ATI Radeon Driver for Plan 9

Implementing R600 Support

Sean Stangl (sstangl@andrew.cmu.edu)

15-412, Carnegie Mellon University

October 23, 2009

Radeon Numbering System

- ATI has released many cards under the Radeon label.
- Modern cards are prefixed with "HD".

Rough Mapping from Chipset to Marketing Name

R200 : Radeon {8500, 9000, 9200, 9250}

R300 : Radeon {9500, 9600, 9700, 9800}

R420 : Radeon {X700, X740, X800, X850}

R520 : Radeon {X1300, X1600, X1800, X1900}

R600 : Radeon HD {2400, 3600, 3800, 3870 X2}

R700 : Radeon HD {4300, 4600, 4800, 4800 X2}

Notable Chipset Deltas

Each family roughly coincides with a DirectX version bump.

- R200, R300, and R420 use a similar architecture.
  - Each family changes the pixel shader.

Notable Chipset Deltas

Each family roughly coincides with a DirectX version bump.

- R200, R300, and R420 use a similar architecture.
  - Each family changes the pixel shader.
- R520 introduces yet another new shader model.
  - ATI develops ATOMBIOS, a collection of data tables and scripts stored in the ROM on each card.

Notable Chipset Deltas

Each family roughly coincides with a DirectX version bump.

- R200, R300, and R420 use a similar architecture.
  - Each family changes the pixel shader.
- R520 introduces yet another new shader model.
  - ATI develops ATOMBIOS, a collection of data tables and scripts stored in the ROM on each card.
- R600 supports the Unified Shader Model.
  - The 2D engine gets unified into the 3D engine.
  - The register specification is completely different.

Notable Chipset Deltas

Each family roughly coincides with a DirectX version bump.

- R200, R300, and R420 use a similar architecture.
  - Each family changes the pixel shader.
- R520 introduces yet another new shader model.
  - ATI develops ATOMBIOS, a collection of data tables and scripts stored in the ROM on each card.
- R600 supports the Unified Shader Model.
  - The 2D engine gets unified into the 3D engine.
  - The register specification is completely different.
- R700 is an optimized R600.

Current Driver Status / Goals

The current Plan 9 Radeon driver supports R200 and R300.

- Exists only as source; has not been merged.
- Currently being maintained by vsrinivas.
- R420 support may exist.

Current Driver Status / Goals

The current Plan 9 Radeon driver supports R200 and R300.

- Exists only as source; has not been merged.
- Currently being maintained by vsrinivas.
- R420 support may exist.

The project goals are to modify this driver to:

- Support multiple chipsets in a clean manner in one driver.
- Include an ATOMBIOS parser.
- Drive the R600 family ($\approx$ Radeon HD 3850).
- Provide 2D acceleration via the 3D engine.

Current Driver Status / Goals

The current Plan 9 Radeon driver supports R200 and R300.

- Exists only as source; has not been merged.
- Currently being maintained by vsrinivas.
- R420 support may exist.

The project goals are to modify this driver to:

- Support multiple chipsets in a clean manner in one driver.
- Include an ATOMBIOS parser.
- Drive the R600 family ($\approx$ Radeon HD 3850).
- Provide 2D acceleration via the 3D engine.

The rest of this talk expands on these ideas in greater detail.

Available Documentation

ATI released register and 3D engine specs for R5xx and R6xx. (developer.amd.com/documentation/guides/Pages/default.aspx)

- They are helpful, although lacking.

Available Documentation

ATI released register and 3D engine specs for R5xx and R6xx. (developer.amd.com/documentation/guides/Pages/default.aspx)

- They are helpful, although lacking.

The Xf86 Radeon and RadeonHD drivers are used as reference.

- Both support R600, but with varying features.
- Both projects share code that we need (register files, etc.).
- Both are humongous and difficult to read.

Available Documentation

ATI released register and 3D engine specs for R5xx and R6xx. (developer.amd.com/documentation/guides/Pages/default.aspx)

- They are helpful, although lacking.

The Xf86 Radeon and RadeonHD drivers are used as reference.

- Both support R600, but with varying features.
- Both projects share code that we need (register files, etc.).
- Both are humongous and difficult to read.

In order to facilitate supporting new cards, the Plan 9 driver must be updated to make use of two features present since R520.

