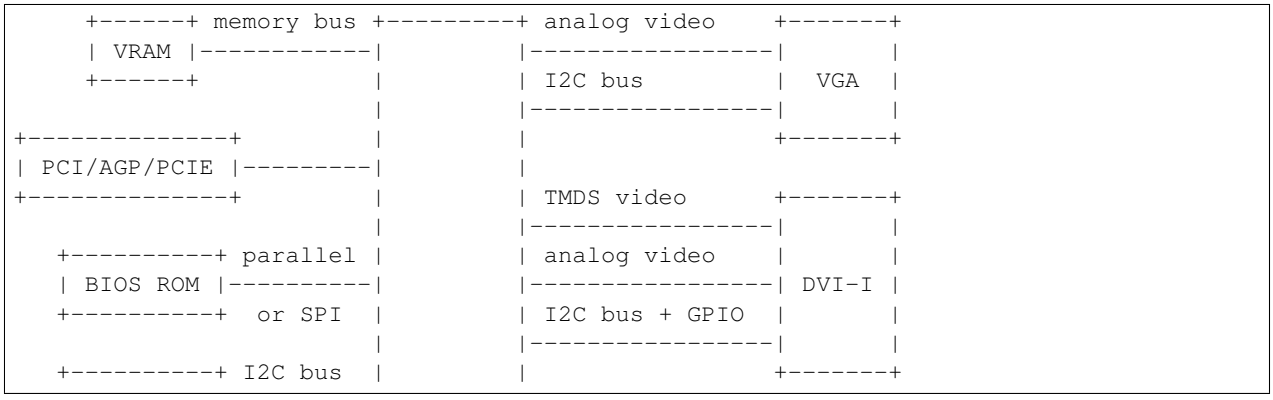
Contents

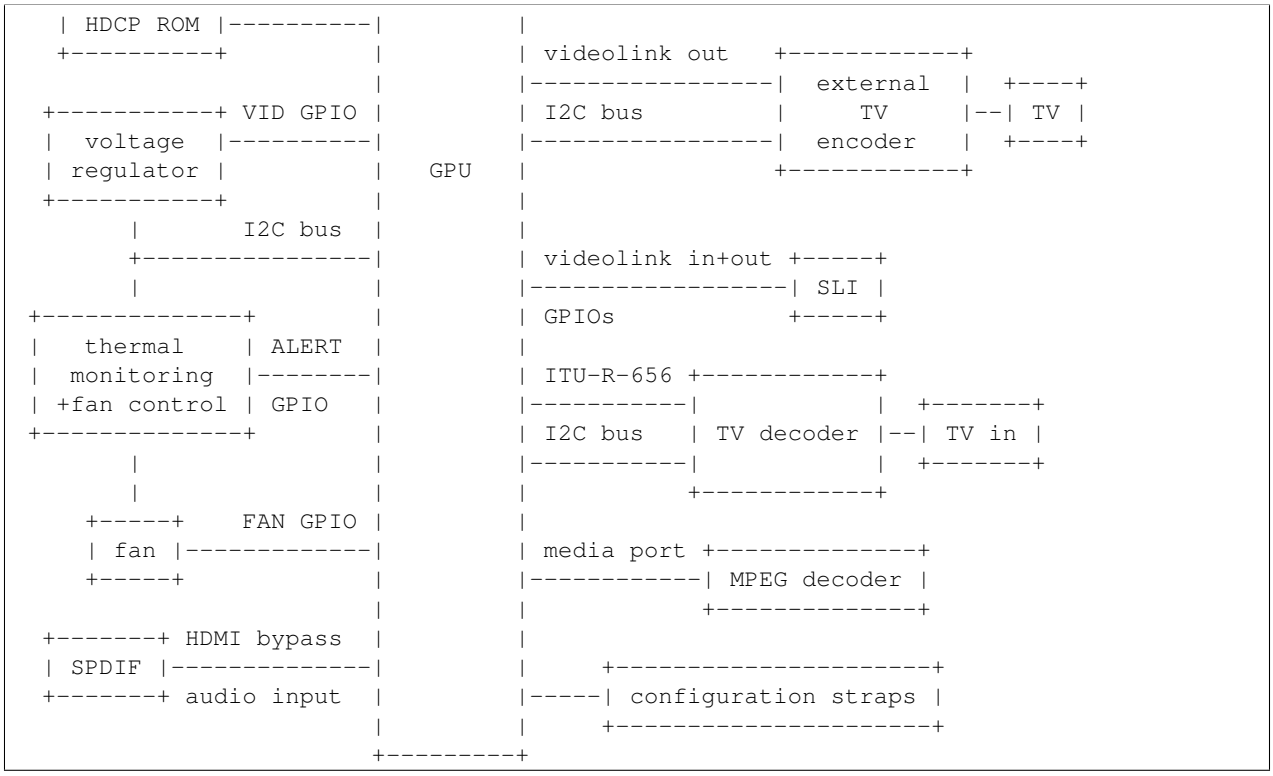
2.1.1 Introduction

This file is a short introduction to nvidia GPUs and graphics cards. Note that the schematics shown here are simplified and do not take all details into account - consult specific unit documentation when needed.

2.1.2 Card schematic

An nvidia-based graphics card is made of a main GPU chip and many supporting chips. Note that the following schematic attempts to show as many chips as possible - not all of them are included on all cards.





Note: while this schematic shows a TV output using an external encoder chip, newer cards have an internal TV encoder and can connect the output directly to the GPU. Also, external encoders are not limitted to TV outputs - they're also used for TMDS, DisplayPort and LVDS outputs on some cards.

Note: in many cases, I2C buses can be shared between various devices even when not shown by the above schema.

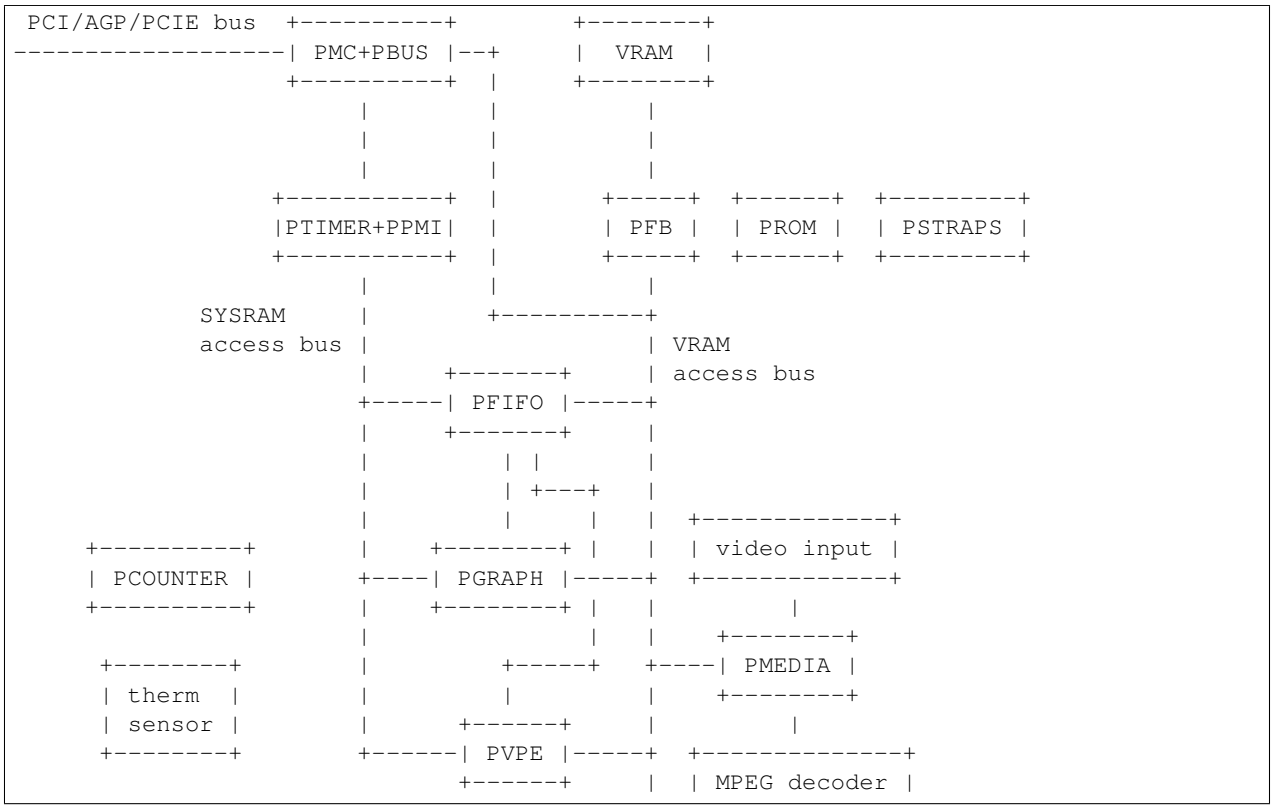
In summary, a card contains:

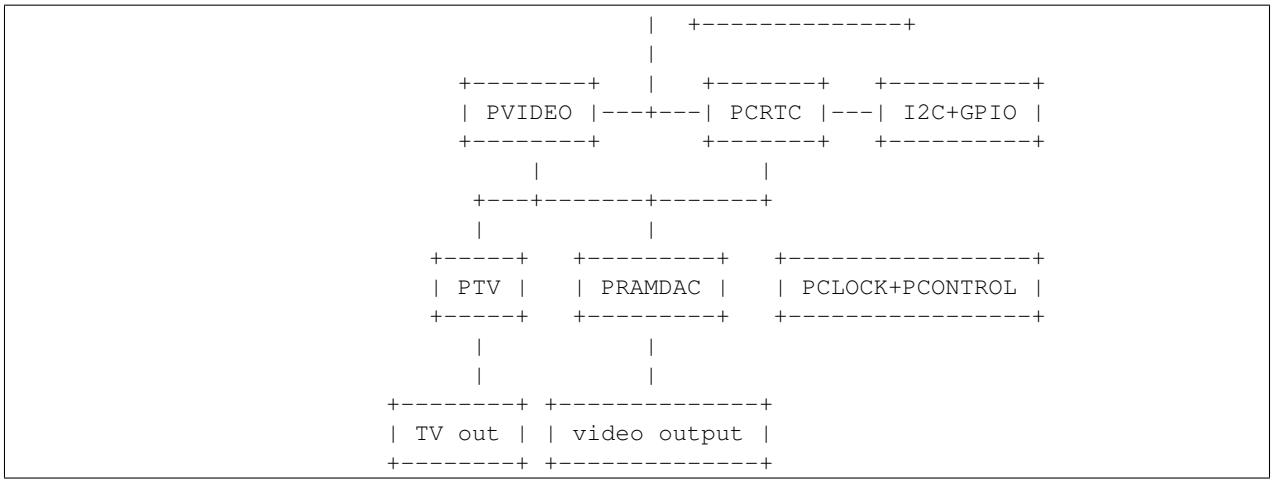
- a GPU chip [see *GPU chips* for a list]
- a PCI, AGP, or PCI-Express host interface
- on-board GPU memory [aka VRAM] - depending on GPU, various memory types can be supported: VRAM, EDO, SGRAM, SDR, DDR, DDR2, GDDR3, DDR3, GDDR5.
- a parallel or SPI-connected flash ROM containing the video BIOS. The BIOS image, in addition to standard VGA BIOS code, contains information about the devices and connectors present on the card and scripts to boot up and manage devices on the card.
- configuration straps - a set of resistors used to configure various functions of the card that need to be up before the card is POSTed.
- a small I2C EEPROM with encrypted HDCP keys [optional, some G84:GT215, now discontinued in favor of storing the keys in fuses on the GPU]
- a voltage regulator [starting with NV10 [?] family] - starting with roughly NV30 family, the target voltage can be set via GPIO pins on the GPU. The voltage regulator may also have "power good" and "emergency shutdown" signals connected to the GPU via GPIOs. In some rare cases, particularly on high-end cards, the voltage regulator may also be accessible via I2C.

- optionally [usually on high-end cards], a thermal monitoring chip accessible via I2C, to supplement/replace the bultin thermal sensor of the GPU. May or may not include autonomous fan control and fan speed measurement capability. Usually has a "thermal alert" pin connected to a GPIO.
- a fan - control and speed measurement done either by the thermal monitoring chip, or by the GPU via GPIOs.
- SPDIF input [rare, some G84:GT215] - used for audio bypass to HDMI-capable TMDS outputs, newer GPUs include a builtin audio codec instead.
- on-chip video outputs - video output connectors connected directly to the GPU. Supported output types depend on the GPU and include VGA, TV [composite, S-Video, or component], TMDS [ie. the protocol used in DVI digital and HDMI], FPD-Link [aka LVDS], DisplayPort.
- external output encoders - usually found with older GPUs which don't support TV, TMDS or FPD-Link outputs directly. The encoder is connected to the GPU via a parallel data bus ["videolink"] and a controlling I2C bus.
- SLI connectors [optional, newer high-end cards only] - video links used to transmit video to display from slave cards in SLI configuration to the master. Uses the same circuitry as outputs to external output encoders.
- TV decoder chip [sometimes with a tuner] connected to the capture port of the GPU and to an I2C bus - rare, on old cards only
- external MPEG decoder chip connected to so-called mediaport on the GPU - alleged to exist on some NV3/NV4/NV10 cards, but never seen in the wild

In addition to normal cards, nvidia GPUs may be found integrated on motherboards - in this case they're often missing own BIOS and HDCP ROMs, instead having them intergrated with the main system ROM. There are also IGPs [Integrated Graphics Processors], which are a special variant of GPU integrated into the main system chipset. They don't have on-board memory or memory controller, sharing the main system RAM instead.

2.1.3 GPU schematic - NV3:G80





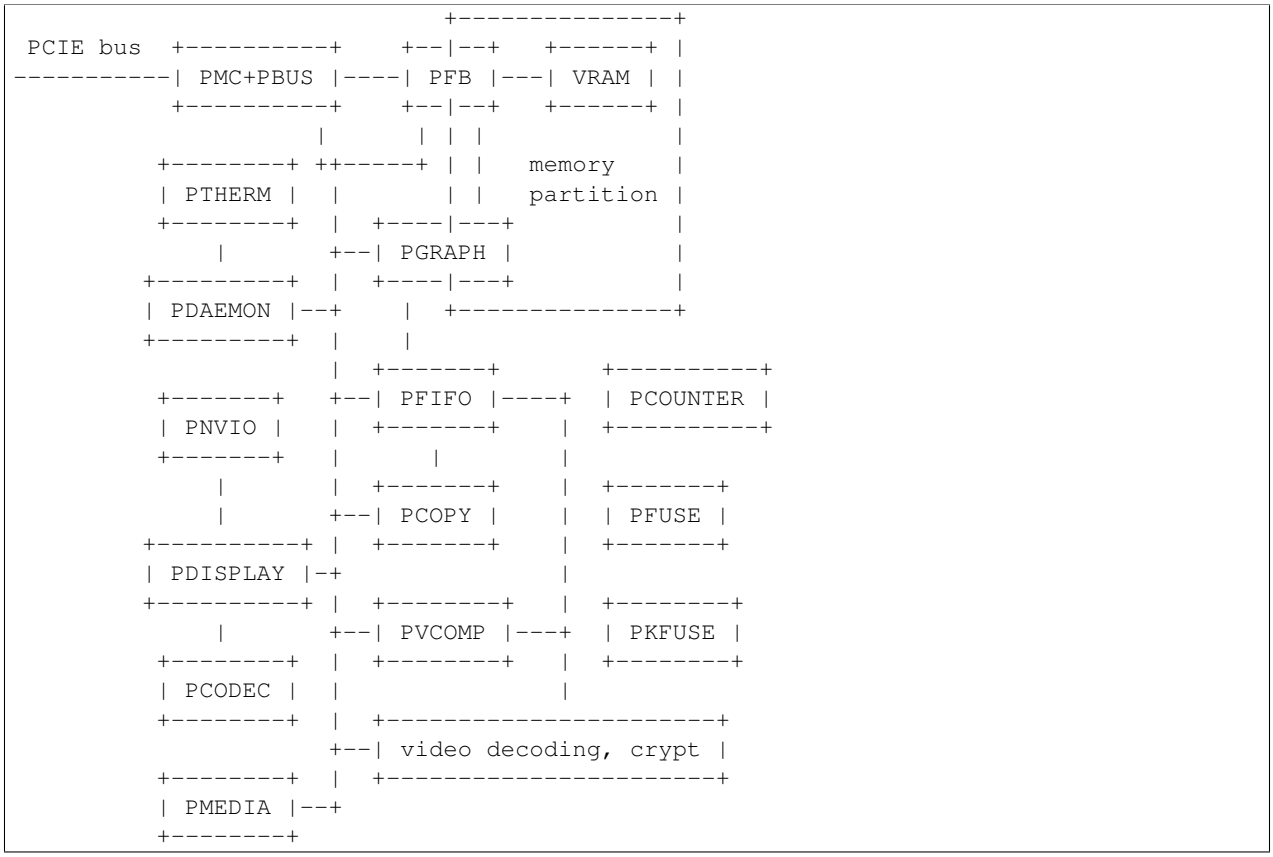
The GPU is made of:

- control circuitry:
  - PMC: master control area
  - PBUS: bus control and an area where "misc" registers are thrown in. Known to contain at least:
    - HWSQ, a simple script engine, can poke card registers and sleep in a given sequence [NV17+]
    - a thermal sensor [NV30+]
    - clock gating control [NV17+]
    - indirect VRAM access from host circuitry [NV30+]
    - ROM timings control
    - PWM controller for fans and panel backlight [NV17+]
  - PPMI: PCI Memory Interface, handles SYSRAM accesses from other units of the GPU
  - PTIMER: measures wall time and delivers alarm interrupts
  - PCLOCK+PCONTROL: clock generation and distribution [contained in PRAMDAC on pre-NV40 GPUs]
  - PFB: memory controller and arbiter
  - PROM: VIOS ROM access
  - PSTRAPS: configuration straps access
- processing engines:
  - *PFIFO*: gathers processing commands from the command buffers prepared by the host and delivers them to PGRAPH and PVPE engines in orderly manner
  - *PGRAPH*: memory copying, 2d and 3d rendering engine
  - PVPE: a trio of video decoding/encoding engines
    - *PMPEG*: MPEG1 and MPEG2 mocomp and IDCT decoding engine [NV17+]
    - PME: motion estimation engine [NV40+]
    - PVP1: VP1 video processor [NV41+]
  - PCOUNTER: performance monitoring counters for the processing engines and memory controller
- display engines:

  - PCRTC: generates display control signals and reads framebuffer data for display, present in two instances on NV11+ cards; also handles GPIO and I2C
  - PVIDEO: reads and preprocesses overlay video data
  - PRAMDAC: multiplexes PCRTC, PVIDEO and cursor image data, applies palette LUT, coverts to output signals, present in two instances on NV11+ cards; on pre-NV40 cards also deals with clock generation
  - PTV: an on-chip TV encoder
- misc engines:
  - PMEDIA: controls video capture input and the mediaport, acts as a DMA controller for them

Almost all units of the GPU are controlled through MMIO registers accessible by a common bus and visible through PCI BAR0 [see *PCI BARs and other means of accessing the GPU*]. This bus is not shown above.

2.1.4 GPU schematic - G80:GF100



The GPU is made of:

- control circuitry:
  - PMC: master control area
  - PBUS: bus control and an area where “misc” registers are thrown in. Known to contain at least:
    - HWSQ, a simple script engine, can poke card registers and sleep in a given sequence
    - clock gating control
    - indirect VRAM access from host circuitry

  - PTIMER: measures wall time and delivers alarm interrupts
  - PCLOCK+PCONTROL: clock generation and distribution
  - PTHERM: thermal sensor and clock throttling circuitry
  - *PDAEMON*: card management microcontroller
  - PFB: memory controller and arbiter
- processing engines:
  - *PFIFO*: gathers processing commands from the command buffers prepared by the host and delivers them to PGRAPH and PVPE engines in orderly manner
  - *PGRAPH*: memory copying, 2d and 3d rendering engine
  - video decoding engines, see below
  - PCOPY: asynchronous copy engine
  - PVCOMP: video compositing engine
  - PCOUNTER: performance monitoring counters for the processing engines and memory controller
- display and IO port units:
  - PNVIO: deals with misc external devices
    - GPIOs
    - fan PWM controllers
    - I2C bus controllers
    - videolink controls
    - ROM interface
    - straps interface
    - PNVIO/PDISPLAY clock generation
  - PDISPLAY: a unified display engine
  - PCODEC: audio codec for HDMI audio
- misc engines:
  - PMEDIA: controls video capture input and the mediaport, acts as a DMA controller for them

2.1.5 GPU schematic - GF100-

Todo

finish file

2.2 GPU chips

Contents

2.2.1 Introduction

Each nvidia GPU has several identifying numbers that can be used to determine supported features, the engines it contains, and the register set. The most important of these numbers is an 8-bit number known as the "GPU id". If two cards have the same GPU id, their GPUs support identical features, engines, and registers, with very minor exceptions. Such cards can however still differ in the external devices they contain: output connectors, encoders, capture chips, temperature sensors, fan controllers, installed memory, supported clocks, etc. You can get the GPU id of a card by reading from its PMC area.

The GPU id is usually written as NVxx, where xx is the id written as uppercase hexadecimal number. Note that, while cards before NV10 used another format for their ID register and don't have the GPU id stored directly, they are usually considered as NV1-NV5 anyway.

Nvidia uses "GPU code names" in their materials. They started out identical to the GPU id, but diverged midway through the NV40 series and started using a different numbering. However, for the most part nvidia code names correspond 1 to 1 with the GPU ids.

The GPU id has a mostly one-to-many relationship with pci device ids. Note that the last few bits [0-6 depending on GPU] of PCI device id are changeable through straps [see pstraps]. When pci ids of a GPU are listed in this file, the following shorthands are used:

1234 PCI device id 0x1234

1234* PCI device ids 0x1234-0x1237, choosable by straps

123X PCI device ids 0x1230-0x123X, choosable by straps

124X+ PCI device ids 0x1240-0x125X, choosable by straps

124X* PCI device ids 0x1240-0x127X, choosable by straps

2.2.2 The GPU families

The GPUs can roughly be grouped into a dozen or so families: NV1, NV3/RIVA, NV4/TNT, Celsius, Kelvin, Rankine, Curie, Tesla, Fermi, Kepler, Maxwell. This aligns with big revisions of PGRAPH, the drawing engine of the card. While most functionality was introduced in sync with PGRAPH revisions, some other functionality [notably video decoding hardware] gets added in GPUs late in a GPU family and sometimes doesn't even get to the first GPU in the next GPU family. For example, NV11 expanded upon the previous NV15 chipset by adding dual-head support, while NV20 added new PGRAPH revision with shaders, but didn't have dual-head - the first GPU to feature both was NV25.

Also note that a bigger GPU id doesn't always mean a newer card / card with more features: there were quite a few places where the numbering actually went backwards. For example, NV11 came out later than NV15 and added several features.

Nvidia's card release cycle always has the most powerful high-end GPU first, subsequently filling in the lower-end positions with new cut-down GPUs. This means that newer cards in a single sub-family get progressively smaller, but also more featureful - the first GPUs to introduce minor changes like DX10.1 support or new video decoding are usually the low-end ones.

The full known GPU list, sorted roughly according to introduced features, is:

- NV1 family: NV1
- NV3 (aka RIVA) family: NV3, NV3T
- NV4 (aka TNT) family: NV4, NV5
- Calsius family: NV10, NV15, NV1A, NV11, NV17, NV1F, NV18
- Kelvin family: NV20, NV2A, NV25, NV28
- Rankine family: NV30, NV35, NV31, NV36, NV34
- Curie family:
  - NV40 subfamily: NV40, NV45, NV41, NV42, NV43, NV44, NV44A
  - G70 subfamily: G70, G71, G73, G72
  - the IGPs: C51, MCP61, MCP67, MCP68, MCP73
- Tesla family:
  - G80 subfamily: G80
  - G84 subfamily: G84, G86, G92, G94, G96, G98
  - G200 subfamily: G200, MCP77, MCP79
  - GT215 subfamily: GT215, GT216, GT218, MCP89
- Fermi family:
  - GF100 subfamily: GF100, GF104, GF106, GF114, GF116, GF108, GF110
  - GF119 subfamily: GF119, GF117
- Kepler family: GK104, GK107, GK106, GK110, GK110B, GK208, GK208B, GK20A
- Maxwell family: GM107, GM200, GM204

Whenever a range of GPUs is mentioned in the documentation, it's written as "NVxx:NVyy". This is left-inclusive, right-noninclusive range of GPU ids as sorted in the preceding list. For example, G200:GT218 means GPUs G200, MCP77, MCP79, GT215, GT216. NV20:NV30 effectively means all NV20 family GPUs.

NV1 family: NV1

The first nvidia GPU. It has semi-legendary status, as it's very rare and hard to get. Information is mostly guesswork from ancient xfree86 driver. The GPU is also known by its SGS-Thomson code number, SGS-2000. The most popular card using this GPU is Diamond EDGE 3D.

The GPU has integrated audio output, MIDI synthetiser and Sega Saturn game controller port. Its rendering pipeline, as opposed to all later families, deals with quadratic surfaces, as opposed to triangles. Its video output circuitry is also totally different from NV3+, and replaces the VGA part as opposed to extending it like NV3:G80 do.

There's also NV2, which has even more legendary status. It was supposed to be another card based on quadratic surfaces, but it got stuck in development hell and never got released. Apparently it never got to the stage of functioning silicon.

The GPU was jointly manufactured by SGS-Thomson and NVidia, earning it pci vendor id of 0x12d2. The pci device ids are 0x0008 and 0x0009. The device id of NV2 was supposed to be 0x0010.

id	GPU	date
0008/0009	NV1	09.1995

NV3 (RIVA) family: NV3, NV3T

The first [moderately] sane GPUs from nvidia, and also the first to use AGP bus. There are two chips in this family, and confusingly both use GPU id NV3, but can be told apart by revision. The original NV3 is used in RIVA 128 cards, while the revised NV3, known as NV3T, is used in RIVA 128 ZX. NV3 supports AGP 1x and a maximum of 4MB of VRAM, while NV3T supports AGP 2x and 8MB of VRAM. NV3T also increased number of slots in PFIFO cache. These GPUs were also manufactured by SGS-Thomson and bear the code name of STG-3000.

The pci vendor id is 0x12d2. The pci device ids are:

id	GPU	date
0018	NV3	??.04.1997
0019	NV3T	23.02.1998

The NV3 GPU is made of the following functional blocks:

- host interface, connected to the host machine via PCI or AGP
- two PLLs, to generate video pixel clock and memory clock
- memory interface, connected to 2MB-8MB of external VRAM via 64-bit or 128-bit memory bus, shared with an 8-bit parallel flash ROM
- PFIFO, controlling command submission to PGRAPH and gathering commands through DMA to host memory or direct MMIO submission
- PGRAPH, the 2d/3d drawing engine, supporting windows GDI and Direct3D 5 acceleration
- VGA-compatible CRTC, RAMDAC, and associated video output circuitry, enabling direct connection of VGA analog displays and TV connection via an external AD722 encoder chip
- i2c bus to handle DDC and control mediaport devices
- double-buffered video overlay and cursor circuitry in RAMDAC
- mediaport, a proprietary interface with ITU656 compatibility mode, allowing connection of external video capture or MPEG2 decoding chip

NV3 introduced RAMIN, an area of memory at the end of VRAM used to hold various control structures for PFIFO and PGRAPH. On NV3, RAMIN can be accessed in BAR1 at addresses starting from 0xc00000, while later cards have it in BAR0. It also introduced DMA objects, a RAMIN structure used to define a VRAM or host memory area that PGRAPH is allowed to use when executing commands on behalf of an application. These early DMA objects are limitted to linear VRAM and paged host memory objects, and have to be switched manually by host. See *NV3 DMA objects* for details.

NV4 (TNT) family: NV4, NV5

Improved and somewhat redesigned NV3. Notable changes:

- AGP x4 support

- redesigned and improved DMA command submission
- separated core and memory clocks
- DMA objects made more orthogonal, and switched automatically by card
- redesigned PGRAPH objects, introducing the concept of object class in hardware
- added BIOS ROM shadow in RAMIN
- Direct3D 6 / multitexturing support in PGRAPH
- bumped max supported VRAM to 16MB
- [NV5] bumped max supported VRAM to 32MB
- [NV5] PGRAPH 2d context object binding in hardware

This family includes the original NV4, used in RIVA TNT cards, and NV5 used in RIVA TNT2 and Vanta cards.

This is the first chip marked as solely nvidia chip, the pci vendor id is 0x10de. The pci device ids are:

id	GPU	date
0020	NV4	23.03.1998
0028*	NV5	15.03.1998
002c*	NV5	15.03.1998
00a0	NVA IGP	08.09.1999

Todo

what the fuck?

Celsius family: NV10, NV15, NV1A, NV11, NV17, NV1F, NV18

The notable changes in this generation are:

- NV10:
  - redesigned memory controller
  - max VRAM bumped to 128MB
  - redesigned VRAM tiling, with support for multiple tiled regions
  - greatly expanded 3d engine: hardware T&L, D3D7, and other features
  - GPIO pins introduced for ???
  - PFIFO: added REF_CNT and NONINC commands
  - added PCOUNTER: the performance monitoring engine
  - new and improved video overlay engine
  - redesigned mediaport
- NV15:
  - introduced vblank wait PGRAPH commands
  - minor 3d engine additions [logic operation, ...]
- NV1A:
  - big endian mode

  - PFIFO: semaphores and subroutines
- NV11:
  - dual head support, meant for laptops with flat panel + external display
- NV17:
  - builtin TV encoder
  - ZCULL
  - added VPE: MPEG2 decoding engine
- NV18:
  - AGP x8 support
  - second straps set

Todo

what were the GPIOs for?

The GPUs are:

pciid	GPU	pixel pipelines and ROPs	texture units	date	notes
0100*	NV10	4	4	11.10.1999	the first GeForce card [GeForce 256]
0150*	NV15	4	8	26.04.2000	the high-end card of GeForce 2 lineup [GeForce 2 Ti, ...]
01a0*	NV1A	2	4	04.06.2001	the IGP of GeForce 2 lineup [nForce]
0110*	NV11	2	4	28.06.2000	the low-end card of GeForce 2 lineup [GeForce 2 MX]
017X	NV17	2	4	06.02.2002	the low-end card of GeForce 4 lineup [GeForce 4 MX]
01fX	NV1F	2	4	01.10.2002	the IGP of GeForce 4 lineup [nForce 2]
018X	NV18	2	4	25.09.2002	like NV17, but with added AGP x8 support

The pci vendor id is 0x10de.

NV1A and NV1F are IGPs and lack VRAM, memory controller, mediaport, and ROM interface. They use the internal interfaces of the northbridge to access an area of system memory set aside as fake VRAM and BIOS image.

Kelvin family: NV20, NV2A, NV25, NV28

The first cards of this family were actually developed before NV17, so they miss out on several features introduced in NV17. The first card to merge NV20 and NV17 additions is NV25. Notable changes:

- NV20:
  - no dual head support again
  - no PTV, VPE
  - no ZCULL
  - a new memory controller with Z compression
  - RAMIN reversal unit bumped to 0x40 bytes
  - 3d engine extensions:

        * programmable vertex shader support
        * D3D8, shader model 1.1
    - PGRAPH automatic context switching
- NV25:
    - a merge of NV17 and NV20: has dual-head, ZCULL, ...
    - still no VPE and PTV
- NV28:
    - AGP x8 support

The GPUs are:

pciid	GPU	vertex shaders	pixel pipelines and ROPs	texture units	date	notes
0200*	NV20	1	4	8	27.02.2001	the only GPU of GeForce 3 lineup [GeForce 3 Ti, ...]
02a0*	NV2A	2	4	8	15.11.2001	the XBOX IGP [XGPU]
025X	NV25	2	4	8	06.02.2002	the high-end GPU of GeForce 4 lineup [GeForce 4 Ti]
028X	NV28	2	4	8	20.01.2003	like NV25, but with added AGP x8 support

NV2A is a GPU designed exclusively for the original xbox, and can't be found anywhere else. Like NV1A and NV1F, it's an IGP.

Todo

verify all sorts of stuff on NV2A

The pci vendor id is 0x10de.

Rankine family: NV30, NV35, NV31, NV36, NV34

The infamous GeForce FX series. Notable changes:

- NV30:
    - 2-stage PLLs introduced [still located in PRAMDAC]
    - max VRAM size bumped to 256MB
    - 3d engine extensions:
        * programmable fragment shader support
        * D3D9, shader model 2.0
    - added PEEPHOLE indirect memory access
    - return of VPE and PTV
    - new-style memory timings
- NV35:
    - 3d engine additions:
        * ???

- NV31:
  - no NV35 changes, this GPU is derived from NV30
  - 2-stage PLLs split into two registers
  - VPE engine extended to work as a PFIFO engine
- NV36:
  - a merge of NV31 and NV35 changes from NV30
- NV34:
  - a comeback of NV10 memory controller!
  - NV10-style mem timings again
  - no Z compression again
  - RAMIN reversal unit back at 16 bytes
  - 3d engine additions:
    * ???

Todo

figure out 3d engine changes

The GPUs are:

pciid	GPU	vertex shaders	pixel pipelines and ROPs	date	notes
030X	NV30	2	8	27.01.2003	high-end GPU [GeForce FX 5800]
033X	NV35	3	8	12.05.2003	very high-end GPU [GeForce FX 59X0]
031X	NV31	1	4	06.03.2003	low-end GPU [GeForce FX 5600]
034X	NV36	3	4	23.10.2003	middle-end GPU [GeForce FX 5700]
032X	NV34	1	4	06.03.2003	low-end GPU [GeForce FX 5200]

The pci vendor id is 0x10de.

Curie family

This family was the first to feature PCIE cards, and many fundamental areas got significant changes, which later paved the way for G80. It is also the family where GPU ids started to diverge from nvidia code names. The changes:

- NV40:
  - RAMIN bumped in size to max 16MB, many structure layout changes
  - RAMIN reversal unit bumped to 512kB
  - 3d engine: support for shader model 3 and other additions
  - Z compression came back
  - PGRAPH context switching microcode
  - redesigned clock setup
  - separate clock for shaders

  - rearranged PCOUNTER to handle up to 8 clock domains
  - PFIFO cache bumped in size and moved location
  - added independent PRMVIO for two heads
  - second set of straps added, new strap override registers
  - new PPCI PCI config space access window
  - MPEG2 encoding capability added to VPE
  - FIFO engines now identify the channels by their context addresses, not chids
  - BIOS uses all-new BIT structure to describe the card
  - individually disablable shader and ROP units.
  - added PCONTROL area to... control... stuff?
  - memory controller uses NV30-style timings again
- NV41:
  - introduced context switching to VPE
  - introduced PVP1, microcoded video processor
  - first natively PCIE card
  - added PCIE GART to memory controller
- NV43:
  - added a thermal sensor to the GPU
- NV44:
  - a new PCIE GART page table format
  - 3d engine: ???
- NV44A:
  - like NV44, but AGP instead of PCIE

Todo

more changes

Todo

figure out 3d engine changes

The GPUs are [vertex shaders : pixel shaders : ROPs]:

pciid	GPU id	GPU names	vertex shaders	pixel shaders	ROPs	date	notes
004X 021X	0x40/0x45	NV40/NV45/NV48	6	16	16	14.04.2004	AGP
00cX	0x41/0x42	NV41/NV42	5	12	12	08.11.2004	
014X	0x43	NV43	3	8	4	12.08.2004	
016X	0x44	NV44	3	4	2	15.12.2004	TURBOCACHE
022X	0x4a	NV44A	3	4	2	04.04.2005	AGP
009X	0x47	G70	8	24	16	22.06.2005	
01dX	0x46	G72	3	4	2	18.01.2006	TURBOCACHE
029X	0x49	G71	8	24	16	09.03.2006	
039X	0x4b	G73	8	12	8	09.03.2006	
024X	0x4e	C51	1	2	1	20.10.2005	IGP, TURBOCACHE
03dX	0x4c	MCP61	1	2	1	??.06.2006	IGP, TURBOCACHE
053X	0x67	MCP67	1	2	2	01.02.2006	IGP, TURBOCACHE
053X	0x68	MCP68	1	2	2	??.07.2007	IGP, TURBOCACHE
07eX	0x63	MCP73	1	2	2	??.07.2007	IGP, TURBOCACHE
-	???	RSX	?	?	?	11.11.2006	FlexIO bus interface, used in PS3

Todo

all geometry information unverified

Todo

any information on the RSX?

It's not clear how NV40 is different from NV45, or NV41 from NV42, or MCP67 from MCP68 - they even share pciid ranges.

The NV4x IGPs actually have a memory controller as opposed to earlier ones. This controller still accesses only host memory, though.

As execution units can be disabled on NV40+ cards, these configs are just the maximum configs - a card can have just a subset of them enabled.

Tesla family

The card where they redesigned everything. The most significant change was the redesigned memory subsystem, complete with a paging MMU [see *Tesla virtual memory*].

- G80:
  - a new VM subsystem, complete with redesigned DMA objects
  - RAMIN is gone, all structures can be placed arbitrarily in VRAM, and usually host memory memory as well
  - all-new channel structure storing page tables, RAMFC, RAMHT, context pointers, and DMA objects
  - PFIFO redesigned, PIO mode dropped
  - PGRAPH redesigned: based on unified shader architecture, now supports running standalone computations, D3D10 support, unified 2d acceleration object

  - display subsystem reinvented from scratch: a stub version of the old VGA-based one remains for VGA compatibility, the new one is not VGA based and is controlled by PFIFO-like DMA push buffers
  - memory partitions tied directly to ROPs
- G84:
  - redesigned channel structure with a new layout
  - got rid of VP1 video decoding and VPE encoding support, but VPE decoder still exists
  - added VP2 xtensa-based programmable video decoding and BSP engines
  - removed restrictions on host memory access by rendering: rendering to host memory and using blocklinear textures from host are now ok
  - added VM stats write support to PCOUNTER
  - PEEPHOLE moved out of PBUS
  - PFIFO_BAR_FLUSH moved out of PFIFO
- G98:
  - introduced VP3 video decoding engines, and the falcon microcode with them
  - got rid of VP2 video decoding
- G200:
  - developped in parallel with G98
  - VP2 again, no VP3
  - PGRAPH rearranged to make room for more MPs/TPCs
  - streamout enhancements [ARB_transform_feedback2]
  - CUDA ISA 1.3: 64-bit g[] atomics, s[] atomics, voting, fp64 support
- MCP77:
  - merged G200 and G98 changes: has both VP3 and new PGRAPH
  - only CUDA ISA 1.2 now: fp64 support got cut out again
- GT215:
  - a new revision of the falcon ISA
  - a revision to VP3 video decoding, known as VP4. Adds MPEG-4 ASP support.
  - added PDAEMON, a falcon engine meant to do card monitoring and power maanagement
  - PGRAPH additions for D3D10.1 support
  - added HDA audio codec for HDMI sound support, on a separate PCI function
  - Added PCOPY, the dedicated copy engine
  - Merged PSEC functionality into PVLD
- MCP89:
  - added PVCOMP, the video compositor engine

The GPUs in this family are:

core	hda	id	name	TPCs	MPs/TPC	PARTs	date	notes
pciid	pciid							
019X	-	0x50	G80	8	2	6	08.11.2006	
040X	-	0x84	G84	2	2	2	17.04.2007	
042X	-	0x86	G86	1	2	2	17.04.2007	
060X+	-	0x92	G92	8	2	4	29.10.2007	
062X+	-	0x94	G94	4	2	4	29.07.2008	
064X+	-	0x96	G96	2	2	2	29.07.2008	
06eX+	-	0x98	G98	1	1	1	04.12.2007	
05eX+	-	0xa0	G200	10	3	8	16.06.2008	
084X+	-	0xaa	MCP77/MCP78	1	1	1	??.06.2008	IGP
086X+	-	0xac	MCP79/MCP7A	1	2	1	??.06.2008	IGP
0caX+	0be4	0xa3	GT215	4	3	2	15.06.2009	
0a2X+	0be2	0xa5	GT216	2	3	2	15.06.2009	
0a6X+	0be3	0xa8	GT218	1	2	1	15.06.2009	
08aX+	-	0xaf	MCP89	2	3	2	01.04.2010	IGP

Like NV40, these are just the maximal numbers.

The pci vendor id is 0x10de.

Todo

geometry information not verified for G94, MCP77

Fermi/Kepler/Maxwell family

The card where they redesigned everything again.

- GF100:
  - redesigned PFIFO, now with up to 3 subfifos running in parallel
  - redesigned PGRAPH:
    - split into a central HUB managing everything and several GPCs doing all actual work
    - GPCs further split into a common part and several TPCs
    - using falcon for context switching
    - D3D11 support
  - redesigned memory controller
    - split into three parts:
      - per-partition low-level memory controllers [PBFB]
      - per-partition middle memory controllers: compression, ECC, ... [PMFB]
      - a single "hub" memory controller: VM control, TLB control, ... [PFFB]
  - memory partitions, GPCs, TPCs have independent register areas, as well as "broadcast" areas that can be used to control all units at once
  - second PCOPY engine

  - redesigned PCOUNTER, now having multiple more or less independent subunits to monitor various parts of GPU
  - redesigned clock setting
  - ...
- GF119:
  - a major revision to VP3 video decoding, now called VP5. vμc microcode removed.
  - another revision to the falcon ISA, allowing 24-bit PC
  - added PUNK1C3 falcon engine
  - redesigned I2C bus interface
  - redesigned PDISPLAY
  - removed second PCOPY engine
- GF117:
  - PGRAPH changes:
    * ???
- GK104:
  - redesigned PCOPY: the falcon controller is now gone, replaced with hardware control logic, partially in PFIFO
  - an additional PCOPY engine
  - PFIFO redesign - a channel can now only access a single engine selected on setup, with PCOPY2+PGRAPH considered as one engine
  - PGRAPH changes:
    * subchannel to object assignments are now fixed
    * m2mf is gone and replaced by a new p2mf object that only does simple upload, other m2mf functions are now PCOPY's responsibility instead
    * the ISA requires explicit scheduling information now
    * lots of setup has been moved from methods/registers into memory structures
    * ???
- GK110:
  - PFIFO changes:
    * ???
  - PGRAPH changes:
    * ISA format change
    * ???

Todo

figure out PGRAPH/PFIFO changes

GPUs in Fermi/Kepler/Maxwell families:

core pciid	hda pciid	id	name	GPCs	TPCs /GPC	PARTs	MCs	ZCULLs /GPC	PCOPYs	HEADs	UNK7	PPCs /GPC	SUBPs /PART	SPOON
06cX+	0be5	0xc0	GF100	4	4	6	[6]	[4]	[2]	[2]	-	-	2	3
0e2X+	0beb	0xc4	GF104	2	4	4	[4]	[4]	[2]	[2]	-	-	2	3
120X+	0e0c	0xce	GF114	2	4	4	[4]	[4]	[2]	[2]	-	-	2	3
0dcX+	0be9	0xc3	GF106	1	4	3	[3]	[4]	[2]	[2]	-	-	2	3
124X+	0bee	0xcf	GF116	1	4	3	[3]	[4]	[2]	[2]	-	-	2	3
0deX+	0bea	0xc1	GF108	1	2	1	2	4	[2]	[2]	-	-	2	1
108X+	0e09	0xc8	GF110	4	4	6	[6]	[4]	[2]	[2]	-	-	2	3
104X*	0e08	0xd9	GF119	1	1	1	1	4	1	2	-	-	1	1
1140	-	0xd7	GF117	1	2	1	1	4	1	4	-	1	2	1
118X*	0e0a	0xe4	GK104	4	2	4	4	4	3	4	-	1	4	3
0fcX*	0e1b	0xe7	GK107	1	2	2	2	4	3	4	-	1	4	3
11cX+	0e0b	0xe6	GK106	3	2	3	3	4	3	4	-	1	4	3
100X+	0e1a	0xf0	GK110	5	3	6	6	4	3	4	-	2	4	3
100X+	0e1a	0xf1	GK110B	5	3	6	6	4	3	4	-	2	4	3
128X+	0e0f	0x108	GK208	1	2	1	1	4	3	4	-	1	2	2
128X+	0e0f	0x106	GK208B	?	?	?	?	?	?	?	-	?	?	?
-	-	0xea	GK20A	?	?	?	?	?	?	-	-	?	?	1
138X+	0fbc	0x117	GM107	1	5	2	2	4	3	4	1	2	4	2
17cX+	0fb0	0x120	GM200	?	?	?	?	?	?	?	?	?	?	?
unk?		0x124	GM204	?	?	?	?	?	?	?	?	?	?	?

Todo

it is said that one of the GPCs [0th one] has only one TPC on GK106

Todo

what the fuck is GK110B? and GK208B?

Todo

GK20A

Todo

GM200

Todo

GM204

Todo

another design counter available on GM107

2.3 nVidia PCI id database

Contents

2.3.1 Introduction

nVidia uses PCI vendor id of `0x10de`, which covers almost all of their products. Other ids used for nVidia products include `0x104a` (SGS-Thompson) and `0x12d2` (SGS-Thompson/nVidia joint venture). The PCI device ids with vendor id `0x104a` related to nVidia are:

device id	product
`0x0008`	NV1 main function, DRAM version (SGS-Thompson branding)
`0x0009`	NV1 VGA function, DRAM version (SGS-Thompson branding)

The PCI device ids with vendor id `0x12d2` are:

device id	product
`0x0018`	NV3 [RIVA 128]
`0x0019`	NV3T [RIVA 128 ZX]

All other nVidia PCI devices use vendor id `0x10de`. This includes:

- GPUs
- motherboard chipsets
- BR03 and NF200 PCIE switches
- the BR02 transparent AGP/PCIE bridge
- GVI, the SDI input card

The PCI device ids with vendor id `0x10de` are:

device id	product
`0x0008`	NV1 main function, VRAM version (nVidia branding)
`0x0009`	NV1 VGA function, VRAM version (nVidia branding)
`0x0020`	NV4 [RIVA TNT]
`0x0028-0x002f`	*NV5*
`0x0030-0x003f`	*MCP04*
`0x0040-0x004f`	*NV40*
`0x0050-0x005f`	*CK804*
`0x0060-0x006e`	*MCP2*
`0x006f-0x007f`	*C19*
`0x0080-0x008f`	*MCP2A*
`0x0090-0x009f`	*G70*
`0x00c0-0x00cf`	*NV41/NV42*
`0x00a0`	NVA [Aladdin TNT2]
`0x00b0`	*NV18 Firewire*
`0x00b4`	*C19*
`0x00d0-0x00d2`	*CK8*
`0x00d3`	*CK804*
`0x00d4-0x00dd`	*CK8*
`0x00df-0x00ef`	*CK8S*
`0x00f0-0x00ff`	*BR02*
`0x0100-0x0103`	*NV10*
`0x0110-0x0113`	*NV11*
`0x0140-0x014f`	*NV43*
`0x0150-0x0153`	*NV15*
`0x0160-0x016f`	*NV44*
`0x0170-0x017f`	*NV17*

Continued on next page

Table 2.1 – continued from previous page

device id	product
0x0180-0x018f	*NV18*
0x0190-0x019f	*G80*
0x01a0-0x01af	*NV1A*
0x01b0-0x01b2	*MCP*
0x01b3	*BR03*
0x01b4-0x01b2	*MCP*
0x01b7	*NV1A*, *NV2A*
0x01b8-0x01cf	*MCP*
0x01d0-0x01df	*G72*
0x01e0-0x01f0	*NV1F*
0x01f0-0x01ff	*NV1F GPU*
0x0200-0x0203	*NV20*
0x0210-0x021f	*NV40*?
0x0220-0x022f	*NV44A*
0x0240-0x024f	*C51 GPU*
0x0250-0x025f	*NV25*
0x0260-0x0272	*MCP51*
0x027e-0x027f	*C51*
0x0280-0x028f	*NV28*
0x0290-0x029f	*G71*
0x02a0-0x02af	*NV2A*
0x02e0-0x02ef	*BR02*
0x02f0-0x02ff	*C51*
0x0300-0x030f	*NV30*
0x0310-0x031f	*NV31*
0x0320-0x032f	*NV34*
0x0330-0x033f	*NV35*
0x0340-0x034f	*NV36*
0x0360-0x037f	*MCP55*
0x0390-0x039f	*G73*
0x03a0-0x03bc	*C55*
0x03d0-0x03df	*MCP61 GPU*
0x03e0-0x03f7	*MCP61*
0x0400-0x040f	*G84*
0x0410-0x041f	*G92* extra IDs
0x0420-0x042f	*G86*
0x0440-0x045f	*MCP65*
0x0530-0x053f	*MCP67 GPU*
0x0540-0x0563	*MCP67*
0x0568-0x0569	*MCP77*
0x056a-0x056f	*MCP73*
0x0570-0x057f	MCP* ethernet alt ID
0x0580-0x058f	MCP* SATA alt ID
0x0590-0x059f	MCP* HDA alt ID
0x05a0-0x05af	MCP* IDE alt ID
0x05b0-0x05bf	*BR04*
0x05e0-0x05ff	*G200*
0x0600-0x061f	*G92*
0x0620-0x063f	*G94*

Continued on next page

Table 2.1 – continued from previous page

device id	product
0x0640-0x065f	*G96*
0x06c0-0x06df	*GF100*
0x06e0-0x06ff	*G98*
0x0750-0x077f	*MCP77*
0x07c0-0x07df	*MCP73*
0x07e0-0x07ef	*MCP73 GPU*
0x07f0-0x07fe	*MCP73*
0x0800-0x081a	*C73*
0x0840-0x085f	*MCP77 GPU*
0x0860-0x087f	*MCP79 GPU*
0x08a0-0x08bf	*MCP89 GPU*
0x0a20-0x0a3f	*GT216*
0x0a60-0x0a7f	*GT218*
0x0a80-0x0ac8	*MCP79*
0x0ad0-0x0adb	*MCP77*
0x0be0-0x0bef	*GPU HDA*
0x0bf0-0x0bf1	*T20*
0x0ca0-0x0cbf	*GT215*
0x0d60-0x0d9d	*MCP89*
0x0dc0-0x0ddf	*GF106*
0x0de0-0x0dff	*GF108*
0x0e00	GVI SDI input
0x0e01-0x0e1b	*GPU HDA*
0x0e1c-0x0e1d	*T30*
0x0e20-0x0e3f	*GF104*
0x0f00-0x0f1f	*GF108* extra IDs
0x0fb0-0x0fbf	*GPU HDA*
0x0fc0-0x0fff	*GK107*
0x1000-0x103f	*GK110/GK110B*
0x1040-0x107f	*GF119*
0x1080-0x109f	*GF110*
0x10c0-0x10df	*GT218* extra IDs
0x1140-0x117f	*GF117*
0x1180-0x11bf	*GK104*
0x11c0-0x11ff	*GK106*
0x1200-0x121f	*GF114*
0x1240-0x125f	*GF116*
0x1280-0x12bf	*GK208*
0x1340-0x137f	*GM108*
0x1380-0x13bf	*GM107*
0x13c0-0x13ff	*GM204*
0x1400-0x143f	*GM206*

2.3.2 GPUs

NV5

device id	product
0x0028	NV5 [RIVA TNT2]
0x0029	NV5 [RIVA TNT2 Ultra]
0x002c	NV5 [Vanta]
0x002d	NV5 [RIVA TNT2 Model 64]

NV10

device id	product
0x0100	NV10 [GeForce 256 SDR]
0x0101	NV10 [GeForce 256 DDR]
0x0102	NV10 [GeForce 256 Ultra]
0x0103	NV10 [Quadro]

NV15

device id	product
0x0150	NV15 [GeForce2 GTS/Pro]
0x0151	NV15 [GeForce2 Ti]
0x0152	NV15 [GeForce2 Ultra]
0x0153	NV15 [Quadro2 Pro]

NV11

device id	product
0x0110	NV11 [GeForce2 MX/MX 400]
0x0111	NV11 [GeForce2 MX 100/200]
0x0112	NV11 [GeForce2 Go]
0x0113	NV11 [Quadro2 MXR/EX/Go]

NV20

device id	product
0x0200	NV20 [GeForce3]
0x0201	NV20 [GeForce3 Ti 200]
0x0202	NV20 [GeForce3 Ti 500]
0x0203	NV20 [Quadro DCC]

NV17

device id	product
0x0170	NV17 [GeForce4 MX 460]
0x0171	NV17 [GeForce4 MX 440]
0x0172	NV17 [GeForce4 MX 420]
0x0173	NV17 [GeForce4 MX 440-SE]
0x0174	NV17 [GeForce4 440 Go]
0x0175	NV17 [GeForce4 420 Go]
0x0176	NV17 [GeForce4 420 Go 32M]
0x0177	NV17 [GeForce4 460 Go]
0x0178	NV17 [Quadro4 550 XGL]
0x0179	NV17 [GeForce4 440 Go 64M]
0x017a	NV17 [Quadro NVS 100/200/400]
0x017b	NV17 [Quadro4 550 XGL]???
0x017c	NV17 [Quadro4 500 GoGL]
0x017d	NV17 [GeForce4 410 Go 16M]

NV18

device id	product
0x0181	NV18 [GeForce4 MX 440 AGP 8x]
0x0182	NV18 [GeForce4 MX 440-SE AGP 8x]
0x0183	NV18 [GeForce4 MX 420 AGP 8x]
0x0185	NV18 [GeForce4 MX 4000]
0x0186	NV18 [GeForce4 448 Go]
0x0187	NV18 [GeForce4 488 Go]
0x0188	NV18 [Quadro4 580 XGL]
0x0189	NV18 [GeForce4 MX AGP 8x (Mac)]
0x018a	NV18 [Quadro NVS 280 SD]
0x018b	NV18 [Quadro4 380 XGL]
0x018c	NV18 [Quadro NVS 50 PCI]
0x018d	NV18 [GeForce4 448 Go]
0x00b0	NV18 Firewire controller

NV1F (GPU)

device id	product
0x01f0	NV1F GPU [GeForce4 MX IGP]

NV25

device id	product
0x0250	NV25 [GeForce4 Ti 4600]
0x0251	NV25 [GeForce4 Ti 4400]
0x0252	NV25 [GeForce4 Ti]
0x0253	NV25 [GeForce4 Ti 4200]
0x0258	NV25 [Quadro4 900 XGL]
0x0259	NV25 [Quadro4 750 XGL]
0x025b	NV25 [Quadro4 700 XGL]

NV28

device id	product
0x0280	NV28 [GeForce4 Ti 4800]
0x0281	NV28 [GeForce4 Ti 4200 AGP 8x]
0x0282	NV28 [GeForce4 Ti 4800 SE]
0x0286	NV28 [GeForce4 Ti 4200 Go]
0x0288	NV28 [Quadro4 980 XGL]
0x0289	NV28 [Quadro4 780 XGL]
0x028c	NV28 [Quadro4 700 GoGL]

NV30

device id	product
0x0301	NV30 [GeForce FX 5800 Ultra]
0x0302	NV30 [GeForce FX 5800]
0x0308	NV35 [Quadro FX 2000]
0x0309	NV35 [Quadro FX 1000]

NV31

device id	product
0x0311	NV31 [GeForce FX 5600 Ultra]
0x0312	NV31 [GeForce FX 5600]
0x0314	NV31 [GeForce FX 5600XT]
0x031a	NV31 [GeForce FX Go5600]
0x031b	NV31 [GeForce FX Go5650]
0x031c	NV31 [GeForce FX Go700]

NV34

device id	product
0x0320	NV34 [GeForce FX 5200]
0x0321	NV34 [GeForce FX 5200 Ultra]
0x0322	NV34 [GeForce FX 5200]
0x0323	NV34 [GeForce FX 5200LE]
0x0324	NV34 [GeForce FX Go5200]
0x0325	NV34 [GeForce FX Go5250]
0x0326	NV34 [GeForce FX 5500]
0x0327	NV34 [GeForce FX 5100]
0x0328	NV34 [GeForce FX Go5200 32M/64M]
0x0329	NV34 [GeForce FX Go5200 (Mac)]
0x032a	NV34 [Quadro NVS 280 PCI]
0x032b	NV34 [Quadro FX 500/FX 600]
0x032c	NV34 [GeForce FX Go5300/Go5350]
0x032d	NV34 [GeForce FX Go5100]

NV35

device id	product
0x0330	NV35 [GeForce FX 5900 Ultra]
0x0331	NV35 [GeForce FX 5900]
0x0332	NV35 [GeForce FX 5900XT]
0x0333	NV35 [GeForce FX 5950 Ultra]
0x0334	NV35 [GeForce FX 5900ZT]
0x0338	NV35 [Quadro FX 3000]
0x033f	NV35 [Quadro FX 700]

NV36

device id	product
0x0341	NV36 [GeForce FX 5700 Ultra]
0x0342	NV36 [GeForce FX 5700]
0x0343	NV36 [GeForce FX 5700LE]
0x0344	NV36 [GeForce FX 5700VE]
0x0347	NV36 [GeForce FX Go5700]
0x0348	NV36 [GeForce FX Go5700]
0x034c	NV36 [Quadro FX Go1000]
0x034e	NV36 [Quadro FX 1100]

NV40

device id	product
0x0040	NV40 [GeForce 6800 Ultra]
0x0041	NV40 [GeForce 6800]
0x0042	NV40 [GeForce 6800 LE]
0x0043	NV40 [GeForce 6800 XE]
0x0044	NV40 [GeForce 6800 XT]
0x0045	NV40 [GeForce 6800 GT]
0x0046	NV40 [GeForce 6800 GT]
0x0047	NV40 [GeForce 6800 GS]
0x0048	NV40 [GeForce 6800 XT]
0x004e	NV40 [Quadro FX 4000]
0x0211	NV40? [GeForce 6800]
0x0212	NV40? [GeForce 6800 LE]
0x0215	NV40? [GeForce 6800 GT]
0x0218	NV40? [GeForce 6800 XT]

Todo

wtf is with that 0x21x ID?

NV41/NV42

device id	product
0x00c0	NV41/NV42 [GeForce 6800 GS]
0x00c1	NV41/NV42 [GeForce 6800]
0x00c2	NV41/NV42 [GeForce 6800 LE]
0x00c3	NV41/NV42 [GeForce 6800 XT]
0x00c8	NV41/NV42 [GeForce Go 6800]
0x00c9	NV41/NV42 [GeForce Go 6800 Ultra]
0x00cc	NV41/NV42 [Quadro FX Go1400]
0x00cd	NV41/NV42 [Quadro FX 3450/4000 SDI]
0x00ce	NV41/NV42 [Quadro FX 1400]

NV43

device id	product
0x0140	NV43 [GeForce 6600 GT]
0x0141	NV43 [GeForce 6600]
0x0142	NV43 [GeForce 6600 LE]
0x0143	NV43 [GeForce 6600 VE]
0x0144	NV43 [GeForce Go 6600]
0x0145	NV43 [GeForce 6610 XL]
0x0146	NV43 [GeForce Go 6200 TE / 6660 TE]
0x0147	NV43 [GeForce 6700 XL]
0x0148	NV43 [GeForce Go 6600]
0x0149	NV43 [GeForce Go 6600 GT]
0x014a	NV43 [Quadro NVS 440]
0x014c	NV43 [Quadro FX 540M]
0x014d	NV43 [Quadro FX 550]
0x014e	NV43 [Quadro FX 540]
0x014f	NV43 [GeForce 6200]

NV44

device id	product
0x0160	NV44 [GeForce 6500]
0x0161	NV44 [GeForce 6200 TurboCache]
0x0162	NV44 [GeForce 6200 SE TurboCache]
0x0163	NV44 [GeForce 6200 LE]
0x0164	NV44 [GeForce Go 6200]
0x0165	NV44 [Quadro NVS 285]
0x0166	NV44 [GeForce Go 6400]
0x0167	NV44 [GeForce Go 6200]
0x0168	NV44 [GeForce Go 6400]
0x0169	NV44 [GeForce 6250]
0x016a	NV44 [GeForce 7100 GS]

NV44A

device id	product
0x0221	NV44A [GeForce 6200 (AGP)]
0x0222	NV44A [GeForce 6200 A-LE (AGP)]

C51 GPU

device id	product
0x0240	C51 GPU [GeForce 6150]
0x0241	C51 GPU [GeForce 6150 LE]
0x0242	C51 GPU [GeForce 6100]
0x0244	C51 GPU [GeForce Go 6150]
0x0245	C51 GPU [Quadro NVS 210S / NVIDIA GeForce 6150LE]
0x0247	C51 GPU [GeForce Go 6100]

G70

device id	product
0x0090	G70 [GeForce 7800 GTX]
0x0091	G70 [GeForce 7800 GTX]
0x0092	G70 [GeForce 7800 GT]
0x0093	G70 [GeForce 7800 GS]
0x0095	G70 [GeForce 7800 SLI]
0x0098	G70 [GeForce Go 7800]
0x0099	G70 [GeForce Go 7800 GTX]
0x009d	G70 [Quadro FX 4500]

G72

device id	product
0x01d0	G72 [GeForce 7350 LE]
0x01d1	G72 [GeForce 7300 LE]
0x01d2	G72 [GeForce 7550 LE]
0x01d3	G72 [GeForce 7300 SE/7200 GS]
0x01d6	G72 [GeForce Go 7200]
0x01d7	G72 [Quadro NVS 110M / GeForce Go 7300]
0x01d8	G72 [GeForce Go 7400]
0x01d9	G72 [GeForce Go 7450]
0x01da	G72 [Quadro NVS 110M]
0x01db	G72 [Quadro NVS 120M]
0x01dc	G72 [Quadro FX 350M]
0x01dd	G72 [GeForce 7500 LE]
0x01de	G72 [Quadro FX 350]
0x01df	G72 [GeForce 7300 GS]

G71

device id	product
0x0290	G71 [GeForce 7900 GTX]
0x0291	G71 [GeForce 7900 GT/GTO]
0x0292	G71 [GeForce 7900 GS]
0x0293	G71 [GeForce 7900 GX2]
0x0294	G71 [GeForce 7950 GX2]
0x0295	G71 [GeForce 7950 GT]
0x0297	G71 [GeForce Go 7950 GTX]
0x0298	G71 [GeForce Go 7900 GS]
0x0299	G71 [GeForce Go 7900 GTX]
0x029a	G71 [Quadro FX 2500M]
0x029b	G71 [Quadro FX 1500M]
0x029c	G71 [Quadro FX 5500]
0x029d	G71 [Quadro FX 3500]
0x029e	G71 [Quadro FX 1500]
0x029f	G71 [Quadro FX 4500 X2]

G73

device id	product
0x0390	G73 [GeForce 7650 GS]
0x0391	G73 [GeForce 7600 GT]
0x0392	G73 [GeForce 7600 GS]
0x0393	G73 [GeForce 7300 GT]
0x0394	G73 [GeForce 7600 LE]
0x0395	G73 [GeForce 7300 GT]
0x0397	G73 [GeForce Go 7700]
0x0398	G73 [GeForce Go 7600]
0x0399	G73 [GeForce Go 7600 GT]
0x039a	G73 [Quadro NVS 300M]
0x039b	G73 [GeForce Go 7900 SE]
0x039c	G73 [Quadro FX 560M]
0x039e	G73 [Quadro FX 560]

MCP61 GPU

device id	product
0x03d0	MCP61 GPU [GeForce 6150SE nForce 430]
0x03d1	MCP61 GPU [GeForce 6100 nForce 405]
0x03d2	MCP61 GPU [GeForce 6100 nForce 400]
0x03d5	MCP61 GPU [GeForce 6100 nForce 420]
0x03d6	MCP61 GPU [GeForce 7025 / nForce 630a]

MCP67 GPU

device id	product
0x0531	MCP67 GPU [GeForce 7150M / nForce 630M]
0x0533	MCP67 GPU [GeForce 7000M / nForce 610M]
0x053a	MCP67 GPU [GeForce 7050 PV / nForce 630a]
0x053b	MCP67 GPU [GeForce 7050 PV / nForce 630a]
0x053e	MCP67 GPU [GeForce 7025 / nForce 630a]

Note: mobile is apparently considered to be MCP67, desktop MCP68

MCP73 GPU

device id	product
0x07e0	MCP73 GPU [GeForce 7150 / nForce 630i]
0x07e1	MCP73 GPU [GeForce 7100 / nForce 630i]
0x07e2	MCP73 GPU [GeForce 7050 / nForce 630i]
0x07e3	MCP73 GPU [GeForce 7050 / nForce 610i]
0x07e5	MCP73 GPU [GeForce 7050 / nForce 620i]

G80

device id	product
0x0191	G80 [GeForce 8800 GTX]
0x0193	G80 [GeForce 8800 GTS]
0x0194	G80 [GeForce 8800 Ultra]
0x0197	G80 [Tesla C870]
0x019d	G80 [Quadro FX 5600]
0x019e	G80 [Quadro FX 4600]

G84

device id	product
0x0400	G84 [GeForce 8600 GTS]
0x0401	G84 [GeForce 8600 GT]
0x0402	G84 [GeForce 8600 GT]
0x0403	G84 [GeForce 8600 GS]
0x0404	G84 [GeForce 8400 GS]
0x0405	G84 [GeForce 9500M GS]
0x0406	G84 [GeForce 8300 GS]
0x0407	G84 [GeForce 8600M GT]
0x0408	G84 [GeForce 9650M GS]
0x0409	G84 [GeForce 8700M GT]
0x040a	G84 [Quadro FX 370]
0x040b	G84 [Quadro NVS 320M]
0x040c	G84 [Quadro FX 570M]
0x040d	G84 [Quadro FX 1600M]
0x040e	G84 [Quadro FX 570]
0x040f	G84 [Quadro FX 1700]

G86

device id	product
0x0420	G86 [GeForce 8400 SE]
0x0421	G86 [GeForce 8500 GT]
0x0422	G86 [GeForce 8400 GS]
0x0423	G86 [GeForce 8300 GS]
0x0424	G86 [GeForce 8400 GS]
0x0425	G86 [GeForce 8600M GS]
0x0426	G86 [GeForce 8400M GT]
0x0427	G86 [GeForce 8400M GS]
0x0428	G86 [GeForce 8400M G]
0x0429	G86 [Quadro NVS 140M]
0x042a	G86 [Quadro NVS 130M]
0x042b	G86 [Quadro NVS 135M]
0x042c	G86 [GeForce 9400 GT]
0x042d	G86 [Quadro FX 360M]
0x042e	G86 [GeForce 9300M G]
0x042f	G86 [Quadro NVS 290]

G92

device id	product
0x0410	G92 [GeForce GT 330]
0x0600	G92 [GeForce 8800 GTS 512]
0x0601	G92 [GeForce 9800 GT]
0x0602	G92 [GeForce 8800 GT]
0x0603	G92 [GeForce GT 230]
0x0604	G92 [GeForce 9800 GX2]
0x0605	G92 [GeForce 9800 GT]
0x0606	G92 [GeForce 8800 GS]
0x0607	G92 [GeForce GTS 240]
0x0608	G92 [GeForce 9800M GTX]
0x0609	G92 [GeForce 8800M GTS]
0x060a	G92 [GeForce GTX 280M]
0x060b	G92 [GeForce 9800M GT]
0x060c	G92 [GeForce 8800M GTX]
0x060f	G92 [GeForce GTX 285M]
0x0610	G92 [GeForce 9600 GSO]
0x0611	G92 [GeForce 8800 GT]
0x0612	G92 [GeForce 9800 GTX/9800 GTX+]
0x0613	G92 [GeForce 9800 GTX+]
0x0614	G92 [GeForce 9800 GT]
0x0615	G92 [GeForce GTS 250]
0x0617	G92 [GeForce 9800M GTX]
0x0618	G92 [GeForce GTX 260M]
0x0619	G92 [Quadro FX 4700 X2]
0x061a	G92 [Quadro FX 3700]
0x061b	G92 [Quadro VX 200]
0x061c	G92 [Quadro FX 3600M]

Continued on next page

Table 2.2 – continued from previous page

device id	product
0x061d	G92 [Quadro FX 2800M]
0x061e	G92 [Quadro FX 3700M]
0x061f	G92 [Quadro FX 3800M]

G94

device id	product
0x0621	G94 [GeForce GT 230]
0x0622	G94 [GeForce 9600 GT]
0x0623	G94 [GeForce 9600 GS]
0x0625	G94 [GeForce 9600 GSO 512]
0x0626	G94 [GeForce GT 130]
0x0627	G94 [GeForce GT 140]
0x0628	G94 [GeForce 9800M GTS]
0x062a	G94 [GeForce 9700M GTS]
0x062b	G94 [GeForce 9800M GS]
0x062c	G94 [GeForce 9800M GTS]
0x062d	G94 [GeForce 9600 GT]
0x062e	G94 [GeForce 9600 GT]
0x0631	G94 [GeForce GTS 160M]
0x0635	G94 [GeForce 9600 GSO]
0x0637	G94 [GeForce 9600 GT]
0x0638	G94 [Quadro FX 1800]
0x063a	G94 [Quadro FX 2700M]

G96

device id	product
0x0640	G96 [GeForce 9500 GT]
0x0641	G96 [GeForce 9400 GT]
0x0643	G96 [GeForce 9500 GT]
0x0644	G96 [GeForce 9500 GS]
0x0645	G96 [GeForce 9500 GS]
0x0646	G96 [GeForce GT 120]
0x0647	G96 [GeForce 9600M GT]
0x0648	G96 [GeForce 9600M GS]
0x0649	G96 [GeForce 9600M GT]
0x064a	G96 [GeForce 9700M GT]
0x064b	G96 [GeForce 9500M G]
0x064c	G96 [GeForce 9650M GT]
0x0651	G96 [GeForce G 110M]
0x0652	G96 [GeForce GT 130M]
0x0653	G96 [GeForce GT 120M]
0x0654	G96 [GeForce GT 220M]
0x0655	G96 [GeForce GT 120]
0x0656	G96 [GeForce GT 120]
0x0658	G96 [Quadro FX 380]
0x0659	G96 [Quadro FX 580]
0x065a	G96 [Quadro FX 1700M]
0x065b	G96 [GeForce 9400 GT]
0x065c	G96 [Quadro FX 770M]
0x065f	G96 [GeForce G210]

G98

device id	product
0x06e0	G98 [GeForce 9300 GE]
0x06e1	G98 [GeForce 9300 GS]
0x06e2	G98 [GeForce 8400]
0x06e3	G98 [GeForce 8400 SE]
0x06e4	G98 [GeForce 8400 GS]
0x06e6	G98 [GeForce G100]
0x06e7	G98 [GeForce 9300 SE]
0x06e8	G98 [GeForce 9200M GS]
0x06e9	G98 [GeForce 9300M GS]
0x06ea	G98 [Quadro NVS 150M]
0x06eb	G98 [Quadro NVS 160M]
0x06ec	G98 [GeForce G 105M]
0x06ef	G98 [GeForce G 103M]
0x06f1	G98 [GeForce G105M]
0x06f8	G98 [Quadro NVS 420]
0x06f9	G98 [Quadro FX 370 LP]
0x06fa	G98 [Quadro NVS 450]
0x06fb	G98 [Quadro FX 370M]
0x06fd	G98 [Quadro NVS 295]
0x06ff	G98 [HICx16 + Graphics]

G200

device id	product
0x05e0	G200 [GeForce GTX 295]
0x05e1	G200 [GeForce GTX 280]
0x05e2	G200 [GeForce GTX 260]
0x05e3	G200 [GeForce GTX 285]
0x05e6	G200 [GeForce GTX 275]
0x05e7	G200 [Tesla C1060]
0x05e9	G200 [Quadro CX]
0x05ea	G200 [GeForce GTX 260]
0x05eb	G200 [GeForce GTX 295]
0x05ed	G200 [Quadro FX 5800]
0x05ee	G200 [Quadro FX 4800]
0x05ef	G200 [Quadro FX 3800]

MCP77 GPU

device id	product
0x0840	MCP77 GPU [GeForce 8200M]
0x0844	MCP77 GPU [GeForce 9100M G]
0x0845	MCP77 GPU [GeForce 8200M G]
0x0846	MCP77 GPU [GeForce 9200]
0x0847	MCP77 GPU [GeForce 9100]
0x0848	MCP77 GPU [GeForce 8300]
0x0849	MCP77 GPU [GeForce 8200]
0x084a	MCP77 GPU [nForce 730a]
0x084b	MCP77 GPU [GeForce 9200]
0x084c	MCP77 GPU [nForce 980a/780a SLI]
0x084d	MCP77 GPU [nForce 750a SLI]
0x084f	MCP77 GPU [GeForce 8100 / nForce 720a]

MCP79 GPU

device id	product
0x0860	MCP79 GPU [GeForce 9400]
0x0861	MCP79 GPU [GeForce 9400]
0x0862	MCP79 GPU [GeForce 9400M G]
0x0863	MCP79 GPU [GeForce 9400M]
0x0864	MCP79 GPU [GeForce 9300]
0x0865	MCP79 GPU [ION]
0x0866	MCP79 GPU [GeForce 9400M G]
0x0867	MCP79 GPU [GeForce 9400]
0x0868	MCP79 GPU [nForce 760i SLI]
0x0869	MCP79 GPU [GeForce 9400]
0x086a	MCP79 GPU [GeForce 9400]
0x086c	MCP79 GPU [GeForce 9300 / nForce 730i]
0x086d	MCP79 GPU [GeForce 9200]
0x086e	MCP79 GPU [GeForce 9100M G]
0x086f	MCP79 GPU [GeForce 8200M G]
0x0870	MCP79 GPU [GeForce 9400M]
0x0871	MCP79 GPU [GeForce 9200]
0x0872	MCP79 GPU [GeForce G102M]
0x0873	MCP79 GPU [GeForce G102M]
0x0874	MCP79 GPU [ION]
0x0876	MCP79 GPU [ION]
0x087a	MCP79 GPU [GeForce 9400]
0x087d	MCP79 GPU [ION]
0x087e	MCP79 GPU [ION LE]
0x087f	MCP79 GPU [ION LE]

GT215

device id	product
0x0ca0	GT215 [GeForce GT 330]
0x0ca2	GT215 [GeForce GT 320]
0x0ca3	GT215 [GeForce GT 240]
0x0ca4	GT215 [GeForce GT 340]
0x0ca5	GT215 [GeForce GT 220]
0x0ca7	GT215 [GeForce GT 330]
0x0ca9	GT215 [GeForce GTS 250M]
0x0cac	GT215 [GeForce GT 220]
0x0caf	GT215 [GeForce GT 335M]
0x0cb0	GT215 [GeForce GTS 350M]
0x0cb1	GT215 [GeForce GTS 360M]
0x0cbc	GT215 [Quadro FX 1800M]

GT216

device id	product
0x0a20	GT216 [GeForce GT 220]
0x0a22	GT216 [GeForce 315]
0x0a23	GT216 [GeForce 210]
0x0a26	GT216 [GeForce 405]
0x0a27	GT216 [GeForce 405]
0x0a28	GT216 [GeForce GT 230M]
0x0a29	GT216 [GeForce GT 330M]
0x0a2a	GT216 [GeForce GT 230M]
0x0a2b	GT216 [GeForce GT 330M]
0x0a2c	GT216 [NVS 5100M]
0x0a2d	GT216 [GeForce GT 320M]
0x0a32	GT216 [GeForce GT 415]
0x0a34	GT216 [GeForce GT 240M]
0x0a35	GT216 [GeForce GT 325M]
0x0a38	GT216 [Quadro 400]
0x0a3c	GT216 [Quadro FX 880M]

GT218

device id	product
0x0a60	GT218 [GeForce G210]
0x0a62	GT218 [GeForce 205]
0x0a63	GT218 [GeForce 310]
0x0a64	GT218 [ION]
0x0a65	GT218 [GeForce 210]
0x0a66	GT218 [GeForce 310]
0x0a67	GT218 [GeForce 315]
0x0a68	GT218 [GeForce G105M]
0x0a69	GT218 [GeForce G105M]
0x0a6a	GT218 [NVS 2100M]
0x0a6c	GT218 [NVS 3100M]
0x0a6e	GT218 [GeForce 305M]
0x0a6f	GT218 [ION]
0x0a70	GT218 [GeForce 310M]
0x0a71	GT218 [GeForce 305M]
0x0a72	GT218 [GeForce 310M]
0x0a73	GT218 [GeForce 305M]
0x0a74	GT218 [GeForce G210M]
0x0a75	GT218 [GeForce 310M]
0x0a76	GT218 [ION]
0x0a78	GT218 [Quadro FX 380 LP]
0x0a7a	GT218 [GeForce 315M]
0x0a7c	GT218 [Quadro FX 380M]
0x10c0	GT218 [GeForce 9300 GS]
0x10c3	GT218 [GeForce 8400GS]
0x10c5	GT218 [GeForce 405]
0x10d8	GT218 [NVS 300]

MCP89 GPU

device id	product
0x08a0	MCP89 GPU [GeForce 320M]
0x08a2	MCP89 GPU [GeForce 320M]
0x08a3	MCP89 GPU [GeForce 320M]
0x08a4	MCP89 GPU [GeForce 320M]

GF100

device id	product
0x06c0	GF100 [GeForce GTX 480]
0x06c4	GF100 [GeForce GTX 465]
0x06ca	GF100 [GeForce GTX 480M]
0x06cb	GF100 [GeForce GTX 480]
0x06cd	GF100 [GeForce GTX 470]
0x06d1	GF100 [Tesla C2050 / C2070]
0x06d2	GF100 [Tesla M2070]
0x06d8	GF100 [Quadro 6000]
0x06d9	GF100 [Quadro 5000]
0x06da	GF100 [Quadro 5000M]
0x06dc	GF100 [Quadro 6000]
0x06dd	GF100 [Quadro 4000]
0x06de	GF100 [Tesla T20 Processor]
0x06df	GF100 [Tesla M2070-Q]

GF104

device id	product
0x0e22	GF104 [GeForce GTX 460]
0x0e23	GF104 [GeForce GTX 460 SE]
0x0e24	GF104 [GeForce GTX 460 OEM]
0x0e30	GF104 [GeForce GTX 470M]
0x0e31	GF104 [GeForce GTX 485M]
0x0e3a	GF104 [Quadro 3000M]
0x0e3b	GF104 [Quadro 4000M]

GF114

device id	product
0x1200	GF114 [GeForce GTX 560 Ti]
0x1201	GF114 [GeForce GTX 560]
0x1202	GF114 [GeForce GTX 560 Ti OEM]
0x1203	GF114 [GeForce GTX 460 SE v2]
0x1205	GF114 [GeForce GTX 460 v2]
0x1206	GF114 [GeForce GTX 555]
0x1207	GF114 [GeForce GT 645 OEM]
0x1208	GF114 [GeForce GTX 560 SE]
0x1210	GF114 [GeForce GTX 570M]
0x1211	GF114 [GeForce GTX 580M]
0x1212	GF114 [GeForce GTX 675M]
0x1213	GF114 [GeForce GTX 670M]

GF106

device id	product
0x0dc0	GF106 [GeForce GT 440]
0x0dc4	GF106 [GeForce GTS 450]
0x0dc5	GF106 [GeForce GTS 450]
0x0dc6	GF106 [GeForce GTS 450]
0x0dcd	GF106 [GeForce GT 555M]
0x0dce	GF106 [GeForce GT 555M]
0x0dd1	GF106 [GeForce GTX 460M]
0x0dd2	GF106 [GeForce GT 445M]
0x0dd3	GF106 [GeForce GT 435M]
0x0dd6	GF106 [GeForce GT 550M]
0x0dd8	GF106 [Quadro 2000]
0x0dda	GF106 [Quadro 2000M]

GF116

device id	product
0x1241	GF116 [GeForce GT 545 OEM]
0x1243	GF116 [GeForce GT 545]
0x1244	GF116 [GeForce GTX 550 Ti]
0x1245	GF116 [GeForce GTS 450 Rev. 2]
0x1246	GF116 [GeForce GT 550M]
0x1247	GF116 [GeForce GT 635M]
0x1248	GF116 [GeForce GT 555M]
0x1249	GF116 [GeForce GTS 450 Rev. 3]
0x124b	GF116 [GeForce GT 640 OEM]
0x124d	GF116 [GeForce GT 555M]
0x1251	GF116 [GeForce GTX 560M]

GF108

device id	product
0x0de0	GF108 [GeForce GT 440]
0x0de1	GF108 [GeForce GT 430]
0x0de2	GF108 [GeForce GT 420]
0x0de3	GF108 [GeForce GT 635M]
0x0de4	GF108 [GeForce GT 520]
0x0de5	GF108 [GeForce GT 530]
0x0de8	GF108 [GeForce GT 620M]
0x0de9	GF108 [GeForce GT 630M]
0x0dea	GF108 [GeForce 610M]
0x0deb	GF108 [GeForce GT 555M]
0x0dec	GF108 [GeForce GT 525M]
0x0ded	GF108 [GeForce GT 520M]
0x0dee	GF108 [GeForce GT 415M]
0x0def	GF108 [NVS 5400M]
0x0df0	GF108 [GeForce GT 425M]
0x0df1	GF108 [GeForce GT 420M]
0x0df2	GF108 [GeForce GT 435M]
0x0df3	GF108 [GeForce GT 420M]
0x0df4	GF108 [GeForce GT 540M]
0x0df5	GF108 [GeForce GT 525M]
0x0df6	GF108 [GeForce GT 550M]
0x0df7	GF108 [GeForce GT 520M]
0x0df8	GF108 [Quadro 600]
0x0df9	GF108 [Quadro 500M]
0x0dfa	GF108 [Quadro 1000M]
0x0dfc	GF108 [NVS 5200M]
0x0f00	GF108 [GeForce GT 630]
0x0f01	GF108 [GeForce GT 620]

GF110

device id	product
0x1080	GF110 [GeForce GTX 580]
0x1081	GF110 [GeForce GTX 570]
0x1082	GF110 [GeForce GTX 560 Ti]
0x1084	GF110 [GeForce GTX 560]
0x1086	GF110 [GeForce GTX 570]
0x1087	GF110 [GeForce GTX 560 Ti]
0x1088	GF110 [GeForce GTX 590]
0x1089	GF110 [GeForce GTX 580]
0x108b	GF110 [GeForce GTX 580]
0x1091	GF110 [Tesla M2090]
0x109a	GF110 [Quadro 5010M]
0x109b	GF110 [Quadro 7000]

GF119

device id	product
0x1040	GF119 [GeForce GT 520]
0x1042	GF119 [GeForce 510]
0x1048	GF119 [GeForce 605]
0x1049	GF119 [GeForce GT 620]
0x104a	GF119 [GeForce GT 610]
0x1050	GF119 [GeForce GT 520M]
0x1051	GF119 [GeForce GT 520MX]
0x1052	GF119 [GeForce GT 520M]
0x1054	GF119 [GeForce 410M]
0x1055	GF119 [GeForce 410M]
0x1056	GF119 [NVS 4200M]
0x1057	GF119 [NVS 4200M]
0x1058	GF119 [GeForce 610M]
0x1059	GF119 [GeForce 610M]
0x105a	GF119 [GeForce 610M]
0x107d	GF119 [NVS 310]

GF117

device id	product
0x1140	GF117 [GeForce GT 620M]

GK104

device id	product
0x1180	GK104 [GeForce GTX 680]
0x1183	GK104 [GeForce GTX 660 Ti]
0x1185	GK104 [GeForce GTX 660]
0x1188	GK104 [GeForce GTX 690]
0x1189	GK104 [GeForce GTX 670]
0x11a0	GK104 [GeForce GTX 680M]
0x11a1	GK104 [GeForce GTX 670MX]
0x11a2	GK104 [GeForce GTX 675MX]
0x11a3	GK104 [GeForce GTX 680MX]
0x11a7	GK104 [GeForce GTX 675MX]
0x11ba	GK104 [Quadro K5000]
0x11bc	GK104 [Quadro K5000M]
0x11bd	GK104 [Quadro K4000M]
0x11be	GK104 [Quadro K3000M]
0x11bf	GK104 [GRID K2]

GK106

device id	product
0x11c0	GK106 [GeForce GTX 660]
0x11c6	GK106 [GeForce GTX 650 Ti]
0x11fa	GK106 [Quadro K4000]

GK107

device id	product
0x0fc0	GK107 [GeForce GT 640]
0x0fc1	GK107 [GeForce GT 640]
0x0fc2	GK107 [GeForce GT 630]
0x0fc6	GK107 [GeForce GTX 650]
0x0fd1	GK107 [GeForce GT 650M]
0x0fd2	GK107 [GeForce GT 640M]
0x0fd3	GK107 [GeForce GT 640M LE]
0x0fd4	GK107 [GeForce GTX 660M]
0x0fd5	GK107 [GeForce GT 650M]
0x0fd8	GK107 [GeForce GT 640M]
0x0fd9	GK107 [GeForce GT 645M]
0x0fe0	GK107 [GeForce GTX 660M]
0x0ff9	GK107 [Quadro K2000D]
0x0ffa	GK107 [Quadro K600]
0x0ffb	GK107 [Quadro K2000M]
0x0ffc	GK107 [Quadro K1000M]
0x0ffd	GK107 [NVS 510]
0x0ffe	GK107 [Quadro K2000]
0x0fff	GK107 [Quadro 410]

GK110/GK110B

device id	product
0x1003	GK110 [GeForce GTX Titan LE]
0x1004	GK110 [GeForce GTX 780]
0x1005	GK110 [GeForce GTX Titan]
0x101f	GK110 [Tesla K20]
0x1020	GK110 [Tesla K20X]
0x1021	GK110 [Tesla K20Xm]
0x1022	GK110 [Tesla K20c]
0x1026	GK110 [Tesla K20s]
0x1028	GK110 [Tesla K20m]

GK208

device id	product
0x1280	GK208 [GeForce GT 635]
0x1282	GK208 [GeForce GT 640 Rev. 2]
0x1284	GK208 [GeForce GT 630 Rev. 2]
0x1290	GK208 [GeForce GT 730M]
0x1291	GK208 [GeForce GT 735M]
0x1292	GK208 [GeForce GT 740M]
0x1293	GK208 [GeForce GT 730M]
0x1294	GK208 [GeForce GT 740M]
0x1295	GK208 [GeForce 710M]
0x12b9	GK208 [Quadro K610M]
0x12ba	GK208 [Quadro K510M]

GM107

device id	product
0x1381	GM107 [GeForce GTX 750]

GM108

device id	product
0x1340	GM108
0x1341	GM108

GM204

device id	product
0x13c0	GM204 [GeForce GTX 980]
0x13c2	GM204 [GeForce GTX 970]

GM206

device id	product
0x1401	GM206 [GeForce GTX 960]

2.3.3 GPU HDA codecs

device id	product
0x0be2	GT216 HDA
0x0be3	GT218 HDA
0x0be4	GT215 HDA
0x0be5	GF100 HDA
0x0be9	GF106 HDA
0x0bea	GF108 HDA
0x0beb	GF104 HDA
0x0bee	GF116 HDA
0x0e08	GF119 HDA
0x0e09	GF110 HDA
0x0e0a	GK104 HDA
0x0e0b	GK106 HDA
0x0e0c	GF114 HDA
0x0e0f	GK208 HDA
0x0e1a	GK110 HDA
0x0e1b	GK107 HDA
0x0fbc	GM107 HDA

2.3.4 BR02

The BR02 aka HSI is a transparent PCI-Express - AGP bridge. It can be used to connect PCIE GPU to AGP bus, or the other way around. Its PCI device id shadows the actual GPU's device id.

device id	product
0x00f1	BR02+NV43 [GeForce 6600 GT]
0x00f2	BR02+NV43 [GeForce 6600]
0x00f3	BR02+NV43 [GeForce 6200]
0x00f4	BR02+NV43 [GeForce 6600 LE]
0x00f5	BR02+G71 [GeForce 7800 GS]
0x00f6	BR02+NV43 [GeForce 6800 GS/XT]
0x00f8	BR02+NV40 [Quadro FX 3400/4400]
0x00f9	BR02+NV40 [GeForce 6800 Series GPU]
0x00fa	BR02+NV36 [GeForce PCX 5750]
0x00fb	BR02+NV35 [GeForce PCX 5900]
0x00fc	BR02+NV34 [GeForce PCX 5300 / Quadro FX 330]
0x00fd	BR02+NV34 [Quadro FX 330]
0x00fe	BR02+NV35 [Quadro FX 1300]
0x00ff	BR02+NV18 [GeForce PCX 4300]
0x02e0	BR02+G73 [GeForce 7600 GT]
0x02e1	BR02+G73 [GeForce 7600 GS]
0x02e2	BR02+G73 [GeForce 7300 GT]
0x02e3	BR02+G71 [GeForce 7900 GS]
0x02e4	BR02+G71 [GeForce 7950 GT]

2.3.5 BR03

The BR03 aka NF100 is a PCI-Express switch with 2 downstream 16x ports. It's used on NV40 generation dual-GPU cards.

device id	product
0x01b3	BR03 [GeForce 7900 GX2/7950 GX2]

2.3.6 BR04

The BR04 aka NF200 is a PCI-Express switch with 4 downstream 16x ports. It's used on Tesla and Fermi generation dual-GPU cards, as well as some SLI-capable motherboards.

device id	product
0x05b1	BR04 [motherboard]
0x05b8	BR04 [GeForce GTX 295]
0x05b9	BR04 [GeForce GTX 590]
0x05be	BR04 [GeForce 9800 GX2/Quadro Plex S4/Tesla S*]

2.3.7 Motherboard chipsets

NV1A [nForce 220 IGP / 420 IGP / 415 SPP]

The northbridge of nForce1 chipset, paired with *MCP*.

device id	product
0x01a0	NV1A GPU [GeForce2 MX IGP]
0x01a4	NV1A host bridge
0x01a5	NV1A host bridge [?]
0x01a6	NV1A host bridge [?]
0x01a8	NV1A memory controller [?]
0x01a9	NV1A memory controller [?]
0x01aa	NV1A memory controller #3, 64-bit
0x01ab	NV1A memory controller #3, 128-bit
0x01ac	NV1A memory controller #1
0x01ad	NV1A memory controller #2
0x01b7	NV1A/NV2A AGP bridge

Note: 0x01b7 is also used on *NV2A*.

NV2A [XGPU]

The northbridge of xbox, paired with *MCP*.

device id	product
0x02a0	NV2A GPU
0x02a5	NV2A host bridge
0x02a6	NV2A memory controller
0x01b7	NV1A/NV2A AGP bridge

Note: 0x01b7 is also used on *NV1A*.

MCP

The southbridge of nForce1 chipset and xbox, paired with *NV1A* or *NV2A*.

device id	product
0x01b0	MCP APU
0x01b1	MCP AC‘97
0x01b2	MCP LPC bridge
0x01b4	MCP SMBus controller
0x01b8	MCP PCI bridge
0x01bc	MCP IDE controller
0x01c1	MCP MC‘97
0x01c2	MCP USB controller
0x01c3	MCP ethernet controller

NV1F [nForce2 IGP/SPP]

The northbridge of nForce2 chipset, paired with *MCP2* or *MCP2A*.

device id	product
0x01e0	NV1F host bridge
0x01e8	NV1F AGP bridge
0x01ea	NV1F memory controller #1
0x01eb	NV1F memory controller #1
0x01ec	NV1F memory controller #4
0x01ed	NV1F memory controller #3
0x01ee	NV1F memory controller #2
0x01ef	NV1F memory controller #5

MCP2

The southbridge of nForce2 chipset, original revision. Paired with *NV1F*.

device id	product
0x0060	MCP2 LPC bridge
0x0064	MCP2 SMBus controller
0x0065	MCP2 IDE controller
0x0066	MCP2 ethernet controller
0x0067	MCP2 USB controller
0x0068	MCP2 USB 2.0 controller
0x0069	MCP2 MC‘97
0x006a	MCP2 AC‘97
0x006b	MCP2 APU
0x006c	MCP2 PCI bridge
0x006d	MCP2 internal PCI bridge for 3com ethernet
0x006e	MCP2 Firewire controller

MCP2A

The southbridge of nForce2 400 chipset. Paired with *NV1F*.

device id	product
0x0080	MCP2A LPC bridge
0x0084	MCP2A SMBus controller
0x0085	MCP2A IDE controller
0x0086	MCP2A ethernet controller (class 0200)
0x0087	MCP2A USB controller
0x0088	MCP2A USB 2.0 controller
0x0089	MCP2A MC‘97
0x008a	MCP2A AC‘97
0x008b	MCP2A PCI bridge
0x008c	MCP2A ethernet controller (class 0680)
0x008e	MCP2A SATA controller

CK8

The nforce3-150 chipset.

device id	product
0x00d0	CK8 LPC bridge
0x00d1	CK8 host bridge
0x00d2	CK8 AGP bridge
0x00d4	CK8 SMBus controller
0x00d5	CK8 IDE controller
0x00d6	CK8 ethernet controller
0x00d7	CK8 USB controller
0x00d8	CK8 USB 2.0 controller
0x00d9	CK8 MC‘97
0x00da	CK8 AC‘97
0x00dd	CK8 PCI bridge

CK8S

The nforce3-250 chipset.

device id	product
0x00df	CK8S ethernet controller (class 0680)
0x00e0	CK8S LPC bridge
0x00e1	CK8S host bridge
0x00e2	CK8S AGP bridge
0x00e3	CK8S SATA controller #1
0x00e4	CK8S SMBus controller
0x00e5	CK8S IDE controller
0x00e6	CK8S ethernet controller (class 0200)
0x00e7	CK8S USB controller
0x00e8	CK8S USB 2.0 controller
0x00e9	CK8S MC‘97
0x00ea	CK8S AC‘97
0x00ec	CK8S ???? (class 0780)
0x00ed	CK8S PCI bridge
0x00ee	CK8S SATA controller #0

CK804

The AMD nforce4 chipset, standalone or paired with C19 or C51 to make nforce4 SLI x16 chipset.

device id	product
0x0050	CK804 LPC bridge
0x0051	CK804 LPC bridge
0x0052	CK804 SMBus controller
0x0053	CK804 IDE controller
0x0054	CK804 SATA controller #0
0x0055	CK804 SATA controller #1
0x0056	CK804 ethernet controller (class 0200)
0x0057	CK804 ethernet controller (class 0680)
0x0058	CK804 MC'97
0x0059	CK804 AC'97
0x005a	CK804 USB controller
0x005b	CK804 USB 2.0 controller
0x005c	CK804 PCI subtractive bridge
0x005d	CK804 PCI-Express port
0x005e	CK804 memory controller #0
0x005f	CK804 memory controller #12
0x00d3	CK804 memory controller #10

C19

The intel nforce4 northbridge, paired with MCP04 or CK804.

device id	product
0x006f	C19 memory controller #3
0x0070	C19 host bridge
0x0071	C19 host bridge
0x0072	C19 host bridge [?]
0x0073	C19 host bridge [?]
0x0074	C19 memory controller #1
0x0075	C19 memory controller #2
0x0076	C19 memory controller #10
0x0078	C19 memory controller #11
0x0079	C19 memory controller #12
0x007a	C19 memory controller #13
0x007b	C19 memory controller #14
0x007c	C19 memory controller #15
0x007d	C19 memory controller #16
0x007e	C19 PCI-Express port
0x007f	C19 memory controller #1
0x00b4	C19 memory controller #4

MCP04

The intel nforce4 southbridge, paired with C19.

device id	product
0x0030	MCP04 LPC bridge
0x0034	MCP04 SMBus controller
0x0035	MCP04 IDE controller
0x0036	MCP04 SATA controller #0
0x0037	MCP04 ethernet controller (class 0200)
0x0038	MCP04 ethernet controller (class 0680)
0x0039	MCP04 MC'97
0x003a	MCP04 AC'97
0x003b	MCP04 USB controller
0x003c	MCP04 USB 2.0 controller
0x003d	MCP04 PCI subtractive bridge
0x003e	MCP04 SATA controller #1
0x003f	MCP04 memory controller

C51

The AMD nforce4xx/nforce5xx northbridge, paired with CK804, MCP51, or MCP55.

device id	product
0x02f0	C51 memory controller #0
0x02f1	C51 memory controller #0
0x02f2	C51 memory controller #0
0x02f3	C51 memory controller #0
0x02f4	C51 memory controller #0
0x02f5	C51 memory controller #0
0x02f6	C51 memory controller #0
0x02f7	C51 memory controller #0
0x02f8	C51 memory controller #3
0x02f9	C51 memory controller #4
0x02fa	C51 memory controller #1
0x02fb	C51 PCI-Express x16 port
0x02fc	C51 PCI-Express x1 port #0
0x02fd	C51 PCI-Express x1 port #1
0x02fe	C51 memory controller #2
0x02ff	C51 memory controller #5
0x027e	C51 memory controller #7
0x027f	C51 memory controller #6

MCP51

The AMD nforce5xx southbridge, paired with C51 or C55.

device id	product
0x0260	MCP51 LPC bridge
0x0261	MCP51 LPC bridge
0x0262	MCP51 LPC bridge [?]
0x0263	MCP51 LPC bridge [?]
0x0264	MCP51 SMBus controller
0x0265	MCP51 IDE controller
0x0266	MCP51 SATA controller #0
0x0267	MCP51 SATA controller #1
0x0268	MCP51 ethernet controller (class 0200)
0x0269	MCP51 ethernet controller (class 0680)
0x026a	MCP51 MC'97
0x026b	MCP51 AC'97
0x026c	MCP51 HDA
0x026d	MCP51 USB controller
0x026e	MCP51 USB 2.0 controller
0x026f	MCP51 PCI subtractive bridge
0x0270	MCP51 memory controller #0
0x0271	MCP51 SMU
0x0272	MCP51 memory controller #12

C55

Paired with MCP51 or MCP55.

device id	product
0x03a0	C55 host bridge [?]
0x03a1	C55 host bridge
0x03a2	C55 host bridge
0x03a3	C55 host bridge
0x03a4	C55 host bridge [?]
0x03a5	C55 host bridge [?]
0x03a6	C55 host bridge [?]
0x03a7	C55 host bridge [?]
0x03a8	C55 memory controller #5
0x03a9	C55 memory controller #3
0x03aa	C55 memory controller #2
0x03ab	C55 memory controller #4
0x03ac	C55 memory controller #1
0x03ad	C55 memory controller #10
0x03ae	C55 memory controller #11
0x03af	C55 memory controller #12
0x03b0	C55 memory controller #13
0x03b1	C55 memory controller #14
0x03b2	C55 memory controller #15
0x03b3	C55 memory controller #16
0x03b4	C55 memory controller #7
0x03b5	C55 memory controller #6
0x03b6	C55 memory controller #20
0x03b7	C55 PCI-Express x16/x8 port
0x03b8	C55 PCI-Express x8 port
0x03b9	C55 PCI-Express x1 port #0
0x03ba	C55 memory controller #22
0x03bb	C55 PCI-Express x1 port #1
0x03bc	C55 memory controller #21

Todo

shouldn't 0x03b8 support x4 too?

MCP55

Standalone or paired with C51, C55 or C73.

device id	product
0x0360	MCP55 LPC bridge
0x0361	MCP55 LPC bridge
0x0362	MCP55 LPC bridge
0x0363	MCP55 LPC bridge
0x0364	MCP55 LPC bridge
0x0365	MCP55 LPC bridge [?]
0x0366	MCP55 LPC bridge [?]
0x0367	MCP55 LPC bridge [?]
0x0368	MCP55 SMBus controller
0x0369	MCP55 memory controller #0
0x036a	MCP55 memory controller #12
0x036b	MCP55 SMU
0x036c	MCP55 USB controller
0x036d	MCP55 USB 2.0 controller
0x036e	MCP55 IDE controller
0x036f	MCP55 SATA [???]
0x0370	MCP55 PCI subtractive bridge
0x0371	MCP55 HDA
0x0372	MCP55 ethernet controller (class 0200)
0x0373	MCP55 ethernet controller (class 0680)
0x0374	MCP55 PCI-Express x1/x4 port #0
0x0375	MCP55 PCI-Express x1/x8 port
0x0376	MCP55 PCI-Express x8 port
0x0377	MCP55 PCI-Express x8/x16 port
0x0378	MCP55 PCI-Express x1/x4 port #1
0x037e	MCP55 SATA controller [?]
0x037f	MCP55 SATA controller

MCP61

Standalone.

device id	product
0x03e0	MCP61 LPC bridge
0x03e1	MCP61 LPC bridge
0x03e2	MCP61 memory controller #0
0x03e3	MCP61 LPC bridge [?]
0x03e4	MCP61 HDA [?]
0x03e5	MCP61 ethernet controller [?]
0x03e6	MCP61 ethernet controller [?]
0x03e7	MCP61 SATA controller [?]
0x03e8	MCP61 PCI-Express x16 port
0x03e9	MCP61 PCI-Express x1 port
0x03ea	MCP61 memory controller #0
0x03eb	MCP61 SMBus controller
0x03ec	MCP61 IDE controller
0x03ee	MCP61 ethernet controller [?]
0x03ef	MCP61 ethernet controller (class 0680)
0x03f0	MCP61 HDA
0x03f1	MCP61 USB controller
0x03f2	MCP61 USB 2.0 controller
0x03f3	MCP61 PCI subtractive bridge
0x03f4	MCP61 SMU
0x03f5	MCP61 memory controller #12
0x03f6	MCP61 SATA controller
0x03f7	MCP61 SATA controller [?]

MCP65

Standalone.

device id	product
0x0440	MCP65 LPC bridge [?]
0x0441	MCP65 LPC bridge
0x0442	MCP65 LPC bridge
0x0443	MCP65 LPC bridge [?]
0x0444	MCP65 memory controller #0
0x0445	MCP65 memory controller #12
0x0446	MCP65 SMBus controller
0x0447	MCP65 SMU
0x0448	MCP65 IDE controller
0x0449	MCP65 PCI subtractive bridge
0x044a	MCP65 HDA
0x044b	MCP65 HDA [?]
0x044c	MCP65 SATA controller (AHCI mode) [?]
0x044d	MCP65 SATA controller (AHCI mode)
0x044e	MCP65 SATA controller (AHCI mode) [?]
0x044f	MCP65 SATA controller (AHCI mode) [?]
0x0450	MCP65 ethernet controller (class 0200)
0x0451	MCP65 ethernet controller [?]
0x0452	MCP65 ethernet controller (class 0680)
0x0453	MCP65 ethernet controller [?]
0x0454	MCP65 USB controller #0

Continued on next page

Table 2.3 – continued from previous page

device id	product
0x0455	MCP65 USB 2.0 controller #0
0x0456	MCP65 USB controller #1
0x0457	MCP65 USB 2.0 controller #1
0x0458	MCP65 PCI-Express x8/x16 port
0x0459	MCP65 PCI-Express x8 port
0x045a	MCP65 PCI-Express x1/x2 port
0x045b	MCP65 PCI-Express x2 port
0x045c	MCP65 SATA controller (compatibility mode) [?]
0x045d	MCP65 SATA controller (compatibility mode)
0x045e	MCP65 SATA controller (compatibility mode) [?]
0x045f	MCP65 SATA controller (compatibility mode) [?]

