r600_demo

HowTo Render a Freaking Triangle

Matthias Hopf

SUSE R&D / Novell

What is r600_demo?

- Bringup tool for r6xx and r7xx GPUs from AMD
  - Originally based on r300_demo

- Needed because:
  - *Big* architectural changes from r5xx to r6xx
  - No open source driver to build upon
  - No documentation available at AMD in ready to use form – created on-the-fly
  - Programming turned out more difficult than anticipated

⇒ Easy to understand, self-contained

The Crew

AMD		
	John Bridgman	Legal, IP
	Alexander Deucher	r600_demo, DRM, EXA, docs
	RichardZ Li	DRI driver, shader compiler
	Cooper Yuan	DRI driver
Novell		
	Egbert Eich	Memory setup, AtomBIOS, etc.
	Matthias Hopf	r600_demo, DRI init & driver, docs
	Luc Verhaegen	Command submission
RedHat		
	Dave Airlie	Initial DRM

- You!

History

- 2007/7 Start of the radeonhd project, first hardware docs
- 2008/1 r5xx programming docs

  r6xx CP docs
- 2008/2 r5xx register docs
- 2008/5 r6xx register & “programming” docs
- 2008/6 Start of r600_demo

  Working CP, DMA
- 2008/7 Working state output, register dump

  First triangle test (dysfunctional), programmed based on docs

  TCore (ATI bringup environment)

History (2)

- 2008/7 Alex adds CP dump from working TCore tests (4MB!) – still not working
- 2008/8/27 Alex: “At long last. Triangle.”
  A bit broken, though, RV770 only.
  Cleanup phase begins
- 2008/9/16 Alex: Triangle on r6xx
- 2008/9/25 Matthias: “I have a triangle!”
- 2008/11 Weird memory setup bug discovered.
  Finally DRM works on all tested cards.
- 2008/12/29 r600_demo released to public
  Approx. 82KB code left of 4MB

Documentation

- Already available:
  - r5xx Programming docs (CP is similar)
  - r6xx Instruction Set (Shader)
    Partially wrong, though (constants: r600_demo)
  - r6xx 3D Register docs
- To-be-released:
  - r[67]xx Programming docs
    (approx. 50 pages, includes CP docs)
  - r[67]xx Lessons learned (Wiki)

- http://developer.amd.com/documentation/guides/Pages/default.aspx#open_gpu
- http://www.x.org/docs/AMD/

r600_demo: Setup

- You need
  - git r600_demo
  - Recent enough radeonhd (1.2.4 or git)
  - Latest DRM (r6xx-r7xx-support branch)
- xorg.conf
  - Option "DRI"
  - Option "AccelMethod" "none"
    Alternatively (for the moment) "force-shadowfb"
- Run “r600_demo <–opts> <tests>” as root

r600_demo: Tests

- **r** Reset GPU – often enough this works
- **c** CPU based clear screen

- **t** Basic triangle test. Options –f, –i, –u, –S
- **T** Clipped transformed triangle test
- **q** Textured quad test
- **e** EXA solid test (blending)
- **E** EXA copy test

