

# A journey with radeon in linux graphics system

Jérôme Glisse

17/04/2008

# Outline

- 1 Why ?
- 2 ATOM bios sweet devil
- 3 From modular hardware to modular code
- 4 Lockup forensic analysis
- 5 Command stream checking
- 6 Fence & context
- 7 Fragment program
- 8 Tearing
- 9 Status

# Why kernel modesetting ?

## Expected outcome

- Kernel in charge of hw like for others drivers.
- Kernel needs to know hw states
- Suspend and resume
- Flickers free boot
- Reducing root code in userspace
- Graphical boot
- Blue screen of death (with customizable color which obviously would be a new feature)

## Why jumping on the wagon

- Now we can post card (at least on hw we care for Intel, AMD, NV)
- Kernel radeon driver have to keep **backward compatibility**.
- **Kernel radeon driver design does not fit well with the new infrastructure**
- Initialization path have to deal with various assumptions of userspace on how things are setup (memory layout), driver needs psychic power

So a **clean start sounds like a good plan**. No more assumptions, time to play & test the API.

# What i did wrong ?

## Evidence

- I rewrite the radeon modesetting (up to dac) which is **bad**
- **Modesetting is tedious**
- **Radeon ddx have a well tested code for modesetting**

## Pleading guilty

- **So i should have use ddx code since begin.**
- This is one of the big item on radeon kernel modesetting todo list.
- **Needed so others people can play with the code !**

# ATOM bios sweet devil

Should we use atom bios or not for modesetting ?

## For ATOM

- Should work as **used by windows** !
- **AMD engineer will go through the painfull modesetting stuff for us.**
- **Abstract low level hw knowledge**

## Against ATOM

- **Not bug free and working around bug might be painfull**
- **Windows doesn't necessarily use it the way we would**
- **It is stupid** doing multiple times some stuff for now reasons)
- Others ?

# User point of view

## What user want ?

- Working 3D
- Accelerated video
- ...

## What user ignore ?

- User largely ignore modesetting
- Having basic functionalities should be enough for 90% of users
- ...

# Trade off

- So use ATOM bios to leverage work done by AMD
- Speeds up support for new hw
- Reduce time spent on modesetting
- We can come back to modesetting power users once we have more features
- Let's have fun with 3d, video, ...

# AMD lego design

- AMD hw engineers are kids playing with lego
- AMD GPU are a collection of modules
- TMDS module, DAC module, 3D engine module, CP module ...
- So **different GPU can share same modules**, nice...
- But **module's registers address space can move** btw GPU
- **Painfull for driver writer**, likely easier for hw engineer

# Radeon kernel modular design

## Code kernel modules

Each modules should not do any assumptions about other parts of the hw

ie each module should work whatever are the other modules

- Modesetting (granularities down to: tmds, dac, crtc ...)
- Memory management & setup
- Bus & GART (AGP, PCIE, ...)
- CP (command processor)
- Command stream checking
- ...

## R300 & R400 HW background

- Pre-R5xx, requests made within the aperture range 0x1400 - 0x1EFF and 0x2000 - 0xFFFF **were queued**. From R5xx, onwards these requests will not be queued.
- It means that **MMIO write to these register goes through a FIFO**
- RBBM\_SOFTRESET is a register to **reset part of the engine** (it's not in the queued area)
- GA\_SOFT\_RESET is to **reset the 3D pipeline** it's in the queued area :(
- RBBM\_STATUS give you status informations about different part of the GPU
- WAIT\_UNTIL **stop fifo processing** until wait condition are met

# "Do bad things and bad things happen"

GPU have **unpredictable** (read lockup) behavior **when programmed in some unexpected way**

## Recipe

- Program VAP to get 1 vertex & 1 color
- Send VAP vertex data but no color (this things happen)
- Tada the VAP hang ! Lucky you !
- But as you were expecting the best (and not assuming the worst) you inserted a `WAIT_UNTIL` after the command stream triggering the VAP hang.
- Luck is standing on your side today ! `WAIT_UNTIL` stop the fifo until the GPU goes idle.

# Luck is all you need

- As you clever, you figure out that the beast hang somewhere **you do a softreset**, program RBBM\_SOFTRESET first to reset VAP (but as you lucky this in fact not enough to reset VAP)
- So **program GA\_SOFT\_RESET to actually trigger the full GA reset**, and here the fun start.
- GA\_SOFT\_RESET are queued in the fifo
- WAIT\_UNTIL has stoped fifo processing
- You fill the cmd fifo with GA\_SOFT\_RESET
- You bring down PCI bus once hw fifo full (ie quickly)

# Outcome

- You want to carefully check the command stream
- You want to avoid WAIT\_UNTIL

# Rules for command stream checking

## Never trust userspace

- FGLRX likely trust userspace more than we do
- We should not do assumption about the code in userspace which is sending command stream to the kernel
- For each packet check if authorized and if any special operations have to be perform (flush, relocation, ...)
- Does not affect speed that much so we should do it for security and robustness