MCP67

Standalone.

device id	product
0x0541	MCP67 memory controller #12
0x0542	MCP67 SMBus controller
0x0543	MCP67 SMU
0x0547	MCP67 memory controller #0
0x0548	MCP67 LPC bridge
0x054c	MCP67 ethernet controller (class 0200)
0x054d	MCP67 ethernet controller [?]
0x054e	MCP67 ethernet controller [?]
0x054f	MCP67 ethernet controller [?]
0x0550	MCP67 SATA controller (compatibility mode)
0x0551	MCP67 SATA controller (compatibility mode) [?]
0x0552	MCP67 SATA controller (compatibility mode) [?]
0x0553	MCP67 SATA controller (compatibility mode) [?]
0x0554	MCP67 SATA controller (AHCI mode)
0x0555	MCP67 SATA controller (AHCI mode) [?]
0x0556	MCP67 SATA controller (AHCI mode) [?]
0x0557	MCP67 SATA controller (AHCI mode) [?]
0x0558	MCP67 SATA controller (AHCI mode) [?]
0x0559	MCP67 SATA controller (AHCI mode) [?]
0x055a	MCP67 SATA controller (AHCI mode) [?]
0x055b	MCP67 SATA controller (AHCI mode) [?]
0x055c	MCP67 HDA
0x055d	MCP67 HDA [?]
0x055e	MCP67 USB controller
0x055f	MCP67 USB 2.0 controller
0x0560	MCP67 IDE controller
0x0561	MCP67 PCI subtractive bridge
0x0562	MCP67 PCI-Express x16 port
0x0563	MCP67 PCI-Express x1 port

C73

Paired with MCP55.

device id	product
0x0800	C73 host bridge
0x0801	C73 host bridge [?]
0x0802	C73 host bridge [?]
0x0803	C73 host bridge [?]
0x0804	C73 host bridge [?]
0x0805	C73 host bridge [?]
0x0806	C73 host bridge [?]
0x0807	C73 host bridge [?]
0x0808	C73 memory controller #1
0x0809	C73 memory controller #2
0x080a	C73 memory controller #3
0x080b	C73 memory controller #4
0x080c	C73 memory controller #5
0x080d	C73 memory controller #6
0x080e	C73 memory controller #7/#17
0x080f	C73 memory controller #10
0x0810	C73 memory controller #11
0x0811	C73 memory controller #12
0x0812	C73 memory controller #13
0x0813	C73 memory controller #14
0x0814	C73 memory controller #15
0x0815	C73 PCI-Express x? port #0
0x0817	C73 PCI-Express x? port #1
0x081a	C73 memory controller #16

MCP73

Standalone.

device id	product
0x056a	MCP73 USB 2.0 controller
0x056c	MCP73 IDE controller
0x056d	MCP73 PCI subtractive bridge
0x056e	MCP73 PCI-Express x16 port
0x056f	MCP73 PCI-Express x1 port
0x07c0	MCP73 host bridge
0x07c1	MCP73 host bridge
0x07c2	MCP73 host bridge [?]
0x07c3	MCP73 host bridge
0x07c4	MCP73 host bridge [?]
0x07c5	MCP73 host bridge
0x07c6	MCP73 host bridge [?]
0x07c7	MCP73 host bridge
0x07c8	MCP73 memory controller #34
0x07cb	MCP73 memory controller #1
0x07cd	MCP73 memory controller #10
0x07ce	MCP73 memory controller #11
0x07cf	MCP73 memory controller #12
0x07d0	MCP73 memory controller #13
0x07d1	MCP73 memory controller #14

Continued on next page

Table 2.4 – continued from previous page

device id	product
0x07d2	MCP73 memory controller #15
0x07d3	MCP73 memory controller #16
0x07d6	MCP73 memory controller #20
0x07d7	MCP73 LPC bridge
0x07d8	MCP73 SMBus controller
0x07d9	MCP73 memory controller #32
0x07da	MCP73 SMU
0x07dc	MCP73 ethernet controller (class 0200)
0x07dd	MCP73 ethernet controller [?]
0x07de	MCP73 ethernet controller [?]
0x07df	MCP73 ethernet controller [?]
0x07f0	MCP73 SATA controller (compatibility mode)
0x07f1	MCP73 SATA controller (compatibility mode) [?]
0x07f2	MCP73 SATA controller (compatibility mode) [?]
0x07f3	MCP73 SATA controller (compatibility mode) [?]
0x07f4	MCP73 SATA controller (AHCI mode)
0x07f5	MCP73 SATA controller (AHCI mode) [?]
0x07f6	MCP73 SATA controller (AHCI mode) [?]
0x07f7	MCP73 SATA controller (AHCI mode) [?]
0x07f8	MCP73 SATA controller (RAID mode)
0x07f9	MCP73 SATA controller (RAID mode) [?]
0x07fa	MCP73 SATA controller (RAID mode) [?]
0x07fb	MCP73 SATA controller (RAID mode) [?]
0x07fc	MCP73 HDA
0x07fd	MCP73 HDA [?]
0x07fe	MCP73 USB controller

MCP77

Standalone.

device id	product
0x0568	MCP77 memory controller #14
0x0569	MCP77 IGP bridge
0x0570-0x057f	MCP* ethernet controller (class 0200 alt) [XXX]
0x0580-0x058f	MCP* SATA controller (alt ID) [XXX]
0x0590-0x059f	MCP* HDA (alt ID) [XXX]
0x05a0-0x05af	MCP* IDE (alt ID) [XXX]
0x0751	MCP77 memory controller #12
0x0752	MCP77 SMBus controller
0x0753	MCP77 SMU
0x0754	MCP77 memory controller #0
0x0755	MCP77 memory controller #0 [?]
0x0756	MCP77 memory controller #0 [?]
0x0757	MCP77 memory controller #0 [?]
0x0759	MCP77 IDE controller
0x075a	MCP77 PCI subtractive bridge
0x075b	MCP77 PCI-Express x1/x4 port
0x075c	MCP77 LPC bridge

Continued on next page

Table 2.5 – continued from previous page

device id	product
0x075d	MCP77 LPC bridge
0x075e	MCP77 LPC bridge
0x0760	MCP77 ethernet controller (class 0200)
0x0761	MCP77 ethernet controller [?]
0x0762	MCP77 ethernet controller [?]
0x0763	MCP77 ethernet controller [?]
0x0764	MCP77 ethernet controller (class 0680)
0x0765	MCP77 ethernet controller [?]
0x0766	MCP77 ethernet controller [?]
0x0767	MCP77 ethernet controller [?]
0x0774	MCP77 HDA
0x0775	MCP77 HDA [?]
0x0776	MCP77 HDA [?]
0x0777	MCP77 HDA [?]
0x0778	MCP77 PCI-Express 2.0 x8/x16 port
0x0779	MCP77 PCI-Express 2.0 x8 port
0x077a	MCP77 PCI-Express x1 port
0x077b	MCP77 USB controller #0
0x077c	MCP77 USB 2.0 controller #0
0x077d	MCP77 USB controller #1
0x077e	MCP77 USB 2.0 controller #1
0x0ad0-0x0ad3	MCP77 SATA controller (compatibility mode)
0x0ad4-0x0ad7	MCP77 SATA controller (AHCI mode)
0x0ad8-0x0adb	MCP77 SATA controller (RAID mode)

MCP79

Standalone.

device id	product
0x0570-0x057f	MCP* ethernet controller (class 0200 alt) [XXX]
0x0580-0x058f	MCP* SATA controller (alt ID) [XXX]
0x0590-0x059f	MCP* HDA (alt ID) [XXX]
0x0a80	MCP79 host bridge
0x0a81	MCP79 host bridge [?]
0x0a82	MCP79 host bridge
0x0a83	MCP79 host bridge
0x0a84	MCP79 host bridge
0x0a85	MCP79 host bridge [?]
0x0a86	MCP79 host bridge
0x0a87	MCP79 host bridge [?]
0x0a88	MCP79 memory controller #1
0x0a89	MCP79 memory controller #33
0x0a8d	MCP79 memory controller #13
0x0a8e	MCP79 memory controller #14
0x0a8f	MCP79 memory controller #15
0x0a90	MCP79 memory controller #16
0x0a94	MCP79 memory controller #23
0x0a95	MCP79 memory controller #24

Continued on next page

Table 2.6 – continued from previous page

device id	product
0x0a98	MCP79 memory controller #34
0x0aa0	MCP79 IGP bridge
0x0aa2	MCP79 SMBus controller
0x0aa3	MCP79 SMU
0x0aa4	MCP79 memory controller #31
0x0aa5	MCP79 USB controller #0
0x0aa6	MCP79 USB 2.0 controller #0
0x0aa7	MCP79 USB controller #1
0x0aa8	MCP79 USB controller [?]
0x0aa9	MCP79 USB 2.0 controller #1
0x0aaa	MCP79 USB 2.0 controller [?]
0x0aab	MCP79 PCI subtractive bridge
0x0aac	MCP79 LPC bridge
0x0aad	MCP79 LPC bridge
0x0aae	MCP79 LPC bridge
0x0aaf	MCP79 LPC bridge
0x0ab0	MCP79 ethernet controller (class 0200)
0x0ab1	MCP79 ethernet controller [?]
0x0ab2	MCP79 ethernet controller [?]
0x0ab3	MCP79 ethernet controller [?]
0x0ab4-0x0ab7	MCP79 SATA controller (compatibility mode)
0x0ab8-0x0abb	MCP79 SATA controller (AHCI mode)
0x0abc-0x0abf	MCP79 SATA controller (RAID mode) [XXX: actually 0x0ab0-0xabb are accepted by hw without trickery]
0x0ac0	MCP79 HDA
0x0ac1	MCP79 HDA [?]
0x0ac2	MCP79 HDA [?]
0x0ac3	MCP79 HDA [?]
0x0ac4	MCP79 PCI-Express 2.0 x16 port
0x0ac5	MCP79 PCI-Express 2.0 x4/x8 port
0x0ac6	MCP79 PCI-Express 2.0 x1/x4 port
0x0ac7	MCP79 PCI-Express 2.0 x1 port
0x0ac8	MCP79 PCI-Express 2.0 x4 port

MCP89

Standalone.

device id	product
0x0580-0x058f	MCP* SATA controller (alt ID) [XXX]
0x0590-0x059f	MCP* HDA (alt ID) [XXX]
0x0d60	MCP89 host bridge
0x0d68	MCP89 memory controller #1
0x0d69	MCP89 memory controller #33
0x0d6d	MCP89 memory controller #10
0x0d6e	MCP89 memory controller #11
0x0d6f	MCP89 memory controller #12
0x0d70	MCP89 memory controller #13
0x0d71	MCP89 memory controller #20
0x0d72	MCP89 memory controller #21
0x0d75	MCP89 memory controller #110
0x0d76	MCP89 IGP bridge
0x0d79	MCP89 SMBus controller
0x0d7a	MCP89 SMU
0x0d7b	MCP89 memory controller #31
0x0d7d	MCP89 ethernet controller (class 0200)
0x0d80	MCP89 LPC bridge
0x0d84-0x0d87	MCP89 SATA controller (compatibility mode)
0x0d88-0x0d8b	MCP89 SATA controller (AHCI mode)
0x0d8c-0x0d8f	MCP89 SATA controller (RAID mode)
0x0d94-0x0d97	MCP89 HDA [XXX: actually 1-0xf]
0x0d9a	MCP89 PCI-Express x1 port #0
0x0d9b	MCP89 PCI-Express x1 port #1
0x0d9c	MCP89 USB controller
0x0d9d	MCP89 USB 2.0 controller

2.3.8 Tegra

T20

device id	product
0x0bf0	T20 PCI-Express x? port #0
0x0bf1	T20 PCI-Express x? port #1

T30

device id	product
0x0e1c	T30 PCI-Express x? port #0
0x0e1d	T30 PCI-Express x? port #1

2.4 PCI/PCIE/AGP bus interface and card management logic

Contents:

2.4.1 PCI BARs and other means of accessing the GPU

Contents

Nvidia GPU BARs, IO ports, and memory areas

The nvidia GPUs expose the following areas to the outside world through PCI:

- PCI configuration space / PCIE extended configuration space
- MMIO registers: BAR0 - memory, 0x1000000 bytes or more depending on card type
- VRAM aperture: BAR1 - memory, 0x1000000 bytes or more depending on card type [NV3+ only]
- indirect memory access IO ports: BAR2 - 0x100 bytes of IO port space [NV3 only]
- ???: BAR2 [only NV1x IGPs?]
- ???: BAR2 [only NV20?]
- RAMIN aperture: BAR2 or BAR3 - memory, 0x1000000 bytes or more depending on card type [NV40+]
- indirect memory access IO ports: BAR5 - 0x80 bytes of IO port space [G80+]
- PCI ROM aperture
- PCI INTA interrupt line
- legacy VGA IO ports: 0x3b0-0x3bb and 0x3c0-0x3df [can be disabled in PCI config]
- legacy VGA memory: 0xa0000-0xbffff [can be disabled in PCI config]

PCI/PCIE configuration space

Nvidia GPUs, like all PCI devices, have PCI configuration space. Its contents are described in pci.

BAR0: MMIO registers

This is the main control space of the card - all engines are controlled through it, and it contains alternate means to access most of the other spaces. This, along with the VRAM / RAMIN apertures, is everything that's needed to fully control the cards.

This space is a 16MB area of memory sparsely populated with areas representing individual engines, which in turn are sparsely populated with registers. The list of engines depends on card type. While there are no known registers outside 16MB range, the BAR itself can have a larger size on NV40+ cards if configured so by straps.

Its address is set up through PCI BAR 0. The BAR uses 32-bit addressing and is non-prefetchable memory.

The registers inside this BAR are 32-bit, with the exception of areas that are aliases of the byte-oriented VGA legacy IO ports. They should be accessed through aligned 32-bit memory reads/writes. On pre-NV1A cards, the registers are always little endian, on NV1A+ cards endianness of the whole area can be selected by a switch in PMC. The endianness switch, however, only affects BAR0 accesses to the MMIO space - accesses from inside the card are always little-endian.

A particularly important subarea of MMIO space is PMC, the card's master control. This subarea is present on all nvidia GPUs at addresses 0x000000 through 0x000fff. It contains GPU id information, Big Red Switches for engines that can be turned off, and master interrupt control. It's described in more detail in pmc.

For full list of MMIO areas, see mmio.

BAR1: VRAM aperture

This is an area of prefetchable memory that maps to the card's VRAM. On native PCIE cards, it uses 64-bit addressing, on native PCI/AGP ones it uses 32-bit addressing.

On non-TURBOCACHE pre-G80 cards and on G80+ cards with BAR1 VM disabled, BAR addresses map directly to VRAM addresses. On TURBOCACHE cards, BAR1 is made of controllable VRAM and GART windows [see *NV44 host memory interface*]. G80+ cards have a mode where all BAR references go through the card's VM subsystem, see g80-host-mem and gf100-host-mem.

On NV3 cards, this BAR also contains RAMIN access aperture at address 0xc00000 [see *NV3 VRAM structure and usage*]

Todo

map out the BAR fully

---

the BAR size depends on card type:

- NV3: 16MB [with RAMIN]
- NV4: 16MB
- NV5: 32MB
- NV10:NV17: 128MB
- NV17:G80: 64MB-512MB, set via straps
- G80-: 64MB-64GB, set via straps

Note that BAR size is independent from actual VRAM size, although on pre-NV30 cards the BAR is guaranteed not to be smaller than VRAM. This means it may be impossible to map all of the card's memory through the BAR on NV30+ cards.

BAR2/BAR3: RAMIN aperture

RAMIN is, on pre-G80 cards, a special area at the end of VRAM that contains various control structures. RAMIN starts from end of VRAM and the addresses go in reverse direction, thus it needs a special mapping to access it the way it'll be used. While pre-NV40 cards limitted its size to 1MB and could fit the mapping in BAR0, or BAR1 for NV3, NV40+ allow much bigger RAMIN addresses. RAMIN BAR provides such RAMIN mapping on NV40 family cards.

G80 did away with a special RAMIN area, but it kept the BAR around. It works like BAR1, but is independent on it and can use a distinct VM DMA object. As opposed to BAR1, all accesses done to BAR3 will be automatically

byte-swapped in 32-bit chunks like BAR0 if the big-endian switch is on. It's commonly used to map control structures for kernel use, while BAR1 is used to map user-accessible memory.

The BAR uses 64-bit addressing on native PCIE cards, 32-bit addressing on native PCI/AGP. It uses BAR2 slot on native PCIE, BAR3 on native PCI/AGP. It is non-prefetchable memory on cards up to and including G200, prefetchable memory on MCP77+. The size is at least 16MB and is set via straps.

BAR2: NV3 indirect memory access

An area of IO ports used to access BAR0 or BAR1 indirectly by real mode code that cannot map high memory addresses. Present only on NV3.

Todo

RE it. or not.

BAR5: G80 indirect memory access

An area of IO ports used to access BAR0, BAR1, and BAR3 indirectly by real mode code that cannot map high memory addresses. Present on G80+ cards. On earlier cards, the indirect access feature of VGA IO ports can be used instead. This BAR can also be disabled via straps.

Todo

It's present on some NV4x

This area is 0x80 bytes of IO ports, but only first 0x20 bytes are actually used; the rest are empty. The ports are all treated as 32-bit ports. They are:

BAR5+0x00: when read, signature: 0x2469fdb9. When written, master enable: write 1 to enable remaining ports, 0 to disable. Only bit 0 of the written value is taken into account. When remaining ports are disabled, they read as 0xffffffff.

BAR5+0x04: enable. if bit 0 is 1, the "data" ports are active, otherwise they're inactive and merely store the last written value.

BAR5+0x08: BAR0 address port. bits 0-1 and 24-31 are ignored.

BAR5+0x0c: BAR0 data port. Reads and writes are translated to BAR0 reads and writes at address specified by BAR0 address port.

BAR5+0x10: BAR1 address port. bits 0-1 are ignored.

BAR5+0x14: BAR1 data port. Reads and writes are translated to BAR1 reads and writes at address specified by BAR1 address port.

BAR5+0x18: BAR3 address port. bits 0-1 and 24-31 are ignored.

BAR5+0x1c: BAR3 data port. Reads and writes are translated to BAR3 reads and writes at address specified by BAR3 address port.

BAR0 addresses are masked to low 24 bits, allowing access to exactly 16MB of MMIO space. The BAR1 addresses aren't masked, and the window actually allows access to more BAR space than the BAR1 itself - up to 4GB of VRAM or VM space can be accessed this way. BAR3 addresses, on the other hand, are masked to low 24 bits even though the real BAR3 is larger.

BAR6: PCI ROM aperture

Todo

figure out size

Todo

figure out NV3

Todo

verify G80

The nvidia GPUs expose their BIOS as standard PCI ROM. The exposed ROM aliases either the actual BIOS EEPROM, or the shadow BIOS in VRAM. This setting is exposed in PCI config space. If the "shadow enabled" PCI config register is 0, the PROM MMIO area is enabled, and both PROM and the PCI ROM aperture will access the EEPROM. Disabling the shadowing has a side effect of disabling video output on pre-G80 cards. If shadow is enabled, EEPROM is disabled, PROM reads will return garbage, and PCI ROM aperture will access the VRAM shadow copy of BIOS. On pre-G80 cards, the shadow BIOS is located at address 0 of RAMIN, on G80+ cards the shadow bios is pointed to by PDISPLAY.VGA.ROM_WINDOW register - see g80-vga for details.

INTA: the card interrupt

Todo

MSI

The GPU reports all interrupts through the PCI INTA line. The interrupt enable and status registers are located in PMC area - see pmc-intr.

Legacy VGA IO ports and memory

The nvidia GPU cards are backwards compatible with VGA and expose the usual VGA ranges: IO ports 0x3b0-0x3bb and 0x3c0-0x3df, memory at 0xa0000-0xbffff. The VGA ranges can however be disabled in PCI config space. The VGA registers and memory are still accessible through their aliases in BAR0, and disabling the legacy ranges has no effect on the operation of the card. The IO range contains an extra top-level register that allows indirect access to the MMIO area for use by real mode code, as well as many nvidia-specific extra registers in the VGA subunits. For details, see nv3-vga.

2.5 Power, thermal, and clock management

Contents:

2.5.1 Clock management

The nvidia GPUs, like most electronic devices, use clock signals to control their operation. Since they're complicated devices made of many subunits with different performance needs, there are multiple clock signals for various parts of the GPU.

The set of available clocks and the method of setting them varies a lot with the card type.

Contents:

NV1:NV40 clocks

Contents

Todo

write me

Introduction

NV1:NV40 GPUs have the following clocks:

- core clock [NVPLL]: used to clock the graphics engine - on NV4+ cards only
- memory clock [MPLL]: used to clock memory and, before NV4, the graphics engine - not present on IGPs
- video clock 1 [VPLL]: used to drive first/only video output
- video clock 2 [VPLL2]: used to drive second video output - only on NV11+ cards

Todo

DLL on NV3

Todo

NV1???

These clocks are set in PDAC [NV1] or PRAMDAC [NV3 and up] areas.

Todo

write me

NVCLK: core clock

Todo

write me

MCLK: memory clock

Todo

write me

2.5.2 PDAEMON: card management microprocesor

Contents:

falcon parameters

Present on:

- **v0:** GT215:MCP89
- **v1:** MCP89:GF100
- **v2:** GF100:GF119
- **v3:** GF119:GK104
- **v4:** GK104:GK110
- **v5:** GK110:GK208
- **v6:** GK208:GM107
- **v7:** GM107+

BAR0 address: 0x10a000

PMC interrupt line:

- **v0-v1:** 18
- **v2+:** 24

PMC enable bit:

- **v0-v1:** none, use reg 0x22210 instead
- **v2+:** 13

Version:

- **v0-v2:** 3
- **v3,v4:** 4
- **v5:** 4.1

v6,v7: 5

Code segment size:

v0: 0x4000

v1:v7: 0x6000

v7: 0x8000

Data segment size:

v0: 0x3000

v1+: 0x6000

Fifo size:

v0-v1: 0x10

v2+: 3

Xfer slots:

v0-v2: 8

v3-v4: 0x10

Secretful:

v0:v7: no

v7: yes

Code TLB index bits:

v0-v2: 8

v3+: 9

Code ports: 1

Data ports: 4

Version 4 unknown caps: 31, 27

Unified address space: yes [on v3+]

IO addressing type:

v0-v2: indexed

v3-v7: simple

Core clock:

v0-v1: gt215-clock-dclk

v2-v7: gf100-clock-dclk

Tesla VM engine: 0xe

Tesla VM client: 0x11

Tesla context DMA: [none]

Fermi VM engine: 0x17

Fermi VM client: HUB 0x12

Interrupts:

Line	Type	Present on	Name	Description
8	edge	GT215:GF100	MEMIF_PORT_INVALID	*MEMIF port not initialised*
9	edge	GT215:GF100	MEMIF_FAULT	*MEMIF VM fault*
9	edge	GF100-	MEMIF_BREAK	*MEMIF breakpoint*
10	level	all	PMC_DAEMON	PMC interrupts routed directly to PDAEMON
11	level	all	SUBINTR	*second-level interrupt*
12	level	all	THERM	*PTHERM subinterrupts routed to PDAEMON*
13	level	all	SIGNAL	*input signal rise/fall interrupts*
14	level	all	TIMER	*the timer interrupt*
15	level	all	IREDIR_PMC	*PMC interrupts redirected to PDAEMON by IREDIR*

Status bits:

Bit	Present on	Name	Description
0	all	FALCON	*Falcon unit*
1	all	EPWR_GRAPH	*PGRAPH engine power gating*
2	all	EPWR_VDEC	*video decoding engine power gating*
3	all	MEMIF	*Memory interface*
4	GT215:MCP89 GF100-	USER	*User controlled*
4	MCP89:GF100	EPWR_VCOMP	*PVCOMP engine power gating*
5	MCP89:GF100	USER	*User controlled*

IO registers: pdaemon-io

PCOUNTER signals

Todo

write me

Todo

discuss mismatched clock thing

- ???
- *IREDIR_STATUS*
- *IREDIR_HOST_REQ*
- *IREDIR_TRIGGER_DAEMON*
- *IREDIR_TRIGGER_HOST*
- *IREDIR_PMC*
- *IREDIR_INTR*
- MMIO_BUSY
- MMIO_IDLE
- MMIO_DISABLED
- *TOKEN_ALL_USED*
- *TOKEN_NONE_USED*
- *TOKEN_FREE*

- *TOKEN_ALLOC*
- *FIFO_PUT_0_WRITE*
- *FIFO_PUT_1_WRITE*
- *FIFO_PUT_2_WRITE*
- *FIFO_PUT_3_WRITE*
- INPUT_CHANGE
- OUTPUT_2
- INPUT_2
- *THERM_ACCESS_BUSY*

Todo

figure out the first signal

Todo

document MMIO_* signals

Todo

document INPUT_*, OUTPUT_*

Second-level interrupts

Because falcon has space for only 8 engine interrupts and PDAEMON needs many more, a second-level interrupt register was introduced:

MMIO 0x688 / I[0x1a200]: SUBINTR

- bit 0: H2D - *host to PDAEMON scratch register written*
- bit 1: FIFO - *host to PDAEMON fifo pointer updated*
- bit 2: EPWR_GRAPH - *PGRAPH engine power control*
- bit 3: EPWR_VDEC - *video decoding engine power control*
- bit 4: MMIO - *indirect MMIO access error*
- bit 5: IREDIR_ERR - *interrupt redirection error*
- bit 6: IREDIR_HOST_REQ - *interrupt redirection request*
- bit 7: ???
- bit 8: ??? - goes to 0x670
- bit 9: EPWR_VCOMP [MCP89] - *PVCOMP engine power control*
- bit 13: ??? [GF119-] - goes to 0x888

Todo

figure out bits 7, 8

Todo

more bits in 10-12?

The second-level interrupts are merged into a single level-triggered interrupt and delivered to falcon interrupt line 11. This line is asserted whenever any bit of SUBINTR register is non-0. A given SUBINTR bit is set to 1 whenever the input second-level interrupt line is 1, but will not auto-clear when the input line goes back to 0 - only writing 1 to that bit in SUBINTR will clear it. This effectively means that SUBINTR bits have to be cleared after the downstream interrupt. Note that SUBINTR has no corresponding enable bit - if an interrupt needs to be disabled, software should use the enable registers corresponding to individual second-level interrupts instead.

Note that IREDIR_HOST_REQ interrupt has special semantics when cleared - see IREDIR_TRIGGER documentation.

User busy indication

To enable the microcode to set the "PDAEMON is busy" flag without actually making any PDAEMON subunit perform computation, bit 4 of the falcon status register is connected to a dummy unit whose busy status is controlled directly by the user:

MMIO 0x420 / I[0x10800]: USER_BUSY Read/write, only bit 0 is valid. If set, falcon status line 4 or 5 [USER] is set to 1 [busy], otherwise it's set to 0 [idle].

Todo

what could possibly use PDAEMON's busy status?

Host <-> PDAEMON communication

Contents

Introduction

There are 4 PDAEMON-specific channels that can be used for communication between the host and PDAEMON:

- FIFO: data submission from host to PDAEMON on 4 independent FIFOs in data segment, with interrupts generated whenever the PUT register is written

- RFIFO: data submission from PDAEMON to host on through a FIFO in data segment
- H2D: a single scratch register for host -> PDAEMON communication, with interrupts generated whenever it's written
- D2H: a single scratch register for PDAEMON -> host communication
- DSCRATCH: 4 scratch registers

Submitting data to PDAEMON: FIFO

These registers are meant to be used for submitting data from host to PDAEMON. The PUT register is FIFO head, written by host, and GET register is FIFO tail, written by PDAEMON. Interrupts can be generated whenever the PUT register is written. How exactly the data buffer works is software's business. Note that due to very limitted special semantics for FIFO uage, these registers may as well be used as [possibly interruptful] scratch registers.

MMIO 0x4a0+i*4 / I[0x12800+i*0x100], i<4: FIFO_PUT[i] The FIFO head pointer, effectively a 32-bit scratch register. Writing it causes bit i of FIFO_INTR to be set.

MMIO 0x4b0+i*4 / I[0x12c00+i*0x100], i<4: FIFO_GET[i] The FIFO tail pointer, effectively a 32-bit scratch register.

MMIO 0x4c0 / I[0x13000]: FIFO_INTR The status register for FIFO_PUT write interrupts. Write a bit with 1 to clear it. Whenever a bit is set both in FIFO_INTR and FIFO_INTR_EN, the FIFO [#1] second-level interrupt line to SUBINTR is asserted. Bit i corresponds to FIFO #i, and only bits 0-3 are valid.

MMIO 0x4c4 / I[0x13100]: FIFO_INTR_EN The enable register for FIFO_PUT write interrupts. Read/write, only 4 low bits are valid. Bit assignment is the same as in FIFO_INTR.

In addition, the FIFO circuitry exports four signals to PCOUNTER:

- FIFO_PUT_0_WRITE: pulses for one cycle whenever FIFO_PUT[0] is written
- FIFO_PUT_1_WRITE: pulses for one cycle whenever FIFO_PUT[1] is written
- FIFO_PUT_2_WRITE: pulses for one cycle whenever FIFO_PUT[2] is written
- FIFO_PUT_3_WRITE: pulses for one cycle whenever FIFO_PUT[3] is written

Submitting data to host: RFIFO

The RFIFO is like one of the 4 FIFOs, except it's supposed to go from PDAEMON to the host and doesn't have the interupt generation powers.

MMIO 0x4c8 / I[0x13200]: RFIFO_PUT MMIO 0x4cc / I[0x13300]: RFIFO_GET

The RFIFO head and tail pointers. Both are effectively 32-bit scratch registers.

Host to PDAEMON scratch register: H2D

H2D is a scratch register supposed to be written by the host and read by PDAEMON. It generates an interrupt when written.

MMIO 0x4d0 / I[0x13400]: H2D A 32-bit scratch register. Sets H2D_INTR when written.

MMIO 0x4d4 / I[0x13500]: H2D_INTR The status register for H2D write interrupt. Only bit 0 is valid. Set when H2D register is written, cleared when 1 is written to bit 0. When this and H2D_INTR_EN are both set, the H2D [#0] second-level interrupt line to SUBINTR is asserted.

MMIO 0x4d8 / I[0x13600]: H2D_INTR_EN The enable register for H2D write interrupt. Only bit 0 is valid.

PDAEMON to host scratch register: D2H

D2H is just a scratch register supposed to be written by PDAEMON and read by the host. It has no interrupt genration powers.

MMIO 0x4dc / I[0x13700]: D2H A 32-bit scratch register.

Scratch registers: DSCRATCH

DSCRATCH[] are just 4 32-bit scratch registers usable for PDAEMON<->HOST communication or any other purposes.

MMIO 0x5d0+i*4 / I[0x17400+i*0x100], i<4: DSCRATCH[i] A 32-bit scratch register.

Hardware mutexes

The PDAEMON has hardware support for 16 busy-waiting mutexes accessed by up to 254 clients simultanously. The clients may be anything able to read and write the PDAEMON registers - code running on host, on PDAEMON, or on any other falcon engine with MMIO access powers.

The clients are identified by tokens. Tokens are 8-bit numbers in 0x01-0xfe range. Tokens may be assigned to clients statically by software, or dynamically by hardware. Only tokens 0x08-0xfe will be dynamically allocated by hardware - software may use statically assigned tokens 0x01-0x07 even if dynamic tokens are in use at the same time. The registers used for dynamic token allocation are:

MMIO 0x488 / I[0x12200]: TOKEN_ALLOC Read-only, each read to this register allocates a free token and gives it as the read result. If there are no free tokens, 0xff is returned.

MMIO 0x48c / I[0x12300]: TOKEN_FREE A write to this register will free a token, ie. return it back to the pool used by TOKEN_ALLOC. Only low 8 bits of the written value are used. Attempting to free a token outside of the dynamic allocation range [0x08-0xff] or a token already in the free queue will have no effect. Reading this register will show the last written value, invalid or not.

The free tokens are stored in a FIFO - the freed tokens will be used by TOKEN_ALLOC in the order of freeing. After reset, the free token FIFO will contain tokens 0x08-0xfe in ascending order.

The actual mutex locking and unlocking is done by the MUTEX_TOKEN registers:

MMIO 0x580+i*4 / I[0x16000+i*0x100], i<16: MUTEX_TOKEN[i] The 16 mutices. A value of 0 means unlocked, any other value means locked by the client holding the corresponding token. Only low 8 bits of the written value are used. A write of 0 will unlock the mutex and will always succeed. A write of 0x01-0xfe will succeed only if the mutex is currently unlocked. A write of 0xff is invalid and will always fail. A failed write has no effect.

The token allocation circuitry additionally exports four signals to PCOUNTER:

- TOKEN_ALL_USED: 1 iff all tokens are currently allocated [ie. a read from TOKEN_ALLOC would return 0xff]
- TOKEN_NONE_USED: 1 iff no tokens are currently allocated [ie. tokens 0x08-0xfe are all in free tokens queue]
- TOKEN_FREE: pulses for 1 cycle whenever TOKEN_FREE is written, even if with invalid value
- TOKEN_ALLOC: pulses for 1 cycle whenever TOKEN_ALLOC is read, even if allocation fails

CRC computation

The PDAEMON has a very simple CRC accelerator. Specifically, it can perform the CRC accumulation operation on 32-bit chunks using the standard CRC-32 polynomial of 0xedb88320. The current CRC residual is stored in the CRC_STATE register:

MMIO 0x494 / I[0x12500]: CRC_STATE The current CRC residual. Read/write.

And the data to add to the CRC is written to the CRC_DATA register:

MMIO 0x490 / I[0x12400]: CRC_DATA When written, appends the 32-bit LE value to the running CRC residual in CRC_STATE. When read, returns the last value written. Write operation:

```
CRC_STATE ^= value;
for (i = 0; i < 32; i++) {
    if (CRC_STATE & 1) {
        CRC_STATE >>= 1;
        CRC_STATE ^= 0xedb88320;
    } else {
        CRC_STATE >>= 1;
    }
}
```

To compute a CRC:

1. Write the initial CRC residue to CRC_STATE
2. Write all data to CRC_DATA, in 32-bit chunks
3. Read CRC_STATE, xor its value with the final constant, use that as the CRC.

If the data block to CRC has size that is not a multiple of 32 bits, the extra bits at the end [or the beginning] have to be handled manually.

The timer

Aside from the usual *falcon timers*, PDAEMON has its own timer. The timer can be configured as either one-shot or periodic, can run on either daemon clock or PTIMER clock divided by 64, and generates interrupts. The following registers deal with the timer:

MMIO 0x4e0 / I[0x13800]: TIMER_START The 32-bit count the timer starts counting down from. Read/write. For periodic mode, the period will be equal to TIMER_START+1 source cycles.

MMIO 0x4e4 / I[0x13900]: TIMER_TIME The current value of the timer, read only. If TIMER_CONTROL.RUNNING is set, this will decrease by 1 on every rising edge of the source clock. If such rising edge causes this register to become 0, the TIMER_INTR bit 8 [TIMER] is set. The behavior of rising edge when this register is already 0 depends on the timer mode: in ONESHOT mode, nothing will happen. In PERIODIC mode, the timer will be reset to the value from TIMER_START. Note that interrupts won't be generated if the timer becomes 0 when copying the value from TIMER_START, whether caused by starting the timer or beginning a new PERIODIC period. This means that using PERIODIC mode with TIMER_START of 0 will never generate any interrupts.

MMIO 0x4e8 / I[0x13a00]: TIMER_CTRL

- bit 0: RUNNING - when 0, the timer is stopped, when 1, the timer is runninng. Setting this bit to 1 when it was previously 0 will also copy the TIMER_START value to TIMER_TIME.
- bit 4: SOURCE - selects the source clock
  - 0: DCLK - daemon clock, effectively timer decrements by 1 on every daemon cycle

  - 1: PTIMER_B5 - PTIMER time bit 5 [ie. bit 10 of TIME_LOW]. Since timer decrements by 1 on every rising edge of the clock, this effectively decrements the counter on every 64th PTIMER clock.

- bit 8: MODE - selects the timer mode
  - 0: ONESHOT - timer will halt after reaching 0
  - 1: PERIODIC - timer will restart from TIMER_START after reaching 0

MMIO 0x680 / I[0x1a000]: TIMER_INTR

- bit 8: TIMER - set whenever TIMER_TIME becomes 0 except by a copy from TIMER_START, write 1 to this bit to clear it. When this and bit 8 of TIMER_INTR_EN are set at the same time, falcon interrupt line #14 [TIMER] is asserted.

MMIO 0x684 / I[0x1a100]: TIMER_INTR_EN

- bit 8: TIMER - when set, timer interupt delivery to falcon interrupt line 14 is enabled.

Channel switching

Todo

write me

PMC interrupt redirection

One of PDAEMON powers is redirecting the PMC INTR_HOST interrupt to itself. The redirection hw may be in one of two states:

- HOST: PMC INTR_HOST output connected to PCI interrupt line [ORed with PMC INTR_NRHOST output], PDAEMON falcon interrupt #15 disconnected and forced to 0
- DAEMON: PMC INTR_HOST output connected to PDAEMON falcon interrupt #15 [IREDIR_PMC], PCI interrupt line connected to INTR_NRHOST output only

In addition, there's a capability enabling host to send "please turn redirect status back to HOST" interrupt with a timeout mechanism that will execute the request in hardware if the PDAEMON fails to respond to the interrupt in a given time.

Note that, as a side effect of having this circuitry, PMC INTR_HOST line will be delivered nowhere [falcon interrupt line #15 will be 0, PCI interrupt line will be connected to INTR_NRHOST only] whenever the IREDIR circuitry is in reset state, due to either whole PDAEMON reset through PMC.ENABLE / PDAEMON_ENABLE or DAEMON circuitry reset via SUBENGINE_RESET with DAEMON set in the reset mask.

The redirection state may be read at:

MMIO 0x690 / I[0x1a400]: IREDIR_STATUS Read-only. Reads as 0 if redirect hw is in HOST state, 1 if it's in DAEMON state.

The redirection state may be controlled by:

MMIO 0x68c / I[0x1a300]: IREDIR_TRIGGER This register is write-only.

- bit 0: HOST_REQ - when written as 1, sends the "request redirect state change to HOST" interrupt, setting SUBINTR bit #6 [IREDIR_HOST_REQ] to 1 and starting the timeout, if enabled. When written as 1 while redirect hw is already in HOST state, will just cause HOST_REQ_REDUNDANT error instead.
- bit 4: DAEMON - when written as 1, sets the redirect hw state to DAEMON. If it was set to DAEMON already, causes DAEMON_REDUNDANT error.

- bit 12: HOST - when written as 1, sets the redirect hw state to HOST. If it was set to HOST already, causes HOST_REDUNDANT error. Does *not* clear IREDIR_HOST_REQ interrupt bit.

Writing a value with multiple bits set is not a good idea - one of them will cause an error.

The IREDIR_HOST_REQ interrupt state should be cleared by writing 1 to the corresponding SUBINTR bit. Once this is done, the timeout counting stops, and redirect hw goes to HOST state if it wasn't already.

The IREDIR_HOST_REQ timeout is controlled by the following registers:

MMIO 0x694 / I[0x1a500]: IREDIR_TIMEOUT The timeout duration in daemon cycles. Read/write, 32-bit.

MMIO 0x6a4 / I[0x1a900]: IREDIR_TIMEOUT_ENABLE The timeout enable. Only bit 0 is valid. When set to 0, timeout mechanism is disabled, when set to 1, it's active. Read/write.

When timeout mechanism is enabled and IREDIR_HOST_REQ interupt is triggered, a hidden counter starts counting down. If IREDIR_TIMEOUT cycles go by without the interrupt being acked, the redirect hw goes to HOST state, the interrupt is cleared, and HOST_REQ_TIMEOUT error is triggered.

The redirect hw errors will trigger the IREDIR_ERR interrupt, which is connected to SUBINTR bit #5. The registers involved are:

MMIO 0x698 / I[0x1a600]: IREDIR_ERR_DETAIL Read-only, shows detailed error status. All bits are auto-cleared when IREDIR_ERR_INTR is cleared

- bit 0: HOST_REQ_TIMEOUT - set when the IREDIR_HOST_REQ interrupt times out
- bit 4: HOST_REQ_REDUNDANT - set when HOST_REQ is poked in IREDIR_TRIGGER while the hw is already in HOST state
- bit 12: DAEMON_REDUNDANT - set when HOST is poked in IREDIR_TRIGGER while the hw is already in DAEMON state
- bit 12: HOST_REDUNDANT - set when HOST is poked in IREDIR_TRIGGER while the hw is already in HOST state

MMIO 0x69c / I[0x1a700]: IREDIR_ERR_INTR The status register for IREDIR_ERR interrupt. Only bit 0 is valid. Set when any of the 4 errors happens, cleared [along with all IREDIR_ERR_DETAIL bits] when 1 is written to bit 0. When this and IREDIR_ERR_INTR_EN are both set, the IREDIR_ERR [#5] second-level interrupt line to SUBINTR is asserted.

MMIO 0x6a0 / I[0x1a800]: IREDIR_ERR_INTR_EN The enable register for IREDIR_ERR interrupt. Only bit 0 is valid.

The interrupt redirection circuitry also exports the following signals to PCOUNTER:

- IREDIR_STATUS: current redirect hw status, like the IREDIR_STATUS reg.
- IREDIR_HOST_REQ: 1 if the IREDIR_HOST_REQ [SUBINTR #6] interrupt is pending
- IREDIR_TRIGGER_DAEMON: pulses for 1 cycle whenever INTR_TRIGGER.DAEMON is written as 1, whether it results in an error or not
- IREDIR_TRIGGER_HOST: pulses for 1 cycle whenever INTR_TRIGGER.HOST is written as 1, whether it results in an error or not
- IREDIR_PMC: 1 if the PMC INTR_HOST line is active and directed to DAEMON [ie. mirrors falcon interrupt #15 input]
- IREDIR_INTR: 1 if any IREDIR interrupt is active - IREDIR_HOST_REQ, IREDIR_ERR, or IREDIR_PMC. IREDIR_ERR does not count if IREDIR_ERR_INTR_EN is not set.

PTHERM interface

PDAEMON can access all PTHERM registers directly, without having to go through the generic MMIO access functionality. The THERM range in the PDAEMON register space is mapped straight to PTHERM MMIO register range.

On GT215:GF119, PTHERM registers are mapped into the I[] space at addresses 0x20000:0x40000, with addresses being shifted left by 6 bits wrt their address in PTHERM - PTHERM register 0x20000+x would be visible at I[0x20000 + x * 0x40] by falcon, or at 0x10a800+x in MMIO [assuming it wouldn't fall into the reserved 0x10afe0:0x10b000 range]. On GF119+, the PTHERM registers are instead mapped into the I[] space at addresses 0x1000:0x1800, without shifting - PTHERM reg 0x20000+x is visible at I[0x1000+x]. On GF119+, the alias area is not visible via MMIO [just access PTHERM registers directly instead].

Reads to the PTHERM-mapped area will always perform 32-bit reads to the corresponding PTHERM regs. Writes, however, have their byte enable mask controlled via a PDAEMON register, enabling writes with sizes other than 32-bit:

MMIO 0x5f4 / I[0x17d00]: THERM_BYTE_MASK Read/write, only low 4 bits are valid, initialised to 0xf on reset. Selects the byte mask to use when writing the THERM range. Bit i corresponds to bits i*8..i*8+7 of the written 32-bit value.

The PTHERM access circuitry also exports a signal to PCOUNTER:

- THERM_ACCESS_BUSY: 1 while a THERM range read/write is in progress - will light up for a dozen or so cycles per access, depending on relative clock speeds.

In addition to direct register access to PTHERM, PDAEMON also has direct access to PTHERM interrupts - falcon interrupt #12 [THERM] comes from PTHERM interrupt aggregator. PTHERM subinterrupts can be individually assigned for PMC or PDAEMON delivery - see ptherm-intr for more information.

Idle counters

Contents

Introduction

PDAEMON's role is mostly about power management. One of the most effective way of lowering the power consumption is to lower the voltage at which the processor is powered at. Lowering the voltage is also likely to require lowering the clocks of the engines powered by this power domain. Lowering the clocks lowers the performance which means it can only be done to engines that are under-utilized. This technique is called Dynamic Voltage/Frequency Scaling (DVFS) and requires being able to read the activity-level/business of the engines clocked with every clock domains.

PDAEMON could use PCOUNTER to read the business of the engines it needs to reclock but that would be a waste of counters. Indeed, contrarily to PCOUNTER that needs to be able to count events, the business of an engine can be polled at any frequency depending on the level of accuracy wanted. Moreover, doing the configuration of PCOUNTER both in the host driver and in PDAEMON would likely require some un-wanted synchronization.

This is most likely why some counters were added to PDAEMON. Those counters are polling idle signals coming from the monitored engines. A signal is a binary value that equals 1 when the associated engine is idle, and 0 if it is active.

Todo

check the frequency at which PDAEMON is polling

MMIO Registers

On GT215:GF100, there were 4 counters while on GF100+, there are 8 of them. Each counter is composed of 3 registers, the mask, the mode and the actual count. There are two counting modes, the first one is to increment the counter every time every bit of COUNTER_SIGNALS selected by the mask are set. The other mode only increments when all the selected bits are cleared. It is possible to set both modes at the same time which results in incrementing at every clock cycle. This mode is interesting because it allows dedicating a counter to time-keeping which eases translating the other counters' values to an idling percentage. This allows for aperiodical polling on the counters without needing to store the last polling time.

The counters are not double-buffered and are independent. This means every counters need to be read then reset at roughly the same time if synchronization between the counters is required. Resetting the counter is done by setting bit 31 of COUNTER_COUNT.

MMIO 0x500 / I[0x14000]: COUNTER_SIGNALS Read-only. Bitfield with each bit indicating the instantenous state of the associated engines/blocks. When the bit is set, the engine/block is idle, when it is cleared, the engine/block is active.

- bit 0: GR_IDLE
- bit 4: PVLD_IDLE
- bit 5: PPDEC_IDLE
- bit 6: PPPP_IDLE
- bit 7: MC_IDLE [GF100-]
- bit 8: MC_IDLE [GT215:GF100]
- bit 19: PCOPY0_IDLE
- bit 20: PCOPY1_IDLE [GF100-]
- bit 21: PCOPY2_IDLE [GK104-]

MMIO 0x504+i*10 / I[0x14100+i*0x400]: COUNTER_MASK The mask that will be applied on COUNTER_SIGNALS before applying the logic set by COUNTER_MODE.

MMIO 0x508+i*10 / I[0x14100+i*0x400]: COUNTER_COUNT

- bit 0-30: COUNT
- bit 31: CLEAR_TRIGGER : Write-only, resets the counter.

MMIO 0x50c+i*10 / I[0x14300+i*0x400]: COUNTER_MODE

- bit 0: INCR_IF_ALL : Increment the counter if all the masked bits are set
- bit 1: INCR_IF_NOT_ALL : Increment the counter if all the masked bits are cleared
- bit 2: UNK2 [GF119-]

General MMIO register access

PDAEMON can access the whole MMIO range through the IO space.

To read from a MMIO address, poke the address into MMIO_ADDR then trigger a read by poking 0x100f1 to MMIO_CTRL. Wait for MMIO_CTRL's bits 12-14 to be cleared then read the value from MMIO_VALUE.

To write to a MMIO address, poke the address into MMIO_ADDR, poke the value to be written into MMIO_VALUE then trigger a write by poking 0x100f2 to MMIO_CTRL. Wait for MMIO_CTRL's bits 12-14 to be cleared if you want to make sure the write has been completed.

Accessing an unexisting address will set MMIO_CTRL's bit 13 after MMIO_TIMEOUT cycles have passed.

GF119 introduced the possibility to choose from which access point should the MMIO request be sent. ROOT can access everything, IBUS accesses everything minus PMC, PBUS, PFIFO, PPCI and a few other top-level MMIO range. On GF119+, accessing an un-existing address with the ROOT access point can lead to a hard-lock. XXX: What's the point of this feature?

It is possible to get an interrupt when an error occurs by poking 1 to MMIO_INTR_EN. The interrupt will be fired on line 11. The error is described in MMIO_ERR.

MMIO 0x7a0 / I[0x1e800]: MMIO_ADDR Specifies the MMIO address that will be written to/read from by MMIO_CTRL.

On GT215:GF119, this register only contains the address to be accessed.

On GF119, this register became a bitfield: bits 0-25: ADDR bit 27: ACCESS_POINT

0: ROOT 1: IBUS

MMIO 0x7a4 / I[0x1e900]: MMIO_VALUE The value that will be written to / is read from MMIO_ADDR when an operation is triggered by MMIO_CTRL.

MMIO 0x7a8 / I[0x1e900]: MMIO_TIMEOUT Specifies the timeout for MMIO access. XXX: Clock source? PDAEMON's core clock, PTIMER's, Host's?

MMIO 0x7ac / I[0x1eb00]: MMIO_CTRL Process the MMIO request with given params (MMIO_ADDR, MMIO_VALUE). bits 0-1: request

0: XXX 1: read 2: write 3: XXX

bits 4-7: BYTE_MASK bit 12: BUSY [RO] bit 13: TIMEOUT [RO] bit 14: FAULT [RO] bit 16: TRIGGER

MMIO 0x7b0 / I[0x1ec00] [MMIO_ERR]

Specifies the MMIO error status:

- TIMEOUT: ROOT/IBUS has not answered PDAEMON's request
- CMD_WHILE_BUSY: a request has been fired while being busy
- WRITE: set if the request was a write, cleared if it was a read
- FAULT: No engine answered ROOT/IBUS's request

On GT215:GF119, clearing MMIO_INTR's bit 0 will also clear MMIO_ERR. On GF119+, clearing MMIO_ERR is done by poking 0xffffffff.

GT215:GF100: bit 0: TIMEOUT bit 1: CMD_WHILE_BUSY bit 2: WRITE bits 3-31: ADDR

GF100:GF119: bit 0: TIMEOUT bit 1: CMD_WHILE_BUSY bit 2: WRITE bits 3-30: ADDR bit 31: FAULT

GF119+: bit 0: TIMEOUT_ROOT bit 1: TIMEOUT_IBUS bit 2: CMD_WHILE_BUSY bit 3: WRITE bits 4-29: ADDR bit 30: FAULT_ROOT bit 31: FAULT_IBUS

MMIO 0x7b4 / I[0x1ed00] [MMIO_INTR] Specifies which MMIO interrupts are active. Clear the associated bit to ACK. bit 0: ERR

Clearing this bit will also clear MMIO_ERR on GT215:GF119.

MMIO 0x7b8 / I[0x1ee00] [MMIO_INTR_EN] Specifies which MMIO interrupts are enabled. Interrupts will be fired on SUBINTR #4. bit 0: ERR

Engine power gating

Todo

write me

Input/output signals

Contents

Todo

write me

Introduction

Todo

write me

Interrupts

Todo

write me

Introduction

PDAEMON is a falcon-based engine introduced on GT215. Its main purpose is autonomous power and thermal management, but it can be used to oversee any part of GPU operation. The PDAEMON has many dedicated connections to various parts of the GPU.

The PDAEMON is made of:

- *a falcon microprocessor core*
- *standard falcon memory interface unit*
- *a simple channel load interface, replacing the usual PFIFO interface*
- *various means of communication betwen falcon and host*
- *engine power gating controllers for the PFIFO-connected engines*
- *"idle" signals from various engines and associated idle counters*
- *misc simple input/output signals to various engines, with interrupt capability*
- *a oneshot/periodic timer, using daemon clock or PTIMER as clock source*
- *PMC interrupt redirection circuitry*
- *indirect MMIO access circuitry*
- *direct interface to all PTHERM registers*
- *CRC computation hardware*

Todo

and unknown stuff.

There are 5 revisions of PDAEMON:

- v0: GT215:MCP89 - the original revision
- v1: MCP89:GF100 - added a third instance of power gating controller for PVCOMP engine
- v2: GF100:GF119 - removed PVCOMP support, added second set of input/output signals and ???
- v3: GF119:GK104 - changed I[] space layout, added ???
- v4: GK104- - a new version of engine power gating controller and ???

Todo

figure out additions

Todo

this file deals mostly with GT215 version now

2.5.3 NV43:G80 thermal monitoring

Contents

Introduction

THERM is an area present in PBUS on NV43:G80 GPUs. This area is reponsible for temperature monitoring, probably on low-end and middle-range GPUs since high-end cards have been using LM89/ADT7473 for a long time. Beside some configuration knobs, THERM can generate IRQs to the HOST when the temperature goes over a configurable ALARM threshold or outside a configurable temperature range. This range has been replaced by PTHERM on G80+ GPUs.

THERM's MMIO range is 0x15b0:0x15c0. There are two major variants of this range:

- NV43:G70
- G70:G80

MMIO register list

Address	Present on	Name	Description
0x0015b0	all	CFG0	sensor enable / IRQ enable / ALARM configuration
0x0015b4	all	STATUS	sensor state / ALARM state / ADC rate configuration
0x0015b8	non-IGP	CFG1	misc. configuration
0x0015bc	all	TEMP_RANGE	LOW and HIGH temperature thresholds

MMIO 0x15b0: CFG0 [NV43:G70]

- bits 0-7: ALARM_HIGH
- bits 16-23: SENSOR_OFFSET (signed integer)
- bit 24: DISABLE
- bit 28: ALARM_INTR_EN

MMIO 0x15b0: CFG0 [G70:G80]

- bits 0-13: ALARM_HIGH
- bits 16-29: SENSOR_OFFSET (signed integer)
- bit 30: DISABLE
- bit 31: ENABLE

MMIO 0x15b4: STATUS [NV43:G70]

- bits 0-7: SENSOR_RAW
- bit 8: ALARM_HIGH

- bits 25-31: ADC_CLOCK_XXX

Todo

figure out what divisors are available

MMIO 0x15b4: STATUS [G70:G80]

- bits 0-13: SENSOR_RAW
- bit 16: ALARM_HIGH
- bits 26-31: ADC_CLOCK_DIV The division is stored right-shifted 4. The possible division values range from 32 to 2016 with the possibility to completely bypass the divider.

MMIO 0x15b8: CFG1 [NV43:G70]

- bit 17: ADC_PAUSE
- bit 23: CONNECT_SENSOR

MMIO 0x15bc: TEMP_RANGE [NV43:G70]

- bits 0-7: LOW
- bits 8-15: HIGH

MMIO 0x15bc: TEMP_RANGE [G70:G80]

- bits 0-13: LOW
- bits 16-29: HIGH

The ADC clock

The source clock for THERM's ADC is:

- NV43:G70: the host clock
- G70:G80: constant (most likely hclck)

(most likely, since the rate doesn't change when I change the HOST clock)

Before reaching the ADC, the clock source is divided by a fixed divider of 1024 and then by ADC_CLOCK_DIV.

MMIO 0x15b4: STATUS [NV43:G70]

- bits 25-31: ADC_CLOCK_DIV

Todo

figure out what divisors are available

MMIO 0x15b4: STATUS [G70:G80]

- bits 26-31: ADC_CLOCK_DIV The division is stored right-shifted 4. The possible division values range from 32 to 2016 with the possibility to completely bypass the divider.

The final ADC clock is:

ADC_clock = source_clock / ADC_CLOCK_DIV

The accuracy of the reading greatly depends on the ADC clock. A clock too fast will produce a lot of noise. A clock too low may actually produce an offseted value. The ADC clock rate under 10 kHz is advised, based on limited testing on a G73.

Todo

Make sure this clock range is safe on all cards

Anyway, it seems like it is clocked at an acceptable frequency at boot time, so, no need to worry too much about it.

Reading temperature

Temperature is read from:

MMIO 0x15b4: STATUS [NV43:G70] bits 0-7: SENSOR_RAW

MMIO 0x15b4: STATUS [G70:G80] bits 0-13: SENSOR_RAW

SENSOR_RAW is the result of the (signed) addition of the actual value read by the ADC and SENSOR_OFFSET:

MMIO 0x15b0: CFG0 [NV43:G70]

- bits 16-23: SENSOR_OFFSET signed

MMIO 0x15b0: CFG0 [G70:G80]

- bits 16-29: SENSOR_OFFSET signed

Starting temperature readouts requires to flip a few switches that are GPU-dependent:

MMIO 0x15b0: CFG0 [NV43:G70]

- bit 24: DISABLE

MMIO 0x15b0: CFG0 [G70:G80]

- bit 30: DISABLE - mutually exclusive with ENABLE
- bit 31: ENABLE - mutually exclusive with DISABLE

MMIO 0x15b8: CFG1 [NV43:G70]

- bit 17: ADC_PAUSE
- bit 23: CONNECT_SENSOR

Both DISABLE and ADC_PAUSE should be clear. ENABLE and CONNECT_SENSOR should be set.

Todo

There may be other switches.

Setting up thresholds and interrupts

Alarm

THERM features the ability to set up an alarm that will trigger interrupt PBUS #16 when SENSOR_RAW > ALARM_HIGH. NV43-47 GPUs require ALARM_INTR_EN to be set in order to get the IRQ. You may need to

set bits 0x40001 in 0x15a0 and 1 in 0x15a4. Their purpose has not been understood yet even though they may be releated to automatic downclocking.

MMIO 0x15b0: CFG0 [NV43:G70]

- bits 0-7: ALARM_HIGH
- bit 28: ALARM_INTR_EN

MMIO 0x15b0: CFG0 [G70:G80]

- bits 0-13: ALARM_HIGH

When SENSOR_RAW > ALARM_HIGH, STATUS.ALARM_HIGH is set.

MMIO 0x15b4: STATUS [NV43:G70]

- bit 8: ALARM_HIGH

MMIO 0x15b4: STATUS [G70:G80]

- bit 16: ALARM_HIGH

STATUS.ALARM_HIGH is unset as soon as SENSOR_RAW < ALARM_HIGH, without any hysteresis cycle.

Temperature range

THERM can check that temperature is inside a range. When the temperature goes outside this range, an interrupt is sent. The range is defined in the register TEMP_RANGE where the thresholds LOW and HIGH are set.

MMIO 0x15bc: TEMP_RANGE [NV43:G70]

- bits 0-7: LOW
- bits 8-15: HIGH

MMIO 0x15bc: TEMP_RANGE [G70:G80]

- bits 0-13: LOW
- bits 16-29: HIGH

When SENSOR_RAW < TEMP_RANGE.LOW, interrupt PBUS #17 is sent. When SENSOR_RAW > TEMP_RANGE.HIGH, interrupt PBUS #18 is sent.

There are no hyteresis cycles on these thresholds.

Extended configuration

Todo

Document reg 15b8

2.6 GPU external device I/O units

Contents:

2.6.1 G80:GF119 GPIO lines

Contents

Todo

write me

Introduction

Todo

write me

Interrupts

Todo

write me

G80 GPIO NVIO specials

This list applies to G80.

Line	Output	Input
0	PWM_0	
1	-	
2	-	
3	tag 0x42?	
4	SLI_SENSE_0?	
5	-	
6	-	
7	-	PTHERM_INPUT_0
8	-	PTHERM_INPUT_2
9	related to e1bc and PTHERM?	
10	-	
11	SLI_SENSE_1?	
12	tag 0x43?	
13	tag 0x0f?	
14	-	

G84 GPIO NVIO specials

This list applies to G84:G94.

Line	Output	Input
4	PWM_0	
8	THERM_SHUTDOWN?	PTHERM_INPUT_0
9	PWM_1	PTHERM_INPUT_1
11	SLI_SENSE_0?	
12		PTHERM_INPUT_2
13	tag 0x0f?	
14	SLI_SENSE_1?	

G94 GPIO NVIO specials

This list applies to G94:GT215.

Line	Output	Input
1		AUXCH_HPD_0
4	PWM_0	
8	THERM_SHUTDOWN?	PTHERM_INPUT_0
9	PWM_1	PTHERM_INPUT_1
12		PTHERM_INPUT_2
15		AUXCH_HPD_2
20		AUXCH_HPD_1
21		AUXCH_HPD_3

GT215 GPIO NVIO specials

This list applies to GT215:GF119.

Line	Output	Input
1		AUXCH_HPD_0
3	SLI_SENSE?	
8	THERM_SHUTDOWN?	PTHERM_INPUT_0
9	PWM_1	PTHERM_INPUT_1
11	SLI_SENSE?	
12		PTHERM_INPUT_2
15		AUXCH_HPD_2
16	PWM_0	
19		AUXCH_HPD_1
21		AUXCH_HPD_3
22	tag 0x42?	
23	tag 0x0f?	
[any]		FAN_TACH

2.7 Memory access and structure

Contents:

2.7.1 Memory structure

Contents

Introduction

While DRAM is often treated as a flat array of bytes, its internal structure is far more complicated. A good understanding of it is necessary for high-performance applications like GPUs.

Looking roughly from the bottom up, VRAM is made of:

1. Memory planes of R rows by C columns, with each cell being one bit
2. Memory banks made of 32, 64, or 128 memory planes used in parallel - the planes are usually spread across several chips, with one chip containing 16 or 32 memory planes
3. Memory ranks made of several [2, 4 or 8] memory banks wired together and selected by address bits - all banks for a given memory plane reside in the same chip
4. Memory subpartitions made of one or two memory ranks wired together and selected by chip select wires - ranks behave similarly to banks, but don't have to have uniform geometry, and are in separate chips
5. Memory partitions made of one or two somewhat independent subpartitions
6. The whole VRAM, made of several [1-8] memory partitions

Memory planes and banks

The most basic unit of DRAM is a memory plane, which is a 2d array of bits organised in so-called columns and rows:

```
      column
row   0  1  2  3  4  5  6  7
0     X  X  X  X  X  X  X  X
1     X  X  X  X  X  X  X  X
2     X  X  X  X  X  X  X  X
3     X  X  X  X  X  X  X  X
4     X  X  X  X  X  X  X  X
5     X  X  X  X  X  X  X  X
6     X  X  X  X  X  X  X  X
7     X  X  X  X  X  X  X  X

buf   X  X  X  X  X  X  X  X
```

A memory plane contains a buffer, which holds a whole row. Internally, DRAM is read/written in row units via the buffer. This has several consequences:

- before a bit can be operated on, its row must be loaded into the buffer, which is slow
- after a row is done with, it needs to be written back to the memory array, which is also slow
- accessing a new row is thus slow, and even slower when there already is an active row
- it's often useful to preemptively close a row after some inactivity time - such operation is called "precharging" a bank
- different columns in the same row, however, can be accessed quickly

Since loading column address itself takes more time than actually accessing a bit in the active buffer, DRAM is accessed in bursts - a series of accesses to 1-8 neighbouring bits in the active row. Usually all bits in a burst have to be located in a single aligned 8-bit group.

The amount of rows and columns in memory plane is always a power of two, and is measured by the count of row selection and column selection bits [ie. log2 of the row/column count]. There are typically 8-10 column bits and 10-14 row bits.

The memory planes are organised in banks - groups of some power of two number of memory planes. The memory planes are wired in parallel, sharing the address and control wires, with only the data / data enable wires separate. This effectively makes a memory bank like a memory plane that's composed of 32/64/128-bit memory cells instead of single bits - all the rules that apply to a plane still apply to a bank, except larger units than a bit are operated on.

A single memory chip usually contains 16 or 32 memory planes for a single bank, thus several chips are often wired together to make wider banks.

Memory banks, ranks, and subpartitions

A memory chip contains several [2, 4, or 8] banks, using the same data wires and multiplexed via bank select wires. While switching between banks is slightly slower than switching between columns in a row, it's much faster than switching between rows in the same bank.

A memory rank is thus made of *(MEMORY_CELL_SIZE / MEMORY_CELL_SIZE_PER_CHIP)* memory chips.

One or two memory ranks connected via common wires [including data] except a chip select wire make up a memory subpartition. Switching between ranks has basically the same performance consequences as switching between banks in a rank - the only differences are the physical implementation and the possibility of using different amount of row selection bits for each rank [though bank count and column count have to match].

The consequences of existence of several banks/ranks:

- it's important to ensure that data accessed together belongs to either the same row, or to different banks [to avoid row switching]
- tiled memory layouts are designed so that a tile corresponds roughly to a row, and neighbouring tiles never share a bank

Memory partitions and subpartitions

A memory subpartition has its own DRAM controller on the GPU. 1 or 2 subpartitions make a memory partition, which is a fairly independent entity with its own memory access queue, own ZROP and CROP units, and own L2 cache on later cards. All memory partitions taken together with the crossbar logic make up the entire VRAM logic for a GPU.

All subpartitions in a partition have to be configured identically. Partitions in a GPU are usually configured identically, but don't have to on newer cards.

The consequences of subpartition/partition existence:

- like banks, different partitions may be utilised to avoid row conflicts for related data
- unlike banks, bandwidth suffers if (sub)partitions are not utilised equally - load balancing is thus very important

Memory addressing

While memory addressing is highly dependent on GPU family, the basic approach is outlined here.

The bits of a memory address are, in sequence, assigned to:

- identifying a byte inside a memory cell - since whole cells always have to be accessed anyway
- several column selection bits, to allow for a burst
- partition/subpartition selection - in low bits to ensure good load balancing, but not too low to keep relatively large tiles in a single partition for ROP's benefit
- remaining column selection bits
- all/most of bank selection bits, sometimes a rank selection bit - so that immediately neighbouring addresses never cause a row conflict
- row bits
- remaining bank bit or rank bit - effectively allows splitting VRAM into two areas, placing color buffer in one and zeta buffer in the other, so that there are never row conflicts between them

2.7.2 NV1:G80 surface formats

Contents

Todo

write me

Introduction

Todo

write me

2.7.3 NV3 VRAM structure and usage

Contents

Todo

write me

Introduction

Todo

write me

2.7.4 NV3 DMA objects

Contents

Todo

write me

Introduction

Todo

write me

2.7.5 NV4:G80 DMA objects

Contents

Todo

write me

Introduction

Todo

write me

2.7.6 NV44 host memory interface

Contents

Todo

write me

Introduction

Todo

write me

MMIO registers

Todo

write me

2.7.7 G80 surface formats

Contents

Introduction

This file deals with G80+ cards only. For older cards, see *NV1:G80 surface formats*.

A "surface" is a 2d or 3d array of elements. Surfaces are used for image storage, and can be bound to at least the following slots on the engines:

- m2mf input and output buffers
- 2d source and destination surfaces
- 3d/compute texture units: the textures
- 3d color render targets
- 3d zeta render target
- compute g[] spaces [G80:GF100]
- 3d/compute image units [GF100+]
- PCOPY input and output buffers
- PDISPLAY: the framebuffer

Todo

vdec stuff

Todo

GF100 ZCULL?

Surfaces on G80+ cards come in two types: pitch and blocklinear. Pitch surfaces have a simple format, but they're are limited to 2 dimensions only, don't support arrays nor mipmapping when used as textures, cannot be used for zeta buffers, and have lower performance than blocklinear textures. Blocklinear surfaces can have up to three dimensions, can be put into arrays and be mipmapped, and use custom element arrangement in memory. However, blocklinear surfaces need to be placed in memory area with special storage type, depending on the surface format.

Blocklinear surfaces have two main levels of element rearrangement: high-level and low-level. Low-level rearrangement is quite complicated, depends on surface's storage type, and is hidden by the VM subsystem - if the surface is accessed through VM with properly set storage type, only the high-level rearrangement is visible. Thus the low-level rearrangement can only be seen when accessing blocklinear system RAM directly from CPU, or accessing blocklinear VRAM with storage type set to 0. Also, low-level rearrangement for VRAM uses several tricks to distribute load evenly across memory partitions, while rearrangement for system RAM skips them and merely reorders elements inside a gob. High-level rearrangement, otoh, is relatively simple, and always visible to the user - its knowledge is needed to calculate address of a given element, or to calculate the memory size of a surface.

Surface elements

A basic unit of surface is an "element", which can be 1, 2, 4, 8, or 16 bytes long. element type is vital in selecting the proper compressed storage type for a surface. For most surface formats, an element means simply a sample. This is different for surfaces storing compressed textures - the elements are compressed blocks. Also, it's different for bitmap textures - in these, an element is a 64-bit word containing 8x8 block of samples.

While texture, RT, and 2d bindings deal only with surface elements, they're ignored by some other binding points, like PCOPY and m2mf - in these, the element size is ignored, and the surface is treated as an array of bytes. That is, a 16x16 surface of 4-byte elements is treated as a 64x16 surface of bytes.

Pitch surfaces

A pitch surface is a 2d array of elements, where each row is contiguous in memory, and each row starts at a fixed distance from start of the previous one. This distance is the surface's "pitch". Pitch surfaces always use storage type 0 [pitch].

The attributes defining a pitch surface are:

- address: 40-bit VM address, aligned to 64 bytes
- pitch: distance between subsequent rows in bytes - needs to be a multiple of 64
- element size: implied by format, or defaulting to 1 if binding point is byte-oriented
- width: surface width in elements, only used when bounds checking / size information is needed
- height: surface height in elements, only used when bounds checking / size information is needed

---

Todo

check pitch, width, height min/max values. this may depend on binding point. check if 64 byte alignment still holds on GF100.

The address of element (x,y) is:

```
address + pitch * y + elem_size * x
```

Or, alternatively, the address of byte (x,y) is:

```
address + pitch * y + x
```

Blocklinear surfaces

A blocklinear surface is a 3d array of elements, stored in memory in units called "gobs" and "blocks". There are two levels of tiling. The lower-level unit is called a "gob" and has a fixed size. This size is 64 bytes × 4 × 1 on G80:GF100 cards, 64 bytes × 8 × 1 for GF100+ cards. The higher-level unit is called a "block", and is of variable size between 1×1×1 and 32×32×32 gobs.

The attributes defining a blocklinear surface are:

- address: 40-bit VM address, aligned to gob size [0x100 bytes on G80:GF100, 0x200 bytes on GF100]
- block width: 0-5, log2 of gobs per block in x dimension
- block height: 0-5, log2 of gobs per block in y dimension
- block depth: 0-5, log2 of gobs per block in z dimension
- element size: implied by format, or defaulting to 1 if the binding point is byte-oriented
- width: surface width [size in x dimension] in elements
- height: surface height [size in y dimension] in elements
- depth: surface depth [size in z dimension] in elements

Todo

check bounduaries on them all, check tiling on GF100.

Todo

PCOPY surfaces with weird gob size

It should be noted that some limits on these parameters are to some extent specific to the binding point. In particular, block width greater than 0 is only supported by the render targets and texture units, with render targets only supporting 0 and 1. block height of 0-5 can be safely used with all blocklinear surface binding points, and block depth of 0-5 can be used with binding points other than G80 g[] spaces, which only support 0.

The blocklinear format works as follows:

First, the block size is computed. This computation depends on the binding point: some binding points clamp the effective block size in a given dimension to the smallest size that would cover the whole surfaces, some do not. The ones that do are called "auto-sizing" binding points. One of such binding ports where it's important is the texture unit: since all mipmap levels of a texture use a single "block size" field in TIC, the auto-sizing is needed to ensure that small mipmaps of a large surface don't use needlessly large blocks. Pseudocode:

```
bytes_per_gob_x = 64;
if (gpu < GF100)
    bytes_per_gob_y = 4;
else
    bytes_per_gob_y = 8;
bytes_per_gob = 1;
eff_block_width = block_width;
eff_block_height = block_height;
eff_block_depth = block_depth;
if (auto_sizing) {
    while (eff_block_width > 0 && (bytes_per_gob_x << (eff_block_width - 1)) >= width * element_size)
        eff_block_width--;
    while (eff_block_height > 0 && (bytes_per_gob_y << (eff_block_height - 1)) >= height)
```

```
        eff_block_height--;
    while (eff_block_depth > 0 && (bytes_per_gob_z << (eff_block_depth - 1)) >= depth)
        eff_block_depth--;
}
gobs_per_block_x = 1 << eff_block_width;
gobs_per_block_y = 1 << eff_block_height;
gobs_per_block_z = 1 << eff_block_depth;
bytes_per_block_x = bytes_per_gob_x * gobs_per_block_x;
bytes_per_block_y = bytes_per_gob_y * gobs_per_block_y;
bytes_per_block_z = bytes_per_gob_z * gobs_per_block_z;
elements_per_block_x = bytes_per_block_x / element_size;
gob_bytes = bytes_per_gob_x * bytes_per_gob_y * bytes_per_gob_z;
block_gobs = gobs_per_bigtils_x * gobs_per_block_y * gobs_per_block_z;
block_bytes = gob_bytes * block_gobs;
```

Due to the auto-sizing being present on some binding points, it's a bad idea to use surfaces that have block size at least two times bigger than the actual surface - they'll be unusable on these binding points [and waste a lot of memory anyway].

Once block size is known, the geometry and size of the surface can be determined. A surface is first broken down into blocks. Each block convers a contiguous elements_per_block_x × bytes_per_block_y × bytes_per_block_z aligned subarea of the surface. If the surface size is not a multiple of the block size in any dimension, the size is aligned up for surface layout purposes and the remaining space is unused. The blocks making up a surface are stored sequentially in memory first in x direction, then in y direction, then in z direction:

```
blocks_per_surface_x = ceil(width * element_size / bytes_per_block_x);
blocks_per_surface_y = ceil(height / bytes_per_block_y);
blocks_per_surface_z = ceil(depth / bytes_per_block_z);
surface_blocks = blocks_per_surface_x * blocks_per_surface_y * blocks_per_surface_z;
// total bytes in surface - surface resides at addresses [address, address+surface_bytes)
surface_bytes = surface_blocks * block_bytes;
block_address = address + floor(x_coord * element_size / bytes_per_block_x) * block_bytes
            + floor(y_coord / bytes_per_block_y) * block_bytes * blocks_per_surface_x;
            + floor(z_coord / bytes_per_block_z) * block_bytes * blocks_per_surface_x * blocks_per_su
x_coord_in_block = (x_coord * element_size) % bytes_per_block_x;
y_coord_in_block = y_coord % bytes_per_block_y;
z_coord_in_block = z_coord % bytes_per_block_z;
```

Like blocks in the surface, gobs inside a block are stored ordered first by x coord, then by y coord, then by z coord:

```
gob_address = block_address
        + floor(x_coord_in_block / bytes_per_gob_x) * gob_bytes
        + floor(y_coord_in_block / bytes_per_gob_y) * gob_bytes * gobs_per_block_x
        + z_coord_in_block * gob_bytes * gobs_per_block_x * gobs_per_block_y; // bytes_per_gob_z alwa
x_coord_in_gob = x_coord_in_block % bytes_per_gob_x;
y_coord_in_gob = y_coord_in_block % bytes_per_gob_y;
```

The elements inside a gob are likewise stored ordered first by x coordinate, and then by y:

```
element_address = gob_address + x_coord_in_gob + y_coord_in_gob * bytes_per_gob_x;
```

Note that the above is the higher-level rearrangement only - the element address resulting from the above pseudocode is the address that user would see by looking through the card's VM subsystem. The lower-level rearrangement is storage type dependent, invisible to the user, and will be covered below.

As an example, let's take a 13 × 17 × 3 surface with element size of 16 bytes, block width of 1, block height of 1, and block depth of 1. Further, the card is assumed to be G80. The surface will be located in memory the following way:

- block size in bytes = 0x800 bytes

- block width: 128 bytes / 8 elements
- block height: 8
- block depth: 2
- surface width in blocks: 2
- surface height in blocks: 3
- surface depth in blocks: 2
- surface memory size: 0x6000 bytes

```
| - x element bounduary
|| - x gob bounduary
||| - x block bounduary
[no line] - y element bounduary
--- - y gob bounduary
=== - y block bounduary

z == 0:
 x -->
y+--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
||  |  0 |  1 |  2 |  3 ||  4 |  5 |  6 |  7 |||  8 |  9 | 10 | 11 || 12 |
|+--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
V| 0|0000|0010|0020|0030||0100|0110|0120|0130|||0800|0810|0820|0830||0900|
 | 1|0040|0050|0060|0070||0140|0150|0160|0170|||0840|0850|0860|0870||0940|
 | 2|0080|0090|00a0|00b0||0180|0190|01a0|01b0|||0880|0890|08a0|08b0||0980|
 | 3|00c0|00d0|00e0|00f0||01c0|01d0|01e0|01f0|||08c0|08d0|08e0|08f0||09c0|
 +--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
 | 4|0200|0210|0220|0230||0300|0310|0320|0330|||0a00|0a10|0a20|0a30||0b00|
 | 5|0240|0250|0260|0270||0340|0350|0360|0370|||0a40|0a50|0a60|0a70||0b40|
 | 6|0280|0290|02a0|02b0||0380|0390|03a0|03b0|||0a80|0a90|0aa0|0ab0||0b80|
 | 7|02c0|02d0|02e0|02f0||03c0|03d0|03e0|03f0|||0ac0|0ad0|0ae0|0af0||0bc0|
 +==+====+====+====+====++====+====+====+====+++====+====+====+====++====+
 | 8|1000|1010|1020|1030||1100|1110|1120|1130|||1800|1810|1820|1830||1900|
 | 9|1040|1050|1060|1070||1140|1150|1160|1170|||1840|1850|1860|1870||1940|
 |10|1080|1090|10a0|10b0||1180|1190|11a0|11b0|||1880|1890|18a0|18b0||1980|
 |11|10c0|10d0|10e0|10f0||11c0|11d0|11e0|11f0|||18c0|18d0|18e0|18f0||19c0|
 +--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
 |12|1200|1210|1220|1230||1300|1310|1320|1330|||1a00|1a10|1a20|1a30||1b00|
 |13|1240|1250|1260|1270||1340|1350|1360|1370|||1a40|1a50|1a60|1a70||1b40|
 |14|1280|1290|12a0|12b0||1380|1390|13a0|13b0|||1a80|1a90|1aa0|1ab0||1b80|
 |15|12c0|12d0|12e0|12f0||13c0|13d0|13e0|13f0|||1ac0|1ad0|1ae0|1af0||1bc0|
 +==+====+====+====+====++====+====+====+====+++====+====+====+====++====+
 |16|2000|2010|2020|2030||2100|2110|2120|2130|||2800|2810|2820|2830||2900|
 +--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
z == 1:
 x -->
y+--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
||  |  0 |  1 |  2 |  3 ||  4 |  5 |  6 |  7 |||  8 |  9 | 10 | 11 || 12 |
|+--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
V| 0|0400|0410|0420|0430||0500|0510|0520|0530|||0c00|0c10|0c20|0c30||0d00|
 | 1|0440|0450|0460|0470||0540|0550|0560|0570|||0c40|0c50|0c60|0c70||0d40|
 | 2|0480|0490|04a0|04b0||0580|0590|05a0|05b0|||0c80|0c90|0ca0|0cb0||0d80|
 | 3|04c0|04d0|04e0|04f0||05c0|05d0|05e0|05f0|||0cc0|0cd0|0ce0|0cf0||0dc0|
 +--+----+----+----+----++----+----+----+----+++----+----+----+----++----+
 | 4|0600|0610|0620|0630||0700|0710|0720|0730|||0e00|0a10|0e20|0a30||0f00|
 | 5|0640|0650|0660|0670||0740|0750|0760|0770|||0e40|0a50|0e60|0a70||0f40|
 | 6|0680|0690|06a0|06b0||0780|0790|07a0|07b0|||0e80|0a90|0ea0|0ab0||0f80|
```

```
 | 7|06c0|06d0|06e0|06f0||07c0|07d0|07e0|07f0|||0ec0|0ad0|0ee0|0af0||0fc0|
 +==+====+====+====+====++====+====+====+====++++====+====+====+====++====+
 | 8|1400|1410|1420|1430||1500|1510|1520|1530|||1c00|1c10|1c20|1c30||1d00|
 | 9|1440|1450|1460|1470||1540|1550|1560|1570|||1c40|1c50|1c60|1c70||1d40|
 |10|1480|1490|14a0|14b0||1580|1590|15a0|15b0|||1c80|1c90|1ca0|1cb0||1d80|
 |11|14c0|14d0|14e0|14f0||15c0|15d0|15e0|15f0|||1cc0|1cd0|1ce0|1cf0||1dc0|
 +--+----+----+----+----++----+----+----+----++++----+----+----+----++----+
 |12|1600|1610|1620|1630||1700|1710|1720|1730|||1e00|1e10|1e20|1e30||1f00|
 |13|1640|1650|1660|1670||1740|1750|1760|1770|||1e40|1e50|1e60|1e70||1f40|
 |14|1680|1690|16a0|16b0||1780|1790|17a0|17b0|||1e80|1e90|1ea0|1eb0||1f80|
 |15|16c0|16d0|16e0|16f0||17c0|17d0|17e0|17f0|||1ec0|1ed0|1ee0|1ef0||1fc0|
 +==+====+====+====+====++====+====+====+====++++====+====+====+====++====+
 |16|2400|2410|2420|2430||2500|2510|2520|2530|||2c00|2c10|2c20|2c30||2d00|
 +--+----+----+----+----++----+----+----+----++++----+----+----+----++----+
[z block bounduary here]
z == 2:
 x -->
y+--+----+----+----+----++----+----+----+----++++----+----+----+----++----+
||  |  0 |  1 |  2 |  3 ||  4 |  5 |  6 |  7 |||  8 |  9 | 10 | 11 || 12 |
|+--+----+----+----+----++----+----+----+----++++----+----+----+----++----+
V| 0|3000|3010|3020|3030||3100|3110|3120|3130|||3800|3810|3820|3830||3900|
 | 1|3040|3050|3060|3070||3140|3150|3160|3170|||3840|3850|3860|3870||3940|
 | 2|3080|3090|30a0|30b0||3180|3190|31a0|31b0|||3880|3890|38a0|38b0||3980|
 | 3|30c0|30d0|30e0|30f0||31c0|31d0|31e0|31f0|||38c0|38d0|38e0|38f0||39c0|
 +--+----+----+----+----++----+----+----+----++++----+----+----+----++----+
 | 4|3200|3210|3220|3230||3300|3310|3320|3330|||3a00|3a10|3a20|3a30||3b00|
 | 5|3240|3250|3260|3270||3340|3350|3360|3370|||3a40|3a50|3a60|3a70||3b40|
 | 6|3280|3290|32a0|32b0||3380|3390|33a0|33b0|||3a80|3a90|3aa0|3ab0||3b80|
 | 7|32c0|32d0|32e0|32f0||33c0|33d0|33e0|33f0|||3ac0|3ad0|3ae0|3af0||3bc0|
 +==+====+====+====+====++====+====+====+====++++====+====+====+====++====+
 | 8|4000|4010|4020|4030||4100|4110|4120|4130|||4800|4810|4820|4830||4900|
 | 9|4040|4050|4060|4070||4140|4150|4160|4170|||4840|4850|4860|4870||4940|
 |10|4080|4090|40a0|40b0||4180|4190|41a0|41b0|||4880|4890|48a0|48b0||4980|
 |11|40c0|40d0|40e0|40f0||41c0|41d0|41e0|41f0|||48c0|48d0|48e0|48f0||49c0|
 +--+----+----+----+----++----+----+----+----++++----+----+----+----++----+
 |12|4200|4210|4220|4230||4300|4310|4320|4330|||4a00|4a10|4a20|4a30||4b00|
 |13|4240|4250|4260|4270||4340|4350|4360|4370|||4a40|4a50|4a60|4a70||4b40|
 |14|4280|4290|42a0|42b0||4380|4390|43a0|43b0|||4a80|4a90|4aa0|4ab0||4b80|
 |15|42c0|42d0|42e0|42f0||43c0|43d0|43e0|43f0|||4ac0|4ad0|4ae0|4af0||4bc0|
 +==+====+====+====+====++====+====+====+====++++====+====+====+====++====+
 |16|5000|5010|5020|2030||5100|5110|5120|5130|||5800|5810|5820|5830||5900|
 +--+----+----+----+----++----+----+----+----++++----+----+----+----++----+
```

Textures, mipmapping and arrays

A texture on G80/GF100 can have one of 9 types:

- 1D: made of 1 or more mip levels, each mip level is a blocklinear surface with height and depth forced to 1
- 2D: made of 1 or more mip levels, each mip level is a blocklinear surface with depth forced to 1
- 3D: made of 1 or more mip levels, each mip level is a blocklinear surface
- 1D_ARRAY: made of some number of subtextures, each subtexture is like a single 1D texture
- 2D_ARRAY: made of some number of subtextures, each subtexture is like a single 2D texture
- CUBE: made of 6 subtextures, each subtexture is like a single 2D texture - has the same layout as a 2D_ARRAY with 6 subtextures, but different semantics

- BUFFER: a simple packed 1D array of elements - not a surface
- RECT: a single pitch surface, or a single blocklinear surface with depth forced to 1
- CUBE_ARRAY [GT215+]: like 2D_ARRAY, but subtexture count has to be divisible by 6, and groups of 6 subtextures behave like CUBE textures

Types other than BUFFER and RECT are made of subtextures, which are in turn made of mip levels, which are blocklinear surfaces. For such textures, only the parameters of the first mip level of the first subtexture are specified - parameters of the following mip levels and subtextures are calculated automatically.

Each mip level has each dimension 2 times smaller than the corresponding dimension of previous mip level, rounding down unless it would result in size of 0. Since texture units use auto-sizing for the block size, the block sizes will be different between mip levels. The surface for each mip level starts right after the previous one ends. Also, the total size of the subtexture is rounded up to the size of the 0th mip level's block size:

```
mip_address[0] = subtexture_address;
mip_width[0] = texture_width;
mip_height[0] = texture_height;
mip_depth[0] = texture_depth;
mip_bytes[0] = calc_surface_bytes(mip[0]);
subtexture_bytes = mip_bytes[0];
for (i = 1; i <= max_mip_level; i++) {
    mip_address[i] = mip_address[i-1] + mip_bytes[i-1];
    mip_width[i] = max(1, floor(mip_width[i-1] / 2));
    mip_height[i] = max(1, floor(mip_height[i-1] / 2));
    mip_depth[i] = max(1, floor(mip_depth[i-1] / 2));
    mip_bytes[i] = calc_surface_bytes(mip[1]);
    subtexture_bytes += mip_bytes[i];
}
subtexture_bytes = alignup(subtexture_bytes, calc_surface_block_bytes(mip[0]));
```

For 1D_ARRAY, 2D_ARRAY, CUBE and CUBE_ARRAY textures, the subtextures are stored sequentially:

```
for (i = 0; i < subtexture_count; i++) {
    subtexture_address[i] = texture_address + i * subtexture_bytes;
}
```

For more information about textures, see graph/g80-texture.txt

Multisampled surfaces

Some surfaces are used as multisampled surfaces. This includes surfaces bound as color and zeta render targets when multisampling type is other than 1X, as well as multisampled textures on GF100+.

A multisampled surface contains several samples per pixel. A "sample" is a single set of RGBA or depth/stencil values [depending on surface type]. These samples correspond to various points inside the pixel, called sample positions. When a multisample surface has to be displayed, it is downsampled to a normal surface by an operation called "resolving".

G80+ GPUs also support a variant of multisampling called "coverage sampling" or CSAA. When CSAA is used, there are two sample types: full samples and coverage samples. Full samples behave as in normal multisampling. Coverage samples have assigned positions inside a pixel, but their values are not stored in the render target surfaces when rendering. Instead, a special component, called C or coverage, is added to the zeta surface, and for each coverage sample, a bitmask of full samples with the same value is stored. During the resolve process, this bitmask is used to assign different weights to the full samples depending on the count of coverage samples with matching values, thus improving picture quality. Note that the C component conceptually belongs to a whole pixel, not to individual samples. However, for surface layout purposes, its value is split into several parts, and each of the parts is stored together with one of the samples.

For the most part, multisampling mode does not affect surface layout - in fact, a multisampled render target is bound as a non-multisampled texture for the resolving process. However, multisampling mode is vital for CSAA zeta surface layout, and for render target storage type selection if compression is to be used - the compression schema used is directly tied to multisampling mode.

The following multisample modes exist:

- mode 0x0: MS1 [1×1] - no multisampling
  - sample 0: (0x0.8, 0x0.8) [0,0]
- mode 0x1: MS2 [2×1]
  - sample 0: (0x0.4, 0x0.4) [0,0]
  - sample 1: (0x0.c, 0x0.c) [1,0]
- mode 0x2: MS4 [2×2]
  - sample 0: (0x0.6, 0x0.2) [0,0]
  - sample 1: (0x0.e, 0x0.6) [1,0]
  - sample 2: (0x0.2, 0x0.a) [0,1]
  - sample 3: (0x0.a, 0x0.e) [1,1]
- mode 0x3: MS8 [4×2]
  - sample 0: (0x0.1, 0x0.7) [0,0]
  - sample 1: (0x0.5, 0x0.3) [1,0]
  - sample 2: (0x0.3, 0x0.d) [0,1]
  - sample 3: (0x0.7, 0x0.b) [1,1]
  - sample 4: (0x0.9, 0x0.5) [2,0]
  - sample 5: (0x0.f, 0x0.1) [3,0]
  - sample 6: (0x0.b, 0x0.f) [2,1]
  - sample 7: (0x0.d, 0x0.9) [3,1]
- mode 0x4: MS2_ALT [2×1] [GT215-]
  - sample 0: (0x0.c, 0x0.c) [1,0]
  - sample 1: (0x0.4, 0x0.4) [0,0]
- mode 0x5: MS8_ALT [4×2] [GT215-]
  - sample 0: (0x0.9, 0x0.5) [2,0]
  - sample 1: (0x0.7, 0x0.b) [1,1]
  - sample 2: (0x0.d, 0x0.9) [3,1]
  - sample 3: (0x0.5, 0x0.3) [1,0]
  - sample 4: (0x0.3, 0x0.d) [0,1]
  - sample 5: (0x0.1, 0x0.7) [0,0]
  - sample 6: (0x0.b, 0x0.f) [2,1]
  - sample 7: (0x0.f, 0x0.1) [3,0]
- mode 0x6: ??? [GF100-] [XXX]

- mode 0x8: MS4_CS4 [2×2]
  - sample 0: (0x0.6, 0x0.2) [0,0]
  - sample 1: (0x0.e, 0x0.6) [1,0]
  - sample 2: (0x0.2, 0x0.a) [0,1]
  - sample 3: (0x0.a, 0x0.e) [1,1]
  - coverage sample 4: (0x0.5, 0x0.7), belongs to 1, 3, 0, 2
  - coverage sample 5: (0x0.9, 0x0.4), belongs to 3, 2, 1, 0
  - coverage sample 6: (0x0.7, 0x0.c), belongs to 0, 1, 2, 3
  - coverage sample 7: (0x0.b, 0x0.9), belongs to 2, 0, 3, 1

  C component is 16 bits per pixel, bitfields:
  - 0-3: sample 4 associations: 0, 1, 2, 3
  - 4-7: sample 5 associations: 0, 1, 2, 3
  - 8-11: sample 6 associations: 0, 1, 2, 3
  - 12-15: sample 7 associations: 0, 1, 2, 3
- mode 0x9: MS4_CS12 [2×2]
  - sample 0: (0x0.6, 0x0.1) [0,0]
  - sample 1: (0x0.f, 0x0.6) [1,0]
  - sample 2: (0x0.1, 0x0.a) [0,1]
  - sample 3: (0x0.a, 0x0.f) [1,1]
  - coverage sample 4: (0x0.4, 0x0.e), belongs to 2, 3
  - coverage sample 5: (0x0.c, 0x0.3), belongs to 1, 0
  - coverage sample 6: (0x0.d, 0x0.d), belongs to 3, 1
  - coverage sample 7: (0x0.4, 0x0.4), belongs to 0, 2
  - coverage sample 8: (0x0.9, 0x0.5), belongs to 0, 1, 2
  - coverage sample 9: (0x0.7, 0x0.7), belongs to 0, 2, 1, 3
  - coverage sample a: (0x0.b, 0x0.8), belongs to 1, 3, 0
  - coverage sample b: (0x0.3, 0x0.8), belongs to 2, 0, 3
  - coverage sample c: (0x0.8, 0x0.c), belongs to 3, 2, 1
  - coverage sample d: (0x0.2, 0x0.2), belongs to 0, 2
  - coverage sample e: (0x0.5, 0x0.b), belongs to 2, 3, 0, 1
  - coverage sample f: (0x0.e, 0x0.9), belongs to 1, 3

  C component is 32 bits per pixel, bitfields:
  - 0-1: sample 4 associations: 2, 3
  - 2-3: sample 5 associations: 0, 1
  - 4-5: sample 6 associations: 1, 3
  - 6-7: sample 7 associations: 0, 2

  - 8-10: sample 8 associations: 0, 1, 2
  - 11-14: sample 9 associations: 0, 1, 2, 3
  - 15-17: sample a associations: 0, 1, 3
  - 18-20: sample b associations: 0, 2, 3
  - 31-23: sample c associations: 1, 2, 3
  - 24-25: sample d associations: 0, 2
  - 26-29: sample e associations: 0, 1, 2, 3
  - 30-31: sample f associations: 1, 3
- mode 0xa: MS8_CS8 [4×2]
  - sample 0: (0x0.1, 0x0.3) [0,0]
  - sample 1: (0x0.6, 0x0.4) [1,0]
  - sample 2: (0x0.3, 0x0.f) [0,1]
  - sample 3: (0x0.4, 0x0.b) [1,1]
  - sample 4: (0x0.c, 0x0.1) [2,0]
  - sample 5: (0x0.e, 0x0.7) [3,0]
  - sample 6: (0x0.8, 0x0.8) [2,1]
  - sample 7: (0x0.f, 0x0.d) [3,1]
  - coverage sample 8: (0x0.5, 0x0.7), belongs to 1, 6, 3, 0
  - coverage sample 9: (0x0.7, 0x0.2), belongs to 1, 0, 4, 6
  - coverage sample a: (0x0.b, 0x0.6), belongs to 5, 6, 1, 4
  - coverage sample b: (0x0.d, 0x0.3), belongs to 4, 5, 6, 1
  - coverage sample c: (0x0.2, 0x0.9), belongs to 3, 0, 2, 1
  - coverage sample d: (0x0.7, 0x0.c), belongs to 3, 2, 6, 7
  - coverage sample e: (0x0.a, 0x0.e), belongs to 7, 3, 2, 6
  - coverage sample f: (0x0.c, 0x0.a), belongs to 5, 6, 7, 3

  C component is 32 bits per pixel, bitfields:

  - 0-3: sample 8 associations: 0, 1, 3, 6
  - 4-7: sample 8 associations: 0, 1, 4, 6
  - 8-11: sample 8 associations: 1, 4, 5, 6
  - 12-15: sample 8 associations: 1, 4, 5, 6
  - 16-19: sample 8 associations: 0, 1, 2, 3
  - 20-23: sample 8 associations: 2, 3, 6, 7
  - 24-27: sample 8 associations: 2, 3, 6, 7
  - 28-31: sample 8 associations: 3, 5, 6, 7
- mode 0xb: MS8_CS24 [GF100-]

Todo

wtf is up with modes 4 and 5?

Todo

nail down MS8_CS24 sample positions

Todo

figure out mode 6

Todo

figure out MS8_CS24 C component

Note that MS8 and MS8_C* modes cannot be used with surfaces that have 16-byte element size due to a hardware limitation. Also, multisampling is only possible with blocklinear surfaces.

Todo

check MS8/128bpp on GF100.

The sample ids are, for full samples, the values appearing in the sampleid register. The numbers in () are the geometric coordinates of the sample inside a pixel, as used by the rasterization process. The dimensions in [] are dimensions of a block represents a pixel in the surface - if it's 4×2, each pixel is represented in the surface as a block 4 elements wide and 2 elements tall. The numbers in [] after each full sample are the coordinates inside this block.

Each coverage sample "belongs to" several full samples. For every such pair of coverage sample and full sample, the C component contains a bit that tells if the coverage sample's value is the same as the full one's, ie. if the last rendered primitive that covered the full sample also covered the coverage sample. When the surface is resolved, each sample will "contribute" to exactly one full sample. The full samples always contribute to themselves, while coverage sample will contribute to the first full sample that they belong to, in order listed above, that has the relevant bit set in C component of the zeta surface. If none of the C bits for a given coverage sample are set, the sample will default to contributing to the first sample in its belongs list. Then, for each full sample, the number of samples contributing to it is counted, and used as its weight when performing the downsample calculation.

Note that, while the belongs list orderings are carefully chosen based on sample locations and to even the weights, the bits in C component don't use this ordering and are sorted by sample id instead.

The C component is 16 or 32 bits per pixel, depending on the format. It is then split into 8-bit chunks, starting from LSB, and each chunk is assigned to one of the full samples. For MS4_CS4 and MS8_CS8, only samples in the top line of each block get a chunk assigned, for MS4_CS12 all samples get a chunk. The chunks are assigned to samples ordered first by x coordinate of the sample, then by its y coordinate.

Surface formats

A surface's format determines the type of information it stores in its elements, the element size, and the element layout. Not all binding points care about the format - m2mf and PCOPY treat all surfaces as arrays of bytes. Also, format specification differs a lot between the binding points that make use of it - 2d engine and render targets use a

big enum of valid formats, with values specifying both the layout and components, while texture units decouple layout specification from component assignment and type selection, allowing arbitrary swizzles.

There are 3 main enums used for specifying surface formats:

- texture format: used for textures, epecifies element size and layout, but not the component assignments nor type
- color format: used for color RTs and the 2d engine, specifies the full format
- zeta format: used for zeta RTs, specifies the full format, except the specific coverage sampling mode, if applicable

The surface formats can be broadly divided into the following categories:

- simple color formats: elements correspond directly to samples. Each element has 1 to 4 bitfields corresponding to R, G, B, A components. Usable for texturing, color RTs, and 2d engine.
- shared exponent color format: like above, but the components are floats sharing the exponent bitfield. Usable for texturing only.
- YUV color formats: element corresponds to two pixels lying in the same horizontal line. The pixels have three components, conventionally labeled as Y, U, V. U and V components are common for the two pixels making up an element, but Y components are separate. Usable for texturing only.
- zeta formats: elements correspond to samples. There is a per-sample depth component, optionally a per-sample stencil component, and optionally a per-pixel coverage value for CSAA surfaces. Usable for texturing and ZETA RT.
- compressed texture formats: elements correspond to blocks of samples, and are decoded to RGBA color values on the fly. Can be used only for texturing.
- bitmap texture format: each element corresponds to 8x8 block of samples, with 1 bit per sample. Has to be used with a special texture sampler. Usable for texturing and 2d engine.

Todo

wtf is color format 0x1d?

Simple color surface formats

A simple color surface is a surface where each element corresponds directly to a sample, each sample has 4 components known as R, G, B, A [in that order], and the bitfields in element correspond directly to components. There can be less bitfields than components - the remaining components will be ignored on write, and get a default value on read, which is 0 for R, G, B and 1 for A.

When bound to texture unit, the simple color formats are specified in three parts. First, the format is specified, which is an enumerated value shared with other format types. This format specifies the format type and, for simple color formats, element size, and location of bitfields inside the element. Then, the type [float/sint/uint/unorm/snorm] of each element component is specified. Finally, a swizzle is specified: each of the 4 component outputs [R, G, B, A] from the texture unit can be mapped to any of the components present in the element [called C0-C3], constant 0, integer constant 1, or float constant 1.

Thanks to the swizzle capability, there's no need to support multiple orderings in the format itself, and all simple color texture formats have C0 bitfield starting at LSB of the first byte, C1 [if present] at the first bit after C0, and so on. Thus it's enough to specify bitfield lengths to uniquely identify a texture type: for example 5_5_6 is a format with 3 components and element size of 2 bytes, C0 at bits 0-4, C1 at bits 5-9, and C2 at bits 10-15. The element is always treated as a little-endian word of the proper size, and bitfields are listed from LSB side. Also, in some cases the texture format has bitfields used only for padding, and not usable as components: these will be listed in the name as X<size>.

For example, 32_8_X24 is a format with element size of 8 bytes, where bits 0-31 are C0, 32-39 are C1, and 40-63 are unusable. [XXX: what exactly happens to element layout in big-endian mode?]

However, when bound to RTs or the 2d engine, all of the format, including element size, element layout, component types, component assignment, and SRGB flag, is specified by a single enumerated value. These formats have a many-to-one relationship to texture formats, and are listed here below the corresponding one. The information listed here for a format is C0-C3 assignments to actual components and component type, plus SRGB flag where applicable. The components can be R, G, B, A, representing a bitfield corresponding directly to a single component, X representing an unused bitfield, or Y representing a bitfield copied to all components on read, and filled with the R value on write.

The formats are:

Element size 16:

- texture format 0x01: 32_32_32_32
  - color format 0xc0: RGBA, float
  - color format 0xc1: RGBA, sint
  - color format 0xc2: RGBA, uint
  - color format 0xc3: RGBX, float
  - color format 0xc4: RGBX, sint
  - color format 0xc5: RGBX, uint

Element size 8:

- texture format 0x03: 16_16_16_16
  - color format 0xc6: RGBA, unorm
  - color format 0xc7: RGBA, snorm
  - color format 0xc8: RGBA, sint
  - color format 0xc9: RGBA, uint
  - color format 0xca: RGBA, float
  - color format 0xce: RGBX, float
- texture format 0x04: 32_32
  - color format 0xcb: RG, float
  - color format 0xcc: RG, sint
  - color format 0xcd: RG, uint
- texture format 0x05: 32_8_X24

Element size 4:

- texture format 0x07: 8_8_8_X8

Todo

htf do I determine if a surface format counts as 0x07 or 0x08?

- texture format 0x08: 8_8_8_8
  - color format 0xcf: BGRA, unorm
  - color format 0xd0: BGRA, unorm, SRGB

    - color format 0xd5: RGBA, unorm
    - color format 0xd6: RGBA, unorm, SRGB
    - color format 0xd7: RGBA, snorm
    - color format 0xd8: RGBA, sint
    - color format 0xd9: RGBA, uint
    - color format 0xe6: BGRX, unorm
    - color format 0xe7: BGRX, unorm, SRGB
    - color format 0xf9: RGBX, unorm
    - color format 0xfa: RGBX, unorm, SRGB
    - color format 0xfd: BGRX, unorm [XXX]
    - color format 0xfe: BGRX, unorm [XXX]
- texture format 0x09: 10_10_10_2
    - color format 0xd1: RGBA, unorm
    - color format 0xd2: RGBA, uint
    - color format 0xdf: BGRA, unorm
- texture format 0x0c: 16_16
    - color format 0xda: RG, unorm
    - color format 0xdb: RG, snorm
    - color format 0xdc: RG, sint
    - color format 0xdd: RG, uint
    - color format 0xde: RG, float
- texture format 0x0d: 24_8
- texture format 0x0e: 8_24
- texture format 0x0f: 32
    - color format 0xe3: R, sint
    - color format 0xe4: R, uint
    - color format 0xe5: R, float
    - color format 0xff: Y, uint [XXX]
- texture format 0x21: 11_11_10
    - color format 0xe0: RGB, float

Element size 2:

- texture format 0x12: 4_4_4_4
- texture format 0x13: 1_5_5_5
- texture format 0x14: 5_5_5_1
    - color format 0xe9: BGRA, unorm
    - color format 0xf8: BGRX, unorm

    - color format 0xfb: BGRX, unorm [XXX]
    - color format 0xfc: BGRX, unorm [XXX]
- texture format 0x15: 5_6_5
    - color format 0xe8: BGR, unorm
- texture format 0x16: 5_5_6
- texture format 0x18: 8_8
    - color format 0xea: RG, unorm
    - color format 0xeb: RG, snorm
    - color format 0xec: RG, uint
    - color format 0xed: RG, sint
- texture format 0x1b: 16
    - color format 0xee: R, unorm
    - color format 0xef: R, snorm
    - color format 0xf0: R, sint
    - color format 0xf1: R, uint
    - color format 0xf2: R, float

Element size 1:

- texture format 0x1d: 8
    - color format 0xf3: R, unorm
    - color format 0xf4: R, snorm
    - color format 0xf5: R, sint
    - color format 0xf6: R, uint
    - color format 0xf7: A, unorm
- texture format 0x1e: 4_4

Todo

which component types are valid for a given bitfield size?

Todo

clarify float encoding for weird sizes

Shared exponent color format

A shared exponent color format is like a simple color format, but there's an extra bitfield, called E, that's used as a shared exponent for C0-C2. The remaining three bitfields correspond to the mantissas of C0-C2, respectively. They can be swizzled arbitrarily, but they have to use the float type.

Element size 4:

- texture format 0x20: 9_9_9_E5

YUV color formats

These formats are also similar to color formats. However, The components are conventionally called Y, U, V: C0 is known as U, C1 is known as Y, and C2 is known as V. An element represents two pixels, and has 4 bitfields: YA representing Y value for first pixel, YB representing Y value for second pixel, U representing U value for both pixels, and V representing V value of both pixels. There are two YUV formats, differing in bitfield order:

Element size 4:

- texture format 0x21: U8_YA8_V8_YB8
- texture format 0x22: YA8_U8_YB8_V8

Todo

verify I haven't screwed up the ordering here

Zeta surface format

A zeta surface, like a simple color surface, has one element per sample. It contains up to three components: the depth component [called Z], optionally the stencil component [called S], and if coverage sampling is in use, the coverage component [called C].

The Z component can be a 32-bit float, a 24-bit normalized unsigned integer, or [on G200+] a 16-bit normalized unsigned integer. The S component, if present, is always an 8-bit raw integer.

The C component is special: if present, it's an 8-bit bitfield in each sample. However, semantically it is a per-pixel value, and the values of the samples' C components are stitched together to obtain a per-pixel value. This stitching process depends on the multisample mode, thus it needs to be specified to bind a coverage sampled zeta surface as a texture. It's not allowed to use a coverage sampling mode with a zeta format without C component, or the other way around.

Like with color formats, there are two different enums that specify zeta formats: texture formats and zeta formats. However, this time the zeta formats have one-to-many relationship with texture formats: Texture format contains information about the specific coverage sampling mode used, while zeta format merely says whether coverage sampling is in use, and the mode is taken from RT multisample configuration.

For textures, Z corresponds to C0, S to C1, and C to C2. However, C cannot be used together with Z and/or S in a single sampler. Z and S sampling works normally, but when C is sampled, the sampler returns preprocessed weights instead of the raw value - see graph/g80-texture.txt for more information about the sampling process.

The formats are:

Element size 2:

- zeta format 0x13: Z16 [G200+ only]
  - texture format 0x3a: Z16 [G200+ only]

Element size 4:

- zeta format 0x0a: Z32
  - texture format 0x2f
- zeta format 0x14: S8_Z24

    - texture format 0x29
- zeta format 0x15: Z24_X8
    - texture format 0x2b
- zeta format 0x16: Z24_S8
    - texture format 0x2a
- zeta format 0x18: Z24_C8
    - texture format 0x2c: MS4_CS4
    - texture format 0x2d: MS8_CS8
    - texture format 0x2e: MS4_CS12

Element size 8:

- zeta format 0x19: Z32_S8_X24
    - texture format 0x30
- zeta format 0x1d: Z24_X8_S8_C8_X16
    - texture format 0x31: MS4_CS4
    - texture format 0x32: MS8_CS8
    - texture format 0x37: MS4_CS12
- zeta format 0x1e: Z32_X8_C8_X16
    - texture format 0x33: MS4_CS4
    - texture format 0x34: MS8_CS8
    - texture format 0x38: MS4_CS12
- zeta format 0x1f: Z32_S8_C8_X16
    - texture format 0x35: MS4_CS4
    - texture format 0x36: MS8_CS8
    - texture format 0x39: MS4_CS12

Todo

figure out the MS8_CS24 formats

Compressed texture formats

Todo

write me

Bitmap surface format

A bitmap surface has only one component, and the component has 1 bit per sample - that is, the component's value can be either 0 or 1 for each sample in the surface. The surface is made of 8-byte elements, with each element representing 8×8 block of samples. The element is treated as a 64-bit word, with each sample taking 1 bit. The bits start from LSB and are ordered first by x coordinate of the sample, then by its y coordinate.

This format can be used for 2d engine and texturing. When used for texturing, it forces using a special "box" filter: result of sampling is a percentage of "lit" area in WxH rectangle centered on the sampled location. See graph/g80-texture.txt for more details.

Todo

figure out more. Check how it works with 2d engine.

The formats are:

Element size 8:

- texture format 0x1f: BITMAP
  - color format 0x1c: BITMAP

G80 storage types

On G80, the storage type is made of two parts: the storage type itself, and the compression mode. The storage type is a 7-bit enum, the compression mode is a 2-bit enum.

The compression modes are:

- 0: NONE - no compression
- 1: SINGLE - 2 compression tag bits per gob, 1 tag cell per 64kB page
- 2: DOUBLE - 4 compression tag bits per gob, 2 tag cells per 64kB page

Todo

verify somehow.

The set of valid compression modes varies with the storage type. NONE is always valid.

As mentioned before, the low-level rearrangement is further split into two sublevels: short range reordering, rearranging bytes in a single gob, and long range reordering, rearranging gobs. Short range reordering is performed for both VRAM and system RAM, and is highly dependent on the storage type. Long range reordering is done only for VRAM, and has only three types:

- none [NONE] - no reordering, only used for storage type 0 [pitch]
- small scale [SSR] - gobs rearranged inside a single 4kB page, used for non-0 storage types
- large scale [LSR] - large blocks of memory rearranged, based on internal VRAM geometry. Boundaries between VRAM areas using NONE/SSR and LSR need to be properly aligned in physical space to prevent conflicts.