- **P** Performance test suite

- **w**, **W**, **b**, **x** Old & temporary tests

Performance Numbers

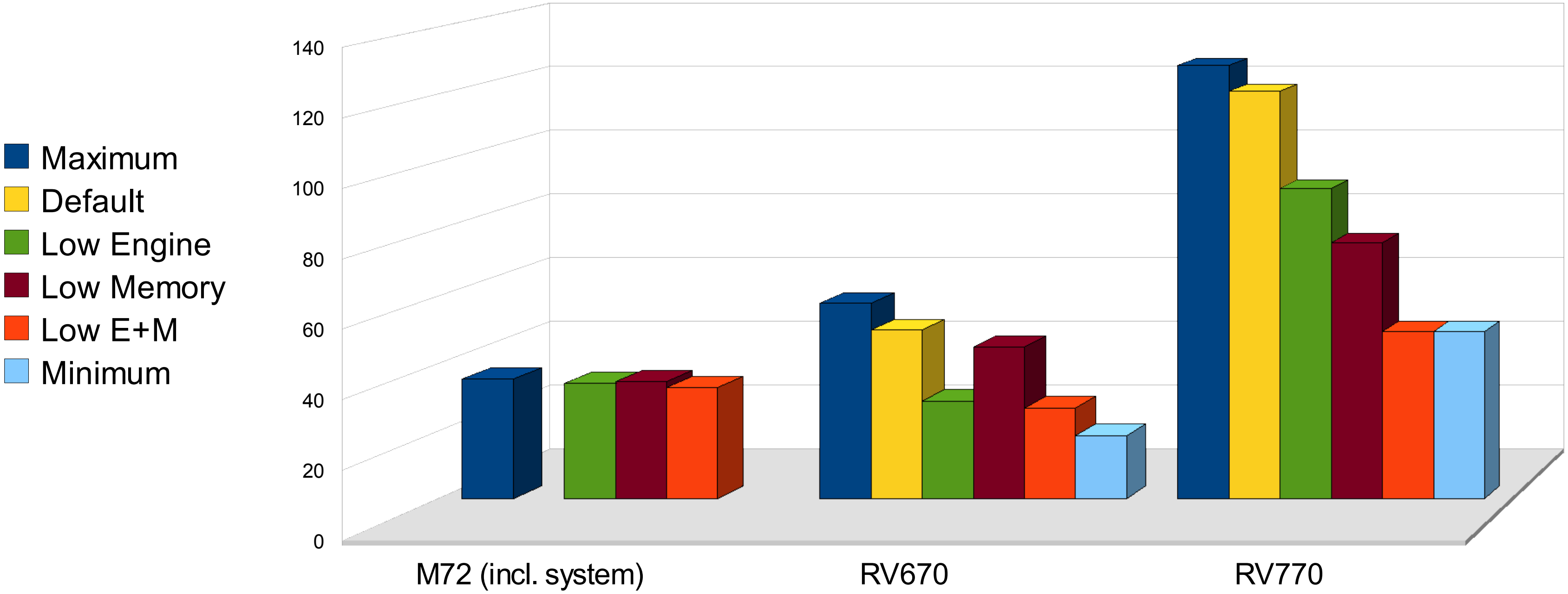
- All measured in GigaFLOPS
  (theoretical peak performance in parens)
- Biggest performance with long ALU clauses, 4 vector and 1 trans unit active, multiply-and-add

• M72	34.69	(35.2)
• RV610	41.46	(42)
• RV670	426.45	(427)
• RV770	1196.43	(1200)

- … unbelievably close to theoretical values…

Power Usage

- Some early experiments
- Only adjusting engine clock, memory clock, core voltage, not PCIe lanes, clock gating, switching off unused blocks, etc.



r[67]xx: Architecture

- CP
  - Ringbuffer
  - Microcode based
  - No real processor, architecture unknown
- Control flow
  - Loops, if-then-else, etc.
  - Works on 2x2 pixels, closely coupled
  - Many threads in flight
- ALU
  - 5-vectors
- Fetch units
  - Vertex fetch, texture fetch
- Blend, scissor, other fixed function units

r[67]xx: CP

- Fed by a ring buffer in system memory
  - Can parse indirect buffers (“Subroutines”)
  - Writes into register space: Type-0 packets
  - Some macro-like commands: Type-3 packets
    Partially adapted in chipset specific way
  - Can block on registers, emit interrupts and fences, write to system memory, etc.
- Very similar to r5xx system
  - Almost all Type-3 packets changed, though
- No control flow instructions
  - Not useful for validation

r[67]xx: Shaders

- Unified shader architecture
- Shaders are loaded dynamically from memory (*very* different to r5xx)
- Cache coherence!
- Easy to lock up GPU (end of program only specified by single bit, length of program not explicitly specified)
- 4 (5) types of shaders
- 3 clause levels in each shader
- Data transported either in explicit GPRs or GPRs defined by *semantic mappings*

r[67]xx: Shader Types

- ES – Export Shader
  - Only used with geometry shader (“Early Shader”)
  - Equivalent to vertex shader otherwise
- GS – Geometry Shader
  - Works on primitives, not only vertices
  - May submit more/less vertices (tessellation / kill)
- **VS – Vertex Shader**
  - Transforms vertices from world to clip coordinates
  - Isn't fed vertices, but has to fetch them (different to r5xx!)
  - Can invoke a *Fetch Shader* subroutine
- **PS – Pixel Shader**
  - Working on fragments

r[67]xx: Shader Layout

- Several levels
  - Control flow instructions
  - ALU clauses
  - Fetch clauses
- Control initiates ALU, Fetch, may run in parallel
- Control and ALU instruction words 64bit
- Fetch instruction words are 128bit,
  need 128bit alignment
- Runs 2x2 pixels a time, flags:
  - **valid** – pixel covers primitive, is not KILLed
  - **active** – pixel in correct branch of if-then-else / loop

r[67]xx: Control

- Commands
  - Invokes ALU and FETCH clauses
  - Loops, breaks, subroutines, jumps, conditions
  - Stack handling, change predication masks
  - Export to memory or ring buffers
- Remember
  - Always 2x2 pixel in flight
  - if-then-else: all clauses have to be executed, only change predication masks
  - Loops only exit if *all* pixels have inactive state
  - Flags for executing insts/clauses depending on valid and active state
    Necessary for computation of derivatives
- CALL_FS and RETURN_FS wrong in r600isa.pdf

r[67]xx: ALU

- 4-component “vector” unit, 1 scalar unit (Trans)
  - All independent from each other
  - Restrictions regarding source operands
  - *Reducing* ops like dot4 or max4
- Clause contains *groups* of 1-5 ALU ops and 2 optional 2-component constants
  - 2- and 3-operand instructions
    - Integer
      add/sub, mul, and/or/xor, cond. move, ...
    - Float
      add, mul, round, cond. kill, muladd, dot4, ...
    - Predication
    - Trans unit only
      shift, int↔float, 1/x, 1/sqrt(x), sin, exp, log, ...