- ATOMBIOS (discovered from RadeonHD)
- Command Processor (discovered from ATI documentation)

What is ATOMBIOS?

ATOMBIOS is:

- A collection of card-specific data tables and scripts stored in ROM on Radeon cards since R520.
- Accessible via a common interface regardless of card family or model.
- A $\approx 10,000$ line parser provided in part by ATI, in part by reverse-engineering from the RadeonHD team.

What is ATOMBIOS?

ATOMBIOS is:

- A collection of card-specific data tables and scripts stored in ROM on Radeon cards since R520.
- Accessible via a common interface regardless of card family or model.
- A $\approx 10,000$ line parser provided in part by ATI, in part by reverse-engineering from the RadeonHD team.

This is a very nice substitute for register twiddling, at the price of dragging along (and porting) an enormous codebase.

- Also, Plan 9 can't use most of the provided features.
- But without ATOMBIOS, we have to discover and hardcode undocumented defines for BIOS data, for each card.

Sample ATOMBIOS Commands

Example ATOMBIOS Exercising Code from Radeon

RHDAtomBiosFunc(pScrn->scrnIndex, NULL, ATOMBIOS_INIT, &atomBiosArg)

A sampling of available commands:

- ATOMBIOS_ALLOCATE_FB_SCRATCH
- GET_DEFAULT_ENGINE_CLOCK
- ATOM_SET_VOLTAGE
- ..hundreds more. (Radeon has a smaller list than RadeonHD.)

What other abstraction layers are there?

There are three supported options for talking to the Graphics Controller in the R5xx documentation:

What other abstraction layers are there?

There are three supported options for talking to the Graphics Controller in the R5xx documentation:

1. Conduct a sequence of register writes to setup a processing engine on the graphics controller, and then start it by toggling the trigger register.

What other abstraction layers are there?

There are three supported options for talking to the Graphics Controller in the R5xx documentation:

1. Conduct a sequence of register writes to setup a processing engine on the graphics controller, and then start it by toggling the trigger register.
2. Push command packets to the graphics controller, and have the hardware translate the packets into register writes.

What other abstraction layers are there?

There are three supported options for talking to the Graphics Controller in the R5xx documentation:

1. Conduct a sequence of register writes to setup a processing engine on the graphics controller, and then start it by toggling the trigger register.
2. Push command packets to the graphics controller, and have the hardware translate the packets into register writes.
3. Construct a ring buffer shared between host and GPU, and have the graphics controller pull from it asynchronously.

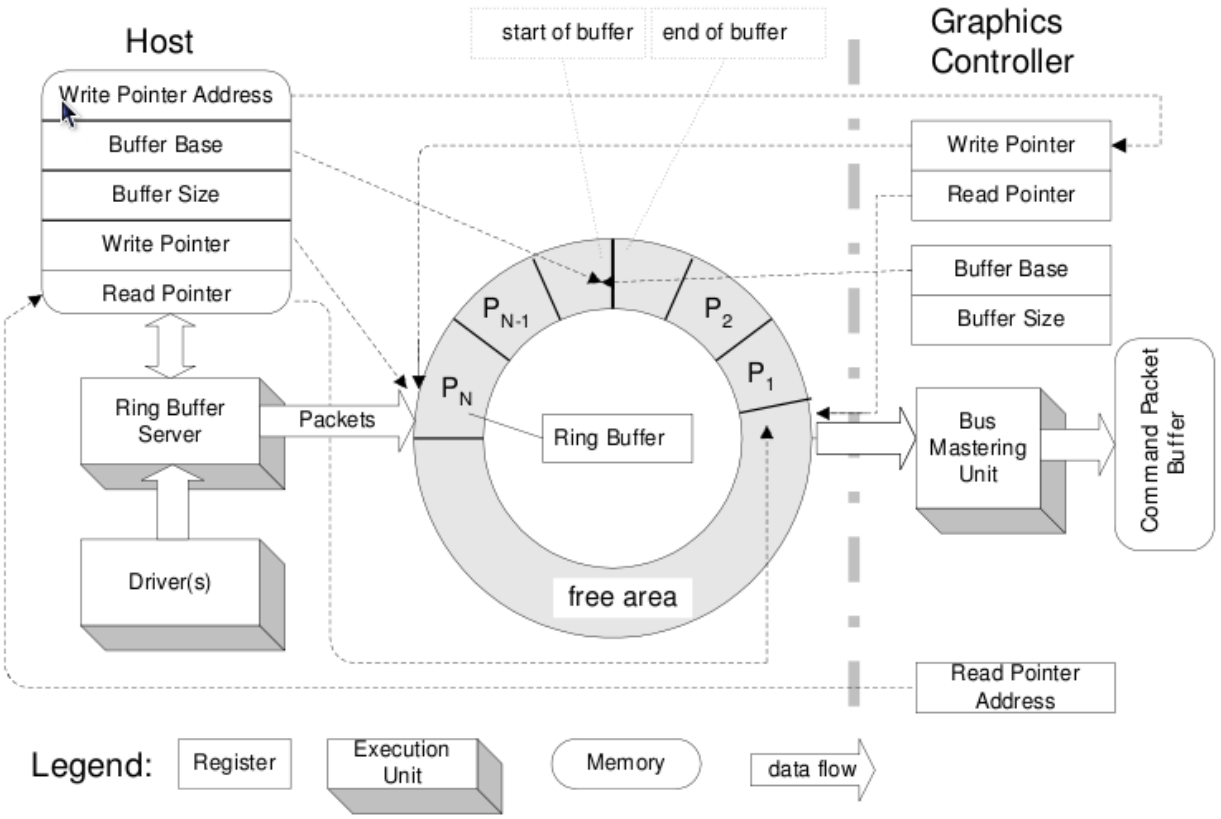
What other abstraction layers are there?