Long range reordering is described in detail in *G80:GF100 VRAM structure and usage*.

The storage types can be roughly split into the following groups:

- pitch storage type: used for pitch surfaces and non-surface buffers

- blocklinear color storage types: used for non-zeta blocklinear surfaces
- zeta storage types: used for zeta surfaces

On the original G80, non-0 storage types can only be used on VRAM, on G84 and later cards they can also be used on system RAM. Compression modes other than NONE can only be used on VRAM. However, due to the G80 limitation, blocklinear surfaces stored in system RAM are allowed to use storage type 0, and will work correctly for texturing and m2mf source/destination - rendering to them with 2d or 3d engine is impossible, though.

Correct storage types are only enforced by texture units and ROPs [ie. 2d and 3d engine render targets + CUDA global/local/stack spaces], which have dedicated paths to memory and depend on the storage types for performance. The other engines have storage type handling done by the common memory controller logic, and will accept any storage type.

The pitch storage type is:

storage type 0x00: PITCH long range reordering: NONE valid compression modes: NONE There's no short range reordering on this storage type - the offset inside a gob is identical between the virtual and physical addresses.

Blocklinear color storage types

Todo

reformat

The following blocklinear color storage types exist:

storage type 0x70: BLOCKLINEAR long range reordering: SSR valid compression modes: NONE valid surface formats: any non-zeta with element size of 1, 2, 4, or 8 bytes valid multisampling modes: any

storage type 0x72: BLOCKLINEAR_LSR long range reordering: LSR valid compression modes: NONE valid surface formats: any non-zeta with element size of 1, 2, 4, or 8 bytes valid multisampling modes: any

storage type 0x76: BLOCKLINEAR_128_LSR long range reordering: LSR valid compression modes: NONE valid surface formats: any non-zeta with element size of 16 bytes valid multisampling modes: any

[XXX]

storage type 0x74: BLOCKLINEAR_128 long range reordering: SSR valid compression modes: NONE valid surface formats: any non-zeta with element size of 16 bytes valid multisampling modes: any

[XXX]

storage type 0x78: BLOCKLINEAR_32_MS4 long range reordering: SSR valid compression modes: NONE, SINGLE valid surface formats: any non-zeta with element size of 4 bytes valid multisampling modes: MS1, MS2*, MS4*

storage type 0x79: BLOCKLINEAR_32_MS8 long range reordering: SSR valid compression modes: NONE, SINGLE valid surface formats: any non-zeta with element size of 4 bytes valid multisampling modes: MS8*

storage type 0x7a: BLOCKLINEAR_32_MS4_LSR long range reordering: LSR valid compression modes: NONE, SINGLE valid surface formats: any non-zeta with element size of 4 bytes valid multisampling modes: MS1, MS2*, MS4*

storage type 0x7b: BLOCKLINEAR_32_MS8_LSR long range reordering: LSR valid compression modes: NONE, SINGLE valid surface formats: any non-zeta with element size of 4 bytes valid multisampling modes: MS8*

[XXX]

storage type 0x7c: BLOCKLINEAR_64_MS4 long range reordering: SSR valid compression modes: NONE, SINGLE valid surface formats: any non-zeta with element size of 8 bytes valid multisampling modes: MS1, MS2*, MS4*

storage type 0x7d: BLOCKLINEAR_64_MS8 long range reordering: SSR valid compression modes: NONE, SINGLE valid surface formats: any non-zeta with element size of 8 bytes valid multisampling modes: MS8*

[XXX]

storage type 0x44: BLOCKLINEAR_24 long range reordering: SSR valid compression modes: NONE valid surface formats: texture format 8_8_8_X8 and corresponding color formats valid multisampling modes: any

storage type 0x45: BLOCKLINEAR_24_MS4 long range reordering: SSR valid compression modes: NONE, SINGLE valid surface formats: texture format 8_8_8_X8 and corresponding color formats valid multisampling modes: MS1, MS2*, MS4*

storage type 0x46: BLOCKLINEAR_24_MS8 long range reordering: SSR valid compression modes: NONE, SINGLE valid surface formats: texture format 8_8_8_X8 and corresponding color formats valid multisampling modes: MS8*

storage type 0x4b: BLOCKLINEAR_24_LSR long range reordering: LSR valid compression modes: NONE valid surface formats: texture format 8_8_8_X8 and corresponding color formats valid multisampling modes: any

storage type 0x4c: BLOCKLINEAR_24_MS4_LSR long range reordering: LSR valid compression modes: NONE, SINGLE valid surface formats: texture format 8_8_8_X8 and corresponding color formats valid multisampling modes: MS1, MS2*, MS4*

storage type 0x4d: BLOCKLINEAR_24_MS8_LSR long range reordering: LSR valid compression modes: NONE, SINGLE valid surface formats: texture format 8_8_8_X8 and corresponding color formats valid multisampling modes: MS8*

[XXX]

Zeta storage types

Todo

write me

GF100 storage types

Todo

write me

2.7.8 Tesla virtual memory

Contents

Introduction

G80 generation cards feature an MMU that translates user-visible logical addresses to physical ones. The translation has two levels: DMA objects, which behave like x86 segments, and page tables. The translation involves the following address spaces:

- logical addresses: 40-bit logical address + channel descriptor address + DMAobj address. Specifies an address that will be translated by the relevant DMAobj, and then by the page tables if DMAobj says so. All addresses appearing in FIFO command streams are logical addresses, or eventually translated to logical addresses
- virtual addresses: 40-bit virtual address + channel descriptor address, specifies an address that will be looked up in the page tables of the relevant channel. Virtual addresses are always a result of logical address translation and can never be specified directly.
- linear addresses: 40-bit linear address + target specifier, which can be VRAM, SYSRAM_SNOOP, or SYSRAM_NOSNOOP. They can refer to:
  - VRAM: 32-bit linear addresses - high 8 bits are ignored - on-board memory of the card. Supports LSR and compression. See *G80:GF100 VRAM structure and usage*
  - SYSRAM: 40-bit linear addresses - accessing this space will cause the card to invoke PCIE read/write transactions to the given bus address, allowing it to access system RAM or other PCI devices' memory. SYSRAM_SNOOP uses normal PCIE transactions, SYSRAM_NOSNOOP uses PCIE transactions with the "no snoop" bit set.

  Mostly, linear addresses are a result of logical address translation, but some memory areas are specified directly by their linear addresses.
- 12-bit tag addresses: select a cell in hidden compression tag RAM, used for compressed areas of VRAM. See *G80 VRAM compression*
- physical address: for VRAM, the partition/subpartition/row/bank/column coordinates of a memory cell; for SYSRAM, the final bus address

Todo

kill this list in favor of an actual explanation

The VM's job is to translate a logical address into its associated data:

- linear address

- target: VRAM, SYSRAM_SNOOP, or SYSRAM_NOSNOOP
- read-only flag
- supervisor-only flag
- storage type: a special value that selects the internal structure of contained data and enables more efficient accesses by increasing cache locality
- compression mode: if set, write accesses will attempt to compress the written data and, if successful, write only a fraction of the original write size to memory and mark the tile as compressed in the hidden tag memory. Read accesses will transparently uncompress the data. Can only be used on VRAM.
- compression tag address: the address of tag cell to be used if compression is enabled. Tag memory is addressed by "cells". Each cell is actually 0x200 tag bits. For SINGLE compression mode, every 0x10000 bytes of compressed VRAM require 1 tag cell. For DOUBLE compression mode, every 0x10000 bytes of VRAM require 2 tag cells.
- partition cycle: either short or long, affecting low-level VRAM storage
- encryption flag [G84+]: for SYSRAM, causes data to be encrypted with a simple cipher before being stored

A VM access can also end unsuccessfully due to multiple reasons, like a non present page. When that happens, a VM fault is triggered. The faulting access data is stored, and fault condition is reported to the requesting engine. Consequences of a faulted access depend on the engine.

VM users

VM is used by several clients, which are identified by VM client id:

A related concept is VM engine, which is a group of clients that share TLBs and stay on the same channel at any single moment. It's possible for a client to be part of several VM engines. The engines are:

Client+engine combination doesn't, however, fully identify the source of the access - to disambiguate that, DMA slot ids are used. The set of DMA slot ids depends on both engine and client id. The DMA slots are [engine/client/slot]:

- 0/0/0: PGRAPH STRMOUT
- 0/3/0: PGRAPH context
- 0/3/1: PGRAPH NOTIFY
- 0/3/2: PGRAPH QUERY
- 0/3/3: PGRAPH COND
- 0/3/4: PGRAPH m2mf BUFFER_IN
- 0/3/5: PGRAPH m2mf BUFFER_OUT
- 0/3/6: PGRAPH m2mf BUFFER_NOTIFY
- 0/5/0: PGRAPH CODE_CB
- 0/5/1: PGRAPH TIC
- 0/5/2: PGRAPH TSC
- 0/7/0: PGRAPH CLIPID
- 0/9/0: PGRAPH VERTEX
- 0/a/0: PGRAPH TEXTURE / SRC2D
- 0/b/0-7: PGRAPH RT 0-7

- 0/b/8: PGRAPH ZETA
- 0/b/9: PGRAPH LOCAL
- 0/b/a: PGRAPH GLOBAL
- 0/b/b: PGRAPH STACK
- 0/b/c: PGRAPH DST2D
- 4/4/0: PEEPHOLE write
- 4/8/0: PEEPHOLE read
- 6/4/0: BAR1 write
- 6/8/0: BAR1 read
- 6/4/1: BAR3 write
- 6/8/1: BAR3 read
- 5/8/0: FIFO pushbuf read
- 5/4/1: FIFO semaphore write
- 5/8/1: FIFO semaphore read
- c/8/1: FIFO background semaphore read
- 1/6/8: PVP1 context [G80:G84]
- 7/6/4: PME context [G80:G84]
- 8/6/1: PMPEG CMD [G80:G98 G200:MCP77]
- 8/6/2: PMPEG DATA [G80:G98 G200:MCP77]
- 8/6/3: PMPEG IMAGE [G80:G98 G200:MCP77]
- 8/6/4: PMPEG context [G80:G98 G200:MCP77]
- 8/6/5: PMPEG QUERY [G84:G98 G200:MCP77]
- b/f/0: PCOUNTER record buffer [G84:GF100]
- 1/c/0-f: PVP2 DMA ports 0-0xf [G84:G98 G200:MCP77]
- 9/d/0-f: PBSP DMA ports 0-0xf [G84:G98 G200:MCP77]
- a/e/0: PCIPHER context [G84:G98 G200:MCP77]
- a/e/1: PCIPHER SRC [G84:G98 G200:MCP77]
- a/e/2: PCIPHER DST [G84:G98 G200:MCP77]
- a/e/3: PCIPHER QUERY [G84:G98 G200:MCP77]
- 1/c/0-7: PPDEC falcon ports 0-7 [G98:G200 MCP77-]
- 8/6/0-7: PPPP falcon ports 0-7 [G98:G200 MCP77-]
- 9/d/0-7: PVLD falcon ports 0-7 [G98:G200 MCP77-]
- a/e/0-7: PSEC falcon ports 0-7 [G98:GT215]
- d/13/0-7: PCOPY falcon ports 0-7 [GT215-]
- e/11/0-7: PDAEMON falcon ports 0-7 [GT215-]
- 7/14/0-7: PVCOMP falcon ports 0-7 [MCP89-]

Todo

PVP1

Todo

PME

Todo

Move to engine doc?

Channels

All VM accesses are done on behalf of some "channel". A VM channel is just a memory structure that contains the DMA objects and page directory. VM channel can be also a FIFO channel, for use by PFIFO and fifo engines and containing other data structures, or just a "bare" VM channel for use with non-fifo engines.

A channel is identified by a "channel descriptor", which is a 30-bit number that points to the base of the channel memory structure:

- bits 0-27: bits 12-39 of channel memory structure linear address
- bits 28-29: the target specifier for channel memory structure - 0: VRAM - 1: invalid, do not use - 2: SYSRAM_SNOOP - 3: SYSRAM_NOSNOOP

The channel memory structure contains a few fixed-offset elements, as well as serving as a container for channel objects, such as DMA objects, that can be placed anywhere inside the structure. Due to the channel objects inside it, the channel structure has no fixed size, although the maximal address of channel objects is 0xffff0. Channel structure has to be aligned to 0x1000 bytes.

The original G80 channel structure has the following fixed elements:

- 0x000-0x200: RAMFC [fifo channels only]
- 0x200-0x400: DMA objects for fifo engines' contexts [fifo channels only]
- 0x400-0x1400: PFIFO CACHE [fifo channels only]
- 0x1400-0x5400: page directory

G84+ cards instead use the following structure:

- 0x000-0x200: DMA objects for fifo engines' contexts [fifo channels only]
- 0x200-0x4200: page directory

The channel objects are specified by 16-bit offsets from start of the channel structure in 0x10-byte units.

DMA objects

The only channel object type that VM subsystem cares about is DMA objects. DMA objects represent contiguous segments of either virtual or linear memory and are the first stage of VM address translation. DMA objects can be paged or unpaged. Unpaged DMA objects directly specify the target space and all attributes, merely adding the base address and checking the limit. Paged DMA objects add the base address, then look it up in the page tables. Attributes can either come from page tables, or be individually overriden by the DMA object.

DMA objects are specifid by 16-bit "selectors". In case of fifo engines, the RAMHT is used to translate from user-visible 32-bit handles to the selectors [see *RAMHT and the FIFO objects*]. The selector is shifted left by 4 bits and added to channel structure base to obtain address of DMAobj structure, which is 0x18 bytes long and made of 32-bit LE words:

word 0:

- bits 0-15: object class. Ignored by VM, but usually validated by fifo engines - should be 0x2 [read-only], 0x3 [write-only], or 0x3d [read-write]
- bits 16-17: target specifier:
  - 0: VM - paged object - the logical address is to be added to the base address to obtain a virtual address, then the virtual address should be translated via the page tables
  - 1: VRAM - unpaged object - the logical address should be added to the base address to directly obtain the linear address in VRAM
  - 2: SYSRAM_SNOOP - like VRAM, but gives SYSRAM address
  - 3: SYSRAM_NOSNOOP - like VRAM, but gives SYSRAM address and uses nosnoop transactions
- bits 18-19: read-only flag
  - 0: use read-only flag from page tables [paged objects only]
  - 1: read-only
  - 2: read-write
- bits 20-21: supervisor-only flag
  - 0: use supervisor-only flag from page tables [paged objects only]
  - 1: user-supervisor
  - 2: supervisor-only
- **bits 22-28: storage type. If the value is 0x7f, use storage type from page** tables, otherwise directly specifies the storage type
- bits 29-30: compression mode
  - 0: no compression
  - 1: SINGLE compression
  - 2: DOUBLE compression
  - 3: use compression mode from page tables
- bit 31: if set, is a supervisor DMA object, user DMA object otherwise

word 1: bits 0-31 of limit address

word 2: bits 0-31 of base address

word 3:

- bits 0-7: bits 32-39 of base address
- bits 24-31: bits 32-39 of limit address

word 4:

- bits 0-11: base tag address
- bits 16-27: limit tag address

word 5:

- bits 0-15: compression base address bits 16-31 [bits 0-15 are forced to 0]
- bits 16-17: partition cycle
  - 0: use partition cycle from page tables
  - 1: short cycle
  - 2: long cycle
- bits 18-19 [G84-]: encryption flag
  - 0: not encrypted
  - 1: encrypted
  - 2: use encryption flag from page tables

First, DMA object selector is compared with 0. If the selector is 0, NULL_DMAOBJ fault happens. Then, the logical address is added to the base address from DMA object. The resulting address is compared with the limit address from DMA object and, if larger or equal, DMAOBJ_LIMIT fault happens. If DMA object is paged, the address is looked up in the page tables, with read-only flag, supervisor-only flag, storage type, and compression mode optionally overriden as specified by the DMA object. Otherwise, the address directly becomes the linear address. For compressed unpaged VRAM objects, the tag address is computed as follows:

- take the computed VRAM linear address and substract compression base address from it. if result is negative, force compression mode to none
- shift result right by 16 bits
- add base tag address to the result
- if result <= limit tag addres, this is the tag address to use. Else, force compression mode to none.

Places where DMA objects are bound, that is MMIO registers or FIFO methods, are commonly called “DMA slots”.

Most engines cache the most recently bound DMA object. To flush the caches, it’s usually enough to rewrite the selector register, or resubmit the selector method.

It should be noted that many engines require the DMA object’s base address to be of some specific alignment. The alignment depends on the engine and slot.

The fifo engine context dmaobjs are a special set of DMA objects worth mentioning. They’re used by the fifo engines to store per-channel state while given channel is inactive on the relevant engine. Their size and structure depend on the engine. They have fixed selectors, and hence reside at fixed positions inside the channel structure. On the original G80, the objects are:

Selector	Address	Engine
0x0020	0x00200	PGRAPH
0x0022	0x00220	PVP1
0x0024	0x00240	PME
0x0026	0x00260	*PMPEG*

On G84+ cards, they are:

Selector	Address	Present on	Engine
0x0002	0x00020	all	PGRAPH
0x0004	0x00040	VP2	PVP2
0x0004	0x00040	VP3-	PPDEC
0x0006	0x00060	VP2	*PMPEG*
0x0006	0x00060	VP3-	PPPP
0x0008	0x00080	VP2	PBSP
0x0008	0x00080	VP3-	PVLD
0x000a	0x000a0	VP2	PCIPHER
0x000a	0x000a0	VP3	PSEC
0x000a	0x000a0	MCP89-	PVCOMP
0x000c	0x000c0	GT215-	PCOPY

Page tables

If paged DMA object is used, the virtual address is further looked up in page tables. The page tables are two-level. Top level is 0x800-entry page directory, where each entry covers 0x20000000 bytes of virtual address space. The page directory is embedded in the channel structure. It starts at offset 0x1400 on the original G80, at 0x200 on G84+. Each page directory entry, or PDE, is 8 bytes long. The PDEs point to page tables and specify the page table attributes. Each page table can use either small, medium [GT215-] or large pages. Small pages are 0x1000 bytes long, medium pages are 0x4000 bytes long, and large pages are 0x10000 bytes long. For small-page page tables, the size of page table can be artificially limitted to cover only 0x2000, 0x4000, or 0x8000 pages instead of full 0x20000 pages - the pages over this limit will fault. Medium- and large-page page tables always cover full 0x8000 or 0x2000 entries. Page tables of both kinds are made of 8-byte page table entries, or PTEs.

Todo

verify GT215 transition for medium pages

The PDEs are made of two 32-bit LE words, and have the following format:

word 0:

- bits 0-1: page table presence and page size
  - 0: page table not present
  - 1: large pages [64kiB]
  - 2: medium pages [16kiB] [GT215-]
  - 3: small pages [4kiB]
- bits 2-3: target specifier for the page table itself
  - 0: VRAM
  - 1: invalid, do not use
  - 2: SYSRAM_SNOOP
  - 3: SYSRAM_NOSNOOP
- bit 4: ??? [XXX: figure this out]
- bits 5-6: page table size [small pages only]
  - 0: 0x20000 entries [full]
  - 1: 0x8000 entries

    - 2: 0x4000 entries
    - 3: 0x2000 entries
- bits 12-31: page table linear address bits 12-31

word 1:

- bits 32-39: page table linear address bits 32-39

The page table start address has to be aligned to 0x1000 bytes.

The PTEs are made of two 32-bit LE words, and have the following format:

word 0:

- bit 0: page present
- bits 1-2: ??? [XXX: figure this out]
- bit 3: read-only flag
- bits 4-5: target specifier
    - 0: VRAM
    - 1: invalid, do not use
    - 2: SYSRAM_SNOOP
    - 3: SYSRAM_NOSNOOP
- bit 6: supervisor-only flag
- bits 7-9: log2 of contig block size in pages [see below]
- bits 12-31: bits 12-31 of linear address [small pages]
- bits 14-31: bits 14-31 of linear address [medium pages]
- bits 16-31: bits 16-31 of linear address [large pages]

word 1:

- bits 32-39: bits 32-39 of linear address
- bits 40-46: storage type
- bits 47-48: compression mode
- bits 49-60: compression tag address
- bit 61: partition cycle
    - 0: short cycle
    - 1: long cycle
- bit 62 [G84-]: encryption flag

Contig blocks are a special feature of PTEs used to save TLB space. When 2^o adjacent pages starting on 2^o page aligned boundary map to contiguous linear addresses [and, if appropriate, contiguous tag addresses] and have identical other attributes, they can be marked as a contig block of order o, where o is 0-7. To do this, all PTEs for that range should have bits 7-9 set equal to o, and linear/tag address fields set to the linear/tag address of the *first* page in the contig block [ie. all PTEs belonging to contig block should be identical]. The starting linear address need not be aligned to contig block size, but virtual address has to be.

TLB flushes

The page table contents are cached in per-engine TLBs. To flush TLB contents, the TLB flush register 0x100c80 should be used:

MMIO 0x100c80:

- bit 0: trigger. When set, triggers the TLB flush. Will auto-reset to 0 when flush is complete.
- bits 16-19: VM engine to flush

A flush consists of writing engine << 16 | 1 to this register and waiting until bit 0 becomes 0. However, note that G86 PGRAPH has a bug that can result in a lockup if PGRAPH TLB flush is initiated while PGRAPH is running, see graph/g80-pgraph.txt for details.

User vs supervisor accesses

Todo

write me

Storage types

Todo

write me

Compression modes

Todo

write me

VM faults

Todo

write me

2.7.9 G80:GF100 VRAM structure and usage

Contents

Introduction

The basic structure of G80 memory is similiar to other card generations and is described in *Memory structure*.

There are two sub-generations of G80 memory controller: the original G80 one and the GT215 one. The G80 memory controller was designed for DDR2 and GDDR3 memory. It's split into several [1-8] partitions, each of them having 64-bit memory bus. The GT215 memory controller added support for DDR3 and GDDR5 memory and split the partitions into two subpartitions, each of them having 32-bit memory bus.

On G80, the combination of DDR2/GDDR3 [ie. 4n prefetch] memory with 64-bit memory bus results in 32-byte minimal transfer size. For that reason, 32-byte units are called sectors. On GT215, DDR3/GDDR5 [ie. 8n prefetch] memory with 32-bit memory bus gives the same figure.

Next level of granularity for memory is 256-byte gobs. Memory is always assigned to partitions in units of whole gobs - all addresses in a gob will stay in a single partition. Also, format dependent memory address reordering is applied within a gob.

The final fixed level of VRAM granularity is a 0x10000-byte [64kiB] large page. While G80 VM supports using smaller page sizes for VRAM, certain features [compression, long partition cycle] should only be enabled on per-large page basis.

Apart from VRAM, the memory controller uses so-called tag RAM, which is used for compression. Compression is a feature that allows a memory block to be stored in a more efficient manner [eg. using 2 sectors instead of the normal 8] if its contents are sufficiently regular. The tag RAM is used to store the compression information for each block: whether it's compressed, and if so, in what way. Note that compression is only meant to save memory bandwidth, not memory capacity: the sectors saved by compression don't have to be transmitted over the memory link, but they're still assigned to that block and cannot be used for anything else. The tag RAM is allocated in units of tag cells, which have varying size depending on the partition number, but always correspond to 1 or 2 large pages, depending on format.

VRAM is addressed by 32-bit linear addresses. Some memory attributes affecting low-level storage are stored together with the linear address in the page tables [or linear DMA object]. These are:

- storage type: a 7-bit enumerated value that describes the memory purpose and low-level storage within a block, and also selects whether normal or alternative bank cycle is used
- compression mode: a 2-bit field selecting whether the memory is:
  - not compressed,
  - compressed with 2 tag bits per block [1 tag cell per large page], or
  - compressed with 4 tag bits per block [2 tag cells per large page]
- compression tag cell: a 12-bit index into the available tag memory, used for compressed memory
- partition cycle: a 1-bit field selecting whether the short [1 block] or long [4 blocks] partition cycle is used

The linear addresses are transformed in the following steps:

1. The address is split into the block index [high 24 bits], and the offset inside the block [low 8 bits].
2. The block index is transformed to partition id and partition block index. The process depends on whether the storage type is blocklinear or pitch and the partition cycle selected. If compression is enabled, the tag cell index is also translated to partition tag bit index.
3. [GT215+ only] The partition block index is translated into subpartition ID and subpartition block index. If compression is enabled, partition tag bit index is also translated to subpartition tag bit index.
4. [Sub]partition block index is split into row/bank/column fields.
5. Row and bank indices are transformed according to the bank cycle. This process depends on whether the storage type selects the normal or alternate bank cycle.
6. Depending on storage type and the compression tag contents, the offset in the block may refer to varying bytes inside the block, and the data may be transformed due to compression. When the required transformed block offsets have been determined, they're split into the remaining low column bits and offset inside memory word.

Partition cycle

Partition cycle is the first address transformation. Its purpose is converting linear [global] addressing to partition index and per-partition addressing. The inputs to this process are:

- the block index [ie. bits 8-31 of linear VRAM address]
- partition cycle selected [short or long]
- pitch or blocklinear mode - pitch is used when storage type is PITCH, blocklinear for all other storage types
- partition count in the system [as selected by PBUS HWUNITS register]

The outputs of this process are:

- partition ID
- partition block index

Partition pre-ID and ID adjust are intermediate values in this process.

On G80 [and G80 only], there are two partition cycles available: short one and long one. The short one switches partitions every block, while the long one switches partitions roughly every 4 blocks. However, to make sure addresses don't "bleed" between large page bounduaries, long partition cycle reverts to switching partitions every block near large page bounduaries:

```
if partition_cycle == LONG and gpu == G80:
    # round down to 4 * partition_count multiple
    group_start = block_index / (4 * partition_count) * 4 * partition_count
    group_end = group_start + 4 * partition_count - 1
    # check whether the group is entirely within one large page
    use_long_cycle = (group_start & ~0xff) == (group_end & ~0xff)
else:
    use_long_cycle = False
```

On G84+, long partition cycle is no longer supported - short cycle is used regardless of the setting.

Todo

verify it's really the G84

When short partition cycle is selected, the partition pre-ID and partition block index are calculated by simple division. The partition ID adjust is low 5 bits of partition block index:

```
if not use_long_cycle:
    partition_preid = block_index % partition_count
    partition_block_index = block_index / partition_count
    partition_id_adjust = partition_block_index & 0x1f
```

When long partition cycle is selected, the same calculation is performed, but with bits 2-23 of block index, and the resulting partition block index is merged back with bits 0-1 of block index:

```
if use_long_cycle:
    quadblock_index = block_index >> 2
    partition_preid = quadblock_index % partition_count
    partition_quadblock_index = quadblock_index / partition_count
    partition_id_adjust = partition_quadblock_index & 0x1f
    partition_block_index = partition_quadblock_index << 2 | (block_index & 3)
```

Finally, the real partition ID is determined. For pitch mode, the partition ID is simply equal to the partition pre-ID. For blocklinear mode, the partition ID is adjusted as follows:

- for 1, 3, 5, or 7-partition GPUs: no change [partition ID = partition pre-ID]
- for 2 or 6-partition GPUs: XOR together all bits of partition ID adjust, then XOR the partition pre-ID with the resulting bit to get the partition ID
- for 4-partition GPUs: add together bits 0-1, bits 2-3, and bit 4 of partition ID adjust, substract it from partition pre-ID, and take the result modulo 4. This is the partition ID.
- for 8-partition GPUs: add together bits 0-2 and bits 3-4 of partition ID adjust, substract it from partition pre-ID, and take the result modulo 8. This is the partition ID.

In summary:

```
if blocklinear or partition_count in [1, 3, 5, 7]:
    partition_id = partition_preid
elif partition_count in [2, 6]:
    xor = 0
    for bit in range(5):
        xor ^= partition_id_adjust >> bit & 1
    partition_id = partition_preid ^ xor
elif partition_count == 4:
    sub = partition_id_adjust & 3
    sub += partition_id_adjust >> 2 & 3
    sub += partition_id_adjust >> 4 & 1
    partition_id = (partition_preid - sub) % 4
elif partition_count == 8:
    sub = partition_id_adjust & 7
    sub += partition_id_adjust >> 3 & 3
    partition_id = (partition_preid - sub) % 8
```

Tag memory addressing

Todo

write me

Subpartition cycle

On GT215+, once the partition block index has been determined, it has to be further transformed to subpartition ID and subpartition block index. On G80, this step doesn't exist - partitions are not split into subpartitions, and "subpartition" in further steps should be taken to actually refer to a partition.

The inputs to this process are:

- partition block index
- subpartition select mask
- subpartition count

The outputs of this process are:

- subpartition ID
- subpartition block index

The subpartition configuration is stored in the following register:

MMIO 0x100268: [GT215-]

- bits 8-10: SELECT_MASK, a 3-bit value affecting subpartition ID selection.
- bits 16-17: ???
- bits 28-29: ENABLE_MASK, a 2-bit mask of enabled subpartitions. The only valid values are 1 [only subpartition 0 enabled] and 3 [both subpartitions enabled].

When only one subpartition is enabled, the subpartition cycle is effectively a NOP - subpartition ID is 0, and subpartition block index is same as partition block index. When both subpartitions are enabled, The subpartition block index is the partition block index shifted right by 1, and the subpartition ID is based on low 14 bits of partition block index:

```
if subpartition_count == 1:
    subpartition_block_index = partition_block_index
    subpartition_id = 0
else:
    subpartition_block_index = partition_block_index >> 1
    # bit 0 and bits 4-13 of the partition block index always used for
    # subpartition ID selection
    subpartition_select_bits = partition_block_index & 0x3ff1
    # bits 1-3 of partition block index only used if enabled by the select
    # mask
    subpartition_select_bits |= partition_block_index & (subpartition_select_mask << 1)
    # subpartition ID is a XOR of all the bits of subpartition_select_bits
    subpartition_id = 0
    for bit in range(14):
        subpartition_id ^= subpartition_select_bits >> bit & 1
```

Todo

tag stuff?

Row/bank/column split

Todo

write me

Bank cycle

Todo

write me

Storage types

Todo

write me

2.7.10 G80 VRAM compression

Contents

Todo

write me

Introduction

Todo

write me

2.7.11 G80:GF100 P2P memory access

Contents

Todo

write me

Introduction

Todo

write me

MMIO registers

Todo

write me

2.7.12 G80:GF100 BAR1 remapper

Contents

Todo

write me

Introduction

Todo

write me

MMIO registers

Todo

write me

2.7.13 GF100 virtual memory

Contents

Todo

write me

Introduction

Todo

write me

2.7.14 GF100- VRAM structure and usage

Contents

Todo

write me

Introduction

Todo

write me

2.7.15 GF100 VRAM compression

Contents

Todo

write me

Introduction

Todo

write me

2.8 PFIFO: command submission to execution engines

Contents:

2.8.1 FIFO overview

Contents

Introduction

Commands to most of the engines are sent through a special engine called PFIFO. PFIFO maintains multiple fully independent command queues, known as "channels" or "FIFO"s. Each channel is controlled through a "channel control area", which is a region of MMIO [pre-GF100] or VRAM [GF100+]. PFIFO intercepts all accesses to that area and acts upon them.

PFIFO internally does time-sharing between the channels, but this is transparent to the user applications. The engines that PFIFO controls are also aware of channels, and maintain separate context for each channel.

The context-switching ability of PFIFO depends on card generation. Since NV40, PFIFO is able to switch between channels at essentially any moment. On older cards, due to lack of backing storage for the CACHE, a switch is only possible when the CACHE is empty. The PFIFO-controlled engines are, however, much worse at switching: they can only switch between commands. While this wasn't a big problem on old cards, since the commands were guaranteed to execute in finite time, introduction of programmable shaders with looping capabilities made it possible to effectively hang the whole GPU by launching a long-running shader.

Todo

check if it still holds on GF100

On NV1:NV4, the only engine that PFIFO controls is PGRAPH, the main 2d/3d engine of the card. In addition, PFIFO can submit commands to the SOFTWARE pseudo-engine, which will trigger an interrupt for every submitted method.

The engines that PFIFO controls on NV4:GF100 are:

Id	Present on	Name	Description
0	all	SOFTWARE	Not really an engine, causes interrupt for each command, can be used to execute driver functions in sync with other commands.
1	all	*PGRAPH*	Main engine of the card: 2d, 3d, compute.
2	NV31:G98 G200:MCP77	*PMPEG*	The PFIFO interface to VPE MPEG2 decoding engine.
3	NV40:G84	PME	VPE motion estimation engine.
4	NV41:G84	PVP1	VPE microcoded vector processor.
4	VP2	PVP2	xtensa-microcoded vector processor.
5	VP2	PCIPHER	AES cryptography and copy engine.
6	VP2	PBSP	xtensa-microcoded bitstream processor.
2	VP3-	PPPP	falcon-based video post-processor.
4	VP3-	PPDEC	falcon-based microcoded video decoder.
5	VP3	PSEC	falcon-based AES crypto engine. On VP4, merged into PVLD.
6	VP3-	PVLD	falcon-based variable length decoder.
3	GT215-	PCOPY	falcon-based memory copy engine.
5	MCP89:GF100	PVCOMP	falcon-based video compositing engine.

The engines that PFIFO controls on GF100- are:

Id	Id	Id	Id	Id	Present on	Name	Description
GF100	GK104	GK208	GK20A	GM107			
1f	1f	1f	1f	1f	all	SOFTWARE	Not really an engine, causes interrupt for each command, can be used to execute driver functions in sync with other commands.
0	0	0	0	0	all	*PGRAPH*	Main engine of the card: 2d, 3d, compute.
1	1	1	?	-	GF100:GM107	PPDEC	falcon-based microcoded picture decoder.
2	2	2	?	-	GF100:GM107	PPPP	falcon-based video post-processor.
3	3	3	?	-	GF100:GM107	PVLD	falcon-based variable length decoder.
4,5	-	-	-	-	GF100:GK104	PCOPY	falcon-based memory copy engines.
-	6	5	?	2	GK104:	PVENC	falcon-based H.264 encoding engine.
-	4,5,7	4,-.6	?	4,-.5	GK104:	PCOPY	Memory copy engines.
-	-	-	?	1	GM107:	PVDEC	falcon-based unified video decoding engine
-	-	-	?	3	GM107:	PSEC	falcon-based AES crypto engine, recycled

This file deals only with the user-visible side of the PFIFO. For kernel-side programming, see nv1-pfifo, nv4-pfifo, g80-pfifo, or gf100-pfifo.

Note: GF100 information can still be very incomplete / not exactly true.

Overall operation

The PFIFO can be split into roughly 4 pieces:

- PFIFO pusher: collects user's commands and injects them to
- PFIFO CACHE: a big queue of commands waiting for execution by

- PFIFO puller: executes the commands, passes them to the proper engine, or to the driver.
- PFIFO switcher: ticks out the time slices for the channels and saves / restores the state of the channels between PFIFO registers and RAMFC memory.

A channel consists of the following:

- channel mode: PIO [NV1:GF100], DMA [NV4:GF100], or IB [G80-]
- PFIFO *DMA pusher* state [DMA and IB channels only]
- PFIFO CACHE state: the commands already accepted but not yet executed
- PFIFO *puller* state
- RAMFC: area of VRAM storing the above when channel is not currently active on PFIFO [not user-visible]
- RAMHT [pre-GF100 only]: a table of "objects" that the channel can use. The objects are identified by arbitrary 32-bit handles, and can be DMA objects [see *NV3 DMA objects*, *NV4:G80 DMA objects*, *DMA objects*] or engine objects [see *Puller - handling of submitted commands by FIFO* and engine documentation]. On pre-G80 cards, individual objects can be shared between channels.
- vspace [G80+ only]: A hierarchy of page tables that describes the virtual memory space visible to engines while executing commands for the channel. Multiple channels can share a vspace. [see *Tesla virtual memory*, *GF100 virtual memory*]
- engine-specific state

Channel mode determines the way of submitting commands to the channel. PIO mode is available on pre-GF100 cards, and involves poking the methods directly to the channel control area. It's slow and fragile - everything breaks down easily when more than one channel is used simultanously. Not recommended. See *PIO submission to FIFOs* for details. On NV1:NV40, all channels support PIO mode. On NV40:G80, only first 32 channels support PIO mode. On G80:GF100 only channel 0 supports PIO mode.

Todo

check PIO channels support on NV40:G80

NV1 PFIFO doesn't support any DMA mode.

NV3 PFIFO introduced a hacky DMA mode that requires kernel assistance for every submitted batch of commands and prevents channel switching while stuff is being submitted. See nv3-pfifo-dma for details.

NV4 PFIFO greatly enhanced the DMA mode and made it controllable directly through the channel control area. Thus, commands can now be submitted by multiple applications simultaneously, without coordination with each other and without kernel's help. DMA mode is described in *DMA submission to FIFOs on NV4*.

G80 introduced IB mode. IB mode is a modified version of DMA mode that, instead of following a single stream of commands from memory, has the ability to stitch together parts of multiple memory areas into a single command stream - allowing constructs that submit commands with parameters pulled directly from memory written by earlier commands. IB mode is described along with DMA mode in *DMA submission to FIFOs on NV4*.

GF100 rearchitectured the whole PFIFO, made it possible to have up to 3 channels executing simultaneously, and introduced a new DMA packet format.

The commands, as stored in CACHE, are tuples of:

- subchannel: 0-7
- method: 0-0x1ffc [really 0-0x7ff] pre-GF100, 0-0x3ffc [really 0-0xfff] GF100+
- parameter: 0-0xffffffff

- submission mode [NV10+]: I or NI

Subchannel identifies the engine and object that the command will be sent to. The subchannels have no fixed assignments to engines/objects, and can be freely bound/rebound to them by using method 0. The "objects" are individual pieces of functionality of PFIFO-controlled engine. A single engine can expose any number of object types, though most engines only expose one.

The method selects an individual command of the object bound to the selected subchannel, except methods 0-0xfc which are special and are executed directly by the puller, ignoring the bound object. Note that, traditionally, methods are treated as 4-byte addressable locations, and hence their numbers are written down multiplied by 4: method 0x3f thus is written as 0xfc. This is a leftover from PIO channels. In the documentation, whenever a specific method number is mentioned, it'll be written pre-multiplied by 4 unless specified otherwise.

The parameter is an arbitrary 32-bit value that accompanies the method.

The submission mode is I if the command was submitted through increasing DMA packet, or NI if the command was submitted through non-increasing packet. This information isn't actually used for anything by the card, but it's stored in the CACHE for certain optimisation when submitting PGRAPH commands.

Method execution is described in detail in *DMA puller* and engine-specific documentation.

Pre-NV1A, PFIFO treats everything as little-endian. NV1A introduced big-endian mode, which affects pushbuffer/IB reads and semaphores. On NV1A:G80 cards, the endianness can be selected per channel via the big_endian flag. On G80+ cards, PFIFO endianness is a global switch.

Todo

look for GF100 PFIFO endian switch

The channel control area endianness is not affected by the big_endian flag or G80+ PFIFO endianness switch. Instead, it follows the PMC MMIO endianness switch.

Todo

is it still true for GF100, with VRAM-backed channel control area?

2.8.2 PIO submission to FIFOs

Contents

Todo

write me

Introduction

Todo

write me

MMIO areas

Todo

write me

Channel submission area

Todo

write me

Free space determination

Todo

write me

RAMRO

Todo

write me

2.8.3 DMA submission to FIFOs on NV4

Contents

Introduction

There are two modes of DMA command submission: The NV4-style DMA mode and IB mode.

Both of them are based on a conception of "pushbuffer": an area of memory that user fills with commands and tells PFIFO to process. The pushbuffers are then assembled into a "command stream" consisting of 32-bit words that make up "commands". In NV4-style DMA mode, the pushbuffer is always read linearly and converted directly to command stream, except when the "jump", "return", or "call" commands are encountered. In IB mode, the jump/call/return commands are disabled, and command stream is instead created with use of an "IB buffer". The IB buffer is a circular buffer of (base,length) pairs describing areas of pushbuffer that will be stitched together to create the command stream. NV4- style mode is available on NV4:GF100, IB mode is available on G80+.

Todo

check for NV4-style mode on GF100

In both cases, the command stream is then broken down to commands, which get executed. For most commands, the execution consists of storing methods into CACHE for execution by the puller.

Pusher state

The following data makes up the DMA pusher state:

type	name	cards	description
dmaobj	dma_pushbuffer	:GF100	[1] the pushbuffer and IB DMA object
b32	dma_limit	:GF100	[1] [2] pushbuffer size limit
b32	dma_put	all	pushbuffer current end address
b32	dma_get	all	pushbuffer current read address
b11/12	dma_state.mthd	all	Current method
b3	dma_state.subc	all	Current subchannel
b24	dma_state.mcnt	all	Current method count
b32	dcount_shadow	NV5:	number of already-processed methods in cmd
bool	dma_state.ni	NV10+	Current command's NI flag

Continued on next page

Table 2.7 – continued from previous page

type	name	cards	description
bool	dma_state.lenp	G80+	[3] Large NI command length pending
b32	ref	NV10+	reference counter [shared with puller]
bool	subr_active	NV1A+	[2] Subroutine active
b32	subr_return	NV1A+	[2] subroutine return address
bool	big_endian	NV11:G80	[1] pushbuffer endian switch
bool	sli_enable	G80+	[1] SLI cond command enabled
b12	sli_mask	G80+	[1] SLI cond mask
bool	sli_active	NV40+	SLI cond currently active
bool	ib_enable	G80+	[1] IB mode enabled
bool	nonmain	G80+	[3] non-main pushbuffer active
b8	dma_put_high	G80+	extra 8 bits for dma_put
b8	dma_put_high_rs	G80+	dma_put_high read shadow
b8	dma_put_high_ws	G80+	[2] dma_put_high write shadow
b8	dma_get_high	G80+	extra 8 bits for dma_get
b8	dma_get_high_rs	G80+	dma_get_high read shadow
b32	ib_put	G80+	[3] IB current end position
b32	ib_get	G80+	[3] IB current read position
b40	ib_address	G80+	[1 3] IB address
b8	ib_order	G80+	[1 3] IB size
b32	dma_mget	G80+	[3] main pushbuffer last read address
b8	dma_mget_high	G80+	[3] extra 8 bits for dma_mget
bool	dma_mget_val	G80+	[3] dma_mget valid flag
b8	dma_mget_high_rs	G80+	[3] dma_mget_high read shadow
bool	dma_mget_val_rs	G80+	[3] dma_mget_val read shadow

Errors

On pre-GF100, whenever the DMA pusher encounters problems, it'll raise a DMA_PUSHER error. There are 6 types of DMA_PUSHER errors:

id	name	reason
1	CALL_SUBR_ACTIVE	call command while subroutine active
2	INVALID_MTHD	attempt to submit a nonexistent special method
3	RET_SUBR_INACTIVE	return command while subroutine inactive
4	INVALID_CMD	invalid command
5	IB_EMPTY	attempt to submit zero-length IB entry
6	MEM_FAULT	failure to read from pushbuffer or IB

Apart from pusher state, the following values are available on NV5+ to aid troubleshooting:

- dma_get_jmp_shadow: value of dma_get before the last jump
- rsvd_shadow: the first word of last-read command
- data_shadow: the last-read data word

Todo

[1] means that this part of state can only be modified by kernel intervention and is normally set just once, on channel setup.
[2] means that state only applies to NV4-style mode,
[3] means that state only applies to IB mode.

verify those

Todo

determine what happens on GF100 on all imaginable error conditions

Channel control area

The channel control area is used to tell card about submitted pushbuffers. The area is at least 0x1000 bytes long, though it can be longer depending on the card generation. Everything in the area should be accessed as 32-bit integers, like almost all of the MMIO space. The following addresses are usable:

addr	R/W	name	description
0x40	R/W	DMA_PUT	dma_put, only writable when not in IB mode
0x44	R	DMA_GET	dma_get
0x48	R	REF	ref
0x4c	R/W	DMA_PUT_HIGH	dma_put_high_rs/ws, only writable when not in IB
0x50	R/W	???	GF100+ only
0x54	R	DMA_CGET	[2] nv40+ only, connected to subr_return when subroutine active, dma_get when inactive.
0x58	R	DMA_MGET	dma_mget
0x5c	R	DMA_MGET_HIGH	dma_mget_high_rs, dma_mget_val_rs
0x60	R	DMA_GET_HIGH	dma_get_high_rs
0x88	R	IB_GET	[3] ib_get
0x8c	R/W	IB_PUT	[3] ib_put

The channel control area is accessed in 32-bit chunks, but on G80+, DMA_GET, DMA_PUT and DMA_MGET are effectively 40-bit quantities. To prevent races, the high parts of them have read and write shadows. When you read the address corresponding to the low part, the whole value is atomically read. The low part is returned as the result of the read, while the high part is copied to the corresponding read shadow where it can be read through a second access to the other address. DMA_PUT also has a write shadow of the high part - when the low part address is written, it's assembled together with the write shadow and atomically written.

To summarise, when you want to read full DMA_PUT/GET/MGET, first read the low part, then the high part. Due to the shadows, the value thus read will be correct. To write the full value of DMA_PUT, first write the high part, then the low part.

Note, however, that two different threads reading these values simultanously can interfere with each other. For this reason, the channel control area shouldn't ever be accessed by more than one thread at once, even for reading.

On NV4:NV40 cards, the channel control area is in BAR0 at address 0x800000 + 0x10000 * channel ID. On NV40, there are two BAR0 regions with channel control areas: the old-style is in BAR0 at 0x800000 + 0x10000 * channel ID, supports channels 0-0x1f, can do both PIO and DMA submission, but does not have DMA_CGET when used in DMA mode. The new-style area is in BAR0 at 0xc0000 + 0x1000 * channel ID, supports only DMA mode, supports all channels, and has DMA_CGET. On G80 cards, channel 0 supports PIO mode and has channel control area at 0x800000, while channels 1-126 support DMA mode and have channel control areas at 0xc00000 + 0x2000 * channel ID. On GF100, the channel control areas are accessed through selectable addresses in BAR1 and are backed by VRAM or host memory - see GF100+ PFIFO for more details.

Todo

check channel numbers

NV4-style mode

In NV4-style mode, whenever dma_get != dma_put, the card read a 32-bit word from the pushbuffer at the address specified by dma_get, increments dma_get by 4, and treats the word as the next word in the command stream. dma_get can also move through the control flow commands: jump [sets dma_get to param], call [copies dma_get to subr_return, sets subr_active and sets dma_get to param], and return [unsets subr_active, copies subr_return to dma_get]. The calls and returns are only available on NV1A+ cards.

The pushbuffer is accessed through the dma_pushbuffer DMA object. On NV4, the DMA object has to be located in PCI or AGP memory. On NV5+, any DMA object is valid. At all times, dma_get has to be <= dma_limit. Going past the limit or getting a VM fault when attempting to read from pushbuffer results in raising DMA_PUSHER error of type MEM_FAULT.

On pre-NV1A cards, the word read from pushbuffer is always treated as little-endian. On NV1A:G80 cards, the endianness is determined by the big_endian flag. On G80+, the PFIFO endianness is a global switch.

Todo

What about GF100?

Note that pushbuffer addresses over 0xffffffff shouldn't be used in NV4-style mode, even on G80 - they cannot be expressed in jump commands, dma_limit, nor subr_return. Why dma_put writing supports it is a mystery.

The usual way to use NV4-style mode is:

1. Allocate a big circular buffer
2. [NV1A+] if you intend to use subroutines, allocate space for them and write them out
3. Point dma_pushbuffer to the buffer, set dma_get and dma_put to its start
4. To submit commands:
   (a) If there's not enough space in the pushbuffer between dma_put and end to fit the command + a jump command, submit a jump-to-beginning command first and set DMA_PUT to buffer start.
   (b) Read DMA_GET/DMA_CGET until you get a value that's out of the range you're going to write. If on pre-NV40 and using subroutines, discard DMA_GET reads that are outside of the main buffer.
   (c) Write out the commands at current DMA_PUT address.
   (d) Set DMA_PUT to point right after the last word of commands you wrote.

IB mode

NV4-style mode, while fairly flexible, can only jump between parts of pushbuffer between commands. IB mode decouples flow control from the command structure by using a second "master" buffer, called the IB buffer.

The IB buffer is a circular buffer of 8-byte structures called IB entries. The IB buffer is, like the pushbuffer, accessed through dma_pushbuffer DMA object. The address of the IB buffer, along with its size, is normally specified on channel creation. The size has to be a power of two and can be in range ???.

Todo

check the ib size range

There are two indices into the IB buffer: ib_get and ib_put. They're both in range of 0..2^ib_order-1. Whenever no pushbuffer is being processed [dma_put =dma_get], and there are unread entries in the IB buffer [ib_put!=ib_get], the

card will read an entry from IB buffer entry #ib_get and increment ib_get by 1. When ib_get would reach 2^ib_order, it insteads wraps around to 0.

Failure to read IB entry due to VM fault will, like pushbuffer read fault, cause DMA_PUSHER error of type MEM_FAULT.

The IB entry is made of two 32-bit words in PFIFO endianness. Their format is:

Word 0:

- bits 0-1: unused, should be 0
- bits 2-31: ADDRESS_LOW, bits 2-31 of pushbuffer start address

Word 1:

- bits 0-7: ADDRESS_HIGH, bits 32-39 of pushbuffer start address
- bit 8: ???
- bit 9: NOT_MAIN, "not main pushbuffer" flag
- bits 10-30: SIZE, pushbuffer size in 32-bit words
- bit 31: NO_PREFETCH (probably; use for pushbuffer data generated by the GPU)

Todo

figure out bit 8 some day

When an IB entry is read, the pushbuffer is prepared for reading:

```
dma_get[2:39] = ADDRESS
dma_put = dma_get + SIZE * 4
nonmain = NOT_MAIN
if (!nonmain) dma_mget = dma_get
```

Subsequently, just like in NV4-style mode, words from dma_get are read until it reaches dma_put. When that happens, processing can move on to the next IB entry [or pause until user sends more commands]. If the nonmain flag is not set, dma_get is copied to dma_mget whenever it's advanced, and dma_mget_val flag is set to 1. dma_limit is ignored in IB mode.

An attempt to submit IB entry with length zero will raise DMA_PUSHER error of type IB_EMPTY.

The nonmain flag is meant to help with a common case where pushbuffers sent through IB can come from two sources: a "main" big circular buffer filled with immediately generated commands, and "external" buffers containing helper data filled and managed through other means. DMA_MGET will then contain the address of the current position in the "main" buffer without being affected by IB entries pulling data from other pushbuffers. It's thus similiar to DMA_CGET's role in NV4-style mode.

The commands - pre-GF100 format

The command stream, as assembled by NV4-style or IB mode pushbuffer read, is then split into individual commands. The command type is determined by its first word. The word has to match one of the following forms:

000CCCCCCCCCC00SSSMMMMMMMMMMM00	increasing methods [NV4+]
0000000000000001MMMMMMMMMMMMXX00	SLI conditional [NV40+, if enabled]
00000000000000100000000000000000	return [NV1A+, NV4-style only]
0000000000000011SSSMMMMMMMMMM00	long non-increasing methods [IB only]
001JJJJJJJJJJJJJJJJJJJJJJJJ00	old jump [NV4+, NV4-style only]
010CCCCCCCCCC00SSSMMMMMMMMMM00	non-increasing methods [NV10+]
JJJJJJJJJJJJJJJJJJJJJJJJJJ01	jump [NV1A+, NV4-style only]
JJJJJJJJJJJJJJJJJJJJJJJJJJ10	call [NV1A+, NV4-style only]

Todo

do an exhaustive scan of commands

If none of the forms matches, or if the one that matches cannot be used in current mode, the INVALID_CMD DMA_PUSHER error is raised.

The commands

There are two command formats the DMA pusher can use: NV4 format and GF100 format. All cards support the NV4 format, while only GF100+ cards support the GF100 format.

NV4 method submission commands

000CCCCCCCCCC00SSSMMMMMMMMMM00	increasing methods [NV4+]
010CCCCCCCCCC00SSSMMMMMMMMMM00	non-increasing methods [NV10+]
0000000000000011SSSMMMMMMMMMM00	long non-increasing methods [IB only]

These three commands are used to submit methods. the MM..M field selects the first method that will be submitted. The SSS field selects the subchannel. The CC..C field is mthd_count and says how many words will be submitted. With the "long non-increasing methods" command, the method count is instead contained in low 24 bits of the next word in the pushbuffer.

The subsequent mthd_count words after the first word [or second word in case of the long command] are the method parameters to be submitted. If command type is increasing methods, the method number increases by 4 [ie. by 1 method] for each submitted word. If type is non-increasing, all words are submitted to the same method.

If sli_enable is set and sli_active is not set, the methods thus assembled will be discarded. Otherwise, they'll be appended to the CACHE.

Todo

didn't mthd 0 work even if sli_active=0?

The pusher watches the submitted methods: it only passes methods 0x100+ and methods in 0..0xfc range that the puller recognises. An attempt to submit invalid method in 0..0xfc range will cause a DMA_PUSHER error of type INVALID_MTHD.

Todo

check pusher reaction on ACQUIRE submission: pause?

NV4 control flow commands

001JJJJJJJJJJJJJJJJJJJJJJJJJ00	old jump [NV4+]
JJJJJJJJJJJJJJJJJJJJJJJJJJ01	jump [NV1A+]
JJJJJJJJJJJJJJJJJJJJJJJJJJ10	call [NV1A+]
00000000000000100000000000000000	return [NV1A+]

For jumps and calls, J..JJ is bits 2-28 or 2-31 of the target address. The remaining bits of target are forced to 0.

The jump commands simply set dma_get to the target - the next command will be read from there. There are two commands, since NV4 originally supported only 29-bit addresses, and used high bits as command type. NV1A introduced the new jump command that instead uses low bits as type, and allows access to full 32 bits of address range.

The call command copies dma_get to subr_return, sets subr_active to 1, and sets dma_get to the target. If subr_active is already set before the call, the DMA_PUSHER error of type CALL_SUBR_ACTIVE is raised.

The return command copies subr_return to dma_get and clears subr_active. If subr_active isn't set, it instead raises DMA_PUSHER error of type RET_SUBR_INACTIVE.

NV4 SLI conditional command

0000000000000001MMMMMMMMMMMMXX00	SLI conditional [NV40+]

NV40 introduced SLI functionality. One of the associated features is the SLI conditional command. In SLI mode, sister channels are commonly created on all cards in SLI set using a common pushbuffer. Since most of the commands set in SLI will be identical for all cards, this saves resources. However, some of the commands have to be sent only to a single card, or to a subgroup of cards. The SLI conditional can be used for that purpose.

The sli_active flag determines if methods should be accepted at the moment: when it's set, methods will be accepted. Otherwise, they'll be ignored. SLI conditional command takes the encoded mask, MM..M, ands it with the per-card value of sli_mask, and sets sli_active flag to 1 if result if non-0, to 0 otherwise.

The sli_enable flag determines if the command is available. If it's not set, the command effectively doesn't exist. Note that sli_enable and sli_mask exist on both NV40:G80 and G80+, but on NV40:G80 they have to be set uniformly for all channels on the card, while G80+ allows independent settings for each channel.

The XX bits in the command are ignored.

GF100 commands

GF100 format follows the same idea, but uses all-new command encoding.

000CCCCCCCCCCC00SSSMMMMMMMMMMMXX	increasing methods [old]
000XXXXXXXXXX01MMMMMMMMMMMMXXXX	SLI conditional
000XXXXXXXXXX10MMMMMMMMMMMMXXXX	SLI user mask store [new]
000XXXXXXXXXX11XXXXXXXXXXXXXXXX	SLI conditional from user mask [new]
001CCCCCCCCCCCCSSSXMMMMMMMMMMMM	increasing methods [new]
010CCCCCCCCCCC00SSSMMMMMMMMMMMXX	non-increasing methods [old]
011CCCCCCCCCCCCSSSXMMMMMMMMMMMM	non-increasing methods [new]
100VVVVVVVVVVVVSSSXMMMMMMMMMMMM	inline method [new]
101CCCCCCCCCCCCSSSXMMMMMMMMMMMM	increase-once methods [new]
110XXXXXXXXXXXXXXXXXXXXXXXXXXXX	??? [XXX] [new]

Todo

check bitfield bounduaries

Todo

check the extra SLI bits

Todo

look for other forms

Increasing and non-increasing methods work like on older cards. Increase-once methods is a new command that works like the other methods commands, but sends the first data word to method M, second and all subsequent data words to method M+4 [ie. the next method].

Inline method command is a single-word command that submits a single method with a short [12-bit] parameter encoded in VV..V field.

GF100 also did away with the INVALID_MTHD error - invalid low methods are pushed into CACHE as usual, puller will complain about them instead when it tries to execute them.

The pusher pseudocode - pre-GF100

```
while(1) {
        if (dma_get != dma_put) {
                /* pushbuffer non-empty, read a word. */
                b32 word;
                try {
                        if (!ib_enable && dma_get >= dma_limit)
                                throw DMA_PUSHER(MEM_FAULT);
                        if (gpu < NV1A)
                                word = READ_DMAOBJ_32(dma_pushbuffer, dma_get, LE);
                        else if (gpu < G80)
                                word = READ_DMAOBJ_32(dma_pushbuffer, dma_get, big_endian?BE:LE);
                        else
                                word = READ_DMAOBJ_32(dma_pushbuffer, dma_get, pfifo_endian);
                        dma_get += 4;
                        if (!nonmain)
                                dma_mget = dma_get;
                } catch (VM_FAULT) {
                        throw DMA_PUSHER(MEM_FAULT);
                }
                /* now, see if we're in the middle of a command */
                if (dma_state.lenp) {
                        /* second word of long non-inc methods command - method count */
                        dma_state.lenp = 0;
                        dma_state.mcnt = word & 0xffffff;
                } else if (dma_state.mcnt) {
                        /* data word of methods command */
                        data_shadow = word;
                        if (!PULLER_KNOWS_MTHD(dma_state.mthd))
                                throw DMA_PUSHER(INVALID_MTHD);
                        if (!sli_enable || sli_active) {
                                CACHE_PUSH(dma_state.subc, dma_state.mthd, word, dma_state.ni);
                        }
```

```
                if (!dma_state.ni)
                        dma_state.mthd++;
                dma_state.mcnt--;
                dcount_shadow++;
        } else {
                /* no command active - this is the first word of a new one */
                rsvd_shadow = word;
                /* match all forms */
                if ((word & 0xe0000003) == 0x20000000 && !ib_enable) {
                        /* old jump */
                        dma_get_jmp_shadow = dma_get;
                        dma_get = word & 0x1fffffff;
                } else if ((word & 3) == 1 && !ib_enable && gpu >= NV1A) {
                        /* jump */
                        dma_get_jmp_shadow = dma_get;
                        dma_get = word & 0xfffffffc;
                } else if ((word & 3) == 2 && !ib_enable && gpu >= NV1A) {
                        /* call */
                        if (subr_active)
                                throw DMA_PUSHER(CALL_SUBR_ACTIVE);
                        subr_return = dma_get;
                        subr_active = 1;
                        dma_get = word & 0xfffffffc;
                } else if (word == 0x00020000 && !ib_enable && gpu >= NV1A) {
                        /* return */
                        if (!subr_active)
                                throw DMA_PUSHER(RET_SUBR_INACTIVE);
                        dma_get = subr_return;
                        subr_active = 0;
                } else if ((word & 0xe0030003) == 0) {
                        /* increasing methods */
                        dma_state.mthd = (word >> 2) & 0x7ff;
                        dma_state.subc = (word >> 13) & 7;
                        dma_state.mcnt = (word >> 18) & 0x7ff;
                        dma_state.ni = 0;
                        dcount_shadow = 0;
                } else if ((word & 0xe0030003) == 0x40000000 && gpu >= NV10) {
                        /* non-increasing methods */
                        dma_state.mthd = (word >> 2) & 0x7ff;
                        dma_state.subc = (word >> 13) & 7;
                        dma_state.mcnt = (word >> 18) & 0x7ff;
                        dma_state.ni = 1;
                        dcount_shadow = 0;
                } else if ((word & 0xffff0003) == 0x00030000 && ib_enable) {
                        /* long non-increasing methods */
                        dma_state.mthd = (word >> 2) & 0x7ff;
                        dma_state.subc = (word >> 13) & 7;
                        dma_state.lenp = 1;
                        dma_state.ni = 1;
                        dcount_shadow = 0;
                } else if ((word & 0xffff0003) == 0x00010000 && sli_enable) {
                        if (sli_mask & ((word >> 4) & 0xfff))
                                sli_active = 1;
                        else
                                sli_active = 0;
                } else {
                        throw DMA_PUSHER(INVALID_CMD);
                }
```

```
                }
        } else if (ib_enable && ib_get != ib_put) {
                /* current pushbuffer empty, but we have more IB entries to read */
                b64 entry;
                try {
                        entry_low = READ_DMAOBJ_32(dma_pushbuffer, ib_address + ib_get * 8, pfifo_enc
                        entry_high = READ_DMAOBJ_32(dma_pushbuffer, ib_address + ib_get * 8 + 4, pfi
                        entry = entry_high << 32 | entry_low;
                        ib_get++;
                        if (ib_get == (1 << ib_order))
                                ib_get = 0;
                } catch (VM_FAULT) {
                        throw DMA_PUSHER(MEM_FAULT);
                }
                len = entry >> 42 & 0x3fffff;
                if (!len)
                        throw DMA_PUSHER(IB_EMPTY);
                dma_get = entry & 0xfffffffffc;
                dma_put = dma_get + len * 4;
                if (entry & 1 << 41)
                        nonmain = 1;
                else
                        nonmain = 0;
        }
        /* otherwise, pushbuffer empty and IB empty or nonexistent - nothing to do. */
}
```

2.8.4 Puller - handling of submitted commands by FIFO

Contents

Introduction

PFIFO puller's job is taking methods out of the CACHE and delivering them to the right place for execution, or executing them directly.

Methods 0-0xfc are special and executed by the puller. Methods 0x100 and up are forwarded to the engine object currently bound to a given subchannel. The methods are:

Method	Present on	Name	Description
0x0000	all	OBJECT	*Binds an engine object*
0x0008	GF100-	NOP	*Does nothing*
0x0010	G84-	SEMAPHORE_ADDRESS_HIGH	*New-style semaphore address high part*
0x0014	G84-	SEMAPHORE_ADDRESS_LOW	*New-style semaphore address low part*
0x0018	G84-	SEMAPHORE_SEQUENCE	*New-style semaphore payload*
0x001c	G84-	SEMAPHORE_TRIGGER	*New-style semaphore trigger*
0x0020	G84-	NOTIFY_INTR	*Triggers an interrupt*
0x0024	G84-	WRCACHE_FLUSH	*Flushes write post caches*
0x0028	MCP89-	???	???
0x002c	MCP89-	???	???
0x0050	NV10-	REF_CNT	*Writes the ref counter*
0x0060	NV1A:GF100	DMA_SEMAPHORE	*DMA object for semaphores*
0x0064	NV1A-	SEMAPHORE_OFFSET	*Old-style semaphore address*
0x0068	NV1A-	SEMAPHORE_ACQUIRE	*Old-style semaphore acquire trigger and payload*
0x006c	NV1A-	SEMAPHORE_RELEASE	*Old-style semaphore release trigger and payload*
0x0070	GF100-	???	???
0x0074	GF100-	???	???
0x0078	GF100-	???	???
0x007c	GF100-	???	???
0x0080	NV40-	YIELD	*Yield PFIFO - force channel switch*
0x0100:0x2000	NV1:NV4	...	Passed down to the engine
0x0100:0x0180	NV4:GF100	...	Passed down to the engine
0x0180:0x0200	NV4:GF100	...	Passed down to the engine, goes through RAMHT lookup
0x0200:0x2000	NV4:GF100	...	Passed down to the engine
0x0100:0x4000	GF100-	...	Passed down to the engine

Todo

missing the GF100+ methods

RAMHT and the FIFO objects

As has been already mentioned, each channel has 8 "subchannels" which can be bound to engine objects. On pre-GF100 GPUs, these objects and DMA objects are collectively known as "FIFO objects". FIFO objects and RAMHT don't exist on GF100+ PFIFO.

The RAMHT is a big hash table that associates arbitrary 32-bit handles with FIFO objects and engine ids. Whenever a method is mentioned to take an object handle, it means the parameter is looked up in RAMHT. When such lookup fails to find a match, a CACHE_ERROR(NO_HASH) error is raised.

NV4:GF100

Internally, a FIFO object is a [usually small] block of data residing in "instance memory". The instance memory is RAMIN for pre-G80 GPUs, and the channel structure for G80+ GPUs. The first few bits of a FIFO object determine its 'class'. Class is 8 bits on NV4:NV25, 12 bits on NV25:NV40, 16 bits on NV40:GF100.

The data associated with a handle in RAMHT consists of engine id, which determines the object's behavior when bound to a subchannel, and its address in RAMIN [pre-G80] or offset from channel structure start [G80+].

Apart from method 0, the engine id is ignored. The suitability of an object for a given method is determined by reading its class and checking if it makes sense. Most methods other than 0 expect a DMA object, although a couple

of pre-G80 graph objects have methods that expect other graph objects.

The following are commonly accepted object classes:

- 0x0002: DMA object for reading
- 0x0003: DMA object for writing
- 0x0030: NULL object - used to effectively unbind a previously bound object
- 0x003d: DMA object for reading/writing

Other object classes are engine-specific.

For more information on DMA objects, see *NV3 DMA objects*, *NV4:G80 DMA objects*, or *DMA objects*.

NV3

NV3 also has RAMHT, but it's only used for engine objects. While NV3 has DMA objects, they have to be bound manually by the kernel. Thus, they're not mentioned in RAMHT, and the 0x180-0x1fc methods are not implemented in hardware - they're instead trapped and emulated in software to behave like NV4+.

NV3 also doesn't use object classes - the object type is instead a 7-bit number encoded in RAMHT along with engine id and object address.

NV1

You don't want to know how NV1 RAMHT works.

Puller state

type	name	GPUs	description
b24[8]	ctx	NV1:NV4	objects bound to subchannels
b3	last_subc	NV1:NV4	last used subchannel
b5[8]	engines	NV4+	engines bound to subchannels
b5	last_engine	NV4+	last used engine
b32	ref	NV10+	reference counter [shared with pusher]
bool	acquire_active	NV1A+	semaphore acquire in progress
b32	acquire_timeout	NV1A+	semaphore acquire timeout
b32	acquire_timestamp	NV1A+	semaphore acquire timestamp
b32	acquire_value	NV1A+	semaphore acquire value
dmaobj	dma_semaphore	NV11:GF100	semaphore DMA object
b12/16	semaphore_offset	NV11:GF100	old-style semaphore address
bool	semaphore_off_val	G80:GF100	semaphore_offset valid
b40	semaphore_address	G84+	new-style semaphore address
b32	semaphore_sequence	G84+	new-style semaphore value
bool	acquire_source	G84:GF100	semaphore acquire address selection
bool	acquire_mode	G84+	semaphore acquire mode

GF100 state is likely incomplete.

Engine objects

The main purpose of the puller is relaying methods to the engines. First, an engine object has to be bound to a subchannel using method 0. Then, all methods >=0x100 on the subchannel will be forwarded to the relevant engine.

On pre-NV4, the bound objects' RAMHT information is stored as part of puller state. The last used subchannel is also remembered and each time the puller is requested to submit commands on subchannel different from the last one, method 0 is submitted, or channel switch occurs, the information about the object will be forwarded to the engine through its method 0. The information about an object is 24-bit, is known as object's "context", and has the following fields:

- bits 0-15 [NV1]: object flags
- bits 0-15 [NV3]: object address
- bits 16-22: object type
- bit 23: engine id

The context for objects is stored directly in their RAMHT entries.

On NV4+ GPUs, the puller doesn't care about bound objects - this information is supposed to be stored by the engine itself as part of its state. The puller only remembers what engine each subchannel is bound to. On NV4:GF100 When method 0 is executed, the puller looks up the object in RAMHT, getting engine id and object address in return. The engine id is remembered in puller state, while object address is passed down to the engine for further processing.

GF100+ did away with RAMHT. Thus, method 0 now takes the object class and engine id directly as parameters:

- bits 0-15: object class. Not used by the puller, simply passed down to the engine.
- bits 16-20: engine id

The list of valid engine ids can be found on *FIFO overview*. The SOFTWARE engine is special: all methods submitted to it, explicitely or implicitely by binding a subchannel to it, will cause a CACHE_ERROR(EMPTY_SUBCHANNEL) interrupt. This interrupt can then be intercepted by the driver to implement a "software object", or can be treated as an actual error and reported.

The engines run asynchronously. The puller will send them commands whenever they have space in their input queues and won't wait for completion of a command before sending more. However, when engines are switched [ie. puller has to submit a command to a different engine than last used by the channel], the puller will wait until the last used engine is done with this channel's commands. Several special puller methods will also wait for engines to go idle.

Todo

verify this on all card families.

On NV4:GF100 GPUs, methods 0x180-0x1fc are treated specially: while other methods are forwarded directly to engine without modification, these methods are expected to take object handles as parameters and will be looked up in RAMHT by the puller before forwarding. Ie. the engine will get the object's address found in RAMHT.

mthd 0x0000 / 0x000: OBJECT On NV1:GF100, takes the handle of the object that should be bound to the subchannel it was submitted on. On GF100+, it instead takes engine+class directly.

```
if (gpu < NV4) {
        b24 newctx = RAMHT_LOOKUP(param);
        if (newctx & 0x800000) {
                /* engine == PGRAPH */
                if (ENGINE_CUR_CHANNEL(PGRAPH) != chan)
                        ENGINE_CHANNEL_SWITCH(PGRAPH, chan);
                ENGINE_SUBMIT_MTHD(PGRAPH, subc, 0, newctx);
                ctx[subc] = newctx;
                last_subc = subc;
        } else {
                /* engine == SOFTWARE */
                while (!ENGINE_IDLE(PGRAPH))
                        ;
```

```
                throw CACHE_ERROR(EMPTY_SUBCHANNEL);
        }
} else {
        /* NV4+ GPU */
        b5 engine; b16 eparam;
        if (gpu >= GF100) {
                eparam = param & 0xffff;
                engine = param >> 16 & 0x1f;
                /* XXX: behavior with more bitfields? does it forward the whole thing? */
        } else {
                engine = RAMHT_LOOKUP(param).engine;
                eparam = RAMHT_LOOKUP(param).addr;
        }
        if (engine != last_engine) {
                while (ENGINE_CUR_CHANNEL(last_engine) == chan && !ENGINE_IDLE(last_engine))
                        ;
        }
        if (engine == SOFTWARE) {
                throw CACHE_ERROR(EMPTY_SUBCHANNEL);
        } else {
                if (ENGINE_CUR_CHANNEL(engine) != chan)
                        ENGINE_CHANNEL_SWITCH(engine, chan);
                ENGINE_SUBMIT_MTHD(engine, subc, 0, eparam);
                last_engine = engines[subc] = engine;
        }
}
```

mthd 0x0100-0x3ffc / 0x040-0xfff: [forwarded to engine]

```
if (gpu < NV4) {
        if (subc != last_subc) {
                if (ctx[subc] & 0x800000) {
                        /* engine == PGRAPH */
                        if (ENGINE_CUR_CHANNEL(PGRAPH) != chan)
                                ENGINE_CHANNEL_SWITCH(PGRAPH, chan);
                        ENGINE_SUBMIT_MTHD(PGRAPH, subc, 0, ctx[subc]);
                        last_subc = subc;
                } else {
                        /* engine == SOFTWARE */
                        while (!ENGINE_IDLE(PGRAPH))
                                ;
                        throw CACHE_ERROR(EMPTY_SUBCHANNEL);
                }
        }
        if (ctx[subc] & 0x800000) {
                /* engine == PGRAPH */
                if (ENGINE_CUR_CHANNEL(PGRAPH) != chan)
                        ENGINE_CHANNEL_SWITCH(PGRAPH, chan);
                ENGINE_SUBMIT_MTHD(PGRAPH, subc, mthd, param);
        } else {
                /* engine == SOFTWARE */
                while (!ENGINE_IDLE(PGRAPH))
                        ;
                throw CACHE_ERROR(EMPTY_SUBCHANNEL);
        }
} else {
        /* NV4+ */
        if (gpu < GF100 && mthd >= 0x180/4 && mthd < 0x200/4) {
                param = RAMHT_LOOKUP(param).addr;
```

```
        }
        if (engines[subc] != last_engine) {
                while (ENGINE_CUR_CHANNEL(last_engine) == chan && !ENGINE_IDLE(last_engine))
                        ;
        }
        if (engines[subc] == SOFTWARE) {
                throw CACHE_ERROR(EMPTY_SUBCHANNEL);
        } else {
                if (ENGINE_CUR_CHANNEL(engine) != chan)
                        ENGINE_CHANNEL_SWITCH(engine, chan);
                ENGINE_SUBMIT_MTHD(engine, subc, mthd, param);
                last_engine = engines[subc];
        }
}
```

Todo

verify all of the pseudocode...

Puller builtin methods

Syncing with host: reference counter

NV10 introduced a "reference counter". It's a per-channel 32-bit register that is writable by the puller and readable through the channel control area [see *DMA submission to FIFOs on NV4*]. It can be used to tell host which commands have already completed: after every interesting batch of commands, add a method that will set the ref counter to monotonically increasing values. The host code can then read the counter from channel control area and deduce which batches are already complete.

The method to set the reference counter is REF_CNT, and it simply sets the ref counter to its parameter. When it's executed, it'll also wait for all previously submitted commands to complete execution.

mthd 0x0050 / 0x014: REF_CNT [NV10:]

```
while (ENGINE_CUR_CHANNEL(last_engine) == chan && !ENGINE_IDLE(last_engine))
        ;
ref = param;
```

Semaphores

NV1A PFIFO introduced a concept of "semaphores". A semaphore is a 32-bit word located in memory. G84 also introduced "long" semaphores, which are 4-word memory structures that include a normal semaphore word and a timestamp.

The PFIFO semaphores can be "acquired" and "released". Note that these operations are NOT the familiar P/V semaphore operations, they're just fancy names for "wait until value == X" and "write X".

There are two "versions" of the semaphore functionality. The "old-style" semaphores are implemented by NV1A:GF100 GPUs. The "new-style" semaphores are supported by G84+ GPUs. The differences are:

Old-style semaphores

- limitted addressing range: 12-bit [NV1A:G80] or 16-bit [G80:GF100] offset in a DMA object. Thus a special DMA object is required.

- release writes a single word
- acquire supports only “wait for value equal to X” mode

New-style semaphores

- full 40-bit addressing range
- release writes word + timestamp, ie. long semaphore
- acquire supports “wait for value equal to X” and “wait for value greater or equal X” modes

Semaphores have to be 4-byte aligned. All values are stored with endianness selected by big_endian flag [NV1A:G80] or by PFIFO endianness [G80+]

On pre-GF100, both old-style semaphores and new-style semaphores use the DMA object stored in dma_semaphore, which can be set through DMA_SEMAPHORE method. Note that this method is buggy on pre-G80 GPUs and accepts only *write-only* DMA objects of class 0x0002. You have to work around the bug by preparing such DMA objects [or using a kernel that intercepts the error and does the binding manually].

Old-style semaphores read/write the location specified in semaphore_offset, which can be set by SEMAPHORE_OFFSET method. The offset has to be divisible by 4 and fit in 12 bits [NV1A:G80] or 16 bits [G80:GF100]. An acquire is triggered by using the SEMAPHORE_ACQUIRE mthd with the expected value as the parameter - further command processing will halt until the memory location contains the selected value. A release is triggered by using the SEMAPHORE_RELEASE method with the value as parameter - the value will be written into the semaphore location.

New-style semaphores use the location specified in semaphore_address, whose low/high parts can be set through SEMAPHORE_ADDRESS_HIGH and _LOW methods. The value for acquire/release is stored in semaphore_sequence and specified by SEMAPHORE_SEQUENCE method. Acquire and release are triggered by using the SEMAPHORE_TRIGGER method with the requested operation as parameter.

The new-style release operation writes the following 16-byte structure to memory at semaphore_address:

- 0x00: [32-bit] semaphore_sequence
- 0x04: [32-bit] 0
- 0x08: [64-bit] PTIMER timestamp [see ptimer]

The new-style “acquire equal” operation behaves exactly like old-style acquire, but uses semaphore_address instead of semaphore_offset and semaphore_sequence instead of SEMAPHORE_RELEASE param. The “acquire greater or equal” operation, instead of waiting for the semaphore value to be equal to semaphore_sequence, it waits for value that satisfies (int32_t)(val - semaphore_sequence) >= 0, ie. for a value that’s greater or equal to semaphore_sequence in 32-bit wrapping arithmetic. The “acquire mask” operation waits for a value that, ANDed with semaphore_sequence, gives a non-0 result [GF100+ only].

Failures of semaphore-related methods will trigger the SEMAPHORE error. The SEMAPHORE error has several subtypes, depending on card generation.

NV1A:G80 SEMAPHORE error subtypes:

- 1: INVALID_OPERAND: wrong parameter to a method
- 2: INVALID_STATE: attempt to acquire/release without proper setup

G80:GF100 SEMAPHORE error subtypes:

- 1: ADDRESS_UNALIGNED: address not divisible by 4
- 2: INVALID_STATE: attempt to acquire/release without proper setup
- 3: ADDRESS_TOO_LARGE: attempt to set >40-bit address or >16-bit offset
- 4: MEM_FAULT: got VM fault when reading/writing semaphore

GF100 SEMAPHORE error subtypes:

Todo

figure this out

If the acquire doesn't immediately succeed, the acquire parameters are written to puller state, and the read will be periodically retried. Further puller processing will be blocked on current channel until acquire succeeds. Note that, on G84+ GPUs, the retry reads are issued from SEMAPHORE_BG VM engine instead of the PFIFO VM engine. There's also apparently a timeout, but it's not REd yet.

Todo

RE timeouts

mthd 0x0060 / 0x018: DMA_SEMAPHORE [O] [NV1A:GF100]

```
obj = RAMHT_LOOKUP(param).addr;
if (gpu < G80) {
        if (OBJECT_CLASS(obj) != 2)
                throw SEMAPHORE(INVALID_OPERAND);
        if (DMAOBJ_RIGHTS(obj) != WO)
                throw SEMAPHORE(INVALID_OPERAND);
        if (!DMAOBJ_PT_PRESENT(obj))
                throw SEMAPHORE(INVALID_OPERAND);
}
/* G80 doesn't bother with verification */
dma_semaphore = obj;
```

Todo

is there ANY way to make G80 reject non-DMA object classes?

mthd 0x0064 / 0x019: SEMAPHORE_OFFSET [NV1A-]

```
if (gpu < G80) {
        if (param & ~0xffc)
                throw SEMAPHORE(INVALID_OPERAND);
        semaphore_offset = param;
} else if (gpu < GF100) {
        if (param & 3)
                throw SEMAPHORE(ADDRESS_UNALIGNED);
        if (param & 0xffff0000)
                throw SEMAPHORE(ADDRESS_TOO_LARGE);
        semaphore_offset = param;
        semaphore_off_val = 1;
} else {
        semaphore_address[0:31] = param;
}
```

mthd 0x0068 / 0x01a: SEMAPHORE_ACQUIRE [NV1A-]

```
if (gpu < G80 && !dma_semaphore)
        /* unbound DMA object */
        throw SEMAPHORE(INVALID_STATE);
if (gpu >= G80 && !semaphore_off_val)
```

```
		throw SEMAPHORE(INVALID_STATE);
	b32 word;
	if (gpu < G80) {
		word = READ_DMAOBJ_32(dma_semaphore, semaphore_offset, big_endian?BE:LE);
	} else {
		try {
			word = READ_DMAOBJ_32(dma_semaphore, semaphore_offset, pfifo_endian);
		} catch (VM_FAULT) {
			throw SEMAPHORE(MEM_FAULT);
		}
	}
	if (word == param) {
		/* already done */
	} else {
		/* acquire_active will block further processing and schedule retries */
		acquire_active = 1;
		acquire_value = param;
		acquire_timestamp = ???;
		/* XXX: figure out timestamp/timeout business */
		if (gpu >= G80) {
			acquire_mode = 0;
			acquire_source = 0;
		}
	}
```

mthd 0x006c / 0x01b: SEMAPHORE_RELEASE [NV1A-]

```
	if (gpu < G80 && !dma_semaphore)
		/* unbound DMA object */
		throw SEMAPHORE(INVALID_STATE);
	if (gpu >= G80 && !semaphore_off_val)
		throw SEMAPHORE(INVALID_STATE);
	if (gpu < G80) {
		WRITE_DMAOBJ_32(dma_semaphore, semaphore_offset, param, big_endian?BE:LE);
	} else {
		try {
			WRITE_DMAOBJ_32(dma_semaphore, semaphore_offset, param, pfifo_endian);
		} catch (VM_FAULT) {
			throw SEMAPHORE(MEM_FAULT);
		}
	}
```

mthd 0x0010 / 0x004: SEMAPHORE_ADDRESS_HIGH [G84:]

```
	if (param & 0xffffff00)
		throw SEMAPHORE(ADDRESS_TOO_LARGE);
	semaphore_address[32:39] = param;
```

mthd 0x0014 / 0x005: SEMAPHORE_ADDRESS_LOW [G84:]

```
	if (param & 3)
		throw SEMAPHORE(ADDRESS_UNALIGNED);
	semaphore_address[0:31] = param;
```

mthd 0x0018 / 0x006: SEMAPHORE_SEQUENCE [G84:]

```
	semaphore_sequence = param;
```

mthd 0x001c / 0x007: SEMAPHORE_TRIGGER [G84:]

bits 0-2: operation

- 1: ACQUIRE_EQUAL
- 2: WRITE_LONG
- 4: ACQUIRE_GEQUAL
- 8: ACQUIRE_MASK [GF100-]

Todo

bit 12 does something on GF100?

```
op = param & 7;
b64 timestamp = PTIMER_GETTIME();
if (param == 2) {
        if (gpu < GF100) {
                try {
                        WRITE_DMAOBJ_32(dma_semaphore, semaphore_address+0x0, param, pfifo_endia
                        WRITE_DMAOBJ_32(dma_semaphore, semaphore_address+0x4, 0, pfifo_endian);
                        WRITE_DMAOBJ_64(dma_semaphore, semaphore_address+0x8, timestamp, pfifo_e
                } catch (VM_FAULT) {
                        throw SEMAPHORE(MEM_FAULT);
                }
        } else {
                WRITE_VM_32(semaphore_address+0x0, param, pfifo_endian);
                WRITE_VM_32(semaphore_address+0x4, 0, pfifo_endian);
                WRITE_VM_64(semaphore_address+0x8, timestamp, pfifo_endian);
        }
} else {
        b32 word;
        if (gpu < GF100) {
                try {
                        word = READ_DMAOBJ_32(dma_semaphore, semaphore_address, pfifo_endian);
                } catch (VM_FAULT) {
                        throw SEMAPHORE(MEM_FAULT);
                }
        } else {
                word = READ_VM_32(semaphore_address, pfifo_endian);
        }
        if ((op == 1 && word == semaphore_sequence) || (op == 4 && (int32_t)(word - semaphore_se
                /* already done */
        } else {
                /* XXX GF100 */
                acquire_source = 1;
                acquire_value = semaphore_sequence;
                acquire_timestamp = ???;
                if (op == 1) {
                        acquire_active = 1;
                        acquire_mode = 0;
                } else if (op == 4) {
                        acquire_active = 1;
                        acquire_mode = 1;
                } else {
                        /* invalid combination - results in hang */
                }
        }
}
```

Misc puller methods

NV40 introduced the YIELD method which, if there are any other busy channels at the moment, will cause PFIFO to switch to another channel immediately, without waiting for the timeslice to expire.

mthd 0x0080 / 0x020: YIELD [NV40:]

:: PFIFO_YIELD();

G84 introduced the NOTIFY_INTR method, which simply raises an interrupt that notifies the host of its execution. It can be used for sync primitives.

mthd 0x0020 / 0x008: NOTIFY_INTR [G84:]

:: PFIFO_NOTIFY_INTR();

Todo

check how this is reported on GF100

The G84+ WRCACHE_FLUSH method can be used to flush PFIFO's write post caches. [see *Tesla virtual memory*]

mthd 0x0024 / 0x009: WRCACHE_FLUSH [G84:]

:: VM_WRCACHE_FLUSH(PFIFO);

The GF100+ NOP method does nothing:

mthd 0x0008 / 0x002: NOP [GF100:]

```
    /* do nothing */
```

2.9 PGRAPH: 2d/3d graphics and compute engine

Contents:

2.9.1 PGRAPH overview

Contents

Introduction

Todo

write me

Todo

WAIT_FOR_IDLE and PM_TRIGGER

NV1/NV3 graph object types

The following graphics objects exist on NV1:NV4:

id	variants	name	description
0x01	all	BETA	*sets beta factor for blending*
0x02	all	ROP	*sets raster operation*
0x03	all	CHROMA	*sets color for color key*
0x04	all	PLANE	*sets the plane mask*
0x05	all	CLIP	*sets clipping rectangle*
0x06	all	PATTERN	*sets pattern, ie. a small repeating image used as one of the inputs to a raster operation or blending*
0x07	NV3:NV4	RECT	*renders solid rectangles*
0x08	all	POINT	*renders single points*
0x09	all	LINE	*renders solid lines*
0x0a	all	LIN	*renders solid lins [ie. lines missing a pixel on one end]*
0x0b	all	TRI	*renders solid triangles*
0x0c	NV1:NV3	RECT	*renders solid rectangles*
0x0c	NV3:NV4	GDI	*renders Windows 95 primitives: rectangles and characters, with font read from a DMA object*
0x0d	NV1:NV3	TEXLIN	*renders quads with linearly mapped textures*
0x0d	NV3:NV4	M2MF	*copies data from one DMA object to another*
0x0e	NV1:NV3	TEXQUAD	*renders quads with quadratically mapped textures*
0x0e	NV3:NV4	SIFM	*Scaled Image From Memory, like NV1's IFM, but with scaling*
0x10	all	BLIT	*copies rectangles of pixels from one place in framebuffer to another*
0x11	all	IFC	*Image From CPU, uploads a rectangle of pixels via methods*
0x12	all	BITMAP	*uploads and expands a bitmap [ie. 1bpp image] via methods*
0x13	NV1:NV3	IFM	*Image From Memory, uploads a rectangle of pixels from a DMA object to framebuffer*
0x14	all	ITM	*Image To Memory, downloads a rectangle of pixels to a DMA object from framebuffer*
0x15	NV3:NV4	SIFC	*Stretched Image From CPU, like IFC, but with image stretching*
0x17	NV3:NV4	D3D	*Direct3D 5 textured triangles*
0x18	NV3:NV4	ZPOINT	*renders single points to a surface with depth buffer*
0x1c	NV3:NV4	SURF	*sets rendering surface parameters*
0x1d	NV1:NV3	TEXLIN-BETA	*renders lit quads with linearly mapped textures*
0x1e	NV1:NV3	TEXQUAD-BETA	*renders lit quads with quadratically mapped textures*

Todo

check Direct3D version

NV4+ graph object classes

Not really graph objects, but usable as parameters for some object-bind methods [all NV4:GF100]:

class	name	description
0x0030	NV1_NULL	does nothing
0x0002	NV1_DMA_R	*DMA object for reading*
0x0003	NV1_DMA_W	*DMA object for writing*
0x003d	NV3_DMA	*read/write DMA object*

Todo

document NV1_NULL

NV1-style *operation objects* [all NV4:NV5]:

class	name	description
0x0010	NV1_OP_CLIP	clipping
0x0011	NV1_OP_BLEND_AND	blending
0x0013	NV1_OP_ROP_AND	raster operation
0x0015	NV1_OP_CHROMA	color key
0x0064	NV1_OP_SRCCOPY_AND	source copy with 0-alpha discard
0x0065	NV3_OP_SRCCOPY	source copy
0x0066	NV4_OP_SRCCOPY_PREMULT	pre-multiplying copy
0x0067	NV4_OP_BLEND_PREMULT	pre-multiplied blending

Memory to memory copy objects:

class	variants	name	description
0x0039	NV4:G80	NV3_M2MF	*copies data from one buffer to another*
0x5039	G80:GF100	G80_M2MF	*copies data from one buffer to another*
0x9039	GF100:GK104	GF100_M2MF	*copies data from one buffer to another*
0xa040	GK104:GK110 GK20A	GK104_P2MF	*copies data from FIFO to memory buffer*
0xa140	GK110:GK20A GM107-	GK110_P2MF	*copies data from FIFO to memory buffer*

Context objects:

class	variants	name	description
0x0012	NV4:G84	NV1_BETA	*sets beta factor for blending*
0x0017	NV4:G80	NV1_CHROMA	*sets color for color key*
0x0057	NV4:G84	NV4_CHROMA	*sets color for color key*
0x0018	NV4:G80	NV1_PATTERN	*sets pattern for raster op*
0x0044	NV4:G84	NV1_PATTERN	*sets pattern for raster op*
0x0019	NV4:G84	NV1_CLIP	*sets user clipping rectangle*
0x0043	NV4:G84	NV1_ROP	*sets raster operation*
0x0072	NV4:G84	NV4_BETA4	*sets component beta factors for pre-multiplied blending*
0x0058	NV4:G80	NV3_SURF_DST	*sets the 2d destination surface*
0x0059	NV4:G80	NV3_SURF_SRC	*sets the 2d blit source surface*
0x005a	NV4:G80	NV3_SURF_COLOR	*sets the 3d color surface*
0x005b	NV4:G80	NV3_SURF_ZETA	*sets the 3d zeta surface*
0x0052	NV4:G80	NV4_SWZSURF	*sets 2d swizzled destination surface*
0x009e	NV10:G80	NV10_SWZSURF	*sets 2d swizzled destination surface*
0x039e	NV30:NV40	NV30_SWZSURF	*sets 2d swizzled destination surface*
0x309e	NV40:G80	NV30_SWZSURF	*sets 2d swizzled destination surface*
0x0042	NV4:G80	NV4_SURF2D	*sets 2d destination and source surfaces*
0x0062	NV10:G80	NV10_SURF2D	*sets 2d destination and source surfaces*
0x0362	NV30:NV40	NV30_SURF2D	*sets 2d destination and source surfaces*
0x3062	NV40:G80	NV30_SURF2D	*sets 2d destination and source surfaces*
0x5062	G80:G84	G80_SURF2D	*sets 2d destination and source surfaces*
0x0053	NV4:NV20	NV4_SURF3D	*sets 3d color and zeta surfaces*
0x0093	NV10:NV20	NV10_SURF3D	*sets 3d color and zeta surfaces*

Solids rendering objects:

class	variants	name	description
0x001c	NV4:NV40	NV1_LIN	*renders a lin*
0x005c	NV4:G80	NV4_LIN	*renders a lin*
0x035c	NV30:NV40	NV30_LIN	*renders a lin*
0x305c	NV40:G84	NV30_LIN	*renders a lin*
0x001d	NV4:NV40	NV1_TRI	*renders a triangle*
0x005d	NV4:G84	NV4_TRI	*renders a triangle*
0x001e	NV4:NV40	NV1_RECT	*renders a rectangle*
0x005e	NV4:NV40	NV4_RECT	*renders a rectangle*

Image upload from CPU objects:

class	variants	name	description
0x0021	NV4:NV40	NV1_IFC	*image from CPU*
0x0061	NV4:G80	NV4_IFC	*image from CPU*
0x0065	NV5:G80	NV5_IFC	*image from CPU*
0x008a	NV10:G80	NV10_IFC	*image from CPU*
0x038a	NV30:NV40	NV30_IFC	*image from CPU*
0x308a	NV40:G84	NV40_IFC	*image from CPU*
0x0036	NV4:G80	NV1_SIFC	*stretched image from CPU*
0x0076	NV4:G80	NV4_SIFC	*stretched image from CPU*
0x0066	NV5:G80	NV5_SIFC	*stretched image from CPU*
0x0366	NV30:NV40	NV30_SIFC	*stretched image from CPU*
0x3066	NV40:G84	NV40_SIFC	*stretched image from CPU*
0x0060	NV4:G80	NV4_INDEX	*indexed image from CPU*
0x0064	NV5:G80	NV5_INDEX	*indexed image from CPU*
0x0364	NV30:NV40	NV30_INDEX	*indexed image from CPU*
0x3064	NV40:G84	NV40_INDEX	*indexed image from CPU*
0x007b	NV10:G80	NV10_TEXTURE	*texture from CPU*
0x037b	NV30:NV40	NV30_TEXTURE	*texture from CPU*
0x307b	NV40:G80	NV40_TEXTURE	*texture from CPU*

Todo

figure out wtf is the deal with TEXTURE objects

Other 2d source objects:

class	variants	name	description
0x001f	NV4:G80	NV1_BLIT	*blits inside framebuffer*
0x005f	NV4:G84	NV4_BLIT	*blits inside framebuffer*
0x009f	NV15:G80	NV15_BLIT	*blits inside framebuffer*
0x0037	NV4:G80	NV3_SIFM	*scaled image from memory*
0x0077	NV4:G80	NV4_SIFM	*scaled image from memory*
0x0063	NV10:G80	NV5_SIFM	*scaled image from memory*
0x0089	NV10:NV40	NV10_SIFM	*scaled image from memory*
0x0389	NV30:NV40	NV30_SIFM	*scaled image from memory*
0x3089	NV40:G80	NV30_SIFM	*scaled image from memory*
0x5089	G80:G84	G80_SIFM	*scaled image from memory*
0x004b	NV4:NV40	NV3_GDI	*draws GDI primitives*
0x004a	NV4:G80	NV4_GDI	*draws GDI primitives*

YCbCr two-source blending objects:

class	variants	name
0x0038	NV4:G80	NV4_DVD_SUBPICTURE
0x0088	NV10:G80	NV10_DVD_SUBPICTURE

Todo

find better name for these two

Unified 2d objects:

class	variants	name
0x502d	G80:GF100	G80_2D
0x902d	GF100-	GF100_2D

NV3-style 3d objects:

class	variants	name	description
0x0048	NV4:NV15	NV3_D3D	*Direct3D textured triangles*
0x0054	NV4:NV20	NV4_D3D5	*Direct3D 5 textured triangles*
0x0094	NV10:NV20	NV10_D3D5	*Direct3D 5 textured triangles*
0x0055	NV4:NV20	NV4_D3D6	*Direct3D 6 multitextured triangles*
0x0095	NV10:NV20	NV10_D3D6	*Direct3D 6 multitextured triangles*

Todo

check NV3_D3D version

NV10-style 3d objects:

class	variants	name	description
0x0056	NV10:NV30	NV10_3D	*Celsius Direct3D 7 engine*
0x0096	NV15:NV30	NV15_3D	*Celsius Direct3D 7 engine*
0x0098	NV17:NV20	NV11_3D	*Celsius Direct3D 7 engine*
0x0099	NV17:NV20	NV17_3D	*Celsius Direct3D 7 engine*
0x0097	NV20:NV34	NV20_3D	*Kelvin Direct3D 8 SM 1 engine*
0x0597	NV25:NV40	NV25_3D	*Kelvin Direct3D 8 SM 1 engine*
0x0397	NV30:NV40	NV30_3D	*Rankine Direct3D 9 SM 2 engine*
0x0497	NV35:NV34	NV35_3D	*Rankine Direct3D 9 SM 2 engine*
0x0697	NV34:NV40	NV34_3D	*Rankine Direct3D 9 SM 2 engine*
0x4097	NV40:G80 !TC	NV40_3D	*Curie Direct3D 9 SM 3 engine*
0x4497	NV40:G80 TC	NV44_3D	*Curie Direct3D 9 SM 3 engine*
0x5097	G80:G200	G80_3D	*Tesla Direct3D 10 engine*
0x8297	G84:G200	G84_3D	*Tesla Direct3D 10 engine*
0x8397	G200:GT215	G200_3D	*Tesla Direct3D 10 engine*
0x8597	GT215:MCP89	GT215_3D	*Tesla Direct3D 10.1 engine*
0x8697	MCP89:GF100	MCP89_3D	*Tesla Direct3D 10.1 engine*
0x9097	GF100:GK104	GF100_3D	*Fermi Direct3D 11 engine*
0x9197	GF108:GK104	GF108_3D	*Fermi Direct3D 11 engine*
0x9297	GF110:GK104	GF110_3D	*Fermi Direct3D 11 engine*
0xa097	GK104:GK110	GK104_3D	*Kepler Direct3D 11.1 engine*
0xa197	GK110:GK20A	GK110_3D	*Kepler Direct3D 11.1 engine*
0xa297	GK20A:GM107	GK20A_3D	*Kepler Direct3D 11.1 engine*
0xb097	GM107-	GM107_3D	*Maxwell Direct3D 12 engine*

And the compute objects:

class	variants	name	description
0x50c0	G80:GF100	G80_COMPUTE	*CUDA 1.x engine*
0x85c0	GT215:GF100	GT215_COMPUTE	*CUDA 1.x engine*
0x90c0	GF100:GK104	GF100_COMPUTE	*CUDA 2.x engine*
0x91c0	GF110:GK104	GF110_COMPUTE	*CUDA 2.x engine*
0xa0c0	GK104:GK110 GK20A:GM107	GK104_COMPUTE	*CUDA 3.x engine*
0xa1c0	GK110:GK20A	GK110_COMPUTE	*CUDA 3.x engine*
0xb0c0	GM107:GM200	GM107_COMPUTE	*CUDA 4.x engine*
0xb1c0	GM200:-	GM200_COMPUTE	*CUDA 4.x engine*

The graphics context

Todo

write something here

Channel context

The following information makes up non-volatile graphics context. This state is per-channel and thus will apply to all objects on it, unless software does trap-swap-restart trickery with object switches. It is guaranteed to be unaffected by subchannel switches and object binds. Some of this state can be set by submitting methods on the context objects, some can only be set by accessing PGRAPH context registers.

- the beta factor - set by BETA object
- the 8-bit raster operation - set by ROP object
- the A1R10G10B10 color for chroma key - set by CHROMA object
- the A1R10G10B10 color for plane mask - set by PLANE object
- the user clip rectangle - set by CLIP object:
  - ???
- the pattern state - set by PATTERN object:
  - shape: 8x8, 64x1, or 1x64
  - 2x A8R10G10B10 pattern color
  - the 64-bit pattern itself
- the NOTIFY DMA object - pointer to DMA object used by NOTIFY methods. NV1 only - moved to graph object options on NV3+. Set by direct PGRAPH access only.
- the main DMA object - pointer to DMA object used by IFM and ITM objects. NV1 only - moved to graph object options on NV3+. Set by direct PGRAPH access only.
- On NV1, framebuffer setup - set by direct PGRAPH access only:
  - ???
- On NV3+, rendering surface setup:
  - ???

  There are 4 copies of this state, one for each surface used by PGRAPH:
  - DST - the 2d destination surface
  - SRC - the 2d source surface [used by BLIT object only]
  - COLOR - the 3d color surface
  - ZETA - the 3d depth surface

  Note that the M2MF source/destination, ITM destination, IFM/SIFM source, and D3D texture don't count as surfaces - even though they may be configured to access the same data as surfaces on NV3+, they're accessed through the DMA circuitry, not the surface circuitry, and their setup is part of volatile state.

Todo

beta factor size

Todo

user clip state

Todo

NV1 framebuffer setup

Todo

NV3 surface setup

Todo

figure out the extra clip stuff, etc.

Todo

update for NV4+

Graph object options

In addition to the per-channel state, there is also per-object non-volatile state, called graph object options. This state is stored in the RAMHT entry for the object [NV1], or in a RAMIN structure [NV3-]. On subchannel switches and object binds, the PFIFO will send this state [NV1] or the pointer to this state [NV3-] to PGRAPH via method 0. On NV1:NV4, this state cannot be modified by any object methods and requires RAMHT/RAMIN access to change. On NV4+, PGRAPH can bind DMA objects on its own when requested via methods, and update the DMA object pointers in RAMIN. On NV5+, PGRAPH can modify most of this state when requested via methods. All NV4+ automatic options modification methods can be disabled by software, if so desired.

The graph options contain the following information:

- *2d pipeline configuration*
- *2d color and mono format*
- NOTIFY_VALID flag - if set, NOTIFY method will be enabled. If unset, NOTIFY method will cause an interrupt. Can be used by the driver to emulate per-object DMA_NOTIFY setting - this flag will be set on objects whose emulated DMA_NOTIFY value matches the one currently in PGRAPH context, and interrupt will cause a switch of the PGRAPH context value followed by a method restart.
- SUBCONTEXT_ID - a single-bit flag that can be used to emulate more than one PGRAPH context on one channel. When an object is bound and its SUBCONTEXT_ID doesn't match PGRAPH's current SUBCONTEXT_ID, a context switch interrupt is raised to allow software to load an alternate context.

Todo

NV3+

See nv1-pgraph for detailed format.

Volatile state

In addition to the non-volatile state described above, PGRAPH also has plenty of "volatile" state. This state deals with the currently requested operation and may be destroyed by switching to a new subchannel or binding a new object [though not by full channel switches - the channels are supposed to be independent after all, and kernel driver is supposed to save/restore all state, including volatile state].

Volatile state is highly object-specific, but common stuff is listed here:

- the "notifier write pending" flag and requested notification type

Todo

more stuff?

Notifiers

The notifiers are 16-byte memory structures accessed via DMA objects, used for synchronization. Notifiers are written by PGRAPH when certain operations are completed. Software can poll on the memory structure, waiting for it to be written by PGRAPH. The notifier structure is:

base+0x0: 64-bit timestamp - written by PGRAPH with current PTIMER time as of the notifier write. The timestamp is a concatenation of current values of TIME_LOW and TIME_HIGH registers When big-endian mode is in effect, this becomes a 64-bit big-endian number as expected.

base+0x8: 32-bit word always set to 0 by PGRAPH. This field may be used by software to put a non-0 value for software-written error-caused notifications.

base+0xc: 32-bit word always set to 0 by PGRAPH. This is used for synchronization - the software is supposed to set this field to a non-0 value before submitting the notifier write request, then wait for it to become 0. Since the notifier fields are written in order, it is guaranteed that the whole notifier structure has been written by the time this field is set to 0.

Todo

verify big endian on non-G80

There are two types of notifiers: ordinary notifiers [NV1-] and M2MF notifiers [NV3-]. Normal notifiers are written when explicitely requested by the NOTIFY method, M2MF notifiers are written on M2MF transfer completion. M2MF notifiers cannot be turned off, thus it's required to at least set up a notifier DMA object if M2MF is used, even if the software doesn't wish to use notifiers for synchronization.

Todo

figure out NV20 mysterious warning notifiers

Todo

describe GF100+ notifiers

The notifiers are always written to the currently bound notifier DMA object. The M2MF notifiers share the DMA object with ordinary notifiers. The layout of the DMA object used for notifiers is fixed:

- 0x00: ordinary notifier #0
- 0x10: M2MF notifier [NV3-]
- 0x20: ordinary notifier #2 [NV3:NV4 only]
- 0x30: ordinary notifier #3 [NV3:NV4 only]
- 0x40: ordinary notifier #4 [NV3:NV4 only]
- 0x50: ordinary notifier #5 [NV3:NV4 only]
- 0x60: ordinary notifier #6 [NV3:NV4 only]
- 0x70: ordinary notifier #7 [NV3:NV4 only]
- 0x80: ordinary notifier #8 [NV3:NV4 only]
- 0x90: ordinary notifier #9 [NV3:NV4 only]
- 0xa0: ordinary notifier #10 [NV3:NV4 only]
- 0xb0: ordinary notifier #11 [NV3:NV4 only]
- 0xc0: ordinary notifier #12 [NV3:NV4 only]
- 0xd0: ordinary notifier #13 [NV3:NV4 only]
- 0xe0: ordinary notifier #14 [NV3:NV4 only]
- 0xf0: ordinary notifier #15 [NV3:NV4 only]

Todo

0x20 - NV20 warning notifier?

Note that the notifiers always have to reside at the very beginning of the DMA object. On NV1 and NV4+, this effectively means that only 1 notifier of each type can be used per DMA object, requiring mulitple DMA objects if more than one notifier per type is to be used, and likely requiring a dedicated DMA object for the notifiers. On NV3:NV4, up to 15 ordinary notifiers may be used in a single DMA object, though that DMA object likely still needs to be dedicated for notifiers, and only one of the notifiers supports interrupt generation.

NOTIFY method

Ordinary notifiers are requested via the NOTIFY method. Note that the NOTIFY method schedules a notifier write on completion of the method *following* the NOTIFY - NOTIFY merely sets "a notifier write is pending" state.

It is an error if a NOTIFY method is followed by another NOTIFY method, a DMA_NOTIFY method, an object bind, or a subchannel switch.

In addition to a notifier write, the NOTIFY method may also request a NOTIFY interrupt to be triggered on PGRAPH after the notifier write.

mthd 0x104: NOTIFY [all NV1:GF100 graph objects] Requests a notifier write and maybe an interrupt. The write/interrupt will be actually performed after the *next* method completes. Possible parameter values are:

0: WRITE - write ordinary notifier #0 1: WRITE_AND_AWAKEN - write ordinary notifier 0, then trigger NOTIFY

interrupt [NV3-]

2: WRITE_2 - write ordinary notifier #2 [NV3:NV4] 3: WRITE_3 - write ordinary notifier #3 [NV3:NV4] [...] 15: WRITE_15 - write ordinary notifier #15 [NV3:NV4]

Operation::

if (!cur_grobj.NOTIFY_VALID) { /* DMA notify object not set, or needs to be swapped in by sw */ throw(INVALID_NOTIFY);

} else if ((param > 0 && gpu == NV1)

|| (param > 15 && gpu >= NV3 && gpu < NV4) || (param > 1 && gpu >= NV4)) {

/* XXX: what state is changed? */ throw(INVALID_VALUE);

} else if (NOTIFY_PENDING) { /* tried to do two NOTIFY methods in row // XXX: what state is changed? */ throw(DOUBLE_NOTIFY);

} else { NOTIFY_PENDING = 1; NOTIFY_TYPE = param;

}

After every method other than NOTIFY and DMA_NOTIFY, the following is done:

```
if (NOTIFY_PENDING) {
    int idx = NOTIFY_TYPE;
    if (idx == 1)
        idx = 0;
    dma_write64(NOTIFY_DMA, idx*0x10+0x0, PTIMER.TIME_HIGH << 32 | PTIMER.TIME_LOW);
    dma_write32(NOTIFY_DMA, idx*0x10+0x8, 0);
    dma_write32(NOTIFY_DMA, idx*0x10+0xc, 0);
    if (NOTIFY_TYPE == 1)
        irq_trigger(NOTIFY);
    NOTIFY_PENDING = 0;
}
```

if a subchannel switch or object bind is done while NOTIFY_PENDING is set, CTXSW_NOTIFY error is raised.