# Command stream checking, today design

```
#define DRM_AMD_CMD_BO_TYPE_INVALID 0
#define DRM_AMD_CMD_BO_TYPE_CMD_RING (1 << 0)
#define DRM_AMD_CMD_BO_TYPE_CMD_INDIRECT (1 << 1)
#define DRM_AMD_CMD_BO_TYPE_DATA (1 << 2)
```

```
struct drm_amd_cmd_bo_offset
```

```
{
    uint64_t      next;
    uint64_t      offset;
    uint32_t      cs_id;
};
```

```
struct drm_amd_cmd_bo
```

```
{
    uint32_t      type;
    uint64_t      next;
    uint64_t      offset;
    struct drm_bo_op_req op_req;
    struct drm_bo_arg_rep op_rep;
};
```

```
struct drm_amd_cmd
```

```
{
    uint32_t      cdw_count;
    uint32_t      bo_count;
    uint64_t      bo;
    struct drm_fence_arg fence_arg;
};
```

- Parse: PACKET0(COLORBUFFER, 1)
- Check that this BO is a DATA BO and that size is big enough against the actual command stream expectation
- End result PACKET0(COLORBUFFER, BO[1].offset)

10

20

# Fence madness

## Wishes

- You **do not want billions of interrupts per second**
- You want to **avoid stalling the command processing**
- You **do not want to flush at each fence**
- 

## Avoid interrupts

Make sure BO busy query driver callback to check if any fence passed

## Avoid flush

**Batch flush together**, delay them to idle time.

## Timeout

- **Fence timeout** can be very helpfull to detect either GPU lockup or to have some kind of fairness in GPU use ie any program overloading the GPU being preempted if its fence timeout

# Stalling GPU is bad

## GPU stall

- Stall happen for many unpipelined register, ie register write is **delayed until it can done safely**
- These registers mainly deal with buffer offset (3d color buffer offset, z buffer offset, ...)
- **Command submission assume that you loose context** so you need to reprogram this register with each command submission, but you **don't want to do that**

# Context or state caching

## How to avoid stalling the GPU

I am thinking to either of this two solutions :

- **State caching in ring submission**, inside the kernel catch register write and remove them if not needed. Drawbacks is that it might be costly.
- **Context** solution, userspace create context which fully describe all unpipelined register and bind them along superioctl, kernel can then just compare context id to know if it needs to restore register.

Comments ? Idea ?

# Fragment program optimization

- **Fragment program are essential**, you got to pay attention to them
- If you not **optimize & be clever** about them you might refuse some which you should be able to run.
- Hopefully llvm should work, hopefully

## Texture indirection

(24) What is a texture indirection, and how is it counted?

RESOLVED: On some implementations, fragment programs that have complex texture dependency chains may not be supported, even if the instruction counts fit within the exported limits. A texture dependency occurs when a texture instruction depends on the result of a previous instruction (ALU or texture) for use as its texture coordinate.

# Texture indirection: reshuffle

## Supplied program

```
TEMP a, b, c, d;
// node 0 - 1 indirection
TEX a, fragment.color, texture[0], 2D;
// node 1 - 2 indirections
TEX b, a, texture[1], 2D;
ADD c, b, 1;
MUL b, c, b;
// node 2 - 3 indirections because c have been written in previous node
TEX c, fragment.color, texture[2], 2D;
// node 3 - 4 indirections
TEX d, c, texture[3], 2D;
ADD a, b, d;
// node 4 - 5 indirections out of limit
TEX result.color, a, texture[4], 2D;
```

## Reshuffled program

```
TEMP a, b, c, d;
// node 0 - 1 indirection
TEX a, fragment.color, texture[0], 2D;
TEX c, fragment.color, texture[2], 2D;
// node 1 - 2 indirections
TEX b, a, texture[1], 2D;
TEX d, c, texture[3], 2D;
ADD c, b, 1;
MUL b, c, b;
ADD a, b, d;
// node 2 - 3 indirections
TEX result.color, a, texture[4], 2D;
```

# Swizzling

## Native swizzle

r300/r400 hw does not support all swizzling. We want to reshuffle to use native swizzle whenever possible. Following program can be optimize (assuming xyzw & wzyx & wyxz are native but xzwy isn't):

## Supplied program

```
TEMP a;  
PARAM coef = {0.5, 0.6, 0.7, 0.8};  
ADD a, fragment.color.xzwy, coef.wyxz;
```

## Reshuffled program

```
TEMP a;  
PARAM coef = {0.5, 0.6, 0.7, 0.8};  
ADD a, fragment.color.wzyx, coef.xyzw;
```

# Tearing

- I don't know a good solution, using **WAIT\_UNTIL is dangerous** and also **kill performance** as it stops the GPU to do any more processing
- Likely going to use a **tasklet**, might need to forbid 2d command in CP and use mmio to do that.
- Idea ? Comments ?

# Status

## Today

- Infrastructure is there, just not many users
- Basic EXA working
- 3D working ie i can draw stuff but doesn't have a proper driver

## Tomorrow

Once i fix a fence or bo bug which sneaked into the schedule

- DRI2 (EXA was needed)
- Radeon winsys with DRI2 and gallium software to test DRI2
- Gears, well a lie, just send enough state so that gears work (very easy)
- Fragprog compiler