There are three supported options for talking to the Graphics Controller in the R5xx documentation:

1. Conduct a sequence of register writes to setup a processing engine on the graphics controller, and then start it by toggling the trigger register.
2. Push command packets to the graphics controller, and have the hardware translate the packets into register writes.
3. Construct a ring buffer shared between host and GPU, and have the graphics controller pull from it asynchronously.

The R6xx documentation only mentions the ring buffer method. The current driver uses the push method in a FIFO.

Ring Buffer Implementation



http://developer.amd.com/gpu_assets/RRG-216M56-03oOEM.pdf

What about the rest of needed functionality?

Well, there's no documentation on that.

- A great component of this project is copying logic from the Radeon driver.
- Another great component of this project is hope.

Structure of Plan 9 Graphics Drivers

Each Plan 9 graphics driver is split into two components.

Structure of Plan 9 Graphics Drivers

Each Plan 9 graphics driver is split into two components.

1. kernel component (vgaradeon.c)
   - Provides hardware acceleration and blanking.
   - Drops the card into linear mode.
     - The VGA driver handles the actual memory mapping.

Structure of Plan 9 Graphics Drivers

Each Plan 9 graphics driver is split into two components.

1. kernel component (vgaradeon.c)
   - Provides hardware acceleration and blanking.
   - Drops the card into linear mode.
     - The VGA driver handles the actual memory mapping.
2. userspace aux/vga component (radeon.c)
   - Reads in BIOS data.
   - Reads in VGA registers.
   - Configures and sets VGA registers appropriately.

Structure of Plan 9 Graphics Drivers

Each Plan 9 graphics driver is split into two components.

1. kernel component (vgaradeon.c)
   - Provides hardware acceleration and blanking.
   - Drops the card into linear mode.
     - The VGA driver handles the actual memory mapping.
2. userspace aux/vga component (radeon.c)
   - Reads in BIOS data.
   - Reads in VGA registers.
   - Configures and sets VGA registers appropriately.

Logically, aux/vga executes first.

aux/vga

aux/vga represents each video card as a Ctlr. Each Ctlr is an interface providing the following functions:

aux/vga

aux/vga represents each video card as a Ctlr. Each Ctlr is an interface providing the following functions:

1. options()
   - Set values before the init() functions of other Ctlrs are called.

aux/vga

aux/vga represents each video card as a Ctlr. Each Ctlr is an interface providing the following functions:

1. options()
   - Set values before the init() functions of other Ctlrs are called.
2. init()
   - Edit the in-memory copy of the registers to implement the specified mode.

aux/vga

aux/vga represents each video card as a Ctlr. Each Ctlr is an interface providing the following functions:

1. options()
   - Set values before the init() functions of other Ctlrs are called.
2. init()
   - Edit the in-memory copy of the registers to implement the specified mode.
3. snarf()
   - Read the Ctrl's registers into memory.