NOTE: NV1 has a 1-bit NOTIFY_PENDING field, allowing it to do notifier writes with interrupts, but lacks support for setting it via the NOTIFY method. This functionality thus has to be emulated by the driver if needed.

DMA_NOTIFY method

On NV4+, the notifier DMA object can be bound by submitting the DMA_NOTIFY method. This functionality can be disabled by the driver in PGRAPH settings registers if not desired.

mthd 0x180: DMA_NOTIFY [all NV4:GF100 graph objects] Sets the notifier DMA object. When submitted through PFIFO, this method will undergo handle -> address translation via RAMHT.

Operation::

if (DMA_METHODS_ENABLE) { /* XXX: list the validation checks */ NOTIFY_DMA = param;

} else { throw(INVALID_METHOD);

}

NOP method

On NV4+ a NOP method was added to enable asking for a notifier write without having to submit an actual method to the object. The NOP method does nothing, but still counts as a graph object method and will thus trigger a notifier write/interrupt if one was previously requested.

mthd 0x100: NOP [all NV4+ graph objects] Does nothing.

Operation:: /* nothing */

Todo

figure out if this method can be disabled for NV1 compat

2.9.2 The memory copying objects

Contents

Introduction

Todo

write me

M2MF objects

Todo

write me

P2MF objects

Todo

write me

Input/output setup

Todo

write me

Operation

Todo

write me

2.9.3 2D pipeline

Contents:

Overview of the 2D pipeline

Contents

Introduction

On nvidia GPUs, 2d operations are done by PGRAPH engine [see graph/intro.txt]. The 2d engine is rather orthogonal and has the following features:

- various data sources:
  - solid color shapes (points, lines, triangles, rectangles)
  - pixels uploaded directly through command stream, raw or expanded using a palette
  - text with in-memory fonts [NV3:G80]
  - rectangles blitted from another area of video memory
  - pixels read by DMA
  - linearly and quadratically textured quads [NV1:NV3]
- color format conversions
- chroma key
- clipping rectangles
- per-pixel operations between source, destination, and pattern:
  - logic operations
  - alpha and beta blending
  - pre-multiplied alpha blending [NV4-]
- plane masking [NV1:NV4]
- dithering
- data output:
  - to the framebuffer [NV1:NV3]
  - to any surface in VRAM [NV3:G84]
  - to arbirary memory [G84-]

The objects

The 2d engine is controlled by the user via PGRAPH objects. On NV1:G84, each piece of 2d functionality has its own object class - a matching set of objects needs to be used together to perform an operation. G80+ have a unified 2d engine object that can be used to control all of the 2d pipeline in one place.

The non-unified objects can be divided into 3 classes:

- source objects: control the drawing operation, choose pixels to draw and their colors
- context objects: control various pipeline settings shared by other objects
- operation objects: connect source and context objects together

The source objects are:

- *POINT*, *LIN*, *LINE*, *TRI*, *RECT*: drawing of solid color shapes
- *IFC*, *BITMAP*, *SIFC*, *INDEX*, *TEXTURE*: drawing of pixel data from CPU
- *BLIT*: copying rectangles from another area of video memory

- *IFM*, *SIFM*: drawing pixel data from DMA
- *GDI*: Drawing solid rectangles and text fonts
- *TEXLIN*, *TEXQUAD*, *TEXLINBETA*, *TEXQUADBETA*: Drawing textured quads

The context objects are:

- *BETA*: blend factor
- *ROP*: logic operation
- *CHROMA*: color for chroma key
- *PLANE*: color for plane mask
- *CLIP*: clipping rectangle
- *PATTERN*: repeating pattern image [graph/pattern.txt]
- *BETA4*: pre-multiplied blend factor
- *SURF*, *SURF2D*, *SWZSURF*: destination and blit source surface setup

The operation objects are:

- OP_CLIP: clipping operation
- OP_BLEND_AND: blending
- OP_ROP_AND: logic operation
- OP_CHROMA: color key
- OP_SRCCOPY_AND: source copy with 0-alpha discard
- OP_SRCCOPY: source copy
- OP_SRCCOPY_PREMULT: pre-multiplying copy
- OP_BLEND_PREMULT: pre-multiplied blending

The unified 2d engine objects are described below.

The objects that, although related to 2d operations, aren't part of the usual 2d pipeline:

- *ITM*: downloading framebuffer data to DMA
- *M2MF*: DMA to DMA copies
- *DVD_SUBPICTURE*: blending of YUV data

Note that, although multiple objects of a single kind may be created, there is only one copy of pipeline state data in PGRAPH. There are thus two usage possibilities:

- aliasing: all objects on a channel access common pipeline state, making it mostly useless to create several objects of single kind
- swapping: the kernel driver or some other piece of software handles PGRAPH interrupts, swapping pipeline configurations as they're needed, and marking objects valid/not valid according to currently loaded configuration

Connecting the objects - NV1 style The objects were originally intended and designed for connecting with so-called patchcords. A patchcord is a dummy object that's conceptually a wire carrying some sort of data. The patchcord types are:

- image patchcord: carries pixel color data
- beta patchcord: carries beta blend factor data

- zeta patchcord: carries pixel depth data
- rop patchcord: carries logic operation data

Each 2d object has patchcord "slots" representing its inputs and outputs. A slot is represented by an object methods. Objects are connected together by creating a patchcord of appropriate type and writing its handle to the input slot method on one object and the output slot method on the other object. For example:

- source objects have an output image patchcord slot [BLIT also has input image slot]
- BETA context object has an output beta slot
- OP_BLEND_AND has two image input slots, one beta input slot, and one image output slot

A valid set of objects, called a "patch" is constructed by connecting patchcords appropriately. Not all possible connections ara valid, though. Only ones that map to the actual hardware pipeline are allowed: one of the source objects must be at the beginning, connected via image patchcord to OP_BLEND_*, OP_ROP_AND, or OP_SRCCOPY_*, optionally connected further through OP_CLIP and/or OP_CHROMA, then finally connected to a SURF object representing the destination surface. Each of the OP_* objects and source objects that needs it must also be connected to the appropriate extra inputs, like the CLIP rectangle, PATTERN or another SURF, or CHROMA key.

No GPU has ever supported connecting patchcords in hardware - the software must deal with all required processing and state swapping. However, NV4:NV20 hardware knows of the methods reserved for these purpose, and raises a special interrupt when they're called. The OP_*, while lacking in any useful hardware methods, are also supported on NV4:NV5.

Connecting the objects - NV5 style A new way of connecting objects was designed for NV5 [but can be used with earlier cards via software emulation]. Instead of treating a patch as a freeform set of objects, the patch is centered on the source object. While context objects are still in use, operation objects are skipped - the set of operations to perform is specified at the source object, instead of being implid by the patchcord topology. The context objects are now connected directly to the source object by writing their handles to appropriate source object methods. The OP_CLIP and OP_CHROMA functionality is replaced by CLIP and CHROMA methods on the source objects: enabling clipping/color keying is done by connecting appropriate context object, while disabling is done by connecting a NULL object. The remaining operation objects are replaced by OPERATION method, which takes an enum selecting the operation to perform.

NV5 added support for the NV5-style connections in hardware - all methods can be processed without software assistance as long as only one object of each type is in use [or they're allowed to alias]. If swapping is required, it's the responsibility of software. The new methods can be globally disabled if NV1-style connections are desired, however. NV5-style connections can also be implemented for older GPUs simply by handling the relevant methods in software.

Color and monochrome formats

Todo

write me

COLOR_FORMAT methods

mthd 0x300: COLOR_FORMAT [NV1_CHROMA, NV1_PATTERN] [NV4-] Sets the color format using NV1 color enum.

Operation:

```
cur_grobj.COLOR_FORMAT = get_nv1_color_format(param);
```

Todo

figure out this enum

mthd 0x300: COLOR_FORMAT [NV4_CHROMA, NV4_PATTERN] Sets the color format using NV4 color enum.

Operation:

```
cur_grobj.COLOR_FORMAT = get_nv4_color_format(param);
```

Todo

figure out this enum

Color format conversions

Todo

write me

Monochrome formats

Todo

write me

mthd 0x304: MONO_FORMAT [NV1_PATTERN] [NV4-] Sets the monochrome format.

Operation:

```
if (param != LE && param != CGA6)
    throw(INVALID_ENUM);
cur_grobj.MONO_FORMAT = param;
```

Todo

check

The pipeline

The 2d pipeline consists of the following stages, in order:

1. Image source: one of the source objects, or one of the three source types on the unified 2d objects [SOLID, SIFC, or BLIT] - see documentation of the relevant object
2. Clipping
3. Source color conversion
4. One of:

(a) Bitwise operation subpipeline, soncisting of:

(a) **Optionally, an arbitrary bitwise operation done on the source,** the destination, and the pattern.

(b) Optionally, a color key operation

(c) Optionally, a plane mask operation [NV1:NV4]

(a) Blending operation subpipeline, consisting of:

(a) Blend factor calculation

(b) Blending

2. Dithering
3. Destination write

In addition, the pipeline may be used in RGB mode [treating colors as made of R, G, B components], or index mode [treating colors as 8-bit palette index]. The pipeline mode is determined automatically by the hardware based on source and destination formats and some configuration bits.

The pixels are rendered to a destination buffer. On NV1:NV4, more than one destination buffer may be enabled at a time. If this is the case, the pixel operations are executed separately for each buffer.

Pipeline configuration: NV1 The pipeline configuration is stored in graph options and other PGRAPH registers. It cannot be changed by user-visible commands other than via rebinding objects. The following options are stored in the graph object:

- the operation, one of:
  - RPOP_DS - RPOP(DST, SRC)
  - ROP_SDD - ROP(SRC, DST, DST)
  - ROP_DSD - ROP(DST, SRC, DST)
  - ROP_SSD - ROP(SRC, SRC, DST)
  - ROP_DDS - ROP(DST, DST, SRC)
  - ROP_SDS - ROP(SRC, DST, SRC)
  - ROP_DSS - ROP(DST, SRC, SRC)
  - ROP_SSS - ROP(SRC, SRC, SRC)
  - ROP_SSS_ALT - ROP(SRC, SRC, SRC)
  - ROP_PSS - ROP(PAT, SRC, SRC)
  - ROP_SPS - ROP(SRC, PAT, SRC)
  - ROP_PPS - ROP(PAT, PAT, SRC)
  - ROP_SSP - ROP(SRC, SRC, PAT)
  - ROP_PSP - ROP(PAT, SRC, PAT)
  - ROP_SPP - ROP(SRC, PAT, PAT)
  - RPOP_SP - ROP(SRC, PAT)
  - ROP_DSP - ROP(DST, SRC, PAT)
  - ROP_SDP - ROP(SRC, DST, PAT)
  - ROP_DPS - ROP(DST, PAT, SRC)

  - ROP_PDS - ROP(PAT, DST, SRC)
  - ROP_SPD - ROP(SRC, PAT, DST)
  - ROP_PSD - ROP(PAT, SRC, DST)
  - SRCCOPY - SRC [no operation]
  - BLEND_DS_AA - BLEND(DST, SRC, SRC.ALPHA^2) [XXX check]
  - BLEND_DS_AB - BLEND(DST, SRC, SRC.ALPHA * BETA)
  - BLEND_DS_AIB - BLEND(DST, SRC, SRC.ALPHA * (1-BETA))
  - BLEND_PS_B - BLEND(PAT, SRC, BETA)
  - BLEND_PS_IB - BLEND(SRC, PAT, (1-BETA))

  If the operation is set to one of the BLEND_* values, blending subpipeline will be active. Otherwise, the bitwise operation subpipeline will be active. For bitwise operation pipeline, RPOP* and ROP* will cause the bitwise operation stage to be enabled with the appropriate options, while the SRCCOPY setting will cause it to be disabled and bypassed.

- chroma enable: if this is set to 1, and the bitwise operation subpipeline is active, the color key stage will be enabled
- plane mask enable: if this is set to 1, and the bitwise operation subpipeline is active, the plane mask stage will be enabled
- user clip enable: if set to 1, the user clip rectangle will be enabled in the clipping stage
- destination buffer mask: selects which destination buffers will be written

The following options are stored in other PGRAPH registers:

- palette bypass bit: determines the value of the palette bypass bit written to the framebuffer
- Y8 expand: determines piepline mode used with Y8 source and non-Y8 destination - if set, Y8 is upconverted to RGB and the RGB mode is used, otherwise the index mode is used
- dither enable: if set, and several conditions are fullfilled, dithering stage will be enabled
- software mode: if set, all drawing operations will trap without touching the framebuffer, allowing software to perform the operation instead

The pipeline mode is selected as follows:

- if blending subpipeline is used, RGB mode is selected [index blending is not supported]
- if bitwise operation subpipeline is used:
  - if destination format is Y8, indexed mode is selected
  - if destination format is D1R5G5B5 or D1X1R10G10B10:
    - if source format is not Y8 or Y8 expand is enabled, RGB mode is selected
    - if source format is Y8 and Y8 expand is not enabled, indexed mode is selected

In RGB mode, the pipeline internally uses 10-bit components. In index mode, 8-bit indices are used.

See nv1-pgraph for more information on the configuration registers.

Clipping

Todo

write me

Source format conversion Firstly, the source color is converted from its original format to the format used for operations.

Todo

figure out what happens on ITM, IFM, BLIT, TEX*BETA

On NV1, all operations are done on A8R10G10B10 or I8 format internally. In RGB mode, colors are converted using the standard color expansion formula. In index mode, the index is taken from the low 8 bits of the color.

```
src.B = get_color_b10(cur_grobj, color);
src.G = get_color_g10(cur_grobj, color);
src.R = get_color_r10(cur_grobj, color);
src.A = get_color_a8(cur_grobj, color);
src.I = color[0:7];
```

In addition, pixels are discarded [all processing is aborted and the destination buffer is left untouched] if the alpha component is 0 [even in index mode].

```
if (!src.A)
    discard;
```

Todo

NV3+

Buffer read In some blending and bitwise operation modes, the current contents of the destination buffer at the drawn pixel location may be used as an input to the 2d pipeline.

Todo

document that and BLIT

Bitwise operation

Todo

write me

Chroma key

Todo

write me

The plane mask

Todo

write me

Blending

Todo

write me

Dithering

Todo

write me

The framebuffer

Todo

write me

NV1 canvas

Todo

write me

NV3 surfaces

Todo

write me

Clip rectangles

Todo

write me

NV1-style operation objects

Todo

write me

Unified 2d objects

Todo

write me

0100 NOP [graph/intro.txt] 0104 NOTIFY [G80_2D] [graph/intro.txt] [XXX: GF100 methods] 0110 WAIT_FOR_IDLE [graph/intro.txt] 0140 PM_TRIGGER [graph/intro.txt] 0180 DMA_NOTIFY [G80_2D] [graph/intro.txt] 0184 DMA_SRC [G80_2D] [XXX] 0188 DMA_DST [G80_2D] [XXX] 018c DMA_COND [G80_2D] [XXX] [XXX: 0200-02ac] 02b0 PATTERN_OFFSET [graph/pattern.txt] 02b4 PATTERN_SELECT [graph/pattern.txt] 02dc ??? [GF100_2D-] [XXX] 02e0 ??? [GF100_2D-] [XXX] 02e8 PATTERN_COLOR_FORMAT [graph/pattern.txt] 02ec PATTERN_BITMAP_FORMAT [graph/pattern.txt] 02f0+i*4, i<2 PATTERN_BITMAP_COLOR [graph/pattern.txt] 02f8+i*4, i<2 PATTERN_BITMAP [graph/pattern.txt] 0300+i*4, i<64 PATTERN_X8R8G8B8 [graph/pattern.txt] 0400+i*4, i<32 PATTERN_R5G6B5 [graph/pattern.txt] 0480+i*4, i<32 PATTERN_X1R5G5B5 [graph/pattern.txt] 0500+i*4, i<16 PATTERN_Y8 [graph/pattern.txt] [XXX: 0540-08dc] 08e0+i*4, i<32 FIRMWARE [graph/intro.txt] [XXX: GF100 methods]

2D pattern

Contents

Introduction

One of the configurable inputs to the bitwise operation and, on NV1:NV4, the blending operation is the pattern. A pattern is an infinitely repeating 8x8, 64x1, or 1x64 image. There are two types of patterns:

- bitmap pattern: an arbitrary 2-color 8x8, 64x1, or 1x64 2-color image
- color pattern: an aribtrary 8x8 R8G8B8 image [NV4-]

The pattern can be set through the NV1-style *_PATTERN context objects, or through the G80-style unified 2d objects. For details on how and when the pattern is used, see *2D pattern*.

The graph context used for pattern storage is made of:

- pattern type selection: bitmap or color [NV4-]
- bitmap pattern state:
  - shape selection: 8x8, 1x64, or 64x1
  - the bitmap: 2 32-bit words
  - 2 colors: A8R10G10B10 format [NV1:NV4]
  - 2 colors: 32-bit word + format selector each [NV4:G80]

  - 2 colors: 32-bit word each [G80-]
  - color format selection [G80-]
  - bitmap format selection [G80-]
- color pattern state [NV4-]:
  - 64 colors: R8G8B8 format
- pattern offset: 2 6-bit numbers [G80-]

PATTERN objects

The PATTERN object family deals with setting up the pattern. The objects in this family are:

- objtype 0x06: NV1_PATTERN [NV1:NV4]
- class 0x0018: NV1_PATTERN [NV4:G80]
- class 0x0044: NV4_PATTERN [NV4:G84]

The methods for this family are:

0100 NOP [NV4-] [graph/intro.txt] 0104 NOTIFY [graph/intro.txt] 0110 WAIT_FOR_IDLE [G80-] [graph/intro.txt] 0140 PM_TRIGGER [NV40-?] [XXX] [graph/intro.txt] 0180 N DMA_NOTIFY [NV4-] [graph/intro.txt] 0200 O PATCH_IMAGE_OUTPUT [NV4:NV20] [see below] 0300 COLOR_FORMAT [NV4-] [see below] 0304 BITMAP_FORMAT [NV4-] [see below] 0308 BITMAP_SHAPE [see below] 030c TYPE [NV4_PATTERN] [see below] 0310+i*4, i<2 BITMAP_COLOR [see below] 0318+i*4, i<2 BITMAP [see below] 0400+i*4, i<16 COLOR_Y8 [NV4_PATTERN] [see below] 0500+i*4, i<32 COLOR_R5G6B5 [NV4_PATTERN] [see below] 0600+i*4, i<32 COLOR_X1R5G5B5 [NV4_PATTERN] [see below] 0700+i*4, i<64 COLOR_X8R8G8B8 [NV4_PATTERN] [see below]

mthd 0x200: PATCH_IMAGE_OUTPUT [*_PATTERN] [NV4:NV20] Reserved for plugging an image patch-cord to output the pattern into.

Operation: throw(UNIMPLEMENTED_MTHD);

Pattern selection

With the *_PATTERN objects, the pattern type is selected using the TYPE and BITMAP_SHAPE methods:

mthd 0x030c: TYPE [NV4_PATTERN]

Sets the pattern type. One of: 1: BITMAP 2: COLOR

Operation::

if (NV4:G80) { PATTERN_TYPE = param;

} else { SHADOW_COMP2D.PATTERN_TYPE = param; if (SHADOW_COMP2D.PATTERN_TYPE == COLOR)

PATTERN_SELECT = COLOR;

else PATTERN_SELECT = SHADOW_COMP2D.PATTERN_BITMAP_SHAPE;

}

mthd 0x308: BITMAP_SHAPE [*_PATTERN]

Sets the pattern shape. One of: 0: 8x8 1: 64x1 2: 1x64

On unified 2d objects, use the PATTERN_SELECT method instead.

Operation::

if (param > 2) throw(INVALID_ENUM);

if (NV1:G80) { PATTERN_BITMAP_SHAPE = param;

} else { SHADOW_COMP2D.PATTERN_BITMAP_SHAPE = param; if (SHADOW_COMP2D.PATTERN_TYPE == COLOR)

PATTERN_SELECT = COLOR;

else PATTERN_SELECT = SHADOW_COMP2D.PATTERN_BITMAP_SHAPE;

}

With the unified 2d objects, the pattern type is selected along with the bitmap shape using the PATTERN_SELECT method:

mthd 0x02bc: PATTERN_SELECT [*_2D]

Sets the pattern type and shape. One of: 0: BITMAP_8X8 1: BITMAP_64X1 2: BITMAP_1X64 3: COLOR

Operation::

if (param < 4) PATTERN_SELECT = SHADOW_2D.PATTERN_SELECT = param;

else throw(INVALID_ENUM);

Pattern coordinates

The pattern pixel is selected according to pattern coordinates: px, py. On NV1:G80, the pattern coordinates are equal to absolute [ie. not canvas-relative] coordinates in the destination surface. On G80+, an offset can be added to the coordinates. The offset is set by the PATTERN_OFFSET method:

mthd 0x02b0: PATTERN_OFFSET [*_2D] Sets the pattern offset. bits 0-5: X offset bits 8-13: Y offset

Operation: PATTERN_OFFSET = param;

The offset values are added to the destination surface X, Y coordinates to obtain px, py coordinates.

Bitmap pattern

The bitmap pattern is made of three parts:

- two-color palette
- 64 bits of pattern: each bit describes one pixel of the pattern and selects which color to use
- shape selector: determines whether the bitmap is 8x8, 64x1, or 1x64

The color to use for given pattern coordinates is selected as follows:

```
b6 bit;
if (shape == 8x8)
    bit = (py&7) << 3 | (px&7);
else if (shape == 64x1)
    bit = px & 0x3f;
else if (shape == 1x64)
    bit = py & 0x3f;
b1 pixel = PATTERN_BITMAP[bit[5]][bit[0:4]];
color = PATTERN_BITMAP_COLOR[pixel];
```

On NV1:NV4, the color is internally stored in A8R10G10B10 format and upconverted from the source format when submitted. On NV4:G80, it's stored in the original format it was submitted with, and is annotated with the format information as of the submission. On G80+, it's also stored as it was submitted, but is not annotated with format information - the format used to interpret it is the most recent pattern color format submitted.

On NV1:G80, the color and bitmap formats are stored in graph options for the PATTERN object. On G80+, they're part of main graph state instead.

The methods dealing with bitmap patterns are:

mthd 0x300: COLOR_FORMAT [NV1_PATTERN] [NV4-]

Sets the color format used for subsequent bitmap pattern colors. One of: 1: X16A8Y8 2: X16A1R5G5B5 3: A8R8G8B8

Operation::

switch (param) { case 1: cur_grobj.color_format = X16A8Y8; break; case 2: cur_grobj.color_format = X16A1R5G5B5; break; case 3: cur_grobj.color_format = A8R8G8B8; break; default: throw(INVALID_ENUM);

}

mthd 0x300: COLOR_FORMAT [NV4_PATTERN]

Sets the color format used for subsequent bitmap pattern colors. One of: 1: A16R5G6B5 2: X16A1R5G5B5 3: A8R8G8B8

Operation::

if (NV1:NV4) {

switch (param) { case 1: cur_grobj.color_format = A16R5G6B5; break; case 2: cur_grobj.color_format = X16A1R5G5B5; break; case 3: cur_grobj.color_format = A8R8G8B8; break; default: throw(INVALID_ENUM);

}

} else { SHADOW_COMP2D.PATTERN_COLOR_FORMAT = param; switch (param) {

case 1: PATTERN_COLOR_FORMAT = A16R5G6B5; break; case 2: PATTERN_COLOR_FORMAT = X16A1R5G5B5; break; case 3: PATTERN_COLOR_FORMAT = A8R8G8B8; break; default: throw(INVALID_ENUM);

}

}

mthd 0x2e8: PATTERN_COLOR_FORMAT [G80_2D]

Sets the color format used for bitmap pattern colors. One of: 0: A16R5G6B5 1: X16A1R5G5B5 2: A8R8G8B8 3: X16A8Y8 4: ??? [XXX] 5: ??? [XXX]

Operation::

if (param < 6) PATTERN_COLOR_FORMAT = SHADOW_2D.PATTERN_COLOR_FORMAT = param;

else throw(INVALID_ENUM);

mthd 0x304: BITMAP_FORMAT [*_PATTERN] [NV4-]

Sets the bitmap format used for subsequent pattern bitmaps. One of: 1: LE 2: CGA6

Operation::

if (NV4:G80) {

switch (param) { case 1: cur_grobj.bitmap_format = LE; break; case 2: cur_grobj.bitmap_format = CGA6; break; default: throw(INVALID_ENUM);

}

} else {

switch (param) { case 1: PATTERN_BITMAP_FORMAT = LE; break; case 2: PATTERN_BITMAP_FORMAT = CGA6; break; default: throw(INVALID_ENUM);

}

}

mthd 0x2ec: PATTERN_BITMAP_FORMAT [*_PATTERN]

Sets the bitmap format used for pattern bitmaps. One of: 0: LE 1: CGA6

Operation::

if (param < 2) PATTERN_BITMAP_FORMAT = param;

else throw(INVALID_ENUM);

mthd 0x310+i*4, i<2: BITMAP_COLOR [*_PATTERN] mthd 0x2f0+i*4, i<2: PATTERN_BITMAP_COLOR [*_2D]

Sets the colors used for bitmap pattern. i=0 sets the color used for pixels corresponding to '0' bits in the pattern, i=1 sets the color used for '1'.

Operation::

if (NV1:NV4) { PATTERN_BITMAP_COLOR[i].B = get_color_b10(cur_grobj, param); PATTERN_BITMAP_COLOR[i].G = get_color_b10(cur_grobj, param); PATTERN_BITMAP_COLOR[i].R = get_color_b10(cur_grobj, param); PATTERN_BITMAP_COLOR[i].A = get_color_b8(cur_grobj, param);

} else if (NV4:G80) { PATTERN_BITMAP_COLOR[i] = param; /* XXX: details */ CONTEXT_FORMAT.PATTERN_BITMAP_COLOR[i] = cur_grobj.color_format;

} else { PATTERN_BITMAP_COLOR[i] = param;

}

mthd 0x318+i*4, i<2: BITMAP [*_PATTERN] mthd 0x2f8+i*4, i<2: PATTERN_BITMAP [*_2D]

Sets the pattern bitmap. i=0 sets bits 0-31, i=1 sets bits 32-63.

Operation:: tmp = param; if (cur_grobj.BITMAP_FORMAT == CGA6 && NV1:G80) { /* XXX: check if also NV4+ */

/* pattern stored internally in LE format - for CGA6, reverse bits in all bytes */

tmp = (tmp & 0xaaaaaaaa) >> 1 | (tmp & 0x55555555) << 1; tmp = (tmp & 0xcccccccc) >> 2 | (tmp & 0x33333333) << 2; tmp = (tmp & 0xf0f0f0f0) >> 4 | (tmp & 0x0f0f0f0f) << 4;

} PATTERN_BITMAP[i] = tmp;

Color pattern

The color pattern is always an 8x8 array of R8G8B8 colors. It is stored and uploaded as an array of 64 cells in raster scan - the color for pattern coordinates (px, py) is taken from PATTERN_COLOR[(py&7) << 3 | (px&7)]. There are 4 sets of methods that set the pattern, corresponding to various color formats. Each set of methods updates the same state internally and converts the written values to R8G8B8 if necessary. Color pattern is available on NV4+ only.

mthd 0x400+i*4, i<16: COLOR_Y8 [NV4_PATTERN] mthd 0x500+i*4, i<16: PATTERN_COLOR_Y8 [*_2D]

Sets 4 color pattern cells, from Y8 source. bits 0-7: color for pattern cell i*4+0 bits 8-15: color for pattern cell i*4+1 bits 16-23: color for pattern cell i*4+2 bits 24-31: color for pattern cell i*4+3

Operation:: PATTERN_COLOR[4*i] = Y8_to_R8G8B8(param[0:7]); PATTERN_COLOR[4*i+1] = Y8_to_R8G8B8(param[8:15]); PATTERN_COLOR[4*i+2] = Y8_to_R8G8B8(param[16:23]); PATTERN_COLOR[4*i+3] = Y8_to_R8G8B8(param[24:31]);

mthd 0x500+i*4, i<32: COLOR_R5G6B5 [NV4_PATTERN] mthd 0x400+i*4, i<32: PATTERN_COLOR_R5G6B5 [*_2D]

Sets 2 color pattern cells, from R5G6B5 source. bits 0-15: color for pattern cell i*2+0 bits 16-31: color for pattern cell i*2+1

Operation:: PATTERN_COLOR[2*i] = R5G6B5_to_R8G8B8(param[0:15]); PATTERN_COLOR[2*i+1] = R5G6B5_to_R8G8B8(param[16:31]);

mthd 0x600+i*4, i<32: COLOR_X1R5G5B5 [NV4_PATTERN] mthd 0x480+i*4, i<32: PATTERN_COLOR_X1R5G5B5 [*_2D]

Sets 2 color pattern cells, from X1R5G5B5 source. bits 0-15: color for pattern cell i*2+0 bits 16-31: color for pattern cell i*2+1

Operation:: PATTERN_COLOR[2*i] = X1R5G5B5_to_R8G8B8(param[0:15]); PATTERN_COLOR[2*i+1] = X1R5G5B5_to_R8G8B8(param[16:31]);

mthd 0x700+i*4, i<64: COLOR_X8R8G8B8 [NV4_PATTERN] mthd 0x300+i*4, i<64: PATTERN_COLOR_X8R8G8B8 [*_2D]

Sets a color pattern cell, from X8R8G8B8 source.

Operation:: PATTERN_COLOR[i] = param[0:23];

Todo

precise upconversion formulas

Context objects

Contents

Introducton

Todo

write m

BETA

The BETA object family deals with setting the beta factor for the BLEND operation. The objects in this family are:

- objtype 0x01: NV1_BETA [NV1:NV4]
- class 0x0012: NV1_BETA [NV4:G84]

The methods are:

0100 NOP [NV4-] 0104 NOTIFY 0110 WAIT_FOR_IDLE [G80-] 0140 PM_TRIGGER [NV40-?] [XXX] 0180 N DMA_NOTIFY [NV4-] 0200 O PATCH_BETA_OUTPUT [NV4:NV20] 0300 BETA

mthd 0x300: BETA [NV1_BETA] Sets the beta factor. The parameter is a signed fixed-point number with a sign bit and 31 fractional bits. Note that negative values are clamped to 0, and only 8 fractional bits are actually implemented in hardware.

Operation:

```
if (param & 0x80000000) /* signed < 0 */
        BETA = 0;
else
        BETA = param & 0x7f800000;
```

mthd 0x200: PATCH_BETA_OUTPUT [NV1_BETA] [NV4:NV20] Reserved for plugging a beta patchcord to output beta factor into.

Operation:: throw(UNIMPLEMENTED_MTHD);

ROP

The ROP object family deals with setting the ROP [raster operation]. The ROP value thus set is only used in the ROP_* operation modes. The objects in this family are:

- objtype 0x02: NV1_ROP [NV1:NV4]
- class 0x0043: NV1_ROP [NV4:G84]

The methods are:

0100 NOP [NV4-] 0104 NOTIFY 0110 WAIT_FOR_IDLE [G80-] 0140 PM_TRIGGER [NV40-?] [XXX] 0180 N DMA_NOTIFY [NV4-] 0200 O PATCH_ROP_OUTPUT [NV4:NV20] 0300 ROP

mthd 0x300: ROP [NV1_ROP] Sets the raster operation.

Operation:

```
if (param & ~0xff)
        throw(INVALID_VALUE);
ROP = param;
```

mthd 0x200: PATCH_ROP_OUTPUT [NV1_ROP] [NV4:NV20] Reserved for plugging a ROP patchcord to output the ROP into.

Operation:

```
throw(UNIMPLEMENTED_MTHD);
```

CHROMA and PLANE

The CHROMA object family deals with setting the color for the color key. The color key is only used when enabled in options for a given graph object. The objects in this family are:

- objtype 0x03: NV1_CHROMA [NV1:NV4]
- class 0x0017: NV1_CHROMA [NV4:G80]
- class 0x0057: NV4_CHROMA [NV4:G84]

The PLANE object family deals with setting the color for plane masking. The plane mask operation is only done when enabled in options for a given graph object. The objects in this family are:

- objtype 0x04: NV1_PLANE [NV1:NV4]

For both objects, colors are internally stored in A1R10G10B10 format. [XXX: check NV4+]

The methods for these families are:

0100 NOP [NV4-] 0104 NOTIFY 0110 WAIT_FOR_IDLE [G80-] 0140 PM_TRIGGER [NV40-?] [XXX] 0180 N DMA_NOTIFY [NV4-] 0200 O PATCH_IMAGE_OUTPUT [NV4:NV20] 0300 COLOR_FORMAT [NV4-] 0304 COLOR

mthd 0x304: COLOR [*_CHROMA, NV1_PLANE] Sets the color.

Operation:

```
struct {
        int B : 10;
        int G : 10;
        int R : 10;
        int A : 1;
} tmp;
tmp.B = get_color_b10(cur_grobj, param);
tmp.G = get_color_g10(cur_grobj, param);
tmp.R = get_color_r10(cur_grobj, param);
tmp.A = get_color_a1(cur_grobj, param);
if (cur_grobj.type == NV1_PLANE)
        PLANE = tmp;
else
        CHROMA = tmp;
```

Todo

check NV3+

mthd 0x200: PATCH_IMAGE_OUTPUT [*_CHROMA, NV1_PLANE] [NV4:NV20] Reserved for plugging an image patchcord to output the color into.

Operation:

```
throw(UNIMPLEMENTED_MTHD);
```

CLIP

The CLIP object family deals with setting up the user clip rectangle. The user clip rectangle is only used when enabled in options for a given graph object. The objects in this family are:

- objtype 0x05: NV1_CLIP [NV1:NV4]
- class 0x0019: NV1_CLIP [NV4:G84]

The methods for this family are:

0100 NOP [NV4-] 0104 NOTIFY 0110 WAIT_FOR_IDLE [G80-] 0140 PM_TRIGGER [NV40-?] [XXX] 0180 N DMA_NOTIFY [NV4-] 0200 O PATCH_IMAGE_OUTPUT [NV4:NV20] 0300 CORNER 0304 SIZE

The clip rectangle state can be loaded in two ways:

- submit CORNER method twice, with upper-left and bottom-right corners
- submit CORNER method with upper-right corner, then SIZE method

To enable that, clip rectangle method operation is a bit unusual.

Todo

check if still applies on NV3+

Note that the clip rectangle state is internally stored relative to the absolute top-left corner of the framebuffer, while coordinates used in methods are relative to top-left corner of the canvas.

mthd 0x300: CORNER [NV1_CLIP] Sets a corner of the clipping rectangle. bits 0-15: X coordinate bits 16-31: Y coordinate

Operation:

```
ABS_UCLIP_XMIN = ABS_UCLIP_XMAX;
ABS_UCLIP_YMIN = ABS_UCLIP_YMAX;
ABS_UCLIP_XMAX = CANVAS_MIN.X + param.X;
ABS_UCLIP_YMAX = CANVAS_MIN.Y + param.Y;
```

Todo

check NV3+

---

mthd 0x304: SIZE [NV1_CLIP] Sets the size of the clipping rectangle. bits 0-15: width bits 16-31: height

Operation:

```
ABS_UCLIP_XMIN = ABS_UCLIP_XMAX;
ABS_UCLIP_YMIN = ABS_UCLIP_YMAX;
ABS_UCLIP_XMAX += param.X;
ABS_UCLIP_YMAX += param.Y;
```

Todo

check NV3+

---

mthd 0x200: PATCH_IMAGE_OUTPUT [NV1_CLIP] [NV4:NV20] Reserved for plugging an image patchcord to output the rectangle into.

Operation:

```
throw(UNIMPLEMENTED_MTHD);
```

BETA4

The BETA4 object family deals with setting the per-component beta factors for the BLEND_PREMULT and SRC-COPY_PREMULT operations. The objects in this family are:

- class 0x0072: NV4_BETA4 [NV4:G84]

The methods are:

0100 NOP [NV4-] 0104 NOTIFY 0110 WAIT_FOR_IDLE [G80-] 0140 PM_TRIGGER [NV40-?] [XXX] 0180 N DMA_NOTIFY [NV4-] 0200 O PATCH_BETA_OUTPUT [NV4:NV20] 0300 BETA4

mthd 0x300: BETA4 [NV4_BETA4] Sets the per-component beta factors. bits 0-7: B bits 8-15: G bits 16-23: R bits 24-31: A

Operation:

```
/* XXX: figure it out */
```

mthd 0x200: PATCH_BETA_OUTPUT [NV4_BETA4] [NV4:NV20] Reserved for plugging a beta patchcord to output beta factors into.

Operation:

```
throw(UNIMPLEMENTED_MTHD);
```

Surface setup

Todo

write me

SURF

Todo

write me

SURF2D

Todo

write me

SURF3D
Todo

write me

SWZSURF
Todo

write me

2D solid shape rendering

Contents

Introduction

One of 2d engine functions is drawing solid [single-color] primitives. The solid drawing functions use the usual 2D pipeline as described in graph/2d.txt and are available on all cards. The primitives supported are:

- points [NV1:NV4 and G80+]
- lines [NV1:NV4]
- lins [half-open lines]
- triangles
- upright rectangles [edges parallel to X/Y axes]

The 2d engine is limitted to integer vertex coordinates [ie. all primitive vertices must lie in pixel centres].

On NV1:G84 cards, the solid drawing functions are exposed via separate source object types for each type of primitive. On G80+, all solid drawing functionality is exposed via the unified 2d object.

Source objects

Each supported primitive type has its own source object class family on NV1:G80. These families are:

- POINT [NV1:NV4]
- LINE [NV1:NV4]
- LIN [NV1:G84]
- TRI [NV1:G84]
- RECT [NV1:NV40]

Common methods The common methods accepted by all solid source objects are:

0100 NOP [NV4-] [graph/intro.txt] 0104 NOTIFY [graph/intro.txt] 010c PATCH [NV4:?] [graph/2d.txt] 0110 WAIT_FOR_IDLE [G80-] [graph/intro.txt] 0140 PM_TRIGGER [NV40-?] [graph/intro.txt] 0180 N DMA_NOTIFY [NV4-] [graph/intro.txt] 0184 N NV1_CLIP [NV5-] [graph/2d.txt] 0188 N NV1_PATTERN [NV5-] [NV1_*] [graph/2d.txt] 0188 N NV4_PATTERN [NV5-] [NV4_* and up] [graph/2d.txt] 018c N NV1_ROP [NV5-] [graph/2d.txt] 0190 N NV1_BETA [NV5-] [graph/2d.txt] 0194 N NV3_SURFACE [NV5-] [NV1_*] [graph/2d.txt] 0194 N NV4_BETA4 [NV5-] [NV4_* and up] [graph/2d.txt] 0198 N NV4_SURFACE [NV5-] [NV4_* and up] [graph/2d.txt] 02fc N OPERATION [NV5-] [graph/2d.txt] 0300 COLOR_FORMAT [NV4-] [graph/solid.txt] 0304 COLOR [graph/solid.txt]

Todo

PM_TRIGGER?

Todo

PATCH?

Todo

add the patchcord methods

Todo

document common methods

POINT The POINT object family draws single points. The objects are:

- objtype 0x08: NV1_POINT [NV1:NV4]

The methods are:

0100:0400 [common solid rendering methods] 0400+i*4, i<32 POINT_XY 0480+i*8, i<16 POINT32_X 0484+i*8, i<16 POINT32_Y 0500+i*8, i<16 CPOINT_COLOR 0504+i*8, i<16 CPOINT_XY

Todo

document point methods

LINE/LIN The LINE/LIN object families draw lines/lins, respectively. The objects are:

- objtype 0x09: NV1_LINE [NV1:NV4]
- objtype 0x0a: NV1_LIN [NV1:NV4]
- class 0x001c: NV1_LIN [NV4:NV40]
- class 0x005c: NV4_LIN [NV4:G80]
- class 0x035c: NV30_LIN [NV30:NV40]
- class 0x305c: NV30_LIN [NV40:G84]

The methods are:

0100:0400 [common solid rendering methods] 0400+i*8, i<16 LINE_START_XY 0404+i*8, i<16 LINE_END_XY 0480+i*16, i<8 LINE32_START_X 0484+i*16, i<8 LINE32_START_Y 0488+i*16, i<8 LINE32_END_X 048c+i*16, i<8 LINE32_END_Y 0500+i*4, i<32 POLYLINE_XY 0580+i*8, i<16 POLYLINE32_X 0584+i*8, i<16 POLYLINE32_Y 0600+i*8, i<16 CPOLYLINE_COLOR 0604+i*8, i<16 CPOLYLINE_XY

Todo

document line methods

TRI The TRI object family draws triangles. The objects are:

- objtype 0x0b: NV1_TRI [NV1:NV4]
- class 0x001d: NV1_TRI [NV4:NV40]
- class 0x005d: NV4_TRI [NV4:G84]

The methods are:

0100:0400 [common solid rendering methods] 0310+j*4, j<3 TRIANGLE_XY 0320+j*8, j<3 TRIANGLE32_X 0324+j*8, j<3 TRIANGLE32_Y 0400+i*4, i<32 TRIMESH_XY 0480+i*8, i<16 TRIMESH32_X 0484+i*8, i<16 TRIMESH32_Y 0500+i*16 CTRIANGLE_COLOR 0504+i*16+j*4, j<3 CTRIANGLE_XY 0580+i*8, i<16 CTRIMESH_COLOR 0584+i*8, i<16 CTRIMESH_XY

Todo

document tri methods

RECT The RECT object family draws upright rectangles. Another object family that can also draw solid rectangles and should be used instead of RECT on cards that don't have RECT is GDI [graph/nv3-gdi.txt]. The objects are:

- objtype 0x0c: NV1_RECT [NV1:NV3]
- objtype 0x07: NV1_RECT [NV3:NV4]
- class 0x001e: NV1_RECT [NV4:NV40]
- class 0x005e: NV4_RECT [NV4:NV40]

The methods are:

0100:0400 [common solid rendering methods] 0400+i*8, i<16 RECT_POINT 0404+i*8, i<16 RECT_SIZE

Todo

document rect methods

Unified 2d object

Todo

document solid-related unified 2d object methods

Rasterization rules

This section describes exact rasterization rules for solids, ie. which pixels are considered to be part of a given solid. The common variables appearing in the pseudocodes are:

- CLIP_MIN_X - the left bounduary of the final clipping rectangle. If user clipping rectangle [see graph/2d.txt] is enabled, this is max(UCLIP_MIN_X, CANVAS_MIN_X). Otherwise, this is CANVAS_MIN_X.
- CLIP_MAX_X - the right bounduary of the final clipping rectangle. If user clipping rectangle is enabled, this is min(UCLIP_MAX_X, CANVAS_MAX_X). Otherwise, this is CANVAS_MAX_X.
- CLIP_MIN_Y - the top bounduary of the final clipping rectangle, defined like CLIP_MIN_X
- CLIP_MAX_Y - the bottom bounduary of the final clipping rectangle, defined like CLIP_MAX_X

A pixel is considered to be inside the clipping rectangle if:

- CLIP_MIN_X <= x < CLIP_MAX_X and
- CLIP_MIN_Y <= y < CLIP_MAX_Y

Points and rectangles A rectangle is defined through the coordinates of its left-top corner [X, Y] and its width and height [W, H] in pixels. A rectangle covers pixels that have x in [X, X+W) and y in [Y, Y+H) ranges.

```
void SOLID_RECT(int X, int Y, int W, int H) {
    int L = max(X, CLIP_MIN_X);
    int R = min(X+W, CLIP_MAX_X);
    int T = max(Y, CLIP_MIN_Y);
    int B = min(Y+H, CLIP_MAX_Y);
    int x, y;
    for (y = T; y < B; y++)
        for (x = L; x < R; x++)
            DRAW_PIXEL(x, y, SOLID_COLOR);
}
```

A point is defined through its X, Y coordinates and is rasterized as if it was a rectangle with W=H=1.

```
void SOLID_POINT(int X, int Y) {
    SOLID_RECT(X, Y, 1, 1);
}
```

Lines and lins Lines and lins are defined through the coordinates of two endpoints [X[2], Y[2]]. They are rasterized via a variant of Bresenham's line algorithm, with the following characteristics:

- rasterization proceeds in the direction of increasing x for y-major lines, and in the direction of increasing y for x-major lines [ie. in the direction of increasing *minor* component]

- when presented with a tie in a decision whether to increase the minor coordinate or not, increase it.
- if rasterizing a lin, the X[1], Y[1] pixel is not rasterized, but calculations are otherwise unaffected
- pixels outside the clipping rectangle are not rasterized, but calculations are otherwise unaffected

Equivalently, the rasterized lines/lins match those constructed via the diamond-exit rule with the following characteristics:

- a pixel is rasterized if the diamond inside it intersects the line/lin, unless it's a lin and the diamond also contains the second endpoint
- pixels outside the clipping rectangle are not rasterized, but calculations are otherwise unaffected
- pixel centres are considered to be on integer coordinates
- the following coordinates are considered to be contained in the diamond for pixel X, Y:
  - abs(x-X) + abs(x-Y) < 0.5 [ie. the inside of the diamond]
  - x = X-0.5, y = Y [ie. top vertex of the diamond]
  - x = X, y = Y-0.5 [ie. leftmost vertex of the diamond]

  [note that the edges don't matter, other than at the vertices - it's impossible to create a line touching them without intersecting them, due to integer endpoint coordinates]

```
void SOLID_LINE_LIN(int X[2], int Y[2], int is_lin) {
    /* determine minor/major direction */
    int xmajor = abs(X[0] - X[1]) > abs(Y[0] - Y[1]);
    int min0, min1, maj0, maj1;
    if (xmajor) {
        maj0 = X[0];
        maj1 = X[1];
        min0 = Y[0];
        min1 = Y[1];
    } else {
        maj0 = Y[0];
        maj1 = Y[1];
        min0 = X[0];
        min1 = X[1];
    }
    if (min1 < min0) {
        /* order by increasing minor */
        swap(min0, min1);
        swap(maj0, maj1);
    }
    /* deltas */
    int dmin = min1 - min0;
    int dmaj = abs(maj1 - maj0);
    /* major step direction */
    int step = maj1 > maj0 ? 1 : -1;
    int min, maj;
    /* scaled error - real error is err/(dmin * dmaj * 2) */
    int err = 0;
    for (min = min0, maj = maj0; maj != maj1 + step; maj += step) {
        if (err >= dmaj) { /* error >= 1/(dmin*2) */
            /* error too large, increase minor */
            min++;
            err -= dmaj * 2; /* error -= 1/dmin */
        }
        int x = xmajor?maj:min;
        int y = xmajor?min:maj;
```

```
        /* if not the final pixel of a lin and inside the clipping
           region, draw it */
        if ((!is_lin || x != X[1] || y != Y[1]) && in_clip(x, y))
            DRAW_PIXEL(x, y, SOLID_COLOR);
        error += dmin * 2; /* error += 1/dmaj */
    }
}
```

Triangles Triangles are defined through the coordinates of three vertices [X[3], Y[3]]. A triangle is rasterized as an intersection of three half-planes, corresponding to the three edges. For the purpose of triangle rasterization, half-planes are defined as follows:

- the edges are (0, 1), (1, 2) and (2, 0)
- if the two vertices making an edge overlap, the triangle is degenerate and is not rasterized
- a pixel is considered to be in a half-plane corresponding to a given edge if it's on the same side of that edge as the third vertex of the triangle [the one not included in the edge]
- if the third vertex lies on the edge, the triangle is degenerate and will not be rasterized
- if the pixel being considered for rasterization lies on the edge, it's considered included in the half-plane if the pixel immediately to its right is included in the half-plane
- if that pixel also lies on the edge [ie. edge is exactly horizontal], the original pixel is instead considered included if the pixel immediately below it is included in the half-plane

Equivalently, a triangle will include exactly-horizontal top edges and left edges, but not exactly-horizontal bottom edges nor right edges.

```
void SOLID_TRI(int X[3], int Y[3]) {
    int cross = (X[1] - X[0]) * (Y[2] - Y[0]) - (X[2] - X[0]) * (Y[1] - Y[0]);
    if (cross == 0) /* degenerate triangle */
        return;
    /* coordinates in CW order */
    if (cross < 0) {
        swap(X[1], X[2]);
        swap(Y[1], Y[2]);
    }
    int x, y, e;
    for (y = CLIP_MIN_Y; y < CLIP_MAX_Y; y++)
        for (x = CLIP_MIN_X; x < CLIP_MAX_X; x++) {
            for (e = 0; e < 3; e++) {
                int x0 = X[e];
                int y0 = Y[e];
                int x1 = X[(e+1)%3];
                int y1 = Y[(e+1)%3];
                /* first attempt */
                cross = (x1 - x0) * (y - y0) - (x - x0) * (y1 - y0);
                /* second attempt - pixel to the right */
                if (cross == 0)
                    cross = (x1 - x0) * (y - y0) - (x + 1 - x0) * (y1 - y0);
                /* third attempt - pixel below */
                if (cross == 0)
                    cross = (x1 - x0) * (y + 1 - y0) - (x - x0) * (y1 - y0);
                if (cross < 0)
                    goto out;
            }
            DRAW_PIXEL(x, y, SOLID_COLOR);
```

```
out:
        }
}
```

2D image from CPU upload

Contents

Introduction

Todo

write me

IFC

Todo

write me

BITMAP

Todo

write me

SIFC

Todo

write me

INDEX

Todo

write me

TEXTURE

Todo

write me

BLIT object

Contents

Introduction

Todo

write me

Methods

Todo

write me

Operation

Todo

write me

Image to/from memory objects

Contents

Introduction

Todo

write me

Methods

Todo

write me

IFM operation

Todo

write me

ITM operation

Todo

write me

NV1 textured quad objects

Contents

Introduction

Todo

write me

The methods

Todo

write me

Linear interpolation process

Todo

write me

Quadratic interpolation process

Todo

write me

GDI objects

Contents

Introduction

Todo

write me

Methods

Todo

write me

Clipped rectangles

Todo

write me

Unclipped rectangles

Todo

write me

Unclipped transparent bitmaps

Todo

write me

Clipped transparent bitmaps

Todo

write me

Clipped two-color bitmaps

Todo

write me

Scaled image from memory object

Contents

Introduction

Todo

write me

Methods

Todo

write me

Operation

Todo

write me

YCbCr blending objects

Contents

Introduction

Todo

write me

Methods

Todo

write me

Operation

Todo

write me

2.9.4 NV1 graphics engine

Contents:

2.9.5 NV3 graphics engine

Contents:

NV3 3D objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.6 NV4 graphics engine

Contents:

NV4 3D objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.7 NV10 Celsius graphics engine

Contents:

NV10 Celsius 3D objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.8 NV20 Kelvin graphics engine

Contents:

NV20 Kelvin 3D objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.9 NV30 Rankine graphics engine

Contents:

NV30 Rankine 3D objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.10 NV40 Curie graphics engine

Contents:

NV40 Curie 3D objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.11 G80 Tesla graphics and compute engine

Contents:

G80 PGRAPH context switching

Contents

Introduction

Todo

write me

G80 Tesla 3D objects

Contents

Todo

write me

Introduction

Todo

write me

G80 Tesla compute objects

Contents

Todo

write me

Introduction

Todo

write me

Tesla CUDA processors

Contents:

Tesla CUDA ISA

Contents

Introduction This file deals with description of Tesla CUDA instruction set. CUDA stands for Completely Unified Device Architecture and refers to the fact that all types of shaders (vertex, geometry, fragment, and compute) use nearly the same ISA and execute on the same processors (called streaming multiprocessors).

The Tesla CUDA ISA is used on Tesla generation GPUs (G8x, G9x, G200, GT21x, MCP77, MCP79, MCP89). Older GPUs have separate ISAs for vertex and fragment programs. Newer GPUs use Fermi, Kepler2, or Maxwell ISAs.

Variants There are seversal variants of Tesla ISA (and the corresponding multiprocessors). The features added to the ISA after the first iteration are:

- breakpoints [G84:]
- new barriers [G84:]
- atomic operations on g[] space [G84:]
- load from s[] instruction [G84:]
- lockable s[] memory [G200:]
- double-precision floating point [G200 *only*]
- 64-bit atomic add on g[] space [G200:]
- vote instructions [G200:]
- D3D10.1 additions [GT215:]: - $sampleid register (for sample shading) - texprep cube instruction (for cubemap array access) - texquerylod instruction - texgather instruction

- preret and indirect bra instructions [GT215:]?

Todo

check variants for preret/indirect bra

Warps and thread types Programs on Tesla MPs are executed in units called "warps". A warp is a group of 32 individual threads executed together. All threads in a warp share common instruction pointer, and always execute the same instruction, but have otherwise independent state (ie. separate register sets). This doesn't preclude independent branching: when threads in a warp disagree on a branch condition, one direction is taken and the other is pushed onto a stack for further processing. Each of the divergent execution paths is tagged with a "thread mask": a bitmask of threads in the warp that satisfied (or not) the branch condition, and hence should be executed. The MP does no work (and modifies no state) for threads not covered by the current thread mask. Once the first path reaches completion, the stack is popped, restoring target PC and thread mask for the second path, and execution continues.

Depending on warp type, the threads in a warp may be related to each other or not. There are 4 warp types, corresponding to 4 program types:

- vertex programs: executed once for each vertex submitted to the 3d pipeline. They're grouped into warps in a rather uninteresting way. Each thread has read-only access to its vertex' input attributes and write-only access to its vertex' output attributes.
- geometry programs: if enabled, executed once for each geometry primitive submitted to the 3d pipeline. Also grouped into warps in an uninteresting way. Each thread has read-only access to input attributes of its primitive's vertices and per-primitive attributes. Each thread also has write-only access to output vertex attributes and instructions to emit a vertex and break the output primitive.
- fragment programs: executed once for each fragment rendered by the 3d pipeline. Always dispatched in groups of 4, called quads, corresponding to aligned 2x2 squares on the screen (if some of the fragments in the square are not being rendered, the fragment program is run on them anyway, and its result discarded). This grouping is done so that approximate screen-space derivatives of all intermediate results can be computed by exchanging data with other threads in the quad. The quads are then grouped into warps in an uninteresting way. Each thread has read-only access to interpolated attribute data and is expected to return the pixel data to be written to the render output surface.
- compute programs: dispatched in units called blocks. Blocks are submitted manually by the user, alone or in so-called grids (basically big 2d arrays of blocks with identical parameters). The user also determines how many threads are in a block. The threads of a block are sequentially grouped into warps. All warps of a block execute in parallel on a single MP, and have access to so-called shared memory. Shared memory is a fast per-block area of memory, and its size is selected by the user as part of block configuration. Compute warps also have random R/W access to so-called global memory areas, which can be arbitrarily mapped to card VM by the user.

Registers The registers in Tesla ISA are:

- up to 128 32-bit GPRs per thread: $r0-$r127. These registers are used for all calculations (with the exception of some address calculations), whether integer or floating-point.

  The amount of available GPRs per thread is chosen by the user as part of MP configuration, and can be selected per program type. For example, if the user enables 16 registers, $r0-$r15 will be usable and $r16-$r127 will be forced to 0. Since the MP has a rather limited amount of storage for GPRs, this configuration parameter determines how many active warps will fit simultanously on an MP.

  If a 16-bit operation is to be performed, each GPR from $r0-$r63 range can be treated as a pair of 16-bit registers: $rXl (low half of $rX) and $rXh (high part of $rX).

If a 64-bit operation is to be performed, any naturally aligned pair of GPRs can be treated as a 64-bit register: $rXd (which has the low half in $rX and the high half in $r(X+1), and X has to even). Likewise, if a 128-bit operation is to be performed, any naturally aligned group of 4 registers can be treated as a 128-bit registers: $rXq. The 32-bit chunks are assigned to $rX..(X+3) in order from lowest to highest.

- 4 16-bit address registers per thread: $a1-$a4, and one additional register per warp ($a7). These registers are used for addressing all memory spaces except global memory (which uses 32-bit addressing via $r register file). In addition to the 4 per-thread registers and 1 per-warp register, there's also $a0, which is always equal to 0.

Todo

wtf is up with $a7?

- 4 4-bit condition code registers per thread: $c0-$c3. These registers can be optionally set as a result of some (mostly arithmetic) instructions and are made of 4 individual bits:
  - bit 0: Z - zero flag. For integer operations, set when the result is equal to 0. For floating-point operations, set when the result is 0 or NaN.
  - bit 1: S - sign flag. For integer operations, set when the high bit of the result is equal to 1. For floating-point operations, set when the result is negative or NaN.
  - bit 2: C - carry flag. For integer addition, set when there is a carry out of the highest bit of the result.
  - bit 3: O - overflow flag. For integer addition, set when the true (infinite-precision) result doesn't fit in the destination (considered to be a signed number).
- A few read-only 32-bit special registers, $sr0-$sr8:
  - $sr0 aka $physid: when read, returns the physical location of the current thread on the GPU:
    - bits 0-7: thread index (inside a warp)
    - bits 8-15: warp index (on an MP)
    - bits 16-19: MP index (on a TPC)
    - bits 20-23: TPC index
  - $sr1 aka $clock: when read, returns the MP clock tick counter.

    Todo

    a bit more detail?

  - $sr2: always 0?

    Todo

    perhaps we missed something?

  - $sr3 aka $vstride: attribute stride, determines the spacing between subsequent attributes of a single vertex in the input space. Useful only in geometry programs.

    Todo

    seems to always be 0x20. Is it really that boring, or does MP switch to a smaller/bigger stride sometimes?

  - $sr4-$sr7 aka $pm0-$pm3: MP performance counters.
  - $sr8 aka $sampleid [GT215:]: the sample ID. Useful only in fragment programs when sample shading is enabled.

Memory The memory spaces in Tesla ISA are:

- C[]: code space. 24-bit, byte-oriented addressing. The only way to access this space is by executing code from it (there's no "read from code space" instruction). There is one code space for each program type, and it's mapped to a 16MB range of VM space by the user. It has three levels of cache (global, TPC, MP) that need to be manually flushed when its contents are modified by the user.
- c0[]-c15[]: const spaces. 16-bit byte-oriented addressing. Read-only and accessible from any program type in 8, 16, and 32-bit units. Like C[], it has three levels of cache. Each of the 16 const spaces of each program type can be independently bound to one of 128 global (per channel) const buffers. In turn, each of the const buffers can be independently bound to a range of VM space (with length divisible by 256) or disabled by the user.
- l[]: local space. 16-bit, byte-oriented addressing. Read-write and per-thread, accessible from any program type in 8, 16, 32, 64, and 128-bit units. It's directly mapped to VM space (although with heavy address mangling), and hence slow. Its per-thread length can be set to any power of two size between 0x10 and 0x10000 bytes, or to 0.
- a[]: attribute space. 16-bit byte-oriented addressing. Read-only, per-thread, accessible in 32-bit units only and only available in vertex and geometry programs. In vertex programs, contains input vertex attributes. In geometry programs, contains pointers to vertices in p[] space and per-primitive attributes.
- p[]: primitive space. 16-bit byte oriented addressing. Read-only, per-MP, available only from geometry programs, accessed in 32-bit units. Contains input vertex attributes.
- o[]: output space. 16-bit byte-oriented addressing. Write-only, per-thread. Available only from vertex and geometry programs, accessed in 32-bit units. Contains output vertex attributes.
- v[]: varying space. 16-bit byte-oriented addressing. Read-only, available only from fragment programs, accessed in 32-bit units. Contains interpolated input vertex attributs. It's a "virtual" construct: there are really three words stored in MP for each v[] word (base, dx, dy) and reading from v[] space will calculate the value for the current fragment by evaluating the corresponding linear function.
- s[]: shared space. 16-bit byte-oriented addressing. Read-write, per-block, available only from compute programs, accessible in 8, 16, and 32-bit units. Length per block can be selected by user in 0x40-byte increments from 0 to 0x4000 bytes. On G200+, has a locked access feature: every warp can have one locked location in s[], and all other warps will block when trying to access this location. Load with lock and store with unlock instructions can thus be used to implement atomic operations.
- g0[]-g15[]: global spaces. 32-bit byte-oriented addressing. Read-write, available only from compute programs, accessible in 8, 16, 32, 64, and 128-bit units. Each global space can be configured in either linear or 2d mode. When in linear mode, a global space is simply mapped to a range of VM memory. When in 2d mode, low 16 bits of gX[] address are the x coordinate, and high 16 bits are the y coordinate. The global space is then mapped to a blocklinear 2d surface in VM space. On G84+, some atomic operations on global spaces are supported.

Todo

when no-one's looking, rename the a[], p[], v[] spaces to something sane.

Other execution state and resources There's also a fair bit of implicit state stored per-warp for control flow:

- 22-bit PC (24-bit address with low 2 bits forced to 0): the current address in C[] space where instructions are executed.

- 32-bit active thread mask: selects which threads are executed and which are not. If a bit is 1 here, instructions will be executed for the given thread.
- 32-bit invisible thread mask: useful only in fragment programs. If a bit is 1 here, the given thread is unused, or corresponds to a pixel on the screen which won't be rendered (ie. was just launched to fill a quad). Texture instructions with "live" flag set won't be run for such threads.
- 32*2-bit thread state: stores state of each thread:
  - 0: active or branched off
  - 1: executed the brk instruction
  - 2: executed the ret instruction
  - 3: executed the exit instruction
- Control flow stack. The stack is made of 64-bit entries, with the following fields:
  - PC
  - thread mask
  - entry type:
    - 1: branch
    - 2: call
    - 3: call with limit
    - 4: prebreak
    - 5: quadon
    - 6: joinat

Todo

discard mask should be somewhere too?

Todo

call limit counter

Other resources available to CUDA code are:

- $t0-$t129: up to 130 textures per 3d program type, up to 128 for compute programs.
- $s0-$s17: up to 18 texture samplers per 3d program type, up to 16 for compute programs. Only used if linked texture samplers are disabled.
- Up to 16 barriers. Per-block and available in compute programs only. A barrier is basically a warp counter: a barrier can be increased or waited for. When a warp increases a barrier, its value is increased by 1. If a barrier would be increased to a value equal to a given warp count, it's set to 0 instead. When a barrier is waited for by a warp, the warp is blocked until the barrier's value is equal to 0.

Todo

there's some weirdness in barriers.

Instruction format Instructions are stored in C[] space as 32-bit little-endian words. There are short (1 word) and long (2 words) instructions. The instruction type can be distinguished as follows:

word 0	word 1	instruction type
bits 0-1	bits 0-1	
0	-	short normal
1	0	long normal
1	1	long normal with `join`
1	2	long normal with `exit`
1	3	long immediate
2	-	short control
3	any	long control

Todo

you sure of control instructions with non-0 w1b0-1?

Long instructions can only be stored on addresses divisible by 8 bytes (ie. on even word address). In other words, short instructions usually have to be issued in pairs (the only exception is when a block starts with a short instruction on an odd word address). This is not a problem, as all short instructions have a long equivalent. Attempting to execute a non-aligned long instruction results in UNALIGNED_LONG_INSTRUCTION decode error.

Long normal instructions can have a `join` or `exit` instruction tacked on. In this case, the extra instruction is executed together with the main instruction.

The instruction group is determined by the opcode fields:

- word 0 bits 28-31: primary opcode field
- word 1 bits 29-31: secondary opcode field (long instructions only)

Note that only long immediate and long control instructions always have the secondary opcode equal to 0.

The exact instruction of an instruction group is determined by group-specific encoding. Attempting to execute an instruction whose primary/secondary opcode doesn't map to a valid instruction group results in ILLEGAL_OPCODE decode error.

Other fields Other fields used in instructions are quite instruction-specific. However, some common bitfields exist. For short normal instructions, these are:

- bits 0-1: 0 (select short normal instruction)
- bits 2-7: destination
- bit 8: modifier 1
- bits 9-14: source 1
- bit 15: modifier 2
- bits 16-21: source 2
- bit 22: modifier 3
- bit 23: source 2 type
- bit 24: source 1 type
- bit 25: $a postincrement flag
- bits 26-27: address register
- bits 28-31: primary opcode

For long immediate instructions:

- word 0:
  - bits 0-1: 1 (select long non-control instruction)
  - bits 2-7: destination
  - bit 8: modifier 1
  - bits 9-14: source 1
  - bit 15: modifier 2
  - bits 16-21: immediate low 6 bits
  - bit 22: modifier 3
  - bit 23: unused
  - bit 24: source 1 type
  - bit 25: $a postincrement flag
  - bits 26-27: address register
  - bits 28-31: primary opcode
- word 1:
  - bits 0-1: 3 (select long immediate instruction)
  - bits 2-27: immediate high 26 bits
  - bit 28: unused
  - bits 29-31: always 0

For long normal instructions:

- word 0:
  - bits 0-1: 1 (select long non-control instruction)
  - bits 2-8: destination
  - bits 9-15: source 1
  - bits 16-22: source 2
  - bit 23: source 2 type
  - bit 24: source 3 type
  - bit 25: $a postincrement flag
  - bits 26-27: address register low 2 bits
  - bits 28-31: primary opcode
- word 1:
  - bits 0-1: 0 (no extra instruction), 1 (`join`), or 2 (`exit`)
  - bit 2: address register high bit
  - bit 3: destination type
  - bits 4-5: destination $c register
  - bit 6: $c write enable

  - bits 7-11: predicate
  - bits 12-13: source $c register
  - bits 14-20: source 3
  - bit 21: source 1 type
  - bits 22-25: c[] space index
  - bit 26: modifier 1
  - bit 27: modifier 2
  - bit 28: unused
  - bits 29-31: secondary opcode

Note that short and long immediate instructions have 6-bit source/destination fields, while long normal instructions have 7-bit ones. This means only half the registers can be accessed in such instructions ($r0-$r63, $r0l-$r31h).

For long control instructions:

- word 0:
  - bits 0-1: 3 (select long control instruction)
  - bits 9-24: code address low 18 bits
  - bits 28-31: primary opcode
- word 1:
  - bit 6: modifier 1
  - bits 7-11: predicate
  - bits 12-13: source $c register
  - bits 14-19: code address high 6 bits

Todo

what about other bits? ignored or must be 0?

Note that many other bitfields can be in use, depending on instruction. These are just the most common ones.

Whenever a half-register ($rXl or $rXh) is stored in a field, bit 0 of that field selects high or low part (0 is low, 1 is high), and bits 1 and up select $r index. Whenever a double register ($rXd) is stored in a field, the index of the low word register is stored. If the value stored is not divisible by 2, the instruction is illegal. Likewise, for quad registers ($rXq), the lowest word register is stored, and the index has to be divisible by 4.

Predicates Most long normal and long control instructions can be predicated. A predicated instruction is only executed if a condition, computed based on a selected $c register, evaluates to 1. The instruction fields involved in predicates are:

- word 1 bits 7-11: predicate field - selects a boolean function of the $c register
- word 1 bits 12-13: $c source field - selects the $c register to use

The predicates are:

encoding	name	description	condition formula
0x00	never	always false	0
0x01	l	less than	(S & ~Z) ^ O
0x02	e	equal	Z & ~S
0x03	le	less than or equal	S ^ (Z | O)
0x04	g	greater than	~Z & ~(S ^ O)
0x05	lg	less or greater than	~Z
0x06	ge	greater than or equal	~(S ^ O)
0x07	lge	ordered	~Z | ~S
0x08	u	unordered	Z & S
0x09	lu	less than or unordered	S ^ O
0x0a	eu	equal or unordered	Z
0x0b	leu	not greater than	Z | (S ^ O)
0x0c	gu	greater than or unordered	~S ^ (Z | O)
0x0d	lgu	not equal to	~Z | S
0x0e	geu	not less than	(~S | Z) ^ O
0x0f	always	always true	1
0x10	o	overflow	O
0x11	c	carry / unsigned not below	C
0x12	a	unsigned above	~Z & C
0x13	s	sign / negative	S
0x1c	ns	not sign / positive	~S
0x1d	na	unsigned not above	Z | ~C
0x1e	nc	not carry / unsigned below	~C
0x1f	no	no overflow	~O

Some instructions read $c registers directly. The operand CSRC refers to the $c register selected by the $c source field. Note that, on such instructions, the $c register used for predicating is necessarily the same as the input register. Thus, one must generally avoid predicating instructions with $c input.

$c destination field Most normal long instructions can optionally write status information about their result to a $c register. The $c destination is selected by $c destination field, located in word 1 bits 4-5, and $c destination enable field, located in word 1 bit 6. The operands using these fields are:

- FCDST (forced condition destination): $c0-$c3, as selected by $c destination field.
- CDST (condition destination):
  - if $c destination enable field is 0, no destination is used (condition output is discarded).
  - if $c destination enable field is 1, same as FCDST.

Memory addressing Some instructions can access one of the memory spaces available to CUDA code. There are two kinds of such instructions:

- Ordinary instructions that happen to be used with memory operands. They have very limitted direct addressing range (since they fit the address in 6 or 7 bits normally used for register selection) and may lack indirect addressing capabilities.
- Dedicated load/store instructions. They have full 16-bit direct addressing range and have indirect addressing capabilities.

The following instruction fields are involved in memory addressing:

- word 0 bit 25: autoincrement flag
- word 0 bits 26-27: $a low field

- word 1 bit 2: $a high field
- word 0 bits 9-16: long offset field (used for dedicated load/store instructions)

There are two operands used in memory addressing:

- `SASRC` (short address source): $a0-$a3, as selected by $a low field.
- `LASRC` (long address source): $a0-$a7, as selected by concatenation of $a low and high fields.

Every memory operand has an associated offset field and multiplication factor (a constant, usually equal to the access size). Memory operands also come in two kinds: direct (no $a field) and indirect ($a field used).

For direct operands, the memory address used is simply the value of the offset field times the multiplication factor.

For indirect operands, the memory address used depends on the value of the autoincrement flag:

- if flag is 0, memory address used is `$aX + offset * factor`, where $a register is selected by `SASRC` (for short and long immediate instructions) or `LASRC` (for long normal instructions) operand. Note that using `$a0` with this addressing mode can emulate a direct operand.
- if flag is 1, memory address used is simply `$aX`, but after the memory access is done, the `$aX` will be increased by `offset * factor`. Attempting to use `$a0` (or `$a5`/`a6`) with this addressing mode results in ILLEGAL_POSTINCR decode error.

Todo

figure out where and how $a7 can be used. Seems to be a decode error more often than not...

Todo

what address field is used in long control instructions?

Shared memory access Most instructions can use an s[] memory access as the first source operand. When s[] access is used, it can be used in one of 4 modes:

- 0: `u8` - read a byte with zero extension, multiplication factor is 1
- 1: `u16` - read a half-word with zero extension, factor is 2
- 2: `s16` - read a half-word with sign extension, factor is 2
- 3: `b32` - read a word, factor is 4

The corresponding source 1 field is split into two subfields. The high 2 bits select s[] access mode, while the low 4 or 5 bits select the offset. Shared memory operands are always indirect operands. The operands are:

- `SSSRC1` (short shared word source 1): use short source 1 field, all modes valid.
- `LSSRC1` (long shared word source 1): use long source 1 field, all modes valid.
- `SSHSRC1` (short shared halfword source 1): use short source 1 field, valid modes `u8`, `u16`, `s16`.
- `LSHSRC1` (long shared halfword source 1): use long source 1 field, valid modes `u8`, `u16`, `s16`.
- `SSUHSRC1` (short shared unsigned halfword source 1): use short source 1 field, valid modes `u8`, `u16`.
- `LSUHSRC1` (long shared unsigned halfword source 1): use long source 1 field, valid modes `u8`, `u16`.
- `SSSHSRC1` (short shared signed halfword source 1): use short source 1 field, valid modes `u8`, `s16`.
- `LSSHSRC1` (long shared signed halfword source 1): use long source 1 field, valid modes `u8`, `s16`.

- LSBSRC1 (long shared byte source 1): use long source 1 field, only u8 mode valid.

Attempting to use b32 mode when it's not valid (because source 1 has 16-bit width) results in ILLEGAL_MEMORY_SIZE decode error. Attempting to use u16/s16 mode that is invalid because the sign is wrong results in ILLEGAL_MEMORY_SIGN decode error. Attempting to use mode other than u8 for cvt instruction with u8 source results in ILLEGAL_MEMORY_BYTE decode error.

Destination fields Most short and long immediate instructions use the short destination field for selecting instruction destination. The field is located in word 0 bits 2-7. There are two common operands using that field:

- SDST (short word destination): GPR $r0-$r63, as selected by the short destination field.
- SHDST (short halfword destination): GPR half $r0l-$r31h, as selected by the short destination field.

Most normal long instructions use the long destination field for selecting instruction destination. The field is located in word 0 bits 2-8. This field is usually used together with destination type field, located in word 1 bit 3. The common operands using these fields are:

- LRDST (long register word destination): GPR $r0-$r127, as selected by the long destination field.
- LRHDST (long register halfword destination): GPR half $r0l-$r63h, as selected by the long destination field.
- LDST (long word destination):
  - if destination type field is 0, same as LRDST.
  - if destination type field is 1, and long destination field is equal to 127, no destination is used (ie. operation result is discarded). This is used on instructions that are executed only for their $c output.
  - if destination type field is 1, and long destination field is not equal to 127, o[] space is written, as a direct memory operand with long destination field as the offset field and multiplier factor 4.

Todo

verify the 127 special treatment part and direct addressing

- LHDST (long halfword destination):
  - if destination type field is 0, same as LRHDST.
  - if destination type field is 1, and long destination field is equal to 127, no destination is used (ie. operation result is discarded).
  - if destination type field is 1, and long destination field is not equal to 127, o[] space is written, as a direct memory operand with long destination field as the offset field and multiplier factor 2. Since o[] can only be written with 32-bit accesses, the address is rounded down to a multiple of 4, and the 16-bit result is duplicated in both low and high half of the 32-bit value written in o[] space. This makes it pretty much useless.
- LDDST (long double destination): GPR pair $r0d-$r126d, as selected by the long destination field.
- LQDST (long quad destination): GPR quad $r0q-$r124q, as selected by the long destination field.

Short source fields

Todo

write me

Long source fields

Todo

write me

Opcode map

Table 2.8: Opcode map

Primary opcode	short normal	long immediate	long normal, secondary 0	long normal, secondary 1	long normal, secondary 2	long normal, secondary 3	long normal, secondary 4	long normal, secondary 5	long normal, secondary 6	long normal, secondary 7	short control	long control
0x0	-	-	ld a[]	mov from $c	mov from $a	mov from $sr	st o[]	mov to $c	shl to $a	st s[]	-	discard
0x1	mov	mov	mov	ld c[]	ld s[]	vote	-	-	-	-	-	bra
0x2	add/sub	add/sub	add/sub	-	-	-	-	-	-	-	-	call
0x3	add/sub	add/sub	add/sub	-	-	set	max	min	shl	shr	-	ret
0x4	mul	mul	mul	-	-	-	-	-	-	-	-	pre-brk
0x5	sad	-	sad	-	-	-	-	-	-	-	-	brk
0x6	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	-	quadon
0x7	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	mul+add	-	quad-pop
0x8	interp	-	interp	-	-	-	-	-	-	-	-	bar
0x9	rcp	-	rcp	-	rsqrt	lg2	sin	cos	ex2	-	trap	trap
0xa	-	-	cvt i2i	cvt i2i	cvt i2f	cvt i2f	cvt f2i	cvt f2i	cvt f2f	cvt f2f	-	joinat
0xb	fadd	fadd	fadd	fadd	-	fset	fmax	fmin	presin/preex2		brkpt	brkpt
0xc	fmul	fmul	fmul	-	fslct	fslct	quadop	-	-	-	-	bra c[]
0xd	-	logic op	logic op	add $a	ld l[]	st l[]	ld g[]	st g[]	red g[]	atomic g[]	-	pre-ret
0xe	fmul+fadd	fmul+fadd	fmul+fadd	fmul+fadd	dfma	dadd	dmul	dmin	dmax	dset	-	-
0xf	tex-auto/fetch	-	tex-auto/fetch	texbias	texlod	tex misc	texc-saa/gather	???	emit/restart	nop/pmevent	-	-

Instructions The instructions are roughly divided into the following groups:

- *Data movement instructions*
- *Integer arithmetic instructions*
- *Floating point instructions*
- *Transcendential instructions*
- *Double precision floating point instructions*
- *Control instructions*
- *Texture instructions*
- *Misc instructions*

Data movement instructions

Contents

Introduction

Todo

write me

Data movement: (h)mov

Todo

write me

```
[lanemask] mov b32/b16 DST SRC

lanemask assumed 0xf for short and immediate versions.

  if (lanemask & 1 << (laneid & 3)) DST = SRC;

Short:    0x10000000 base opcode
      0x00008000 0: b16, 1: b32
      operands: S*DST, S*SRC1/S*SHARED

Imm:      0x10000000 base opcode
      0x00008000 0: b16, 1: b32
      operands: L*DST, IMM

Long:     0x10000000 0x00000000 base opcode
```

```
      0x00000000 0x04000000 0: b16, 1: b32
      0x00000000 0x0003c000 lanemask
      operands: LL*DST, L*SRC1/L*SHARED
```

Condition registers

Reading condition registers: mov (from $c)

Todo

write me

```
mov DST COND

DST is 32-bit $r.

  DST = COND;

Long:     0x00000000 0x20000000 base opcode
      operands: LDST, COND
```

Writing condition registers: mov (to $c)

Todo

write me

```
mov CDST SRC

SRC is 32-bit $r. Yes, the 0x40 $c write enable flag in second word is
actually ignored.

  CDST = SRC;

Long:     0x00000000 0xa0000000 base opcode
      operands: CDST, LSRC1
```

Address registers

Reading address registers: mov (from $a)

Todo

write me

```
mov DST AREG

DST is 32-bit $r. Setting flag normally used for autoincrement mode doesn't
work, but still causes crash when using non-writable $a's.

  DST = AREG;

Long:     0x00000000 0x40000000 base opcode
```

```
      0x02000000 0x00000000 crashy flag
      operands: LDST, AREG
```

Writing address registers: shl (to $a)

Todo

write me

```
shl ADST SRC SHCNT

SRC is 32-bit $r.

  ADST = SRC << SHCNT;

Long:     0x00000000 0xc0000000 base opcode
      operands: ADST, LSRC1/LSHARED, HSHCNT
```

Increasing address registers: add ($a)

Todo

write me

```
add ADST AREG OFFS

Like mov from $a, setting flag normally used for autoincrement mode doesn't
work, but still causes crash when using non-writable $a's.

  ADST = AREG + OFFS;

Long:     0xd0000000 0x20000000 base opcode
      0x02000000 0x00000000 crashy flag
      operands: ADST, AREG, OFFS
```

Reading special registers: mov (from $sr)

Todo

write me

```
mov DST physid    S=0
mov DST clock     S=1
mov DST sreg2     S=2
mov DST sreg3     S=3
mov DST pm0       S=4
mov DST pm1       S=5
mov DST pm2       S=6
mov DST pm3       S=7

DST is 32-bit $r.

  DST = SREG;

Long:     0x00000000 0x60000000 base opcode
```

```
      0x00000000 0x0001c000 S
      operands: LDST
```

Memory space access

Const space access: ld c[]

Todo

write me

Local space access: ld l[], st l[]

Todo

write me

Shared space access: ld s[], st s[]

Todo

write me

```
mov lock CDST DST s[]

  Tries to lock a word of s[] memory and load a word from it. CDST tells
  you if it was successfully locked+loaded, or no. A successfully locked
  word can't be locked by any other thread until it is unlocked.

mov unlock s[] SRC

  Stores a word to previously-locked s[] word and unlocks it.
```

Input space access: ld a[]

Todo

write me

Output space access: st o[]

Todo

write me

Global space access

Global load/stores: ld g[], st g[]

Todo

write me

Global atomic operations: ld (add|inc|dec|max|min|and|or|xor) g[], xchg g[], cas g[]

Todo

write me

Global reduction operations: (add|inc|dec|max|min|and|or|xor) g[]

Todo

write me

Integer arithmetic instructions

Contents

Introduction

Todo

write me

```
S(x): 31th bit of x for 32-bit x, 15th for 16-bit x.
SEX(x): sign-extension of x
ZEX(x): zero-extension of x
```

Addition/substraction: (h)add, (h)sub, (h)subr, (h)addc

Todo

write me

```
add [sat] b32/b16 [CDST] DST SRC1 SRC2        O2=0, O1=0
sub [sat] b32/b16 [CDST] DST SRC1 SRC2        O2=0, O1=1
subr [sat] b32/b16 [CDST] DST SRC1 SRC2       O2=1, O1=0
addc [sat] b32/b16 [CDST] DST SRC1 SRC2 COND      O2=1, O1=1

All operands are 32-bit or 16-bit according to size specifier.

  b16/b32 s1, s2;
  bool c;
```

```
  switch (OP) {
      case add: s1 = SRC1, s2 = SRC2, c = 0; break;
      case sub: s1 = SRC1, s2 = ~SRC2, c = 1; break;
      case subr: s1 = ~SRC1, s2 = SRC2, c = 1; break;
      case addc: s1 = SRC1, s2 = SRC2, c = COND.C; break;
  }
  res = s1+s2+c;  // infinite precision
  CDST.C = res >> (b32 ? 32 : 16);
  res = res & (b32 ? 0xffffffff : 0xffff);
  CDST.O = (S(s1) == S(s2)) && (S(s1) != S(res));
  if (sat && CDST.O)
      if (S(res)) res = (b32 ? 0x7fffffff : 0x7fff);
      else res = (b32 ? 0x80000000 : 0x8000);
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Short/imm:    0x20000000 base opcode
      0x10000000 O2 bit
      0x00400000 O1 bit
      0x00008000 0: b16, 1: b32
      0x00000100 sat flag
      operands: S*DST, S*SRC1/S*SHARED, S*SRC2/S*CONST/IMM, $c0

Long:     0x20000000 0x00000000 base opcode
      0x10000000 0x00000000 O2 bit
      0x00400000 0x00000000 O1 bit
      0x00000000 0x04000000 0: b16, 1: b32
      0x00000000 0x08000000 sat flag
      operands: MCDST, LL*DST, L*SRC1/L*SHARED, L*SRC3/L*CONST3, COND
```

Multiplication: mul(24)

Todo

write me

```
mul [CDST] DST u16/s16 SRC1 u16/s16 SRC2

DST is 32-bit, SRC1 and SRC2 are 16-bit.

  b32 s1, s2;
  if (src1_signed)
      s1 = SEX(SRC1);
  else
      s1 = ZEX(SRC1);
  if (src2_signed)
      s2 = SEX(SRC2);
  else
      s2 = ZEX(SRC2);
  b32 res = s1*s2;    // modulo 2^32
  CDST.O = 0;
  CDST.C = 0;
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;
```

```
Short/imm:    0x40000000 base opcode
      0x00008000 src1 is signed
      0x00000100 src2 is signed
      operands: SDST, SHSRC/SHSHARED, SHSRC2/SHCONST/IMM

Long:     0x40000000 0x00000000 base opcode
      0x00000000 0x00008000 src1 is signed
      0x00000000 0x00004000 src2 is signed
      operands: MCDST, LLDST, LHSRC1/LHSHARED, LHSRC2/LHCONST2
```

```
mul [CDST] DST [high] u24/s24 SRC1 SRC2

All operands are 32-bit.

  b48 s1, s2;
  if (signed) {
      s1 = SEX((b24)SRC1);
      s2 = SEX((b24)SRC2);
  } else {
      s1 = ZEX((b24)SRC1);
      s2 = ZEX((b24)SRC2);
  }
  b48 m = s1*s2;  // modulo 2^48
  b32 res = (high ? m >> 16 : m & 0xffffffff);
  CDST.O = 0;
  CDST.C = 0;
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Short/imm:    0x40000000 base opcode
      0x00008000 src are signed
      0x00000100 high
      operands: SDST, SSRC/SSHARED, SSRC2/SCONST/IMM

Long:     0x40000000 0x00000000 base opcode
      0x00000000 0x00008000 src are signed
      0x00000000 0x00004000 high
      operands: MCDST, LLDST, LSRC1/LSHARED, LSRC2/LCONST2
```

Multiply-add: madd(24), msub(24), msubr(24), maddc(24)

Todo

write me

```
addop [CDST] DST mul u16 SRC1 SRC2 SRC3       O1=0 O2=000 S2=0 S1=0
addop [CDST] DST mul s16 SRC1 SRC2 SRC3       O1=0 O2=001 S2=0 S1=1
addop sat [CDST] DST mul s16 SRC1 SRC2 SRC3       O1=0 O2=010 S2=1 S1=0
addop [CDST] DST mul u24 SRC1 SRC2 SRC3       O1=0 O2=011 S2=1 S1=1
addop [CDST] DST mul s24 SRC1 SRC2 SRC3       O1=0 O2=100
addop sat [CDST] DST mul s24 SRC1 SRC2 SRC3       O1=0 O2=101
addop [CDST] DST mul high u24 SRC1 SRC2 SRC3  O1=0 O2=110
addop [CDST] DST mul high s24 SRC1 SRC2 SRC3  O1=0 O2=111
addop sat [CDST] DST mul high s24 SRC1 SRC2 SRC3  O1=1 O2=000

addop is one of:
```

```
add   O3=00   S4=0 S3=0
sub   O3=01   S4=0 S3=1
subr  O3=10   S4=1 S3=0
addc  O3=11   S4=1 S3=1

If addop is addc, insn also takes an additional COND parameter. DST and
SRC3 are always 32-bit, SRC1 and SRC2 are 16-bit for u16/s16 variants,
32-bit for u24/s24 variants. Only a few of the variants are encodable as
short/immediate, and they're restricted to DST=SRC3.

  if (u24 || s24) {
      b48 s1, s2;
      if (s24) {
          s1 = SEX((b24)SRC1);
          s2 = SEX((b24)SRC2);
      } else {
          s1 = ZEX((b24)SRC1);
          s2 = ZEX((b24)SRC2);
      }
      b48 m = s1*s2;  // modulo 2^48
      b32 mres = (high ? m >> 16 : m & 0xffffffff);
  } else {
      b32 s1, s2;
      if (s16) {
          s1 = SEX(SRC1);
          s2 = SEX(SRC2);
      } else {
          s1 = ZEX(SRC1);
          s2 = ZEX(SRC2);
      }
      b32 mres = s1*s2;   // modulo 2^32
  }
  b32 s1, s2;
  bool c;
  switch (OP) {
      case add: s1 = mres, s2 = SRC3, c = 0; break;
      case sub: s1 = mres, s2 = ~SRC3, c = 1; break;
      case subr: s1 = ~mres, s2 = SRC3, c = 1; break;
      case addc: s1 = mres, s2 = SRC3, c = COND.C; break;
  }
  res = s1+s2+c;  // infinite precision
  CDST.C = res >> 32;
  res = res & 0xffffffff;
  CDST.O = (S(s1) == S(s2)) && (S(s1) != S(res));
  if (sat && CDST.O)
      if (S(res)) res = 0x7fffffff;
      else res = 0x80000000;
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Short/imm:    0x60000000 base opcode
      0x00000100 S1
      0x00008000 S2
      0x00400000 S3
      0x10000000 S4
      operands: SDST, S*SRC/S*SHARED, S*SRC2/S*CONST/IMM, SDST, $c0
```

```
Long:     0x60000000 0x00000000 base opcode
      0x10000000 0x00000000 O1
      0x00000000 0xe0000000 O2
      0x00000000 0x0c000000 O3
      operands: MCDST, LLDST, L*SRC1/L*SHARED, L*SRC2/L*CONST2, L*SRC3/L*CONST3, COND
```

Sum of absolute differences: sad, hsad

Todo

write me

```
sad [CDST] DST u16/s16/u32/s32 SRC1 SRC2 SRC3

Short variant is restricted to DST same as SRC3. All operands are 32-bit or
16-bit according to size specifier.

  int s1, s2; // infinite precision
  if (signed) {
      s1 = SEX(SRC1);
      s2 = SEX(SRC2);
  } else {
      s1 = ZEX(SRC1);
      s2 = ZEX(SRC2);
  }
  b32 mres = abs(s1-s2);  // modulo 2^32
  res = mres+s3;      // infinite precision
  CDST.C = res >> (b32 ? 32 : 16);
  res = res & (b32 ? 0xffffffff : 0xffff);
  CDST.O = (S(mres) == S(s3)) && (S(mres) != S(res));
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Short:    0x50000000 base opcode
      0x00008000 0: b16 1: b32
      0x00000100 src are signed
      operands: DST, SDST, S*SRC/S*SHARED, S*SRC2/S*CONST, SDST

Long:     0x50000000 0x00000000 base opcode
      0x00000000 0x04000000 0: b16, 1: b32
      0x00000000 0x08000000 src sre signed
      operands: MCDST, LLDST, L*SRC1/L*SHARED, L*SRC2/L*CONST2, L*SRC3/L*CONST3
```

Min/max selection: (h)min, (h)max

Todo

write me

```
min u16/u32/s16/s32 [CDST] DST SRC1 SRC2
max u16/u32/s16/s32 [CDST] DST SRC1 SRC2

All operands are 32-bit or 16-bit according to size specifier.

  if (SRC1 < SRC2) { // signed comparison for s16/s32, unsigned for u16/u32.
      res = (min ? SRC1 : SRC2);
```

```
  } else {
      res = (min ? SRC2 : SRC1);
  }
  CDST.O = 0;
  CDST.C = 0;
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Long:     0x30000000 0x80000000 base opcode
      0x00000000 0x20000000 0: max, 1: min
      0x00000000 0x08000000 0: u16/u32, 1: s16/s32
      0x00000000 0x04000000 0: b16, 1: b32
      operands: MCDST, LL*DST, L*SRC1/L*SHARED, L*SRC2/L*CONST2
```

Comparison: set, hset

Todo

write me

```
set [CDST] DST cond u16/s16/u32/s32 SRC1 SRC2

cond can be any subset of {l, g, e}.

All operands are 32-bit or 16-bit according to size specifier.

  int s1, s2; // infinite precision
  if (signed) {
      s1 = SEX(SRC1);
      s2 = SEX(SRC2);
  } else {
      s1 = ZEX(SRC1);
      s2 = ZEX(SRC2);
  }
  bool c;
  if (s1 < s2)
      c = cond.l;
  else if (s1 == s2)
      c = cond.e;
  else /* s1 > s2 */
      c = cond.g;
  if (c) {
      res = (b32?0xffffffff:0xffff);
  } else {
      res = 0;
  }
  CDST.O = 0;
  CDST.C = 0;
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Long:     0x30000000 0x60000000 base opcode
      0x00000000 0x08000000 0: u16/u32, 1: s16/s32
      0x00000000 0x04000000 0: b16, 1: b32
      0x00000000 0x00010000 cond.g
```

```
      0x00000000 0x00008000 cond.e
      0x00000000 0x00004000 cond.l
      operands: MCDST, LL*DST, L*SRC1/L*SHARED, L*SRC2/L*CONST2
```

Bitwise operations: (h)and, (h)or, (h)xor, (h)mov2

Todo

write me

```
and b32/b16 [CDST] DST [not] SRC1 [not] SRC2      O2=0, O1=0
or b32/b16 [CDST] DST [not] SRC1 [not] SRC2       O2=0, O1=1
xor b32/b16 [CDST] DST [not] SRC1 [not] SRC2      O2=1, O1=0
mov2 b32/b16 [CDST] DST [not] SRC1 [not] SRC2     O2=1, O1=1

Immediate forms only allows 32-bit operands, and cannot negate second op.

  s1 = (not1 ? ~SRC1 : SRC1);
  s2 = (not2 ? ~SRC2 : SRC2);
  switch (OP) {
      case and: res = s1 & s2; break;
      case or: res = s1 | s2; break;
      case xor: res = s1 ^ s2; break;
      case mov2: res = s2; break;
  }
  CDST.O = 0;
  CDST.C = 0;
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Imm:      0xd0000000 base opcode
      0x00400000 not1
      0x00008000 O2 bit
      0x00000100 O1 bit
      operands: SDST, SSRC/SSHARED, IMM
      assumed: not2=0 and b32.

Long:     0xd0000000 0x00000000 base opcode
      0x00000000 0x04000000 0: b16, 1: b32
      0x00000000 0x00020000 not2
      0x00000000 0x00010000 not1
      0x00000000 0x00008000 O2 bit
      0x00000000 0x00004000 O1 bit
      operands: MCDST, LL*DST, L*SRC1/L*SHARED, L*SRC2/L*CONST2
```

Bit shifts: (h)shl, (h)shr, (h)sar

Todo

write me

```
shl b16/b32 [CDST] DST SRC1 SRC2
shl b16/b32 [CDST] DST SRC1 SHCNT
shr u16/u32 [CDST] DST SRC1 SRC2
shr u16/u32 [CDST] DST SRC1 SHCNT
shr s16/s32 [CDST] DST SRC1 SRC2
```

```
shr s16/s32 [CDST] DST SRC1 SHCNT

  All operands 16/32-bit according to size specifier, except SHCNT. Shift
  counts are always treated as unsigned, passing negative value to shl
  doesn't get you a shr.

      int size = (b32 ? 32 : 16);
  if (shl) {
      res = SRC1 << SRC2; // infinite precision, shift count doesn't wrap.
      if (SRC2 < size) { // yes, <. So if you shift 1 left by 32 bits, you DON'T get CDST.C set. but
          CDST.C = (res >> size) & 1; // basically, the bit that got shifted out.
      } else {
          CDST.C = 0;
      }
      res = res & (b32 ? 0xffffffff : 0xffff);
  } else {
      res = SRC1 >> SRC2; // infinite precision, shift count doesn't wrap.
      if (signed && S(SRC1)) {
          if (SRC2 < size)
              res |= (1<<size)-(1<<(size-SRC2)); // fill out the upper bits with 1's.
          else
              res |= (1<<size)-1;
      }
      if (SRC2 < size && SRC2 > 0) {
          CDST.C = (SRC1 >> (SRC2-1)) & 1;
      } else {
          CDST.C = 0;
      }
  }
  if (SRC2 == 1) {
      CDST.O = (S(SRC1) != S(res));
  } else {
      CDST.O = 0;
  }
  CDST.S = S(res);
  CDST.Z = res == 0;
  DST = res;

Long:     0x30000000 0xc0000000 base opcode
      0x00000000 0x20000000 0: shl, 1: shr
      0x00000000 0x08000000 0: u16/u32, 1: s16/s32 [shr only]
      0x00000000 0x04000000 0: b16, 1: b32
      0x00000000 0x00010000 0: use SRC2, 1: use SHCNT
      operands: MCDST, LL*DST, L*SRC1/L*SHARED, L*SRC2/L*CONST2/SHCNT
```

Floating point instructions

Contents

Introduction

Todo

write me

Addition: fadd

Todo

write me

```
add [sat] rn/rz f32 DST SRC1 SRC2

  Adds two floating point numbers together.
```

Multiplication: fmul

Todo

write me

```
mul [sat] rn/rz f32 DST SRC1 SRC2

  Multiplies two floating point numbers together
```

Multiply+add: fmad

Todo

write me

```
add f32 DST mul SRC1 SRC2 SRC3

  A multiply-add instruction. With intermediate rounding. Nothing
  interesting. DST = SRC1 * SRC2 + SRC3;
```

Min/max: fmin, fmax

Todo

write me

```
min f32 DST SRC1 SRC2
max f32 DST SRC1 SRC2

  Sets DST to the smaller/larger of two SRC1 operands. If one operand is NaN,
  DST is set to the non-NaN operand. If both are NaN, DST is set to NaN.
```

Comparison: fset

Todo

write me

```
set [CDST] DST <cmpop> f32 SRC1 SRC2

  Does given comparison operation on SRC1 and SRC2. DST is set to 0xffffffff
  if comparison evaluats true, 0 if it evaluates false. if used, CDST.SZ are
  set according to DST.
```

Selection: fslct

Todo

write me

```
slct b32 DST SRC1 SRC2 f32 SRC3

  Sets DST to SRC1 if SRC3 is positive or 0, to SRC2 if SRC3 negative or NaN.
```

Transcendential instructions

Contents

Introduction

Todo

write me

Preparation: pre

Todo

write me

```
presin f32 DST SRC
preex2 f32 DST SRC

  Preprocesses a float argument for use in subsequent sin/cos or ex2
  operation, respectively.
```

Reciprocal: rcp

Todo

write me

```
rcp f32 DST SRC

  Computes 1/x.
```

Reciprocal square root: rsqrt

Todo

write me

```
rsqrt f32 DST SRC

  Computes 1/sqrt(x).
```

Base-2 logarithm: lg2

Todo

write me

```
lg2 f32 DST SRC

  Computes log_2(x).
```

Sinus/cosinus: sin, cos

Todo

write me

```
sin f32 DST SRC
cos f32 DST SRC

  Computes sin(x) or cos(x), needs argument preprocessed by pre.sin.
```

Base-2 exponential: ex2

Todo

write me

```
ex2 f32 DST SRC

  Computes 2**x, needs argument preprocessed by pre.ex2.
```

Double precision floating point instructions

Contents

Introduction

Todo

write me

Addition: dadd

Todo

write me

Multiplication: dmul

Todo

write me

Fused multiply+add: dfma

Todo

write me

```
fma f64 DST SRC1 SRC2 SRC3

  Fused multiply-add, with no intermediate rounding.
```

Min/max: dmin, dmax

Todo

write me

```
min f64 DST SRC1 SRC2
max f64 DST SRC1 SRC2

  Sets DST to the smaller/larger of two SRC1 operands. If one operand is NaN,
  DST is set to the non-NaN operand. If both are NaN, DST is set to NaN.
```

Comparison: dset

Todo

write me

```
set [CDST] DST <cmpop> f64 SRC1 SRC2

  Does given comparison operation on SRC1 and SRC2. DST is set to 0xffffffff
  if comparison evaluats true, 0 if it evaluates false. if used, CDST.SZ are
  set according to DST.
```

Control instructions

Contents

Introduction

Todo

write me

Halting program execution: exit

Todo

write me

```
exit

  Actually, not a separate instruction, just a modifier available on all
  long insns. Finishes thread's execution after the current insn ends.
```

Branching: bra

Todo

write me

```
bra <code target>

  Branches to the given place in the code. If only some subset of threads
  in the current warp executes it, one of the paths is chosen as the active
  one, and the other is suspended until the active path exits or rejoins.
```

Indirect branching: bra c[]

Todo

write me

Setting up a rejoin point: joinat

Todo

write me

```
joinat <code target>

  The arugment is address of a future join instruction and gets pushed
  onto the stack, together with a mask of currently active threads, for
  future rejoining.
```

Rejoining execution paths: join

Todo

write me

```
join

  Also a modifier. Switches to other diverged execution paths on the same
  stack level, until they've all reached the join point, then pops off the
  entry and continues execution with a rejoined path.
```

Preparing a loop: prebrk

Todo

write me

```
breakaddr <code target>

  Like call, except doesn't branch anywhere, uses given operand as the
  return address, and pushes a different type of entry onto the stack.
```

Breaking out of a loop: brk

Todo

write me

```
break

  Like ret, except accepts breakaddr's stack entry type, not call's.
```

Calling subroutines: call

Todo

write me

```
call <code target>

  Pushes address of the next insn onto the stack and branches to given place.
  Cannot be predicated.
```

Returning from a subroutine: ret

Todo

write me

```
ret

  Returns from a called function. If there's some not-yet-returned divergent
  path on the current stack level, switches to it. Otherwise pops off the
  entry from stack, rejoins all the paths to the pre-call state, and
  continues execution from the return address on stack. Accepts predicates.
```

Pushing a return address: preret

Todo

write me

Aborting execution: trap

Todo

write me

```
trap

  Causes an error, killing the program instantly.
```

Debugger breakpoint: brkpt

Todo

write me

```
brkpt

  Doesn't seem to do anything, probably generates a breakpoint when enabled
  somewhere in PGRAPH, somehow.
```

Enabling whole-quad mode: quadon, quadpop

Todo

write me

```
quadon

  Temporarily enables all threads in the current quad, even if they were
  disabled before [by diverging, exitting, or not getting started at all].
  Nesting this is probably a bad idea, and so is using any non-quadpop
  control insns while this is active. For diverged threads, the saved PC
  is unaffected by this temporal enabling.

quadpop

  Undoes a previous quadon command.
```

Discarding fragments: discard

Todo

write me

Block thread barriers: bar

Todo

write me

```
bar sync <barrier number>

  Waits until all threads in the block arrive at the barrier, then continues
  execution... probably... somehow...
```

Texture instructions

Contents

Introduction

Todo

write me

Automatic texture load: texauto

Todo

write me

```
texauto [deriv] live/all <texargs>

  Does a texture fetch. Inputs are: x, y, z, array index, dref [skip all
  that your current sampler setup doesn't use]. x, y, z, dref are floats,
  array index is integer. If running in FP or the deriv flag is on,
  derivatives are computed based on coordinates in all threads of current
  quad. Otherwise, derivatives are assumed 0. For FP, if the live flag
  is on, the tex instruction is only run for fragments that are going to
  be actually written to the render target, ie. for ones that are inside
  the rendered primitive and haven't been discarded yet. all executes
  the tex even for non-visible fragments, which is needed if they're going
  to be used for further derivatives, explicit or implicit.
```

Raw texel fetch: texfetch

Todo

write me

```
texfetch live/all <texargs>

  A single-texel fetch. The inputs are x, y, z, index, lod, and are all
  integer.
```

Texture load with LOD bias: texbias

Todo

write me

```
texbias [deriv] live/all <texargs>

  Same as texauto, except takes an additional [last] float input specifying
  the LOD bias to add. Note that bias needs to be the same for all threads
  in the current quad executing the texbias insn.
```

Texture load with manual LOD: texlod

Todo

write me

```
Does a texture fetch with given coordinates and LOD. Inputs are like
texbias, except you have explicit LOD instead of the bias. Just like
in texbias, the LOD should be the same for all threads involved.
```

Texture size query: texsize

Todo

write me

```
texsize live/all <texargs>

  Gives you (width, height, depth, mipmap level count) in output, takes
  integer LOD parameter as its only input.
```

Texture cube calculations: texprep

Todo

write me

Texture LOD query: texquerylod

Todo

write me

Texture CSAA load: texcsaa

Todo

write me

Texture quad load: texgather

Todo

write me

Misc instructions

Contents

Introduction

Todo

write me

Data conversion: cvt

Todo

write me

```
cvt <integer dst> <integer src>
cvt <integer rounding modifier> <integer dst> <float src>
cvt <rounding modifier> <float dst> <integer src>
cvt <rounding modifier> <float dst> <float src>
cvt <integer rounding modifier> <float dst> <float src>

  Converts between formats. For integer destinations, always clamps result
  to target type range.
```

Attribute interpolation: interp

Todo

write me

```
interp [cent] [flat] DST v[] [SRC]

  Gets interpolated FP input, optionally multiplying by a given value
```

Intra-quad data movement: quadop

Todo

write me

```
quadop f32 <op1> <op2> <op3> <op4> DST <srclane> SRC1 SRC2

  Intra-quad information exchange instruction. Mad as a hatter.
```

```
  First, SRC1 is taken from the given lane in current quad. Then
  op<currentlanenumber> is executed on it and SRC2, results get
  written to DST. ops can be add [SRC1+SRC2], sub [SRC1-SRC2],
  subr [SRC2-SRC1], mov2 [SRC2]. srclane can be at least l0, l1,
  l2, l3, and these work everywhere. If you're running in FP, looks
  like you can also use dox [use current lane number ^ 1] and doy
  [use current lane number ^ 2], but using these elsewhere results
  in always getting 0 as the result...
```

Intra-warp voting: vote

Todo

write me

```
PREDICATE vote any/all CDST

  This instruction doesn't use the predicate field for conditional execution,
  abusing it instead as an input argument. vote any sets CDST to true iff the
  input predicate evaluated to true in any of the warp's active threads.
  vote all sets it to true iff the predicate evaluated to true in all acive
  threads of the current warp.
```

Vertex stream output control: emit, restart

Todo

write me

```
emit

  GP-only instruction that emits current contents of $o registers as the
  next vertex in the output primitive and clears $o for some reason.

restart

  GP-only instruction that finishes current output primitive and starts
  a new one.
```

Nop / PM event triggering: nop, pmevent

Todo

write me

Vertex fetch: VFETCH

Contents

Todo

write me

PCOUNTER signals

Mux 0:

- 0x0e: geom_vertex_in_count[0]
- 0x0f: geom_vertex_in_count[1]
- 0x10: geom_vertex_in_count[2]
- 0x19: CG_IFACE_DISABLE [G80]

Mux 1:

- 0x02: input_assembler_busy[0]
- 0x03: input_assembler_busy[1]
- 0x08: geom_primitive_in_count
- 0x0b: input_assembler_waits_for_fb [G200:]
- 0x0e: input_assembler_waits_for_fb [G80:G200]
- 0x14: input_assembler_busy[2] [G200:]
- 0x15: input_assembler_busy[3] [G200:]
- 0x17: input_assembler_busy[2] [G80:G200]
- 0x18: input_assembler_busy[3] [G80:G200]

Mux 2 [G84:]:

- 0x00: CG[0]
- 0x01: CG[1]
- 0x02: CG[2]

Pre-ROP: PROP

Contents

Todo

write me

PCOUNTER signals

- 0x00:
  - 2: rop_busy[0]
  - 3: rop_busy[1]
  - 4: rop_busy[2]
  - 5: rop_busy[3]
  - 6: rop_waits_for_shader[0]
  - 7: rop_waits_for_shader[1]
- 0x03: shaded_pixel_count...?
- 0x15:
  - 0-5: rop_samples_in_count_1
  - 6: rop_samples_in_count_0[0]
  - 7: rop_samples_in_count_0[1]
- 0x16:
  - 0-5: rasterizer_pixels_out_count_1
  - 6: rasterizer_pixels_out_count_0[0]
  - 7: rasterizer_pixels_out_count_0[1]
- 0x1a:
  - 0-5: rop_samples_killed_by_earlyz_count
- 0x1b:
  - 0-5: rop_samples_killed_by_latez_count
- 0x1c: shaded_pixel_count...?
- 0x1d: shaded_pixel_count...?
- 0x1e:
  - 0: CG_IFACE_DISABLE [G80]
  - 0: CG[0] [G84:]
  - 1: CG[1] [G84:]
  - 2: CG[2] [G84:]

Color raster output: CROP

Contents

Todo

write me

PCOUNTER signals

- 0x1:
  - 0: CG_IFACE_DISABLE [G80]
  - 2: rop_waits_for_fb[0]
  - 3: rop_waits_for_fb[1]

Zeta raster output: ZROP

Contents

Todo

write me

PCOUNTER signals

- 0x1:
  - 2: rop_waits_for_fb[0]
  - 3: rop_waits_for_fb[1]
- 0x4:
  - 1: CG_IFACE_DISABLE [G80]

2.9.12 Fermi graphics and compute engine

Contents:

Fermi macro processor

Contents

Introduction

Todo

write me

Registers

Todo

write me

```
404400+i*4, i<8: REG[]
404420: OPCODE [at PC]
404424: PC
404428: NEXTPC
40442c: STATE
                b0: ?
                b4: ?
                b8: ACTIVE
                b9: PARM/MADDR?
404430: ??? 117ffff
404434: ??? 17ffff
404438: ??? 13ffff
404460: ??? 7f
404464: ??? 7ff
404468: WATCHDOG_TIMEOUT [30-bit]
40446c: WATCHDOG_TIME [30-bit]
404480: ??? 3
404488: MCACHE_CTRL
40448c: MCACHE_DATA
404490: TRAP
        b0: TOO_FEW_PARAMS
        b1: TOO_MANY_PARAMS
        b2: ILLEGAL_OPCODE
        b3: DOUBLE_BRANCH
        b4: TIMEOUT
        b29: ?
        b30: CLEAR
        b31: ENABLE
404494: TRAP_PC and something?
404498: 1/11/0
40449c: TRAP_OPCODE?
4044a0: STATUS [0000000f - idle]
```

Fermi context switching units

Contents:

Fermi context switching units

Todo

convert

Present on:

cc0: GF100:GK104

cc1: GK104:GK208

cc2: GK208:GM107

cc3: GM107:

BAR0 address:

HUB: 0x409000

GPC: 0x502000 + idx * 0x8000

PMC interrupt line: ??? PMC enable bit: 12 [all of PGRAPH] Version:

cc0, cc1: 3

cc2, cc3: 5

Code segment size: HUB cc0: 0x4000 HUB cc1, cc2: 0x5000 HUB cc3: 0x6000 GPC cc0: 0x2000 GPC cc1, cc2: 0x2800 GPC cc3: 0x3800

Data segment size: HUB: 0x1000 GPC cc0-cc2: 0x800 GPC cc3: 0xc00

Fifo size: HUB cc0-cc1: 0x10 HUB cc2-cc3: 0x8 GPC cc0-cc1: 0x8 GPC cc2-cc3: 0x4

Xfer slots: 8

Secretful: no

Code TLB index bits: 8

Code ports: 1

Data ports:

cc0, cc1: 1

cc2, cc3: 4

IO addressing type: indexed

Core clock:

HUB: hub clock [GF100 clock #9]

GPC: GPC clock [GF100 clock #0] [XXX: divider]

The IO register ranges:

400/10000:500/14000 CC misc CTXCTL support [graph/gf100-ctxctl/intro.txt] 500/14000:600/18000 FIFO command FIFO submission [graph/gf100-ctxctl/intro.txt] 600/18000:700/1c000 MC PGRAPH master control [graph/gf100-ctxctl/intro.txt] 700/1c000:800/20000 MMIO MMIO bus access [graph/gf100-ctxctl/mmio.txt] 800/20000:900/24000 MISC misc/unknown stuff [graph/gf100-ctxctl/intro.txt] 900/24000:a00/28000 STRAND context strand control [graph/gf100-ctxctl/strand.txt] a00/28000:b00/2c000 MEMIF memory interface [graph/gf100-ctxctl/memif.txt] b00/2c000:c00/30000 CSREQ PFIFO switch requests [graph/gf100-ctxctl/intro.txt] c00/30000:d00/34000 GRAPH PGRAPH status/control [graph/gf100-ctxctl/intro.txt] d80/36000:dc0/37000 ??? ??? - related to MEMIF? [XXX] [GK104-]

Registers in CC range: 400/10000 INTR interrupt signals 404/101xx INTR_ROUTE falcon interrupt routing 40c/1030x BAR_REQMASK[0] barrier required bits 410/1040x BAR_REQMASK[1] barrier required bits 414/1050x BAR barrier state 418/10600 BAR_SET[0] set barrier bits, barrier 0 41c/10700 BAR_SET[1] set barrier bits, barrier 1 420/10800 IDLE_STATUS CTXCTL subunit idle status 424/10900 USER_BUSY user busy flag 430/10c00 WATCHDOG watchdog timer 484/12100H ??? [XXX]

Registers in FIFO range: 500/14000 DATA FIFO command argument 504/14100 CMD FIFO command submission

Registers in MC range: 604/18100H HUB_UNITS PART/GPC count 608/18200G GPC_UNITS TPC/ZCULL count 60c/18300H ??? [XXX] 610/18400H ??? [XXX] 614/18500 RED_SWITCH enable/power/pause master control 618/18600G GPCID the id of containing GPC 620/18800 UC_CAPS falcon code and data size 698/1a600G ??? [XXX] 69c/1a700G ??? [XXX]

Registers in MISC range: 800/20000:820/20800 SCRATCH scratch registers 820/20000:820/21000 SCRATCH_SET set bits in scratch registers 840/20000:820/21800 SCRATCH_CLEAR clear bits in scratch registers 86c/21b00 ??? related to strands? [XXX] 870/21c00 ??? [XXX] 874/21d00 ??? [XXX] 878/21e00 ??? [XXX] 880/22000 STRANDS strand count 884/22100 ??? [XXX] 890/22400 ??? JOE? [XXX] 894/22500 ??? JOE? [XXX] 898/22600 ??? JOE? [XXX] 89c/22700 ??? JOE? [XXX] 8a0/22800 ??? [XXX] 8a4/22900 ??? [XXX] 8a8/22a00 ??? [XXX] 8b0/22c00 ??? [XXX] [GK104-] 8b4/22d00 ??? [XXX] [GK104-]

Registers in CSREQ range: b00/2c000H CHAN_CUR current channel b04/2c100H CHAN_NEXT next channel b08/2c200H INTR_EN interrupt enable? b0c/2c300H INTR interrupt b80/2e000H ??? [XXX] b84/2e100H ??? [XXX]

Registers in GRAPH range: c00/30000H CMD_STATUS some PGRAPH status bits? c08/30200H CMD_TRIGGER triggers misc commands to PGRAPH? c14/305xxH INTR_UP_ROUTE upstream interrupt routing c18/30600H INTR_UP_STATUS upstream interrupt status c1c/30700H INTR_UP_SET upstream interrupt trigger c20/30800H INTR_UP_CLEAR upstream interrupt clear c24/30900H INTR_UP_ENABLE upstream interrupt enable [XXX: more bits on GK104] c80/32000G VSTATUS_0 subunit verbose status c84/32100G VSTATUS_1 subunit verbose status c88/32200G VSTATUS_2 subunit verbose status c8c/32300G VSTATUS_3 subunit verbose status c90/32400G TRAP GPC trap status c94/32500G TRAP_EN GPC trap enable

Interrupts: 0-7: standard falcon intterrupts 8-15: controlled by INTR_ROUTE

[XXX: IO regs] [XXX: interrupts] [XXX: status bits]

[XXX: describe CTXCTL]

Signals 0x00-0x1f: engine dependent [XXX] 0x20: ZERO - always 0 0x21: ??? - bit 9 of reg 0x128 of corresponding IBUS piece [XXX] 0x22: STRAND - strand busy executing command [graph/gf100-ctxctl/strand.txt] 0x23: ???, affected by RED_SWITCH [XXX] 0x24: IB_UNK40, last state of IB_UNK40 bit, from DISPATCH.SUBCH reg 0x25: MMCTX - MMIO transfer complete [graph/gf100-ctxctl/mmio.txt] 0x26: MMIO_RD - MMIO read complete [graph/gf100-ctxctl/mmio.txt] 0x27: MMIO_WRS - MMIO synchronous write complete [graph/gf100-ctxctl/mmio.txt] 0x28: BAR_0 - barrier #0 reached [see below] 0x29: BAR_1 - barrier #1 reached [see below] 0x2a: ??? - related to PCOUNTER [XXX] 0x2b: WATCHDOG - watchdog timer expired [see below] 0x2c: ??? - related to MEMIF [XXX] 0x2d: ??? - related to MEMIF [XXX] 0x2e: ??? - related to MEMIF [XXX]

Fermi CUDA processors

Contents:

Fermi CUDA ISA

Contents

Introduction This file deals with description of Fermi CUDA instruction set. CUDA stands for Completely Unified Device Architecture and refers to the fact that all types of shaders (vertex, tesselation, geometry, fragment, and compute) use nearly the same ISA and execute on the same processors (called streaming multiprocessors).