r[67]xx: ALU (2)

- Sources:
  - GPR 0-127 – top n(4) clause local
  - Constants (inline, 256 registers / memory cache)
  - Constants FLOAT 0, 1, 0.5 or INT -1, 0, 1
  - Previous group registers PV, PS
  - Swizzle, modify (negate, absolute)
- Destination
  - GPR0-127
  - Swizzle, modify (x2/x4/x.5, clamp, mask)
  - Predication masks (clause and global)
- Up to 128 ops per clause (more on r7xx?)

r[67]xx: ALU Pitfalls

- Source Restrictions
  - Order of ops is always x,y,z,w,Trans
    Last op of group indicated by LAST bit
  - Max. 3 different GPR sources per group
  - Max. 4 different reg constant compnts per group
  - Max. 2 different constants for Trans
  - May use either reg constants or constant cache
- Write to registers is delayed → PV, PS
  Hidden by logic except for indirect GPR addressing
- Format of 2- and 3-operand ops different
- Format of 2-operand ops different between r6xx and r7xx

r[67]xx: ALU Pitfalls (2)

- Bank swizzling: (r600isa.pdf pp. 49ff)
  - Only one of each x,y,z,w GPR component can be loaded per cycle (3 cycles per instr, called 0-2).
  - Per scalar instruction bank_swizzle can select which cycle each operand comes from. e.g.:

SRC0	SRC1	SRC2	SWIZZLE	cycle0	cycle1	cycle2
1.x	2.x		012	1.x	2.x	-
3.x	1.y		201	1.y	-	3.x
2.x	1.y		102	(1.y)	(2.x)	-

  - Multiple ops can reference same data of a cycle
  - Special case: square() - i.e. 1.x*1.x ignores cycle 1
  - No restrictions for constants or PV/PS.
  - Trans shares cycles, but can load multiple components in a single cycle slot
- And more...

r[67]xx: Fetch Units

- Clause is either vertex or texture fetch clause
  - Wrong type works on systems w/o vertex cache...
- Up to 6 fetches per clause (more on r7xx?)

- 160 buffers per shader type VS, GS, PS (FS: 16)
  - Yes, vertex fetches can be done in PS...
- Buffers set up as either vertex or texture buffers
- 18 sampler units for interpolating textures
  - Can be reused → only one per format needed
  - Reusing in same clause? Probably not...

r[67]xx: Vertex Fetches

- Vertex buffers
  - Address (40bit)
  - Total size, number of entries
  - Defaults for vertex fetch clauses
- Vertex fetch op
  - Format (FMT_8_8_8_8, FMT_32_32_FLT, etc.)
  - Destination component swizzle (x,y,z,w,0,1,-)
  - Scaling (normalize, integer, scaled), sign
    Performs int→float conversion
  - Endianess
  - Offset
- First fetch: *Mega*fetch, indicating #bytes

r[67]xx: Texture Fetches

- Textures
  - Address, MIP address (?)
  - Dimensionality
  - Format, tiling mode
  - Width/height/depth, pitch
  - Swizzling
  - MIP levels
- Texture Samplers
  - Clamping
  - Minifying, magnifying filter (point, linear, cubic)
  - MIP filter
  - More to be analyzed (anisotropic, etc.)

r[67]xx: Texture Fetches (2)

- Texture Fetch op
  - Instruction type
  - Resource ID (0-160), Sampler ID (0-17)
  - Coordinate GPR
  - Destination GPR
  - Scaling (0-1 w/ repeat+mirroring / 0-size)
  - Coordinate swizzling (x,y,z,w,0,1)
  - Destination swizzling
  - Fixed point offsets, LoD bias
- Instructions partially unclear in docs
  - Not only texture fetch, but also derivative + LoD calculation, weights, etc.
  - TEX_INST_SAMPLE_C_G_LB ?!?

r[67]xx: Fetch Units Pitfalls

- Scaling doesn't seem to work as indicated
  - E.g. Floats work FMT_32_32_32_32_FLOAT
  - FMT_16_16 integers scaled in fetch unit work fine
  - FMT_32_32 integers don't...
  - - fetched as FMT_32_32_FLOAT normalized and converted in ALU.Trans unit work fine (signed only)
  - - fetched as FMT_32_32 work on all except RV610 and RV620...
  - - fetched as FMT_32_32 integers and converted in ALU.Trans unit show flat shaded triangles only on RV620, RV670, and RV770...
  - - same fetched as FMT_32_32_FLOAT works on RV620...
- WTF?!?