aux/vga

aux/vga represents each video card as a Ctlr. Each Ctlr is an interface providing the following functions:

1. options()
   - Set values before the init() functions of other Ctlrs are called.
2. init()
   - Edit the in-memory copy of the registers to implement the specified mode.
3. snarf()
   - Read the Ctrl's registers into memory.
4. load()
   - Write the Ctrl's modified registers out to the card.

kernel driver

Most interesting logic is handled by the kernel VGA driver.
The kernel driver for the Radeon is responsible for:

kernel driver

Most interesting logic is handled by the kernel VGA driver.
The kernel driver for the Radeon is responsible for:

- Setting the dedicated mmio regions.

kernel driver

Most interesting logic is handled by the kernel VGA driver.
The kernel driver for the Radeon is responsible for:

- Setting the dedicated mmio regions.
- Calling the appropriate functions to drop the card and vga driver into linear mode (unless you want segmented mode).

kernel driver

Most interesting logic is handled by the kernel VGA driver.
The kernel driver for the Radeon is responsible for:

- Setting the dedicated mmio regions.
- Calling the appropriate functions to drop the card and vga driver into linear mode (unless you want segmented mode).
- Providing hardware acceleration functions.
  - In our case, that means using the 3D engine.

Figures and Estimates

The R200/R300 driver is approximately $1,200$ LOC.

Figures and Estimates

The R200/R300 driver is approximately $1,200$ LOC.

- Adding generalized code to enable R200 and R600 in one driver, $+100$.

Figures and Estimates

The R200/R300 driver is approximately $1,200$ LOC.

- Adding generalized code to enable R200 and R600 in one driver, $+100$.
- Adding ATOMBIOS to aux/vga: $+10,000$.
  - I will attempt to trim it.
  - The ATI-provided code is actually reasonable.

Figures and Estimates

The R200/R300 driver is approximately $1,200$ LOC.

- Adding generalized code to enable R200 and R600 in one driver, $+100$.
- Adding ATOMBIOS to aux/vga: $+10,000$.
  - I will attempt to trim it.
  - The ATI-provided code is actually reasonable.
- Adding the pull-based ring buffer: $+300$.
  - May not be necessary?

Figures and Estimates

The R200/R300 driver is approximately $1,200$ LOC.

- Adding generalized code to enable R200 and R600 in one driver, $+100$.
- Adding ATOMBIOS to aux/vga: $+10,000$.
  - I will attempt to trim it.
  - The ATI-provided code is actually reasonable.
- Adding the pull-based ring buffer: $+300$.
  - May not be necessary?
- 3D Engine code: $+2000$ (header files).

Figures and Estimates

The R200/R300 driver is approximately $1,200$ LOC.

- Adding generalized code to enable R200 and R600 in one driver, $+100$.
- Adding ATOMBIOS to aux/vga: $+10,000$.
  - I will attempt to trim it.
  - The ATI-provided code is actually reasonable.
- Adding the pull-based ring buffer: $+300$.
  - May not be necessary?
- 3D Engine code: $+2000$ (header files).

Only about $1,500$ lines of actual logic, hopefully.

Order of Operations

The project is being completed in roughly the following order:

Order of Operations

The project is being completed in roughly the following order:

1. Read BIOS data from card (aux/vga).

Order of Operations

The project is being completed in roughly the following order:

1. Read BIOS data from card (aux/vga).
2. Implement snarf(), then init(), then load().

Order of Operations

The project is being completed in roughly the following order:

1. Read BIOS data from card (aux/vga).
2. Implement snarf(), then init(), then load().
   - Test that VGA modesetting works, even if we see garbage.

Order of Operations

The project is being completed in roughly the following order:

1. Read BIOS data from card (aux/vga).
2. Implement snarf(), then init(), then load().
   - Test that VGA modesetting works, even if we see garbage.
3. Figure out what kernel bits need to be changed, if any, to get into linear mode.

Order of Operations

The project is being completed in roughly the following order:

1. Read BIOS data from card (aux/vga).
2. Implement snarf(), then init(), then load().
   - Test that VGA modesetting works, even if we see garbage.
3. Figure out what kernel bits need to be changed, if any, to get into linear mode.
4. Work on enabling the 3D Engine.

questions?