REclock

RE'ing NVA3/5/8 Voltage+Frequency Scaling

Roy Spliet

Clocks

Boot clock $<$ maximum clock.

dmesg:

```
1  nouveau  [     CLK][0000:01:00.0] 03: core 135 MHz shader 270 MHz memory 135 MHz
2  nouveau  [     CLK][0000:01:00.0] 07: core 405 MHz shader 810 MHz memory 405 MHz
3  nouveau  [     CLK][0000:01:00.0] 0f: core 606 MHz shader 1468 MHz memory 790 MHz
4  nouveau  [     CLK][0000:01:00.0] --: core 405 MHz shader 810 MHz memory 405 MHz
```

Table of Contents

Challenge

Process

Results

Future work

Table of Contents

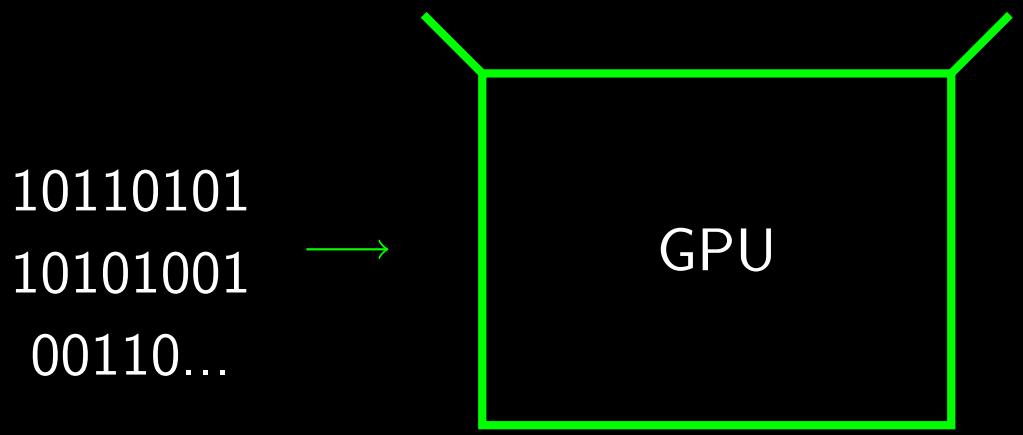
Challenge

Process

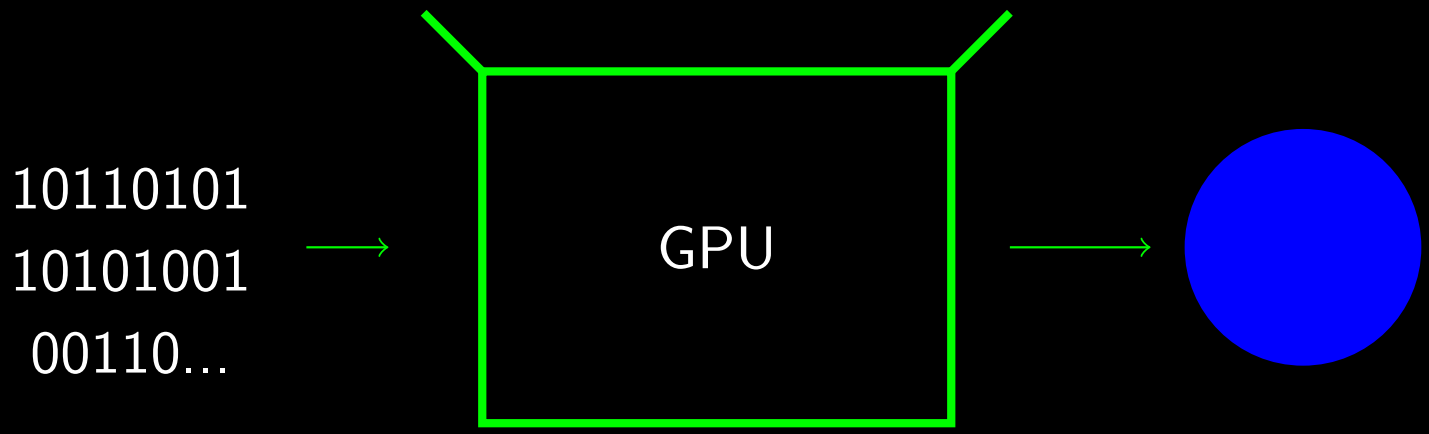
Results

Future work

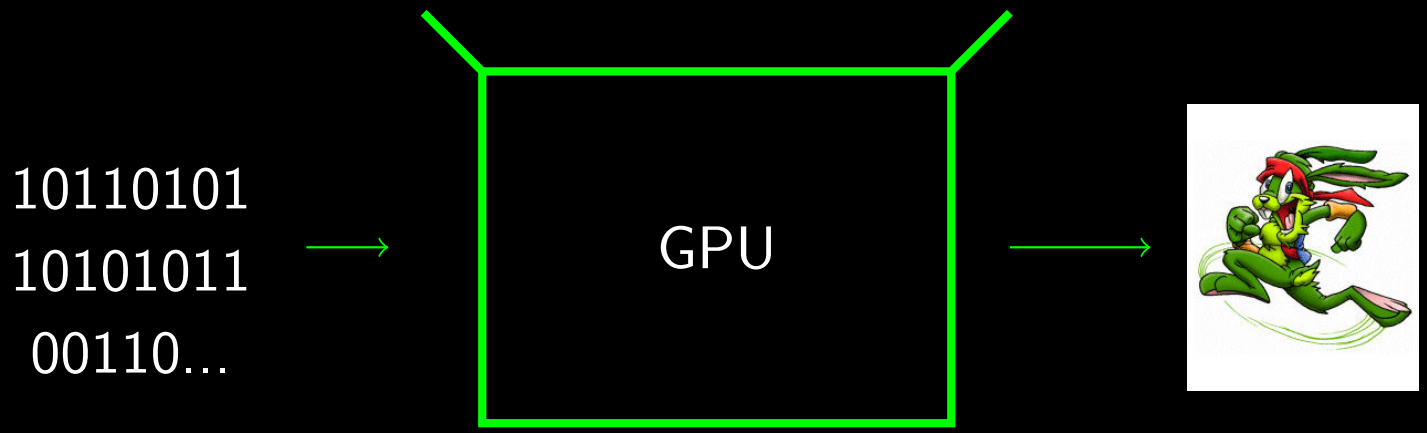
Challenge - draw me a red triangle