The Fermi CUDA ISA is used on Fermi (GF1xx) and older Kepler (GK10x) GPUs. Older (Tesla) CUDA GPUs use the Tesla ISA. Newer Kepler ISAs use the Kepler2 ISA.

Variants There are two variants of the Fermi ISA: the GF100 variant (used on Fermi GPUs) and the GK104 variant (used on first-gen Kepler GPUs). The differences are:

- GF100:
  - surface access based on 8 bindable slots
- GK104:
  - surface access based on descriptor structures stored in c[]?
  - some new instructions
  - texbar instruction
  - every 8th instruction slot should be filled by a special `sched` instruction that describes dependencies and execution plan for the next 7 instructions

Todo

rather incomplete.

Warps and thread types Like on Tesla, programs are executed in *warps*.

There are 6 program types on Fermi:

- vertex programs
- tesselation control programs
- tesselation evaluation programs
- geometry programs
- fragment programs

- compute programs

Todo

and vertex programs 2?

Todo

figure out the exact differences between these & the pipeline configuration business

Registers The registers in Fermi ISA are:

- up to 63 32-bit GPRs per thread: $r0-$r62. These registers are used for all calculations, whether integer or floating-point. In addition, $r63 is a special register that's always forced to 0.

  The amount of available GPRs per thread is chosen by the user as part of MP configuration, and can be selected per program type. For example, if the user enables 16 registers, $r0-$r15 will be usable and $r16-$r62 will be forced to 0. Since the MP has a rather limitted amount of storage for GPRs, this configuration parameter determines how many active warps will fit simultanously on an MP.

  If a 64-bit operation is to be performed, any naturally aligned pair of GPRs can be treated as a 64-bit register: $rXd (which has the low half in $rX and the high half in $r(X+1), and X has to even). Likewise, if a 128-bit operation is to be performed, any naturally aligned group of 4 registers can be treated as a 128-bit registers: $rXq. The 32-bit chunks are assigned to $rX..(X+3) in order from lowest to highest.

  Unlike Tesla, there is no way to access a 16-bit half of a register.

- 7 1-bit predicate registers per thread: $p0-$p6. There's also $p7, which is always forced to 1. Used for conditional execution of instructions.
- 1 4-bit condition code register: $c. Has 4 bits:
  - bit 0: Z - zero flag. For integer operations, set when the result is equal to 0. For floating-point operations, set when the result is 0 or NaN.
  - bit 1: S - sign flag. For integer operations, set when the high bit of the result is equal to 1. For floating-point operations, set when the result is negative or NaN.
  - bit 2: C - carry flag. For integer addition, set when there is a carry out of the highest bit of the result.
  - bit 3: O - overflow flag. For integer addition, set when the true (infinite-precision) result doesn't fit in the destination (considered to be a signed number).

  Overall, works like one of the Tesla $c0-$c3 registers.
- $flags, a flags register, which is just an alias to $c and $pX registers, allowing them to be saved/restored with one mov:
  - bits 0-6: $p0-$p6
  - bits 12-15: $c
- A few dozen read-only 32-bit special registers, $sr0-$sr127:
  - $sr0 aka $laneid: XXX
  - $sr2 aka $nphysid: XXX
  - $sr3 aka $physid: XXX
  - $sr4-$sr11 aka $pm0-$pm7: XXX

- $sr16 aka $vtxcnt: XXX
- $sr17 aka $invoc: XXX
- $sr18 aka $ydir: XXX
- $sr24-$sr27 aka $machine_id0-$machine_id3: XXX
- $sr28 aka $affinity: XXX
- $sr32 aka $tid: XXX
- $sr33 aka $tidx: XXX
- $sr34 aka $tidy: XXX
- $sr35 aka $tidz: XXX
- $sr36 aka $launcharg: XXX
- $sr37 aka $ctaidx: XXX
- $sr38 aka $ctaidy: XXX
- $sr39 aka $ctaidz: XXX
- $sr40 aka $ntid: XXX
- $sr41 aka $ntidx: XXX
- $sr42 aka $ntidy: XXX
- $sr43 aka $ntidz: XXX
- $sr44 aka $gridid: XXX
- $sr45 aka $nctaidx: XXX
- $sr46 aka $nctaidy: XXX
- $sr47 aka $nctaidz: XXX
- $sr48 aka $swinbase: XXX
- $sr49 aka $swinsz: XXX
- $sr50 aka $smemsz: XXX
- $sr51 aka $smembanks: XXX
- $sr52 aka $lwinbase: XXX
- $sr53 aka $lwinsz: XXX
- $sr54 aka $lpossz: XXX
- $sr55 aka $lnegsz: XXX
- $sr56 aka $lanemask_eq: XXX
- $sr57 aka $lanemask_lt: XXX
- $sr58 aka $lanemask_le: XXX
- $sr59 aka $lanemask_gt: XXX
- $sr60 aka $lanemask_ge: XXX
- $sr64 aka $trapstat: XXX
- $sr66 aka $warperr: XXX

- $sr80 aka $clock: XXX
- $sr81 aka $clockhi: XXX

Todo

figure out and document the SRs

Memory The memory spaces in Fermi ISA are:

- C[]: code space. The only way to access this space is by executing code from it (there's no "read from code space" instruction). Unlike Tesla, the code segment is shared between all program types. It has three levels of cache (global, GPC, MP) that need to be manually flushed when its contents are modified by the user.
- c0[] - c17[]: const spaces. Read-only and accessible from any program type in 8, 16, 32, 64, and 128-bit chunks. Each of the 18 const spaces of each program type can be independently bound to a range of VM space (with length divisible by 256) or disabled by the user. Cached like C[].

  Todo

  figure out the semi-special c16[]/c17[].

- l[]: local space. Read-write and per-thread, accessible from any program type in 8, 16, 32, 64, and 128-bit units. It's directly mapped to VM space (although with heavy address mangling), and hence slow. Its per-thread length can be set to any multiple of 0x10 bytes.
- s[]: shared space. Read-write, per-block, available only from compute programs, accessible in 8, 16, 32, 64, and 128-bit units. Length per block can be selected by user. Has a locked access feature: every warp can have one locked location in s[], and all other warps will block when trying to access this location. Load with lock and store with unlock instructions can thus be used to implement atomic operations.

  Todo

  size granularity?

  Todo

  other program types?

- g[]: global space. Read-write, accessible from any program type in 8, 16, 32, 64, and 128-bit units. Mostly mapped to VM space. Supports some atomic operations. Can have two holes in address space: one of them mapped to s[] space, the other to l[] space, allowing unified addressing for the 3 spaces.

All memory spaces use 32-bit addresses, except g[] which uses 32-bit or 64-bit addresses.

Todo

describe the shader input spaces

Other execution state and resources There's also a fair bit of implicit state stored per-warp for control flow:

Todo

describe me

Other resources available to CUDA code are:

- $t0-$t129: up to 130 textures per 3d program type, up to 128 for compute programs.
- $s0-$s17: up to 18 texture samplers per 3d program type, up to 16 for compute programs. Only used if linked texture samplers are disabled.
- $g0-$g7: up to 8 random-access read-write image surfaces.
- Up to 16 barriers. Per-block and available in compute programs only. A barrier is basically a warp counter: a barrier can be increased or waited for. When a warp increases a barrier, its value is increased by 1. If a barrier would be increased to a value equal to a given warp count, it's set to 0 instead. When a barrier is waited for by a warp, the warp is blocked until the barrier's value is equal to 0.

Todo

not true for GK104. Not complete either.

Instruction format

Todo

write me

Instructions

Todo

write me

Notes about scheduling data and dual-issue on GK104+ There should be one "sched instructions" at each 0x40 byte boundary, i.e. one for each group of 7 "normal" instructions. For each of these 7 instructions, "sched" containts 1 byte of information:

```
0x00     : no scheduling info, suspend warp for 32 cycles
0x04     : dual-issue the instruction together with the next one **
0x20 | n : suspend warp for n cycles before trying to issue the next instruction
           (0 <= n < 0x20)
0x40     : ?
0x80     : ?

** obviously you can't use 0x04 on 2 consecutive instructions
```

If latency information is inaccurate and you encounter an instruction where its dependencies are not yet satisfied, the instruction is re-issued each cycle until they are.

EXAMPLE sched 0x28 0x20: inst_issued1/inst_executed = 6/2 sched 0x29 0x20: inst_issued1/inst_executed = 5/2 sched 0x2c 0x20: inst_issued1/inst_executed = 2/2 for mov b32 $r0 c0[0] set $p0 eq u32 $r0 0x1

DUAL ISSUE General constraints for which instructions can be dual-issued:

- not if same dst
- not if both access different 16-byte ranges inside cX[]
- not if any performs larger than 32 bit memory access
- a = b, b = c is allowed
- g[] access can't be dual-issued, ld seems to require 2 issues even for b32
- f64 ops seem to count as 3 instruction issues and can't be dual-issued with anything (GeForce only ?)

SPECIFIC (a X b means a cannot be dual-issued with any of b) mov gpr X mov sreg X mov sreg add int X shift X shift, mul int, cvt any, ins, popc mul int X mul int, shift, cvt any, ins, popc cvt any X cvt any, shift, mul int, ins, popc ins X ins, shift, mul int, cvt any, popc popc X popc, shift, mul int, cvt any, ins set any X set any logop X slct X ld l X ld l, ld s ld s X ld s, ld l

GF100 Fermi 3D objects

Contents

Todo

write me

Introduction

Todo

write me

GF100 Fermi compute objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.13 GK104 Kepler graphics and compute engine

Contents:

GK104 Kepler 3D objects

Contents

Todo

write me

Introduction

Todo

write me

GK104 Kepler compute objects

Contents

Todo

write me

Introduction

Todo

write me

2.9.14 GM107 Maxwell graphics and compute engine

Contents:

GM107 Maxwell 3D objects

Contents

Todo

write me

Introduction

Todo

write me

GM107 Maxwell compute objects

Contents

Todo

write me

Introduction

Todo

write me

2.10 falcon microprocessor

Contents:

2.10.1 Introduction

falcon is a class of general-purpose microprocessor units, used in multiple instances on nvidia GPUs starting from G98. Originally developed as the controlling logic for VP3 video decoding engines as a replacement for xtensa used on VP2, it was later used in many other places, whenever a microprocessor of some sort was needed.

A single falcon unit is made of:

- the core microprocessor with its code and data SRAM [see *Processor control*]
- an IO space containing control registers of all subunits, accessible from the host as well as from the code running on the falcon microprocessor [see *IO space*]
- common support logic:
  - interrupt controller [see *Interrupt delivery*]
  - periodic and watchdog timers [see *Timers*]
  - scratch registers for communication with host [see *Scratch registers*]
  - PCOUNTER signal output [see *Performance monitoring signals*]
  - some unknown other stuff
- optionally, FIFO interface logic, for falcon units used as PFIFO engines and some others [see *FIFO interface*]
- optionally, common memory interface logic [see *Memory interface*]. However, some engines have their own type of memory interface.
- optionally, a cryptographic AES coprocessor. A falcon unit with such coprocessor is called a "secretful" unit. [see *Cryptographic coprocessor*]
- any unit-specific logic the microprocessor is supposed to control

---

Todo

figure out remaining circuitry

The base falcon hardware comes in several different revisions:

- version 0: used on G98, MCP77, MCP79
- version 3: used on GT215+, adds a crude VM system for the code segment, edge/level interrupt modes, new instructions [division, software traps, bitfield manipulation, ...], and other features
- version 4: used on GF119+ for some engines [others are still version 3]: adds support for 24-bit code addressing, debugging and ???
- version 4.1: used on GK110+ for some engines, changes unknown
- version 5: used on GK208+ for some engines, redesigned ISA encoding

---

Todo

figure out v4 new stuff

Todo

figure out v4.1 new stuff

Todo

figure out v5 new stuff

The falcon units present on nvidia cards are:

- The VP3/VP4/VP5 engines [G98 and MCP77:GM107]:
  - PVLD, the variable length decoder
  - PPDEC, the picture decoder
  - PPPP, the video post-processor
- the VP6 engine [GM107-]:
  - PVDEC, the video decoder
- The VP3 security engine [G98, MCP77, MCP79, GM107-]:
  - PSEC, the security engine
- The GT215:GK104 copy engines:
  - PCOPY[0] [GT215:GK104]
  - PCOPY[1] [GF100:GK104]
- The GT215+ daemon engines:
  - *PDAEMON [GT215+]*
  - *PDISPLAY.DAEMON [GF119+]*
  - PUNK1C3 [GF119+]
- The Fermi PGRAPH CTXCTL engines:
  - PGRAPH.CTXCTL ../graph/gf100-ctxctl/intro.txt
  - PGRAPH.GPC[*].CTXCTL ../graph/gf100-ctxctl/intro.txt
- PVCOMP, the video compositing engine [MCP89:GF100]
- PVENC, the H.264 encoding engine [GK104+]

2.10.2 ISA

This file deals with description of the ISA used by the falcon microprocessor, which is described in *Introduction*.

Contents

Registers

There are 16 32-bit GPRs, $r0-$r15. There are also a dozen or so special registers:

Index	Name	Present on	Description
$sr0	$iv0	all units	*Interrupt 0 vector*
$sr1	$iv1	all units	*Interrupt 1 vector*
$sr3	$tv	all units	*Trap vector*
$sr4	$sp	all units	*Stack pointer*
$sr5	$pc	all units	*Program counter*
$sr6	$xcbase	all units	*Code xfer external base*
$sr7	$xdbase	all units	*Data xfer external base*
$sr8	$flags	all units	*Misc flags*
$sr9	$cx	crypto units	*Crypt xfer mode*
$sr10	$cauth	crypto units	*Crypt auth code selection*
$sr11	$xtargets	all units	*Xfer port selection*
$sr12	$tstatus	v3+ units	*Trap status*

$flags register

$flags [*$sr8*] register contains various flags controlling the operation of the falcon microprocessor. It is split into the following bitfields:

Bits	Name	Present on	Description
0-7	$p0-$p7	all units	*General-purpose predicates*
8	c	all units	*Carry flag*
9	o	all units	*Signed overflow flag*
10	s	all units	*Sign/negative flag*
11	z	all units	*Zero flag*
16	ie0	all units	*Interrupt 0 enable*
17	ie1	all units	*Interrupt 1 enable*
18	???	v4+ units	???
20	is0	all units	*Interrupt 0 saved enable*
21	is1	all units	*Interrupt 1 saved enable*
22	???	v4+ units	???
24	ta	all units	*Trap handler active*
26-28	???	v4+ units	???
29-31	???	v4+ units	???

Todo

figure out v4+ stuff

$p predicates $flags.p0-p7 are general-purpose single-bit flags. They can be used to store single-bit variables. They can be set via *bset, bclr, btgl*, and *setp* instructions. They can be read by *xbit* instruction, or checked by *sleep* and *bra* instructions.

Instructions

Instructions have 2, 3, or 4 bytes. First byte of instruction determines its length and format. High 2 bits of the first byte determine the instruction's operand size; 00 means 8-bit, 01 means 16-bit, 10 means 32-bit, and 11 means an instruction that doesn't use operand sizing. The set of available opcodes varies greatly with the instruction format.

The subopcode can be stored in one of the following places:

- O1: subopcode goes to low 4 bits of byte 0
- O2: subopcode goes to low 4 bits of byte 1
- OL: subopcode goes to low 6 bits of byte 1
- O3: subopcode goes to low 4 bits of byte 2

The operands are denoted as follows:

- R1x: register encoded in low 4 bits of byte 1
- R2x: register encoded in high 4 bits of byte 1
- R3x: register encoded in high 4 bits of byte 2
- RxS: register used as source
- RxD: register used as destination
- RxSD: register used as both source and destination
- I8: 8-bit immediate encoded in byte 2
- I16: 16-bit immediate encoded in bytes 2 [low part] and 3 [high part]

Sized

Sized opcodes are [low 6 bits of opcode]:

- 0x: O1 R2S R1S I8
- 1x: O1 R1D R2S I8
- 2x: O1 R1D R2S I16
- 30: O2 R2S I8
- 31: O2 R2S I16
- 34: O2 R2D I8
- 36: O2 R2SD I8
- 37: O2 R2SD I16

- 38: O3 R2S R1S
- 39: O3 R1D R2S
- 3a: O3 R2D R1S
- 3b: O3 R2SD R1S
- 3c: O3 R3D R2S R1S
- 3d: O2 R2SD

Todo

long call/branch

The subopcodes are as follows:

Instruction	0x	1x	2x	30	31	34	36	37	38	39	3a	3b	3c	3d	imm	flg0	flg3+	Cycles	Present on	Description
st	0								0						U	-	-	1	all units	*store*
st [sp]				1					1						U	-	-		all units	*store*
cmpu				4	4				4						U	CZ	CZ	1	all units	*unsigned compare*
cmps				5	5				5						S	CZ	CZ	1	all units	*signed compare*
cmp				6	6				6						S	N/A	COSZ	1	v3+ units	*compare*
add		0	0				0	0				0	0		U	COSZ	COSZ	1	all units	*add*
adc		1	1				1	1				1	1		U	COSZ	COSZ	1	all units	*add with carry*
sub		2	2				2	2				2	2		U	COSZ	COSZ	1	all units	*substract*
sbb		3	3				3	3				3	3		U	COSZ	COSZ	1	all units	*substract with borrow*
shl		4					4					4	4		U	C	COSZ	1	all units	*shift left*
shr		5					5					5	5		U	C	COSZ	1	all units	*shift right*
sar		7					7					7	7		U	C	COSZ	1	all units	*shift right with sign*
ld		8											8		U	-	-	1	all units	*load*
shlc		c					c					c	c		U	C	COSZ	1	all units	*shift left with carry*
shrc		d					d					d	d		U	C	COSZ	1	all units	*shift right with carry*
ld [sp]						0					0				U	-	-		all units	*load*
not										0				0		OSZ	OSZ	1	all units	*bitwise not*
neg										1				1		OSZ	OSZ	1	all units	*sign negation*
movf										2				2		OSZ	N/A	1	v0 units	*move*
mov										2				2		N/A	-	1	v3+ units	*move*
hswap										3				3		OSZ	OSZ	1	all units	*swap halves*

Unsized

Unsized opcodes are:

- cx: O1 R1D R2S I8
- dx: O1 R2S R1S I8
- ex: O1 R1D R2S I16
- f0: O2 R2SD I8
- f1: O2 R2SD I16
- f2: O2 R2S I8
- f4: OL I8
- f5: OL I16
- f8: O2
- f9: O2 R2S
- fa: O3 R2S R1S
- fc: O2 R2D
- fd: O3 R2SD R1S
- fe: O3 R1D R2S
- ff: O3 R3D R2S R1S

The subopcodes are as follows:

Instruction	cx	dx	ex	f0	f1	f2	f4	f5	f8	f9	fa	fc	fd	fe	ff	imm	flg0	flg3+	cycl
mulu	0		0	0	0								0		0	U	-	-	1
muls	1		1	1	1								1		1	S	-	-	1
sext	2			2									2		2	U	SZ	SZ	1
extrs	3		3												3	U	N/A	SZ	1
sethi				3	3											H	-	-	1
and	4		4	4	4								4		4	U	-	COSZ	1
or	5		5	5	5								5		5	U	-	COSZ	1
xor	6		6	6	6								6		6	U	-	COSZ	1
extr	7		7												7	U	N/A	SZ	1
mov				7	7											S	-	-	1
xbit	8														8	U	-	SZ	1
bset				9									9			U	-	-	1
bclr				a									a			U	-	-	1
btgl				b									b			U	-	-	1
ins	b		b													U	N/A	-	1
xbit[fl]				c										c		U	-	SZ	
div	c		c												c	U	N/A	-	30-3
mod	d		d												d	U	N/A	-	30-3
???	e														e	U	-	-	
iord	f														f	U	-	-	~1-x
iowr		0									0					U	-	-	1-x
iowrs		1									1					U	N/A	-	9-x
xcld											4						-	-	

Table 2.9 – continued from previous page

Instruction	cx	dx	ex	f0	f1	f2	f4	f5	f8	f9	fa	fc	fd	fe	ff	imm	flg0	flg3+	cycl
xdld											5						-	-	
xdst											6						-	-	
setp						8					8						-	-	
ccmd						c	3c	3c									-	-	
bra							0x	0x								S	-	-	5
bra							1x	1x								S	-	-	5
jmp							20	20		4						U	-	-	4-5
call							21	21		5						U	-	-	4-5
sleep							28									U	-	-	NA
add [sp]							30	30		1						S	-	-	1
bset[fl]							31			9						U	-	-	
bclr[fl]							32			a						U	-	-	
btgl[fl]							33			b						U	-	-	
ret									0								-	-	5-6
iret									1								-	-	
exit									2								-	-	
xdwait									3								-	-	
???									6								-	-	
xcwait									7								-	-	
trap 0									8								N/A	-	
trap 1									9								N/A	-	
trap 2									a								N/A	-	
trap 3									b								N/A	-	
push										0							-	-	1
itlb										8							N/A	-	
pop												0					-	-	1
mov[>sr]														0			-	-	
mov[<sr]														1			-	-	
ptlb														2			N/A	-	
vtlb														3			N/A	-	

Code segment

falcon has separate code and data spaces. Code segment, like data segment, is located in small piece of SRAM in the microcontroller. Its size can be determined by looking at MMIO address falcon+0x108, bits 0-8 shifted left by 8.

Code is byte-oriented, but can only be accessed by 32-bit words from outside, and can only be modified in 0x100-byte [page] units.

On v0, code segment is just a flat piece of RAM, except for the per-page secret flag. See *v0 code/data upload registers* for information on uploading code and data.

On v3+, code segment is paged with virtual -> physical translation and needs special handling. See *IO space* for details.

Code execution is started by host via MMIO from arbitrary entry point, and is stopped either by host or by the microcode itself, see *Halting microcode execution: exit*, *Processor execution control registers*.

Invalid opcode handling

When an invalid opcode is hit, $pc is unmodified and a trap is generated. On v3+, $tstatus reason field is set to 8. v0 engines don't have $tstatus register, but this is the only trap type they support anyway.

2.10.3 Arithmetic instructions

Contents

Introduction

The arithmetic/logical instructions do operations on $r0-$r15 GPRs, sometimes setting bits in $flags register according to the result. The instructions can be "sized" or "unsized". Sized instructions have 8-bit, 16-bit, and 32-bit variants. Unsized instructions don't have variants, and always operate on full 32-bit registers. For 8-bit and 16-bit sized instructions, high 24 or 16 bits of destination registers are unmodified.

$flags result bits

The *$flags* bits often affected by ALU instructions are:

- bit 8: c, carry flag
- bit 9: o, signed overflow flag
- bit 10: s, sign flag
- bit 11: z, zero flag

Also, a few ALU instructions operate on $flags register as a whole.

Pseudocode conventions

All operations are done in infinite-precision arithmetic, all temporaries are infinite-precision too.

sz, for sized instructions, is the selected size of operation: 8, 16, or 32.

S(x) evaluates to (x >> (sz - 1) & 1), ie. the sign bit of x. If insn is unsized, assume sz = 32.

Comparison: cmpu, cmps, cmp

Compare two values, setting flags according to results of comparison. cmp sets the usual set of 4 flags. cmpu sets only c and z. cmps sets z normally, and sets c if SRC1 is less then SRC2 when treated as signed number.

cmpu/cmps are the only comparison instructions available on falcon v0. Both of them set only the c and z flags, with cmps setting c flag in an unusual way to enable signed comparisons while using unsigned flags and condition codes. To do an unsigned comparison, use cmpu and the unsigned branch conditions [b/a/e]. To do a signed comparison, use cmps, also with unsigned branch conditions.

The falcon v3+ new cmp instruction sets the full set of flags. To do an unsigned comparison on v3+, use cmp and the unsigned branch conditions. To do a signed comparison, use cmp and the signed branch conditions [l/g/e].

Instructions:

Name	Description	Present on	Subopcode
cmpu	compare unsigned	all units	4
cmps	compare signed	all units	5
cmp	compare	v3+ units	6

Instruction class: sized

Execution time: 1 cycle

Operands: SRC1, SRC2

Forms:

Form	Opcode
R2, I8	30
R2, I16	31
R2, R1	38

Immediates:

cmpu: zero-extended

cmps: sign-extended

cmp: sign-extended

Operation:

```
diff = SRC1 - SRC2; // infinite precision
S = S(diff);
O = S(SRC1) != S(SRC2) && S(SRC1) != S(diff);
$flags.z = (diff == 0);
if (op == cmps)
        $flags.c = S ^ O;
else if (op == cmpu)
        $flags.c = diff >> sz & 1;
else if (op == cmp) {
        $flags.c = diff >> sz & 1;
        $flags.o = O;
        $flags.s = S;
}
```

Addition/substraction: add, adc, sub, sbb

Add or substract two values, possibly with carry/borrow.

Instructions:

Name	Description	Subopcode
add	add	0
adc	add with carry	1
sub	substract	2
sbb	substrace with borrow	3

Instruction class: sized

Execution time: 1 cycle

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	10
R1, R2, I16	20
R2, R2, I8	36
R2, R2, I16	37
R2, R2, R1	3b
R3, R2, R1	3c

Immediates: zero-extended

Operation:

```
    s2 = SRC2;
    if (op == adc || op == sbb)
            s2 += $flags.c;
    if (op == sub || op == sbb)
            s2 = -s2;
    res = SRC1 + s2;
    DST = res;
    $flags.c = (res >> sz) & 1;
    if (op == add || op == adc) {
            $flags.o = S(SRC1) == S(SRC2) && S(SRC1) != S(res);
    } else {
            $flags.o = S(SRC1) != S(SRC2) && S(SRC1) != S(res);
    }
    $flags.s = S(DST);
    $flags.z = (DST == 0);
```

Shifts: shl, shr, sar, shlc, shrc

Shift a value. For shl/shr, the extra bits "shifted in" are 0. For sar, they're equal to sign bit of source. For shlc/shrc, the first such bit is taken from carry flag, the rest are 0.

Instructions:

Name	Description	Subopcode
shl	shift left	4
shr	shift right	5
sar	shift right with sign bit	6
shlc	shift left with carry in	c
shrc	shift right with carry in	d

Instruction class: sized

Execution time: 1 cycle

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	10
R2, R2, I8	36
R2, R2, R1	3b
R3, R2, R1	3c

Immediates: truncated

Operation:

```
    if (sz == 8)
            shcnt = SRC2 & 7;
    else if (sz == 16)
            shcnt = SRC2 & 0xf;
    else // sz == 32
            shcnt = SRC2 & 0x1f;
    if (op == shl || op == shlc) {
            res = SRC1 << shcnt;
            if (op == shlc && shcnt != 0)
                    res |= $flags.c << (shcnt - 1);
            $flags.c = res >> sz & 1;
            DST = res;
    } else { // shr, sar, shrc
            res = SRC1 >> shcnt;
            if (op == shrc && shcnt != 0)
                    res |= $flags.c << (sz - shcnt);
            if (op == sar && S(SRC1))
                    res |= ~0 << (sz - shcnt);
            if (shcnt == 0)
                    $flags.c = 0;
            else
                    $flags.c = SRC1 >> (shcnt - 1) & 1;
            DST = res;
    }
    if (falcon_version != 0) {
            $flags.o = 0;
            $flags.s = S(DST);
            $flags.z = (DST == 0);
    }
```

Unary operations: not, neg, mov, movf, hswap

not flips all bits in a value. neg negates a value. mov and movf move a value from one register to another. mov is the v3+ variant, which just does the move. movf is the v0 variant, which additionally sets flags according to the moved value. hswap rotates a value by half its size.

Instructions:

Name	Description	Present on	Subopcode
not	bitwise complement	all units	0
neg	negate a value	all units	1
movf	move a value and set flags	v0 units	2
mov	move a value	v3+ units	2
hswap	Swap halves	all units	3

Instruction class: sized

Execution time: 1 cycle

Operands: DST, SRC

Forms:

Form	Opcode
R1, R2	39
R2, R2	3d

Operation:

```
    if (op == not) {
            DST = ~SRC;
            $flags.o = 0;
    } else if (op == neg) {
            DST = -SRC;
            $flags.o = (DST == 1 << (sz - 1));
    } else if (op == movf) {
            DST = SRC;
            $flags.o = 0;
    } else if (op == mov) {
            DST = SRC;
    } else if (op == hswap) {
            DST = SRC >> (sz / 2) | SRC << (sz / 2);
            $flags.o = 0;
    }
    if (op != mov) {
            $flags.s = S(DST);
            $flags.z = (DST == 0);
    }
```

Loading immediates: mov, sethi

mov sets a register to an immediate. sethi sets high 16 bits of a register to an immediate, leaving low bits untouched. mov can be thus used to load small [16-bit signed] immediates, while mov+sethi can be used to load any 32-bit immediate.

Instructions

Name	Description	Subopcode
mov	Load an immediate	7
sethi	Set high bits	3

Instruction class: unsized

Execution time: 1 cycle

Operands: DST, SRC

Forms:

Form	Opcode
R2, I8	f0
R2, I16	f1

Immediates:

mov: sign-extended

sethi: zero-extended

Operation:

```
    if (op == mov)
            DST = SRC;
    else if (op == sethi)
            DST = DST & 0xffff | SRC << 16;
```

Clearing registers: clear

Sets a register to 0.

Instructions:

Name	Description	Subopcode
clear	Clear a register	4

Instruction class: sized

Operands: DST

Forms:

Form	Opcode
R2	3d

Operation:

```
DST = 0;
```

Setting flags from a value: setf

Sets flags according to a value.

Instructions:

Name	Description	Present on	Subopcode
setf	Set flags according to a value	v3+ units	5

Instruction class: sized

Execution time: 1 cycle

Operands: SRC

Forms:

Form	Opcode
R2	3d

Operation:

```
$flags.o = 0;
$flags.s = S(SRC);
$flags.z = (SRC == 0);
```

Multiplication: mulu, muls

Does a 16x16 -> 32 multiplication.

Instructions:

Name	Description	Subopcode
mulu	Multiply unsigned	0
muls	Multiply signed	1

Instruction class: unsized

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	c0
R1, R2, I16	e0
R2, R2, I8	f0
R2, R2, I16	f1
R2, R2, R1	fd
R3, R2, R1	ff

Immediates:

mulu: zero-extended

muls: sign-extended

Operation:

```
s1 = SRC1 & 0xffff;
s2 = SRC2 & 0xffff;
if (op == muls) {
        if (s1 & 0x8000)
                s1 |= 0xffff0000;
        if (s2 & 0x8000)
                s2 |= 0xffff0000;
}
DST = s1 * s2;
```

Sign extension: sext

Does a sign-extension of low X bits of a value.

Instructions:

Name	Description	Subopcode
sext	Sign-extend	2

Instruction class: unsized

Execution time: 1 cycle

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	c0
R2, R2, I8	f0
R2, R2, R1	fd
R3, R2, R1	ff

Immediates: truncated

Operation:

```
bit = SRC2 & 0x1f;
if (SRC1 & 1 << bit) {
        DST = SRC1 & ((1 << bit) - 1) | -(1 << bit);
} else {
        DST = SRC1 & ((1 << bit) - 1);
}
$flags.s = S(DST);
$flags.z = (DST == 0);
```

Bitfield extraction: extr, extrs

Extracts a bitfield.

Instructions:

Name	Description	Present on	Subopcode
extrs	Extract signed bitfield	v3+ units	3
extr	Extract unsigned bitfield	v3+ units	7

Instruction class: unsized

Execution time: 1 cycle

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	c0
R1, R2, I16	e0
R3, R2, R1	ff

Immediates: zero-extended

Operation:

```
low = SRC2 & 0x1f;
size = (SRC2 >> 5 & 0x1f) + 1;
bf = (SRC1 >> low) & ((1 << size) - 1);
if (op == extrs) {
        signbit = (low + size - 1) & 0x1f; // depending on the mask is probably a bad idea.
        if (SRC1 & 1 << signbit)
                bf |= -(1 << size);
}
DST = bf;
```

Bitfield insertion: ins

Inserts a bitfield.

Instructions:

Name	Description	Present on	Subopcode
ins	Insert a bitfield	v3+ units	b

Instruction class: unsized

Execution time: 1 cycle

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	c0
R1, R2, I16	e0

Immediates: zero-extended.

Operation:

```
low = SRC2 & 0x1f;
size = (SRC2 >> 5 & 0x1f) + 1;
if (low + size <= 32) { // nop if bitfield out of bounds - I wouldn't depend on it, though...
        DST &= ~(((1 << size) - 1) << low); // clear the current contents of the bitfield
        bf = SRC1 & ((1 << size) - 1);
        DST |= bf << low;
}
```

Bitwise operations: and, or, xor

Ands, ors, or xors two operands.

Instructions:

Name	Description	Subopcode
and	Bitwise and	4
or	Bitwise or	5
xor	Bitwise xor	6

Instruction class: unsized

Execution time: 1 cycle

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	c0
R1, R2, I16	e0
R2, R2, I8	f0
R2, R2, I16	f1
R2, R2, R1	fd
R3, R2, R1	ff

Immediates: zero-extended

Operation:

```
    if (op == and)
            DST = SRC1 & SRC2;
    if (op == or)
            DST = SRC1 | SRC2;
    if (op == xor)
            DST = SRC1 ^ SRC2;
    if (falcon_version != 0) {
            $flags.c = 0;
            $flags.o = 0;
            $flags.s = S(DST);
            $flags.z = (DST == 0);
    }
```

Bit extraction: xbit

Extracts a single bit of a specified register. On v0, the bit is stored to bit 0 of DST, other bits are unmodified. On v3+, the bit is stored to bit 0 of DST, and all other bits of DST are set to 0.

Instructions:

Name	Description	Subopcode - opcodes c0, ff	Subopcode - opcodes f0, fe
xbit	Extract a bit	8	c

Instruction class: unsized

Execution time: 1 cycle

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	c0
R3, R2, R1	ff
R2, $flags, I8	f0
R1, $flags, R2	fe

Immediates: truncated

Operation:

```
    if (falcon_version == 0) {
            DST = DST & ~1 | (SRC1 >> bit & 1);
    } else {
            DST = SRC1 >> bit & 1;
            $flags.s = S(DST); // always works out to 0.
            $flags.z = (DST == 0);
    }
```

Bit manipulation: bset, bclr, btgl

Set, clear, or flip a specified bit of a register.

Instructions:

Name	Description	Subopcode - opcodes f0, fd, f9	Subopcode - opcode f4
bset	Set a bit	9	31
bclr	Clear a bit	a	32
btgl	Flip a bit	b	33

Instruction class: unsized

Execution time: 1 cycle

Operands: DST, SRC

Forms:

Form	Opcode
R2, I8	f0
R2, R1	fd
$flags, I8	f4
$flags, R2	f9

Immediates: truncated

Operation:

```
bit = SRC & 0x1f;
if (op == bset)
        DST |= 1 << bit;
else if (op == bclr)
        DST &= ~(1 << bit);
else // op == btgl
        DST ^= 1 << bit;
```

Division and remainder: div, mod

Does unsigned 32-bit division / modulus.

Instructions:

Name	Description	Present on	Subopcode
div	Divide	v3+ units	c
mod	Take modulus	v3+ units	d

Instruction class: unsized

Execution time: 30-33 cycles

Operands: DST, SRC1, SRC2

Forms:

Form	Opcode
R1, R2, I8	c0
R1, R2, I16	e0
R3, R2, R1	ff

Immediates: zero-extended

Operation:

```
if (SRC2 == 0) {
        dres = 0xffffffff;
} else {
        dres = SRC1 / SRC2;
}
```

```
    if (op == div)
            DST = dres;
    else // op == mod
            DST = SRC1 - dres * SRC2;
```

Setting predicates: setp

Sets bit #SRC2 in $flags to bit 0 of SRC1.

Instructions:

Name	Description	Subopcode
setp	Set predicate	8

Instruction class: unsized

Execution time: 1 cycle

Operands: SRC1, SRC2

Forms:

Form	Opcode
R2, I8	f2
R2, R1	fa

Immediates: truncated

Operation:

```
    bit = SRC2 & 0x1f;
    $flags = ($flags & ~(1 << bit)) | (SRC1 & 1) << bit;
```

2.10.4 Data space

Contents

Todo

document UAS

Introduction

Data segment of the falcon is inside the microcontroller itself. Its size can be determined by looking at *UC_CAPS register*, bits 9-16 shifted left by 8.

The segment has byte-oriented addressing and can be accessed in units of 8, 16, or 32 bits. Unaligned accesses are not supported and cause botched reads or writes.

Multi-byte quantities are stored as little-endian.

The stack

The stack is also stored in data segment. Stack pointer is stored in $sp special register and is always aligned to 4 bytes. Stack grows downwards, with $sp pointing at the last pushed value. The low 2 bits of $sp and bits higher than what's needed to span the data space are forced to 0.

Pseudocode conventions

sz, for sized instructions, is the selected size of operation: 8, 16, or 32.

LD(size, address) returns the contents of size-bit quantity in data segment at specified address:

```
int LD(size, addr) {
        if (size == 32) {
                addr &= ~3;
                return D[addr] | D[addr + 1] << 8 | D[addr + 2] << 16 | D[addr + 3] << 24;
        } else if (size == 16) {
                addr &= ~1;
                return D[addr] | D[addr + 1] << 8;
        } else { // size == 8
                return D[addr];
        }
}
```

ST(size, address, value) stores the given size-bit value to data segment:

```
void ST(size, addr, val) {
        if (size == 32) {
                if (addr & 1) { // fuck up the written datum as penalty for unaligned access.
                        val = (val & 0xff) << (addr & 3) * 8;
                } else if (addr & 2) {
                        val = (val & 0xffff) << (addr & 3) * 8;
                }
                addr &= ~3;
                D[addr] = val;
                D[addr + 1] = val >> 8;
                D[addr + 2] = val >> 16;
                D[addr + 3] = val >> 24;
        } else if (size == 16) {
                if (addr & 1) {
                        val = (val & 0xff) << (addr & 1) * 8;
                }
                addr &= ~1;
                D[addr] = val;
                D[addr + 1] = val >> 8;
        } else { // size == 8
                D[addr] = val;
        }
}
```

Load: ld

Loads 8-bit, 16-bit or 32-bit quantity from data segment to register.

Instructions:

Name	Description	Subopcode - normal	Subopcode - with $sp
ld	Load a value from data segment	8	0

Instruction class: sized

Operands: DST, BASE, IDX

Forms:

Form	Opcode
R1, R2, I8	10
R2, $sp, I8	34
R2, $sp, R1	3a
R3, R2, R1	3c

Immediates: zero-extended

Operation:

```
    DST = LD(sz, BASE + IDX * (sz/8));
```

Store: st

Stores 8-bit, 16-bit or 32-bit quantity from register to data segment.

Instructions:

Name	Description	Subopcode - normal	Subopcode - with $sp
st	Store a value to data segment	0	1

Instruction class: sized

Operands: BASE, IDX, SRC

Forms:

Form	Opcode
R2, I8, R1	00
$sp, I8, R2	30
R2, 0, R1	38
$sp, R1, R2	38

Immediates: zero-extended

Operation:

```
    ST(sz, BASE + IDX * (sz/8), SRC);
```

Push onto stack: push

Decrements $sp by 4, then stores a 32-bit value at top of the stack.

Instructions:

Name	Description	Subopcode
push	Push a value onto stack	0

Instruction class: unsized

Operands: SRC

Forms:

Form	Opcode
R2	f9

Operation:

```
    $sp -= 4;
    ST(32, $sp, SRC);
```

Pop from stack: pop

Loads 32-bit value from top of the stack, then incrments $sp by 4.

Instructions:

Name	Description	Subopcode
pop	Pops a value from the stack	0

Instruction class: unsized

Operands: DST

Forms:

Form	Opcode
R2	f2

Operation:

```
    DST = LD(32, $sp);
    $sp += 4;
```

Adjust stack pointer: add

Adds a value to the stack pointer.

Instructions:

Name	Description	Subopcode - opcodes f4, f5	Subopcode - opcode f9
add	Add a value to the stack pointer.	30	1

Instruction class: unsized

Operands: DST, SRC

Forms:

Form	Opcode
$sp, I8	f4
$sp, I16	f5
$sp, R2	f9

Immediates: sign-extended

Operation:

```
    $sp += SRC;
```

Accessing data segment through IO

On v3+, the data segment is accessible through normal IO space through index/data reg pairs. The number of available index/data pairs is accessible by *UC_CAPS2 register*. This number is equal to 4 on PDAEMON, 1 on other engines:

MMIO 0x1c0 + i * 8 / I[0x07000 + i * 0x200]: DATA_INDEX Selects the place in D[] accessed by DATA reg. Bits:

- bits 2-15: bits 2-15 of the data address to poke
- bit 24: write autoincrement flag: if set, every write to corresponding DATA register increments the address by 4
- bit 25: read autoincrement flag: like 24, but for reads

MMIO 0x1c4 + i * 8 / I[0x07100 + i * 0x200]: DATA Writes execute ST(32, DATA_INDEX & 0xfffc, value); and increment the address if write autoincrement is enabled. Reads return the result of LD(32, DATA_INDEX & 0xfffc); and increment if read autoincrement is enabled.

i should be less than DATA_PORTS value from *UC_CAPS2 register*.

On v0, the data segment is instead accessible through the high falcon MMIO range, see *v0 code/data upload registers* for details.

2.10.5 Branch instructions

Contents

Todo

document ljmp/lcall

Introduction

The flow control instructions on falcon include conditional relative branches, unconditional absolute branches, absolute calls, and returns. Calls use the stack in data segment for storage for return addresses [see *The stack*]. The conditions available for branching are based on the low 12 bits of $flags register:

- bits 0-7: p0-p7, general-purpose predicates
- bit 8: c, carry flag
- bit 9: o, signed overflow flag
- bit a: s, sign flag
- bit b: z, zero flag

c, o, s, z flags are automatically set by many ALU instructions, p0-p7 have to be explicitely manipulated. See *$flags result bits* for more details.

When a branching instruction is taken, the execution time is either 4 or 5 cycles. The execution time depends on the address of the next instruction to be executed. If this instruction can be loaded in one cycle (the instruction is contained in a single aligned 32-bit memory block in the code section), 4 cycles will be necessary. If the instruction is split in two blocks, 5 cycles will then be necessary.

$pc register

Address of the current instruction is always available through the read-only $pc special register.

Pseudocode conventions

$pc is usually automatically incremented by opcode length after each instruction - documentation for other kinds of instructions doesn't mention it explicitely for each insn. However, due to the nature of this category of instructions, all effects on $pc are mentioned explicitely in this file.

oplen is the length of the currently executed instruction in bytes.

See also conventions for *arithmetic* and *<data* instructions.

Conditional branch: bra

Branches to a given location if the condition evaluates to true. Target is $pc-relative.

Instructions:

Name	Description	Present on	Subopcode
bra pX	if predicate true	all units	00+X
bra c	if carry	all units	08
bra b	if unsigned below	all units	08
bra o	if overflow	all units	09
bra s	if sign set / negative	all units	0a
bra z	if zero	all units	0b
bra e	if equal	all units	0b
bra a	if unsigned above	all units	0c
bra na	if not unsigned above	all units	0d
bra be	if unsigned below or equal	all units	0d
bra	always	all units	0e
bra npX	if predicate false	all units	10+X
bra nc	if not carry	all units	18
bra nb	if not unsigned below	all units	18
bra ae	if unsigned above or equal	all units	18
bra no	if not overflow	all units	19
bra ns	if sign unset / positive	all units	1a
bra nz	if not zero	all units	1b
bra ne	if not equal	all units	1b
bra g	if signed greater	v3+ units	1c
bra le	if signed less or equal	v3+ units	1d
bra l	if signed less	v3+ units	1e
bra ge	if signed greater or equal	v3+ units	1f

Instruction class: unsized

Execution time: 1 cycle if not taken, 4-5 cycles if taken

Operands: DIFF

Forms:

Form	Opcode
I8	f4
I16	f5

Immediates: sign-extended

Operation:

```
    switch (cc) {
            case $pX: // $p0..$p7
                    cond = $flags.$pX;
                    break;
```

```
                case c:
                        cond = $flags.c;
                        break;
                case o:
                        cond = $flags.o;
                        break;
                case s:
                        cond = $flags.s;
                        break;
                case z:
                        cond = $flags.z;
                        break;
                case a:
                        cond = !$flags.c && !$flags.z;
                        break;
                case na:
                        cond = $flags.c || $flags.z;
                        break;
                case (none):
                        cond = 1;
                        break;
                case not $pX: // $p0..$p7
                        cond = !$flags.$pX;
                        break;
                case nc:
                        cond = !$flags.c;
                        break;
                case no:
                        cond = !$flags.o;
                        break;
                case ns:
                        cond = !$flags.s;
                        break;
                case nz:
                        cond = !$flags.z;
                        break;
                case g:
                        cond = !($flags.o ^ $flags.s) && !$flags.z;
                        break;
                case le:
                        cond = ($flags.o ^ $flags.s) || $flags.z;
                        break;
                case l:
                        cond = $flags.o ^ $flags.s;
                        break;
                case ge:
                        cond = !($flags.o ^ $flags.s);
                        break;
        }
        if (cond)
                $pc = $pc + DIFF;
        else
                $pc = $pc + oplen;
```

Unconditional branch: jmp

Branches to the target. Target is specified as absolute address. Yes, the immediate forms are pretty much redundant with the relative branch form.

Instructions:

Name	Description	Subopcode - opcodes f4, f5	Subopcode - opcode f9
jmp	Unconditional jump	20	4

Instruction class: unsized

Execution time: 4-5 cycles

Operands: TRG

Forms:

Form	Opcode
I8	f4
I16	f5
R2	f9

Immediates: zero-extended

Operation:

```
$pc = TRG;
```

Subroutine call: call

Pushes return address onto stack and branches to the target. Target is specified as absolute address.

Instructions:

Name	Description	Subopcode - opcodes f4, f5	Subopcode - opcode f9
call	Call a subroutine	21	5

Instruction class: unsized

Execution time: 4-5 cycles

Operands: TRG

Forms:

Form	Opcode
I8	f4
I16	f5
R2	f9

Immediates: zero-extended

Operation:

```
$sp -= 4;
ST(32, $sp, $pc + oplen);
$pc = TRG;
```

Subroutine return: ret

Returns from a previous call.

Instructions:

Name	Description	Subopcode
ret	Return from a subroutine	0

Instruction class: unsized

Execution time: 5-6 cycles

Operands: [none]

Forms:

Form	Opcode
[no operands]	f8

Operation:

```
    $pc = LD(32, $sp);
    $sp += 4;
```

2.10.6 Processor control

Contents

Todo

write me

Introduction

Todo

write me

Execution state

The falcon processor can be in one of three states:

- RUNNING: processor is actively executing instructions
- STOPPED: no instructions are being executed, interrupts are ignored
- SLEEPING: no instructions are being executed, but interrupts can restart execution

The state can be changed as follows:

From	To	Cause
any	STOPPED	Reset [non-crypto]
any	RUNNING	Reset [crypto]
STOPPED	RUNNING	*Start by UC_CTRL*
RUNNING	STOPPED	*Exit instruction*
RUNNING	STOPPED	*Double trap*
RUNNING	SLEEPING	*Sleep instruction*
SLEEPING	RUNNING	*Interrupt*

The EXIT interrupt

Whenever falcon execution state is changed to STOPPED for any reason other than reset (exit instruction, double trap, or the crypto reset scrubber finishing), falcon interrupt line 4 is active for one cycle (triggering the EXIT interrupt if it's set to level mode).

Halting microcode execution: exit

Halts microcode execution, raises EXIT interrupt.

Instructions:

Name	Description	Subopcode
exit	Halt microcode execution	2

Instruction class: unsized

Operands: [none]

Forms:

Form	Opcode
[no operands]	f8

Operation:

```
    EXIT;
```

Waiting for interrupts: sleep

If the *$flags* bit given as argument is set, puts the microprocessor in sleep state until an unmasked interrupt is received. Otherwise, is a nop. If interrupted, return pointer will point to start of the sleep instruction, restarting it if the $flags bit hasn't been cleared.

Instructions:

Name	Description	Subopcode
sleep	Wait for interrupts	28

Instruction class: unsized

Operands: FLAG

Forms:

Form	Opcode
I8	f4

Operation:

```
    if ($flags & 1 << FLAG)
            state = SLEEPING;
```

Processor execution control registers

Todo

write me

Accessing special registers: mov

Todo

write me

Processor capability readout

Todo

write me

2.10.7 Code virtual memory

Contents

Introduction

On v3+, the falcon code segment uses primitive paging/VM via simple reverse page table. The page size is 0x100 bytes.

The physical<->virtual address mapping information is stored in hidden TLB memory. There is one TLB cell for each physical code page, and it specifies the virtual address corresponding to it + some flags. The flags are:

- bit 0: usable. Set if page is mapped and complete.
- bit 1: busy. Set if page is mapped, but is still being uploaded.
- bit 2: secret. Set if page contains secret code. [see *Cryptographic coprocessor*]

Todo

check interaction of secret / usable flags and entering/exitting auth mode

A TLB entry is considered valid if any of the three flags is set. Whenever a virtual address is accessed, the TLBs are scanned for a valid entry with matching virtual address. The physical page whost TLB matched is then used to complete the access. It's an error if no page matched, or if there's more than one match.

The number of physical pages in the code segment can be determined by looking at *UC_CAPS register*, bits 0-8. Number of usable bits in virtual page index can be determined by looking at UC_CAPS2 register, bits 16-19. Ie. valid virtual addresses of pages are *0 .. (1 << (UC_CAPS2[16:19])) * 0x100*.

The TLBs can be modified/accessed in 6 ways:

- executing code - reads TLB corresponding to current $pc
- PTLB - looks up TLB for a given physical page
- VTLB - looks up TLB for a given virtual page
- ITLB - invalidates TLB of a given physical page
- uploading code via IO access window
- uploading code via xfer

We'll denote the flags of TLB entry of physical page i as TLB[i].flags, and the virtual page index as TLB[i].virt.

TLB operations: PTLB, VTLB, ITLB

These operations take 24-bit parameters, and except for ITLB return a 32-bit result. They can be called from falcon microcode as instructions, or through IO ports.

ITLB(physidx) clears the TLB entry corresponding to a specified physical page. The page is specified as page index. ITLB, however, cannot clear pages containing secret code - the page has to be reuploaded from scratch with non-secret data first.

```
void ITLB(b24 physidx) {
        if (!(TLB[physidx].flags & 4)) {
                TLB[physidx].flags = 0;
                TLB[physidx].virt = 0;
        }
}
```

PTLB(physidx) returns the TLB of a given physical page. The format of the result is:

- bits 0-7: 0
- bits 8-23: virtual page index
- bits 24-26: flags
- bits 27-31: 0

```
b32 PTLB(b24 physidx) {
        return TLB[physidx].flags << 24 | TLB[physidx].virt << 8;
}
```

VTLB(virtaddr) returns the TLB that covers a given virtual *address*. The result is:

- bits 0-7: physical page index

- bits 8-23: 0
- bits 24-26: flags, ORed across all matches
- bit 30: set if >1 TLB matches [multihit error]
- bit 31: set if no TLB matches [no hit error]

```
b32 VTLB(b24 virtaddr) {
        phys = 0;
        flags = 0;
        matches = 0;
        for (i = 0; i < UC_CAPS.CODE_PAGES; i++) {
                if (TLB[i].flags && TLB[i].virt == (virtaddr >> 8 & ((1 << UC_CAPS2.VM_PAGES_LOG2) -
                        flags |= TLB[i].flags;
                        phys = i;
                        matches++;
                }
        }
        res = phys | flags << 24;
        if (matches == 0)
                res |= 0x80000000;
        if (matches > 1)
                res |= 0x40000000;
        return res;
}
```

Executing TLB operations through IO

The three *TLB operations can be executed by poking TLB_CMD register. For PTLB and VTLB, the result will then be visible in TLB_CMD_RES register:

MMIO 0x140 / I[0x05000]: TLB_CMD Runs a given TLB command on write, returns last value written on read.

- bits 0-23: Parameter to the TLB command
- bits 24-25: TLB command to execute
  - 1: ITLB
  - 2: PTLB
  - 3: VTLB

MMIO 0x144 / I[0x05100]: TLB_CMD_RES Read-only, returns the result of the last PTLB or VTLB operation launched through TLB_CMD.

TLB readout instructions: ptlb, vtlb

These instructions run the corresponding TLB readout commands and return their results.

Instructions:

Name	Description	Present on	Subopcode
ptlb	run PTLB operation	v3+ units	2
vtlb	run VTLB operation	v3+ units	3

Instruction class: unsized

Operands: DST, SRC

Forms:

Form	Opcode
R1, R2	fe

Operation:

```
    if (op == ptlb)
            DST = PTLB(SRC);
    else
            DST = VTLB(SRC);
```

TLB invalidation instruction: itlb

This instructions runs the ITLB command.

Instructions:

Name	Description	Present on	Subopcode
itlb	run ITLB operation	v3+ units	8

Instruction class: unsized

Operands: SRC

Forms:

Form	Opcode
R2	f9

Operation:

```
    ITLB(SRC);
```

VM usage on code execution

Whenever instruction fetch is attempted, the VTLB operation is done on fetch address. If it returns no-hit or multihit error, a trap is generated and the $tstatus reason field is set to 0xa [for no-hit] or 0xb [for multihit]. Note that, if the faulting instruction happens to cross a page bounduary and the second page triggered a fault, the $pc register saved in $tstatus wiill not point to the page that faulted.

If no error was triggered, flag 0 [usable] is checked. If it's set, the access is finished using the physical page found by VTLB. If usable isn't set, but flag 1 [busy] is set, the fetch is paused and will be retried when TLBs are modified in any way. Otherwise, flag 2 [secret] must be the only flag set. In this case, a switch to authenticated mode is attempted - see *Cryptographic coprocessor* for details.

Code upload and peeking

Code can be uploaded in two ways: direct upload via a window in IO space, or by an xfer [see *Code/data xfers to/from external memory*]. The IO registers relevant are:

MMIO 0x180 / I[0x06000]: CODE_INDEX Selects the place in code segment accessed by CODE reg.

- bits 2-15: bits 2-15 of the physical code address to poke
- bit 24: write autoincrement flag: if set, every write to corresponding CODE register increments the address by 4
- bit 25: read autoincrement flag: like 24, but for reads
- bit 28: secret: if set, will attempt a switch to secret lockdown on next CODE write attempt and will mark uploaded code as secret.
- bit 29: secret lockdown [RO]: if set, currently in secret lockdown mode - CODE_INDEX cannot be modified manually until a complete page is uploaded and will auto-increment on CODE writes irrespective of write autoincrement flag. Reads will fail and won't auto-increment.

- bit 30: secret fail [RO]: if set, entering secret lockdown failed due to attempt to start upload from not page aligned address.
- bit 31: secret reset scrubber active [RO]: if set, the window isn't currently usable because the reset scrubber is busy.

See *Cryptographic coprocessor* for the secret stuff.

MMIO 0x184 / I[0x06100]: CODE Writes execute CST(CODE_INDEX & 0xfffc, value); and increment the address if write autoincrement is enabled or secret lockdown is in effect. Reads return the contents of code segment at physical address CODE_INDEX & 0xfffc and increment if read autoincrement is enabled and secret lockdown is not in effect. Attempts to read from physical code pages with the secret flag will return 0xdead5ec1 instead of the real contents. The values read/written are 32-bit LE numbers corresponding to 4 bytes in the code segment.

MMIO 0x188 / I[0x06200]: CODE_VIRT Selects the virtual page index for uploaded code. The index is sampled when writing word 0 of each page.

CST is defined thus:

```
void CST(addr, value) {
        physidx = addr >> 8;
        // if secret lockdown needed for the page, but starting from non-0 address, fail.
        if ((addr & 0xfc) != 0 && (CODE_INDEX.secret || TLB[physidx] & 4) && !CODE_INDEX.secret_lock
                CODE_INDEX.secret_fail = 1;
        if (CODE_INDEX.secret_fail || CODE_INDEX.secret_scrubber_active) {
                // nothing.
        } else {
                enter_lockdown = 0;
                exit_lockdown = 0;
                if ((addr & 0xfc) == 0) {
                        // if first word uploaded...
                        if (CODE_INDEX.secret || TLB[physidx].flags & 4) {
                                // if uploading secret code, or uploading code to replace secret code
                                enter_lockdown = 1;
                        }
                        // store virt addr
                        TLB[physid].virt = CODE_VIRT;
                        // clear usable flag, set busy flag
                        TLB[physid].flags = 2;
                        if (CODE_INDEX.secret)
                                TLB[physid].flags |= 4;
                }
                code[addr] = value; // write 4 bytes to code segment
                if ((addr & 0xfc) == 0xfc) {
                        // last word uploaded, page now complete.
                        exit_lockdown = 1;
                        // clear busy, set usable or secret
                        if (CODE_INDEX.secret)
                                TLB[physid].flags = 4;
                        else
                                TLB[physid].flags = 1;
                }
                if (CODE_INDEX.write_autoincrement || CODE_INDEX.secret_lockdown)
                        addr += 4;
                if (enter_lockdown)
                        CODE_INDEX.secret_lockdown = 1;
                if (exit_lockdown)
                        CODE_INDEX.secret_lockdown = 0;
}
```

In summary, to upload a single page of code:

1. Set CODE_INDEX to physical_addr | 0x1000000 [and | 0x10000000 if uploading secret code]
2. Set CODE_VIRT to virtual page index it should be mapped at
3. Write 0x40 words to CODE

Uploading code via xfers will set TLB[physid].virt = ext_offset >> 8 and TLB[physid].flags = (secret ? 6 : 2) right after the xfer is started, then set TLB[physid].flags = (secret ? 4 : 1) when it's complete. See *Code/data xfers to/from external memory* for more information.

2.10.8 Interrupts

Contents

Introduction

falcon has interrupt support. There are 16 interrupt lines on each engine, and two interrupt vectors on the microprocessor. Each of the interrupt lines can be independently routed to one of the microprocessor vectors, or to the PMC interrupt line, if the engine has one. The lines can be individually masked as well. They can be triggered by hw events, or by the user.

The lines are:

Line	v3+ type	Name	Description
0	edge	PERIODIC	*periodic timer*
1	edge	WATCHDOG	*watchdog timer*
2	level	FIFO	*FIFO data available*
3	edge	CHSW	*PFIFO channel switch*
4	edge	EXIT	*processor stopped*
5	edge	???	[related to falcon+0x0a4]
6-7	edge	SCRATCH	scratch [unused by hw, user-defined]
8-9	edge by default	-	engine-specific
10-15	level by default	-	engine-specific

Todo

figure out interrupt 5

Each interrupt line has a physical wire assigned to it. For edge-triggered interrupts, there's a flip-flop that's set by 0-to-1 edge on the wire or a write to INTR_SET register, and cleared by writing to INTR_CLEAR register. For level-triggered interrupts, interrupt status is wired straight to the input.

Interrupt status and enable registers

The interrupt and interrupt enable registers are actually visible as set/clear/status register triples: writing to the set register sets all bits that are 1 in the written value to 1. Writing to clear register sets them to 0. The status register shows the current value when read, but cannot be written.

MMIO 0x000 / I[0x00000]: INTR_SET MMIO 0x004 / I[0x00100]: INTR_CLEAR MMIO 0x008 / I[0x00200]: INTR [status]

> A mask of currently pending interrupts. Write to SET to manually trigger an interrupt. Write to CLEAR to ack an interrupt. Attempts to SET or CLEAR level-triggered interrupts are ignored.

MMIO 0x010 / I[0x00400]: INTR_EN_SET MMIO 0x014 / I[0x00500]: INTR_EN_CLEAR MMIO 0x018 / I[0x00600]: INTR_EN [status]

> A mask of enabled interrupts. If a bit is set to 0 here, the interrupt handler isn't run if a given interrupt happens [but the INTR bit is still set and it'll run once INTR_EN bit is set again].

Interrupt mode setup

MMIO 0x00c / I[0x00300]: INTR_MODE [v3+ only] Bits 0-15 are modes for the corresponding interrupt lines. 0 is edge trigered, 1 is level triggered.

> Setting a sw interrupt to level-triggered, or a hw interrupt to mode it wasn't meant to be set is likely a bad idea.
>
> This register is set to 0xfc04 on reset.

Todo

check edge/level distinction on v0

Interrupt routing

MMIO 0x01c / I[0x00700]: INTR_ROUTING

> - bits 0-15: bit 0 of interrupt routing selector, one for each interrupt line
> - bits 16-31: bit 1 of interrupt routing selector, one for each interrupt line
>
> For each interrupt line, the two bits from respective bitfields are put together to find its routing destination:
>
> - 0: falcon vector 0
> - 1: PMC HOST/DAEMON line
> - 2: falcon vector 1
> - 3: PMC NRHOST line [GF100+ selected engines only]

If the engine has a PMC interrupt line and any interrupt set for PMC irq delivery is active and unmasked, the corresponding PMC interrupt input line is active.

Interrupt delivery

falcon interrupt delivery is controlled by $iv0, $iv1 registers and ie0, ie1, is0, is1 $flags bits. $iv0 is address of interrupt vector 0. $iv1 is address of interrupt vector 1. ieX are interrupt enable bits for corresponding vectors. isX are interrupt enable save bits - they store previous status of ieX bits during interrupt handler execution. Both ieX bits are always cleared to 0 when entering an interrupt handler.

Whenever there's an active and enabled interrupt set for vector X delivery, and ieX flag is set, vector X is called:

```
$sp -= 4;
ST(32, $sp, $pc);
$flags.is0 = $flags.ie0;
$flags.is1 = $flags.ie1;
$flags.ie0 = 0;
$flags.ie1 = 0;
if (falcon_version >= 4) {
        $flags.unk16 = $flags.unk12;
        $flags.unk1d = $flags.unk1a;
        $flags.unk12 = 0;
}
if (vector 0)
        $pc = $iv0;
else
        $pc = $iv1;
```

Trap delivery

falcon trap delivery is controlled by $tv, $tstatus registers and ta $flags bit. Traps behave like interrupts, but are triggered by events inside the UC.

$tv is address of trap vector. ta is trap active flag. $tstatus is present on v3+ only and contains information about last trap. The bitfields of $tstatus are:

- bits 0-19 [or as many bits as required]: faulting $pc
- bits 20-23: trap reason

The known trap reasons are:

Reason	Name	Description
0-3	SOFTWARE	*software trap*
8	INVALID_OPCODE	*invalid opcode*
0xa	VM_NO_HIT	*page fault - no hit*
0xb	VM_MULTI_HIT	*page fault - multi hit*
0xf	BREAKPOINT	*breakpoint hit*

Whenever a trapworthy event happens on the uc, a trap is delivered:

```
if ($flags.ta) { // double trap?
        EXIT;
}
$flags.ta = 1;
if (falcon_version != 0) // on v0, there's only one possible trap reason anyway [8]
        $tstatus = $pc | reason << 20;
if (falcon_version >= 4) {
        $flags.is0 = $flags.ie0;
        $flags.is1 = $flags.ie1;
        $flags.unk16 = $flags.unk12;
        $flags.unk1d = $flags.unk1a;
        $flags.ie0 = 0;
        $flags.ie1 = 0;
        $flags.unk12 = 0;
}
$sp -= 4;
ST(32, $sp, $pc);
$pc = $tv;
```

Todo

didn't ieX -> isX happen before v4?

Returning form an interrupt: iret

Returns from an interrupt handler.

Instructions:

Name	Description	Subopcode
iret	Return from an interrupt	1

Instruction class: unsized

Operands: [none]

Forms:

Form	Opcode
[no operands]	f8

Operation:

```
$pc = LD(32, $sp);
$sp += 4;
$flags.ie0 = $flags.is0;
$flags.ie1 = $flags.is1;
if (falcon_version >= 4) {
        $flags.unk12 = $flags.unk16;
        $flags.unk1a = $flags.unk1d;
}
```

Software trap trigger: trap

Triggers a software trap.

Instructions:

Name	Description	Present on	Subopcode
trap 0	software trap #0	v3+ units	8
trap 1	software trap #1	v3+ units	9
trap 2	software trap #2	v3+ units	a
trap 3	software trap #3	v3+ units	b

Instruction class: unsized

Operands: [none]

Forms:

Form	Opcode
[no operands]	f8

Operation:

```
$pc += oplen; // return will be to the insn after this one
TRAP(arg);
```

2.10.9 Code/data xfers to/from external memory

Contents

Introduction

The falcon has a builtin DMA controller that allows running asynchronous copies beteween falcon data/code segments and external memory.

An xfer request consists of the following:

- mode: code load [external -> falcon code], data load [external -> falcon data], or data store [falcon data -> external]
- external port: 0-7. Specifies which external memory space the xfer should use.
- external base: 0-0xffffffff. Shifted left by 8 bits to obtain the base address of the transfer in external memory.
- external offset: 0-0xffffffff. Offset in external memory, and for v3+ code segments, virtual address that code should be loaded at.
- local address: 0-0xffff. Offset in falcon code/data segment where data should be transferred. Physical address for code xfers.
- xfer size: 0-6 for data xfers, ignored for code xfers [always effectively 6]. The xfer copies (4<<size) bytes.
- secret flag: Secret engines code xfers only. Specifies if the xfer should load secret code.

Todo

one more unknown flag on secret engines

Note that xfer functionality is greatly enhanced on secret engines to cover copying data to/from crypto registers. See *Cryptographic coprocessor* for details.

xfer requests can be submitted either through special falcon instructions, or through poking IO registers. The requests are stored in a queue and processed asynchronously.

A data load xfer copies (4<<$size) bytes from external memory port $port at address ($ext_base << 8) + $ext_offset to falcon data segment at address $local_address. external offset and local address have to be aligned to the xfer size.

A code load xfer copies 0x100 bytes from external memory port $port at address ($ext_base << 8) + $ext_offset to falcon code segment at physical address $local_address. Right after queuing the transfer, the code page is marked "busy" and, for v3+, mapped to virtual address $ext_offset. If the secret flag is set, it'll also be set for the page. When the transfer is finished, The page flags are set to "usable" for non-secret pages, or "secret" for secret pages.

xfer special registers

There are 3 falcon special registers that hold parameters for uc-originated xfer requests. $xdbase stores ext_base for data loads/stores, $xcbase stores ext_base for code loads. $xtargets stores the ports for various types of xfer:

- bits 0-2: port for code loads
- bits 8-10: port for data loads
- bits 12-14: port for data stores

The external memory that falcon will use depends on the particular engine. See ../graph/gf100-ctxctl/memif.txt for GF100 PGRAPH CTXCTLs, *Memory interface* for the other engines.

Submitting xfer requests: xcld, xdld, xdst

These instruction submit xfer requests of the relevant type. ext_base and port are taken from $xdbase/$xcbase and $xtargets special registers. ext_offset is taken from first operand, local_address is taken from low 16 bits of second operand, and size [for data xfers] is taken from bits 16-18 of the second operand. Secret flag is taken from $cauth bit 16.

Instructions:

Name	Description	Subopcode
xcld	code load	4
xdld	data load	5
xdst	data store	6

Instruction class: unsized

Operands: SRC1, SRC2

Forms:

Form	Opcode
R2, R1	fa

Operation:

```
    if (op == xcld)
            XFER(mode=code_load, port=$xtargets[0:2], ext_base=$xcbase,
                    ext_offset=SRC1, local_address=(SRC2&0xffff),
                            secret=($cauth[16:16]));
    else if (op == xdld)
            XFER(mode=data_load, port=$xtargets[8:10], ext_base=$xdbase,
                    ext_offset=SRC1, local_address=(SRC2&0xffff),
                            size=(SRC2>>16));
    else // xdst
            XFER(mode=data_store, port=$xtargets[12:14], ext_base=$xdbase,
                    ext_offset=SRC1, local_address=(SRC2&0xffff),
                            size=(SRC2>>16));
```

Waiting for xfer completion: xcwait, xdwait

These instructions wait until all xfers of the relevant type have finished.

Instructions:

Name	Description	Subopcode
xdwait	wait for all data loads/stores to finish	3
xcwait	wait for all code loads to finish	7

Instruction class: unsized

Operands: [none]

Forms:

Form	Opcode
[no operands]	f8

Operation:

```
    if (op == xcwait)
            while (XFER_ACTIVE(mode=code_load));
    else
            while (XFER_ACTIVE(mode=data_load) || XFER_ACTIVE(mode=data_store));
```

Submitting xfer requests via IO space

There are 4 IO registers that can be used to manually submit xfer reuqests. The request is sent out by writing XFER_CTRL register, other registers have to be set beforehand.

MMIO 0x110 / I[0x04400]: XFER_EXT_BASE Specifies the ext_base for the xfer that will be launched by XFER_CTRL.

MMIO 0x114 / I[0x04500]: XFER_LOCAL_ADDRESS Specifies the local_address for the xfer that will be launched by XFER_CTRL.

MMIO 0x118 / I[0x04600]: XFER_CTRL Writing requests a new xfer with given params, reading shows the last value written + two status flags

- bit 0: pending [RO]: The last write to XFER_CTRL is still waiting for place in the queue. XFER_CTRL shouldn't be written until this bit clears.
- bit 1: ??? [RO]
- bit 2: secret flag [secret engines only]
- bit 3: ??? [secret engines only]
- bits 4-5: mode
  - 0: data load
  - 1: code load
  - 2: data store
- bits 8-10: size
- bits 12-14: port

Todo

figure out bit 1. Related to 0x10c?

MMIO 0x11c / I[0x04700]: XFER_EXT_OFFSET Specifies the ext_offset for the xfer that will be launched by XFER_CTRL.

Todo

how to wait for xfer finish using only IO?

---

xfer queue status registers

The status of the xfer queue can be read out through an IO register:

MMIO 0x120 / I[0x04800]: XFER_STATUS

- bit 1: busy. 1 if any data xfer is pending.

- bits 4-5: ??? writable
- bits 16-18: number of data stores pending
- bits 24-26: number of data loads pending

Todo

bits 4-5

Todo

RE and document this stuff, find if there's status for code xfers

2.10.10 IO space

Contents

Introduction

Every falcon engine has an associated IO space. The space consists of 32-bit IO registers, and is accessible in two ways:

- host access by MMIO areas in BAR0
- falcon access by io* instructions

The IO space contains control registers for the microprocessor itself, interrupt and timer setup, code/data space access ports, PFIFO communication registers, as well as registers for the engine-specific hardware that falcon is meant to control.

The addresses are different between falcon and host. From falcon POV, the IO space is word-addressable 0x40000-byte space. However, most registers are duplicated 64 times: bits 2-7 of the address are ignored. The few registers that don't ignore these bits are called "indexed" registers. From host POV, the falcon IO space is a 0x1000-byte window in BAR0. Its base address is engine-dependent. First 0xf00 bytes of this window are tied to the falcon IO space, while last 0x100 bytes contain several host-only registers. On G98:GF119, host mmio address falcon_base + X is directed to falcon IO space address X << 6 | HOST_IO_INDEX << 2. On GF119+, some engines stopped using the indexed accesses. On those, host mmio address falcon_base + X is directed to falcon IO space address X. HOST_IO_INDEX is specified in the host-only MMIO register falcon_base + 0xffc:

MMIO 0xffc: HOST_IO_INDEX bits 0-5: selects bits 2-7 of the falcon IO space when accessed from host.

Unaligned accesses to the IO space are unsupported, both from host and falcon. Low 2 bits of addresses should be 0 at all times.

Todo

document v4 new addressing

Common IO register list

Host	Falcon	Present on	Name	Description
0x000	0x00000	all units	INTR_SET	*trigger interrupt*
0x004	0x00100	all units	INTR_CLEAR	*clear interrupt*
0x008	0x00200	all units	INTR	*interrupt status*
0x00c	0x00300	v3+ units	INTR_MODE	*interrupt edge/level*
0x010	0x00400	all units	INTR_EN_SET	*interrupt enable set*
0x014	0x00500	all units	INTR_EN_CLR	*interrupt enable clear*
0x018	0x00600	all units	INTR_EN	*interrupt enable status*
0x01c	0x00700	all units	INTR_DISPATCH	*interrupt routing*
0x020	0x00800	all units	PERIODIC_PERIOD	*periodic timer period*
0x024	0x00900	all units	PERIODIC_TIME	*periodic timer counter*
0x028	0x00a00	all units	PERIODIC_ENABLE	*periodic interrupt enable*
0x02c	0x00b00	all units	TIME_LOW	*PTIMER time low*
0x030	0x00c00	all units	TIME_HIGH	*PTIMER time high*
0x034	0x00d00	all units	WATCHDOG_TIME	*watchdog timer counter*
0x038	0x00e00	all units	WATCHDOG_ENABLE	*watchdog interrupt enable*
0x040	0x01000	all units	SCRATCH0	*scratch register*
0x044	0x01100	all units	SCRATCH1	*scratch register*
0x048	0x01200	all units	FIFO_ENABLE	*PFIFO access enable*
0x04c	0x01300	all units	STATUS	busy/idle status [falcon/io.txt]
0x050	0x01400	all units	CHANNEL_CUR	*current PFIFO channel*
0x054	0x01500	all units	CHANNEL_NEXT	*next PFIFO channel*
0x058	0x01600	all units	CHANNEL_CMD	*PFIFO channel control*
0x05c	0x01700	all units	STATUS_MASK	busy/idle status mask? [falcon/io.txt]
0x060	0x01800	all units	???	???
0x064	0x01900	all units	FIFO_DATA	*FIFO command data*
0x068	0x01a00	all units	FIFO_CMD	*FIFO command*
0x06c	0x01b00	v4+ units	???	*FIFO ???*
0x070	0x01c00	all units	FIFO_OCCUPIED	*FIFO commands available*
0x074	0x01d00	all units	FIFO_ACK	*FIFO command ack*
0x078	0x01e00	all units	FIFO_LIMIT	*FIFO size*
0x07c	0x01f00	all units	SUBENGINE_RESET	reset subengines [falcon/io.txt]
0x080	0x02000	all units	SCRATCH2	*scratch register*
0x084	0x02100	all units	SCRATCH3	*scratch register*
0x088	0x02200	all units	PM_TRIGGER	*perfmon triggers*
0x08c	0x02300	all units	PM_MODE	*perfmon signal mode*
0x090	0x02400	all units	???	???
0x094	0x02500	v3+ units	???	???
0x098	0x02600	v3+ units	BREAKPOINT[0]	*code breakpoint*
0x09c	0x02700	v3+ units	BREAKPOINT[1]	*code breakpoint*
0x0a0	0x02800	v3+ units	???	???
0x0a4	0x02900	v3+ units	???	???
0x0a8	0x02a00	v4+ units	PM_SEL	perfmon signal select [falcon/perf.txt]

Continued on next page

Table 2.10 – continued from previous page

Host	Falcon	Present on	Name	Description
0x0ac	0x02b00	v4+ units	HOST_IO_INDEX	IO space index for host [falcon/io.txt] [XXX: doc]
0x100	0x04000	all units	UC_CTRL	microprocessor control [falcon/proc.txt]
0x104	0x04100	all units	UC_ENTRY	microcode entry point [falcon/proc.txt]
0x108	0x04200	all units	UC_CAPS	microprocessor caps [falcon/proc.txt]
0x10c	0x04300	all units	UC_BLOCK_ON_FIFO	microprocessor block [falcon/proc.txt]
0x110	0x04400	all units	XFER_EXT_BASE	*xfer external base*
0x114	0x04500	all units	XFER_FALCON_ADDR	*xfer falcon address*
0x118	0x04600	all units	XFER_CTRL	*xfer control*
0x11c	0x04700	all units	XFER_EXT_ADDR	*xfer external offset*
0x120	0x04800	all units	XFER_STATUS	*xfer status*
0x124	0x04900	crypto units	CX_STATUS	crypt xfer status [falcon/crypt.txt]
0x128	0x04a00	v3+ units	UC_STATUS	microprocessor status [falcon/proc.txt]
0x12c	0x04b00	v3+ units	UC_CAPS2	microprocessor caps [falcon/proc.txt]
0x140	0x05000	v3+ units	TLB_CMD	*code VM command*
0x144	0x05100	v3+ units	TLB_CMD_RES	*code VM command result*
0x148	0x05200	v4+ units	???	???
0x14c	0x05300	v4+ units	???	???
0x150	0x05400	UNK31 units	???	???
0x154	0x05500	UNK31 units	???	???
0x158	0x05600	UNK31 units	???	???
0x160	0x05800	UAS units	UAS_IO_WINDOW	UAS I[] space window [falcon/data.txt]
0x164	0x05900	UAS units	UAS_CONFIG	UAS configuration [falcon/data.txt]
0x168	0x05a00	UAS units	UAS_FAULT_ADDR	UAS MMIO fault address [falcon/data.txt]
0x16c	0x05b00	UAS units	UAS_FAULT_STATUS	UAS MMIO fault status [falcon/data.txt]
0x180	0x06000	v3+ units	CODE_INDEX	*code access window addr*
0x184	0x06100	v3+ units	CODE	*code access window*
0x188	0x06200	v3+ units	CODE_VIRT_ADDR	*code access virt addr*
0x1c0	0x07000	v3+ units	DATA_INDEX[0]	*data access window addr*
0x1c4	0x07100	v3+ units	DATA[0]	*data access window*
0x1c8	0x07200	v3+ units	DATA_INDEX[1]	*data access window addr*
0x1cc	0x07300	v3+ units	DATA[1]	*data access window*
0x1d0	0x07400	v3+ units	DATA_INDEX[2]	*data access window addr*
0x1d4	0x07500	v3+ units	DATA[2]	*data access window*
0x1d8	0x07600	v3+ units	DATA_INDEX[3]	*data access window addr*
0x1dc	0x07700	v3+ units	DATA[3]	*data access window*
0x1e0	0x07800	v3+ units	DATA_INDEX[4]	*data access window addr*
0x1e4	0x07900	v3+ units	DATA[4]	*data access window*
0x1e8	0x07a00	v3+ units	DATA_INDEX[5]	*data access window addr*
0x1ec	0x07b00	v3+ units	DATA[5]	*data access window*
0x1f0	0x07c00	v3+ units	DATA_INDEX[6]	*data access window addr*
0x1f4	0x07d00	v3+ units	DATA[6]	*data access window*
0x1f8	0x07e00	v3+ units	DATA_INDEX[7]	*data access window addr*
0x1fc	0x07f00	v3+ units	DATA[7]	*data access window*
0x200	0x08000	v4+ units	DEBUG_CMD	debuging command [falcon/debug.txt]
0x204	0x08100	v4+ units	DEBUG_ADDR	address for DEBUG_CMD [falcon/debug.txt]
0x208	0x08200	v4+ units	DEBUG_DATA_WR	debug data to write [falcon/debug.txt]
0x20c	0x08300	v4+ units	DEBUG_DATA_RD	debug data last read [falcon/debug.txt]
0xfe8	-	GF100- v3	PM_SEL	perfmon signal select [falcon/perf.txt]
0xfec	-	v0, v3	UC_SP	microprocessor $sp reg [falcon/proc.txt]

Continued on next page

Table 2.10 – continued from previous page

Host	Falcon	Present on	Name	Description
0xff0	-	v0, v3	UC_PC	microprocessor $pc reg [falcon/proc.txt]
0xff4	-	v0, v3	UPLOAD	*old code/data upload*
0xff8	-	v0, v3	UPLOAD_ADDR	*old code/data up addr*
0xffc	-	v0, v3	HOST_IO_INDEX	IO space index for host [falcon/io.txt]

Todo

list incomplete for v4

Registers starting from 0x400/0x10000 are engine-specific and described in engine documentation.

Scratch registers

MMIO 0x040 / I[0x01000]: SCRATCH0 MMIO 0x044 / I[0x01100]: SCRATCH1 MMIO 0x080 / I[0x02000]: SCRATCH2 MMIO 0x084 / I[0x02100]: SCRATCH3

Scratch 32-bit registers, meant for host <-> falcon communication.

Engine status and control registers

MMIO 0x04c / I[0x01300]: STATUS Status of various parts of the engine. For each bit, 1 means busy, 0 means idle. bit 0: UC. Microcode. 1 if microcode is running and not on a sleep insn. bit 1: ??? Further bits are engine-specific.

MMIO 0x05c / I[0x01700]: STATUS_MASK A bitmask of nonexistent status bits. Each of bits 0-15 is set to 0 if corresponding STATUS line is tied to anything in this particular engine, 1 if it's unused. [?]

Todo

clean. fix. write. move.

MMIO 0x07c / I[0x01f00]: SUBENGINE_RESET When written with value 1, resets all subengines that this falcon engine controls - that is, everything in IO space addresses 0x10000:0x20000. Note that this includes the memory interface - using this register while an xfer is in progress is ill-advised.

v0 code/data upload registers

MMIO 0xff4: UPLOAD The data to upload, see below

MMIO 0xff8: UPLOAD_ADDR bits 2-15: bits 2-15 of the code/data address being uploaded. bit 20: target segment. 0 means data, 1 means code. bit 21: readback. bit 24: xfer busy [RO] bit 28: secret flag - secret engines only [see falcon/crypt.txt] bit 29: code busy [RO]

This pair of registers can be used on v0 to read/write code and data segments. It's quite fragile and should only be used when no xfers are active. bit 24 of UPLOAD_ADDR is set when this is the case. On v3+, this pair is broken and should be avoided in favor of the new-style access via *CODE* and *DATA* ports.

To write data, poke address to UPLOAD_ADDR, then poke the data words to UPLOAD. The address will auto-increment as words are uploaded.

To read data or code, poke address + readback flag to UPLOAD_ADDR, then read the word from UPLOAD. This only works for a single word, and you need to poke UPLOAD_ADDR again for each subsequent word.

The code segment is organised in 0x100-byte pages. On secretful engines, each page can be secret or not. Reading from secret pages doesn't work and you just get 0. Writing code segment can only be done in aligned page units.

To write a code page, write start address of the page + secret flag [if needed] to UPLOAD_ADDR, then poke multiple of 0x40 words to UPLOAD. The address will autoincrement. The process cannot be interrupted except between pages. The "code busy" flag in UPLOAD_ADDR will be lit when this is the case.

IO space writes: iowr, iowrs

Writes a word to IO space. iowr does asynchronous writes [queues the write, but doesn't wait for completion], iowrs does synchronous write [write is guaranteed to complete before executing next instruction]. On v0 cards, iowrs doesn't exist and synchronisation can instead be done by re-reading the relevant register.

Instructions:

Name	Description	Present on	Subopcode
iowr	Asynchronous IO space write	all units	0
iowrs	Synchronous IO space write	v3+ units	1

Instruction class: unsized

Operands: BASE, IDX, SRC

Forms:

Form	Subopcode
R2, I8, R1	d0
R2, 0, R1	fa

Immediates: zero-extended

Operation:

```
    if (op == iowr)
            IOWR(BASE + IDX * 4, SRC);
    else
            IOWRS(BASE + IDX * 4, SRC);
```

IO space reads: iord

Reads a word from IO space.

Instructions:

Name	Description	Present on	Subopcode
???	???	v3+ units	e
iord	IO space read	all units	f

Instruction class: unsized

Operands: DST, BASE, IDX

Forms:

Form	Subopcode
R1, R2, I8	c0
R3, R2, R1	ff

Immediates: zero-extended

Operation:

```
    if (op == iord)
            DST = IORD(BASE + IDX * 4);
```

```
    else
            ???;
```

Todo

subop e

2.10.11 Timers

Contents

Introduction

Time and timer-related registers are the same on all falcon engines, except PGRAPH CTXCTLs which lack PTIMER access.

You can:

- Read PTIMER's clock
- Use a periodic timer: Generate an interrupt periodically
- Use a watchdog/one-shot timer: Generate an interrupt once in the future

Also note that the CTXCTLs have another watchdog timer on their own - see ../graph/gf100-ctxctl/intro.txt for more information.

Periodic timer

All falcon engines have a periodic timer. This timer generates periodic interrupts on interrupt line. The registers controlling this timer are:

MMIO 0x020 / I[0x00800]: PERIODIC_PERIOD A 32-bit register defining the period of the periodic timer, minus 1.

MMIO 0x024 / I[0x00900]: PERIODIC_TIME A 32-bit counter storing the time remaining before the tick.

MMIO 0x028 / I[0x00a00]: PERIODIC_ENABLE bit 0: Enable the periodic timer. If 0, the counter doesn't change and no interrupts are generated.

When the counter is enabled, PERIODIC_TIME decreases by 1 every clock cycle. When PERIODIC_TIME reaches 0, an interrupt is generated on line 0 and the counter is reset to PERIODIC_PERIOD.

Operation (after each falcon core clock tick):

```
if (PERIODIC_ENABLE) {
        if (PERIODIC_TIME == 0) {
                PERIODIC_TIME = PERIODIC_PERIOD;
                intr_line[0] = 1;
```

```
        } else {
                PERIODIC_TIME--;
                intr_line[0] = 0;
        }
} else {
        intr_line[0] = 0;
}
```

= PTIMER access =

The falcon engines other than PGRAPH's CTXCTLs have PTIMER's time registers aliased into their IO space. aliases are:

MMIO 0x02c / I[0x00b00]: TIME_LOW Alias of PTIMER's TIME_LOW register [MMIO 0x9400]

MMIO 0x030 / I[0x00c00]: TIME_HIGH Alias of PTIMER's TIME_HIGH register [MMIO 0x9410]

Both of these registers are read-only. See ptimer for more information about PTIMER.

Watchdog timer

Apart from a periodic timer, the falcon engines also have an independent one-shot timer, also called watchdog timer. It can be used to set up a single interrupt in near future. The registers are:

MMIO 0x034 / I[0x00d00]: WATCHDOG_TIME A 32-bit counter storing the time remaining before the interrupt.

MMIO 0x038 / I[0x00e00]: WATCHDOG_ENABLE bit 0: Enable the watchdog timer. If 0, the counter doesn't change and no interrupts are generated.

A classic use of a watchdog is to set it before calling a sensitive function by initializing it to, for instance, twice the usual time needed by this function to be executed.

In falcon's case, the watchdog doesn't reboot the µc. Indeed, it is very similar to the periodic timer. The differences are:

- it generates an interrupt on line 1 instead of 0.
- it needs to be reset manually

Operation (after each falcon core clock tick):

```
if (WATCHDOG_ENABLE) {
        if (WATCHDOG_TIME == 0) {
                intr_line[1] = 1;
        } else {
                WATCHDOG_TIME--;
                intr_line[1] = 0;
        }
} else {
        intr_line[1] = 0;
}
```

2.10.12 Performance monitoring signals

Contents

Todo

write me

Introduction

Todo

write me

Main PCOUNTER signals

The main signals exported by falcon to PCOUNTER are:

Todo

docs & RE, please

- 0x00: SLEEPING
- 0x01: ??? fifo idle?
- 0x02: IDLE
- 0x03: ???
- 0x04: ???
- 0x05: TA
- 0x06: ???
- 0x07: ???
- 0x08: ???
- 0x09: ???
- 0x0a: ???
- 0x0b: ???
- 0x0c: PM_TRIGGER
- 0x0d: WRCACHE_FLUSH
- 0x0e-0x13: USER

User signals

MMIO 0x088 / I[0x02200]: PM_TRIGGER A WO "trigger" register for various things. write 1 to a bit to trigger the relevant event, 0 to do nothing.

- bits 0-5: ??? [perf counters?]
- bit 16: WRCACHE_FLUSH
- bit 17: ??? [PM_TRIGGER?]

MMIO 0x08c / I[0x02300]: PM_MODE bits 0-5: ??? [perf counters?]

Todo

write me

2.10.13 Debugging

Contents

Todo

write me

Breakpoints

Todo

write me

2.10.14 FIFO interface

Contents

Todo

write me

Introduction

Todo

write me

PFIFO access control

Todo

write me

Method FIFO

Todo

write me

Channel switching

Todo

write me

2.10.15 Memory interface

Contents

Todo

write me

Introduction

Todo

write me

IO Registers

Todo

write me

Error interrupts

Todo

write me

Breakpoints

Todo

write me

Busy status

Todo

write me

2.10.16 Cryptographic coprocessor

Contents

Todo

write me

Introduction

Todo

write me

IO registers

Todo

write me

Interrupts

Todo

write me

Submitting crypto commands: ccmd

Todo

write me

Code authentication control

Todo

write me

Crypto xfer control

Todo

write me

2.11 Video decoding, encoding, and processing

Contents:

2.11.1 VPE video decoding and encoding

Contents:

PMPEG: MPEG1/MPEG2 video decoding engine

Contents

Todo

write me

Introduction

Todo

write me

MMIO registers

Todo

write me

Interrupts

Todo

write me

PME: motion estimation

Contents:

PVP1: video processor

Contents:

Scalar unit

Contents

Introduction The scalar unit is one of the four execution units of VP1. It is used for general-purpose arithmetic.

Scalar registers The scalar unit has 31 GPRs, `$r0`–`$r30`. They are 32 bits wide, and are usually used as 32-bit integers, but there are also SIMD instructions treating them as arrays of 4 bytes. In such cases, array notation is used to denote the individual bytes. Bits 0-7 are considered to be `$rX[0]`, bits 8-15 are `$rX[1]` and so on. `$r31` is a special register hardwired to 0.

There are also 8 bits in each `$c` register belonging to the scalar unit. Most scalar instructions can (if requested) set these bits according to the computation result. The bits are:

- bit 0: sign flag - set equal to bit 31 of the result
- bit 1: zero flag - set if the result is 0
- bit 2: b19 flag - set equal to bit 19 of the result
- bit 3: b20 difference flag - set if bit 20 of the result is different from bit 20 of the first source
- bit 4: b20 flag - set equal to bit 20 of the result
- bit 5: b21 flag - set equal to bit 21 of the result

- bit 6: alt b19 flag (G80 only) - set equal to bit 19 of the result
- bit 7: b18 flag (G80 only) - set equal to bit 18 of the result

The purpose of the last 6 bits is so far unknown.

Scalar to vector data bus In addition to performing computations of its own, the scalar unit is also used in tandem with the vector unit to perform complex instructions. Certain scalar opcodes expose data on so-called s2v path (scalar to vector data bus), and certain vector opcodes consume this data.

The data is ephemeral and only exists during the execution of a single bundle - the producing and consuming instructions must be located in the same bundle. If a consuming instruction is used without a producing instruction, it'll read junk. If a producing instruction is used without a consuming instruction, the data is discarded.

The s2v data consists of:

- 4 signed 10-bits factors, used for multiplication
- `$vc` selection and transformation, for use as mask input in vector unit, made of:
  - valid flag: 1 if s2v data was emitted by proper s2v-emitting instruction (if false, vector unit will use an alternate source not involving s2v)
  - 2-bit `$vc` register index
  - 1-bit zero flag or sign flag selection (selects which half of `$vc` will be used)
  - 3-bit transform mode: used to mangle the `$vc` value before use as mask

The factors can alternatively be treated as two 16-bit masks by some instructions. In that case, mask 0 consists of bits 1-8 of factor 0, then bits 1-8 of factor 1 and mask 1 likewise consists of bits 1-8 of factors 2 and 3:

```
s2v.mask[0] = (s2v.factor[0] >> 1 & 0xff) | (s2v.factor[1] >> 1 & 0xff) << 8
s2v.mask[1] = (s2v.factor[2] >> 1 & 0xff) | (s2v.factor[3] >> 1 & 0xff) << 8
```

The `$vc` based mask is derived as follows:

```
def xfrm(val, tab):
    res = 0
    for idx in range(16):
        # bit x of result is set if bit tab[x] of input is set
        if val & 1 << tab[idx]:
            res |= 1 << idx
    return res

val = $vc[s2v.vcsel.idx]
# val2 is only used for transform mode 7
val2 = $vc[s2v.vcsel.idx | 1]

if s2v.vcsel.flag == 'sf':
    val = val & 0xffff
    val2 = val2 & 0xffff
else: # 'zf'
    val = val >> 16 & 0xffff
    val2 = val2 >> 16 & 0xffff

if s2v.vcsel.xfrm == 0:
    # passthrough
    s2v.vcmask = val
elif s2v.vcsel.xfrm == 1:
    s2v.vcmask = xfrm(val, [2,  2,  2,  2,  6,  6,  6,  6, 10, 10, 10, 10, 14, 14, 14, 14])
```

```
elif s2v.vcsel.xfrm == 2:
    s2v.vcmask = xfrm(val, [4,  5,  4,  5,  4,  5,  4,  5, 12, 13, 12, 13, 12, 13, 12, 13])
elif s2v.vcsel.xfrm == 3:
    s2v.vcmask = xfrm(val, [0,  0,  2,  0,  4,  4,  6,  4,  8,  8, 10,  8, 12, 12, 14, 12])
elif s2v.vcsel.xfrm == 4:
    s2v.vcmask = xfrm(val, [1,  1,  1,  3,  5,  5,  5,  7,  9,  9,  9, 11, 13, 13, 13, 15])
elif s2v.vcsel.xfrm == 5:
    s2v.vcmask = xfrm(val, [0,  0,  2,  2,  4,  4,  6,  6,  8,  8, 10, 10, 12, 12, 14, 14])
elif s2v.vcsel.xfrm == 6:
    s2v.vcmask = xfrm(val, [1,  1,  1,  1,  5,  5,  5,  5,  9,  9,  9,  9, 13, 13, 13, 13])
elif s2v.vcsel.xfrm == 7:
    # mode 7 is special: it uses two $vc inputs and takes every second bit
    s2v.vcmask = xfrm(val | val2 << 16, [0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,
```

Instruction format The instruction word fields used in scalar instructions are:

- bits 0-2: `CDST` - if < 4, index of the `$c` register to set according to the instruction's result. Otherwise, an indication that `$c` is not to be written (nVidia appears to use 7 in such case).
- bits 0-7: `BIMMBAD` - an immediate field used only in *bad opcodes*
- bits 0-18: `IMM19` - a signed 19-bit immediate field used only by the mov instruction
- bits 0-15: `IMM16` - a 16-bit immediate field used only by the sethi instruction
- bits 1-9: `FACTOR1` - a 9-bit signed immediate used as vector factor
- bits 10-18: `FACTOR2` - a 9-bit signed immediate used as vector factor
- bit 1: `SIGN2` - determines if byte multiplication source 2 is signed
  - 0: `u` - unsigned
  - 1: `s` - signed
- bit 2: `SIGN1` - likewise for source 1
- bits 3-10: `BIMM`: an 8-bit immediate for bytewise operations, signed or unsigned depending on instruction.
- bits 3-13: `IMM`: signed 13-bit immediate.
- bits 3-6: `BITOP`: selects the bit operation to perform
- bits 3-7: `RFILE`: selects the other register file for mov to/from other register file
- bits 3-4: `COND` - if source mangling is used, the `$c` register index to use for source mangling.
- bits 5-8: `SLCT` - if source mangling is used, the condition to use for source mangling.
- bit 8: `RND` - determines byte multiplication rounding behaviour
  - 0: `rd` - round down
  - 1: `rn` - round to nearest, ties rounding up
- btis 9-13: `SRC2` - the second source `$r` register, often mangled via source mangling.
- bits 9-13 (low 5 bits) and bit 0 (high bit): `BIMMMUL` - a 6-bit immediate for bytewise multiplication, signed or unsigned depending on instruction.
- bits 14-18: `SRC1` - the first source `$r` register.
- bits 19-23: `DST` - the destination `$r` register.
- bits 19-20: `VCIDX` - the `$vc` register index for s2v

- bit 21: VCFLAG - the $vc flag selection for s2v:
  - 0: sf
  - 1: zf
- bits 22-23 (low part) and 0 (high part): VCXFRM - the $vc transformation for s2v
- bits 24-31: OP - the opcode.