r[67]xx: Interpolator Setup

- Maps GPR output of VS to GPR input of PS
  - Input of VS is fixed: GPR0.x has index
- Defines GPR interpolation type
  - Flat, linear, perspective correct

r[67]xx: Vertex Submission

- No direct vertex submission mode like on r5xx
  - Only vertex buffers
  - No automatic fetch, explicitly code in VS or FS
  - Draw initiation with Type-3 packet
- Indices
  - 16 and 32 bit indices
  - Automatic numbering, included in Type-3 packet, or from additional buffer
- Pitfalls
  - IT_DRAW_* Type-3 needs correctly associated VGT_DRAW_INITIATOR, e.g. IT_DRAW_INDEX_IMMD / DI_SRC_SEL_IMMEDIATE

r[67]xx: Fix Function Parts

- Frame + Depth buffers
  - Multiple render targets
  - Need additional buffers for hierarchical Z
  - High performance improvement with tiling
    - Much more complex than r5xx
    - Possible to map memory regions to CPU with de-tiling
- Clipping
  - Tons of possibilities
  - OpenGL clipping planes
  - Scissors: generic, screen, window, viewport, 4 clip
  - Viewport transformation
- Blending, Multisampling etc.
- Theoretically: Care about CPU cache

r[67]xx: Cache Coherence

- Complex caching mechanisms with multiple source and destination caches, no snooping!
- Cache invalidation
  - Address ranges, cache types
  - Wait for finished draws, flushing, fence notifications
- Source cache flushes
  - On vertex buffer + texture uploads, shader changes
- Need to wait for finished draws + flushed cache
  - On hardware→software transition
  - On binding textures after render to texture
  - On glFinish()
- Support for interrupt based pipelined fences and cache flush notifications

r600_demo: Basic Layout

- r600_demo.c
  - Opens DRM, binds buffers, parses options, calls tests
  - Dead ugly...
- r600_emit.h
  - Packet emission, uses api defined in r600_hwapi.h
  - Abstraction for later use in DRI driver
- r600_lib.c, r600_lib.h
  - Support functions, buffer submission, etc.
- r600_init.c, r600_state.h
  - Initialization, subsystem setup, draw initiation
- r600_reg.h
  - Register definitions, most autogenerated from docs
- r600_shader.h
  - Macros for shader definitions

r600_demo: Ringbuffer Handling

- Uses mechanism reserved for X11 driver
  - Doesn't work well with X11 acceleration
  - Alternative: Direct ring programming
    Only on 32bit due to DRM bug
- Irrelevant for tests
  - Tests just define when to actually submit buffers
    flush_cmds()

r600_demo: Command Emission

- Reserve ring space, define variables etc.
  CMD_BUFFER_PREAMBLE (dwords)
  CMD_BUFFER_ALLOC (dwords)
- Basic macro for emitting a 32bit value to ring
  E32 (dword)
- Emit float value
  EFLOAT (float)
- Initiate Type-0 / Type-3 packet
  EPACK0 (reg, num) / EPACK3 (cmd, num)
- Write single register (Type-0 packet)
  EREG (reg, dword) / EREGFLOAT (reg, float)
- Wait for engine finished
  EMIT_WAIT_3D_IDLE_CLEAN ()

r600_demo: Chip Initialization

- set_default_state ()
  - Contains quite some stuff to be moved to DRM
  - Contains some magic values for magic registers
  - Does extra cleanup if CLEAN_SETUP defined
  - Tons of subsystem initialization

r600_demo: The Triangle

- What you absolutely need
  - Vertex buffer: set_vtx_resource ()
  - Vertex shader: vs_setup ()
  - Pixel shader: ps_setup ()
  - Initialization: start_3d (), set_default_state ()
  - Render target: set_render_target ()
  - VB, VS, PS uploaded to GPU or GART memory, cache flushing: upload ()
  - Viewport setup *or* VTX_XY_FMT_bit
  - set polygon mode, enable RT0
  - Interpolator setup
  - Draw: draw_auto ()
- Easiest starting point: r600_texture.c

The Future

- DRI driver development
  - Currently based on DRI, not DRI2 / gallium / etc.
    - Never good to make two radical things at the same time
  - End of last year: Basis for driver (Software fallbacks only)
  - Currently: hello.c from Red Book works with fixed shaders + buffers
  - AMD adds shader compiler
    - That means some IP issues have to be solved
    - Needs a bit of time

The End

Questions ?