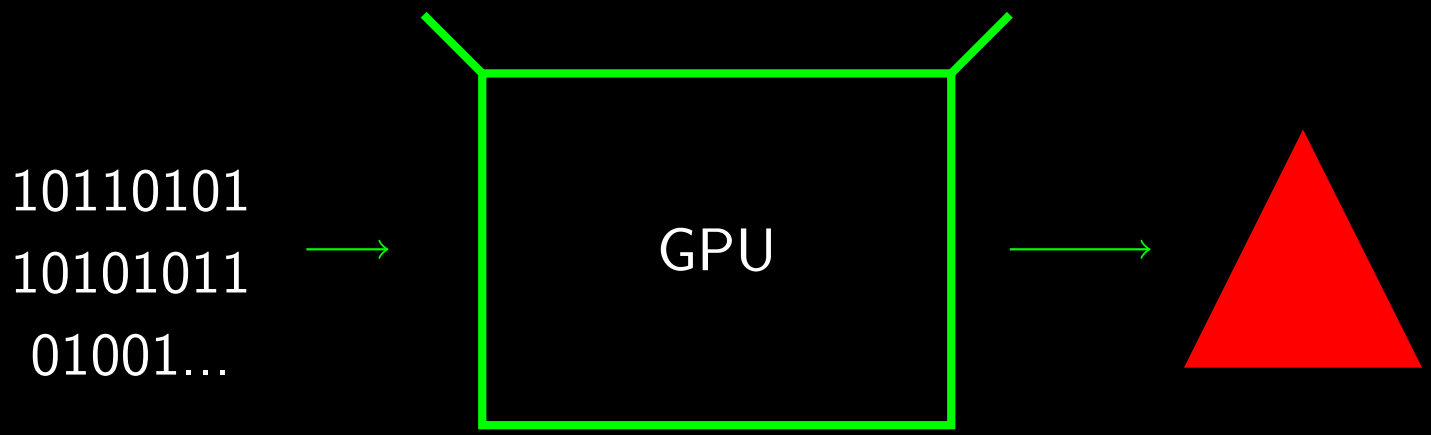
Challenge - draw me a red triangle



Challenge - draw me a red triangle



Challenge - draw me a red triangle



Challenge - change my clocks

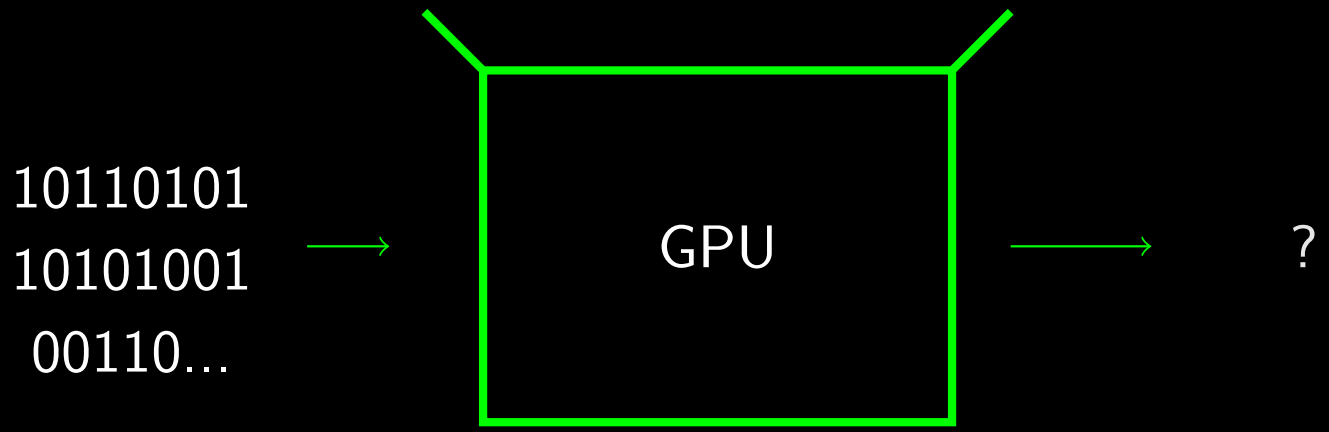


Table of Contents

Challenge

Process

Results

Future work

Process

Step 1: Understand the clock tree

Process - clock tree

Prior work: cycle counters

- Measure speed of clocks
- Observe result of clock register changes

- Map register to clock
- Map clock to VBIOS entry

Read-out tool 'nvatiming' in Envytools[1]

[1] http://github.com/envytools/envytools

Process - clock tree

VBIOS contains lots of parameters

```
 1   000cfb0 c35c e6ff 40c3 1012 0903 0004 ff00 0001
 2   000cfc0 0007 0100 1e00 0305 5040 0000 0000 0000
 3   000cfd0 9300 ff19 0000 8700 0000 010e 8700 0000
 4   000cfe0 001b cb00 0000 0115 8700 0080 0000 0000
 5   000cff0 0000 4007 0055 0000 0000 0000 0896 0055
 6   000d000 0064 0195 2a00 0043 0195 9500 0001 0195
 7   000d010 1500 0001 80ca 0000 0000 0000 ff00 0000
 8   000d020 0000 0000 0000 0000 0000 0000 0000 0000
 9   *
10   000d040 0000 0000 0000 0000 000f 0064 0000 0000
11   000d050 0000 0a96 0055 0000 025e bc00 0005 0316
12   000d060 1c00 0002 0195 1500 0001 810e 9500 0001
13   000d070 432a 1000 0508 080a 0004 0000 c700 0e00
14   000d080 ff00 0000 0000 0000 0000 ff00 0000 0000
15   000d090 0000 0000 0200 107f 9000 109f f310 0200
16   000d0a0 107f 9000 109f f210 ff00 0000 0000 0000
17   000d0b0 0000 ff00 0000 0000 0000 0000 ff00 0000
```