Opcodes The opcode range assigned to the scalar unit is 0x00–0x7f. The opcodes are:

- 0x01, 0x11, 0x21, 0x31: *bytewise multiplication: bmul*
- 0x02, 0x12, 0x22, 0x32: *bytewise multiplication: bmul (bad opcode)*
- 0x04: *s2v multiply/add/send: bvecmad*
- 0x24: *s2v immediate send: vec*
- 0x05: *s2v multiply/add/select/send: bvecmadsel*
- 0x25: *bytewise immediate and: band*
- 0x26: *bytewise immediate or: bor*
- 0x27: *bytewise immediate xor: bxor*
- 0x08, 0x18, 0x28, 0x38: *bytewise minimum: bmin*
- 0x09, 0x19, 0x29, 0x39: *bytewise maximum: bmax*
- 0x0a, 0x1a, 0x2a, 0x3a: *bytewise absolute value: babs*
- 0x0b, 0x1b, 0x2b, 0x3b: *bytewise negate: bneg*
- 0x0c, 0x1c, 0x2c, 0x3c: *bytewise addition: badd*
- 0x0d, 0x1d, 0x2d, 0x3d: *bytewise substract: bsub*
- 0x0e, 0x1e, 0x2e, 0x3e: *bytewise shift: bshr, bsar*
- 0x0f: *s2v send: bvec*
- 0x41, 0x51, 0x61, 0x71: *16-bit multiplication: mul*
- 0x42: *bitwise operation: bitop*
- 0x62: *immediate and: and*
- 0x63: *immediate xor: xor*
- 0x64: *immediate or: or*
- 0x45: *s2v 4-bit mask send and shift: vecms*
- 0x65: *load immediate: mov*
- 0x75: *set high bits immediate: sethi*
- 0x6a: *mov to other register file: mov*
- 0x6b: *mov from other register file: mov*
- 0x48, 0x58, 0x68, 0x78: *minimum: min*
- 0x49, 0x59, 0x69, 0x79: *maximum: max*
- 0x4a, 0x5a, 0x7a: *absolute value: abs*

- 0x4b, 0x5b, 0x7b: *negation: neg*
- 0x4c, 0x5c, 0x6c, 0x7c: *addition: add*
- 0x4d, 0x5d, 0x6d, 0x7d: *substraction: sub*
- 0x4e, 0x5e, 0x6e, 0x7e: *shift: shr, sar*
- 0x4f: the canonical scalar nop opcode

Todo

some unused opcodes clear $c, some don't

Bad opcodes Some of the VP1 instructions look like they're either buggy or just unintended artifacts of incomplete decoding hardware. These are known as bad opcodes and are characterised by using colliding bitfields. It's probably a bad idea to use them, but they do seem to reliably perform as documented here.

Source mangling Some instructions perform source mangling: the source register(s) they use are not taken directly from a register index bitfield in the instruction. Instead, the register index from the instruction is... "adjusted" before use. There are several algorithms used for source mangling, most of them used only in a single instruction.

The most common one, known as SRC2S, takes the register index from SRC2 field, a $c register index from COND, and $c bit index from SLCT. If SLCT is anything other than 4, the selected bit is extracted from $c and XORed into the lowest bit of the register index to use. Otherwise (SLCT is 4), bits 4-5 of $c are extracted, and added to bits 0-1 of the register index, discarding overflow out of bit 1:

```
if SLCT == 4:
    adjust = $c[COND] >> 4 & 3
    SRC2S = (SRC2 & ~3) | ((SRC2 + adjust) & 3)
else:
    adjust = $c[COND] >> SLCT & 1
    SRC2S = SRC2 ^ adjust
```

Instructions

Load immediate: mov Loads a 19-bit signed immediate to the selected register. If you need to load a const that doesn't fit into 19 signed bits, use this instruction along with *sethi*.

Instructions:

Instruction	Operands	Opcode
mov	$r[DST] IMM19	0x65

Operation:

```
    $r[DST] = IMM19
```

Set high bits: sethi Loads a 16-bit immediate to high bits of the selected register. Low 16 bits are unaffected.

Instructions:

Instruction	Operands	Opcode
sethi	$r[DST] IMM16	0x75

Operation:

```
    $r[DST] = ($r[DST] & 0xffff) | IMM16 << 16
```

Move to/from other register file: mov Does what it says on the tin. There is `$c` output capability, but it always outputs 0. The other register file is selected by `RFILE` field, and the possibilities are:

- 0: `$v` word 0 (ie. bytes 0-3)
- 1: `$v` word 1 (bytes 4-7)
- 2: `$v` word 2 (bytes 8-11)
- 3: `$v` word 3 (bytes 12-15)
- 4: ??? (NV41:G80 only)
- 5: ??? (NV41:G80 only)
- 6: ??? (NV41:G80 only)
- 7: ??? (NV41:G80 only)
- 8: `$sr`
- 9: `$mi`
- 10: `$uc`
- 11: `$l` (indices over 3 are ignored on writes, wrapped modulo 4 on reads)
- 12: `$a`
- 13: `$c` - read only (indices over 3 read as 0)
- 18: curiously enough, aliases 2, for writes only
- 20: `$m[0-31]`
- 21: `$m[32-63]`
- 22: `$d` (indices over 7 are wrapped modulo 8) (G80 only)
- 23: `$f` (indices over 1 are wrapped modulo 2)
- 24: `$x` (indices over 15 are wrapped modulo 16) (G80 only)

Todo

figure out the pre-G80 register files

Attempts to read or write unknown register file are ignored. In case of reads, the destination register is left unmodified.

Instructions:

Instruction	Operands	Opcode
mov	[$c[CDST]] $<RFILE>[DST] $r[SRC1]	0x6a
mov	[$c[CDST]] $r[DST] $<RFILE>[SRC1]	0x6b

Operation:

```
    if opcode == 0x6a:
        $<RFILE>[DST] = $r[SRC1]
    else:
        $r[DST] = $<RFILE>[SRC1]

    if CDST < 4:
        $c[CDST].scalar = 0
```

Arithmetic operations: mul, min, max, abs, neg, add, sub, shr, sar `mul` performs a 16x16 multiplication with 32 bit result. `shr` and `sar` do a bitwise shift right by given amount, with negative amounts interpreted as left shift (and the shift amount limitted to `-0x1f..0x1f`). The other operations do what it says on the tin. `abs`, `min`, `max`, `mul`, `sar` treat the inputs as signed, `shr` as unsigned, for others it doesn't matter.

The first source comes from a register selected by `SRC1`, and the second comes from either a register selected by mangled field `SRC2S` or a 13-bit signed immediate `IMM`. In case of `abs` and `neg`, the second source is unused, and the immediate versions are redundant (and in fact one set of opcodes is used for mov to/from other register file instead).

Most of these operations have duplicate opcodes. The canonical one is the lowest one.

All of these operations set the full set of scalar condition codes.

Instructions:

Instruction	Operands	Opcode
mul	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x41, 0x51
min	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x48, 0x58
max	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x49, 0x59
abs	[$c[CDST]] $r[DST] $r[SRC1]	0x4a, 0x5a, 0x7a
neg	[$c[CDST]] $r[DST] $r[SRC1]	0x4b, 0x5b, 0x7b
add	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x4c, 0x5c
sub	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x4d, 0x5d
sar	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x4e
shr	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x5e
mul	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x61, 0x71
min	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x68, 0x78
max	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x69, 0x79
add	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x6c, 0x7c
sub	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x6d, 0x7d
sar	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x6e
shr	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x7e

Operation:

```
s1 = sext($r[SRC1], 31)
if opcode & 0x20:
    s2 = sext(IMM, 12)
else:
    s2 = sext($r[SRC2], 31)

if op == 'mul':
    res = sext(s1, 15) * sext(s2, 15)
elif op == 'min':
    res = min(s1, s2)
elif op == 'max':
    res = max(s1, s2)
elif op == 'abs':
    res = abs(s1)
elif op == 'neg':
    res = -s1
elif op == 'add':
    res = s1 + s2
elif op == 'sub':
    res = s1 - s2
elif op == 'shr' or op == 'sar':
    # shr/sar are unsigned/signed versions of the same insn
    if op == 'shr':
        s1 &= 0xffffffff
    # shift amount is 6-bit signed number
```

```
        shift = sext(s2, 5)
        # and -0x20 is invalid
        if shift == -0x20:
            shift = 0
        # negative shifts mean a left shift
        if shift < 0:
            res = s1 << -shift
        else:
            # sign of s1 matters here
            res = s1 >> shift

    $r[DST] = res
    # build $c result
    cres = 0
    if res & 1 << 31:
        cres |= 1
    if res == 0:
        cres |= 2
    if res & 1 << 19:
        cres |= 4
    if (res ^ s1) & 1 << 20:
        cres |= 8
    if res & 1 << 20:
        cres |= 0x10
    if res & 1 << 21:
        cres |= 0x20
    if variant == 'G80':
        if res & 1 << 19:
            cres |= 0x40
        if res & 1 << 18:
            cres |= 0x80
    if CDST < 4:
        $c[CDST].scalar = cres
```

Bit operations: bitop Performs an *arbitrary two-input bit operation* on two registers, selected by `SRC1` and `SRC2`. `$c` output works, but only with a subset of flags.

Instructions:

Instruction	Operands	Opcode
`bitop`	`BITOP [$c[CDST]] $r[DST] $r[SRC1] $r[SRC2]`	`0x42`

Operation:

```
    s1 = $r[SRC1]
    s2 = $r[SRC2]

    res = bitop(BITOP, s2, s1) & 0xffffffff

    $r[DST] = res
    # build $c result
    cres = 0
    # bit 0 not set
    if res == 0:
        cres |= 2
    if res & 1 << 19:
        cres |= 4
    # bit 3 not set
    if res & 1 << 20:
        cres |= 0x10
```

```
    if res & 1 << 21:
        cres |= 0x20
    if variant == 'G80':
        if res & 1 << 19:
            cres |= 0x40
        if res & 1 << 18:
            cres |= 0x80
    if CDST < 4:
        $c[CDST].scalar = cres
```

Bit operations with immediate: and, or, xor Performs a given bitwise operation on a register and 13-bit immediate. Like for *bitop*, `$c` output only works partially.

Instructions:

Instruction	Operands	Opcode
and	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x62
xor	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x63
or	[$c[CDST]] $r[DST] $r[SRC1] IMM	0x64

Operation:

```
    s1 = $r[SRC1]

    if op == 'and':
        res = s1 & IMM
    elif op == 'xor':
        res = s1 ^ IMM
    elif op == 'or':
        res = s1 | IMM

    $r[DST] = res
    # build $c result
    cres = 0
    # bit 0 not set
    if res == 0:
        cres |= 2
    if res & 1 << 19:
        cres |= 4
    # bit 3 not set
    if res & 1 << 20:
        cres |= 0x10
    if res & 1 << 21:
        cres |= 0x20
    if variant == 'G80':
        if res & 1 << 19:
            cres |= 0x40
        if res & 1 << 18:
            cres |= 0x80
    if CDST < 4:
        $c[CDST].scalar = cres
```

Simple bytewise operations: bmin, bmax, babs, bneg, badd, bsub Those perform the corresponding operation (minumum, maximum, absolute value, negation, addition, substraction) in SIMD manner on 8-bit signed or unsigned numbers from one or two sources. Source 1 is always a register selected by `SRC1` bitfield. Source 2, if it is used (ie. instruction is not `babs` nor `bneg`), is either a register selected by `SRC2S` mangled bitfield, or immediate taken from `BIMM` bitfield.

Each of these instructions comes in signed and unsigned variants and both perform result clipping. Note that abs is rather uninteresting in its unsigned variant (it's just the identity function), and so is neg (result is always 0 or clipped to 0.

These instruction have a `$c` output, but it's always set to all-0 if used.

Also note that `babs` and `bneg` have two redundant opcodes each: the bit that normally selects immediate or register second source doesn't apply to them.

Instructions:

Instruction	Operands	Opcode
bmin s	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x08
bmax s	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x09
babs s	[$c[CDST]] $r[DST] $r[SRC1]	0x0a
bneg s	[$c[CDST]] $r[DST] $r[SRC1]	0x0b
badd s	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x0c
bsub s	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x0d
bmin u	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x18
bmax u	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x19
babs u	[$c[CDST]] $r[DST] $r[SRC1]	0x1a
bneg u	[$c[CDST]] $r[DST] $r[SRC1]	0x1b
badd u	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x1c
bsub u	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x1d
bmin s	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x28
bmax s	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x29
babs s	[$c[CDST]] $r[DST] $r[SRC1]	0x2a
bneg s	[$c[CDST]] $r[DST] $r[SRC1]	0x2b
badd s	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x2c
bsub s	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x2d
bmin u	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x38
bmax u	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x39
babs u	[$c[CDST]] $r[DST] $r[SRC1]	0x3a
bneg u	[$c[CDST]] $r[DST] $r[SRC1]	0x3b
badd u	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x3c
bsub u	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x3d

Operation:

```
    for idx in range(4):
        s1 = $r[SRC1][idx]
        if opcode & 0x20:
            s2 = BIMM
        else:
            s2 = $r[SRC2S][idx]

        if opcode & 0x10:
            # unsigned
            s1 &= 0xff
            s2 &= 0xff
        else:
            # signed
            s1 = sext(s1, 7)
            s2 = sext(s2, 7)

        if op == 'bmin':
            res = min(s1, s2)
        elif op == 'bmax':
            res = max(s1, s2)
```

```
        elif op == 'babs':
            res = abs(s1)
        elif op == 'bneg':
            res = -s1
        elif op == 'badd':
            res = s1 + s2
        elif op == 'bsub':
            res = s1 - s2

        if opcode & 0x10:
            # unsigned: clip to 0..0xff
            if res < 0:
                res = 0
            if res > 0xff:
                res = 0xff
        else:
            # signed: clip to -0x80..0x7f
            if res < -0x80:
                res = -0x80
            if res > 0x7f:
                res = 0x7f

        $r[DST][idx] = res

    if CDST < 4:
        $c[CDST].scalar = 0
```

Bytewise bit operations: band, bor, bxor Performs a given bitwise operation on a register and an 8-bit immediate replicated 4 times. Or, intepreted differently, performs such operation on every byte of a register idependently. `$c` output is present, but always outputs 0.

Instructions:

Instruction	Operands	Opcode
and	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x25
or	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x26
xor	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x27

Operation:

```
    for idx in range(4):
        if op == 'and':
            $r[DST][idx] = $r[SRC1][idx] & BIMM
        elif op == 'or':
            $r[DST][idx] = $r[SRC1][idx] | BIMM
        elif op == 'xor':
            $r[DST][idx] = $r[SRC1][idx] ^ BIMM

    if CDST < 4:
        $c[CDST].scalar = 0
```

Bytewise bit shift operations: bshr, bsar Performs a bytewise SIMD right shift. Like the usual shift instruction, the shift amount is considered signed and negative amounts result in left shift. In this case, the shift amount is a 4-bit signed number. Operands are as in usual *bytewise operations*.

Instructions:

Instruction	Operands	Opcode
bsar	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x0e
bshr	[$c[CDST]] $r[DST] $r[SRC1] $r[SRC2S]	0x1e
bsar	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x2e
bshr	[$c[CDST]] $r[DST] $r[SRC1] BIMM	0x3e

Operation:

```
for idx in range(4):
    s1 = $r[SRC1][idx]
    if opcode & 0x20:
        s2 = BIMM
    else:
        s2 = $r[SRC2S][idx]

    if opcode & 0x10:
        # unsigned
        s1 &= 0xff
    else:
        # signed
        s1 = sext(s1, 7)

    shift = sext(s2, 3)

    if shift < 0:
        res = s1 << -shift
    else:
        res = s1 >> shift

    $r[DST][idx] = res

if CDST < 4:
    $c[CDST].scalar = 0
```

Bytewise multiplication: bmul These instructions perform bytewise fractional multiplication: the inputs and outputs are considered to be fixed-point numbers with 8 fractional bits (unsigned version) or 7 fractional bits (signed version). The signedness of both inputs and the output can be controlled independently (the signedness of the output is controlled by the opcode, and of the inputs by instruction word flags `SIGN1` and `SIGN2`). The results are clipped to the output range. There are two rounding modes: round down and round to nearest with ties rounded up.

The first source is always a register selected by `SRC1` bitfield. The second source can be a register selected by `SRC2` bitfield, or 6-bit immediate in `BIMMMUL` bitfield padded with two zero bits on the right.

Note that besides proper `0xX1` opcodes, there are also `0xX2` *bad opcodes*. In case of register-register ops, these opcodes are just aliases of the sane ones, but for immediate opcodes, a colliding bitfield is used.

The instructions have no `$c` output capability.

Instructions:

Instruction	Operands	Opcode
bmul s	RND $r[DST] SIGN1 $r[SRC1] SIGN2 $r[SRC2]	0x01, 0x02
bmul u	RND $r[DST] SIGN1 $r[SRC1] SIGN2 $r[SRC2]	0x11, 0x12
bmul s	RND $r[DST] SIGN1 $r[SRC1] SIGN2 BIMMMUL	0x21
bmul u	RND $r[DST] SIGN1 $r[SRC1] SIGN2 BIMMMUL	0x31
bmul s	RND $r[DST] SIGN1 $r[SRC1] SIGN2 BIMMBAD	0x22 (bad opcode)
bmul u	RND $r[DST] SIGN1 $r[SRC1] SIGN2 BIMMBAD	0x32 (bad opcode)

Operation:

```
    for idx in range(4):
        # read inputs
        s1 = $r[SRC1][idx]
        if opcode & 0x20:
            if opcode & 2:
                s2 = BIMMBAD
            else:
                s2 = BIMMMUL << 2
        else:
            s2 = $r[SRC2S][idx]

        # convert inputs to 8 fractional bits - unsigned inputs are already ok
        if SIGN1:
            ss1 = sext(ss1, 7) << 1
        if SIGN2:
            ss2 = sext(ss2, 7) << 1

        # multiply - the result has 16 fractional bits
        res = ss1 * ss2

        if opcode & 0x10:
            # unsigned result
            # first, if round to nearest is selected, apply rounding correction
            if RND == 'rn':
                res += 0x80
            # convert to 8 fractional bits
            res >>= 8
            # clip
            if res < 0:
                res = 0
            if res > 0xff:
                res = 0xff
        else:
            # signed result
            if RND == 'rn':
                res += 0x100
            # convert to 7 fractional bits
            res >>= 9
            # clip
            if res < -0x80:
                res = -0x80
            if res > 0x7f:
                res = 0x7f

        $r[DST][idx] = res
```

Send immediate to vector unit: vec This instruction takes two 9-bit immediate operands and sends them as factors to the vector unit. The first immediate is used as factors 0 and 1, and the second is used as factors 2 and 3. `$vc` selection is sent as well.

Instructions:

Instruction	Operands	Opcode
vec	FACTOR1 FACTOR2 $vc[VCIDX] VCFLAG VCXFRM	0x24

Operation:

```
    s2v.factor[0] = s2v.factor[1] = FACTOR1
    s2v.factor[2] = s2v.factor[3] = FACTOR2
    s2v.vcsel.idx = VCIDX
```

```
    s2v.vcsel.flag = VCFLAG
    s2v.vcsel.xfrm = VCXFRM
```

Send mask to vector unit and shift: vecms This instruction shifts a register right by 4 bits and uses the bits shifted out as s2v mask 0 after expansion (each bit is replicated 4 times). The s2v factors are derived from that mask and are not very useful. The right shift is sign-filling. `$vc` selection is sent as well.

Instructions:

Instruction	Operands	Opcode
vecms	$r[SRC1] $vc[VCIDX] VCFLAG VCXFRM	0x45

Operation:

```
    val = sext($r[SRC1], 31)
    $r[SRC1] = val >> 4
    # the factors are made so that the mask derived from them will contain
    # each bit from the short mask repeated 4 times
    f0 = 0
    f1 = 0
    if val & 1:
        f0 |= 0x1e
    if val & 2:
        f0 |= 0x1e0
    if val & 4:
        f1 |= 0x1e
    if val & 8:
        f1 |= 0x1e0
    s2v.factor[0] = f0
    s2v.factor[1] = f1
    s2v.factor[2] = s2v.factor[3] = 0
    s2v.vcsel.idx = VCIDX
    s2v.vcsel.flag = VCFLAG
    s2v.vcsel.xfrm = VCXFRM
```

Send bytes to vector unit: bvec Treats a register as 4-byte vector, sends the bytes as s2v factors (treating them as signed with 7 fractional bits). `$vc` selection is sent as well. If the s2v output is used as masks, this effectively takes mask 0 from source bits 0-15 and mask 1 from source bits 16-31.

Instructions:

Instruction	Operands	Opcode
bvec	$r[SRC1] $vc[VCIDX] VCFLAG VCXFRM	0x0f

Operation:

```
    for idx in range(4):
        s2v.factor[idx] = sext($r[SRC1][idx], 7) << 1
    s2v.vcsel.idx = VCIDX
    s2v.vcsel.flag = VCFLAG
    s2v.vcsel.xfrm = VCXFRM
```

Bytewise multiply, add, and send to vector unit: bvecmad, bvecmadsel Figure out this one yourself. It sends s2v factors based on SIMD multiply & add, uses weird source mangling, and even weirder source 1 bitfields.

Instructions:

Instruction	Operands	Opcode
bvecmad	$r[SRC1] $r[SRC2]q $vc[VCIDX] VCFLAG VCXFRM	0x04
bvecmadsel	$r[SRC1] $r[SRC2]q $vc[VCIDX] VCFLAG VCXFRM	0x05

Operation:

```
    if SLCT== 4:
            adjust = $c[COND] >> 4 & 3
    else:
            adjust = $c[COND] >> SLCT & 1

    # SRC1 selects the pre-factor, which will be multiplied by source 3
    if op == 'bvecmad':
        prefactor = $r[SRC1] >> 11 & 0xff
    elif op == 'bvecmadsel':
        prefactor = $r[SRC1] >> 11 & 0x7f

    s2a = $r[SRC2 | adjust]
    s2b = $r[SRC2 | 2 | adjust]

    for idx in range(4):
        # this time source is mangled by OR, not XOR - don't ask me

        if op == 'bvecmad'
            midx = idx
        elif op == 'bvecmadsel':
            midx = idx & 2
            if SLCT == 2 and $c[COND] & 0x80:
                midx |= 1

        # baseline (res will have 16 fractional bits, sources have 8)
        res = s2a[midx] << 8
        # throw in the multiplication result
        res += prefactor * s2b[idx]
        # and rounding correction (for round to nearest, ties up)
        res += 0x40
        # and round to 9 fractional bits
        s2v.factor[idx] = res >> 7

    s2v.vcsel.idx = VCIDX
    s2v.vcsel.flag = VCFLAG
    s2v.vcsel.xfrm = VCXFRM
```

Vector unit

Contents

Introduction The vector unit is one of the four execution units of VP1. It operates in SIMD manner on 16-element vectors.

Vector registers The vector unit has 32 vector registers, `$v0-$v31`. They are 128 bits wide and are treated as 16 components of 8 bits each. Depending on element, they can be treated as signed or unsigned.

There are also 4 vector condition code registers, `$vc0-$vc3`. They are like `$c` for vector registers - each of them has 16 "sign flag" and 16 "zero flag" bits, one of each per vector component. When read as a 32-word, bits 0-15 are the sign flags and bits 16-31 are the zero flags.

Further, the vector unit has a singular 448-bit vector accumulator register, `$va`. It is made of 16 components, each of them a 28-bit signed number with 16 fractional bits. It's used to store intermediate unrounded results of multiply-add computations.

Finally, there's an extra 128-bit register, `$vx`, which works quite like the usual `$v` registers. It's only read by *vlrp4b* instructions and written only by special *load to vector extra register* instructions. The reasons for its existence are unclear.

Instruction format The instruction word fields used in vector instructions in addition to *the ones used in scalar instructions* are:

- bit 0: `S2VMODE` - selects how s2v data is used:
  - 0: `factor` - s2v data is interpreted as factors

  - 1: `mask` - s2v data is interpreted as masks
- bits 0-2: `VCDST` - if < 4, index of `$vc` register to set according to the instruction's results. Otherwise, an indication that `$vc` is not to be written (the canonical value for such case appears to be 7).
- bits 0-1: `VCSRC` - selects `$vc` input for `vlrp2`
- bit 2: `VCSEL` - the `$vc` flag selection for `vlrp2`:
  - 0: `sf`
  - 1: `zf`
- bit 3: `SWZLOHI` - selects how the swizzle selectors are decoded:
  - 0: `lo` - bits 0-3 are component selector, bit 4 is source selector
  - 1: `hi` - bits 4-7 are component selector, bit 0 is source selector
- bit 3: `FRACTINT` - selects whether the multiplication is considered to be integer or fixed-point:
  - 0: `fract`: fixed-point
  - 1: `int`: integer
- bit 4: `HILO` - selects which part of multiplication result to read:
  - 0: `hi`: high part
  - 1: `lo`: low part
- bits 5-7: `SHIFT` - a 3-bit signed immediate, used as an extra right shift factor
- bits 4-8: `SRC3` - the third source `$v` register.
- bit 9: `ALTRND` - like `RND`, but for different instructions
- bit 9: `SIGNS` - determines if double-interpolation input is signed
  - 0: `u` - unsigned
  - 1: `s` - signed
- bit 10: `LRP2X` - determines if base input is XORed with 0x80 for `vlrp2`.
- bit 11: `VAWRITE` - determines if `$va` is written for `vlrp2`.
- bits 11-13: `ALTSHIFT` - a 3-bit signed immediate, used as an extra right shift factor
- bit 12: `SIGND` - determines if double-interpolation output is signed
  - 0: `u` - unsigned
  - 1: `s` - signed
- bits 19-22: `CMPOP`: selects the bit operation to perform on comparison result and previous flag value

Opcodes The opcode range assigned to the vector unit is `0x80-0xbf`. The opcodes are:

- `0x80`, `0xa0`, `0xb0`, `0x81`, `0x91`, `0xa1`, `0xb1`: *multiplication: vmul*
- `0x90`: *linear interpolation: vlrp*
- `0x82`, `0x92`, `0xa2`, `0xb2`, `0x83`, `0x93`, `0xa3`: *multiplication with accumulation: vmac*
- `0x84`, `0x85`, `0x95`: *dual multiplication with accumulation: vmac2*
- `0x86`, `0x87`, `0x97`: *dual multiplication with addition: vmad2*
- `0x96`, `0xa6`, `0xa7`: *dual multiplication with addition: vmad2 (bad opcode)*

- `0x94`: *bitwise operation: vbitop*
- `0xa4`: *clip to range: vclip*
- `0xa5`: *minimum of absolute values: vminabs*
- `0xb3`: *dual linear interpolation: vlrp2*
- `0xb4`: *quad linear interpolation, part 1: vlrp4a*
- `0xb5`: *factor linear interpolation: vlrpf*
- `0xb6`, `0xb7`: *quad linear interpolation, part 2: vlrp4b*
- `0x88`, `0x98`, `0xa8`, `0xb8`: *minimum: vmin*
- `0x89`, `0x99`, `0xa9`, `0xb9`: *maximum: vmax*
- `0x8a`, `0x9a`: *absolute value: vabs*
- `0xaa`: *immediate and: vand*
- `0xba`: *move: mov*
- `0x8b`: *negation: vneg*
- `0x9b`: *swizzle: vswz*
- `0xab`: *immediate xor: vxor*
- `0xbb`: *move from $vc: mov*
- `0x8c`, `0x9c`, `0xac`, `0xbc`: *addition: vadd*
- `0x8d`, `0x9d`, `0xbd`: *substraction: vsub*
- `0xad`: *move immediate: vmov*
- `0x8e`, `0x9e`, `0xae`, `0xbe`: *shift: vshr, vsar*
- `0x8f`: *compare with absolute difference: vcmpad*
- `0x9f`: *add 9-bit: vadd9*
- `0xaf`: *immediate or: vor*
- `0xbf`: the canonical vector nop opcode

Multiplication, accumulation, and rounding The most advanced vector instructions involve multiplication and the vector accumulator. The vector unit has two multipliers (signed 10-bit * 10-bit -> signed 20-bit) and three wide adders (performing 28-bit addition): the first two add the multiplication results, and the third adds a rounding correction. In other words, it can compute A + (B * C << S) + (D * E << S) + R, where A is 28-bit input, B, C, D, E are signed 10-bit inputs, S is either 0 or 8, and R is the rounding correction, determined from the readout parameters. The B, C, D, E inputs can in turn be computed from other inputs using one of the narrower ALUs.

The A input can come from the vector accumulator, be fixed to 0, or come from a vector register component shifted by some shift amount. The shift amount, if used, is the inverse of the shift amount used by the readout process.

There are three things that can happen to the result of the multiply-accumulate calculations:

- written in its entirety to the vector accumulator
- shifted, rounded, clipped, and written to a vector register
- both of the above

The vector register readout process takes the following parameters:

- sign: whether the result should be unsigned or signed
- fract/int selection: if int, the multiplication is considered to be done on integers, and the 16-bit result is at bits 8-23 of the value added to the accumulator (ie. S is 8). Otherwise, the multiplication is performed as if the inputs were fractions (unsigned with 8 fractional bits, signed with 7), and the results are aligned so that bits 16-27 of the accumulator are integer part, and 0-15 are fractional part.
- hi/lo selection: selects whether high or low 8 bits of the results are read. For integers, the result is treated as 16-bit integer. For fractions, the high part is either an unsigned fixed-point number with 8 fractional bits, or a signed number with 7 fractional bits, and the low part is always 8 bits lower than the high part.
- a right shift, in range of -4..3: the result is shifted right by that amount before readout (as usual, negative means left shift).
- rounding mode: either round down, or round to nearest. If round to nearest is selected, a configuration bit in `$uccfg` register selects if ties are rounded up or down (to accomodate video codecs which switch that on frame basis).

First, any inputs from vector registers are read, converted as signed or unsigned integers, and normalized if needed:

```
def mad_input(val, fractint, isign):
    if isign == 'u':
        return val & 0xff
    else:
        if fractint == 'int':
            return sext(val, 7)
        else:
            return sext(val, 7) << 1
```

The readout shift factor is determined as follows:

```
def mad_shift(fractint, sign, shift):
    if fractint == 'int':
        return 16 - shift
    elif sign == 'u':
        return 8 - shift
    elif sign == 's':
        return 9 - shift
```

If A is taken from a vector register, it's expanded as follows:

```
def mad_expand(val, fractint, sign, shift):
    return val << mad_shift(fractint, sign, shift)
```

The actual multiply-add process works like that:

```
def mad(a, b, c, d, e, rnd, fractint, sign, shift, hilo):
    res = a

    if fractint == 'fract':
        res += b * c + d * e
    else:
        res += (b * c + d * e) << 8

    # rounding correction
    if rnd == 'rn':

        # determine the final readout shift
        if hilo == 'lo':
            rshift = mad_shift(fractint, sign, shift) - 8
        else:
```

```
            rshift = mad_shift(fractint, sign, shift)

        # only add rounding correction if there's going to be an actual
        # right shift
        if rshift > 0:
            res += 1 << (rshift - 1)
            if $uccfg.tiernd == 'down':
                res -= 1

    # the accumulator is only 28 bits long, and it wraps
    return sext(res, 27)
```

And the readout process is:

```
def mad_read(val, fractint, sign, shift, hilo):
    # first, shift it to the position
    rshift = mad_shift(fractint, sign, shift) - 8
    if rshift >= 0:
        res = val >> rshift
    else:
        res = val << -rshift

    # second, clip to 16-bit signed or unsigned
    if sign == 'u':
        if res < 0:
            res = 0
        if res > 0xffff:
            res = 0xffff
    else:
        if res < -0x8000:
            res = -0x8000
        if res > 0x7fff:
            res = 0x7fff

    # finally, extract high/low part of the final result
    if hilo == 'hi':
        return res >> 8 & 0xff
    else:
        return res & 0xff
```

Note that high/low selection, apart from actual result readout, also affects the rounding computation. This means that, if rounding is desired and the full 16-bit result is to be read, the low part should be read first with rounding (which will add the rounding correction to the accumulator) and then the high part should be read without rounding (since the rounding correction is already applied).

Instructions

Move: mov Copies one register to another. `$vc` output supported for zero flag only.

Instructions:

Instruction	Operands	Opcode
mov	[$vc[VCDST]] $v[DST] $v[SRC1]	0xba

Operation:

```
    for idx in range(16):
        $v[DST][idx] = $v[SRC1][idx]
        if VCDST < 4:
```

```
            $vc[VCDST].sf[idx] = 0
            $vc[VCDST].zf[idx] = $v[DST][idx] == 0
```

Move immediate: vmov Loads an 8-bit immediate to each component of destination. `$vc` output is fully supported, with sign flag set to bit 7 of the value.

Instructions:

Instruction	Operands	Opcode
vmov	[$vc[VCDST]] $v[DST] BIMM	0xad

Operation:

```
    for idx in range(16):
        $v[DST][idx] = BIMM
        if VCDST < 4:
            $vc[VCDST].sf[idx] = BIMM >> 7 & 1
            $vc[VCDST].zf[idx] = BIMM == 0
```

Move from $vc: mov Reads the contents of all `$vc` registers to a selected vector register. Bytes 0-3 correspond to `$vc0`, bytes 4-7 to `$vc1`, and so on. The sign flags are in bytes 0-1, and the zero flags are in bytes 2-3.

Instructions:

Instruction	Operands	Opcode
mov	$v[DST] $vc	0xbb

Operation:

```
    for idx in range(4):
        $v[DST][idx * 4] = $vc[idx].sf & 0xff;
        $v[DST][idx * 4 + 1] = $vc[idx].sf >> 8 & 0xff;
        $v[DST][idx * 4 + 2] = $vc[idx].zf & 0xff;
        $v[DST][idx * 4 + 3] = $vc[idx].zf >> 8 & 0xff;
```

Swizzle: vswz Performs a swizzle, also known as a shuffle: builds a result vector from arbitrarily selected components of two input vectors. There are three source vectors: sources 1 and 2 supply the data to be used, while source 3 selects the mapping of output vector components to input vector components. Each component of source 3 consists of source selector and component selector. They select the source (1 or 2) and its component that will be used as the corresponding component of the result.

Instructions:

Instruction	Operands	Opcode
vswz	SWZLOHI $v[DST] $v[SRC1] $v[SRC2] $v[SRC3]	0x9b

Operation:

```
    for idx in range(16):
        # read the component and source selectors
        if SWZLOHI == 'lo':
            comp = $v[SRC3][idx] & 0xf
            src = $v[SRC3][idx] >> 4 & 1
        else:
            comp = $v[SRC3][idx] >> 4 & 0xf
            src = $v[SRC3][idx] & 1

        # read the source & component
        if src == 0:
            $v[DST][idx] = $v[SRC1][comp]
        else:
            $v[DST][idx] = $v[SRC2][comp]
```

Simple arithmetic operations: vmin, vmax, vabs, vneg, vadd, vsub Those perform the corresponding operation (minumum, maximum, absolute value, negation, addition, substraction) in SIMD manner on 8-bit signed or unsigned numbers from one or two sources. Source 1 is always a register selected by `SRC1` bitfield. Source 2, if it is used (ie. instruction is not `vabs` nor `vneg`), is either a register selected by `SRC2` bitfield, or immediate taken from `BIMM` bitfield.

Most of these instructions come in signed and unsigned variants and both perform result clipping. The exception is `vneg`, which only has a signed version. Note that `vabs` is rather uninteresting in its unsigned variant (it's just the identity function). Note that `vsub` lacks a signed version with immediat: it can be replaced with `vadd` with negated immediate.

`$vc` output is fully supported. For signed variants, the sign flag output is the sign of the result. For unsigned variants, the sign flag is used as an overflow flag: it's set if the true unclipped result is not in `0..0xff` range.

Instructions:

Instruction	Operands	Opcode
vmin s	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x88
vmax s	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x89
vabs s	[$vc[VCDST]] $v[DST] $v[SRC1]	0x8a
vneg s	[$vc[VCDST]] $v[DST] $v[SRC1]	0x8b
vadd s	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x8c
vsub s	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x8d
vmin u	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x98
vmax u	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x99
vabs u	[$vc[VCDST]] $v[DST] $v[SRC1]	0x9a
vadd u	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x9c
vsub u	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x9d
vmin s	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xa8
vmax s	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xa9
vadd s	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xac
vmin u	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xb8
vmax u	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xb9
vadd u	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xbc
vsub u	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xbd

Operation:

```
    for idx in range(16):
        s1 = $v[SRC1][idx]
        if opcode & 0x20:
            s2 = BIMM
        else:
            s2 = $v[SRC2][idx]

        if opcode & 0x10:
            # unsigned
            s1 &= 0xff
            s2 &= 0xff
        else:
            # signed
            s1 = sext(s1, 7)
            s2 = sext(s2, 7)

        if op == 'vmin':
            res = min(s1, s2)
        elif op == 'vmax':
            res = max(s1, s2)
        elif op == 'vabs':
```

```
            res = abs(s1)
        elif op == 'vneg':
            res = -s1
        elif op == 'vadd':
            res = s1 + s2
        elif op == 'vsub':
            res = s1 - s2

        sf = 0
        if opcode & 0x10:
            # unsigned: clip to 0..0xff
            if res < 0:
                res = 0
                sf = 1
            if res > 0xff:
                res = 0xff
                sf = 1
        else:
            # signed: clip to -0x80..0x7f
            if res < 0:
                sf = 1
            if res < -0x80:
                res = -0x80
            if res > 0x7f:
                res = 0x7f

        $v[DST][idx] = res

        if VCDST < 4:
            $vc[VCDST].sf[idx] = sf
            $vc[VCDST].zf[idx] = res == 0
```

Clip to range: vclip Performs a SIMD range clipping operation: first source is the value to clip, second and third sources are the range endpoints. Or, equivalently, calculates the median of three inputs. `$vc` output is supported, with the sign flag set if clipping was performed (value equal to range endpoint is considered to be clipped) or the range is improper (second endpoint not larger than the first). All inputs are treated as signed.

Instructions:

Instruction	Operands	Opcode
vclip	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2] $v[SRC3]	0xa4

Operation:

```
    for idx in range(16):
        s1 = sext($v[SRC1][idx], 7)
        s2 = sext($v[SRC2][idx], 7)
        s3 = sext($v[SRC3][idx], 7)

        sf = 0

        # determine endpoints
        if s2 < s3:
            # proper order
            start = s2
            end = s3
        else:
            # reverse order
            start = s3
            end = s2
```

```
            sf = 1

        # and clip
        res = s1
        if res <= start:
            res = start
            sf = 1
        if res >= end:
            res = end
            sf = 1

        $v[DST][idx] = res

        if VCDST < 4:
            $vc[VCDST].sf[idx] = sf
            $vc[VCDST].zf[idx] = res == 0
```

Minimum of absolute values: vminabs Performs `min(abs(a), abs(b))`. Both inputs are treated as signed. `$vc` output is supported for zero flag only. The result is clipped to `0..0x7f` range (which only matters if both inputs are `-0x80`).

Instructions:

Instruction	Operands	Opcode
vminabs	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0xa5

Operation:

```
    for idx in range(16):
        s1 = sext($v[SRC1][idx], 7)
        s2 = sext($v[SRC2][idx], 7)

        res = min(abs(s1, s2))

        if res > 0x7f:
            res = 0x7f

        $v[DST][idx] = res

        if VCDST < 4:
            $vc[VCDST].sf[idx] = 0
            $vc[VCDST].zf[idx] = res == 0
```

Add 9-bit: vadd9 Performs an 8-bit unsigned + 9-bit signed addition (ie. exactly what's needed for motion compensation). The first source provides the 8-bit inputs, while the second and third are uniquely treated as vectors of 8 16-bit components (of which only low 9 are actually used). Second source provides components 0-7, and third provides 8-15. The result is unsigned and clipped. `$vc` output is supported, with sign flag set to 1 if the true result was out of 8-bit unsigned range.

Instructions:

Instruction	Operands	Opcode
vadd9	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2] $v[SRC3]	0x9f

Operation:

```
    for idx in range(16):
        # read source 1
        s1 = $v[SRC1][idx]

        if idx < 8:
```

```
            # 0-7: SRC2
            s2l = $v[SRC2][idx * 2]
            s2h = $v[SRC2][idx * 2 + 1]
        else:
            # 8-15: SRC3
            s2l = $v[SRC3][(idx - 8) * 2]
            s2h = $v[SRC3][(idx - 8) * 2 + 1]

        # read as 9-bit signed number
        s2 = sext(s2h << 8 | s2l, 8)

        # add
        res = s1 + s2

        # clip
        sf = 0
        if res > 0xff:
            sf = 1
            res = 0xff
        if res < 0:
            sf = 1
            res = 0

        $v[DST][idx] = res

        if VCDST < 4:
            $vc[VCDST].sf[idx] = sf
            $vc[VCDST].zf[idx] = res == 0
```

Compare with absolute difference: vcmpad This instruction performs the following operations:

- substract source 1.1 from source 2
- take the absolute value of the difference
- compare the result with source 1.2
- if equal, set zero flag of selected `$vc` output
- set sign flag of `$vc` output to *an arbitrary bitwise operation* of s2v `$vc` input and "less than" comparison result

All inputs are treated as unsigned. If s2v scalar instruction is not used together with this instruction, `$vc` input defaults to sign flag of the `$vc` register selected as output, with no transformation.

This instruction has two sources: source 1 is a register pair, while source 2 is a single register. The second register of the pair is selected by ORing 1 to the index of the first register of the pair. Source 2 is selected by mangled field `SRC2S`.

Instructions:

Instruction	Operands	Opcode
`vcmppad`	`CMPOP [$vc[VCDST]] $v[SRC1]d $v[SRC2S]`	`0x8f`

Operation:

```
    if s2v.vcsel.valid:
        vcin = s2v.vcmask
    else:
        vcin = $vc[VCDST & 3].sf

    for idx in range(16):
        ad = abs($v[SRC2S][idx] - $v[SRC1][idx])
```

```
        other = $v[SRC1 | 1][idx]

        if VCDST < 4:
            $vc[VCDST].sf[idx] = sf
            $vc[VCDST].zf[idx] = ad == bitop(CMPOP, vcin >> idx & 1, ad < other)
```

Bit operations: vbitop Performs an *arbitrary two-input bit operation* on two registers. `$vc` output supported for zero flag only.

Instructions:

Instruction	Operands	Opcode
vbitop	BITOP [$vc[CDST]] $v[DST] $v[SRC1] $v[SRC2]	0x94

Operation:

```
    for idx in range(16):
        s1 = $v[SRC1][idx]
        s2 = $v[SRC2][idx]

        res = bitop(BITOP, s2, s1) & 0xff

        $v[DST][idx] = res
        if VCDST < 4:
            $vc[VCDST].sf[idx] = 0
            $vc[VCDST].zf[idx] = res == 0
```

Bit operations with immediate: vand, vor, vxor Performs a given bitwise operation on a register and an 8-bit immediate replicated for each component. `$vc` output supported for zero flag only.

Instructions:

Instruction	Operands	Opcode
vand	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xaa
vxor	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xab
vor	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xaf

Operation:

```
    for idx in range(16):
        s1 = $v[SRC1][idx]

        if op == 'vand':
            res = s1 & BIMM
        elif op == 'vxor':
            res = s1 ^ BIMM
        elif op == 'vor':
            res = s1 | BIMM

        $v[DST][idx] = res
        if VCDST < 4:
            $vc[VCDST].sf[idx] = 0
            $vc[VCDST].zf[idx] = res == 0
```

Shift operations: vshr, vsar Performs a SIMD right shift, like the *scalar bytewise shift instruction*. `$vc` output is fully supported, with bit 7 of the result used as the sign flag.

Instructions:

Instruction	Operands	Opcode
vsar	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x8e
vshr	[$vc[VCDST]] $v[DST] $v[SRC1] $v[SRC2]	0x9e
vsar	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xae
vshr	[$vc[VCDST]] $v[DST] $v[SRC1] BIMM	0xbe

Operation:

```
for idx in range(16):
    s1 = $v[SRC1][idx]
    if opcode & 0x20:
        s2 = BIMM
    else:
        s2 = $v[SRC2][idx]

    if opcode & 0x10:
        # unsigned
        s1 &= 0xff
    else:
        # signed
        s1 = sext(s1, 7)

    shift = sext(s2, 3)

    if shift < 0:
        res = s1 << -shift
    else:
        res = s1 >> shift

    $v[DST][idx] = res

    if VCDST < 4:
        $vc[VCDST].sf[idx] = res >> 7 & 1
        $vc[VCDST].zf[idx] = res == 0
```

Linear interpolation: vlrp A SIMD linear interpolation instruction. Takes two sources: a register pair containing the two values to interpolate, and a register containing the interpolation factor. The result is basically `SRC1.1 * (SRC2 >> SHIFT) + SRC1.2 * (1 - (SRC2 >> SHIFT))`. All inputs are unsigned fractions.

Instructions:

Instruction	Operands	Opcode
vlrp	RND SHIFT $v[DST] $v[SRC1]d $v[SRC2]	0x90

Operation:

```
for idx in range(16):
    val1 = $v[SRC1][idx]
    val2 = $v[SRC1 | 1][idx]
    a = mad_expand(val2, 'fract', 'u', SHIFT)
    res = mad(a, val1 - val2, $v[SRC2][idx], 0, 0, RND, 'fract', 'u', SHIFT, 'hi')
    $v[DST][idx] = mad_read(res, 'fract', 'u', SHIFT, 'hi')
```

Multiply and multiply with accumulate: vmul, vmac Performs a simple multiplication of two sources (but with the full set of weird options available). The result is either added to the vector accumulator (`vmac`) or replaces it (`vmul`). The result can additionally be read to a vector register, but doesn't have to be.

The instructions come in many variants: they can store the result in a vector register or not, have unsigned or signed output, and register or immediate second source. The set of available combinations is incomplete, however: while the

`$v`-writing variants have all combinations available, there are no unsigned variants of register-register `vmul` with no `$v` write, nor unsigned register-immediate `vmac` with no `$v` write. Also, unsigned register-immediate `vmul` with no `$v` output is a *bad opcode*.

Instructions:

Instruction	Operands	Opcode
`vmul s`	`RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] SIGN2 $v[SRC2]`	`0x80`
`vmul s`	`RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] SIGN2 BIMMMUL`	`0xa0`
`vmul u`	`RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] SIGN2 BIMMBAD`	`0xb0` (bad opcode)
`vmul s`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 $v[SRC2]`	`0x81`
`vmul u`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 $v[SRC2]`	`0x91`
`vmul s`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 BIMMMUL`	`0xa1`
`vmul u`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 BIMMMUL`	`0xb1`
`vmac s`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 $v[SRC2]`	`0x82`
`vmac u`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 $v[SRC2]`	`0x92`
`vmac s`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 BIMMMUL`	`0xa2`
`vmac u`	`RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] SIGN2 BIMMMUL`	`0xb2`
`vmac s`	`RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] SIGN2 $v[SRC2]`	`0x83`
`vmac u`	`RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] SIGN2 $v[SRC2]`	`0x93`
`vmac s`	`RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] SIGN2 BIMMMUL`	`0xa3`

Operation:

```
    for idx in range(16):
        # read inputs
        s1 = $v[SRC1][idx]
        if opcode & 0x20:
            if op == 0x30:
                s2 = BIMMBAD
            else:
                s2 = BIMMMUL << 2
        else:
            s2 = $v[SRC2][idx]

        # convert inputs
        s1 = mad_input(s1, FRACTINT, SIGN1)
        s2 = mad_input(s2, FRACTINT, SIGN2)

        # do the computation
        if op == 'vmac':
            a = $va[idx]
        else:
```

```
            a = 0
        res = mad(a, s1, s2, 0, 0, RND, FRACTINT, op.sign, SHIFT, HILO)

        # write result
        $va[idx] = res
        if DST is not None:
            $v[DST][idx] = mad_read(res, FRACTINT, op.sign, SHIFT, HILO)
```

Dual multiply and add/accumulate: vmac2, vmad2 Performs two multiplications and adds the result to a given source or to the vector accumulator. The result is written to the vector accumulator and can also be written to a `$v` register. For each multiplication, one input is a register source, and the other is s2v factor. The register sources for the multiplications are a register pair. The s2v sources for the multiplications are either s2v factors (one factor from each pair is selected according to s2v `$vc` input) or 0/1 as decided by s2v mask.

The instructions come in signed and unsigned variants. Apart from some bad opcodes (which overlay `SRC3` with mad param fields), only `$v` writing versions have unsigned variants.

Instructions:

Instruction	Operands	Opcode
vmad2s	S2VMODE RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1]d SIGN2 $v[SRC2]	0x84
vmad2s	S2VMODE RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1]d SIGN2 $v[SRC2]	0x85
vmad2u	S2VMODE RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1]d SIGN2 $v[SRC2]	0x95
vmac2s	S2VMODE RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1]d	0x86
vmac2u	S2VMODE RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] $v[SRC3]	0x96 (bad opcode)
vmac2s	S2VMODE RND FRACTINT SHIFT HILO # SIGN1 $v[SRC1] $v[SRC3]	0xa6 (bad opcode)
vmac2s	S2VMODE RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1]d	0x87
vmac2u	S2VMODE RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1]d	0x97
vmac2s	S2VMODE RND FRACTINT SHIFT HILO $v[DST] SIGN1 $v[SRC1] $v[SRC3]	0xa7 (bad opcode)

Operation:

```
    for idx in range(16):
        # read inputs
        s11 = $v[SRC1][idx]
        if opcode in (0x96, 0xa6, 0xa7):
            # one of the bad opcodes
            s12 = $v[SRC3][idx]
        else:
            s12 = $v[SRC1 | 1][idx]

        s2 = $v[SRC2][idx]

        # convert inputs
        s11 = mad_input(s11, FRACTINT, SIGN1)
        s12 = mad_input(s12, FRACTINT, SIGN1)
        s2 = mad_input(s2, FRACTINT, SIGN2)
```

```
        # prepare A value
        if op == 'vmad2':
            a = mad_expand(s2, FRACTINT, sign, SHIFT)
        else:
            a = $va[idx]

        # prepare factors
        if S2VMODE == 'mask':
            if s2v.mask[0] & 1 << idx:
                f1 = 0x100
            else:
                f1 = 0
            if s2v.mask[1] & 1 << idx:
                f2 = 0x100
            else:
                f2 = 0
        else:
            # 'factor'
            cc = s2v.vcmask >> idx & 1
            f1 = s2v.factor[0 | cc]
            f2 = s2v.factor[2 | cc]

        # do the operation
        res = mad(a, s11, f1, s12, f2, RND, FRACTINT, sign, SHIFT, HILO)

        # write result
        $va[idx] = res
        if DST is not None:
            $v[DST][idx] = mad_read(res, FRACTINT, op.sign, SHIFT, HILO)
```

Dual linear interpolation: vlrp2 This instruction performs the following steps:

- read a quad register source selected by `SRC1`
- rotate the source quad by the amount selected by bits 4-5 of a selected `$c` register
- for each component:
  - treat register 0 of the quad as function value at (0, 0)
  - treat register 2 as value at (1, 0)
  - treat register 3 as value at (0, 1)
  - select a pair of factors from s2v input based on selected flag of selected `$vc` register
  - treat the factors as a coordinate pair and interpolate function value at these coordinates
  - write result to `$v` register and optionally `$va`

The inputs and outputs may be signed or unsigned. A shift and rounding mode can be selected. Additionally, there's an option to XOR register 0 with 0x80 before use as the base value (but not for the differences used in interpolation). Don't ask me.

Instructions:

Instruction	Operands	Opcode
`vlrp2`	`SIGND VAWRITE RND SHIFT $v[DST] SIGNS LRP2X $v[SRC1]q $c[COND] $vc[VCSRC] VCSEL`	`0xb3`

Operation:

```
    # a function selecting the factors
    def get_lrp2_factors(idx):
        if VCSEL == 'sf':
            vcmask = $vc[VCSRC].sf
        else:
            vcmask = $vc[VCSRC].zf

        cc = vcmask >> idx & 1;
        f1 = s2v.factor[0 | cc]
        f2 = s2v.factor[2 | cc]

        return f1, f2

    # determine rotation
    rot = $c[COND] >> 4 & 3

    for idx in range(16):
        # read inputs, maybe do the xor
        s10x = s10 = $v[(SRC1 & 0x1c) | ((SRC1 + rot) & 3)][idx]
        s12 = $v[(SRC1 & 0x1c) | ((SRC1 + rot + 2) & 3)][idx]
        s13 = $v[(SRC1 & 0x1c) | ((SRC1 + rot + 3) & 3)][idx]
        if LRP2X:
            s10x ^= 0x80

        # convert inputs if necessary
        s10 = mad_input(s10, 'fract', SIGNS)
        s12 = mad_input(s12, 'fract', SIGNS)
        s13 = mad_input(s13, 'fract', SIGNS)
        s10x = mad_input(s10x, 'fract', SIGNS)

        # do it
        a = mad_expand(s10x, 'fract', SIGND, SHIFT)
        f1, f2 = get_lrp2_factors(idx)
        res = mad(a, s12 - s10, f1, s13 - s10, f2, RND, 'fract', SIGND, SHIFT, 'hi')

        # write outputs
        if VAWRITE:
            $va[idx] = res
        $v[DST][idx] = mad_read(res, 'fract', SIGND, SHIFT, 'hi')
```

Quad linear interpolation, part 1: vlrp4a Works like the previous variant, but only outputs to `$va` and lacks some flags. Both outputs and inputs are unsigned.

Instructions:

Instruction	Operands	Opcode
vlrp4a	RND SHIFT # $v[SRC1]q $c[COND] $vc[VCSRC] VCSEL	0xb4

Operation:

```
    rot = $c[COND] >> 4 & 3

    for idx in range(16):

        s10 = $v[(SRC1 & 0x1c) | ((SRC1 + rot) & 3)][idx]
        s12 = $v[(SRC1 & 0x1c) | ((SRC1 + rot + 2) & 3)][idx]
        s13 = $v[(SRC1 & 0x1c) | ((SRC1 + rot + 3) & 3)][idx]

        a = mad_expand(s10, 'fract', 'u', SHIFT)
        f1, f2 = get_lrp2_factors(idx)
```

```
        $va[idx] = mad(a, s12 - s10, f1, s13 - s10, f2, RND, 'fract', 'u', SHIFT, 'lo')
```

Factor linear interpolation: vlrpf Has similiar input processing to `vlrp2`, but instead uses source 1 registers 2 and 3 to interpolate s2v input. Result is `SRC2 + SRC1.2 * F1 + SRC1.3 * (F2 - F1)`.

Instructions:

Instruction	Operands	Op-code
vlrpf	RND SHIFT # $v[SRC1]q $c[COND] $v[SRC2] $vc[VCSRC] VCSEL	0xb5

Operation:

```
    rot = $c[COND] >> 4 & 3

    for idx in range(16):

        s12 = $v[(SRC1 & 0x1c) | ((SRC1 + rot + 2) & 3)][idx]
        s13 = $v[(SRC1 & 0x1c) | ((SRC1 + rot + 3) & 3)][idx]
        s2 = sext($v[SRC2][idx], 7)

        a = mad_expand(s2, 'fract', 'u', SHIFT)
        f1, f2 = get_lrp2_factors(idx)

        $va[idx] = mad(a, s12 - s13, f1, s13, f2, RND, 'fract', 'u', SHIFT, 'lo')
```

Quad linear interpolation, part 2: vlrp4b Can be used together with `vlrp4a` for quad linear interpolation. First s2v factor is the interpolation coefficient for register 1, and second factor is the interpolation coefficient for the extra register (`$vx`).

Alternatively, can be coupled with `vlrpf`.

Instructions:

Instruction	Operands	Op-code
vlrp4bu	ALTRND ALTSHIFT $v[DST] $v[SRC1]q $c[COND] SLCT $vc[VCSRC] VCSEL	0xb6
vlrp4bs	ALTRND ALTSHIFT $v[DST] $v[SRC1]q $c[COND] SLCT $vc[VCSRC] VCSEL	0xb7

Operation:

```
    for idx in range(16):
        if SLCT == 4:
                rot = $c[COND] >> 4 & 3
                s10 = $v[(SRC1 & 0x1c) | ((SRC1 + rot) & 3)][idx]
                s11 = $v[(SRC1 & 0x1c) | ((SRC1 + rot + 1) & 3)][idx]
        else:
                adjust = $c[COND] >> SLCT & 1
                s10 = s11 = $v[src1 ^ adjust][idx]

        f1, f2 = get_lrp2_factors(idx)

        res = mad($va[idx], s11 - s10, f1, $vx[idx] - s10, f2, ALTRND, 'fract', op.sign, ALTSHIFT, '

        $va[idx] = res
        $v[DST][idx] = mad_read(res, 'fract', op.sign, ALTSHIFT, 'hi')
```

Branch unit

Contents

Todo

write me

Introduction

Todo

write me

Branch registers

Todo

write me

Address unit

Contents

Introduction The address unit is one of the four execution units of VP1. It transfers data between that data store and registers, controls the *DMA unit*, and performs address calculations.

The data store The data store is the working memory of VP1, 8kB in size. Data can be transferred between the data store and `$r`/`$v` registers using load/store instructions, or between the data store and main memory using *the DMA engine*. It's often treated as two-dimensional, with row stride selectable between `0x10`, `0x20`, `0x40`, and `0x80` bytes: there are "load vertical" instructions which gather consecutive bytes vertically rather than horizontally.

Because of its 2D capabilities, the data store is internally organized into 16 independently addressable 16-bit wide banks of 256 cells each, and the memory addresses are carefully spread between the banks so that both horizontal and vertical loads from any address will require at most one access to every bank. The bank assignments differ between the supported strides, so row stride is basically a part of the address, and an area of memory always has to be accessed with the same stride (unless you don't care about its previous contents). Specifically, the translation of (address, stride) pair into (bank, cell index, high/low byte) is as follows:

```
def address_xlat(addr, stride):
    bank = addr & 0xf
    hilo = addr >> 4 & 1
    cell = addr >> 5 & 0xff
    if stride == 0:
        # 0x10 bytes
        bank += (addr >> 5) & 7
    elif stride == 1:
        # 0x20 bytes
        bank += addr >> 5
    elif stride == 0x40:
        # 0x40 bytes
        bank += addr >> 6
    elif stride == 0x80:
        # 0x80 bytes
        bank += addr >> 7
    bank &= 0xf
    return bank, cell, hilo
```

In pseudocode, data store bytes are denoted by `DS[bank, cell, hilo]`.

In case of vertical access with 0x10 bytes stride, all 16 bits of 8 banks will be used by a 16-byte access. In all other cases, 8 bits of all 16 banks will be used for such access. DMA transfers can make use of the full 256-bit width of the data store, by transmitting 0x20 consecutive bytes at a time.

The data store can be accessed by load/store instructions in one of four ways:

- horizontal: 16 consecutive naturally aligned addresses are used:

```
    def addresses_horizontal(addr, stride):
        addr &= 0x1ff0
        return [address_xlat(addr | idx, stride) for idx in range(16)]
```

- vertical: 16 addresses separated by stride bytes are used, also naturally aligned:

```
    def addresses_vertical(addr, stride):
        addr &= 0x1fff
        # clear the bits used for y coord
        addr &= ~(0xf << (4 + stride))
        return [address_xlat(addr | idx << (4 + stride)) for idx in range(16)]
```

- scalar: like horizontal, but 4 bytes:

```
    def addresses_horizontal_short(addr, stride):
        addr &= 0x1ffc
        return [address_xlat(addr | idx, stride) for idx in range(4)]
```

- raw: the raw data store coordinates are provided directly

Address registers The address unit has 32 address registers, `$a0-$a31`. These are used for address storage. If they're used to store data store addresses (and not DMA command parameters), they have the following bitfields:

- bits 0-15: `addr` - the actual data store address
- bits 16-29: `limit` - can store the high bounduary of an array, to assist in looping
- bits 30-31: `stride` - selects data store stride:
  - 0: 0x10 bytes
  - 1: 0x20 bytes
  - 2: 0x40 bytes
  - 3: 0x80 bytes

There are also 3 bits in each `$c` register belonging to the address unit. They are:

- bits 8-9: long address flags
  - bit 8: sign flag - set equal to bit 31 of the result
  - bit 9: zero flag - set if the result is 0
- bit 10: short address flag
  - bit 10: end flag - set if `addr` field of the result is greater than or equal to `limit`

Some address instructions set either the long or short flags of a given `$c` register according to the result.

Instruction format The instruction word fields used in address instructions in addition to *the ones used in scalar instructions* are:

- bit 0: for opcode `0xd7`, selects the subopcode:
  - 0: *load raw: ldr*
  - 1: *store raw and add: star*
- bits 3-13: `UIMM`: unsigned 13-bit immediate.

Todo

list me

Opcodes The opcode range assigned to the address unit is `0xc0-0xdf`. The opcodes are:

- `0xc0`: *load vector horizontal and add: ldavh*
- `0xc1`: *load vector vertical and add: ldavv*
- `0xc2`: *load scalar and add: ldas*
- `0xc3`: ??? (`xdld`)

- `0xc4`: *store vector horizontal and add: stavh*
- `0xc5`: *store vector vertical and add: stavv*
- `0xc6`: *store scalar and add: stas*
- `0xc7`: ??? (`xdst`)
- `0xc8`: *load extra horizontal and add: ldaxh*
- `0xc9`: *load extra vertical and add: ldaxv*
- `0xca`: *address addition: aadd*
- `0xcb`: *addition: add*
- `0xcc`: *set low bits: setlo*
- `0xcd`: *set high bits: sethi*
- `0xce`: ??? (`xdbar`)
- `0xcf`: ??? (`xdwait`)
- `0xd0`: *load vector horizontal and add: ldavh*
- `0xd1`: *load vector vertical and add: ldavv*
- `0xd2`: *load scalar and add: ldas*
- `0xd3`: *bitwise operation: bitop*
- `0xd4`: *store vector horizontal and add: stavh*
- `0xd5`: *store vector vertical and add: stavv*
- `0xd6`: *store scalar and add: stas*
- `0xd7`: depending on instruction bit 0:
  - 0: *load raw: ldr*
  - 1: *store raw and add: star*
- `0xd8`: *load vector horizontal: ldvh*
- `0xd9`: *load vector vertical: ldvv*
- `0xda`: *load scalar: lds*
- `0xdb`: ???
- `0xdc`: *store vector horizontal: stvh*
- `0xdd`: *store vector vertical: stvv*
- `0xde`: *store scalar: sts*
- `0xdf`: the canonical address nop opcode

Todo

complete the list

Instructions

Set low/high bits: setlo, sethi Sets low or high 16 bits of a register to an immediate value. The other half is unaffected.

Instructions:

Instruction	Operands	Opcode
setlo	$a[DST] IMM16	0xcc
sethi	$a[DST] IMM16	0xcd

Operation:

```
if op == 'setlo':
    $a[DST] = ($a[DST] & 0xffff0000) | IMM16
else:
    $a[DST] = ($a[DST] & 0xffff) | IMM16 << 16
```

Addition: add Does what it says on the tin. The second source comes from a mangled register index. The long address flags are set.

Instructions:

Instruction	Operands	Opcode
add	[$c[CDST]] $a[DST] $a[SRC1] $a[SRC2S]	0xcb

Operation:

```
res = $a[SRC1] + $a[SRC2S]

$a[DST] = res

cres = 0
if res & 1 << 31:
    cres |= 1
if res == 0:
    cres |= 2
if CDST < 4:
    $c[CDST].address.long = cres
```

Bit operations: bitop Performs an *arbitrary two-input bit operation* on two registers, selected by SRC1 and SRC2. The long address flags are set.

Instructions:

Instruction	Operands	Opcode
bitop	BITOP [$c[CDST]] $a[DST] $a[SRC1] $a[SRC2]	0xd3

Operation:

```
res = bitop(BITOP, $a[SRC2], $a[SRC1]) & 0xffffffff

$a[DST] = res

cres = 0
if res & 1 << 31:
    cres |= 1
if res == 0:
    cres |= 2
if CDST < 4:
    $c[CDST].address.long = cres
```

Address addition: aadd Adds the contents of a register to the addr field of another register. Short address flag is set.

Instructions:

Instruction	Operands	Opcode
aadd	[$c[CDST]] $a[DST] $a[SRC2S]	0xca

Operation:

```
$a[DST].addr += $a[SRC2S]

if CDST < 4:
    $c[CDST].address.short = $a[DST].addr >= $a[DST].limit
```

Load: ldvh, ldvv, lds Loads from the given address ORed with an unsigned 11-bit immediate. `ldvh` is a horizontal vector load, `ldvv` is a vertical vector load, and `lds` is a scalar load. Curiously, while register is ORed with the immdiate to form the address, they are *added* to make `$c` output.

Instructions:

Instruction	Operands	Opcode
ldvh	$v[DST] [$c[CDST]] $a[SRC1] UIMM	0xd8
ldvv	$v[DST] [$c[CDST]] $a[SRC1] UIMM	0xd9
lds	$r[DST] [$c[CDST]] $a[SRC1] UIMM	0xda

Operation:

```
if op == 'ldvh':
    addr = addresses_horizontal($a[SRC1].addr | UIMM, $a[SRC1].stride)
    for idx in range(16):
        $v[DST][idx] = DS[addr[idx]]
elif op == 'ldvv':
    addr = addresses_vertical($a[SRC1].addr | UIMM, $a[SRC1].stride)
    for idx in range(16):
        $v[DST][idx] = DS[addr[idx]]
elif op == 'lds':
    addr = addresses_scalar($a[SRC1].addr | UIMM, $a[SRC1].stride)
    for idx in range(4):
        $r[DST][idx] = DS[addr[idx]]

if CDST < 4:
    $c[CDST].address.short = (($a[SRC1].addr + UIMM) & 0xffff) >= $a[SRC1].limit
```

Load and add: ldavh, ldavv, ldas Loads from the given address, then post-increments the address by the contents of a register (like *the aadd instruction*) or an immediate. `ldavh` is a horizontal vector load, `ldavv` is a vertical vector load, and `ldas` is a scalar load.

Instructions:

Instruction	Operands	Opcode
ldavh	$v[DST] [$c[CDST]] $a[SRC1] $a[SRC2S]	0xc0
ldavv	$v[DST] [$c[CDST]] $a[SRC1] $a[SRC2S]	0xc1
ldas	$r[DST] [$c[CDST]] $a[SRC1] $a[SRC2S]	0xc2
ldavh	$v[DST] [$c[CDST]] $a[SRC1] IMM	0xd0
ldavv	$v[DST] [$c[CDST]] $a[SRC1] IMM	0xd1
ldas	$r[DST] [$c[CDST]] $a[SRC1] IMM	0xd2

Operation:

```
if op == 'ldavh':
    addr = addresses_horizontal($a[SRC1].addr, $a[SRC1].stride)
    for idx in range(16):
        $v[DST][idx] = DS[addr[idx]]
elif op == 'ldavv':
    addr = addresses_vertical($a[SRC1].addr, $a[SRC1].stride)
```

```
        for idx in range(16):
            $v[DST][idx] = DS[addr[idx]]
    elif op == 'ldas':
        addr = addresses_scalar($a[SRC1].addr, $a[SRC1].stride)
        for idx in range(4):
            $r[DST][idx] = DS[addr[idx]]

    if IMM is None:
        $a[SRC1].addr += $a[SRC2S]
    else:
        $a[SRC1].addr += IMM

    if CDST < 4:
        $c[CDST].address.short = $a[SRC1].addr >= $a[SRC1].limit
```

Store: stvh, stvv, sts Like corresponding *ld* instructions*, but store instead of load. `SRC1` and `DST` fields are exchanged.

Instructions:

Instruction	Operands	Opcode
stvh	$v[SRC1] [$c[CDST]] $a[DST] UIMM	0xdc
stvv	$v[SRC1] [$c[CDST]] $a[DST] UIMM	0xdd
sts	$r[SRC1] [$c[CDST]] $a[DST] UIMM	0xde

Operation:

```
    if op == 'stvh':
        addr = addresses_horizontal($a[DST].addr | UIMM, $a[DST].stride)
        for idx in range(16):
            DS[addr[idx]] = $v[SRC1][idx]
    elif op == 'stvv':
        addr = addresses_vertical($a[DST].addr | UIMM, $a[DST].stride)
        for idx in range(16):
            DS[addr[idx]] = $v[SRC1][idx]
    elif op == 'sts':
        addr = addresses_scalar($a[DST].addr | UIMM, $a[DST].stride)
        for idx in range(4):
            DS[addr[idx]] = $r[SRC1][idx]

    if CDST < 4:
        $c[CDST].address.short = (($a[DST].addr + UIMM) & 0xffff) >= $a[DST].limit
```

Store and add: stavh, stavv, stas Like corresponding *lda* instructions*, but store instead of load. `SRC1` and `DST` fields are exchanged.

Instructions:

Instruction	Operands	Opcode
stavh	$v[SRC1] [$c[CDST]] $a[DST] $a[SRC2S]	0xc4
stavv	$v[SRC1] [$c[CDST]] $a[DST] $a[SRC2S]	0xc5
stas	$r[SRC1] [$c[CDST]] $a[DST] $a[SRC2S]	0xc6
stavh	$v[SRC1] [$c[CDST]] $a[DST] IMM	0xd4
stavv	$v[SRC1] [$c[CDST]] $a[DST] IMM	0xd5
stas	$r[SRC1] [$c[CDST]] $a[DST] IMM	0xd6

Operation:

```
    if op == 'stavh':
        addr = addresses_horizontal($a[DST].addr, $a[DST].stride)
```

```
        for idx in range(16):
            DS[addr[idx]] = $v[SRC1][idx]
    elif op == 'stavv':
        addr = addresses_vertical($a[DST].addr, $a[DST].stride)
        for idx in range(16):
            DS[addr[idx]] = $v[SRC1][idx]
    elif op == 'stas':
        addr = addresses_scalar($a[DST].addr, $a[DST].stride)
        for idx in range(4):
            DS[addr[idx]] = $r[SRC1][idx]

    if IMM is None:
        $a[DST].addr += $a[SRC2S]
    else:
        $a[DST].addr += IMM

    if CDST < 4:
        $c[CDST].address.short = $a[DST].addr >= $a[DST].limit
```

Load raw: ldr A raw load instruction. Loads one byte from each bank of the data store. The banks correspond directly to destination register components. The addresses are composed from ORing an address register with components of a vector register shifted left by 4 bits. Specifically, for each component, the byte to access is determined as follows:

- take address register value
- shift it right 4 bits (they're discarded)
- OR with the corresponding component of vector source register
- bit 0 of the result selects low/high byte of the bank
- bits 1-8 of the result select the cell index in the bank

This instruction shares the `0xd7` opcode with *star*. They are differentiated by instruction word bit 0, set to 0 in case of `ldr`.

Instructions:

Instruction	Operands	Opcode
`ldr`	`$v[DST] $a[SRC1] $v[SRC2]`	`0xd7.0`

Operation:

```
    for idx in range(16):
        addr = $a[SRC1].addr >> 4 | $v[SRC2][idx]
        $v[DST][idx] = DS[idx, addr >> 1 & 0xff, addr & 1]
```

Store raw and add: star A raw store instruction. Stores one byte to each bank of the data store. As opposed to raw load, the addresses aren't controllable per component: the same byte and cell index is accessed in each bank, and it's selected by post-incremented address register like for *sta**. `$c` output is not supported.

This instruction shares the `0xd7` opcode with *lda*. They are differentiated by instruction word bit 0, set to 1 in case of `star`.

Instructions:

Instruction	Operands	Opcode
`star`	`$v[SRC1] $a[DST] $a[SRC2S]`	`0xd7.1`

Operation:

```
    for idx in range(16):
        addr = $a[DST].addr >> 4
        DS[idx, addr >> 1 & 0xff, addr & 1] = $v[SRC1][idx]

    $a[DST].addr += $a[SRC2S]
```

Load extra and add: ldaxh, ldaxv Like *ldav**, except the data is loaded to `$vx`. If a selected `$c` flag is set (the same one as used for `SRC2S` mangling), the same data is also loaded to a `$v` register selected by `DST` field mangled in the same way as in *vlrp2* family of instructions.

Instructions:

Instruction	Operands	Opcode
`ldaxh`	`$v[DST]q [$c[CDST]] $a[SRC1] $a[SRC2S]`	`0xc8`
`ldaxv`	`$v[DST]q [$c[CDST]] $a[SRC1] $a[SRC2S]`	`0xc9`

Operation:

```
    if op == 'ldaxh':
        addr = addresses_horizontal($a[SRC1].addr, $a[SRC1].stride)
        for idx in range(16):
            $vx[idx] = DS[addr[idx]]
    elif op == 'ldaxv':
        addr = addresses_vertical($a[SRC1].addr, $a[SRC1].stride)
        for idx in range(16):
            $vx[idx] = DS[addr[idx]]

    if $c[COND] & 1 << SLCT:
        for idx in range(16):
            $v[(DST & 0x1c) | ((DST + ($c[COND] >> 4)) & 3)][idx] = $vx[idx]

    $a[SRC1].addr += $a[SRC2S]

    if CDST < 4:
        $c[CDST].address.short = $a[SRC1].addr >= $a[SRC1].limit
```

DMA transfers

Contents

Todo

write me

Introduction

Todo

write me

DMA registers

Todo

write me

FIFO interface

Contents

Todo

write me

Introduction

Todo

write me

Method registers

Todo

write me

FIFO access registers

Todo

write me

Introduction

Todo

write me

2.11.2 VP2/VP3 vμc processor

Contents:

Overview of VP2/VP3/VP4 vμc hardware

Contents

Introduction

vμc is a microprocessor unit used as the second stage of the VP2 [in H.264 mode only], VP3 and VP4 video decoding pipelines. The same name is also used to refer to the instruction set of this microprocessor. vμc's task is to read decoded bitstream data written by VLD into the MBRING structure, do any required calculations on this data, then construct instructions for the VP stage regarding processing of the incoming macroblocks. The work required of vμc is dependent on the codec and may include eg. motion vector derivation, calculating quantization parameters, converting macroblock type to prediction modes, etc.

On VP2, the vμc is located inside the PBSP engine [see vdec/vp2/pbsp.txt]. On VP3 and VP4, it is located inside the PPDEC engine [see vdec/vp3/ppdec.txt].

The vμc unit is made of the following subunits:

- the vμc microcprocessor - oversees everything and does the calculations that are not performance-sensitive enough to be done in hardware
- MBRING input and parsing circuitry - reads bitstream data parsed by the VLD
- MVSURF input and output circuitry - the MVSURF is a storage buffer attached to all reference pictures in H.264 and to P pictures in VC-1, MPEG-4. It stores the motion vectors and other data used for direct prediction in B pictures. There are two MVSURFs that can be used: the output MVSURF that will store the data of the current picture, and the input MVSURF that should store the data for the first picture in L1 list [H.264] or the last P picture [other codecs]
- VPRINGs output circuitry [VP2 only] - the VPRINGs are ring buffers filled by vμc with instructions for various VP subunits. There are three VPRINGs: VPRING_DEBLOCK used for deblocking commands, VPRIND_RESIDUAL used for the residual transform coefficients, and VPRINT_CTRL used for the motion vectors and other control data.
- direct VP connection [VP3, VP4 only] - the VP3+ vμc is directly connected to the VP engine, instead of relying on ring buffers in memory.

The MMIO registers - VP2

The vμc registers are located in PBSP XLMI space at addresses 0x08000:0x10000 [BAR0 addresses 0x103200:0x103400]. They are:

08000:0a000/103200:103280: DATA - vμc microprocessor data space [vdec/vuc/isa.txt]

0a000/103280: ICNT - executed instructions counter, aliased to vμc special register $sr15 [$icnt]

0a100/103284: WDCNT - watchdog count - when ICNT reaches WDCNT value and WDCNT is not equal to 0xffff, a watchdog interrupt is raised

0a200/103288: CODE_CONTROL - code execution control [vdec/vuc/isa.txt] 0a300/10328c: CODE_WINDOW - code access window [vdec/vuc/isa.txt] 0a400/103290: H2V - host to vμc scratch register [vdec/vuc/isa.txt] 0a500/103294: V2H - vμc to host scratch register [vdec/vuc/isa.txt] 0a600/103298: PARM - sequence/picture/slice parameters required by vμc

hardware, aliased to vμc special register $sr7 [$parm]

0a700/10329c: PC - program counter [vdec/vuc/isa.txt] 0a800/1032a0: VPRING_RESIDUAL.OFFSET - the VPRING_RESIDUAL offset 0a900/1032a4: VPRING_RESIDUAL.HALT_POS - the VPRING_RESIDUAL halt position 0aa00/1032a8: VPRING_RESIDUAL.WRITE_POS - the VPRING_RESIDUAL write position 0ab00/1032ac: VPRING_RESIDUAL.SIZE - the VPRING_RESIDUAL size 0ac00/1032b0: VPRING_CTRL.OFFSET - the VPRING_CTRL offset 0ad00/1032b4: VPRING_CTRL.HALT_POS - the VPRING_CTRL halt position 0ae00/1032b8: VPRING_CTRL.WRITE_POS - the VPRING_CTRL write position 0af00/1032bc: VPRING_CTRL.SIZE - the VPRING_CTRL size 0b000/1032c0: VPRING_DEBLOCK.OFFSET - the VPRING_DEBLOCK offset 0b100/1032c4: VPRING_DEBLOCK.HALT_POS - the VPRING_DEBLOCK halt position 0b200/1032c8: VPRING_DEBLOCK.WRITE_POS - the VPRING_DEBLOCK write position 0b300/1032cc: VPRING_DEBLOCK.SIZE - the VPRING_DEBLOCK size 0b400/1032d0: VPRING_TRIGGER - flush/resume triggers the for VPRINGs 0b500/1032d4: INTR - interrupt status 0b600/1032d8: INTR_EN - interrupt enable mask 0b700/1032dc: VPRING_ENABLE - enables VPRING access 0b800/1032e0: MVSURF_IN_OFFSET - MVSURF_IN offset [vdec/vuc/mvsurf.txt] 0b900/1032e4: MVSURF_IN_PARM - MVSURF_IN parameters [vdec/vuc/mvsurf.txt] 0ba00/1032e8: MVSURF_IN_LEFT - MVSURF_IN data left [vdec/vuc/mvsurf.txt] 0bb00/1032ec: MVSURF_IN_POS - MVSURF_IN position [vdec/vuc/mvsurf.txt] 0bc00/1032f0: MVSURF_OUT_OFFSET - MVSURF_OUT offset [vdec/vuc/mvsurf.txt] 0bd00/1032f4: MVSURF_OUT_PARM - MVSURF_OUT parameters [vdec/vuc/mvsurf.txt] 0be00/1032f8: MVSURF_OUT_LEFT - MVSURF_OUT space left [vdec/vuc/mvsurf.txt] 0bf00/1032fc: MVSURF_OUT_POS - MVSURF_OUT position [vdec/vuc/mvsurf.txt] 0c000/103300: MBRING_OFFSET - the MBRING offset 0c100/103304: MBRING_SIZE - the MBRING size 0c200/103308: MBRING_READ_POS - the MBRING read position 0c300/10330c: MBRING_READ_AVAIL - the bytes left to read in MBRING

The MMIO registers - VP3/VP4

The vμc registers are located in PPDEC falcon IO space at addresses 0x10000:0x14000 [BAR0 addresses 0x085400:0x085500]. They are:

10000:11000/085400:085440: DATA - vμc microprocessor data space [vdec/vuc/isa.txt]

11000/085440: CODE_CONTROL - code execution control [vdec/vuc/isa.txt] 11100/085444: CODE_WINDOW - code access window [vdec/vuc/isa.txt] 11200/085448: ICNT - executed instructions counter, aliased to vμc special

register $sr15 [$icnt]

11300/08544c: WDCNT - watchdog count - when ICNT reaches WDCNT value and WDCNT is not equal to 0xffff, a watchdog interrupt is raised

11400/085450: H2V - host to vμc scratch register [vdec/vuc/isa.txt] 11500/085454: V2H - vμc to host scratch register [vdec/vuc/isa.txt] 11600/085458: PARM - sequence/picture/slice parameters required by vμc

hardware, aliased to vμc special register $sr7 [$parm]

11700/08545c: PC - program counter [vdec/vuc/isa.txt] 11800/085460: RPITAB - the address of refidx -> RPI translation table 11900/085464: REFTAB - the address of RPI -> VM address translation table 11a00/085468: BUSY - a status reg showing which subunits of vμc are busy 11c00/085470: INTR - interrupt status 11d00/085474: INTR_EN - interrupt enable mask 12000/085480: MVSURF_IN_ADDR - MVSURF_IN address [vdec/vuc/mvsurf.txt] 12100/085484: MVSURF_IN_PARM - MVSURF_IN parameters [vdec/vuc/mvsurf.txt] 12200/085488: MVSURF_IN_LEFT - MVSURF_IN data left [vdec/vuc/mvsurf.txt] 12300/08548c: MVSURF_IN_POS - MVSURF_IN position [vdec/vuc/mvsurf.txt] 12400/085490: MVSURF_OUT_ADDR - MVSURF_OUT address [vdec/vuc/mvsurf.txt] 12500/085494: MVSURF_OUT_PARM - MVSURF_OUT parameters [vdec/vuc/mvsurf.txt]

12600/085498: MVSURF_OUT_LEFT - MVSURF_OUT space left [vdec/vuc/mvsurf.txt] 12700/08549c: MVSURF_OUT_POS - MVSURF_OUT position [vdec/vuc/mvsurf.txt] 12800/0854a0: MBRING_OFFSET - the MBRING offset 12900/0854a4: MBRING_SIZE - the MBRING size 12a00/0854a8: MBRING_READ_POS - the MBRING read position 12b00/0854ac: MBRING_READ_AVAIL - the bytes left to read in MBRING 12c00/0854b0: ??? [XXX] 12d00/0854b4: ??? [XXX] 12e00/0854b8: ??? [XXX] 12f00/0854bc: STAT - control/status register [vdec/vuc/isa.txt] 13000/0854c0: ??? [XXX] 13100/0854c4: ??? [XXX]

Interrupts

Todo

write me

VP2/VP3/VP4 vμc ISA

Contents

Introduction

This file deals with description of the ISA used by the vμc microprocessor, which is described in vdec/vuc/intro.txt.

The microprocessor registers, instructions and memory spaces are mostly 16-bit oriented. There are 3 ISA register files:

- $r0-$r15, 16-bit general-purpose registers, for arithmetic and addressing
  - $r0: read-only and hardwired to 0
  - $r1-$r15: read/write
- $p0-$p15, 1-bit predicate registers, for conditional execution
  - $p0: read/write
  - $p1: read only and hardwired to !$p0
  - $p2-$p14: read/write
  - $p15: read-only and hardwired to 1
- $sr0-$sr63, 16-bit special registers
  - $sr0/$asel: A neighbour read selection [VP2 only] [vdec/vuc/vreg.txt]
  - $sr1/$bsel: B neighbour read selection [VP2 only] [vdec/vuc/vreg.txt]
  - $sr2/$spidx: [sub]partition selection [vdec/vuc/vreg.txt]
  - $sr3/$baddr: B neighbour read address [VP2 only] [vdec/vuc/vreg.txt]
  - $sr3/$absel: A and B neighbour selection [VP3+ only] [vdec/vuc/vreg.txt]
  - $sr4/$h2v: host to vμc scratch register [vdec/vuc/isa.txt]
  - $sr5/$v2h: vμc to host scratch register [vdec/vuc/isa.txt]
  - $sr6/$stat: a control/status register [vdec/vuc/isa.txt]
  - $sr7/$parm: video parameters [vdec/vuc/vreg.txt]
  - $sr8/$pc: program counter [vdec/vuc/isa.txt]
  - $sr9/$cspos: call stack position [vdec/vuc/isa.txt]
  - $sr10/$cstop: call stack top [vdec/vuc/isa.txt]
  - $sr11/$rpitab: RPI lut pointer [VP2 only] [vdec/vuc/vreg.txt]
  - $sr12/$lhi: long arithmetic high word [vdec/vuc/isa.txt]
  - $sr13/$llo: long arithmetic low word [vdec/vuc/isa.txt]
  - $sr14/$pred: alias of $p register file [vdec/vuc/isa.txt]
  - $sr15/$icnt: cycle counter [vdec/vuc/isa.txt]
  - $sr16/$mvxl0: motion vector L0 X component [vdec/vuc/vreg.txt]
  - $sr17/$mvyl0: motion vector L0 Y component [vdec/vuc/vreg.txt]
  - $sr18/$mvxl1: motion vector L1 X component [vdec/vuc/vreg.txt]
  - $sr19/$mvyl1: motion vector L1 Y component [vdec/vuc/vreg.txt]
  - $sr20/$refl0: L0 refidx [vdec/vuc/vreg.txt]
  - $sr21/$refl1: L1 refidx [vdec/vuc/vreg.txt]

  - $sr22/$rpil0: L0 RPI [vdec/vuc/vreg.txt]
  - $sr23/$rpil1: L1 RPI [vdec/vuc/vreg.txt]
  - $sr24/$mbflags: macroblock flags [vdec/vuc/vreg.txt]
  - $sr25/$qpy: luma quantiser and intra chroma pred mode [vdec/vuc/vreg.txt]
  - $sr26/$qpc: chroma quantisers [vdec/vuc/vreg.txt]
  - $sr27/$mbpart: macroblock partitioning schema [vdec/vuc/vreg.txt]
  - $sr28/$mbxy: macroblock X and Y position [vdec/vuc/vreg.txt]
  - $sr29/$mbaddr: macroblock address [vdec/vuc/vreg.txt]
  - $sr30/$mbtype: macroblock type [vdec/vuc/vreg.txt]
  - $sr31/$submbtype: submacroblock types [VP2 only] [vdec/vuc/vreg.txt]
  - $sr31/???: ??? [XXX] [VP3+ only] [vdec/vuc/vreg.txt]
  - $sr32/$amvxl0: A neighbour's $mvxl0 [vdec/vuc/vreg.txt]
  - $sr33/$amvyl0: A neighbour's $mvyl0 [vdec/vuc/vreg.txt]
  - $sr34/$amvxl1: A neighbour's $mvxl1 [vdec/vuc/vreg.txt]
  - $sr35/$amvyl1: A neighbour's $mvyl1 [vdec/vuc/vreg.txt]
  - $sr36/$arefl0: A neighbour's $refl0 [vdec/vuc/vreg.txt]
  - $sr37/$arefl1: A neighbour's $refl1 [vdec/vuc/vreg.txt]
  - $sr38/$arpil0: A neighbour's $rpil0 [vdec/vuc/vreg.txt]
  - $sr39/$arpil1: A neighbour's $rpil1 [vdec/vuc/vreg.txt]
  - $sr40/$ambflags: A neighbour's $mbflags [vdec/vuc/vreg.txt]
  - $sr41/$aqpy: A neighbour's $qpy [VP2 only] [vdec/vuc/vreg.txt]
  - $sr42/$aqpc: A neighbour's $qpc [VP2 only] [vdec/vuc/vreg.txt]
  - $sr48/$bmvxl0: B neighbour's $mvxl0 [vdec/vuc/vreg.txt]
  - $sr49/$bmvyl0: B neighbour's $mvyl0 [vdec/vuc/vreg.txt]
  - $sr50/$bmvxl1: B neighbour's $mvxl1 [vdec/vuc/vreg.txt]
  - $sr51/$bmvyl1: B neighbour's $mvyl1 [vdec/vuc/vreg.txt]
  - $sr52/$brefl0: B neighbour's $refl0 [vdec/vuc/vreg.txt]
  - $sr53/$brefl1: B neighbour's $refl1 [vdec/vuc/vreg.txt]
  - $sr54/$brpil0: B neighbour's $rpil0 [vdec/vuc/vreg.txt]
  - $sr55/$brpil1: B neighbour's $rpil1 [vdec/vuc/vreg.txt]
  - $sr56/$bmbflags: B neighbour's $mbflags [vdec/vuc/vreg.txt]
  - $sr57/$bqpy: B neighbour's $qpy [vdec/vuc/vreg.txt]
  - $sr58/$bqpc: B neighbour's $qpc [vdec/vuc/vreg.txt]

There are 7 address spaces the vμc can access:

- D[] - user data [vdec/vuc/isa.txt]

- PWT[] - pred weight table data, read-only. This space is filled when a packet of type 4 is read from the MBRING. Byte-addressed, 0x200 bytes long, loads are in byte units.
- VP[] - VPRING output data, write-only. Data stored here will be written to VPRING_DEBLOCK and VPRING_CTRL when corresponding commands are invoked. Byte-addressed, 0x400 bytes long. Stores are in byte or word units depending on the address.
- MVSI[] - MVSURF input data [read-only] [vdec/vuc/mvsurf.txt]
- MVSO[] - MVSURF output data [write-only] [vdec/vuc/mvsurf.txt]
- B6[] - io address space? [XXX]
- B7[] - io address space? [XXX]

The vμc code resides in the code space, separate from the above spaces. The code space is a dedicated SRAM of 0x800 instruction words. An instruction word consists of 40 bits on VP2, 30 bits on VP3.

The delays

The vμc lacks interlocks - on every cycle when vμc microcprocessor is active and not sleeping/waiting, one instruction begins execution. Most instructions finish in one cycle. However, when an instruction takes more than one cycle to finish, vμc will continue to fetch and execute subsequent instructions even if they have dependencies on the current instruction - it is thus required to manually insert nops in the code or schedule instructions to avoid such situations.

An X-cycle instruction happens in three phases:

- cycle 0: source read - the inputs to the instruction are gathered
- cycles 0..(X-1): result computation -
- cycle X: destination writeout - the results are stored into the destination registers

For example, add $r1 $r2 $r3 is a 1-cycle instruction. On cycle 0, the sources are read and the result is computed. On cycle 1, in parallel with executing the next instruction, the result is written out to $r1.

The extra cycle for destination writeout means that, in general, it's required to have at least 1 unrelated instruction between writing a register and reading it. However, vμc implements store-to-load forwarding for some common cases - the result value, which is already known on cycle (X-1), is transferred directly into the next instruction, if there's a match betwen the next instruction's source register index and current instruction's destination register index. Store-to-load forwarding happens in the following situations:

- all $r register reads and writes
- all $p register reads and writes, except by accessing them through $pred special register
- $lhi/$llo register reads and writes done implicitely by long arithmetic instructions

Store-to-load forwarding does NOT happen in the following situations:

- $sr register reads and writes

Example 1:

:: add $r1 $r2 $r3 add $r4 $r1 $r5

No delay needed, store-to-load forwarding happens:

- cycle 0: $r2 and $r3 read, $r2+$r3 computed
- cycle 1: $r5 read, previous result read due to l-t-s forwarding match for $r1, prev+$r5 computed, previous result written to $r1
- cycle 2: next instruction begins execution, insn 1 result written to $r5

Example 2 [missing delay]:

:: add $mvxl0 $r2 $r3 add $r4 $mvxl0 $r5

Delay needed, but not supplied - store-to-load forwarding doesn't happen and old value is read:

- cycle 0: $r2 and $r3 read, $r2+$r3 computed
- cycle 1: $mvxl0 and $r5 read, $mvxl0+$r5 computed, previous result written to $mvxl0
- cycle 2: next instruction begins execution, insn 1 result written to $r5

Code is equivalent to:

```
$r4 = $mvxl0 + $r5;
$mvxl0 = $r2 + $r3;
```

Example 3 [proper delay]:

:: add $mvxl0 $r2 $r3 nop add $r4 $mvxl0 $r5

Delay needed and supplied:

- cycle 0: $r2 and $r3 read, $r2+$r3 computed
- cycle 1: nop executes, previous result written to $mvxl0
- cycle 2: new $mvxl0 and $r5 read, $mvxl0+$r5 computed
- cycle 3: next instruction begins execution, insn 2 result written to $r5

Code is equivalent to:

```
$mvxl0 = $r2 + $r3;
$r4 = $mvxl0 + $r5;
```

Since long-running instructions use execution units during their execution, it's usually forbidden to launch other instructions using the same execution units until the first instruction is finished. When such execution unit conflict happens, the old instruction is aborted.

It is possible that two instructions with different write delays will try to perform a register write in the same cycle (e.g. ld-nop-mov sequence). If the write destinations are different, both writes will happen as expected. If the write destinations are the same, destination carries the value of the last write.

The branch instructions take two cycles to finish - the instruction after the jump [the delay slot] is executed regardless of whether the jump is taken or not.

The opcode format

The opcode bits are:

- 0-4: opcode selection [OP]
- 5-6, base opcodes: predicate output mode [POM]
  - 00: $p &= predicate output
  - 01: $p |= predicate output
  - 10: $p = predicate output
  - 11: predicate output discarded
- 7, base opcodes: predicate output negation flag [PON]
- 5-7, special opcodes: special opcode class selection [OC]

  - 000: control flow
  - 001: io control
  - 010: predicate manipulation
  - 100: load/store
  - 101: multiplication
- 8-11: source 1 [SRC1]
- 12-15: source 2 [SRC2]
- 16-19: destionation [DST]
- 8-18: branch target [BTARG]
- 20-23: predicate [PRED]
- 24-25: extra bits for immediate and $sr [EXT]
- 26: opcode type 0 [OT0]
- 27: source 2 immediate flag [IMMF]
- 28: opcode type 1 [OT1]
- 29: predicate enable flag [PE]
- 30-32: relative branch predicate [RBP] - VP2 only
- 33: relative branch predicate negation flag [RBN] - VP2 only
- 34-39: relative branch target [RBT] - VP2 only

On VP2, a single instruction word holds two instruction slots - the normal instruction slot in bits 0-29, and the relative branch instruction slot in bits 30-39. When the instruction is executed, both instruction slots are executed simultanously and independently.

The relative branch slot can hold only one type of instruction, which is the relative branch. The main slot can hold all other types of instructions. It's possible to encode two different jumps in one opcode by utilising both the branch slot and the main instruction slot for a branch. The branch will take place if any of the two branch conditions match. If both branch conditions match, the actual branch executed is the one in the main slot.

On VP3+, the relative branch slot no longer exists, and the main slot makes up the whole instruction word.

There are two major types of opcodes that can be stored in the main slot: base opcodes and special opcodes. The type of instruction in the main slot is determined by OT0 and OT1 bits:

- OT0 = 0, OT1 = 0: base opcode, $r destination, $r source 1
- OT0 = 1, OT1 = 0: base opcode, $r destination, $sr source 1
- OT0 = 0, OT1 = 1: base opcode, $sr destination, $r source 1
- OT0 = 1, OT1 = 1: special opcode

For base opcodes, the OP bits determine the final opcode:

- 00000: slct [slct form] select
- 00001: mov [mov form] move
- 00100: add [binary form] add
- 00101: sub [binary form] substract
- 00110: subr [binary form] substract reverse [VP2 only]

- 00110: avgs [binary form] average signed [VP3+ only]
- 00111: avgu [binary form] average unsigned [VP3+ only]
- 01000: setgt [set form] set if greater than
- 01001: setlt [set form] set if less than
- 01010: seteq [set form] set if equal to
- 01011: setlep [set form] set if less or equal and positive
- 01100: clamplep [binary form] clamp to less or equal and positive
- 01101: clamps [binary form] clamp signed
- 01110: sext [binary form] sign extension
- 01111: setzero [set form] set if both zero [VP2 only]
- 01111: div2s [unary form] divide by 2 signed [VP3+ only]
- 10000: bset [binary form] bit set
- 10001: bclr [binary form] bit clear
- 10010: btest [set form] bit test
- 10100: hswap [unary form] swap reg halves
- 10101: shl [binary form] shift left
- 10110: shr [binary form] shift right
- 10111: sar [binary form] shift arithmetic right
- 11000: and [binary form] bitwise and
- 11001: or [binary form] bitwise or
- 11010: xor [binary form] bitwise xor
- 11011: not [unary form] bitwise not
- 11100: lut [binary form] video LUT lookup
- 11101: min [binary form] minimum [VP3+ only]
- 11110: max [binary form] maximum [VP3+ only]

For special opcodes, the OC bits determine the opcode class, and OP bits further determine the opcode inside that class. The classes and opcodes are:

- OC 000: control flow
  - 00000: bra [branch form] branch
  - 00010: call [branch form] call
  - 00011: ret [simple form] return
  - 00100: sleep [simple form] sleep
  - 00101: wstc [immediate form] wait for status bit clear
  - 00110: wsts [immediate form] wait for status bit set
- OC 001: io control
  - 00000: clicnt [simple form] clear instruction counter

  - 00001: ??? [XXX] [simple form]
  - 00010: ??? [XXX] [simple form]
  - 00011: ??? [XXX] [simple form]
  - 00100: mbiread [simple form] macroblock input read
  - 00101: ??? [XXX] [simple form]
  - 00110: ??? [XXX] [simple form]
  - 01000: mbinext [simple form] macroblock input next
  - 01001: mvsread [simple form] MVSURF read
  - 01010: mvswrite [simple form] MVSURF write
  - 01011: ??? [XXX] [simple form]
  - 01100: ??? [XXX] [simple form]
- OC 010: predicate manipulation
  - xxx00: and [predicate form] and
  - xxx01: or [predicate form] or
  - xxx10: xor [predicate form] xor
  - xxx11: nop [simple form] no operation
- OC 100: load/store
  - xxxx0: st [store form] store
  - xxxx1: ld [load form] load
- OC 101: long arithmetic
  - 00000: lmulu [long binary form] long multiply unsigned
  - 00001: lmuls [long binary form] long multiply signed
  - 00010: lsrr [long unary form] long shift right with round
  - 00100: ladd [long unary form] long add [VP3+ only]
  - 01000: lsar [long unary form] long shift right arithmetic [VP3+ only]
  - 01100: ldivu [long unary form] long divide unsigned [VP4 only]

All main slot opcodes can be predicated by an arbitrary $p register. The PE bit enables predication. If PE bit is 1, the main slot instruction will only have an effect if the $p register selected by PRED field has value 1. Note that PE bit also has an effect on instruction format - longer immediates are allowed, and the predicate destination field changes.

Note that, for some formats, opcode fields may be used for multiple purposes. For example, mov instruction with PE=1 and IMMF=1 uses PRED bitfield both as the predicate selector and as the middle part of the immediate operand. Such formats should be avoided unless it can be somehow guaranteed that the value in the field will fit all purposes it's used for.

The base opcodes have the following operands:

- binary form: pdst, dst, src1, src2
- unary form: pdst, dst, src1
- set form: pdst, src1, src2
- slct form: pdst, dst, pred, src1, src2

- mov form: pdst, dst, lsrc

The operands and their encodings are:

- pdst: predicate destination - this operand is special, as it can be used in several modes. First, the instruction generates a boolean predicate result. Then, if PON bit is set, this output is negated. Finally, it is stored to a $p register in one of 4 modes:
  - POM = 00: $p &= output
  - POM = 01: $p |= output
  - POM = 10: $p = output
  - POM = 11: output is discarded

  The $p output register is:
  - PE = 0: $p register selected by PRED field
  - PE = 1: $p register selected by DST field
- dst: main destination
  - OT0 = 1 or OT1 = 0: $r register selected by DST field
  - OT0 = 0 and OT1 = 1: $sr register selected by DST [low bits] and EXT [high bits] fields
- pred - predicate source
  - all cases: $p register selected by PRED field
- src1: first source
  - OT0 = 0 or OT1 = 1: $r register selected by SRC1 field,
  - OT0 = 1 and OT1 = 0: $sr register selected by SRC1 [low bits] and EXT [high bits] fields.
- src2: second source
  - IMMF = 0: $r register selected by SRC2 field
  - IMMF = 1 and OT0 = OT1:. zero-extended 6-bit immediate value stored in SRC2 [low bits] and EXT [high bits] fields.
  - IMMF = 1 and OT0 != OT1: zero-extended 4-bit immediate value stored in SRC2 field.
- lsrc: long source
  - IMMF = 0: $r register selected by SRC2 field
  - IMMF = 1 and OT1 = 0:. zero-extended 14-bit immediate value stored in SRC1 [low bits], SRC2 [low middle bits], PRED [high middle bits] and EXT [high bits] fields.
  - IMMF = 1 and OT1 = 1:. zero-extended 12-bit immediate value stored in SRC1 [low bits], SRC2 [middle bits] and PRED [high bits] fields

The special opcodes have the following operands:

- simple form: [none]
- immediate form: imm4
- branch form: btarg
- predicate form: spdst, psrc1, psrc2
- store form: space[dst + src1 * 2], src2 [if IMMF is 0]
- store form: space[src1 + stoff], src2 [if IMMF is 1]

- load form: dst, space[src1 + ldoff] [if IMMF is 0]
- load form: dst, space[src1 + src2] [if IMMF is 1]
- long binary form: src1, src2
- long unary form: src2

The operands and their encodings are:

- src1, src2, dst: like for base opcodes
- imm4: 4-bit immediate
  - all cases: 4-bit immediate stored in SRC2 field
- btarg: code address
  - all cases: 11-bit immediate stored in BTARG field
- spdst: predicate destination
  - PE = 0: $p register selected by PRED field
  - PE = 1: $p register selected by DST field
- psrc1: predicate source 1, optionally negated
  - all cases: $p register selected by SRC1 field, negated if bit 3 of OP field is set
- psrc2: predicate source 2, optionally negated
  - all cases: $p register selected by SRC2 field, negated if bit 2 of OP field is set
- space: memory space selection, OP field bits 1-4:
  - 0000: D[]
  - 0001: PWT[] - ld only
  - 0010: VP[] - st only
  - 0100: MVSI[] - ld only
  - 0101: MVSO[] - st only
  - 0110: B6[]
  - 0111: B7[]
- stoff: store offset
  - PE = 0: 10-bit zero-extended immediate stored in DST [low bits], PRED [middle bits] and EXT [high bits] fields
  - PE = 1: 6-bit zero-extended immediate stored in DST [low bits] and EXT [high bits] fields
- ldoff: load offset
  - PE = 0: 10-bit zero-extended immediate stored in SRC2 [low bits], PRED [middle bits] and EXT [high bits] fields
  - PE = 1: 6-bit zero-extended immediate stored in SRC2 [low bits] and EXT [high bits] fields

The code space and execution control

The vμc executes instructions from dedicated code SRAM. The code SRAM is made of 0x800 cells, with each cell holding one opcode. Thus, a cell is 40 bits wide on VP2, 30 bits wide on VP3+. The code space is addressed in opcode units, with addresses 0-0x7ff. The only way to access the code space other than via executing instructions from it is through the code port:

BAR0 0x103288 / XLMI 0x0a200: CODE_CONTROL [VP2] BAR0 0x085440 / I[0x11000]: CODE_CONTROL [VP3+]

bits 0-10: ADDR, cell address to access by CODE_WINDOW bit 16: STATE, code execution control: 0 - code is being executed,

CODE_WINDOW doesn't work, 1 - microprocessor is halted, CODE_WINDOW is enabled

BAR0 0x10328c / XLMI 0x0a300: CODE_WINDOW [VP2] BAR0 0x085444 / I[0x11100]: CODE_WINDOW [VP3+]

Accesses the code space - see below

On VP3+, reading or writing the CODE_WINDOW register will cause a read/write of the code space cell selected by ADDR, with the cell value taken from / appearing at bits 0-29 of CODE_WINDOW. ADDR is auto-incremented by 1 with each access.

On VP2, since code space cells are 40 bits long, accessing a cell requires two accesses to CODE_WINDOW. The cell is divided into 32-bit low part and 8-bit high part. There is an invisible 1-bit flipflop that selects whether the high part or the low part will be accessed next. The low part is accessed first, then the high part. Writing CODE_CONTROL will reset the flipflop to the low part. Accessing CODE_WINDOW with the flipflop set to the low part will access the low part, then switch the flipflop to the high part. Accessing CODE_WINDOW with the flipflop set to the high part will access the high part [through bits 0-7 of CODE_WINDOW], switch the flipflop to the low part, and auto-increment ADDR by 1. In addition, writes through CODE_WINDOW are buffered - writing the low part writes a shadow register, writing the high part assembles it with the current shadow register value and writes the concatenated result to the code space.

The STATE bit is used to control vμc execution. This bit is set to 1 when the vμc is reset. When this bit is changed from 1 to 0, the vμc starts executing instructions starting from code address 0. When this bit is changed from 1 to 0, the vμc execution is halted.

The data space

D[] is a read-write memory space consisting of 0x800 16-bit cells. Every address in range 0-0x7ff corresponds to one cell. The D[] space is used for three purposes:

- to store general-purpose data by microcode/host and communicate between the microcode and the host
- to store the RPI table, a mapping from bitstream reference indices to hw surface indices [RPIs], used directly by hardware [vdec/vuc/vreg.txt]
- to store the REF table, a mapping from RPIs to surface VM addresses, used directly by hardware [VP3+] [vdec/vuc/vreg.txt]

On VP2, the D[] space can be accessed from the host directly by using the DATA window:

BAR0 0x103200 + (i >> 6) * 4 [index i & 0x3f] / XLMI 0x08000 + i * 4, i < 0x800: DATA[i] [VP2] Accesses the data space - low 16 bits of DATA[i] go to D[] cell i, high 16 bits are unused.

On VP3+, the DATA window also exists, but cells are accessed in pairs:

BAR0 0x085400 + (i >> 6) * 4 [index i & 0x3f] / I[0x10000 + i * 4], i < 0x400: DATA[i] [VP3+] Accesses the data space - low 16 bits of DATA[i] go to D[] cell i*2, high 16 bits go to D[] cell i*2+1.

The D[] space can be both read and written via the DATA window.

Instruction reference

In the pseudocode, all intermediate computation results and temporary variables are assumed to be infinite-precision signed integers: non-negative integers are padded at the left with infinite number of 0 bits, while negative integers are padded with infinite number of 1 bits.

When assigning a result to a finite-precision register, any extra bits are chopped off. When reading a value from a finite-precision register, it's padded with infinite number of 0 bits at the left by default. A sign-extension read, where the register value is padded with infinite number of copies of its MSB instead, is written as SEX(reg).

Operators used in the pseudocode behave as in C.

Some instructions are described elsewhere. They are:

- lut [vdec/vuc/vreg.txt]
- sleep [in $stat register description]
- wstc [in $stat register description]
- wsts [in $stat register description]
- clicnt [XXX]
- mbiread [vdec/vuc/vreg.txt]
- mbinext [vdec/vuc/vreg.txt]
- mvsread [vdec/vuc/mvsurf.txt]
- mvswrite [vdec/vuc/mvsurf.txt]

Data movement instructions: slct, mov mov sets the destination to the value of the only source. slct sets the destination to the value of one of the sources, as selected by a predicate.

Instruction: slct pdst, dst, pred, src1, src2 Opcode: base opcode, OP = 00000 Operation:

```
result = (pred ? src1 : src2);
dst = result;
pdst = result & 1;
```

Execution time: 1 cycle Predicate output: LSB of normal result

Instruction: mov pdst, dst, lsrc Opcode: base opcode, OP = 00001 Operation:

```
result = lsrc;
dst = result;
pdst = result & 1;
```

Execution time: 1 cycle Predicate output: LSB of normal result

Addition instructions: add, sub, subr, avgs, avgu add performs an addition of two 16-bit quantities, sub and subr perform substraction, subr with reversed order of operands. avgs and avgu compute signed and unsigned average of two sources, rounding up. If predicate output is used, the predicate is set to the lowest bit of the result.

Instructions:: add pdst, dst, src1, src2 OP=00100 sub pdst, dst, src1, src2 OP=00101 subr pdst, dst, src1, src2 OP=00110 [VP2 only] avgs pdst, dst, src1, src2 OP=00110 [VP3+ only] avgu pdst, dst, src1, src2 OP=00111 [VP3+ only]

Opcode: base opcode, OP as above Operation:

```
if (op == add) result = src1 + src2;
if (op == sub) result = src1 - src2;
if (op == subr) result = src2 - src1;
if (op == avgs) result = (SEX(src1) + SEX(src2) + 1) >> 1;
if (op == avgu) result = (src1 + src2 + 1) >> 1;
dst = result;
pdst = result & 1;
```

Execution time: 1 cycle Predicate output: LSB of normal result

Comparison instructions: setgt, setlt, seteq, setlep, setzero setgt, setlt, seteq perform signed >, <, == comparison on two source operands and return the result as pdst. setlep returns 1 if src1 is in range [0, src2]. All comparisons are signed 16-bit. setzero returns 1 if both src1 and src2 are equal to 0.

Instructions:: setgt pdst, src1, src2 OP=01000 setlt pdst, src1, src2 OP=01001 seteq pdst, src1, src2 OP=01010 setlep pdst, src1, src2 OP=01011 setzero pdst, src1, src2 OP=01111 [VP2 only]

Opcode: base opcode, OP as above Operation:

```
if (op == setgt) result = SEX(src1) < SEX(src2);
if (op == setlt) result = SEX(src1) > SEX(src2);
if (op == seteq) result = src1 == src2;
if (op == setlep) result = SEX(src1) <= SEX(src2) && SEX(src1) >= 0;
if (op == setzero) result = src1 == 0 && src2 == 0;
pdst = result;
```

Execution time: 1 cycle Predicate output: the comparison result

Clamping and sign extension instructions: clamplep, clamps, sext clamplep clamps src1 to [0, src2] range. clamps, like the xtensa instruction of the same name, clamps src1 to [-(1 << src2), (1 << src2) - 1] range, ie. to the set of (src2+1)-bit signed integers. sext, like the xteansa and falcon instructions of the same name, replaces bits src2 and up with a copy of bit src2, effectively doing a sign extension from a (src2+1)-bit signed number.

Instructions:: clamplep pdst, dst, src1, src2 OP=01100 clamps pdst, dst, src1, src2 OP=01101 sext pdst, dst, src1, src2 OP=01110

Opcode: base opcode, OP as above Operation:

```
if (op == clamplep) {
    result = src1;
    presult = 0;
    if (SEX(src1) < 0) {
        presult = 1;
        result = 0;
    }
    if (SEX(src1) > SEX(src2)) {
        presult = 1;
        result = src2;
    }
}
if (op == clamps) {
    bit = src2 & 0xf;
    result = src1;
    presult = 0;
    if (SEX(src1) < -(1 << bit)) {
        result = -(1 << bit);
```

```
        presult = 1;
    }
    if (SEX(src1) > (1 << bit) - 1) {
        result = (1 << bit) - 1;
        presult = 1;
    }
}
if (op == sext) {
    bit = src2 & 0xf;
    presult = src1 >> bit & 1;
    if (presult)
        result = jrc1 | -(1 << bit);
    else
        result = src1 & ((1 << bit) - 1);
}
dst = result;
pdst = presult;
```

Execution time: 1 cycle Predicate output:

clamplep, clamps: 1 if clamping happened sext: 1 if result < 0

Division by 2 instruction: div2s div2s divides a signed number by 2, rounding to 0.

Instructions:: div2s pdst, dst, src1 OP=01111 [VP3+ only]

Opcode: base opcode, OP as above Operation:

```
if (SEX(src1) < 0) {
    result = (SEX(src1) + 1) >> 1;
} else {
    result = src1 >> 1;
}
dst = result;
pdst = result < 0;
```

Execution time: 1 cycle Predicate output: 1 if result is negative

Bit manipulation instructions: bset, bclr, btest bset and bclr set or clear a single bit in a value. btest copies a selected bit to a $p register.

Instructions:: bset pdst, dst, src1, src2 OP=10000 bclr pdst, dst, src1, src2 OP=10001 btest pdst, src1, src2 OP=10010

Opcode: base opcode, OP as above Operation:

```
bit = src2 & 0xf;
if (op == bset) {
    result = src1 | 1 << bit;
    presult = result & 1;
    dst = result;
}
if (op == bclr) {
    dst = result = src1 & ~(1 << bit)
    presult = result & 1;
    dst = result;
}
if (op == btest) {
    presult = src1 >> bit & 1;
```

```
}
pdst = presult;
```

Execution time: 1 cycle Predicate output:

bset, bclr: bit 0 of the result btest: the selected bit

Swapping reg halves: hswap hswap, like the falcon instruction of the same name, rotates a value by half its size, which is always 8 bits for vμc.

Instructions:: hswap pdst, dst, src1 OP=10100

Opcode: base opcode, OP as above Operation:

```
result = src1 >> 8 | src1 << 8;
dst = result;
pdst = result & 1;
```

Execution time: 1 cycle Predicate output: bit 0 of the result

Shift instructions: shl, shr, sar shl does a left shift, shr does a logical right shift, sar does an arithmetic right shift.

Instructions:: shl pdst, dst, src1, src2 OP=10101 shr pdst, dst, src1, src2 OP=10110 sar pdst, dst, src1, src2 OP=10111

Opcode: base opcode, OP as above Operation:

```
shift = src2 & 0xf;
if (op == shl) {
    result = src1 << shift;
    presult = result >> 16 & 1;
}
if (op == shr) {
    result = src1 >> shift;
    if (shift != 0) {
        presult = presult = src1 >> (shift - 1) & 1;
    } else {
        presult = 0;
    }
}
if (op == sar) {
    result = SEX(src1) >> shift;
    if (shift != 0) {
        presult = presult = src1 >> (shift - 1) & 1;
    } else {
        presult = 0;
    }
}
dst = result;
pdst = presult;
```

Execution time: 1 cycle Predicate output: the last bit shifted out

Bitwise instructions: and, or, xor, not No comment.

Instructions:: and pdst, dst, src1, src2 OP=11000 or pdst, dst, src1, src2 OP=11001 xor pdst, dst, src1, src2 OP=11010 not pdst, dst, src1 OP=11011

Opcode: base opcode, OP as above Operation:

```
if (op == and) result = src1 & src2;
if (op == or) result = src1 | src2;
if (op == xor) result = src1 ^ src2;
if (op == not) result = ~src1;
dst = result;
pdst = result & 1;
```

Execution time: 1 cycle Predicate output: bit 0 of the result

Minmax instructions: min, max These instructions perform the signed min/max operations.

Instructions:: min pdst, dst, src1, src2 OP=11101 [VP3+ only] max pdst, dst, src1, src2 OP=11110 [VP3+ only]

Opcode: base opcode, OP as above Operation:

```
if (op == min) which = (SEX(src2) < SEX(src1));
if (op == max) which = (SEX(src2) >= SEX(src1));
dst = (which ? src2 : src1);
pdst = which;
```

Execution time: 1 cycle Predicate output: 0 if src1 is selected as the result, 1 if src2 is selected

Predicate instructions: and, or, xor These instruction perform the corresponding logical ops on $p registers. Note that one of both inputs can be negates, as mentioned in psrc1/psrc2 operand description.

Instructions:: and spdst, psrc1, psrc2 OP=xxx00 or spdst, psrc1, psrc2 OP=xxx01 xor spdst, psrc1, psrc2 OP=xxx10

Opcode: special opcode with OC=010, OP as above. Note that bits 2 and 3 of OP are used for psrc1 and psrc2 negation flags.

Operation:: if (op == and) spdst = psrc1 & psrc2; if (op == or) spdst = psrc1 | psrc2; if (op == xor) spdst = psrc1 ^ psrc2;

Execution time: 1 cycle

No operation: nop Does nothing.

Instructions:: nop OP=xxx11

Opcode: special opcode with OC=010, OP as above. Operation:

```
/* nothing */
```

Execution time: 1 cycle

Long multiplication instructions: lmulu, lmuls These instructions perform signed and unsigned 16x11 -> 32 bit multiplication. src1 holds the 16-bit source, while low 11 bits of src2 hold the 11-bit source. The result is written to $lhi:$llo.

Instructions:: lmulu src1, src2 OP=00000 lmuls src1, src2 OP=00001

Opcode: special opcode with OC=101, OP as above Operation:

```
if (op == umul) {
    result = src1 * (src2 & 0x7ff);
if (op == smul) {
    /* sign extension from 11-bit number */
```

```
    s2 = src2 & 0x7ff;
    if (s2 & 0x400)
        s2 -= 0x800;
    result = SEX(src1) * s2;
}
$llo = result;
$lhi = result >> 16;
```

Execution time: 3 cycles Execution unit conflicts: lmulu, lmuls, lsrr, ladd, lsar, ldivu

Long arithmetic unary instructions: lsrr, ladd, lsar, ldivu These instruction operate on the 32-bit quantity in $lhi:$llo. ladd adds a signed 16-bit quantity to it. lsar shifts it right arithmetically by a given amount. ldivu does an unsigned 32/16 -> 32 division. lsrr divides it by 2^(src2 + 1), rounding to nearest with ties rounded up.

Instructions:: lsrr src2 OP=00010 ladd src2 OP=00100 [VP3+ only] lsar src2 OP=01000 [VP3+ only] ldivu src2 OP=01100 [VP4 only]

Opcode: special opcode with OC=101, OP as above Operation:

```
val = SEX($lhi) << 16 | $llo;
if (op == lsrr) {
    bit = src2 & 0x1f;
    val += 1 << bit;
    val >>= (bit + 1);
}
if (op == ladd) val += SEX(src2);
if (op == lsar) val >>= src2 & 0x1f;
if (op == ldivu)
    val &= 0xffffffff;
    if (src2)
        val /= src2;
    else
        val = 0xffffffff;
}
$llo = val;
$lhi = val >> 16;
```

Execution time: lsrr: 1 cycle ladd: 1 cycle lsar: 1 cycle ldivu: 34 cycles

Execution unit conflicts: lmulu, lmuls, lsrr, ladd, lsar, ldivu

Control flow instructions: bra, call, ret
Todo

write me

- **Flow:**

  0x00: [bra TARGET]

  bra IMM?

  Branch to address. Delay: 1 instruction

  0x02: [call TARGET]

  call IMM?

  XXX: stack and calling convention

0x03: [ret]

ret

TODO: delay (blob: 1) XXX: stack and calling convention

Memory access instructions: ld, st These instructions load and store values from/to one of the memory spaces available to the vμc microprocessor. The exact semantics of such operation depend on the space being accessed.

Instructions:: st space[dst + src1 * 2], src2 OP=xxxx0 [if IMMF is 0] st space[src1 + stoff], src2 OP=xxxx0 [if IMMF is 1] ld dst, space[src1 + ldoff] OP=xxxx1 [if IMMF is 0] ld dst, space[src1 + src2] OP=xxxx1 [if IMMF is 1]

Opcode: Special opcode with OC=100, OP as above. Note that btis 1-4 of OP are used to select memory space.

Operation::

if (op == st) space.STORE(address, src2);

else dst = space.LOAD(address);

Execution time: ld: 3 cycles st: 1 cycle

The scratch special registers

The vμc has two 16-bit scratch registers that may be used for communication between vμc and the host [xtensa/falcon code counts at the host in this case]. One of them is for host -> vμc direction, the other for vμc -> host.

The host -> vμc register is called $h2v. It's RW on the host side, RO on vμc side. Writing this register causes bit 11 of $stat register to light up and stay up until $h2v is read on vμc side.

$sr4/$h2v: host->vμc 16-bit scratch register. Reading this register will clear bit 11 of $stat. This register is read-only.

BAR0 0x103290 / XLMI 0x0a400: H2V [VP2] BAR0 0x085450 / I[0x11400]: H2V [VP3+]

A read-write alias of $h2v. Does not clear $stat bit 11 when read. Writing sets bit 11 of $stat

$stat bit 11: $h2v write pending. This bit is set when H2V is written by host, cleared when $h2v is read by vμc.

The vμc -> host register is called $v2h. It's RW on the vμc side, RO on host side. Writing this register causes an interrupt to be triggered.

$sr5/$v2h: vμc->host 16-bit scratch register, read-write. Writing this register will trigger V2H vμc interrupt.

BAR0 0x103294 / XLMI 0x0a500: V2H [VP2] BAR0 0x085454 / I[0x11500]: V2H [VP3+]

A read-only alias of $v2h.

The $stat special register

Every bit in this register performs a different function. All of them can be read. For the ones that can be written, value 0 serves as a noop, while value 1 triggers some operation.

$sr6/$stat: Control and status register.

- bit 0 [VP2]: VPRING_DEBLOCK buffer 0 write trigger [vdec/vuc/vpring.txt]
- bit 1 [VP2]: VPRING_DEBLOCK buffer 1 write trigger [vdec/vuc/vpring.txt]
- bit 2 [VP2]: VPRING_CTRL buffer 0 write trigger [vdec/vuc/vpring.txt]
- bit 3 [VP2]: VPRING_CTRL buffer 1 write trigger [vdec/vuc/vpring.txt]

- bit 0 [VP3+]: ??? [XXX]
- bit 1 [VP3+]: ??? [XXX]
- bit 2 [VP3+]: ??? [XXX]
- bit 3 [VP3+]: ??? [XXX]
- bit 4: ??? [XXX]
- bit 5: mvsread done status [vdec/vuc/mvsurf.txt]
- bit 6: MVSURF_OUT full status [vdec/vuc/mvsurf.txt]
- bit 7: mvswrite busy status [vdec/vuc/mvsurf.txt]
- bit 8: ??? [XXX]
- bit 9: ??? [XXX]
- bit 10: macroblock input available [vdec/vuc/vreg.txt]
- bit 11: $h2v write pending [vdec/vuc/isa.txt]
- bit 12: watchdog triggered [vdec/vuc/isa.txt]
- bit 13 [VP4+?]: ??? [XXX]
- bit 14: user-controlled pulse PCOUNTER signal [vdec/vuc/perf.txt]
- bit 15: user-controlled continuousPCOUNTER signal [vdec/vuc/perf.txt]

Three special instructions are available that read $stat implicitely. sleep instruction switches to a low-power sleep mode until bit 10 or bit 11 is set. wstc instruction does a busy-wait until a selected bit in $stat goes to 0, wsts likewise waits until a selected bit goes to 1.

On VP3+, a read-only alias of $stat is available in the MMIO space:

BAR0 0x0854bc / I[0x12f00]: STAT Aliases $stat vμc register, read only.

Sleep instruction: sleep This instruction waits until a full macroblock has been read from the MBRING [ie. $stat bit 10 is set] or host writes $h2v register [ie. $stat bit 11 is set]. While this instruction is waiting, vμc microprocessor goes into a low power mode, and sends 0 on its "busy" signal, thus counting as idle.

Instructions:: sleep OP=00100

Opcode: special opcode with OC=001, OP as above Operation:

```
while (!($stat & 0xc00)) idle();
```

Execution time: as long as necessary, at least 1 cycle, blocks subsequent instructions until finished

Wait for status bit instructions: wstc, wsts These instructions wait for a given $stat bit to become 0 [wstc] or 1 [wsts]. Execution of all subsequent instructions is delayed until this happens.

Instructions:: wstc imm4 OP=00101 wsts imm4 OP=00110

Opcode: special opcode with OC=001, OP as above Operation:

```
while (($stat >> imm4 & 1) != (op == wsts));
```

Execution time: as long as necessary, at least 1 cycle, blocks subsequent instructions until finished

The watchdog counter

Todo

write me

Clear watchdog counter instruction: clicnt

Todo

write me

Misc special registers

This section describes various special registers that don't fit anywhere else.

$sr8/$pc: The program counter. When read, always returns the address of the instruction doing the read.

BAR0 0x10329c / XLMI 0x0a700: PC [VP2] BAR0 0x08545c / I[0x11700]: PC [VP3+]

A host-accessible alias of $pc. Shows the address of currently executing instruction.

$sr12/$lhi: long arithmetic high word register $sr13/$llo: long arithmetic low word register

These two registers together make a 32-bit quantity used in long arithmetic operations - see the documentation of long arithmetic instructions for details. These registers may be read after long arithmetic instructions to get their results. On VP3+, these registers may be written manually, on VP2 they're read-only and only modifiable by long arithmetic instructions.

$sr14/$pred: predicate register file alias

This register aliases the $p register file - bit X corresponds to $pX. The bits behave like the corresponding $p registers - bit 15 is read-only and always 1, while bit 1 is read-only and is always the negation of bit 0.

VP2/VP3/VP4 vμc MVSURF

1. MVSURF format
2. MVSURF_OUT setup
3. MVSURF_IN setup
4. MVSO[] address space
5. MVSI[] address space
6. Writing MVSURF: mvswrite
7. Reading MVSURF: mvsread

Introduction

H.264, VC-1 and MPEG4 all support "direct" prediction mode where the forward and backward motion vectors for a macroblock are calculated from co-located motion vector from the reference picture and relative ordering of the pictures. To implement it in vμc, intermediate storage of motion vectors and some related data is required. This storage is called MVSURF.

A single MVSURF object stores data for a single frame, or for two fields. Each macroblock takes 0x40 bytes in the MVSURF. The macroblocks in MVSURF are first grouped into macroblock pairs, just like in H.264 MBAFF frames. If the MVSURF corresponds to a single field, one macroblock of each pair is just left unused. The pairs are then stored in the MVSURF ordered first by X coordinate, then by Y coordinate, with no gaps.

The vμc has two MVSURF access ports: MVSURF_IN for reading the MVSURF of a reference picture [first picture in L1 list for H.264, the most recent I or P picture for VC-1 and MPEG4], MVSURF_OUT for writing the MVSURF of the current picture. Usage of both ports is optional - if there's no reason to use one of them [MVSURF_IN in non-B picture, or MVSURF_OUT in non-reference picture], it can just be ignored.

Both MVSURF_IN and MVSURF_OUT have to be set up via MMIO registers before use. To write data to MVSURF_OUT, it first has to be stored by the vμc into MVSO[] memory space, then the mvswrite instruction executed [while making sure the previous mvswrite instruction, if any, has already completed]. Reading MVSURF_IN is done by executing the mvsread instruction, waiting for its completion, then reading the MVSI[] memory space [or letting it be read implicitly by the vμc fixed-function hardware].

Note that MVSURF_OUT writes in units of macroblocks, while NVSURF_IN reads in units of macroblock pairs - see details below.

A single MVSURF entry, corresponding to a single macroblock, consists of:

- for the whole macroblock:
  - frame/field flag [1 bit]: for H.264, 1 if mb_field_decoding_flag set or in a field picture; for MPEG4, 1 if field-predicted macroblock
  - inter/intra flag [1 bit]: 1 for intra macroblocks
- for each partition:
  - RPI [5 bits]: the persistent id of the reference picture used for this subpartition and the top/bottom field selector, if applicable - same as the $rpil0/$rpil1 value.
- for each subpartition of each partition:
  - X component of motion vector [14 bits]
  - Y component of motion vector [12 bits]
  - zero flag [1 bit]: set if both components of motion vector are in -1..1 range and refIdx [not RPI] is 0 - partial term used in H.264 colZeroFlag computation

For H.264, the RPI and motion vector are from the partition's L0 prediction if present, L1 otherwise. Since vμc was originally designed for H.264, a macroblock is always considered to be made of 4 partitions, which in turn are made of 4 subpartitions each - if macroblock is more coarsely subdivided, each piece of data is duplicated for all covered 8x8 partitions and 4x4 subpartitions. Partitions and subpartitions are indexed in the same way as for $spidx.

MVSURF format

A single macroblock is represented by 0x10 32-bit LE words in MVSURF. Each word has the following format [i refers to word index, 0-15]:

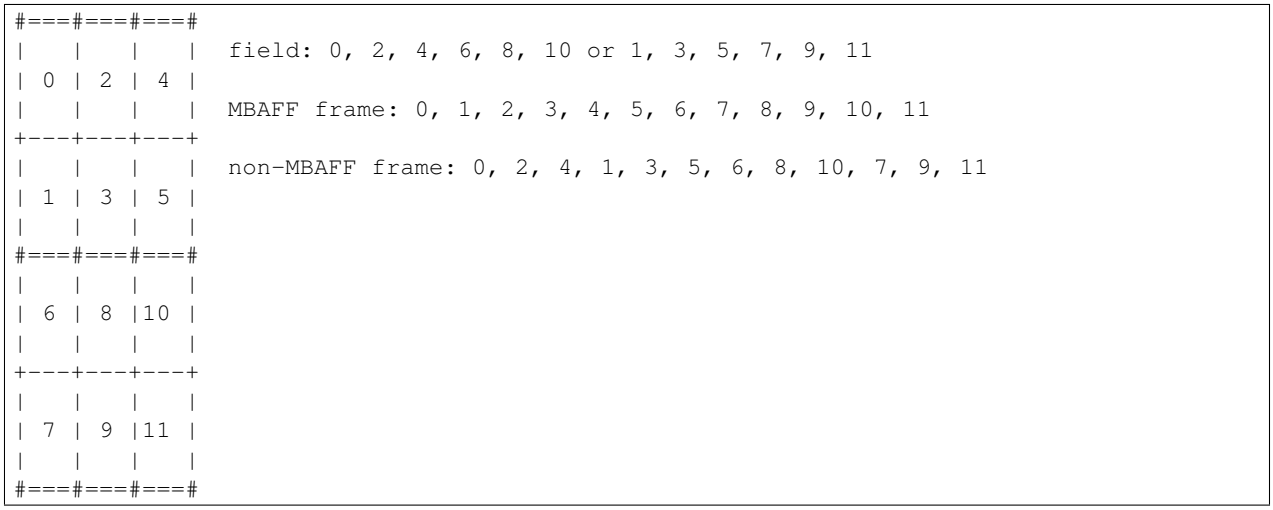
- bits 0-13, each word: X component of motion vector for subpartition i.
- bits 14-25, each word: Y component of motion vector for subpartition i.
- bits 26-30, word 0, 4, 8, 12: RPI for partition i>>2.
- bit 26, word 1, 5, 9, 13: zero flag for subpartition i-1
- bit 27, word 1, 5, 9, 13: zero flag for subpartition i
- bit 28, word 1, 5, 9, 13: zero flag for subpartition i+1

- bit 29, word 1, 5, 9, 13: zero flag for subpartition i+2
- bit 26, word 15: frame/field flag for the macroblock
- bit 27, word 15: inter/intra flag for the macroblock

MVSURF_OUT setup

The MVSURF_OUT has three different output modes:

- field picture output mode: each write writes one MVSURF macroblock and skips one MVSURF macroblock, each line is passed once
- MBAFF frame picture output mode: each write writes one MVSURF macroblock, each line is passed once
- non-MBAFF frame picture output mode: each write writes one MVSURF macroblock and skips one macroblock, each line is passed twice, with first pass writing even-numbered macroblocks, second pass writing odd-numbered macroblocks



The following registers control MVSURF_OUT behavior:

BAR0 0x1032f0 / XLMI 0x0bc00: MVSURF_OUT_OFFSET [VP2] The offset of MVSURF_OUT from the start of the MEMIF MVSURF port. The offset is in bytes and has to be divisible by 0x40.

BAR0 0x085490 / I[0x12400]: MVSURF_OUT_ADDR [VP3+] The address of MVSURF_OUT in falcon port #2, shifted right by 8 bits.

BAR0 0x1032f4 / XLMI 0x0bd00: MVSURF_OUT_PARM [VP2] BAR0 0x085494 / I[0x12500]: MVSURF_OUT_PARM [VP3+]

bits 0-7: WIDTH, length of a single pass in writable macroblocks bit 8: MBAFF_FRAME_FLAG, 1 if MBAFF frame picture mode enabled bit 9: FIELD_PIC_FLAG, 1 if field picture mode enabled

If neither bit 8 nor 9 is set, non-MBAFF frame picture mode is used. Bit 8 and bit 9 shouldn't be set at the same time.

BAR0 0x1032f8 / XLMI 0x0be00: MVSURF_OUT_LEFT [VP2] BAR0 0x085498 / I[0x12600]: MVSURF_OUT_LEFT [VP3+]

bits 0-7: X, the number of writable macroblocks left in the current pass bits 8-15: Y, the number of passes left, including the current pass

BAR0 0x1032fc / XLMI 0x0bf00: MVSURF_OUT_POS [VP2] BAR0 0x08549c / I[0x12700]: MVSURF_OUT_POS [VP3+]

bits 0-12: MBADDR, the index of the current macroblock from the start of MVSURF.

bit 13: PASS_ODD, 1 if the current pass is odd-numbered pass

All of these registers are RW by the host. LEFT and POS registers are also modified by the hardware when it writes macroblocks.

The whole write operation is divided into so-called "passes", which correspond to a line of macroblocks [field, non-MBAFF frame] or half a line [MBAFF frame]. When a macroblock is written to the MVSURF, it's written at the position indicated by POS.MBADDR, LEFT.X is decremented by 1, and POS.MBADDR is incremented by 1 [MBAFF frame] or 2 [field, non-MBAFF frame]. If this causes LEFT.X to drop to 0, a new pass is started, as follows:

- LEFT.X is reset to PARM.WIDTH
- LEFT.Y is decreased by 1
- POS.PASS_ODD is flipped
- if non-MBAFF frame picture mode is in use:
  - if PASS_ODD is 1, POS.MBADDR is decreased by PARM.WIDTH * 2 and bit 0 is set to 1
  - otherwise [PASS_ODD is 0], POS.MBADDR bit 0 is set to 0

When either LEFT.X or LEFT.Y is 0, writes to MVSURF_OUT are ignored.

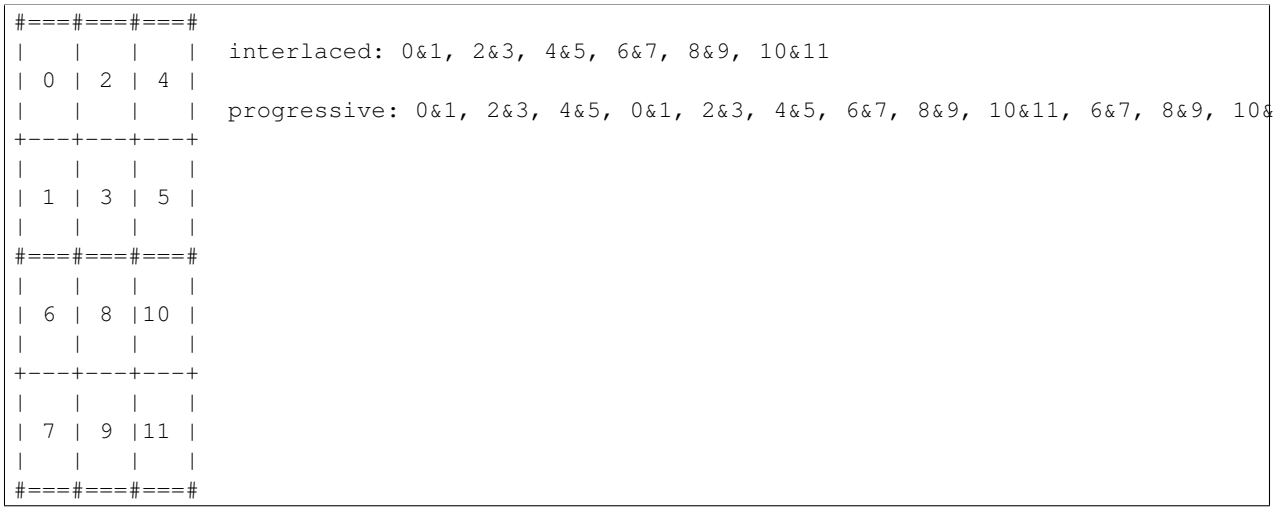
The MVSURF_OUT port has an output buffer of about 4 macroblocks - mvswrite will queue data into that buffer, and it'll auto-flush as MEMIF bandwidth allows. To determine whether the buffer is full [ie. if it's safe to queue any more data with mvswrite], use $stat bit 6:

$stat bit 6: MVSURF_OUT buffer full - no more space is available currently for writes, mvswrite instruction will be ignored and shouldn't be attempted until this bit drops to 0 [when MEMIF accepts more data].

MVSURF_IN setup

The MVSURF_OUT has two input modes:

- interlaced mode: used for field and MBAFF frame pictures, each read reads one macroblock pair, each line is passed once
- progressive mode: used for non-MBAFF frame pictures, each read reads one macroblock pair, each line is passed twice



The following registers control MVSURF_IN behavior:

BAR0 0x1032e0 / XLMI 0x0b800: MVSURF_IN_OFFSET [VP2] The offset of MVSURF_IN from the start of the MEMIF MVSURF port. The offset is in bytes and has to be divisible by 0x40.

BAR0 0x085480 / I[0x12000]: MVSURF_IN_ADDR [VP3+] The address of MVSURF_IN in falcon port #2, shifted right by 8 bits.

BAR0 0x1032e4 / XLMI 0x0b900: MVSURF_IN_PARM [VP2] BAR0 0x085484 / I[0x12100]: MVSURF_IN_PARM [VP3+]

bits 0-7: WIDTH, length of a single line in macroblock pairs bit 8: PROGRESSIVE, 1 if progressive mode enabled, 0 if interlaced mode

enabled

BAR0 0x1032e8 / XLMI 0x0ba00: MVSURF_IN_LEFT [VP2] BAR0 0x085488 / I[0x12200]: MVSURF_IN_LEFT [VP3+]

bits 0-7: X, the number of macroblock pairs left in the current line bits 8-15: Y, the number of lines left, including the current line

BAR0 0x1032ec / XLMI 0x0bb00: MVSURF_IN_POS [VP2] BAR0 0x08548c / I[0x12300]: MVSURF_IN_POS [VP3+]

bits 0-11: MBPADDR, the index of the current macroblock pair from the start of MVSURF.

bit 12: PASS, 0 for first pass, 1 for second pass

All of these registers are RW by the host. LEFT and POS registers are also modified by the hardware when it writes macroblocks.

The read operation is divided into lines. In interlaced mode, each line is read once, in progressive mode each line is read twice. A single read of a line is called a pass. When a macroblock pair is read, it's read from the position indicated by POS.MBPADDR, LEFT.X is decremented by 1, and POS.MBPADDR is incremented by 1. If this causes LEFT.X to drop to 0, a new line or a new pass over the same line is started:

- LEFT.X is reset to PARM.WIDTH
- if progressive mode is in use and POS.PASS is 0:
  - PASS is set to 1
  - POS.MBPADDR is decreased by PARM.WIDTH
- otherwise [interlaced mode is in use or PASS is 1]:
  - PASS is set to 0
  - LEFT.Y is decremented by 1

When either LEFT.X or LEFT.Y is 0, reads from MVSURF_IN will fail and won't affect MVSURF_IN registers in any way.

The MVSURF_IN port has an input buffer of 2 macroblock pairs. It will attempt to fill this buffer as soon as it's possible to read a macroblock pair [ie. LEFT.X and LEFT.Y are non-0]. For this reason, LEFT must always be the last register to be written when setting up MVSURF_IN. In addition, this makes it impossible to seamlessly switch to a new MVSURF_IN buffer without reading the previous one until the end.

The MVSURF_IN always operates on units of macroblock pairs. This means that the following special handling is necessary:

- field pictures: use interlaced mode, execute mvsread for each processed macroblock
- MBAFF frame pictures: use interlaced mode, execute mvsread for each processed macroblock pair [when starting to process the first macroblock in pair].
- non-MBAFF frame pictures: use progressive mode, execute mvsread for each processed macroblock

In all cases, Care must be taken to use the right macroblock from the pair in computations.

MVSO[] address space

MVSO[] is a write-only memory space consisting of 0x80 16-bit cells. Every address in range 0-0x7f corresponds to one cell. However, not all cells and not all bits of each cell are actually used. The usable cells are:

- MVSO[i * 8 + 0], i in 0..15: X component of motion vector for subpartition i
- MVSO[i * 8 + 1], i in 0..15: Y component of motion vector for subpartition i
- MVSO[i * 0x20 + j * 8 + 2], i in 0..3, j in 0..3: RPI of partition i, j is ignored
- MVSO[i * 8 + 3], i in 0..15: the "zero flag" for subpartition i
- MVSO[i * 0x20 + 4], i in 0..15: macroblock flags, i is ignored:
  - bit 0: frame/field flag
  - bit 1: inter/intra flag
- MVSO[i * 0x20 + 5], i in 0..15: macroblock partitioning schema, same format as $mbpart register, i is ignored [10 bits used]

If the address of some datum has some ignored fields, writing to any two addresses with only the ignored fields differing will actually access the same data.

MVSI[] address space

MVSI[] is a read-only memory space consisting of 0x100 16-bit cells. Every address in range 0-0xff corresponds to one cell. The cells are:

- MVSI[mb * 0x80 + i * 8 + 0], i in 0..15: X component of motion vector for subpartition i [sign extended to 16 bits]
- MVSI[mb * 0x80 + i * 8 + 1], i in 0..15: Y component of motion vector for subpartition i [sign extended to 16 bits]
- MVSI[mb * 0x80 + i * 0x20 + j * 8 + 2], i in 0..3, j in 0..3: RPI of partition i, j is ignored
- MVSI[mb * 0x80 + i * 8 + 3], i in 0..15: the "zero flag" for subpartition i
- MVSI[mb * 0x80 + i * 8 + 4 + j], i in 0..15, j in 0..3: macroblock flags, i and j are ignored:
  - bit 0: frame/field flag
  - bit 1: inter/intra flag

mb is 0 for the top macroblock in pair, 1 for the bottom macroblock.

If the address of some datum has some ignored fields, all addresses will alias and read the same datum.

Note that, aside of explicit loads from MVSI[], the MVSI[] data is also implicitely accessed by some fixed-function vμc hardware to calculate MV predictors and other values.

Writing MVSURF: mvswrite

Data is sent to MVSURF_OUT via the mvswrite instruction. A single invocation of mvswrite writes a single macroblock. The data is gathered from MVSO[] space. mvswrite is aware of macroblock partitioning and will use the partitioning schema to gather data from the right cells of MVSO[] - for instance, if 16x8 macroblock partitioning is used, only subpartitions 0 and 8 are used, and their data is duplicated for all 8x8/4x4 blocks they cover.

This instruction should not be used if MVSURF_OUT output buffer is currently full - the code should execute a wstc instruction on $stat bit 6 beforehand.

Note that this instruction takes 17 cycles to gather the data from MVSO[] space - in that time, MVSO[] contents shouldn't be modified. On cycles 1-16 of execution, $stat bit 7 will be lit up:

$stat bit 7: mvswrite MVSO[] data gathering in progress - this bit is set at the end of cycle 1 of mvswrite execution, cleared at the end of cycle 17 of mvswrite execution, ie. when it's safe to modify MVSO[]. Note that this means that the instruction right after mvswrite will still read 0 in this bit - to wait for mvswrite completion, use mvswrite; nop; wstc 7 sequence. This bit going back to 0 doesn't mean that MVSURF write is complete - it merely means that data has been gathered and queued for a write through the MEMIF.

If execution of this instruction causes the MVSURF_OUT buffer to become full, bit 6 of $stat is set to 1 on the same cycle as bit 7.

Instructions: mvswrite

Opcode: special opcode, OP=01010, OPC=001 Operation:

```
b32 tmp[0x10] = { 0 };
if (MVSURF_OUT.no_space_left())
        break;
$stat[7] = 1; /* cycle 1 */
if (MVSURF_OUT.full_after_next_mb())
        $stat[6] = 1;
b2 partlut[4] = { 0, 2, 1, 3 };
b10 mbpart = MVSO[5];
for (i = 0; i < 0x10; i++) {
        pidx = i >> 2;
        pmask = partlut[mbpart & 3];
        spmask = pmask << 2 | partlut[mbpart >> (pidx * 2 + 2) & 3];
        mpidx = pidx & pmask;
        mspidx = i & spmask;
        tmp[i] |= MVSO[mspidx * 8 + 0] | MVSO[mspidx * 8 + 1] << 14;
        tmp[(i & 0xc) | 1] |= MVSO[mspidx * 8 + 3] << (26 + (i & 3));
}
for (i = 0; i < 4; i++) {
        pidx = i >> 2;
        pmask = partlut[mbpart & 3];
        mpidx = pidx & pmask;
        tmp[i * 4] |= MVSO[mpidx * 0x20 + 2] << 26;
}
tmp[0xf] |= MVSO[4] << 26;
$stat[7] = 0; /* cycle 17 */
MVSURF_OUT.write(tmp);
```

Execution time: 18 cycles [submission to MVSURF_OUT port only, doesn't include the time needed by MVSURF_OUT to actually flush the data to memory]

Execution unit conflicts: mvswrite

Reading MVSURF: mvsread

Data is read from MVSURF_IN via the mvsread instruction. A single invocation of mvsread reads a single macroblock pair. The data is storred into MVSI[] space.

Since MVSURF resides in VRAM, which doesn't have a deterministic access time, this instruction may take an arbitrarily long time to complete the read. The read is done asynchronously and a $stat bit is provided to let the microcode know when it's finished:

$stat bit 5: mvsread MVSI[] write done - this bit is cleared on cycle 1 of mvsread execution and set by the mvsread instruction once data for a complete macroblock pair has been read and stored into MVSI[]. Note that this means that the instruction right after mvsread may stil read 1 in this bit - to wait for mvsread completion, use mvsread ; nop ; wsts 5 sequence. Also note that if the read fails because one of MVSURF_IN_LEFT fields is 0, this bit will never become 1. Also, note that the initial state of this bit after vμc reset is 0, even though no mvsread execution is in progress.

Instructions: mvsread

Opcode: special opcode, OP=01001, OPC=001 Operation:

```
b32 tmp[2][0x10];
$stat[5] = 0; /* cycle 1 */
MVSURF_IN.read(tmp); /* arbitrarily long */
for (mb = 0; mb < 2; mb++) {
        for (i = 0; i < 0x10; i++) {
                MVSI[mb * 0x80 + i * 8 + 0] = SEX(tmp[mb][i][0:13]);
                MVSI[mb * 0x80 + i * 8 + 1] = SEX(tmp[mb][i][14:25]);
                MVSI[mb * 0x80 + i * 8 + 2] = tmp[mb][i&0xc][26:30];
                MVSI[mb * 0x80 + i * 8 + 3] = tmp[mb][(i&0xc) | 1][26 + (i & 3)];
                MVSI[mb * 0x80 + i * 8 + 4] = tmp[mb][15][26:27];
        }
}
$stat[5] = 1;
```

Execution time: >= 37 cycles Execution unit conflicts: mvsread

VP2/VP3/VP4 vμc video registers

2. The video MMIO registers
3. $parm register
4. The RPIs and rpitab
5. Macroblock input: mbiread, mbinext
6. Table lookup instruction: lut

Introduction

Todo

write me

The video special registers

Todo

the following information may only be valid for H.264 mode for now

- $sr0: ??? controls $sr48-$sr58 (bits 0-6 when set separately) [XXX] [VP2 only]
- $sr1: ??? similar to $sr0 (bits 0-4, probably more) [XXX] [VP2 only]

- $sr2/$spidx: partition and subpartition index, used to select which [sub]partitions some other special registers access:
  - bits 0-1: subpartition index
  - bits 2-3: partition index

  Note that, for indexing purposes, each partition index is considered to refer to an 8x8 block, and each subpartition index to 4x4 block. If partition/subpartition size is bigger than that, the indices will alias. Thus, for 16x8 partitioning mode, $spidx equal to 0-7 wil select the top partition, $spidx equal to 8-15 will select the bottom one. For 8x16 partitioning, $spidx equal to 0-3 and 8-11 will select the left partition, 4-7 and 12-15 will select the right partition.
- $sr3: ??? bits 0-4 affect $sr32-$sr42 [XXX] [VP2 only]
- $sr3: ??? [XXX] [VP3+ only]
- $sr4/$h2v: a scratch register to pass data from host to vμc [see vdec/vuc/intro.txt]
- $sr5/$v2h: a scratch register to pass data from vμc to host [see vdec/vuc/intro.txt]
- $sr6/$stat: some sort of control/status reg, writing 0x8000 alternates values between 0x8103 and 0 [XXX]
  - bit 10: macroblock input available - set whenever there's a complete macroblock available from MBRING, cleared when mbinext instruction skips past the last currently available macroblock. Will break out of sleep instruction when set.
  - bit 11: $h2v modified - set when host writes H2V, cleared when $h2v is read by vμc, will break out of sleep instruction when set.
  - bit 12: watchdog hit - set 1 cycle after $icnt reaches WDCNT and it's not equal to 0xffff, cleared when $icnt or WDCNT is modified in any way.
- $sr7/$parm: sequence/picture/slice parameters required by vμc hardware [see vdec/vuc/intro.txt]
- $sr9/$cspos: call stack position, 0-8. Equal to the number of used entries on the stack.
- $sr10/$cstop: call stack top. Writing to this register causes the written value to be pushed onto the stack, reading this register pops a value off the stack and returns it.
- $sr11/$rpitab: D[] address of refidx -> dpb index translation table [VP2 only]
- $sr15/$icnt: instruction/cycle counter (?: check nops, effect of delays)
- $sr16/$mvxl0: sign-extended mvd_l0[$spidx][0] [input]
- $sr17/$mvyl0: sign-extended mvd_l0[$spidx][1] [input]
- $sr18/$mvxl1: sign-extended mvd_l1[$spidx][0] [input]
- $sr19/$mvyl1: sign-extended mvd_l1[$spidx][1] [input]
- $sr20/$refl0: ref_idx_l0[$spidx>>2] [input]
- $sr21/$refl1: ref_idx_l1[$spidx>>2] [input]
- $sr22/$rpil0: dpb index of L0 reference picture for $spidx-selected partition
- $sr23/$rpil1: dpb index of L1 reference picture for $spidx-selected partition
- $sr24/$mbflags:
  - bit 0: mb_field_decoding_flag [RW]
  - bit 1: is intra macroblock [RO]
  - bit 2: is I_NxN macroblock [RO]

  - bit 3: transform_size_8x8_flag [RW]
  - bit 4: ??? [XXX]
  - bit 5: is I_16x16 macroblock [RO]
  - bit 6: partition selected by $spidx uses L0 or Bi prediction [RO]
  - bit 7: partition selected by $spidx uses L1 or Bi prediction [RO]
  - bit 8: mb_field_decoding_flag for next macroblock [only valid if $sr6 bit 10 is set] [RO]
  - bit 9: mb_skip_flag for next macroblock [only valid if $sr6 bit 10 is set] [RO]
  - bit 10: partition selected by $spidx uses Direct prediction [RO]
  - bit 11: any partition of macroblock uses Direct prediction [RO]
  - bit 12: is I_PCM macroblock [RO]
  - bit 13: is P_SKIP macroblock [RO]
- $sr25/$qpy:
  - bits 0-5: mb_qp_delta [input] / QPy [output] [H.264]
  - bits 0-5: quantiser_scale_code [input and output] [MPEG1/MPEG2]
  - bits 8-11: intra_chroma_pred_mode, values:
    - 0: DC [input], **DC**_??? [output] [XXX]
    - 1: horizontal [input, output]
    - 2: vertical [input, output]
    - 3: plane [input, output]
    - 4: **DC**_??? [output]
    - 5: **DC**_??? [output]
    - 6: **DC**_??? [output]
    - 7: **DC**_??? [output]
    - 8: **DC**_??? [output]
    - 9: **DC**_??? [output]
    - 0xa: **DC**_??? [output]
- $sr26/$qpc:
  - bits 0-5: QPc for Cb [output] [H.264]
  - bits 8-13: QPc for Cr [output] [H.264]
- $sr27/$mbpart: - bits 0-1: macroblock partitioning type
  - 0: 16x16
  - 1: 16x8
  - 2: 8x16
  - 3: 8x8
  - bits 2-3: partition 0 subpartitioning type
  - bits 4-5: partition 0 subpartitioning type

  - bits 6-7: partition 0 subpartitioning type
  - bits 8-9: partition 0 subpartitioning type
    - 0: 8x8
    - 1: 8x4
    - 2: 4x8
    - 3: 4x4
- $sr28/$mbxy:
  - bits 0-7: macroblock Y position
  - bits 8-15: macroblock X position
- $sr29/$mbaddr:
  - bits 0-12: macroblock address
  - bit 15: first macroblock in slice flag
- $sr30/$mbtype: macroblock type, for H.264:
  - 0x00: I_NxN
  - 0x01: I_16x16_0_0_0
  - 0x02: I_16x16_1_0_0
  - 0x03: I_16x16_2_0_0
  - 0x04: I_16x16_3_0_0
  - 0x05: I_16x16_0_1_0
  - 0x06: I_16x16_1_1_0
  - 0x07: I_16x16_2_1_0
  - 0x08: I_16x16_3_1_0
  - 0x09: I_16x16_0_2_0
  - 0x0a: I_16x16_1_2_0
  - 0x0b: I_16x16_2_2_0
  - 0x0c: I_16x16_3_2_0
  - 0x0d: I_16x16_0_0_1
  - 0x0e: I_16x16_1_0_1
  - 0x0f: I_16x16_2_0_1
  - 0x10: I_16x16_3_0_1
  - 0x11: I_16x16_0_1_1
  - 0x12: I_16x16_1_1_1
  - 0x13: I_16x16_2_1_1
  - 0x14: I_16x16_3_1_1
  - 0x15: I_16x16_0_2_1
  - 0x16: I_16x16_1_2_1

  - 0x17: I_16x16_2_2_1
  - 0x18: I_16x16_3_2_1
  - 0x19: I_PCM
  - 0x20: P_L0_16x16
  - 0x21: P_L0_L0_16x8
  - 0x22: P_L0_L0_8x16
  - 0x23: P_8x8
  - 0x24: P_8x8ref0
  - 0x40: B_Direct_16x16
  - 0x41: B_L0_16x16
  - 0x42: B_L1_16x16
  - 0x43: B_Bi_16x16
  - 0x44: B_L0_L0_16x8
  - 0x45: B_L0_L0_8x16
  - 0x46: B_L1_L1_16x8
  - 0x47: B_L1_L1_8x16
  - 0x48: B_L0_L1_16x8
  - 0x49: B_L0_L1_8x16
  - 0x4a: B_L1_L0_16x8
  - 0x4b: B_L1_L0_8x16
  - 0x4c: B_L0_Bi_16x8
  - 0x4d: B_L0_Bi_8x16
  - 0x4e: B_L1_Bi_16x8
  - 0x4f: B_L1_Bi_8x16
  - 0x50: B_Bi_L0_16x8
  - 0x51: B_Bi_L0_8x16
  - 0x52: B_Bi_L1_16x8
  - 0x53: B_Bi_L1_8x16
  - 0x54: B_Bi_Bi_16x8
  - 0x55: B_Bi_Bi_8x16
  - 0x56: B_8x8
  - 0x7e: B_SKIP
  - 0x7f: P_SKIP
- $sr31/$submbtype: [VP2 only]
  - bits 0-3: sub_mb_type[0]
  - bits 4-7: sub_mb_type[1]

  - bits 8-11: sub_mb_type[2]
  - bits 12-15: sub_mb_type[3]
- $sr31: ??? [XXX] [VP3+ only]
- $sr32-$sr40: ??? affected by $sr3, unko21, read only [XXX]
- $sr41-$sr42: ??? affected by $sr3, unko21, read only [XXX] [VP2 only]
- $sr48-$sr58: ??? affected by writing $sr0 and $sr1, unko22, read only [XXX]

Table lookup instruction: lut

Performs a lookup of src1 in the lookup table selected by low 4 bits of src2. The tables are codec-specific and generated by hardware from the current contents of the video special registers.

Todo

recheck this instruction on VP3 and other codecs

Tables 0-3 are an alternate way of accessing H.264 inter prediction registers [$sr16-$sr23]. The table index is 1-bit. Index 0 selects the l0 register, index 1 selects the l1 register. Table 0 is $mvxl* registers, 1 is $mvyl*, 2 is $refl*, 3 is $rpil*.

Tables 4-7 behave like tables 0-3, except the lookup returns 0 if $mbtype is equal to 0x7f [P_SKIP].

Table 8, known as pcnt, is used to look up partition and subpartition counts. The index is 3-bit. Indices 0-3 return the subpartition count of corresponding partition, while indices 4-7 return the partition count of the macroblock.

Tables 9 and 10 are indexed in a special manner: the index selects a partition and a subpartition. Bits 0-7 of the index are partition index, bits 8-15 of the index are subpartition index. The partition and subpartition indices bahave as in the H.264 spec: valid indices are 0, 0-1, or 0-3 depending on the partitioning/subpartitioning mode.

Table 9, known as spidx, translates indices of the form given above into $spidx values. If both partition and subpartition index are valid for the current partitioning and subpartitioning mode, the value returned is the value that has to be poked into $spidx to access the selected [sub]partition. Otherwise, junk may be returned.

Table 10, known as pnext, advances the partition/subpartition index to the next valid subpartition or partition. The returned value is an index in the same format as the input index. Additionally, the predicate output is set if the partition index was not incremented [transition to the next subpartition of a partition], cleared if the partition index was incremented [transition to the first subpartition of the next partition].

Table 11, known as pmode, returns the inter prediction mode for a given partition. The index is 2-bit and selects the partition. If index is less then pcnt[4] and $mbtype is inter-predicted, returns inter prediction mode, otherwise returns 0. The prediction modes are:

- 0 direct
- 1 L0
- 2 L1
- 3 Bi

Tables 12-15 are unused and always return 0. [XXX: 12 used for VC-1 on VP3]

Instructions: lut pdst, dst, src1, src2 OP=11100

Opcode: base opcode, OP as above Operation:

```
/* helper functions */
int pcnt() {
    switch ($mbtype) {
        case 0:     /* I_NxN */
        case 0x19:  /* I_PCM */
            return 4;
        case 1..0x18:   /* I_16x16_* */
            return 1;
        case 0x20:  /* P_L0_16x16 */
            return 1;
        case 0x21:  /* P_L0_L0_16x8 */
        case 0x22:  /* P_L0_L0_8x16 */
            return 2;
        case 0x23:  /* P_8x8 */
        case 0x24:  /* P_8x8ref0 */
            return 4;
        case 0x40:  /* B_Direct_16x16 */
        case 0x41:  /* B_L0_16x16 */
        case 0x42:  /* B_L1_16x16 */
        case 0x43:  /* B_Bi_16x16 */
            return 1;
        case 0x44:  /* B_L0_L0_16x8 */
        case 0x45:  /* B_L0_L0_8x16 */
        case 0x46:  /* B_L1_L1_16x8 */
        case 0x47:  /* B_L1_L1_8x16 */
        case 0x48:  /* B_L0_L1_16x8 */
        case 0x49:  /* B_L0_L1_8x16 */
        case 0x4a:  /* B_L1_L0_16x8 */
        case 0x4b:  /* B_L1_L0_8x16 */
        case 0x4c:  /* B_L0_Bi_16x8 */
        case 0x4d:  /* B_L0_Bi_8x16 */
        case 0x4e:  /* B_L1_Bi_16x8 */
        case 0x4f:  /* B_L1_Bi_8x16 */
        case 0x50:  /* B_Bi_L0_16x8 */
        case 0x51:  /* B_Bi_L0_8x16 */
        case 0x52:  /* B_Bi_L1_16x8 */
        case 0x53:  /* B_Bi_L1_8x16 */
        case 0x54:  /* B_Bi_Bi_16x8 */
        case 0x55:  /* B_Bi_Bi_8x16 */
            return 2;
        case 0x56:  /* B_8x8 */
            return 4;
        case 0x7e:  /* B_SKIP */
            return 4;
        case 0x7f:  /* P_SKIP */
            return 1;
        /* in other cases returns junk */
    }
}
int spcnt(int idx) {
    if (pcnt() < 4) {
        return 1;
    } else if ($mbtype == 0 || $mbtype == 0x19) { /* I_NxN or I_PCM */
        return ($mbflags[3:3] ? 1 : 4); /* transform_size_8x8_flag */
    } else {
        smt = $submbtype >> (idx * 4)) & 0xf;
        /* XXX */
    }
```

```
}
int mbpartmode_16x8() {
    switch ($mbtype) {
        case 0x21: /* P_L0_L0_16x8 */
        case 0x44: /* B_L0_L0_16x8 */
        case 0x46: /* B_L1_L1_16x8 */
        case 0x48: /* B_L0_L1_16x8 */
        case 0x4a: /* B_L1_L0_16x8 */
        case 0x4c: /* B_L0_Bi_16x8 */
        case 0x4e: /* B_L1_Bi_16x8 */
        case 0x50: /* B_Bi_L0_16x8 */
        case 0x52: /* B_Bi_L1_16x8 */
        case 0x54: /* B_Bi_Bi_16x8 */
            return 1;
        default:
            return 0;
    }
}
int submbpartmode_8x4(int idx) {
    smt = $submbtype >> (idx * 4) & 0xf;
    switch(submbtype) {
        /* XXX */
    }
}
int mbpartpredmode(int idx) {
    /* XXX */
}
/* end of helper functions */
table = src2 & 0xf;
if (table < 8) {
    which = src1 & 1;
    switch (table & 3) {
        case 0: result = (which ? $mvxl1 : $mvxl0); break;
        case 1: result = (which ? $mvyl1 : $mvyl0); break;
        case 2: result = (which ? $refl1 : $refl0); break;
        case 3: result = (which ? $rpil1 : $rpil0); break;
    }
    if ((table & 4) && $mbtype == 0x7f)
        result = 0;
    presult = result & 1;
} else if (table == 8) {    /* pcnt */
    idx = src1 & 7;
    if (idx < 4) {
        result = spcnt(idx);
    } else {
        result = pcnt();
    }
} else if (table == 9 || table == 10) {
    pidx = src1 & 7;
    sidx = src1 >> 8 & 3;
    if (table == 9) {   /* spidx */
        if (mbpartmode_16x8())
            resp = (pidx & 1) << 1;
        else
            resp = (pidx & 3);
        if (submbpartmode_8x4(resp >> 2))
            ress = (sidx & 1) << 1;
        else
```

```
            ress = (sidx & 3);
        result = resp << 2 | ress;
        presult = result & 1;
    } else {        /* pnext */
        if (pidx < 4) {
            c = spcnt(idx);
        } else {
            c = pcnt();
        }
        ress = sidx + 1;
        if (ress >= c) {
            resp = (pidx & 3) + 1;
            ress = 0;
        } else {
            resp = pidx & 3;
        }
        result = ress << 8 | resp;
        presult = ress != 0;
    }
} else if (table == 10) {   /* pmode */
    result = mbpartpredmode(src1 & 3);
    presult = result & 1;
} else {
    result = 0;
    presult = 0;
}
dst = result;
pdst = presult;
```

Execution time: 1 cycle Predicate output:

Tables 0-9 and 11-15: bit 0 of the result Table 10: 1 if transition to next subpartition in a partition, 0 if transition to next partition

VP2 vμc output

Contents

Introduction

Todo

write me

vμc performance monitoring signals

Contents

Introduction

Todo

write me

2.11.3 VP2 video decoding

Contents:

VP2 xtensa processors

Todo

write me

Configured options:

- Code Density Option
- Loop Option
- 16-bit Integer Multiply Option
- Miscellaneous Operations Option: - InstructionCLAMPS = 0 - InstructionMINMAX = 1 - InstructionNSA = 0 - InstructionSEXT = 0
- Boolean Option
- Exception Option - NDEPC = 1 - ResetVector = 0xc0000020 - UserExceptionVector = 0xc0000420 - KernelExceptionVector = 0xc0000600 - DoubleExceptionVector = 0xc0000a00
- Interrupt Option - NINTERRUPT = 10 - INTTYPE[0]: Timer - INTTYPE[1]: Timer - INTTYPE[2]: Level - INTTYPE[3]: XXX Level/Edge/WriteErr - INTTYPE[4]: NMI - INTTYPE[5]: Level - INTTYPE[6]: Level - INTTYPE[7]: Level - INTTYPE[8]: Level - INTTYPE[9]: Level
- High-priority Interrupt Option - NLEVEL: 6 - LEVEL[0]: 1 - LEVEL[1]: 1 - LEVEL[2]: 2 - LEVEL[3]: 3 - LEVEL[4]: 7 - LEVEL[5]: 4 - LEVEL[6]: 5 - LEVEL[7]: 5 - LEVEL[8]: 5 - LEVEL[9]: 5 - EXCMLEVEL: 1 - NNMI: 1 - InterruptVector[2] = 0xc0000b40 - InterruptVector[3] = 0xc0000c00 - InterruptVector[4] = 0xc0000d20 - InterruptVector[5] = 0xc0000e00 - InterruptVector[6] = 0xc0000f00 - InterruptVector[7] = 0xc0001000
- Timer Interrupt Option - NCOMPARE = 2 - TIMERINT[0]: 0 - TIMERINT[1]: 1
- Instruction Cache Option - InstCacheWayCount: 3 - InstCacheLineBytes: 0x20 - InstCacheBytes: 0x3000
- Instruction Cache Test Option
- Instruction Cache Index Lock Option

- Data Cache Option - DataCacheWayCount: 2 - DataCacheLineBytes: 0x20 - DataCacheBytes: 0x1000 - IsWriteback: Yes
- Data Cache Test Option
- Data Cache Index Lock Option
- XLMI Option - XLMIBytes = 256kB - XLMIPAddr = 0xcffc0000
- Region Protection Option
- Windowed Register Option - WindowOverflow4 = 0xc0000800 - WindowUnderflow4 = 0xc0000840 - WindowOverflow8 = 0xc0000880 - WindowUnderflow8 = 0xc00008c0 - WindowOverflow12 = 0xc0000900 - WindowUnderflow12 = 0xc0000940 - NAREG = 32
- Processor Interface Option
- Debug Option - DEBUGLEVEL = 6 - NIBREAK = 2 - NDBREAK = 2 - SZICOUNT = 32 - OCD: XXX
- Trace Port Option? [XXX]

VLD: variable length decoding

Contents

Introduction

The VLD is the first stage of the VP2 decoding pipeline. It is part of PBSP and deals with decoding the H.264 bitstream into syntax elements.

The input to the VLD is the raw H.264 bitstream. The output of VLD is MBRING, a ring buffer structure storing the decoded syntax elements in the form of word-aligned packets.

The VLD only deals with parsing the NALs containing the slice data - the remaining NAL types are supposed to be parsed by the host. Further, the hardware can only parse pred_weight_table and slice_data elements efficiently - the remaining parts of the slice NAL are supposed to be parsed by the firmware controlling the VLD in a semi-manual manner: the VLD provides commands that parse single syntax elements.

The following H.264 profiles are supported:

- Constrained Baseline
- Baseline [only in single-macroblock mode if FMO used - see below]
- Main
- Progressive High
- High
- Multiview High
- Stereo High

The limitations are:

- max picture width and height: 128 macroblocks
- max macroblocks in picture: 8192

Todo

width/height max may be 255?

There are two modes of operation that VLD can be used with: single-macroblock mode and whole-slice mode. In the single-macroblock mode, parsing for each macroblock has to be manually triggered by the firmware. In whole-slice mode, the firmware triggers processing of a whole slice, and the hardware automatically iterates over all macroblocks in the slice. However, whole-slice mode doesn't support Flexible Macroblock Ordering aka. slice groups. Thus, single-macroblock mode has to be used for sequences with non-zero value of num_slice_groups_minus1.

The VLD keeps extensive hidden internal state, including:

- pred_weight_table data, to be prepended to the next emitted macroblock
- bitstream position, zero byte count [for escaping], and lookahead buffer
- CABAC valMPS, pStateIdx, codIOFfset, codIRange state
- previously decoded parts of macroblock data, used for CABAC and CAVLC context selection algorithms
- already queued but not yet written MBRING output data

The registers

The VLD registers are located in PBSP XLMI space at addresses 0x00000:0x08000 [BAR0 addresses 0x103000:0x103200]. They are:

XLMI	MMIO	Name	Description
0x00000	0x103000	PARM_0	*parameters from sequence/picture parameter structs and the slice header*
0x00100	0x103004	PARM_1	*parameters from sequence/picture parameter structs and the slice header*
0x00200	0x103008	MB_POS	*position of the current macroblock*
0x00300	0x10300c	COMMAND	*writing executes a VLD command*
0x00400	0x103010	STATUS	*shows busy status of various parts of the VLD*
0x00500	0x103014	RESULT	*result of a command*
0x00700	0x10301c	INTR_EN	*interrupt enable mask*
0x00800	0x103020	???	???
0x00900	0x103024	INTR	*interrupt status*
0x00a00	0x103028	RESET	*resets the VLD and its registers to initial state*
0x01000+i*0x100	0x103040+i*4	CONF[0:8]	*length and enable bit of stream buffer i*
0x01100+i*0x100	0x103044+i*4	OFFSET[0:8]	*offset of stream buffer i*
0x02000	0x103080	BITPOS	*the bit position in the stream*
0x04000	0x103100	OFFSET	*the MBRING offset*
0x04100	0x103104	HALT_POS	*the MBRING halt position*
0x04200	0x103108	WRITE_POS	*the MBRING write position*
0x04300	0x10310c	SIZE	*the MBRING size*
0x04400	0x103110	TRIGGER	*writing executes MBRING commands*

Todo

reg 0x00800

Reset

The engine may be reset at any time by poking the RESET register.

BAR0 0x103028 / XLMI 0x00a00: RESET Any write will cause the VLD to be reset. All internal state is reset to default values. All user-writable registers are set to 0, except UNK8 which is set to 0xffffffff.

Parameter and position registers

The values of these registers are used by some of the VLD commands. PARM registers should be initialised with values derived from sequence parameters, picture parameters, and slice header. MB_POS should be set to the address of currently processed macroblock [for single-macroblock operation] or the first macroblock of the slice [for whole-slice operation]. In whole-slice operation, MB_POS is updated by the hardware to the position of the last macroblock in the parsed slice.

For details on use of this information by specific commands, see their documentation.

BAR0 0x103000 / XLMI 0x00000: PARM_0

- bit 0: entropy_coding_mode_flag - set as in picture parameters
- bits 1-8: width_mbs - set to pic_width_in_mbs_minus1 + 1
- bit 9: mbaff_frame_flag - set to (mb_adaptive_frame_field_flag && !field_pic_flag)
- bits 10-11: picture_structure - one of:

  - 0: frame - set if !field_pic_flag
  - 1: top field - set if field_pic_flag && !bottom_field_flag
  - 2: bottom field - set if field_pic_flag && bottom_field_flag
- bits 12-16: nal_unit_type - set as in the slice NAL header [XXX: check]
- bit 17: constrained_intra_pred - set as in picture parameters [XXX: check]
- bits 18-19: cabac_init_idc - set as in slice header, for P and B slices
- bits 20-21: chroma_format_idc - if parsing auxiliary coded picture, set to 0, otherwise set as in sequence parameters
- bit 22: direct_8x8_inference_flag - set as in sequence parameters
- bit 23: transform_8x8_mode_flag - set as in picture parameters

BAR0 0x103004 / XLMI 0x00100: PARM_1

- bits 0-1: slice_type - set as in slice header
- bits 2-14: slice_tag - used to tag macroblocks in internal state with their slices, for determining availability status in CABAC/CAVLC context selection algorithms. See command description.
- bits 15-19: num_ref_idx_l0_active_minus1 - set as in slice header, for P and B slices
- bits 20-24: num_ref_idx_l1_active_minus1 - set as in slice header, for B slices
- bits 25-30: sliceqpy - set to (pic_init_qp_minus26 + 26 + slice_qp_delta)

BAR0 0x103008 / XLMI 0x00200: MB_POS

- bits 0-12: addr - address of the macroblock
- bits 13-20: x - x coordinate of the macroblock in macroblock units
- bits 21-28: y - y coordinate of the macroblock in macroblock units
- bit 29: first - 1 if the described macroblock is the first macroblock in its slice, 0 otherwise

Internal state for context selection

Both CAVLC and CABAC sometimes use decoded data of previous macroblocks in the slice to determine the decoding algorithm for syntax elements of the current macroblock. The VLD thus stores this data in its internal hidden memory.

Todo

what macroblocks are stored, indexing, tagging, reset state

For each macroblock, the following data is stored:

- slice_tag
- mb_field_decoding_flag
- mb_skip_flag
- mb_type
- coded_block_pattern
- transform_size_8x8_flag

- intra_chroma_pred_mode
- ref_idx_lX[i]
- mvd_lX[i][j]
- coded_block_flag for each block
- total_coeffs for each luma 4x4 / luma AC block

Todo

and availability status?

Additionally, the following data of the previous decoded macroblock [not indexed by macroblock address] is stored:

- mb_qp_delta

Interrupts

Todo

write me

BAR0 0x10301c / XLMI 0x00700: INTR_EN

- bit 0: UNK_INPUT_1
- bit 1: END_OF_STREAM
- bit 2: UNK_INPUT_3
- bit 3: MBRING_HALT
- bit 4: SLICE_DATA_DONE

BAR0 0x103024 / XLMI 0x00900: INTR

- bits 0-3: INPUT - 0: no interrupt pending - 1: UNK_INPUT_1 - 2: END_OF_STREAM - 3: UNK_INPUT_3 - 4: SLICE_DATA_DONE
- bit 4: MBRING_FULL

Stream input

Todo

RE and write me

MBRING output

Todo

write me

Command and status registers

Todo

write me

Command 0: GET_UE

Parameter: none

Result: the decoded value of parsed bitfield, or 0xffffffff if out of range

Parses one ue(v) element as defined in H.264 spec. Only elements in range 0..0xfffe [up to 31 bits in the bitstream] are supported by this command. If the next bits of the bitstream are a valid ue(v) element in supported range, the element is parsed, the bitstream pointer advances past it, and its parsed value is returned as the result. Otherwise, bitstream pointer is not modified and 0xffffffff is returned.

Operation:

```
if (nextbits(16) != 0) {
    int bitcnt = 0;
    while (getbits(1) == 0)
        bitcnt++;
    return (1 << bitcnt) - 1 + getbits(bitcnt);
} else {
    return 0xffffffff;
}
```

Command 1: GET_SE

Parameter: none

Result: the decoded value of parsed bitfield, or 0x80000000 if out of range

Parses one se(v) element as defined in H.264 spec. Only elements in range -0x7fff..0x7fff [up to 31 bits in the bitstream] are supported by this command. If the next bits of the bitstream are a valid se(v) element in supported range, the element is parsed, the bitstream pointer advances past it, and its parsed value is returned as the result. Otherwise, bitstream pointer is not modified and 0x80000000 is returned.

Operation:

```
if (nextbits(16) != 0) {
    int bitcnt = 0;
    while (getbits(1) == 0)
        bitcnt++;
    int tmp = (1 << bitcnt) - 1 + getbits(bitcnt);
    if (tmp & 1)
        return (tmp+1) >> 1;
    else
        return -(tmp >> 1);
} else {
    return 0x80000000;
}
```

Command 2: GETBITS

Parameter: number of bits to read, or 0 to read 32 bits [5 bits]

Result: the bits from the bitstream

Given parameter n, returns the next (n?n:32) bits from the bitstream as an unsigned integer.

Operation:

```
return getbits(n?n:32);
```

Command 3: NEXT_START_CODE

Parameter: none

Result: the next start code found

Skips bytes in the raw bitstream until the start code [00 00 01] is found. Then, read the byte after the start code and return it as the result. The bitstream pointer is advanced to point after the returned byte.

Operation:

```
byte_align();
while (nextbytes_raw(3) != 1)
    getbits_raw(8);
getbits_raw(24);
return getbits_raw(8);
```

Command 4: CABAC_START

Parameter: none

Result: none

Skips bits in the bitstream until the current bit position is byte-aligned, then initialises the arithmetic decoding engine registers codIRange and codIOffset, as per H.264.9.3.1.2.

Oprtation:

```
byte_align();
cabac_init_engine();
```

Command 5: MORE_RBSP_DATA

Parameter: none

Result: 1 if there's more data in RBSP, 0 otherwise

Returns 0 if there's a valid RBSP trailing bits element at the current bit position, 1 otherwise. Does not modify the bitstream pointer.

Operation:

```
return more_rbsp_data();
```

Command 6: MB_SKIP_FLAG

Parameter: none

Result: value of parsed mb_skip_flag

Parses the CABAC mb_skip_flag element. The SLICE_POS has to be set to the address of the macroblock to which this element applies.

Operation:

```
return cabac_mb_skip_flag();
```

Command 7: END_OF_SLICE_FLAG

Parameter: none

Result: value of parsed end_of_slice_flag

Parses the CABAC end_of_slice_flag element.

Operation:

```
return cabac_terminate();
```

Command 8: CABAC_INIT_CTX

Parameter: none

Result: none

Initialises the CABAC context variables, as per H.264.9.3.1.1. slice_type, cabac_init_idc [for P/B slices], and sliceqpy have to be set in the PARM registers for this command to work properly.

Operation:

```
cabac_init_ctx();
```

Command 9: MACROBLOCK_SKIP_MBFDF

Parameter: mb_field_decoding_flag presence [1 bit]

Result none

If parameter is 1, mb_field_decoding_flag syntax element is parsed. Otherwise, the value of mb_field_decoding_flag is inferred from preceding macroblocks. A skipped macroblock with thus determined value of mb_field_decoding_flag is emitted into the MBRING, and its data stored into internal state. SLICE_POS has to be set to the address of this macroblock.

Operation:

```
if (param) {
    if (entropy_coding_mode_flag)
        this_mb.mb_field_decoding_flag = cabac_mb_field_decoding_flag();
    else
        this_mb.mb_field_decoding_flag = getbits(1);
} else {
    this_mb.mb_field_decoding_flag = mb_field_decoding_flag_infer();
}
this_mb.mb_skip_flag = 1;
this_mb.slice_tag = slice_tag;
mbring_emit_macroblock();
```

Todo

more inferred crap

Command 0xa: MACROBLOCK_LAYER_MBFDF

Parameter: mb_field_decoding_flag presence [1 bit]

Result: none

If parameter is 1, mb_field_decoding_flag syntax element is parsed. Otherwise, the value of mb_field_decoding_flag is inferred from preceding macroblocks. A macroblock_layer syntax structure is parsed from the bitstream, data for the decoded macroblock is emitted into the MBRING, and stored into internal state. SLICE_POS has to be set to the address of this macroblock.

Operation:

```
if (param) {
    if (entropy_coding_mode_flag)
        this_mb.mb_field_decoding_flag = cabac_mb_field_decoding_flag();
    else
        this_mb.mb_field_decoding_flag = getbits(1);
} else {
    this_mb.mb_field_decoding_flag = mb_field_decoding_flag_infer();
}
this_mb.mb_skip_flag = 0;
this_mb.slice_tag = slice_tag;
macroblock_layer();
```

Command 0xb: PRED_WEIGHT_TABLE

Parameter: none

Result: none

Parses the pred_weight_table element, stores its contents in internal memory, and advances the bitstream to the end of the element.

Operation:

Todo

write me

Command 0xc: SLICE_DATA

Parameter: none

Result: none

Writes the stored pred_weight_table data to MBRING, parses the slice_data element, storing decoded data into MBRING, halting when the RBSP trailing bit sequence is encountered. When done, raises the MACROBLOCKS_DONE interrupt. Bitstream pointer is updated to point to the RBSP traling bits. SLICE_POS has to be set to the address of the first macroblock on slice before this command is called. When this command finishes, SLICE_POS is updated to the address of the last macroblock in the parsed slice.

Operation:

```
if (entropy_coding_mode_flag) {
    cabac_init_ctx();
    byte_align();
    cabac_init_engine();
}
mb_pos.first = 1;
first = 1;
skip_pending = 0;
end = 0;
bottom = 0;
while (1) {
    if (slice_type == P || slice_type == B) {
        if (entropy_coding_mode_flag) {
            while (1) {
                tmp = cabac_mb_skip_flag();
                if (!tmp)
                    break;
                skip_pending++;
                if (!mbaff_frame_flag || bottom) {
                    end = cabac_terminate();
                    if (end)
                        break;
                }
                bottom = !bottom;
            }
        } else {
            skip_pending = get_ue();
            end = !more_rbsp_data();
            bottom ^= skip_pending & 1;
        }
    } else {
        skip_pending = 0;
    }
    while (1) {
        if (!skip_pending)
            break;
        if (mbaff_frame_flag && bottom && skip_pending < 2)
            break;
        if (first) {
            first = 0;
        } else {
            mb_pos_advance();
        }
        macroblock_skip_mbfdf(0);
        skip_pending--;
    }
    if (end)
        break;
    if (first) {
        first = 0;
    } else {
        mb_pos_advance();
    }
    if (mbaff_frame_flag) {
        if (skip_pending) {
            macroblock_skip_mbfdf(1);
            mb_pos_advance();
```

```
            macroblock_layer_mbfdf(0);
            skip_pending = 0;
        } else {
            if (bottom) {
                macroblock_layer_mbfdf(0);
            } else {
                macroblock_layer_mbfdf(1);
            }
        }
        bottom = !bottom;
    } else {
        macroblock_layer_mbfdf(0);
    }
    if (entropy_coding_mode) {
        if (mbaff_frame_flag && bottom)) {
            end = 0;
        } else {
            end = cabac_terminate();
        }
    } else {
        end = !more_rbsp_data();
    }
    if (end) break;
}
trigger_intr(SLICE_DATA_DONE);
```

MBRING format

Contents

Introduction

An invocation of SLICE_DATA VLD command writes the decoded data into the MBRING. The MBRING is a ring buffer located in VM memory, made of 32-bit word oriented packets. Each packet starts with a header word, whose high 8 bits signify the packet type.

An invocation of SLICE_DATA command writes the following packets, in order:

- pred weight table [packet type 4] - if PRED_WEIGHT_TABLE command has been invoked previously
- for each macroblock [including skipped] in slice, in decoding order:
  - motion vectors [packet type 1] - if macroblock is not skipped and not intra coded
  - macroblock info [packet type 0] - always
  - residual data [packet type 2] - if at least one non-zero coefficient present

  - coded block mask [packet type 3] - if macroblock is not skipped

Packet type 0: macroblock info

Packet is made of a header word and 3 or 6 payload words.

- Header word:
  - bits 0-23: number of payload words [3 or 6]
  - bits 24-31: packet type [0]
- Payload word 0:
  - bits 0-12: macroblock address
- Payload word 1:
  - bits 0-7: y coord in macroblock units
  - bits 8-15: x coord in macroblock units
- Payload word 2:
  - bit 0: first macroblock of a slice flag
  - bit 1: mb_skip_flag
  - bit 2: mb_field_coding_flag
  - bits 3-8: mb_type
  - bits 9+i*4 - 12+i*4, i < 4: sub_mb_type[i]
  - bit 25: transform_size_8x8_flag
- Payload word 3:
  - bits 0-5: mb_qp_delta
  - bits 6-7: intra_chroma_pred_mode
- Payload word 4:
  - bits i*4+0 - i*4+2, i < 8: rem_intra_pred_mode[i]
  - bit i*4+3, i < 8: prev_intra_pred_mode_flag[i]
- Payload word 5:
  - bits i*4+0 - i*4+2, i < 8: rem_intra_pred_mode[i+8]
  - bit i*4+3, i < 8: prev_intra_pred_mode_flag[i+8]

Packet has 3 payload words when macroblock is skipped, 6 when it's not skipped. This packet type is present for all macroblocks. The mb_type and sub_mb_type values correspond to values used in CAVLC mode for current slice_type - thus for example I_NxN is mb_type 0 when decoding I slices, mb_type 5 when decoding P slices. For I_NxN macroblocks encoded in 4x4 transform mode, rem_intra_pred_mode[i] and pred_intra_pred_mode_flag[i] correspond to rem_intra4x4_pred_mode[i] and pred_intra4x4_pred_mode_flag[i] for i = 0..15. For I_NxN macroblocks encoded in 8x8 transform mode, rem_intra_pred_mode[i] and pred_intra_pred_mode_flag[i] correspond to rem_intra8x8_pred_mode[i] and pred_intra8x8_pred_mode_flag[i] for i = 0..3, and are unused for i = 4..15.

Packet type 1: motion vectors

Packet is made of two header words + 1 word for each motion vector.

- Header word:
  - bits 0-23: number of motion vectors [always 0x20]
  - bits 24-31: packet type [1]
- Second header word:
  - bit i = bit 4 of ref_idx[i]
- Motion vector word i:
  - bits 0-12: mvd[i] Y coord
  - bits 13-27: mvd[i] X coord
  - bits 28-31: bits 0-3 of ref_idx[i]

Indices 0..15 correspond to mvd_l0 and ref_idx_l0, indices 16-31 correspond to mvd_l1 and ref_idx_l1. Each index corresponds to one 4x4 block, in the usual scan order for 4x4 blocks. Data is always included for all blocks - if macroblock/sub-macroblock partition size greater than 4x4 is used, its data is duplicated for all covered blocks.

Packet type 2: residual data

Packet is made of a header word + 1 halfword for each residual coefficient + 0 or 1 halfwords of padding to the next multiple of word size

- Header word:
  - bits 0-23: number of residual coefficients
  - bits 24-31: packet type [2]
- Payload halfword:
  - bits 0-15: residual coefficient

For I_PCM macroblocks, this packet contains one coefficient for each pcm_sample_* element present in the bitstream, stored in bitstream order.

For other types of macroblocks, this packet contains data for all blocks that have at least one non-zero coefficient. If a block has a non-zero coefficient, all coefficients for this block, including zero ones, are stored in this packet. Otherwise, The block is entirely skipped. The coefficients stored in this packet type are dezigzagged - their order inside a single block corresponds to raster scan order. The blocks are stored in decoding order. The mask of blocks stored in this packet is stored in packet type 3. If there are no non-zero coefficients in the whole macroblock, this packet is not present.

Packet type 3: coded block mask

Packet is made of a header word and a payload word.

- Header word:
  - bits 0-23: number of payload words [1]
  - bits 24-31: packet type [3]
- Payload word [4x4 mode]:

  - bits 0-15: luma 4x4 blocks 0-15 [16 coords each]
  - bit 16: Cb DC block [4 coords]
  - bit 17: Cr DC block [4 coords]
  - bits 18-21: Cb AC blocks 0-3 [15 coords each]
  - bits 22-25: Cr AC blocks 0-3 [15 coords each]
- Payload word [8x8 mode]:
  - bits 0-3: luma 8x8 blocks 0-3 [64 coords each]
  - bit 4: Cb DC block [4 coords]
  - bit 5: Cr DC block [4 coords]
  - bits 6-9: Cb AC blocks 0-3 [15 coords each]
  - bits 10-13: Cr AC blocks 0-3 [15 coords each]
- Payload word [intra 16x16 mode]:
  - bit 0: luma DC block [16 coords]
  - bits 1-16: luma AC blocks 0-15 [15 coords each]
  - bit 17: Cb DC block [4 coords]
  - bit 18: Cr DC block [4 coords]
  - bits 19-22: Cb AC blocks 0-3 [15 coords each]
  - bits 23-26: Cr AC blocks 0-3 [15 coords each]
- Payload word [PCM mode]: [all 0]

This packet stores the mask of blocks present in preceding packet of type 2 [if any]. The bit corresponding to a block is 1 if the block has at least one non-zero coefficient and is stored in the residual data packet, 0 if all its coefficients are zero and it's not stored in the residual data packet. This packet type is present for all non-skipped macroblocks, including I_PCM macroblocks - but its payload word is always equal to 0 for I_PCM.

Packet type 4: pred weight table

Packet is made of a header word and a variable number of table write requests, each request being two words long.

- Header word:
  - bits 0-23: number of write requests
  - bits 24-31: packet type [4]
- Request word 0: table index to write
- Request word 1: data value to write

The pred weight table is treated as an array of 0x81 32-bit numbers. This packet is made of "write requests" which are supposed to modify the table entries in the receiver.

The table indices are:

- Index i * 2, 0 <= i <= 0x1f:
  - bits 0-7: luma_offset_l0[i]
  - bits 8-15: luma_weight_l0[i]

  - bit 16: chroma_weight_l0_flag[i]
  - bit 17: luma_weight_l0_flag[i]
- Index i * 2 + 1, 0 <= i <= 0x1f:
  - bits 0-7: chroma_offset_l0[i][1]
  - bits 8-15: chroma_weight_l0[i][1]
  - bits 16-23: chroma_offset_l0[i][0]
  - bits 24-31: chroma_weight_l0[i][0]
- Index 0x40 + i * 2, 0 <= i <= 0x1f:
  - bits 0-7: luma_offset_l1[i]
  - bits 8-15: luma_weight_l1[i]
  - bit 16: chroma_weight_l1_flag[i]
  - bit 17: luma_weight_l1_flag[i]
- Index 0x40 + i * 2 + 1, 0 <= i <= 0x1f:
  - bits 0-7: chroma_offset_l1[i][1]
  - bits 8-15: chroma_weight_l1[i][1]
  - bits 16-23: chroma_offset_l1[i][0]
  - bits 24-31: chroma_weight_l1[i][0]
- Index 0x80:
  - bits 0-2: chroma_log2_weight_denom
  - bits 3-5: luma_log2_weight_denom

The requests are emitted in the following order:

- 0x80
- for 0 <= i <= num_ref_idx_l0_active_minus1: 2*i, 2*i + 1
- for 0 <= i <= num_ref_idx_l1_active_minus1: 0x40 + 2*i, 0x40 + 2*i + 1

The fields corresponding to data not present in the bitstream are set to 0, they're *not* set to their inferred values.

VP2 command macro processor

Contents

Introduction

The VP2 macro processor is a small programmable processor that can emit vector processor commands when triggered by special commands from xtensa. All vector commands first go through the macro processor, which checks whether they're in macro command range, and either passes them down to vector processor, or interprets them itself, possibly launching a macro and submitting other vector commands. It is one of the four major blocks making up the PVP2 engine.

The macro processor has:

- 64-bit VLIW opcodes, controlling two separate execution paths, one primarily for processing/emitting commands, the other for command parameters
- dedicated code RAM, 512 64-bit words in size
- 32 * 32-bit word LUT data space, RW by host and RO by the macro code
- 6 32-bit global [not banked] GPRs visible to macro code and host [$g0-$g5]
- 8 32-bit banked GPRs visible to macro code and host, meant for passing parameters - one bank is writable by the param commands, the other is in use by macro code at any time [$p0-$p7]
- 3 1-bit predicates, with conditional execution [$p1-$p3]
- instruction set consisting of bit operations, shifts, and 16-bit addition
- no branch/loop capabilities
- a 32-bit command path accumulator [$cacc]
- a 32-bit data path accumulator [$dacc]
- a 7-bit LUT address register [$lutidx]
- 15-bit command, 32-bit data, and 8-bit high data registers for command submission [$cmd, $data, $datahi]
- 64-entry input command FIFO
- 2-entry output command FIFO

- a single hardware breakpoint

MMIO registers

The macro processor registers occupy 0x00f600:0x00f700 range in BAR0 space, corresponding to 0x2c000:0x2e000 range in PVP2's XLMI space. They are:

XLMI	MMIO	Name	Description
0x2c000	0x00f600	CONTROL	*master control*
0x2c100	0x00f608	STATUS	*detailed status*
0x2c180	0x00f60c	IDLE	*a busy/idle status*
0x2c200	0x00f610	INTR_EN	*interrupt enable*
0x2c280	0x00f614	INTR	*interrupt status*
0x2c300	0x00f618	BREAKPOINT	*breakpoint address and enable*
0x2c800:0x2c880	0x00f640	LUT[0:32]	*the LUT data*
0x2c880:0x2c8a0	0x00f644	PARAM_A[0:8]	*$p bank A*
0x2c900:0x2c920	0x00f648	PARAM_B[0:8]	*$p bank B*
0x2c980:0x2c9a0	0x00f64c	GLOBAL[0:8]	*$g registers*
0x2cb80	0x00f65c	PARAM_SEL	*$p bank selection switch*
0x2cc00	0x00f660	RUNNING	*code execution in progress switch*
0x2cc80	0x00f664	PC	*program counter*
0x2cd00	0x00f668	DATAHI	*$datahi register*
0x2cd80	0x00f66c	LUTIDX	*$lutidx register*
0x2ce00	0x00f670	CACC	*$cacc register*
0x2ce80	0x00f674	CMD	*$cmd register*
0x2cf00	0x00f678	DACC	*$dacc register*
0x2cf80	0x00f67c	DATA	*$data register*
0x2d000	0x00f680	IFIFO_DATA	*input FIFO data*
0x2d080	0x00f684	IFIFO_ADDR	*input FIFO command*
0x2d100	0x00f688	IFIFO_TRIGGER	*input FIFO manual read/write trigger*
0x2d180	0x00f66c	IFIFO_SIZE	*input FIFO size limitter*
0x2d200	0x00f670	IFIFO_STATUS	*input FIFO status*
0x2d280	0x00f674	OFIFO_DATA	*output FIFO data*
0x2d300	0x00f678	OFIFO_ADDR	*output FIFO command & high data*
0x2d380	0x00f67c	OFIFO_TRIGGER	*output FIFO manual read/write trigger*
0x2d400	0x00f680	OFIFO_SIZE	*output FIFO size limitter*
0x2d480	0x00f684	OFIFO_STATUS	*output FIFO status*
0x2d780	0x00f6bc	CODE_SEL	*selects high or low part of code RAM for code window*
0x2d800:0x2e000	0x00f6c0:0x00f700	CODE	*a 256-word window to code space*

Control and status registers

Todo

write me

Interrupts

Todo

write me

FIFOs

Todo

write me

Commands

The macro processor processes commands in 0xc000-0xdfff range from the input FIFO, passing down all other commands directly to the output FIFO [provided that no macro is executing at the moment]. The macro processor commands are:

Command	Name	Description
0xc000+i*4	MACRO_PARAM[0:8]	*write to $p host register bank*
0xc020+i*4	MACRO_GLOBAL[0:8]	*write to $g registers*
0xc080+i*4	MACRO_LUT[0:32]	*write to given LUT entry*
0xc100	MACRO_EXEC	*execute a macro*
0xc200	MACRO_DATAHI	*write to $datahi register*
0xd000+i*4	MACRO_CODE[0:0x400]	*upload half of a code word*

Execution state and registers

Code RAM The code RAM contains 512 opcodes. Opcodes are 64 bits long and are accessible by the host as pairs of 32-bit words. Code may be read or written using MMIO window:

BAR0 0x00f6bc / XLMI 0x2d780: CODE_SEL 1-bit RW register. Writing 0 selects code RAM entries 0:0x100 to be mapped to the CODE window, writing 1 selects code RAM entries 0x100:0x200.

BAR0 0x00f6c0 + (i >> 5) * 4 [index i & 0x1f] / XLMI 0x2d800 + i * 4, i < 0x200: CODE[i] The code window. Reading or writing CODE[i] is equivalent to reading or writing low [if i is even] or high [if i is odd] 32 bits of code RAM cell i >> 1 | CODE_SEL << 8.

They can also be written in pipelined manner by the MACRO_CODE command:

VP command 0xd000 + i * 4, i < 0x400: MACRO_CODE[i] Write the parameter to low [if i is even] or high [if i is odd] 32 bits of code RAM cell i >> 1. If a macro is currently executing, execution of this command is blocked until it finishes. Valid only on macro input FIFO.

Execution control

Todo

write me

Parameter registers Parameter registers server dual purpose: they're meant for passing parameters to macros, but can also be used as GPRs by the code. There are two banks of parameter registers, bank A and bank B. Each bank contains 8 32-bit registers. At any time, one of the banks is in use by the macro code, while the other can be written by the host via MACRO_PARAM commands for next macro execution. Each time a macro is launched, the bank assignments are swapped. The current assignment is controlled by the PARAM_SEL register:

BAR0 0x00f65c / XLMI 0x2cb80: PARAM_SEL 1-bit RW register. Can be set to one of:

- 0: CODE_A_CMD_B - bank A is in use by the macro code, commands will write to bank B
- 1: CODE_B_CMD_A - bank B is in use by the macro code, commands will write to bank A

This register is toggled on every MACRO_EXEC command execution.

The parameter register banks can be accessed through MMIO registers:

BAR0 0x00f644 [index i] / XLMI 0x2c880 + i * 4, i < 8: PARAM_A[i] BAR0 0x00f648 [index i] / XLMI 0x2c900 + i * 4, i < 8: PARAM_B[i]

These MMIO registers are mapped straight to corresponding parameter registers.

The bank not currently in use by code can also be written by MACRO_PARAM commands:

VP command 0xc000 + i * 4, i < 8: MACRO_PARAM[i] Write the command data to parameter register i of the bank currently not assigned to the macro code. Execution of this command won't wait for the current macro execution to finish. Valid only on macro input FIFO.

The parameter registers are visible to the macro code as GPR registers 0-7.

Global registers There are 6 normal global registers, $g0-$g5. They are simply 32-bit GPRs for use by macro computations. There are also two special global pseudo-registers, $g6 and $g7.

$g6 is the LUT readout register. Any attempt to read from it will read from the LUT entry selected by $lutidx register. Any attempt to write to it will be ignored.

$g7 is the special predicate register, $pred. Its 4 low bits are mapped to the four predicates, $p0-$p3. Any attempt to read from this register will read the predicates, and fill high 28 bits with zeros. Any attempt to write this register will write the predicates.

$p0 is always forced to 1, while $p1-$p3 are writable. The predicates are used for conditional execution in macro code. In addition to access through $pred, the predicates can also be written by macro code individually as a result of various operations.

All 8 global registers are accessible through MMIO and the command stream:

BAR0 0x00f64c [index i] / XLMI 0x2c980 + i * 4, i < 8: GLOBAL[i] These registers are mapped straight to corresponding global registers.

VP command 0xc020 + i * 4, i < 8: MACRO_GLOBAL[i] Write the command data to global register i. If a macro is currently executing, execution of this command is blocked until it finishes. Valid only on macro input FIFO.

The global registers are visible to the macro code as GPR registers 8-15.

Special registers In addition to the GPRs, the macro code can use 6 special registers. There are 4 special registers belonging to the command execution path, identified by a 2-bit index:

- 0: $cacc, command accumulator
- 1: $cmd, output command register
- 2: $lutidx, LUT index
- 3: $datahi, output high data register

There are also 2 special registers belonging to the data execution path, identified by a 1-bit index:

- 0: $dacc, data accumulator
- 1: $data, output data register

The $cacc and $dacc registers are 32-bit and can be read back by the macro code, and so are usable for general purpose computations.

The $cmd, $data, and $datahi registers are write-only by the macro code, and their contents are submitted to the macro output FIFO when a submit opcode is executed. $data is 32-bit, $datahi is 8-bit, mapping to bits 0-7 of written values. $cmd is 15-bit, mapping to bits 2-16 of written values. The $datahi register is also used to fill the high data bits in output FIFO whenever a command is bypassed from the input FIFO.

The $lutidx register is 5-bit and write-only by the macro code. It maps to bits 0-4 of written values. Its value selects the LUT entry visible in $g6 pseudo-register.

All 6 special registers can be accessed through MMIO, and the $datahi register can be additionally set by a command:

MMIO 0x00f668 / XLMI 0x2cd00: DATAHI MMIO 0x00f66c / XLMI 0x2cd80: LUTIDX MMIO 0x00f670 / XLMI 0x2ce00: CACC MMIO 0x00f674 / XLMI 0x2ce80: CMD MMIO 0x00f678 / XLMI 0x2cf00: DACC MMIO 0x00f67c / XLMI 0x2cf80: DATA

> These registers map directly to corresponding special registers. For $cacc, $dacc, and $data, all bits are valid. For $cmd, bits 2-16 are valid. For $lutidx, bits 0-4 are valid. For $datahi, bits 0-7 are valid. Remaining bits are forced to 0.

VP command 0xc200: MACRO_DATAHI Sets $datahi to low 8 bits of the command data. If a macro is currently executing, execution of this command is blocked until it finishes. Valid only on macro input FIFO.

The LUT The LUT is a small indexable RAM that's read-only by the macro code, but freely writable by the host. It's made of 32 32-bit words. The LUT entry selected by $lutidx register can be read by macro code simply by reading from the $g6 pseudo-register. The LUT can be accessed by the host through MMIO and the command stream:

BAR0 0x00f640 [index i] / XLMI 0x2c800 + i * 4, i < 32: LUT[i] These registers are mapped straight to corresponding LUT entries.

VP command 0xc080 + i * 4, i < 32: MACRO_LUT[i] Write the command data to LUT entry i. If a macro is currently executing, execution of this command is blocked until it finishes. Valid only on macro input FIFO.

Opcodes

The code opcodes are 64 bits long. They're divided in several major parts:

- bits 0-2: conditional execution predicate selection.
  - bits 0-1: PRED, the predicate to use [selected from $p0-$p3]
  - bit 2: PNOT, selects whether the predicate is negated before use.
- bit 3: EXIT, exit flag
- bit 4: SUBMIT, submit flag
- bits 5-30: command opcode
- bits 31-32: PDST, predicate destination [selected from $p0-$p3]
- bits 33-63: data opcode

When a macro is launched, opcodes are executed sequentially from the macro start address until an opcode with the exit flag set is executed. An opcode is executed as follows:

1. If the SUBMIT bit is set, the current values of $cmd, $data, $datahi are sent to the output FIFO.
2. Conditional execution status is determined: the predicate selected by PRED is read. If PNOT is set to 0, conditional execution will be enabled if the predicate is set to 1. Otherwise [PNOT set to 1], conditional execution will be enabled if the predicate is set to 0. Unconditional opcodes are simply opcodes using non-negated predicate $p0 [PRED = 0, PNOT = 0].
3. If the SUBMIT bit is set, conditional execution is enabled, and ($cmd & 0x1fe80) == 0xb000 [ie. the submitted command was in 0xb000-0xb07c or 0xb100-0xb17c ranges, correnspoding to vector processor param commands], $cmd is incremented by 4. This enables submitting several parameters in a row without having to update the $cmd register.
4. If conditional execution is enabled, the command opcode is executed, and the command result, command predicate result, and the C2D intermediate value are computed.
5. If conditional execution is enabled, the data opcode is executed, and the data result and data predicate result are computed.
6. If conditional execution is enabled, the command and data results are written to their destination registers.
7. If the EXIT bit is set, macro execution halts.

Effectively, conditional execution affects all computations [including auto $cmd increment], but doesn't affect submit and exit opcodes.

Command opcodes The command processing path is mainly meant for processing commands and data going to $lutidx/$datahi register, but can also exchange data with the data processing path if needed.

The command opcode bitfields are:

- bits 5-9: CBFSTART - bitfield start [CINSRT_R, CINSRT_I, some data ops]
- bits 10-14: CBFEND - bitfield end [CINSRT_R, CINSRT_I, some data ops]
- bits 15-19: CSHIFT - shift count [CINSRT_R]
- bit 20: CSHDIR - shift direction [CINSRT_R]
- bits 15-20: CIMM6 - 6-bit unsigned immediate [CINSRT_I]
- bits 21-22: CSRC2 - selects command source #2 [CINSRT_I, CINSRT_R], one of:
  - 0: ZERO, source #2 is 0
  - 1: CACC, source #2 is current value of $cacc
  - 2: DACC, source #2 is current value of $dacc
  - 3: GPR, source #2 is same as command source #1
- bits 15-22: CIMM8 - 8-bit unsigned immediate [CEXTRADD8]
- bits 5-22: CIMM18 - 18-bit signed immediate [CMOV_I]
- bits 23-26: CSRC1 - selects command source #1 [CINSRT_R, CEXTRADD8, DSHIFT_R, DADD16_R]. The command source #1 is the GPR with index selected by this bitfield.
- bits 27-28: CDST - the command destination, determines where the command result will be written; one of:
  - 0: CACC
  - 1: CMD
  - 2: LUTIDX
  - 3: DATAHI

- bits 29-30: COP - the command operation, one of:
  - 0: CINSRT_R, bitfield insertion with shift, register sources
  - 1: CINSRT_I, bitfield insertion with 6-bit immediate source
  - 2: CMOV_I, 18-bit immediate value load
  - 3: CEXTRADD8, bitfield extraction + 8-bit immediate addition

The command processing path computes four values for further processing:

- the command result, ie. the 32-bit value that will later be written to the command destination register
- the command predicate result, ie. the 1-bit value that may later be written to the destination predicate
- the C2D value, a 32-bit intermediate result used in some data opcodes
- the command bitfield mask [CBFMASK], a 32-bit value used in some command and data opcodes

The command bitfield mask is used by the bitfield insertion operations. It is computed from the command bitfield start and end as follows:

```
if (CBFEND >= CBFSTART) {
    CBFMASK = (2 << CBFEND) - (1 << CBFSTART); // bits CBFSTART-CBFEND are 1
} else {
    CBFMASK = 0;
}
```

Since the CBFEND and CBFSTART fields conflict with CIMM18 field, the data ops using the command mask should not be used together with the CMOV_I operation.

The CINSRT_R operation has the following semantics:

```
if (CSHDIR == 0) /* 0 is left shift, 1 is right logical shift */
    shifted_source = command_source_1 << CSHIFT;
else
    shifted_source = command_source_1 >> CSHIFT;
C2D = command_result = (shifted_source & CBFMASK) | (command_source_2 & ~CBFMASK);
command_predicate_result = (shifted_source & CBFMASK) == 0;
```

The CINSRT_I operation has the following semantics:

```
C2D = command_result = (CIMM6 << CBFSTART & CBFMASK) | (command_source_2 & ~CBFMASK);
command_predicate_result = 0;
```

The CMOV_I operation has the following semantics:

```
C2D = command_result = sext(CIMM18, 17); /* sign-extend 18-bit immediate to 32 bits */
command_predicate_result = 0;
```

The CEXTRADD8 operation has the following semantics:

```
C2D = (command_source_1 & CBFMASK) >> CBFSTART;
command_result = ((C2D + CIMM8) & 0xff) | (C2D & ~0xff); /* add immediate to low 8 bits of extracted
command_predicate_result = 0;
```

Data opcodes The command processing path is mainly meant for processing command data, but can also exchange data with the command processing path if needed.

The data opcode bitfields are:

- bits 33-37: DBFSTART - bitfield start [DINSRT_R, DINSRT_I, DSEXT]

- bits 38-42: DBFEND - bitfield end [DINSRT_R, DINSRT_I, DSEXT]
- bits 43-47: DSHIFT - shift count and SEXT bit position [DINSRT_R, DSEXT]
- bit 48: DSHDIR - shift direction [DINSRT_R, DSHIFT_R]
- bits 43-48: DIMM6 - 6-bit unsigned immediate [DINSRT_I]
- bits 33-48: DIMM16 - 16-bit immediate [DADD16_I, DLOGOP16_I]
- bit 49: C2DEN - enables double bitfield insertion, using C2D value [DINSRT_R, DINSRT_I, DSEXT]
- bit 49: DDSTSKIP - skips DDST write if set [DADD16_I]
- bit 49: DSUB - selects whether DADD16_R operation does an addition or substraction
- bits 49-50: DLOGOP - the DLOGOP16_I suboperation, one of:
  - 0: MOV, result is set to immediate
  - 1: AND, result is source ANDed with the immediate
  - 2: OR, result is source ORed with the immediate
  - 3: XOR, result is source XORed with the immediate
- bits 50-51: DSRC2 - selects data source #2 [DINSRT_R, DINSRT_I], one of:
  - 0: ZERO, source #2 is 0
  - 1: CACC, source #2 is current value of $cacc
  - 2: DACC, source #2 is current value of $dacc
  - 3: GPR, source #2 is same as data source #1
- bit 50: DHI2 - selects low or high 16 bits of second operand [DADD16_R]
- bit 51: DHI - selects low or high 16 bits of an operand [DADD16_I, DLOGOP16_I, DADD16_R]
- bits 52-55: DSRC1 - selects data source #1 [DINSRT_R, DINSRT_I, DADD16_I, DLOGOP16_I, DSHIFT_R, DSEXT, DADD16_R]. The data source #1 is the GPR with index selected by this bitfield.
- bits 33-55: DIMM23 - 23-bit signed immediate [DMOV_I]
- bits 56-59: DRDST - selects data GPR destination register. The GPR destination is the GPR with index selected by this bitfield. The data result will be written here, along with the special register selected by DDST.
- bit 60: DDST - the data special register destination, determines where the data result will be written (along with DRDST); one of:
  - 0: DACC
  - 1: DATA
- bits 61-63: DOP - the data operation, one of:
  - 0: DINSRT_R, bitfield insertion with shift, register sources
  - 1: DINSRT_I, bitfield insertion with 6-bit immediate source
  - 2: DMOV_I, 23-bit immediate value load
  - 3: DADD16_I, 16-bit addition with immediate
  - 4: DLOGOP16_I, 16-bit logic operation with immediate
  - 5: DSHIFT_R, shift by the value of a register
  - 6: DSEXT, sign extension

- 7: DADD16_R, 16-bit addition/substraction with register operands

The data processing path computes three values:

- the data result, ie. the 32-bit value that will be written to the data destination registers
- the data predicate result, ie. the 1-bit value that will be written to the destination predicate
- the skip special destination flag, a 1-bit flag that disables write to the data special register if set

Not all data operations produce a predicate result. For ones that don't, the command predicate result will be output instead.

The DINSRT_R operation has the following semantics:

```
if (DBFEND >= DBFSTART) {
    DBFMASK = (2 << DBFEND) - (1 << DBFSTART); // bits DBFSTART-DBFEND are 1
} else {
    DBFMASK = 0;
}
if (DSHDIR == 0) /* 0 is left shift, 1 is right arithmetic shift */
    shifted_source = data_source_1 << DSHIFT;
else
    shifted_source = (-1 << 32 | data_source_1) >> DSHIFT;
data_result = (data_source_2 & ~DBFMASK) | (shifted_source & DBFMASK);
if (C2DEN)
    data_result = (data_result & ~CBFMASK) | (C2D & CBFMASK);
data_predicate_result = (shifted_source & DBFMASK) == 0;
skip_special_destination = false;
```

The DINSRT_I operation has the following semantics:

```
if (DBFEND >= DBFSTART) {
    DBFMASK = (2 << DBFEND) - (1 << DBFSTART); // bits DBFSTART-DBFEND are 1
} else {
    DBFMASK = 0;
}
data_result = (data_source_2 & ~DBFMASK) | (DIMM6 << DBFSTART & DBFMASK);
if (C2DEN)
    data_result = (data_result & ~CBFMASK) | (C2D & CBFMASK);
data_predicate_result = command_predicate_result;
skip_special_destination = false;
```

The DMOV_I operation has the following semantics:

```
data_result = sext(DIMM23, 22); /* sign-extend 23-bit immediate to 32 bits */
data_predicate_result = command_predicate_result;
skip_special_destination = false;
```

The DADD16_I operation has the following semantics:

```
sum = ((data_source_1 >> (16 * DHI)) + DIMM16) & 0xffff;
data_result = (data_source_1 & ~(0xffff << (16 * DHI))) | sum << (16 * DHI);
data_predicate_result = sum >> 15 & 1;
skip_special_destionation = DDSTSKIP;
```

The DLOGOP16_I operation has the following semantics:

```
src = (data_source_1 >> (16 * DHI)) & 0xffff;
switch (DLOGOP) {
    case MOV: res = DIMM16; break;
    case AND: res = src & DIMM16; break;
```

```
    case OR: res = src | DIMM16; break;
    case XOR: res = src ^ DIMM16; break;
}
data_result = (data_source_1 & ~(0xffff << (16 * DHI))) | res << (16 * DHI);
data_predicate_result = (res == 0);
skip_special_destination = false;
```

The DSHIFT_R operation has the following semantics:

```
shift = command_source_1 & 0x1f;
if (DSHDIR == 0) /* 0 is left shift, 1 is right arithmetic shift */
    data_result = data_source_1 << shift;
else
    data_result = (-1 << 32 | data_source_1) >> shift;
data_predicate_result = command_predicate_result;
skip_special_destination = false;
```

The DSEXT operation has the following semantics:

```
bfstart = max(DBFSTART, DSHIFT);
if (DBFEND >= bfstart) {
    DBFMASK = (2 << DBFEND) - (1 << bfstart); // bits bfstart-DBFEND are 1
} else {
    DBFMASK = 0;
}
sign = data_source_2 >> DSHIFT & 1;
data_result = (data_source_2 & ~DBFMASK) | (sign ? DBFMASK : 0);
if (C2DEN)
    data_result = (data_result & ~CBFMASK) | (C2D & CBFMASK);
data_predicate_result = sign;
skip_special_destination = false;
```

The DADD16_R operation has the following semantics:

```
src1 = (data_source_1 >> (16 * DHI)) & 0xffff;
src2 = (command_source_1 >> (16 * DHI2)) & 0xffff;
if (DSUB == 0)
    sum = (src1 + src2) & 0xffff;
else
    sum = (src1 - src2) & 0xffff;
data_result = (data_source_1 & ~(0xffff << (16 * DHI))) | sum << (16 * DHI);
data_predicate_result = sum >> 15 & 1;
skip_special_destionation = false;
```

Destination write Once both command and data processing is done, the results are written to the destination registers, as follows:

- command_result is written to command special register selected by CDST.
- data_result is written to data special register selected by DDST, unless skip_special_destionation is true.
- data_result is written to GPR selected by DRDST. This can be effectively disabled by setting DRDST to $g6.
- data_predicate_result is written to predicate selected by PDST. This can be effectively disabled by setting PDST to $p0.

Introduction

Todo

write me

2.11.4 VP3/VP4/VP5 video decoding

Contents:

VP3 MBRING format

Contents

Introduction

The macroblock ring outputted from VLD is packet based, and aligned on 32-bit word size.

A packet has the header type in bits [24..31] and length in bits [0..23]. The data length is in words, and doesn't include the header itself.

type 00: Macro block header

MPEG2 The macro block header contains 4 data words:

- Word 0:
  - [0:15] Absolute address in macroblock units, 0 based
- Word 1:
  - [0:7] Y coord in macroblock units, 0 based
  - [8:15] X coord in macroblock units, 0 based

- Word 2:
  - [0] not_coded[??]
  - [1] skipped[??]
  - [3] quant
  - [4] motion_forward
  - [5] motion_backward
  - [6] coded_block_pattern
  - [7] intra
  - [26:26] dct_type
  - [27:28] motion_type
    - 0: field motion
    - 1: frame-based motion
    - 2: 16x8 field
    - 3: dual prime motion
- Word 3:
  - [6:7] motion_vector_count
  - [8:12] quantiser_scale_code

H.264

- Payload word 0:
  - bits 0-12: macroblock address
- Payload word 1:
  - bits 0-7: y coord in macroblock units
  - bits 8-15: x coord in macroblock units
- Payload word 2:
  - bit 0: first macroblock of a slice flag
  - bit 1: mb_skip_flag
  - bit 2: mb_field_coding_flag
  - bits 3-8: mb_type
  - bits 9+i*4 - 12+i*4, i < 4: sub_mb_type[i]
  - bit 25: transform_size_8x8_flag
- Payload word 3:
  - bits 0-5: mb_qp_delta
  - bits 6-7: intra_chroma_pred_mode
- Payload word 4:
  - bits i*4+0 - i*4+2, i < 8: rem_intra_pred_mode[i]

  - bit i*4+3, i < 8: prev_intra_pred_mode_flag[i]
- Payload word 5:
  - bits i*4+0 - i*4+2, i < 8: rem_intra_pred_mode[i+8]
  - bit i*4+3, i < 8: prev_intra_pred_mode_flag[i+8]

Error The macro block header contains 3 data words:

- Word 0:
  - [0:15] Absolute address in macroblock units, 0 based
  - [16] error flag, always set
- Word 1:
  - [0:7] Y coord in macroblock units, 0 based
  - [8:15] X coord in macroblock units, 0 based
- Word 2: all 0

type 01: Motion vector

MPEG2

Todo

Verify whether X or Y is in the lowest 16 bits. I assume X

The motion vector has a length of 4 data words, and contains a total of 8 PMVs with a size of 16 bits each. The motion vectors are likely encoded in order of the spec with PMV[r][s][t].

The layout of each 16 bit PMV:

- [0:5] motion code
- [6:13] residual
- [14] motion_vertical_field_select
- [14:15] dmvector (0, 1, or 3)

motion_vertical_field_select and dmvector occupy same bits, but the mpeg spec makes them mutually exclusive, so they don't conflict.

H.264 Payload like VP2, except length is in 32-bit words.

type 02: DCT coordinates

A packet of this type is created for each pattern enabled in coded_block_pattern. This packet type is byte oriented, rather than word oriented. It splits the coordinates up in chunks of 4 coordinates each, so 0..3 becomes 0, 4..7 becomes 1, 60..63 becomes chunk 15. The first 2 bytes contain a 16-bit bitmask indicating the presence of each chunk. If a chunks bit is set it will be encoded further.

For each present chunk a 8-bit bitmask will be created, which contains the size of each coordinate in that chunk. 2 bits are used for each coordinate, indicating the size (0 = not present, 1 = 1 byte, 2 = 2 bytes). This is followed by all coordinates present in this chunk, the last chunk is padded with 0s to align to word size.

For example: 0x10 0x00 0x40 0xff

Chunk 4 (0x0010>>4)&1 has pos 3 (0x40 >> (2*3))&3 set to -1

type 03: PCM data

Payload length is 0x60 words. Packet is byte oriented, instead of word oriented. Payload is raw PCM data from bitstream.

type 04: Coded block pattern

MPEG2 This packet puts coded_block_pattern in 1 data word.

H.264 Payload like VP2.

type 05: Pred weight table

Payload like VP2, except length is in 32-bit words.

type 06: End of stream

This header has no length, and signals the parser it's done.

Macroblock

A macroblock is created in this order:

- motion vector (optional)
- macro block header
- DCT coordinates / PCM samples (optional, and repeated as many times as needed)
- coded_block_pattern (optional)

'optional' is relative to the MPEG spec. For example intra frames always require a coded_block_pattern.

Introduction

Todo

write me

2.12 Performance counters

Contents:

2.12.1 NV10:NV40 signals

Contents

Todo

convert

```
=== NV10 signals ===

0x70: PGRAPH.PM_TRIGGER
0x87: PTIMER_TIME_B12 [bus/ptimer.txt]
0x80: trailer base

=== NV15 signals ===

0x70: PGRAPH.PM_TRIGGER
0x87: PTIMER_TIME_B12 [bus/ptimer.txt]
0x80: trailer base

=== NV1F signals ===

0x70: PGRAPH.PM_TRIGGER
0x86: HEAD0_VBLANK
0x87: HEAD1_VBLANK
0x80: trailer base


=== NV20 signals ===

domain 0 [nvclk]:
0xaa: HEAD0_VBLANK
0xa0: trailer base

domain 1 [mclk]:
0x20: trailer base


=== NV28 signals ===

domain 0 [nvclk]:
0xaa: HEAD0_VBLANK
0xa0: trailer base

domain 1 [mclk]:
0x20: trailer base


=== NV35 signals ===

domain 0 [nvclk]:
0xf8: HEAD0_VBLANK
0xf9: HEAD1_VBLANK
```

```
0xe0: trailer base

domain 1 [mclk]:
0x20: trailer base

=== NV31 signals ===

domain 0 [nvclk]:
0xf8: HEAD0_VBLANK
0xf9: HEAD1_VBLANK
0xe0: trailer base

domain 1 [mclk]:
0x20: trailer base

=== NV34 signals ===

domain 0 [nvclk]:
0xda: HEAD0_VBLANK
0xdb: HEAD1_VBLANK
0xe0: trailer base

domain 1 [mclk]:
0x20: trailer base
```

2.12.2 NV40:G80 signals

Contents

Introduction

NV40 generation cards have the following counter domains:

- NV40 generation cards without turbocache:
  - 0: host clock
  - 1: core clock [PGRAPH front]
  - 2: geometry[?] clock [PGRAPH back]
  - 3: shader clock
  - 4: memory clock
- NV40 generation with turbocache that are not IGPs:
  - 0: host clock
  - 1: core clock [PGRAPH front]
  - 2: shader clock
  - 3: memory clock

- NV40 IGP:
  - 0: host clock
  - 1: core clock [PGRAPH probably]
  - 2: core clock [shaders probably]
  - 3: unknown, could be the memory interface

Todo

figure it out

Todo

find some, I don't know, signals?

2.12.3 G80:GF100 signals

Contents

Introduction

G80 generation cards have the following counter domains:

- G80:
  - 0: host clock
  - 1: core clock A
  - 2: core clock B
  - 3: shader clock
  - 4: memory clock
- G84:GF100 except MCP7x:
  - 0: host clock
  - 1: core clock A

  - 2: core clock B
  - 3: shader clock
  - 4: memory clock
  - 5: core clock C
  - 6: vdec clock
  - 7: core clock D
- MCP7x:
  - 0: host clock
  - 1: core clock A
  - 2: core clock B
  - 3: shader clock
  - 4: core clock C
  - 5: vdec clock
  - 6: core clock D

Todo

figure out roughly what stuff goes where

Todo

find signals.

Host clock

signal	G80	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79	GT215	GT216	GT218	MCP89	documentation
HOST_MEM_WR	04	04	04	04	04	04	04	05	??	??	1a	1a	1a	??	[XXX]
PCOUNTER.USER	–	–	–	–	–	–	–	–	–	–	2a-2b	2a-2b	2a-2b	3a-3b	pcounter/intro.txt
???	??	??	??	??	??	??	??	0a	??	??	??	??	??	??	all PFIFO engines enabled and idle???
???	??	??	??	??	??	??	28	??	??	??	??	??	??	??	happens once with PFIFO write or PDISPLAY access [not PFIFO read]
???	??	??	??	??	??	??	??	29	??	??	??	??	??	??	??? on for 10%
???	??	??	??	??	??	??	??	2a	??	??	??	??	??	??	??? on for 10%
???	??	??	??	??	??	??	??	2b	??	??	??	??	??	??	pcie activity wakeups [long]?!?
???	??	??	??	??	??	??	??	2c	??	??	??	??	??	??	pcie activity bursts?!?
???	??	??	??	??	??	??	??	??	??	??	??	74	??	??	MMIO reads?
HOST_MEM_RD	1d	1f	1f	27	2a	2a	2a	2e	??	??	96	96	96	??	[XXX]
???	1c	21	??	29	??	2c	??	30	??	??	??	??	98	??	triple MMIO read?
PBUS_PCIE_RD	1e	22	22	2a	2d	2d	2d	31	??	??	99	99	99	??	[XXX]
PTIMER_TIME_B	27	2c	2c	34	37	37	37	3b	53	53	a3	a3	a3	4a	bus/ptimer.txt
PBUS_PCIE_TRANS	29	2e	2e	36	39	39	39	3d	??	??	a5	a5	a5	??	[XXX]
PBUS_PCIE_WR	2a	2f	2f	37	3a	3a	3a	3e	??	??	a6	a6	a6	??	[XXX]
PCOUNTER.TRAILER	2c-3f	4c-5f	4c-5f	4c-5f	4c-5f	4c-5f	4c-5f	6c-7f	8c-9f	8c-9f	ec-ff	ec-ff	ec-ff	8c-9f	pcounter/intro.txt

Core clock A

signal	G80	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79
TPC.GEOM.MUX	10-16	00-06	00-06	00-06	00-06	00-06	00-06	??	00-06	00-06
ZCULL.???	20-25	07-0c	07-0c	07-0c	07-0c	07-0c	07-0c	07-0c	??	??
TPC.RAST.???	??	19	19	19	19	19	??	??	??	??
TPC.RAST.???	??	1a	1a	1a	1a	1a	??	??	??	??
PREGEOM.???	??	??	??	??	??	??	??	??	??	??
PREGEOM.???	??	??	??	??	??	??	??	??	??	??
POSTGEOM.???	??	??	??	??	??	??	??	??	??	??
POSTGEOM.???	??	??	??	??	??	??	??	??	??	??
RATTR.???	??	??	??	??	??	??	??	??	??	??
APLANE.CG	–	31-33	31-33	31-33	31-33	31-33	31-33	??	39-3b	39-3b
RATTR.CG	–	37-39	37-39	37-39	37-39	37-39	37-39	??	43-45	43-45
ZCULL.???	??	??	4f	4f	4f	4f	4f	??	??	??
VFETCH.MUX	26-3f	66-7f	66-7f	66-7f	66-7f	66-7f	66-7f	46-5f	46-5f	46-5f
TPC.RAST.CG	–	??	??	??	??	??	??	??	??	??
PCOUNTER.USER	–	–	–	–	–	–	–	–	–	–
ZCULL.???	6e	??	??	??	??	??	??	??	??	??
ZCULL.???	??	??	??	??	??	??	??	??	??	75
ZCULL.???	??	??	??	??	??	??	??	??	??	??
APLANE.CG_IFACE_DISABLE	73	–	–	–	–	–	–	–	–	–

Table 2.12 – continued from previous page

signal	G80	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79
VATTR.???	77-7b	??	??	??	??	??	??	??	??	??
VATTR.???	??	57	??	57	57	57	57	??	7d	??
VATTR.???	??	59	??	59	59	59	59	??	7f	??
VATTR.???	7c	5c	5c	5c	5c	5c	5c	82	??	??
VATTR.???	7d	5d	5d	5d	5d	5d	5d	83	??	??
VATTR.CG_IFACE_DISABLE	7e	–	–	–	–	–	–	–	–	–
STRMOUT.???	7f	5e	5e	5e	5e	5e	5e	84	??	??
STRMOUT.???	80	5f	5f	5f	5f	5f	5f	85	??	??
STRMOUT.???	81	??	??	??	??	??	??	??	??	??
STRMOUT.???	??	??	??	??	??	??	??	??	85	??
CLIPID.???	??	??	??	??	??	??	??	??	??	8a
CLIPID.???	??	??	??	??	??	??	??	??	??	8c
RMASK.???	??	??	??	??	??	??	??	??	8e	??
STRMOUT.CG_IFACE_DISABLE	82	–	–	–	–	–	–	–	–	–
TPC.GEOM.???	8d	85	85	85	85	85	85	??	??	91
TPC.GEOM.???	8f	87	87	87	87	87	87	??	??	93
TPC.GEOM.???	91	89	89	89	89	89	89	??	??	95
TPC.GEOM.???	93	8b	8b	8b	8b	8b	8b	??	??	97
TPC.GEOM.???	??	??	??	??	??	??	??	??	91	??
TPC.GEOM.???	??	??	??	??	??	??	??	??	93	??
TPC.GEOM.???	??	??	??	??	??	??	??	??	95	??
RATTR.CG_IFACE_DISABLE	95	–	–	–	–	–	–	–	–	–
RATTR.???	96	??	??	??	??	??	??	??	??	??
RATTR.???	97	??	??	??	??	??	??	??	??	??
RATTR.???	98	??	??	??	??	??	??	??	??	??
RATTR.???	99	??	??	??	??	??	??	??	??	??
RATTR.???	??	8d	8d	8d	8d	8d	8d	??	97	??
TPC.RAST.???	9b	92	92	92	92	92	92	??	9c	9e
TPC.RAST.???	9d	94	94	94	94	94	94	??	9e	a0
ENG2D.???	??	??	9b	9b	9b	9b	9b	??	??	a7
ENG2D.???	??	??	9d	9d	9d	9d	9d	??	??	a9
ENG2D.CG_IFACE_DISABLE	a7	–	–	–	–	–	–	–	–	–
???	ae	a4	a4	a4	a4	a4	a4	b0	??	??
VCLIP.???	b8	ae	??	ae	ae	ae	ae	??	b8	ba
VCLIP.???	ba	b0	??	b0	b0	b0	b0	??	ba	bc
VCLIP.CG_IFACE_DISABLE	bb	–	–	–	–	–	–	–	–	–
DISPATCH.???	??	??	??	??	??	??	??	??	??	??
PGRAPH.IDLE	c8	bd	bd	bd	bd	bd	bd	c9	??	c9
PGRAPH.INTR	ca	bf	bf	bf	bf	bf	bf	cb	??	cb
CTXCTL.USER	d2-d5	c7-ca	c7-ca	c7-ca	c7-ca	c7-ca	c7-ca	d3-d6	d1-d4	d3-d6
TRAST.???	dc	d2	d2	d2	d2	d2	d2	de	??	??
TRAST.???	dd	d3	d3	d3	d3	d3	d3	df	??	??
TRAST.???	de	d4	d4	d4	d4	d4	d4	e0	??	??
TRAST.???	df	d5	d5	d5	d5	d5	d5	e1	??	??
TRAST.???	e2	d8	d8	d8	d8	d8	d8	e4	??	??
TRAST.???	e3	d9	d9	d9	d9	d9	d9	e5	e3	e5
TRAST.???	e5	db	db	db	db	db	db	??	e5	e7
TRAST.CG_IFACE_DISABLE	e6	–	–	–	–	–	–	–	–	–
PCOUNTER.TRAILER	ee-ff	ec-ff	ec-ff	ec-ff	ec-ff	ec-ff	ec-ff	ec-ff	ec-ff	ec-ff

Core clock B

signal	G80	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79
PROP.MUX	00-07	00-07	00-07	00-07	00-07	00-07	00-07	00-07	00-07	00-07
PVPE.???	3a	??	??	??	??	??	–	??	–	–
CCACHE.???	??	??	??	??	??	??	??	??	??	??
CCACHE.???	??	??	??	??	??	??	??	??	??	??
TEX.???	40	1a	1a	1a	1a	1a	1a	32	??	??
TEX.???	41	1b	1b	1b	1b	1b	1b	33	??	??
TEX.???	42	1c	1c	1c	1c	1c	1c	34	??	??
VATTR.???	??	??	??	??	??	??	??	??	??	3c
VATTR.???	??	??	??	??	??	??	??	??	??	3e
STRMOUT.???	??	??	??	??	??	??	??	??	??	46
STRMOUT.???	??	??	??	??	??	??	??	??	??	48
CBAR.MUX0	4a-4d	24-27	24-27	24-27	24-27	24-27	24-27	??	49-4c	49-4c
CBAR.MUX1	4e-51	28-2b	28-2b	28-2b	28-2b	28-2b	28-2b	??	4d-50	4d-50
CROP.MUX	52-55	30-33	30-33	30-33	30-33	30-33	30-33	55-58	55-58	55-58
ENG2D.???	??	??	??	36-37	36-37	36-37	??	??	??	??
ZBAR.MUX	56-59	38-3b	38-3b	38-3b	38-3b	38-3b	38-3b	??	68-6b	68-6b
???	6d	??	??	??	??	??	??	??	??	??
???	5e	??	??	??	??	??	??	??	??	??
???	64	??	??	??	??	??	??	??	??	??
???	68	??	??	??	??	??	??	??	??	??
VCLIP.???	??	??	??	??	??	??	??	??	64	??
VCLIP.???	??	??	??	??	??	??	??	??	65	??
ZROP.MUX	6c-6f	44-47	44-47	44-47	44-47	44-47	44-47	74-77	74-77	74-77
TEX.???	70-73	48-4b	48-4b	48-4b	48-4b	48-4b	48-4b	78-7b	78-7b	78-7b
PCOUNTER.USER	–	–	–	–	–	–	–	–	–	–
???	80	??	??	??	??	??	??	??	??	??
PVPE.???	89-a6	??	??	??	??	??	–	??	–	–
PROP.???	ab	??	??	??	??	??	??	??	??	??
MMU.CG_IFACE_DISABLE	ac	–	–	–	–	–	–	–	–	–
MMU.BIND	ad	–	–	–	–	–	–	–	–	–
PFB.CG_IFACE_DISABLE	b8	–	–	–	–	–	–	–	–	–
PFB.WRITE	c3	–	–	–	–	–	–	–	–	–
PFB.READ	c4	–	–	–	–	–	–	–	–	–
PFB.FLUSH	c5	–	–	–	–	–	–	–	–	–
ZCULL.CG	–	58-5a	58-5a	58-5a	58-5a	58-5a	58-5a	??	5d-5f	5d-5f
VATTR.CG	–	–	–	–	–	–	–	??	84-86	84-86
STRMOUT.CG	–	–	–	–	–	–	–	??	87-89	87-89
CLIPID.CG	–	–	–	–	–	–	–	??	8a-8c	8a-8c
ENG2D.CG	–	60-62	60-62	60-62	60-62	60-62	60-62	??	8d-8f	8d-8f
VCLIP.CG	–	–	–	–	–	–	–	??	90-92	90-92
RMASK.CG	–	–	–	–	–	–	–	??	93-95	93-95
TRAST.CG	–	63-65	63-65	63-65	63-65	63-65	63-65	??	96-98	96-98
TEX.CG	–	66-68	66-68	66-68	66-68	66-68	66-68	??	99-9b	99-9b
TEX.CG_IFACE_DISABLE	dd	–	–	–	–	–	–	–	–	–
TEX.UNK6.???	df	7d	7d	7d	7d	7d	75	??	ad	ad
CCACHE.CG_IFACE_DISABLE	ea	–	–	–	–	–	–	–	–	–
PSEC.PM_TRIGGER_ALT	–	–	–	–	–	–	–	–	c4	c4
PSEC.WRCACHE_FLUSH_ALT	–	–	–	–	–	–	–	–	c5	c5

Table 2.13 – continued from previou

signal	G80	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79
PSEC.FALCON	–	–	–	–	–	–	–	–	c6-d9	c6-d9
PCOUNTER.TRAILER	ee-ff	8c-9f	8c-9f	8c-9f	8c-9f	8c-9f	8c-9f	ec-ff	ec-ff	ec-ff

Shader clock

- 0x00-0x03: MPC GROUP 0
- 0x04-0x07: MPC GROUP 1
- 0x08-0x0b: MPC GROUP 2
- 0x0c-0x0f: MPC GROUP 3
- [XXX]
- 0x13-0x14: PCOUNTER.USER [GT215:]
- 0x2e-0x3f: PCOUNTER.TRAILER [G80]
- 0x2c-0x3f: PCOUNTER.TRAILER [G84:]

Memory clock

MCP7x don't have this set. MCP89 does.

signal	G80	G84	G86	G92	G94	G96	G98	G200	GT215	GT216	GT218	MCP89	documentation
PFB.UNK6.CG_IFACE_DISABLE	–	–	–	–	–	–	–	–	–	–	–	–	
PFB.UNK6.CG	–	14-16	14-16	14-16	14-16	14-16	14-16	??	1a-1c	1a-1c	1a-1c	??	
PCOUNTER,USER	–	–	–	–	–	–	–	–	3b-3c	3b-3c	37-38	6a-6b	pcounter/intro.txt
PCOUNTER.TRAILER	e-3f	4c-5f	4c-5f	4c-5f	4c-5f	4c-5f	4c-5f	6c-7f	6c-7f	6c-7f	6c-7f	ec-ff	pcounter/intro.txt

Core clock C

signal	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79	GT216	GT218	GT215	MCP89	documentation
PBSP.USER	??	??	–	??	??	–	00-07	–	–	–	–	–	–	[also on core clock D]
PVP2.USER	??	??	–	??	??	–	08-0f	–	–	–	–	–	–	[also on core clock D]
VCLIP.???	20	20	20	20	20	20	??	??	??	??	??	??	??	
VCLIP.???	21	21	21	21	21	21	??	??	??	??	??	??	??	
VATTR.CG	24-26	24-26	24-26	24-26	24-26	24-26	??	–	–	–	–	–	–	[also on core B]
STR-MOUT.CG	27-29	27-29	27-29	27-29	27-29	27-29	??	–	–	–	–	–	–	[also on core B]
VCLIP.CG	2a-2c	2a-2c	2a-2c	2a-2c	2a-2c	2a-2c	??	–	–	–	–	–	–	[also on core B]
VUC_IDLE	??	??	??	??	??	–	34	–	–	–	–	–	–	vdec/vuc/perf.txt
VUC_SLEEP	??	??	??	??	??	–	36	–	–	–	–	–	–	vdec/vuc/perf.txt
VUC_WATCHDOG	??	??	??	??	??	–	38	–	–	–	–	–	–	vdec/vuc/perf.txt
VUC_USER_PULSE	??	??	??	??	??	–	39	–	–	–	–	–	–	vdec/vuc/perf.txt
VUC_USER_CONT	??	??	??	??	??	–	3a	–	–	–	–	–	–	vdec/vuc/perf.txt
PSEC.PM_TRIGGER_ALT	–	–	–	–	–	37	–	–	–	–	–	–	–	[this and other PSEC stuff on core clock B on MCP*]
PSEC.WRCACHE_FLUSH_ALT	–	–	–	–	–	38	–	–	–	–	–	–	–	
PSEC.FALCON	–	–	–	–	–	39-4c	–	–	–	–	–	–	–	
PCOUNTER.USER	–	–	–	–	–	–	–	–	–	10-11	10-11	10-11	10-11	pcounter/intro.txt
PCOPY.PM_TRIGGER_ALT	–	–	–	–	–	–	–	–	–	1d	1d	1d	1d	
PCOPY.WRCACHE_FLUSH_ALT	–	–	–	–	–	–	–	–	–	1e	1e	1e	1e	
PCOPY.FALCON	–	–	–	–	–	–	–	–	–	1f-32	1f-32	1f-32	1f-32	falcon/perf.txt
PDAE-MON.PM_TRIGGER_ALT	–	–	–	–	–	–	–	–	–	3e	3e	3e	3e	
PDAE-MON.WRCACHE_FLUSH_ALT	–	–	–	–	–	–	–	–	–	3f	3f	3f	3f	
PDAE-MON.FALCON	–	–	–	–	–	–	–	–	–	40-53	40-53	40-53	40-53	falcon/perf.txt
PCOUNTER.TRAILER	4c-5f	4c-5f	4c-5f	4c-5f	4c-5f	6c-7f	6c-7f	0c-1f	0c-1f	6c-7f	6c-7f	6c-7f	6c-7f	pcounter/intro.txt

Vdec clock (VP2)

signal	G84	G86	G92	G94	G96	G200	documentation
PVP2_USER_0	??	??	00-07	??	??	00-07	vdec/vp2/intro.txt
PVP2.CG_IFACE_DISABLE	28	28	28	28	r28	??	what?
PCOUNTER.TRAILER	ac-bf	ac-bf	ac-bf	ac-bf	ac-bf	ac-bf	pcounter/intro.txt

Vdec clock (VP3/VP4)

signal	G98	MCP77	MCP79	GT215	GT216	GT218	MCP89	documentation
PCOUNTER.USER	–	–	–	10-11	10-11	10-11	10-11	pcounter/intro.txt
PVLD.FALCON	10-23	10-23	10-23	16-29	16-29	16-29	16-29	falcon/perf.txt
PPPP.FALCON	40-53	40-53	40-53	2a-3d	2a-3d	2a-3d	2a-3d	falcon/perf.txt
VUC_IDLE	5d	??	??	??	88	??	??	vdec/vuc/perf.txt
VUC_SLEEP	5e	??	??	??	89	??	??	vdec/vuc/perf.txt
VUC_WATCHDOG	5f	??	??	??	8a	??	??	vdec/vuc/perf.txt
VUC_USER_CONT	60	??	??	??	8b	??	??	vdec/vuc/perf.txt
VUC_USER_PULSE	61	??	??	??	8c	??	??	vdec/vuc/perf.txt
PPDEC.FALCON	8e-a1	8e-a1	8e-a1	3e-51	3e-51	3e-51	3e-51	falcon/perf.txt
PVCOMP.FALCON	–	–	–	–	–	–	52-65	falcon/perf.txt
PVLD.???	??	??	??	??	54-58	??	??	
PPPP.???	??	??	??	??	5f-7e	??	??	
PPDEC.XFRM.???	??	??	??	??	a0-a4	??	??	
PPDEC.UNK580.???	??	??	??	??	ad-af	??	??	
PPDEC.UNK680.???	??	??	??	??	b6	??	??	
PVLD.CRYPT.???	??	??	??	??	c0-c5	??	??	
PCOUNTER.TRAILER	ac-bf	ac-bf	ac-bf	cc-df	cc-df	cc-df	ec-ff	pcounter/intro.txt

Core clock D

signal	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79	GT2
PBSP.USER	??	??	00-07	??	??	–	–	–	–	–
PVP2.USER	??	??	08-0f	??	??	–	–	–	–	–
PFB.CG	10-12	10-12	10-12	10-12	10-12	00-02	??	00-02	00-02	00-0
???	??	??	??	??	??	07	??	??	??	??
MMU.CG	3a-3c	3a-3c	3a-3c	3a-3c	3a-3c	1d-1f	??	24-26	24-26	1d-1f
PBSP.CG	5b-5d	3d-3f	63-65	5b-5d	5b-5d	–	??	–	–	–
???	??	??	??	??	??	22	??	??	??	??
???	??	??	??	??	??	23	??	??	??	??
???	??	??	??	??	??	24	??	??	??	??
???	??	??	??	??	??	2c	??	??	??	??
???	??	??	??	??	??	2e	??	??	??	??
???	??	??	??	??	??	30	??	??	??	??
???	??	??	??	??	??	32	??	??	??	??
PCOUNTER.USER	–	–	–	–	–	–	–	–	–	4f-50
MMU.BIND	??	5a	??	??	??	34	??	32	32	5d
PFB_WRITE	??	6f	??	??	??	4b	75	40	40	7d
PFB_READ	??	70	??	??	??	4c	76	41	41	7e
PFB_FLUSH	??	71	??	??	??	4d	77	42	42	7f
PVLD.PM_TRIGGER_ALT	–	–	–	–	–	65	–	6d	6f	9a
PVLD.WRCACHE_FLUSH_ALT	–	–	–	–	–	66	–	6e	70	9b
PPPP.PM_TRIGGER_ALT	–	–	–	–	–	71	–	79	7b	a7
PPPP.WRCACHE_FLUSH_ALT	–	–	–	–	–	72	–	7a	7c	a8

Table 2.14 – continued from previo

signal	G84	G86	G92	G94	G96	G98	G200	MCP77	MCP79	GT2
PPDEC.PM_TRIGGER_ALT	–	–	–	–	–	8c	–	94	96	b4
PPDEC.WRCACHE_FLUSH_ALT	–	–	–	–	–	8d	–	95	97	b5
PVCOMP.PM_TRIGGER_ALT	–	–	–	–	–	–	–	–	–	–
PVCOMP.WRCACHE_FLUSH_ALT	–	–	–	–	–	–	–	–	–	–
IREDIR_STATUS	–	–	–	–	–	–	–	–	–	c6
IREDIR_HOST_REQ	–	–	–	–	–	–	–	–	–	c7
IREDIR_TRIGGER_DAEMON	–	–	–	–	–	–	–	–	–	c8
IREDIR_TRIGGER_HOST	–	–	–	–	–	–	–	–	–	c9
IREDIR_PMC	–	–	–	–	–	–	–	–	–	ca
IREDIR_INTR	–	–	–	–	–	–	–	–	–	cb
MMIO_BUSY	–	–	–	–	–	–	–	–	–	cc
MMIO_IDLE	–	–	–	–	–	–	–	–	–	cd
MMIO_DISABLED	–	–	–	–	–	–	–	–	–	ce
TOKEN_ALL_USED	–	–	–	–	–	–	–	–	–	cf
TOKEN_NONE_USED	–	–	–	–	–	–	–	–	–	d0
TOKEN_FREE	–	–	–	–	–	–	–	–	–	d1
TOKEN_ALLOC	–	–	–	–	–	–	–	–	–	d2
FIFO_PUT_0_WRITE	–	–	–	–	–	–	–	–	–	d3
FIFO_PUT_1_WRITE	–	–	–	–	–	–	–	–	–	d4
FIFO_PUT_2_WRITE	–	–	–	–	–	–	–	–	–	d5
FIFO_PUT_3_WRITE	–	–	–	–	–	–	–	–	–	d6
INPUT_CHANGE	–	–	–	–	–	–	–	–	–	d7
OUTPUT_2	–	–	–	–	–	–	–	–	–	d8
INPUT_2	–	–	–	–	–	–	–	–	–	d9
THERM_ACCESS_BUSY	–	–	–	–	–	–	–	–	–	da
PCOUNTER.TRAILER	ec-ff	cc-df	ec-ff	ec-ff	ec-ff	ac-bf	8c-9f	ac-bf	ac-bf	ec-ff

2.12.4 Fermi+ signals

Contents

Todo

convert

GF100

HUB domain 2:

- source 0: CTXCTL
  - 0x18: ???
  - 0x1b: ???

    - 0x22-0x27: CTXCTL.USER
- source 1: ???
    - 0x2e-0x2f: ???

HUB domain 6:

- source 1: DISPATCH
    - 0x01-0x06: DISPATCH.MUX
- source 8: CCACHE
    - 0x08-0x0f: CCACHE.MUX
- source 4: UNK6000
    - 0x28-0x2f: UNK6000.MUX
- source 2:
    - 0x36: ???
- source 5: UNK5900
    - 0x39-0x3c: UNK5900.MUX
- source 7: UNK7800
    - 0x42: UNK7800.MUX
- source 0: UNK5800
    - 0x44-0x47: UNK5800.MUX
- source 6:
    - 0x4c: ???

GPC domain 0:

- source 0x16:
    - 0x02-0x09: GPC.TPC.L1.MUX
- source 0x19: TEX.MUX_C_D
    - 0x0a-0x12: GPC.TPC.TEX.MUX_C_D
- source 0: CCACHE.MUX_A
    - 0x15-0x19: GPC.CCACHE.MUX_A
- source 5:
    - 0x1a-0x1f: GPC.UNKC00.MUX
- source 0x14:
    - 0x21-0x28: GPC.TPC.UNK400.MUX
- source 0x17:
    - 0x31-0x38: GPC.TPC.MP.MUX
- source 0x13: TPC.UNK500
    - 0x3a-0x3c: TPC.UNK500.MUX
- source 0xa: PROP

  - 0x40-0x47: GPC.PROP.MUX
- source 0x15: POLY
  - 0x48-0x4d: POLY.MUX
- source 0x11: FFB.MUX_B
  - 0x4f-0x53: GPC.FFB.MUX_B
- source 0xe: ESETUP
  - 0x54-0x57: GPC.ESETUP.MUX
- source 0x1a:
  - 0x5b-0x5e: GPC.TPC.TEX.MUX_A
- source 0x18:
  - 0x61-0x64: GPC.TPC.TEX.MUX_B
- source 0xb: UNKB00
  - 0x66-0x68: GPC.UNKB00.MUX
- source 0xc: UNK600
  - 0x6a: GPC.UNK600.MUX
- source 3: ???
  - 0x6e: ???
- source 8: FFB.MUX_A
  - 0x72: ???
  - 0x74: ???
- source 4:
  - 0x76-0x78: GPC.UNKD00.MUX
- source 6:
  - 0x7c-0x7f: GPC.UNKC80.MUX
- source 0xd: UNK380
  - 0x81-0x83: GPC.UNK380.MUX
- source 0x12:
  - 0x84-0x87: GPC.UNKE00.MUX
- source 0xf: UNK700
  - 0x88-0x8b: GPC.UNK700.MUX
- source 1: CCACHE.MUX_B
  - 0x8e: GPC.CCACHE.MUX_B
- source 0x1c:
  - 0x91-0x93: GPC.UNKF00.MUX
- source 0x10: UNK680
  - 0x95: GPC.UNK680.MUX

- source 0x1b: TPC.UNK300
  - 0x98-0x9b: MUX
- source 2: GPC.CTXCTL
  - 0x9c: ???
  - 0xa1-0xa2: GPC.CTXCTL.TA
  - 0xaf-0xba: GPC.CTXCTL.USER
- source 9: ???
  - 0xbf: ???

PART domain 1:

- source 1: CROP.MUX_A
  - 0x00-0x0f: CROP.MUX_A
- source 2: CROP.MUX_B
  - 0x10-0x16: CROP.MUX_B
- source 3: ZROP
  - 0x18-0x1c: ZROP.MUX_A
  - 0x23: ZROP.MUX_B
- source 0: ???
  - 0x27: ???

GF116 signals

[XXX: figure out what the fuck is going on]

HUB domain 0:

- source 0: ???
- source 1: ???
  - 0x01-0x02: ???

HUB domain 1:

- source 0: ???
  - 0x00-0x02: ???
- source 1: ???
- source 2: ???
  - 0x13-0x14: ???
- source 3: ???
  - 0x16: ???

HUB domain 2:

- source 0: CTXCTL [?]
  - 0x18: CTXCTL ???

  - 0x22-0x25: CTXCTL USER_0..USER_5
- source 1: ???
  - 0x2e-0x2f: ???
- 2: PDAEMON
  - 0x14,0x15: PDAEMON PM_SEL_2,3
  - 0x2c: PDAEMON PM_SEL_0
  - 0x2d: PDAEMON PM_SEL_1
  - 0x30: PDAEMON ???

HUB domain 3:

- source 0: PCOPY[0].???
  - 0x00: ???
  - 0x02: ???
  - 0x38: PCOPY[0].SRC0 ???
- source 1: PCOPY[0].FALCON
  - 0x17,0x18: PM_SEL_2,3
  - 0x2e: PCOPY[0].FALCON ???
  - 0x39: PCOPY[0].FALCON ???
- source 2: PCOPY[0].???
  - 0x12: ???
  - 0x3a: PCOPY[0].SRC2 ???
- source 3: PCOPY[1].???
  - 0x05-0x07: ???
  - 0x3b: PCOPY[1].SRC3 ???
- source 4: PCOPY[1].FALCON
  - 0x19,0x1a: PM_SEL_2,3
  - 0x34: PCOPY[1].FALCON ???
  - 0x3c: PCOPY[1].FALCON ???
- source 5: PCOPY[1].???
  - 0x14: ???
  - 0x16: ???
  - 0x3d: PCOPY[1].SRC5 ???
- source 6: PPDEC.???
  - 0x0c: ???
  - 0x22: ???
  - 0x24: ???
  - 0x3e: ???

- source 7: PPPP.???
  - 0x0a: ???
  - 0x1d: ???
  - 0x1f: ???
  - 0x3f: ???
- source 8: PVLD.???
  - 0x0e-0x10: ???
  - 0x27: ???
  - 0x29: ???
  - 0x40: ???

HUB domain 4:

- 0: PPDEC.???
- 1: PPDEC.FALCON
- 2: PPPP.???
- 3: PPPP.FALCON
- 4: PVLD.???
- 5: PVLD.FALCON

HUB domain 4 signals:

- 0x00-0x03: PPPP.SRC2 ???
- 0x06-0x07: PPDEC.SRC0 ???
- 0x09: PVLD.SRC4 ???
- 0x0b: PVLD.SRC4 ???
- 0x0c,0x0d: PPPP.FALCON PM_SEL_2,3
- 0x0e,0x0f: PPDEC.FALCON PM_SEL_2,3
- 0x10,0x11: PVLD.FALCON PM_SEL_2,3
- 0x16-0x17: PPPP.FALCON ???
- 0x1c-0x1d: PPDEC.FALCON ???
- 0x1e: PVLD.FALCON ???
- 0x24-0x25: PPDEC.SRC0 ???
- 0x26: PPDEC.FALCON ???
- 0x27: PPPP.SRC2 ???
- 0x28: PPPP.FALCON ???
- 0x29: PVLD.SRC4 ???
- 0x2a: PVLD.FALCON ???

HUB domain 5 sources:

- 0: ???

HUB domain 5 signals:

- 0x00: SRC0 ???
- 0x05-0x06: SRC0 ???
- 0x09: SRC0 ???
- 0x0c: SRC0 ???

HUB domain 6 sources:

- 0: ???
- 1: ???
- 2: ???
- 3: ???
- 4: ???
- 5: ???
- 6: ???
- 7: ???
- 8: ???

HUB domain 6 signals:

- 0x0a-0x0b: SRC8 ???
- 0x36: SRC2 ???
- 0x39: SRC5 ???
- 0x45: SRC0 ???
- 0x47: SRC0 ???
- 0x4c: SRC6 ???

2.13 Display subsystem

Contents:

2.13.1 NV1 display subsystem

Contents:

2.13.2 NV3:G80 display subsystem

Contents:

VGA stack

Contents

Introduction

A dedicated RAM made of 0x200 8-bit cells arranged into a hw stack. NFI what it is for, apparently related to VGA. Present on NV41+ cards.

MMIO registers

On NV41:G80, the registers are located in PBUS area:

- 001380 VAL
- 001384 CTRL
- 001388 CONFIG
- 00138c SP

They are also aliased in the VGA CRTC register space:

- CR90 VAL
- CR91 CTRL

On G80+, the registers are located in PDISPLAY.VGA area:

- 619e40 VAL
- 619e44 CTRL
- 619e48 CONFIG
- 619e4c SP

And aliased in VGA CRTC register space, but in a different place:

- CRA2 VAL
- CRA3 CTRL

Description

The stack is made of the following data:

- an array of 0x200 bytes [the actual stack]
- a write shadow byte, WVAL [G80+ only]
- a read shadow byte, RVAL [G80+ only]

- a 10-bit stack pointer [SP]
- 3 config bits: - push mode: auto or manual - pop mode: auto or manual - manual pop mode: read before pop or read after pop
- 2 sticky error bits: - stack underflow - stack overflow

The stack grows upwards. The stack pointer points to the cell that would be written by a push. The valid values for stack pointer are thus 0-0x200, with 0 corresponding to an empty stack and 0x200 to a full stack. If stack is ever accessed at position >= 0x200 [which is usually an error], the address wraps modulo 0x200.

There are two major modes the stack can be operated in: auto mode and manual mode. The mode settings are independent for push and pop accesses - one can use automatic pushes and manual pops, for example. In automatic mode, the read/write access to the VAL register automatically performs the push/pop operation. In manual mode, the push/pop needs to be manually triggered in addition to accessing the VAL reg. For manual pushes, the push should be triggered after writing the value. For pops, the pop should be triggered before or after reading the value, depending on selected manual pop mode.

The stack also keeps track of overflow and underflow errors. On NV41:G80, while these error conditions are detected, the offending access is still executed [and the stack pointer wraps]. On G80+, the offending access is discarded. The error status is sticky. On NV41:G80, it can only be cleared by poking the CONFIG register clear bits. On G80+, the overflow status is cleared by executing a pop, and the underflow status is cleared by executing a push.

Stack access registers

The stack data is read or written through the VAL register:

MMIO 0x001380 / CR 0x90: VAL [NV41:G80]

MMIO 0x619e40 / CR 0xa2: VAL [G80-] Accesses a stack entry. A write to this register stored the low 8 bits of written data as a byte to be pushed. If automatic push mode is set, the value is pushed immediately. Otherwise, it is pushed after PUSH_TRIGGER is set. A read from this register returns popped data [causing a pop in the process if automatic pop mode is set]. If manual read-before-pop mode is in use, the returned byte is the byte that the next POP_TRIGGER would pop. In manual pop-before-read, it is the byte that the last POP_TRIGGER popped.

The CTRL register is used to manually push/pop the stack and check its status:

MMIO 0x001384 / CR 0x91: CTRL [NV41:G80]

MMIO 0x619e44 / CR 0xa3: CTRL [G80-]

- bit 0: PUSH_TRIGGER - when written as 1, executes a push. Always reads as 0.
- bit 1: POP_TRIGGER - like above, for pop.
- bit 4: EMPTY - read-only, reads as 1 when SP == 0.
- bit 5: FULL - read-only, reads as 1 when SP >= 0x200.
- bit 6: OVERFLOW - read-only, the sticky overflow error bit
- bit 7: UNDERFLOW - read-only, the sticky underflow error bit

= Stack configuration registers =

To configure the stack, the CONFIG register is used:

MMIO 0x001388: CONFIG [NV41:G80]

MMIO 0x619e48: CONFIG [G80-]

- bit 0: PUSH_MODE - selects push mode [see above]

  - 0: MANUAL
  - 1: AUTO
- bit 1: POP_MODE - selects pop mode [see above]
  - 0: MANUAL
  - 1: AUTO
- bit 2: MANUAL_POP_MODE - for manual pop mode, selects manual pop submode. Unused for auto pop mode.
  - 0: POP_READ - pop before read
  - 1: READ_POP - read before pop
- bit 6: OVERFLOW_CLEAR [NV41:G80] - when written as 1, clears CTRL.OVERFLOW to 0. Always reads as 0.
- bit 7: UNDERFLOW_CLEAR [NV41:G80] - like above, for CTRL.UNDERFLOW

The stack pointer can be accessed directly by the SP register:

MMIO 0x00138c: SP [NV41:G80]

MMIO 0x619e4c: SP [G80-] The stack pointer. Only low 10 bits are valid.

Internal operation

NV41:G80 VAL write:

```
if (SP >= 0x200)
        CTRL.OVERFLOW = 1;
STACK[SP] = val;
if (CONFIG.PUSH_MODE == AUTO)
        PUSH();
```

NV41:G80 PUSH:

```
SP++;
```

NV41:G80 VAL read:

```
if (SP == 0)
        CTRL.UNDERFLOW = 1;
if (CONFIG.POP_MODE == AUTO) {
        POP();
        res = STACK[SP];
} else {
        if (CONFIG.MANUAL_POP_MODE == POP_READ)
                res = STACK[SP];
        else
                res = STACK[SP-1];
}
```

NV41:G80 POP:

```
SP--;
```

G80+ VAL write:

```
WVAL = val;
if (CONFIG.PUSH_MODE == AUTO)
        PUSH();
```

G80+ PUSH:

```
if (SP >= 0x200)
        CTRL.OVERFLOW = 1;
else
        STACK[SP++] = WVAL;
CTRL.UNDERFLOW = 0;
```

G80+ VAL read:

```
if (CONFIG.POP_MODE == AUTO) {
        POP();
        res = RVAL;
} else {
        if (CONFIG.MANUAL_POP_MODE == POP_READ || SP == 0)
                res = RVAL;
        else
                res = STACK[SP-1];
}
```

G80+ POP:

```
if (SP == 0)
        CTRL.UNDERFLOW = 1;
else
        RVAL = STACK[--SP];
CTRL.OVERFLOW = 0;
```

2.13.3 G80 display subsystem

Contents:

PDISPLAY's monitoring engine

Contents

Todo

write me

Introduction

Todo

write me

falcon parameters

Present on:

v0: GF119:GK104

v1: GK104:GK110

v2: GK110+

BAR0 address: 0x627000

PMC interrupt line: 26 [shared with the rest of PDISPLAY], also INTR_HOST_SUMMARY bit 8

PMC enable bit: 30 [all of PDISPLAY]

Version:

v0,v1: 4

v2: 4.1

Code segment size: 0x4000

Data segment size: 0x2000

Fifo size: 3

Xfer slots: 8

Secretful: no

Code TLB index bits: 8

Code ports: 1

Data ports: 4

Version 4 unknown caps: 31, 27

Unified address space: no

IO adressing type: full

Core clock: ???

Fermi VM engine: none

Fermi VM client: HUB 0x03 [shared with rest of PDISPLAY]

Interrupts:

Line	Type	Present on	Name	Description
12	level	all	PDISPLAY	DISPLAY_DAEMON-routed interrupt
13	level	all	FIFO	
14	level	all	???	520? 524 apparently not required
15	level	v1-	PNVIO	DISPLAY_DAEMON-routed interrupt, but also 554?

Status bits:

Bit	Name	Description
0	FALCON	*Falcon unit*
1	MEMIF	*Memory interface*

IO registers: *MMIO registers*

Todo

more interrupts?

Todo

interrupt refs

Todo

MEMIF interrupts

Todo

determine core clock

MMIO registers

Address	Present on	Name	Description
0x627000:0x627400	all	N/A	*Falcon registers*
0x627400	all	???	[alias of 610018]
0x627440+i*4	all	FIFO_PUT	
0x627450+i*4	all	FIFO_GET	
0x627460	all	FIFO_INTR	
0x627464	all	FIFO_INTR_EN	
0x627470+i*4	all	RFIFO_PUT	
0x627480+i*4	all	RFIFO_GET	
0x627490	all	RFIFO_STATUS	
0x6274a0	v1-	???	[ffffffff/ffffffff/0]
0x627500+i*4	all	???	
0x627520	v1-?	???	interrupt 14
0x627524	v1-	???	[0/ffffffff/0]
0x627550	v1-	???	[2710/ffffffff/0]
0x627554	v1-	???	interrupt 15 [0/1/0]
0x627600:0x627680	all	MEMIF	*Memory interface*
0x627680:0x627700	all	-	[alias of 627600+]

Todo

refs

G80 VGA mutexes

Contents

Introduction

Dedicated mutex support hardware supporting trylock and unlock operations on 64 mutexes by 2 clients. Present on G80+ cards.

MMIO registers

On G80+, the registers are located in PDISPLAY.VGA area:

- 619e80 MUTEX_TRYLOCK_A[0]
- 619e84 MUTEX_TRYLOCK_A[1]
- 619e88 MUTEX_UNLOCK_A[0]
- 619e8c MUTEX_UNLOCK_A[1]
- 619e90 MUTEX_TRYLOCK_B[0]
- 619e94 MUTEX_TRYLOCK_B[1]
- 619e98 MUTEX_UNLOCK_B[0]
- 619e9c MUTEX_UNLOCK_B[1]

Operation

There are 64 mutexes and 2 clients. The clients are called A and B. Each mutex can be either unlocked, locked by A, or locked by B at any given moment. Each of the clients has two register sets: TRYLOCK and UNLOCK. Each register set contains two MMIO registers, one controlling mutexes 0-31, the other mutexes 32-63. Bit i of a given register corresponds directly to mutex i or i+32.

Writing a value to the TRYLOCK register will execute a trylock operation on all mutexes whose corresponding bit is set to 1. The trylock operation makes an unlocked mutex locked by the requesting client, and does nothing on an already locked mutex.

Writing a value to the UNLOCK register will likewise execute an unlock operation on selected mutexes. The unlock operation makes a mutex locked by the requesting client unlocked. It doesn't affect mutexes that are unlocked or locked by the other client.

Reading a value from either the TRYLOCK or UNLOCK register will return 1 for mutexes locked by the requesting client, 0 for unlocked mutexes and mutexes locked by the other client.

MMIO 0x619e80+i*4, i < 2: MUTEX_TRYLOCK_A Writing executes the trylock operation as client A, treating the written value as a mask of mutexes to lock. Reading returns a mask of mutexes locked by client A. Bit j of the value corresponds to mutex i*32+j.

MMIO 0x619e88+i*4, i < 2: MUTEX_UNLOCK_A Like MUTEX_TRYLOCK_A, but executes the unlock operation on write.

MMIO 0x619e90+i*4, i < 2: MUTEX_TRYLOCK_B Like MUTEX_TRYLOCK_A, but for client B.

MMIO 0x619e98+i*4, i < 2: MUTEX_UNLOCK_B Like MUTEX_UNLOCK_A, but for client B.

Todo

convert glossary

CHAPTER 3

nVidia Resource Manager documentation

Contents:

3.1 PMU

PMU is NVIDIA's firmware for PDAEMON, used for DVFS and several other power-management related functions.

Contents:

3.1.1 SEQ Scripting ISA

Contents

Introduction

NVIDIA uses PDAEMON for power-management related functions, including DVFS. For this they extended the firmware, PMU, with a scripting language called seq. Scripts are uploaded through *falcon data I/O*.

SEQ conventions

Operations are represented as 32-bit opcodes, follwed by 0 or more 32-bit parameters. The opcode is encoded as follows:

- Bit 0-7: operation
- Bit 31-16: total operation length in 32-bit words (# parameters + 1)

A script ends with 0x0. In the pseudo-code in the rest of this document, the following conventions hold:

- $r3 is reserved as the script program counter, aliased pc
- op aliases *pc & 0xffff
- params aliases (*pc & 0xffff0000) >> 16
- param[] points to the first parameter, the first word after *pc
- PMU reserves 0x5c bytes on the stack for general usage, starting at sp+0x24
- scratch[] is a pointer to scratchpad memory from 0x3e0 onward.

Stack layout

address	Type	Alias	Description
0x00-0x20	u32[9]		Callers $r[0]..$r[8]
0x24	u32	*packet.data	Pointer to data structure
0x2a	u16	in_words	Number of words in the program.
0x2c	u32	*in_end	Pointer to the end of the program
0x30	u32	insn_len	Length of the currently executed instruction
0x54	u32	*head_vert	&(PDISPLAY.HEAD_STAT[0].VERT)+head_off
0x58	u32	head_off	Offset for current HEAD from PDISPLAY[0]
0x5c	u32	*in_start	Pointer to the start of the program
0x62	u16	word_exit	
0x64	u32	timestamp	

Scratch layout

Type	Name	Description
u8	out_words	Size of the `out` memory section, in 32-bit units
u24		Unused, padding
u32	*out_start	Pointer to the `out` memory section
u8	flag_eq	1 if compare val_last == param
u8	flag_lt	1 if compare val_last < param
u16		Unused, padding
u32	val_last	Holds the register last read or written. Can be set manually
u32	reg_last	The value last read or written. Can be set manually
u32	val_ret	Holds a return value written back to sp[80] after successful execution

Opcodes

XXX: Gaps are all sorts of exit routines. Not clear how the exit procedure works wrt status propagation.

Opcode	Params	Description
0x00	1	*SET last value*
0x01	1	*SET last register*
0x02	1	*OR last value*
0x03	1	*OR last register*
0x04	1	*AND last value*
0x05	1	*AND last register*
0x06	1	*ADD last value*
0x07	1	*ADD last register*
0x08	1	*SHIFT last value*
0x09	1	*SHIFT last register*
0x0a	0	*READ last register*
0x0b	1	*READ last register*
0x0c	1	*READ last register*
0x0d	0	*WRITE last register*
0x0e	1	*WRITE last register*
0x0f	1	*WRITE last register*
0x10	0	*EXIT*
0x11	0	*EXIT*
0x12	0	*EXIT*
0x13	1	*WAIT*
0x14	2	*WAIT STATUS*
0x15	2	*WAIT BITMASK last*
0x16	1	*EXIT*
0x17	1	*COMPARE last value*
0x18	1	*BRANCH EQ*
0x19	1	*BRANCH NEQ*
0x1a	1	*BRANCH LT*
0x1b	1	*BRANCH GT*
0x1c	1	*BRANCH*
0x1d	0	*IRQ_DISABLE*
0x1e	0	*IRQ_ENABLE*
0x1f	1	*AND last value, register*
0x20	1	*FB PAUSE/RESUME*
0x21	2n	*SET register(s)*
0x22	1	*WRITE OUT last value*
0x23	1	*WRITE OUT indirect last value*
0x24	2	*WRITE OUT*
0x25	2	*WRITE OUT indirect*
0x26	1	*READ OUT last value*
0x27	1	*READ OUT indirect last value*
0x28	1	*READ OUT last register*
0x29	1	*READ OUT indirect last register*
0x2a	2	*ADD OUT*
0x2b	1	*COMPARE OUT*
0x2c	1	*OR last value, register*
0x2d	2	XXX: Display-related
0x2e	1	*WAIT*
0x2f	0	*EXIT*
0x30	1	*OR OUT last value*
0x31	1	*OR OUT indirect last value*

Continued on next page

Table 3.1 – continued from previous page

Opcode	Params	Description
0x32	1	*AND OUT last value*
0x33	1	*AND OUT indirect last value*
0x34	1	*WRITE OUT TIMESTAMP*
0x35	1	*WRITE OUT TIMESTAMP indirect*
0x38	0	NOP
0x3b	1	*ADD last value, OUT*
0x3c	1	*ADD last value, OUT indirect*
other	0	*EXIT*

Memory

SET last

Set the last register/value in scratch memory.

Opcode: 0x00 0x01

Parameters: 1

Operation:

```
    scratch[3 + (op & 1)] = param[0];
```

READ last register

Do a read of the last register and/or a register/offset given by parameter 1, and write back to the last value.

Opcode: 0x0a 0x0b 0x0c

Parameters: 0/1

Operation:

```
    reg = 0;
    if(op == 0xa || op == 0xc)
            reg += scratch->reg_last;
    if(op == 0xb || op == 0xc)
            reg += param[0];

    scratch->val_last = mmrd(reg);
```

WRITE last register

Do a write to the last register and/or a register/offset given by parameter 1 of the last value.

Opcode: 0x0d 0x0e 0x0f

Parameters: 0/1

Operation:

```
    reg = 0;
    if(op == 0xd || op == 0xf)
            reg += scratch->reg_last;
```

```
    if(op == 0xe || op == 0xf)
            reg += param[0];

    mmwr_seq(reg, scratch->val_last);
```

SET register(s)

For each register/value pair, this operation performs a (locked) register write. through

Opcode: 0x21

Parameters: 2n for n > 0

Operation:

```
    IRQ_DISABLE;
    for (i = 0; i < params; i += 2) {
            mmwr_unlocked(param[i],param[i+1]);
    }
    IRQ_ENABLE;
    scratch->reg_last = param[i-2];
    scratch->val_last = param[i-1];
```

WRITE OUT last value

Write a word to the OUT memory section, offset by the first parameter. For indirect read, the parameter points to an 8-bit value describing the offset of the address to write to.

Opcode: 0x22 0x23

Parameters: 1

Operation:

```
    if (!out_start)
            exit(pc);
    idx = $param[0].u08;
    if (idx >= out_words.u08)
            exit(pc);

    /* Indirect */
    if (op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }

    out_start[idx] = scratch->val_last;
```

WRITE OUT

Write a word to the OUT memory section, offset by the first parameter. For indirect read, the parameter points to an 8-bit value describing the offset of the address to write to.

Opcode: 0x24 0x25

Parameters: 2

Operation:

```
    if (!out_start)
            exit(pc);
    idx = $param[0].u08;
    if (idx >= out_words.u08)
            exit(pc);

    /* Indirect */
    if (op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }

    out_start[idx] = param[1];
```

READ OUT last value

Read a word from the OUT memory section, into the val_last location. Parameter is the offset inside the out page. For indirect read, the parameter points to an 8-bit value describing the offset of the read out value.

Opcode: 0x26 0x27

Parameters: 1

Operation:

```
    if (!out_start)
            exit(pc);
    idx = $param[0].u08;
    if (idx >= out_words.u08)
            exit(pc);

    /* Indirect */
    if (op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }

    scratch->val_last = out_start[idx];
```

READ OUT last register

Read a word from the OUT memory section, into the reg_last location. Parameter is the offset inside the out page. For indirect read, the parameter points to an 8-bit value describing the offset of the read out value.

Opcode: 0x28 0x29

Parameters: 1

Operation:

```
    if (!out_start)
            exit(pc);
    idx = $param[0].u08;
    if (idx >= out_words.u08)
```

```
            exit(pc);

    /* Indirect */
    if (op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }

    scratch->reg_last = out_start[idx];
```

WRITE OUT TIMESTAMP

Write the current timestamp to the OUT memory section, offset by the first parameter. For indirect read, the parameter points to an 8-bit value describing the offset of the address to write to.

Opcode: 0x34 0x35

Parameters: 2

Operation:

```
    if (!out_start)
            exit(pc);
    idx = $param[0].u08;
    if (idx >= out_words.u08)
            exit(pc);

    /* Indirect */
    if (op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }

    call_timer_read(&value)
    out_start[idx] = value;
```

Arithmetic

OR last

OR the last register/value in scratch memory.

Opcode: 0x02 0x03

Parameters: 1

Operation:

```
    scratch[3 + (op & 1)] |= param[0];
```

AND last

AND the last register/value in scratch memory.

Opcode: 0x04 0x05

Parameters: 1

Operation:

```
scratch[3 + (op & 1)] &= param[0];
```

ADD last

ADD the last register/value in scratch memory.

Opcode: 0x06 0x07

Parameters: 1

Operation:

```
scratch[3 + (op & 1)] += param[0];
```

SHIFT-left last

Shift the last register/value in scratch memory to the left, negative parameter shifts right.

Opcode: 0x08 0x09

Parameters: 1

Operation:

```
if(param[0].s08 >= 0) {
        scratch[3 + (op & 1)] <<= sex($param[0].s08);
        break;
} else {
        scratch[3 + (op & 1)] >>= -sex($param[0].s08);
        break;
}
```

AND last value, register

AND the last value with value read from register.

Opcode: 0x1f

Parameters: 1

Operation:

```
scratch->val_last &= mmrd(param[0]);
```

ADD OUT

ADD an immediate value to a value in the OUT memory region.

Opcode: 0x2a

Parameters: 2

Operation:

```
    if (!out_start)
            exit(pc);
    idx = param[0];
    if (idx >= out_len)
            exit(pc);

    out_start[idx] += param[1];
```

OR last value, register

OR the last value with value read from register

Opcode: 0x2c

Parameters: 1

Operation:

```
    scratch->val_last |= mmrd(param[0]);
```

OR OUT last value

OR the contents of last_val with a value in the OUT memory region.

Opcode: 0x30 0x31

Parameters: 1

Operation:

```
    if (!out_start)
            exit(pc);
    idx = param[0];
    if (idx >= out_len)
            exit(pc);

    /* Indirect */
    if (op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }

    out_start[idx] |= scratch->val_last;
```

ADD last value, OUT

Add a value in OUT to val_last.

Opcode: 0x3b 0x3c

Parameters: 1

Operation:

```
    if (!out_start)
            exit(pc);
    idx = param[0];
    if(idx >= out_len)
            exit(pc);

    /* Indirect
    if(!op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }
    val_last += out_start[idx];
```

AND OUT last value

AND the contents of last_val with a value in the OUT memory region.

Opcode: 0x32 0x33

Parameters: 1

Operation:

```
    if (!out_start)
            exit(pc);
    idx = param[0];
    if (idx >= out_len)
            exit(pc);

    /* Indirect */
    if (op & 0x1) {
            idx = out_start[idx];
            if (idx >= out_words.u08)
                    exit(pc);
    }

    out_start[idx] &= scratch->val_last;
```

Control flow

EXIT

Exit

Opcode: 0x10..0x12 0x16 0x2f

Parameters: 0/1

Operation:

```
    if(op == 0x16)
            exit(param[0].s08);
    else
            exit(-1);
```

COMPARE last value

Compare last value with a parameter. If smaller, set flag_lt. If equal, set flag_eq.

Opcode: 0x17

Parameters: 1

Operation:

```
    flag_eq = 0;
    flag_lt = 0;

    if(scratch->val_last < param[0])
            flag_lt = 1;
    else if(scratch->val_last == param[0])
            flag_eq = 1;
```

BRANCH EQ

When compare resulted in eq flag set, branch to an absolute location in the program.

Opcode: 0x18

Parameters: 1

Operation:

```
    if(flag_eq)
            BRANCH param[0];
```

BRANCH NEQ

When compare resulted in eq flag unset, branch to an absolute location in the program.

Opcode: 0x19

Parameters: 1

Operation:

```
    if(!flag_eq)
            BRANCH param[0];
```

BRANCH LT

When compare resulted in lt flag unset, branch to an absolute location in the program.

Opcode: 0x1a

Parameters: 1

Operation:

```
    if(flag_lt)
            BRANCH param[0];
```

BRANCH GT

When compare resulted in lt and eq flag unset, branch to an absolute location in the program.

Opcode: 0x1b

Parameters: 1

Operation:

```
    if(!flag_lt && !flag_eq)
            BRANCH param[0];
```

BRANCH

Branch to an absolute location in the program.

Opcode: 0x1c

Parameters: 1

Operation:

```
    target = param[0].s16;
    if(target >= in_words)
            exit(target);

    word_exit = $r9.s16
    target &= 0xffff;
    target <<= 2;
    pc = in_start + target;

    if(pc >= in_end)
            exit(in_end);
```

COMPARE OUT

Compare word in OUT with a parameter. If smaller, set flag_lt. If equal, set flag_eq.

Opcode: 0x2b

Parameters: 1

Operation:

```
    if(!out_start)
            exit(pc);

    idx = param[0];
    if(idx >= out_words.u08)
            exit(pc);

    flag_eq = 0;
    flag_lt = 0;

    if(out_start[idx] < param[1])
            flag_lt = 1;
    else if(out_start[idx] == param[1])
            flag_eq = 1;
```

Miscellaneous

WAIT

Waits for desired number of nanoseconds, synchronous for 0x2e.

Opcode: 0x13 0x2e

Parameters: 1

Operation:

```
    if(op == 0x2e)
            mmrd(0);
    call_timer_wait_nf(param[0]);
```

WAIT STATUS

Shifts val_ret left by 1 position, and waits until a status bit is set/unset. Sets flag_eq and the LSB of val_ret on success. The second parameter contains the timeout.The first parameter encodes the desired status.

Old blob

param[0]	Test
0	UNKNOWN(0x01)
1	!UNKNOWN(0x01)
2	FB_PAUSED
3	!FB_PAUSED
4	HEAD0_VBLANK
5	!HEAD0_VBLANK
6	HEAD1_VBLANK
7	!HEAD1_VBLANK
8	HEAD0_HBLANK
9	!HEAD0_HBLANK
10	HEAD1_HBLANK
11	!HEAD1_HBLANK

New blob

In newer blobs (like 337.25), bit 16 encodes negation. Bit 8:10 the status type to wait for, and where applicable bit 0 chooses the HEAD.

param[0]	Test
0x0	HEAD0_VBLANK
0x1	HEAD1_VBLANK
0x100	HEAD0_HBLANK
0x101	HEAD1_HBLANK
0x300	FB_PAUSED
0x400	PGRAPH_IDLE
0x10000	!HEAD0_VBLANK
0x10001	!HEAD1_VBLANK
0x10100	!HEAD0_HBLANK
0x10101	!HEAD1_HBLANK
0x10300	!FB_PAUSED
0x10400	!PGRAPH_IDLE

Todo: Why isn't flag_eq unset on failure? Find out switching point from old to new format?

Opcode: 0x14

Parameters: 2

Operation *OLD BLOB*:

```
    val_ret *= 2;
    test_params[1] = param[0] & 1;
    test_params[2] = I[0x7c4];

    switch ((param[0] & ~1) - 2) {
            default:
                    test_params[0] = 0x01;
                    break;
            case 0:
                    test_params[0] = 0x04;
                    break;
            case 2:
                    test_params[0] = 0x08;
                    break;
            case 4:
                    test_params[0] = 0x20;
                    break;
            case 6:
                    test_params[0] = 0x10;
                    break;
            case 8:
                    test_params[0] = 0x40;
                    break;
    }

    if (call_timer_wait(&input_bittest, test_params, param[1])) {
            flag_eq = 1;
            val_ret |= 1;
    }
```

Operation *NEW BLOB*:

```
    b32 func(b32 *) *f;
    unk3ec[2] <<= 1;

    test_params[2] = 0x1f100; // 7c4
    test_params[1] = (param[0] >> 16) & 0x1;

    switch(param[0] & 0xffff) {
    case 0x0:
            test_params[0] = 0x8;
            f = &input_test
            break;
    case 0x1:
            test_params[0] = 0x20;
            f = &input_test
            break;
    case 0x100:
            test_params[0] = 0x10;
            f = &input_test
            break;
    case 0x101:
            test_params[0] = 0x40;
            f = &input_test
```

```
            break;
    case 0x300:
            test_params[0] = 0x04;
            f = &input_test
            break;
    case 0x400:
            test_params[0] = 0x400;
            f = &pgraph_test;
            break;
    default:
            f = NULL;
            break;
    }

    if(f && timer_wait(f, param, timeout) != 0) {
            unk3e8 = 1;
            unk3ec[2] |= 1;
    }
```

WAIT BITMASK last

Shifts val_ret left by 1 position, and waits until the AND operation of the register pointed in reg_last and the first parameter equals val_last. Sets flag_eq and the LSB of val_ret on success. The first parameter encodes the bitmask to test. The second parameter contains the timeout.

Todo: Why isn't flag_eq unset on failure?

Opcode: 0x15

Parameters: 2

Operation:

```
    b32 seq_cb_wait(b32 parm) {
            return (mmrd(last_reg) & parm) == last_val;
    }

    val_ret *= 2;
    if (call_timer_wait(seq_cb_wait, param[0], param[1]))
            break;

    val_ret |= 1;
    flag_eq = 1;
```

IRQ_DISABLE

Disable IRQs, increment reference counter `irqlock_lvl`

Opcode: 0x1f

Parameters: 1

Operation:

```
    interrupt_enable_0 = interrupt_enable_1 = false;
    irqlock_lvl++;
```

IRQ_ENABLE

Decrement reference counter `irqlock_lvl`, enable IRQs if 0.

Opcode: 0x1f

Parameters: 1

Operation:

```
    if(!irqlock_lvl--)
            interrupt_enable_0 = interrupt_enable_1 = true;
```

FB PAUSE/RESUME

If parameter 1, disable IRQs on PDAEMON and pause framebuffer (memory), otherwise resume FB and enable IRQs.

Opcode: 0x20

Parameters: 1

Operation:

```
    if (param[0]) {
            IRQ_DISABLE;

            /* XXX What does this bit do? */
            mmwrs(0x1610, (mmrd(0x1610) & ~3) | 2);
            mmrd(0x1610);

            mmwrs(0x1314, (mmrd(0x1314) & ~0x10001) | 0x10001);

            /* RNN:PDAEMON.INPUT0_STATUS.FB_PAUSED */
            while (!(RD(0x7c4) & 4));

            mmwr_seq = &mmwr_unlocked;
    } else {
            mmwrs(0x1314, mmrd(0x1314) & ~0x10001);

            while (RD(0x7c4) & 4);

            mmwrs(0x1610, mmrd(0x1610) & ~0x33);
            IRQ_ENABLE;

            mmwr_seq = &mmwrs;
    }
```

CHAPTER 4

envydis documentation

Contents

4.1 Using envydis

`envydis` reads from standard input and prints the disassembly to standard output. By default, input is parsed as sequence space- or comma-separated hexadecimal numbers representing the bytes to disassemble. The options are:

4.1.1 Input format

-w
Instead of sequence of hexadecimal bytes, treat input as sequence of hexadecimal 32-bit words

-W
Instead of sequence of hexadecimal bytes, treat input as sequence of hexadecimal 64-bit words

-i
Treat input as pure binary

4.1.2 Input subranging

-b `<base>`
Assume the start of input to be at address <base> in code segment

-s `<skip>`
Skip/discard that many bytes of input before starting to read code

-l `<limit>`
Don't disassemble more than <limit> bytes.

4.1.3 Variant selection

-m <machine>
Select the ISA to disassemble. One of:

- [****] g80: tesla CUDA/shader ISA
- [***] gf100: fermi CUDA/shader ISA
- [**] gk110: kepler GK110 CUDA/shader ISA
- [***] gm107: maxwell CUDA/shader ISA
- [**] ctx: nv40 and g80 PGRAPH context-switching microcode
- [***] falcon: falcon microcode, used to power various engines on G98+ cards
- [****] hwsq: PBUS hardware sequencer microcode
- [****] xtensa: xtensa variant as used by video processor 2 [g84-gen]
- [***] vuc: video processor 2/3 master/mocomp microcode
- [****] macro: gf100 PGRAPH macro method ISA
- [**] vp1: video processor 1 [nv41-gen] code
- [****] vcomp: PVCOMP video compositor microcode

Where the quality level is:

- [ ]: Bare beginnings
- [*]: Knows a few instructions
- [**]: Knows enough instructions to write some simple code
- [***]: Knows most instructions, enough to write advanced code
- [****]: Knows all instructions, or very close to.

-V <variant>
Select variant of the ISA.

For g80:

- g80: The original G80 [aka compute capability 1.0]
- g84: G84, G86, G92, G94, G96, G98 [aka compute capability 1.1]
- g200: G200 [aka compute capability 1.3]
- mcp77: MCP77, MCP79 [aka compute capability 1.2]
- gt215: GT215, GT216, GT218, MCP89 [aka compute capability 1.2 + d3d10.1]

For gf100:

- gf100: GF100:GK104 cards
- gk104: GK104+ cards

For ctx:

- nv40: NV40:G80 cards
- g80: G80:G200 cards
- g200: G200:GF100 cards

For hwsq:

- nv17: NV17:NV41 cards
- nv41: NV41:G80 cards
- g80: G80:GF100 cards

For falcon:

- fuc0: falcon version 0 [G98, MCP77, MCP79]
- fuc3: falcon version 3 [GT215 and up]
- fuc4: falcon version 4 [GF119 and up, selected engines only]
- fuc5: falcon version 5 [GK208 and up, selected engines only]

For vuc:

- vp2: VP2 video processor [G84:G98, G200]
- vp3: VP3 video processor [G98, MCP77, MCP79]
- vp4: VP4 video processor [GT215:GF119]

-F <feature>
Enable optional ISA feature. Most of these are auto-selected by *-V*, but can also be specified manually. Can be used multiple times to enable several features.

For g80:

- sm11: SM1.1 new opcodes [selected by g84, g200, mcp77, gt215]
- sm12: SM1.2 new opcodes [selected by g200, mcp77, gt215]
- fp64: 64-bit floating point [selected by g200]
- d3d10_1: Direct3D 10.1 new features [selected by gt215]

For gf100:

- gf100op: GF100:GK104 exclusive opcodes [selected by gf100]
- gk104op: GK104+ exclusive opcodes [selected by gk104]

For ctx:

- nv40op: NV40:G80 exclusive opcodes [selected by nv40]
- g80op: G80:GF100 exclusive opcodes [selected by g80, g200]
- callret: call/ret opcodes [selected by g200]

For hwsq:

- nv17f: NV17:G80 flags [selected by nv17, nv41]
- nv41f: NV41:G80 flags [selected by nv41]
- nv41op: NV41 new opcodes [selected by nv41, g80]

For falcon:

- fuc0op: falcon version 0 exclusive opcodes [selected by fuc0]
- fuc3op: falcon version 3+ exclusive opcodes [selected by fuc3, fuc4]
- pc24: 24-bit PC opcodes [selected by fuc4]
- crypt: Cryptographic coprocessor opcodes [has to be manually selected]

For vuc:

- vp2op: VP2 exclusive opcodes [selected by vp2]
- vp3op: VP3+ exclusive opcodes [selected by vp3, vp4]
- vp4op: VP4 exclusive opcodes [selected by vp4]

-O <mode>
Select processor mode.

For g80:

- vp: Vertex program
- gp: Geometry program
- fp: Fragment program
- cp: Compute program

4.1.4 Output format

-n
Disable output coloring

-q
Disable printing address + opcodes.

CHAPTER 5

TODO list

Todo

map out the BAR fully

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/bars.rst, line 88.)

Todo

RE it. or not.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/bars.rst, line 133.)

Todo

It's present on some NV4x

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/bars.rst, line 144.)

Todo

figure out size

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/bars.rst, line 184.)

Todo

figure out NV3

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/bars.rst, line 185.)

Todo

verify G80

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/bars.rst, line 186.)

Todo

MSI

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/bars.rst, line 203.)

Todo

are EVENTS variants right?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 54.)

Todo

cleanup, crossref

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 56.)

Todo

writ me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 60.)

Todo

8, 9, 13 seem used by microcode!

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 279.)

Todo

check variants for 15f4, 15fc

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 280.)

Todo

check variants for 4-7, some NV4x could have it

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 281.)

Todo

check variants for 14, 15

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 282.)

Todo

doc 1084 bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/hwsq.rst, line 283.)

Todo

connect

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pbus.rst, line 46.)

Todo

loads and loads of unknown registers not shown

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pbus.rst, line 73.)

Todo

document other known stuff

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pbus.rst, line 93.)

Todo

cleanup

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pbus.rst, line 101.)

Todo

description, maybe move somewhere else

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pbus.rst, line 180.)

Todo

verify that it's host cycles

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pbus.rst, line 189.)

Todo

nuke this file and write a better one - it sucks.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 39.)

Todo

wrong on NV3]

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 56.)

Todo

this register and possibly some others doesn't get written when poked through actual PCI config accesses - PBUS writes work fine

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 57.)

Todo

NV40 has something at 0x98

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 66.)

Todo

MCP77, MCP79, MCP89 stolen memory regs at 0xf4+

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 67.)

Todo

very incomplete

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 74.)

Todo

is that all?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 97.)

Todo

find it

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pci.rst, line 103.)

Todo

more info

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pfuse.rst, line 18.)

Todo

fill me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pfuse.rst, line 31.)

Todo

unk bitfields

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 99.)

Todo

what is this? when was it introduced? seen non-0 on at least G92

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 108.)

Todo

there are cards where the steppings don't match between registers - does this mean something or is it just a random screwup?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 119.)

Todo

figure out the CS thing, figure out the variants. Known not to exist on NV40, NV43, NV44, C51, G71; known to exist on MCP73

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 205.)

Todo

unknowns

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 240.)

Todo

RE these three

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 302.)

Todo

change all this duplication to indexing

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 326.)

Todo

check

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 412.)

Todo

figure out unknown interrupts. They could've been introduced much earlier, but we only know them from bitscanning the INTR_MASK regs. on GT215+.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 468.)

Todo

unknowns

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 501.)

Todo

document these two

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 503.)

Todo

verify variants for these?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pmc.rst, line 513.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pring.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pring.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pring.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pring.rst, line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/pring.rst, line 39.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/prma.rst, line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/prma.rst, line 44.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/prma.rst, line 48.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/prma.rst, line 52.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/prma.rst, line 56.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/prma.rst, line 60.)

Todo

document that some day].

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/ptimer.rst, line 29.)

Todo

figure these out

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/ptimer.rst, line 215.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/ptimer.rst, line 235.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/ptimer.rst, line 239.)

Todo

document MMIO_FAULT_*

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/ptimer.rst, line 241.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/punits.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/punits.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/bus/punits.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pco
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pco
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pco
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
line 35.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
line 43.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
daemon.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
daemon.rst, line 15.)

Todo

more interrupts?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
daemon.rst, line 88.)

Todo

interrupt refs

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
daemon.rst, line 89.)

Todo

MEMIF interrupts

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
daemon.rst, line 90.)

Todo

determine core clock

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
daemon.rst, line 91.)

Todo

refs

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pdis
daemon.rst, line 121.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pkfu
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pkfu
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/pkfu
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/punl
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/punl
line 15.)

Todo

MEMIF interrupts

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/punl
line 77.)

Todo

determine core clock

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/punl
line 78.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/punl
line 88.)

Todo

figure out unknowns

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/punl
line 104.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/vga.
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/vga.
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/vga.
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/g80/vga.
line 27.)

Todo

regs 0x1c-0xff

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 90.)

Todo

regs 0x1xx and 0x5xx

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 91.)

Todo

regs 0xf0xx

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 92.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 137.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 147.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 153.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 169.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 173.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 179.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 185.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 187.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 296.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 310.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 319.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 344.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 346.)

Todo

some newer DACs have more functionality?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pdac
line 376.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.
line 44.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.
line 63.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 67.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 71.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 75.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 83.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 89.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 93.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 97.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 101.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 105.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 109.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 113.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 117.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 121.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.r
line 125.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.
line 129.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/pfb.
line 133.)

Todo

figure out what the fuck this engine does

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/prm
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/prm
line 30.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/prm
line 43.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/prm
line 63.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/prm
line 113.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv1/prm line 121.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pcrt line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pcrt line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pcrt line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pcrt line 25.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pcrt line 29.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pcrt line 37.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pcrt
line 45.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pran
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pran
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pran
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pran
line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pran
line 29.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/ptv.r
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/ptv.r
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/ptv.r
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pvid
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pvid
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pvid
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/pvid
line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/vga.
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/vga.
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/vga.
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/display/nv3/vga.
line 27.)

Todo

document ljmp/lcall

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/branch.rst
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/crypt.rst,
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/crypt.rst,
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/crypt.rst,
line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/crypt.rst, line 31.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/crypt.rst, line 39.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/crypt.rst, line 47.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/crypt.rst, line 55.)

Todo

document UAS

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/data.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/debug.rst, line 7.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/debug.rst, line 17.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/fifo.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/fifo.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/fifo.rst, line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/fifo.rst, line 32.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/fifo.rst, line 41.)

Todo

figure out interrupt 5

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/intr.rst, line 33.)

Todo

check edge/level distinction on v0

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/intr.rst, line 83.)

Todo

didn't ieX -> isX happen before v4?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/intr.rst, line 194.)

Todo

figure out remaining circuitry

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/intro.rst, line 35.)

Todo

figure out v4 new stuff

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/intro.rst, line 48.)

Todo

figure out v4.1 new stuff

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/intro.rst, line 49.)

Todo

figure out v5 new stuff

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/intro.rst, line 50.)

Todo

document v4 new addressing

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/io.rst, line 42.)

Todo

list incomplete for v4

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/io.rst, line 150.)

Todo

clean. fix. write. move.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/io.rst, line 185.)

Todo

subop e

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/io.rst, line 307.)

Todo

figure out v4+ stuff

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/isa.rst, line 67.)

Todo

long call/branch

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/isa.rst, line 131.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/memif.rst line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/memif.rst line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/memif.rst line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/memif.rst line 32.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/memif.rst line 40.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/memif.rst line 48.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/perf.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/perf.rst, line 15.)

Todo

docs & RE, please

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/perf.rst, line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/perf.rst, line 58.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/proc.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/proc.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/proc.rst, line 121.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/proc.rst, line 129.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/proc.rst, line 137.)

Todo

check interaction of secret / usable flags and entering/exitting auth mode

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/vm.rst, line 24.)

Todo

one more unknown flag on secret engines

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/xfer.rst, line 33.)

Todo

figure out bit 1. Related to 0x10c?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/xfer.rst, line 189.)

Todo

how to wait for xfer finish using only IO?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/xfer.rst, line 194.)

Todo

bits 4-5

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/xfer.rst, line 210.)

Todo

RE and document this stuff, find if there's status for code xfers

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/falcon/xfer.rst, line 212.)

Todo

check for NV4-style mode on GF100

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 27.)

Todo

verify those

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 107.)

Todo

determine what happens on GF100 on all imaginable error conditions

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 109.)

Todo

check channel numbers

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 171.)

Todo

What about GF100?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 195.)

Todo

check the ib size range

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 231.)

Todo

figure out bit 8 some day

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 257.)

Todo

do an exhaustive scan of commands

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 302.)

Todo

didn't mthd 0 work even if sli_active=0?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 340.)

Todo

check pusher reaction on ACQUIRE submission: pause?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 347.)

Todo

check bitfield bounduaries

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 425.)

Todo

check the extra SLI bits

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 427.)

Todo

look for other forms

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/dma-pusher.rst, line 429.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 25.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 33.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 42.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 50.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 56.)

Todo

document me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 60.)

Todo

document me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/g80-pfifo.rst, line 64.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pfifo.rst, line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pfifo.rst, line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pfifo.rst, line 25.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pfifo.rst, line 29.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pfifo.rst, line 35.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pfifo.rst, line 43.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pspoon.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pspoon.rst, line 14.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pspoon.rst, line 22.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/gf100-pspoon.rst, line 30.)

Todo

check if it still holds on GF100

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/intro.rst, line 32.)

Todo

check PIO channels support on NV40:G80

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/intro.rst, line 129.)

Todo

look for GF100 PFIFO endian switch

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/intro.rst, line 189.)

Todo

is it still true for GF100, with VRAM-backed channel control area?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/intro.rst, line 194.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 126.)

Todo

document gray code

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 211.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 215.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 219.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 223.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 227.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 231.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 235.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 239.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 243.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 247.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 253.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 257.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 261.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 267.)

Todo

document me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 271.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 279.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 283.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 287.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 291.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 299.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 303.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 307.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 311.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 315.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 319.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 323.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 327.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 331.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 337.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 341.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 349.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 357.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 365.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv1-pfifo.rst, line 369.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv4-pfifo.rst, line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv4-pfifo.rst, line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv4-pfifo.rst, line 25.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv4-pfifo.rst, line 33.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv4-pfifo.rst, line 39.)

Todo

document me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv4-pfifo.rst, line 43.)

Todo

document me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/nv4-pfifo.rst, line 47.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pcopy.rst, line 114.)

Todo

describe PCOPY

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pcopy.rst, line 116.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pio.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pio.rst, line 14.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pio.rst, line 22.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pio.rst, line 28.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pio.rst, line 34.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/pio.rst, line 42.)

Todo

missing the GF100+ methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 50.)

Todo

verify this on all card families.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 198.)

Todo

verify all of the pseudocode...

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 303.)

Todo

figure this out

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 424.)

Todo

RE timeouts

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 433.)

Todo

is there ANY way to make G80 reject non-DMA object classes?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 450.)

Todo

bit 12 does something on GF100?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 546.)

Todo

check how this is reported on GF100

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/fifo/puller.rst, line 620.)

Todo

what the fuck?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 220.)

Todo

what were the GPIOs for?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 265.)

Todo

verify all sorts of stuff on NV2A

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 336.)

Todo

figure out 3d engine changes

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 385.)

Todo

more changes

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 452.)

Todo

figure out 3d engine changes

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 453.)

Todo

all geometry information unverified

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 478.)

Todo

any information on the RSX?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 480.)

Todo

geometry information not verified for G94, MCP77

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 584.)

Todo

figure out PGRAPH/PFIFO changes

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 662.)

Todo

it is said that one of the GPCs [0th one] has only one TPC on GK106

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 692.)

Todo

what the fuck is GK110B? and GK208B?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 694.)

Todo

GK20A

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 696.)

Todo

GM200

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 698.)

Todo

GM204

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 700.)

Todo

another design counter available on GM107

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/gpu.rst, line 702.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/blit.rst, line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/blit.rst, line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/blit.rst, line 25.)

Todo

write m

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 11.)

Todo

check NV3+

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 142.)

Todo

check if still applies on NV3+

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 181.)

Todo

check NV3+

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 198.)

Todo

check NV3+

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 211.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 261.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 269.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 277.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 285.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ctxobj.
line 293.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/dvd.rst
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/dvd.rst
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/dvd.rst
line 25.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/gdi.rst,
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/gdi.rst,
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/gdi.rst,
line 25.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/gdi.rst, line 31.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/gdi.rst, line 37.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/gdi.rst, line 43.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/gdi.rst, line 49.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifc.rst, line 11.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifc.rst, line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifc.rst, line 27.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifc.rst, line 35.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifc.rst, line 43.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifc.rst, line 51.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifm.rst, line 14.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifm.rst, line 20.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifm.rst, line 26.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/ifm.rst, line 32.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 183.)

Todo

figure out this enum

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 195.)

Todo

figure out this enum

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 203.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 209.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 215.)

Todo

check

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 225.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 353.)

Todo

figure out what happens on ITM, IFM, BLIT, TEX*BETA

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 362.)

Todo

NV3+

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 385.)

Todo

document that and BLIT

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 395.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 401.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 407.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 413.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 419.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 425.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 431.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 437.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 443.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 448.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 456.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/intro.rs line 464.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/nv1-tex.rst, line 16.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/nv1-tex.rst, line 22.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/nv1-tex.rst, line 28.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/nv1-tex.rst, line 34.)

Todo

precise upconversion formulas

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/pattern
line 351.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/sifm.rs
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/sifm.rs
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/sifm.rs
line 25.)

Todo

PM_TRIGGER?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 65.)

Todo

PATCH?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 67.)

Todo

add the patchcord methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 69.)

Todo

document common methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 71.)

Todo

document point methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 92.)

Todo

document line methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 125.)

Todo

document tri methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 153.)

Todo

document rect methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 176.)

Todo

document solid-related unified 2d object methods

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/2d/solid.rs
line 182.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/celsius/3d
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/celsius/3d
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/celsius/pg
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/celsius/pg
line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/celsius/pg
line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/celsius/pg
line 33.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/curie/3d.rs
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/curie/3d.rs
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/curie/pgra
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/curie/pgra
line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/curie/pgra
line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/curie/pgra
line 33.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/curie/pgra
line 39.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/3d.r
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/3d.r
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/com
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/com
line 15.)

Todo

convert

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/ctxc
line 5.)

Todo

rather incomplete.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 43.)

Todo

and vertex programs 2?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 59.)

Todo

figure out the exact differences between these & the pipeline configuration business

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 61.)

Todo

figure out and document the SRs

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 161.)

Todo

figure out the semi-special c16[]/c17[].

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 179.)

Todo

size granularity?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 193.)

Todo

other program types?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 195.)

Todo

describe the shader input spaces

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 205.)

Todo

describe me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 212.)

Todo

not true for GK104. Not complete either.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 231.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 237.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/cuda line 243.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/mac line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/mac line 19.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/pgra
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/pgra
line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/pgra
line 29.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/pgra
line 35.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/pgra
line 41.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/fermi/pgra
line 47.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst,
line 13.)

Todo

WAIT_FOR_IDLE and PM_TRIGGER

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 15.)

Todo

check Direct3D version

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 56.)

Todo

document NV1_NULL

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 74.)

Todo

figure out wtf is the deal with TEXTURE objects

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 175.)

Todo

find better name for these two

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 205.)

Todo

check NV3_D3D version

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 228.)

Todo

write something here

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 280.)

Todo

beta factor size

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 335.)

Todo

user clip state

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 337.)

Todo

NV1 framebuffer setup

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 339.)

Todo

NV3 surface setup

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 341.)

Todo

figure out the extra clip stuff, etc.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 343.)

Todo

update for NV4+

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 345.)

Todo

NV3+

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 377.)

Todo

more stuff?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 396.)

Todo

verify big endian on non-G80

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 425.)

Todo

figure out NV20 mysterious warning notifiers

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 434.)

Todo

describe GF100+ notifiers

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 436.)

Todo

0x20 - NV20 warning notifier?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 459.)

Todo

figure out if this method can be disabled for NV1 compat

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/intro.rst, line 567.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kelvin/3d.
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kelvin/3d.
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kelvin/pgr
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kelvin/pgr
line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kelvin/pgr
line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kelvin/pgr
line 33.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kepler/3d.
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kepler/3d.
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kepler/con
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/kepler/con
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/m2mf.rst,
line 11.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/m2mf.rst,
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/m2mf.rst,
line 27.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/m2mf.rst,
line 33.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/m2mf.rst,
line 39.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/maxwell/3
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/maxwell/3
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/maxwell/c
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/maxwell/c
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 21.)

Todo

more bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 139.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 215.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 219.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 223.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 227.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 231.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 235.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 239.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 243.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 247.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 251.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 255.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 259.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 263.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 267.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 271.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 275.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 279.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 283.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 287.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 291.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 293.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 299.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 305.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/nv1/pgrap
line 311.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/rankine/3d
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/rankine/3d
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/3d.rst
line 10.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/3d.rst
line 16.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pdma
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pdma
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pdma
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pdma
line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 29.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 35.)

Todo

figure out the bits, should be similiar to the NV1 options

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 52.)

Todo

check M2MF source

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 59.)

Todo

check

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 65.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 75.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 81.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/riva/pgrap
line 87.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/3d.rs
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/3d.rs
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/comp
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/comp
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/crop.
line 8.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/ctxct
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 32.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 46.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 52.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 66.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 80.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 93.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 105.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 118.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 133.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 139.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 151.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 164.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 184.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 190.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 49.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 65.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 86.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 104.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 120.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 139.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 167.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 172.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 177.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 194.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 199.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 208.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 213.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 218.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 25.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 31.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 44.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 58.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 31.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 43.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 56.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 70.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 84.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 25.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 73.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 143.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 227.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 267.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 297.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 345.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 389.)

Todo

check variants for preret/indirect bra

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 45.)

Todo

wtf is up with $a7?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 134.)

Todo

a bit more detail?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 166.)

Todo

perhaps we missed something?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 170.)

Todo

seems to always be 0x20. Is it really that boring, or does MP switch to a smaller/bigger stride sometimes?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 176.)

Todo

when no-one's looking, rename the a[], p[], v[] spaces to something sane.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 247.)

Todo

discard mask should be somewhere too?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 287.)

Todo

call limit counter

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 289.)

Todo

there's some weirdness in barriers.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 306.)

Todo

you sure of control instructions with non-0 w1b0-1?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 329.)

Todo

what about other bits? ignored or must be 0?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 448.)

Todo

figure out where and how $a7 can be used. Seems to be a decode error more often than not...

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 571.)

Todo

what address field is used in long control instructions?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 574.)

Todo

verify the 127 special treatment part and direct addressing

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 647.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 671.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 676.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 36.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 48.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 68.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 84.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 102.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 40.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 53.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 67.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 79.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 92.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 98.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 104.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 110.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 13.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 19.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 33.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 45.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 57.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 69.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/cuda/
line 82.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/pgra
line 20.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/pgra
line 28.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/pgra
line 36.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/pgra
line 42.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/pgra
line 48.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/pgra
line 54.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/prop.
line 8.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/vfetc
line 8.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tesla/zrop.
line 8.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/3d.rst,
line 10.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/3d.rst,
line 16.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 29.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 35.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 41.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 47.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 53.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/graph/tnt/pgraph
line 59.)

Todo

convert glossary

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/index.rst, line 26.)

Todo

finish file

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/intro.rst, line 345.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/g80-gpio.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/g80-gpio.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/g80-gpio.rst, line 23.)

Todo

figure out what else is stored in the EEPROM, if anything.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv1-peeprom.rst, line 31.)

Todo

figure out *how* the chip ID is stored in the EEPROM.

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv1-peeprom.rst, line 32.)

Todo

figure out wtf the chip ID is used for

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv1-peeprom.rst, line 33.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv10-gpio.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv10-gpio.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv10-gpio.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv10-gpio.rst, line 32.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv10-gpio.rst, line 40.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv10-gpio.rst, line 44.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/nv10-gpio.rst, line 48.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pmedia.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pmedia.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pmedia.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pmedia.rst, line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pnvio.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pnvio.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pnvio.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pnvio.rst, line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 35.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 43.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 47.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 53.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 61.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 63.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 71.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/prom.rst, line 75.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pstraps.rst, line 304.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pstraps.rst, line 310.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pstraps.rst, line 316.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/io/pstraps.rst, line 322.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-comp.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-comp.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-host-mem.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-host-mem.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-host-mem.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-host-mem.rst, line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-host-mem.rst, line 35.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-p2p.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-p2p.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-p2p.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-pfb.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-pfb.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-pfb.rst, line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-remap.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-remap.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-remap.rst, line 23.)

Todo

vdec stuff

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 28.)

Todo

GF100 ZCULL?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 29.)

Todo

check pitch, width, height min/max values. this may depend on binding point. check if 64 byte alignment still holds on GF100.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 90.)

Todo

check bounduaries on them all, check tiling on GF100.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 124.)

Todo

PCOPY surfaces with weird gob size

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 125.)

Todo

wtf is up with modes 4 and 5?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 551.)

Todo

nail down MS8_CS24 sample positions

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 552.)

Todo

figure out mode 6

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 553.)

Todo

figure out MS8_CS24 C component

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 554.)

Todo

check MS8/128bpp on GF100.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 560.)

Todo

wtf is color format 0x1d?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 638.)

Todo

htf do I determine if a surface format counts as 0x07 or 0x08?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 721.)

Todo

which component types are valid for a given bitfield size?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 809.)

Todo

clarify float encoding for weird sizes

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 810.)

Todo

verify I haven't screwed up the ordering here

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 841.)

Todo

figure out the MS8_CS24 formats

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 933.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 939.)

Todo

figure out more. Check how it works with 2d engine.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 957.)

Todo

verify somehow.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 981.)

Todo

reformat

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 1033.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 1136.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-surface.rst, line 1142.)

Todo

kill this list in favor of an actual explanation

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 45.)

Todo

PVP1

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 309.)

Todo

PME

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 310.)

Todo

Move to engine doc?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 311.)

Todo

verify GT215 transition for medium pages

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 518.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 618.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 624.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 630.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vm.rst, line 636.)

Todo

verify it's really the G84

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vram.rst, line 128.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vram.rst, line 187.)

Todo

tag stuff?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vram.rst, line 241.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vram.rst, line 247.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vram.rst, line 253.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/g80-vram.rst, line 259.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-comp.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-comp.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-host-mem.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-host-mem.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-host-mem.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-host-mem.rst, line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-host-mem.rst, line 36.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-p2p.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-p2p.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-p2p.rst, line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-vm.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-vm.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-vram.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/gf100-vram.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-pdma.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-pdma.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-pdma.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-pdma.rst, line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-pdma.rst, line 39.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-pdma.rst, line 45.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-surface.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-surface.rst, line 15.)

Todo

space?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-vram.rst, line 71.)

Todo

figure out what UNK1 nad UNK2 are for

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv1-vram.rst, line 102.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv10-pfb.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv10-pfb.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv10-pfb.rst, line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv3-dmaobj.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv3-dmaobj.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv3-pfb.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv3-pfb.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv3-pfb.rst, line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv3-vram.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv3-vram.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv4-dmaobj.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv4-dmaobj.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv4-vram.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv4-vram.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv4-vram.rst, line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv4-vram.rst, line 25.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv40-pfb.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv40-pfb.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv40-pfb.rst, line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv44-host-mem.rst, line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv44-host-mem.rst, line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv44-host-mem.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv44-pfb.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv44-pfb.rst, line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/nv44-pfb.rst, line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pbfb.rst line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pbfb.rst line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pbfb.rst line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pbfb.rst line 27.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pbfb.rst line 35.)

Todo

convert

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/peepho
line 58.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pffb.rst
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pffb.rst
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pffb.rst
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pffb.rst
line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pmfb.rs
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pmfb.rs
line 15.)

Todo

fill me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pmfb.rs
line 29.)

Todo

fill me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pmfb.rs
line 33.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pmfb.rs
line 41.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pxbar.rs
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pxbar.rs
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/memory/pxbar.rs
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst,
line 13.)

Todo

check UNK005000 variants [sorta present on NV35, NV34, C51, MCP73; present on NV5, NV11, NV17, NV1A, NV20; not present on NV44]

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 128.)

Todo

check PCOUNTER variants

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 129.)

Todo

some IGP don't have PVPE/PVP1 [C51: present, but without PME; MCP73: not present at all]

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 130.)

Todo

check PSTRAPS on IGPs

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 131.)

Todo

check PROM on IGPs

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 132.)

Todo

PMEDIA not on IGPs [MCP73 and C51: not present] and some other cards?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 133.)

Todo

PFB not on IGPs

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 134.)

Todo

merge PCRTC+PRMCIO/PRAMDAC+PRMDIO?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 135.)

Todo

UNK6E0000 variants

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 136.)

Todo

UNK006000 variants

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 137.)

Todo

UNK00E000 variants

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 138.)

Todo

102000 variants; present on MCP73, not C51

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 139.)

Todo

10f000:112000 range on GT215-

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 208.)

Todo

verified accurate for GK104, check on earlier cards

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 273.)

Todo

did they finally kill off PMEDIA?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 274.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 306.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 312.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 318.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 324.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 330.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 334.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 338.)

Todo

NV4x? NVCx?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 344.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 348.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 352.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 356.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 360.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 366.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 370.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 374.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 378.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 382.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 386.)

Todo

RE me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/mmio.rst, line 390.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/nv1-paudio.rst, line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/nv1-paudio.rst, line 21.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/nv1-paudio.rst, line 29.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/nv1-paudio.rst, line 35.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/nv1-paudio.rst, line 41.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/nv1-paudio.rst, line 47.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/nv1-paudio.rst, line 53.)

Todo

wtf is with that 0x21x ID?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pciid.rst, line 462.)

Todo

shouldn't `0x03b8` support x4 too?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pciid.rst, line 1955.)

Todo

convert

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/fermi.r
line 9.)

Todo

crossrefs

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 9.)

Todo

why? any others excluded? NV25, NV2A, NV30, NV36 pending a check

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 19.)

Todo

figure out what else happened on GF100

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 50.)

Todo

make it so

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 55.)

Todo

figure out interupt business

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 96.)

Todo

convert

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 111.)

Todo

wtf is CYCLES_ALT for?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 146.)

Todo

convert

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 154.)

Todo

C51 has no PCOUNTER, but has a7f4/a7f8 registers

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 191.)

Todo

MCP73 also has a7f4/a7f8 but also has normal PCOUNTER

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 193.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 201.)

Todo

PAUSED?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 328.)

Todo

unk bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 330.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 374.)

Todo

UNK8

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 451.)

Todo

check bits 16-20 on GF100

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 651.)

Todo

figure out how single event mode is supposed to be used on GF100+

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 670.)

Todo

wtf is CYCLES_ALT?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 694.)

Todo

figure out what's the deal with GF100 counters

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 762.)

Todo

figure out if there's anything new on GF100

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 823.)

Todo

unk bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 925.)

Todo

more bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 937.)

Todo

GF100

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 939.)

Todo

threshold on GF100

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 1054.)

Todo

check if still valid on GF100

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 1181.)

Todo

figure out record mode setup for GF100

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/intro.rs
line 1257.)

Todo

convert

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/nv10.rs
line 9.)

Todo

figure it out

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/nv40.rs
line 37.)

Todo

find some, I don't know, signals?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/nv40.rs
line 39.)

Todo

figure out roughly what stuff goes where

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/tesla.rs
line 44.)

Todo

find signals.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pcounter/tesla.rs
line 46.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 9.)

Todo

figure out IOCLK, ZPLL, DOM6

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 38.)

Todo

figure out 4010, 4018, 4088

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 39.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 44.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 52.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 56.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 60.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 68.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 76.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 84.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/g80-clock.rst, line 92.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 9.)

Todo

how many RPLLs are there exactly?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 33.)

Todo

figure out where host clock comes from

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 34.)

Todo

VM clock is a guess

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 35.)

Todo

memory clock uses two PLLs, actually

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 36.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 48.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 52.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 60.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 68.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gf100-clock.rst, line 76.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 9.)

Todo

figure out unk clocks

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 32.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 37.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 45.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 49.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 53.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 73.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 81.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/gt215-clock.rst, line 89.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv1-clock.rst, line 9.)

Todo

DLL on NV3

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv1-clock.rst, line 24.)

Todo

NV1???

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv1-clock.rst, line 25.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv1-clock.rst, line 29.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv1-clock.rst, line 37.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv1-clock.rst, line 45.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 9.)

Todo

figure out where host clock comes from

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 26.)

Todo

figure out 4008/shader clock

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 27.)

Todo

figure out 4050, 4058

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 28.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 33.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 41.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 45.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv40-clock.rst, line 53.)

Todo

figure out what divisors are available

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv43-therm.rst, line 57.)

Todo

figure out what divisors are available

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv43-therm.rst, line 95.)

Todo

Make sure this clock range is safe on all cards

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv43-therm.rst, line 111.)

Todo

There may be other switches.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv43-therm.rst, line 152.)

Todo

Document reg 15b8

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/nv43-therm.rst, line 215.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/chs line 8.)

Todo

check the frequency at which PDAEMON is polling

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/cou line 33.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/epv
line 9.)

Todo

and unknown stuff.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/ind
line 53.)

Todo

figure out additions

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/ind
line 65.)

Todo

this file deals mostly with GT215 version now

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/ind
line 67.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/io.
line 9.)

Todo

reset doc

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/io.
line 74.)

Todo

unknown v3+ regs at 0x430+

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/io.
line 75.)

Todo

5c0+

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/io.
line 76.)

Todo

660+

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/io.
line 77.)

Todo

finish the list

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/io.
line 78.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/per
line 7.)

Todo

discuss mismatched clock thing

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/per
line 9.)

Todo

figure out the first signal

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/per
line 34.)

Todo

document MMIO_* signals

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/per
line 35.)

Todo

document INPUT_*, OUTPUT_*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/per
line 36.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/sig
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/sig
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/sig
line 23.)

Todo

figure out bits 7, 8

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/sub
line 25.)

Todo

more bits in 10-12?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/sub
line 26.)

Todo

what could possibly use PDAEMON's busy status?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/pdaemon/use
line 17.)

Todo

check the possible dividers

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/ptherm.rst, line 80.)

Todo

verify the priorities of each threshold (if two thresholds are active

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/ptherm.rst, line 131.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/ptherm.rst, line 137.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/ptherm.rst, line 142.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/ptherm.rst, line 146.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/ptherm.rst, line 154.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/pm/ptherm.rst, line 162.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvcomp.rst line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvcomp.rst line 15.)

Todo

status bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvcomp.rst line 87.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvcomp.rst line 97.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvcomp.rst line 99.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvcomp.rst line 107.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 15.)

Todo

status bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 100.)

Todo

interrupts

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 101.)

Todo

MEMIF ports

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 102.)

Todo

core clock

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 103.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 113.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 117.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/pvenc.rst, line 119.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/index.r line 22.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/macro. line 96.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/macro. line 104.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/macro. line 112.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/macro. line 174.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pbsp.r line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pbsp.r
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pbsp.r
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pbsp.r
line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pciphe
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pciphe
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pciphe
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pciphe
line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pvp2.r
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pvp2.r
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pvp2.r
line 23.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/pvp2.r
line 31.)

Todo

width/height max may be 255?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst,
line 42.)

Todo

reg 0x00800

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst,
line 94.)

Todo

what macroblocks are stored, indexing, tagging, reset state

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst,
line 171.)

Todo

and availability status?

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst, line 187.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst, line 201.)

Todo

RE and write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst, line 225.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst, line 233.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst, line 243.)

Todo

more inferred crap

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst, line 451.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/vld.rst, line 496.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp2/xtensa.
line 5.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/index.r
line 21.)

Todo

Verify whether X or Y is in the lowest 16 bits. I assume X

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/mbring
line 118.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 15.)

Todo

interrupts

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 137.)

Todo

more MEMIF ports?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 138.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 148.)

Todo

unknowns

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 173.)

Todo

fix list

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 174.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 180.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/ppdec.
line 188.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 15.)

Todo

interrupts

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 123.)

Todo

more MEMIF ports?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 124.)

Todo

status bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 125.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 135.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 157.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 165.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 173.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 179.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 187.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 195.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 203.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 209.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 217.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 225.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 233.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 239.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 247.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 256.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 264.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 270.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 278.)

Todo

write

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pppp.r
line 286.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/psec.rs
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/psec.rs
line 15.)

Todo

clock divider in 1530?

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/psec.rs
line 103.)

Todo

find out something about the GM107 version

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/psec.rs
line 105.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/psec.rs
line 115.)

Todo

update for GM107

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/psec.rs
line 129.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.rst line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.rst line 15.)

Todo

interrupts

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.rst line 70.)

Todo

VM engine/client

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.rst line 71.)

Todo

MEMIF ports

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.rst line 72.)

Todo

status bits

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.rst line 73.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.rst line 83.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvdec.
line 92.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 15.)

Todo

MEMIF ports

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 130.)

Todo

unknowns

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 153.)

Todo

fix list

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 154.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 162.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 172.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 174.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 182.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vp3/pvld.rs
line 190.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/index.r
line 19.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 13.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 22.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 58.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 62.)

Todo

figure these out

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 68.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 74.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 80.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 87.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 94.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 101.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 108.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 117.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 124.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 130.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 136.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 142.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 148.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 154.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 160.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 166.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 172.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 178.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 184.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 190.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 196.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 202.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pme/in
line 210.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pmpeg
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pmpeg
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pmpeg
line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pmpeg line 31.)

Todo

list me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/a line 144.)

Todo

complete the list

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/a line 190.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/b line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/b line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/b line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/d line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/d
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/d
line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/fi
line 9.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/fi
line 15.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/fi
line 23.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/fi
line 31.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/i
line 42.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/i
line 50.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/i
line 56.)

Todo

incomplete for <G80

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/i
line 130.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/i
line 154.)

Todo

mov from $sr, $uc, $mi, $f, $d

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/i
line 224.)

Todo

some unused opcodes clear $c, some don't

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/s
line 238.)

Todo

figure out the pre-G80 register files

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvp1/s
line 351.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvpe.r
line 9.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvpe.r
line 15.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvpe.r
line 25.)

Todo

write me

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vpe/pvpe.r
line 33.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/intro.r
line 147.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/isa.rst,
line 979.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/isa.rst,
line 1130.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/isa.rst, line 1136.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/perf.rst line 11.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/vpring. line 11.)

Todo

write me

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/vreg.rs line 15.)

Todo

the following information may only be valid for H.264 mode for now

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/vreg.rs line 21.)

Todo

recheck this instruction on VP3 and other codecs

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/envytools/checkouts/latest/docs/hw/vdec/vuc/vreg.rs line 228.)

CHAPTER 6

Indices and tables

- genindex
- search

Symbols

C