Process - clock tree

Envytools[2]

```
PM_Mode table at 0xcfb5. Version 4.0. RamCFG 0x2. Info_length 16.
Subentry length 3. Subentry count 9. Subentry Offset 16
Boot perflvl: 0x7
Header:
0xcfb5: 40 12 10 03 09 04 00 00 ff 01 00 07 00 00 01 00 1e 05
4 performance levels

-- ID 0x3 Voltage entry 80 PCIe link 2.5 GT/s--
0xcfc7: 03 40 50 00 00 00 00 00 00 00 93 19 ff 00 00 00
        0:0xcfd7: 87 00 00                          : core freq = 135 MHz
        1:0xcfda: 0e 01 00                          : shader freq = 270 MHz
        2:0xcfdd: 87 00 00                          : memclk freq = 135 MHz
        3:0xcfe0: 1b 00 00                          : vdec freq = 27 MHz
        4:0xcfe3: cb 00 00                          : unka0 freq = 203 MHz
        5:0xcfe6: 15 01 00                          : host freq = 277 MHz
        6:0xcfe9: 87 80 00                          : core intm freq = 135 MHz (force no PLL)
        7:0xcfec: 00 00 00                          : unk_engine freq = 0 MHz
        8:0xcfef: 00 00 00                          : unk_engine freq = 0 MHz
[...]
```

[2] http://github.com/envytools/envytools

Process

Step 2: Generate parameters

Process - parameters

Generate

- RAM timings
- Other reliability features - ODT, DLL...
- Link-training
- ... And a whole load of unknown bits

Strategy: mimic the blob

Process - parameters

MMIOTrace: Intercept communication between blob and device.

```
 1  [0] R 0x001538 0x00011111 PBUS+0x538 => 0x11111
 2  [0] W 0x001538 0x00011111 PBUS+0x538 <= 0x11111
 3  [0] R 0x0041a4 0x00063131 PCLOCK.VDCLK => { VCO_ENABLE | VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | ENABL
 4  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
 5  [0] R 0x0041a4 0x00063131 PCLOCK.VDCLK => { VCO_ENABLE | VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | ENABL
 6  [0] W 0x0041a4 0x00043131 PCLOCK.VDCLK <= { VCO_ENABLE | VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | ENABL
 7  [0] W 0x0041a4 0x00043131 PCLOCK.VDCLK <= { VCO_ENABLE | VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | ENABL
 8  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
 9  [0] R 0x00c040 0x20001000 PCONTROL.MASTER => { UNK12 | HOST = 277MHz }
10  [0] W 0x00c040 0x20001000 PCONTROL.MASTER <= { UNK12 | HOST = 277MHz }
11  [0] R 0x004160 0x00063131 PCLOCK.NVCLK => { VCO_ENABLE | VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | ENABL
12  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
13  [0] R 0x004120 0x00063031 PCLOCK.NVPLL_SRC => { VCO_ENABLE | VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | O
14  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
15  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
16  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
17  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
18  [0] R 0x004164 0x00023030 PCLOCK.SCLK => { VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | OUTPUT = VCO | VCO_
19  [0] R 0x00e8a4 0x05063c01 PNVIO.RPLL2.COEF => { M = 0x1 | N = 0x3c | UNK16 = 0x6 | UNK28 = 0x5 }
20  [0] R 0x004124 0x00063131 PCLOCK.SPLL_SRC => { VCO_ENABLE | VCO_SRC = RPLL2 | OUTPUT_1 = 100000 | EN
```

Process - parameters

SEQ

- Script language used for changing memory clock
- complete ISA
  - Read/write registers
  - Read/write memory
  - Sleep
  - Arithmetic
  - (Conditional) Branching
  - ...
- Runs on the GPU, bus can be cut off

Process

Step 3: Execute

Process - Execute

- Reclock memory
  - “Prepare”
  - Wait for VBLANK
  - Pause engines
  - Set clock
  - Set params
  - Reset clock-dependent subcomponents (DLL)
  - Resume engines
- Disable interrupts
- Pause engines
- Change other clocks + voltages (GPIO)
- Resume engines

Process

Step 4: Generalise

Table of Contents

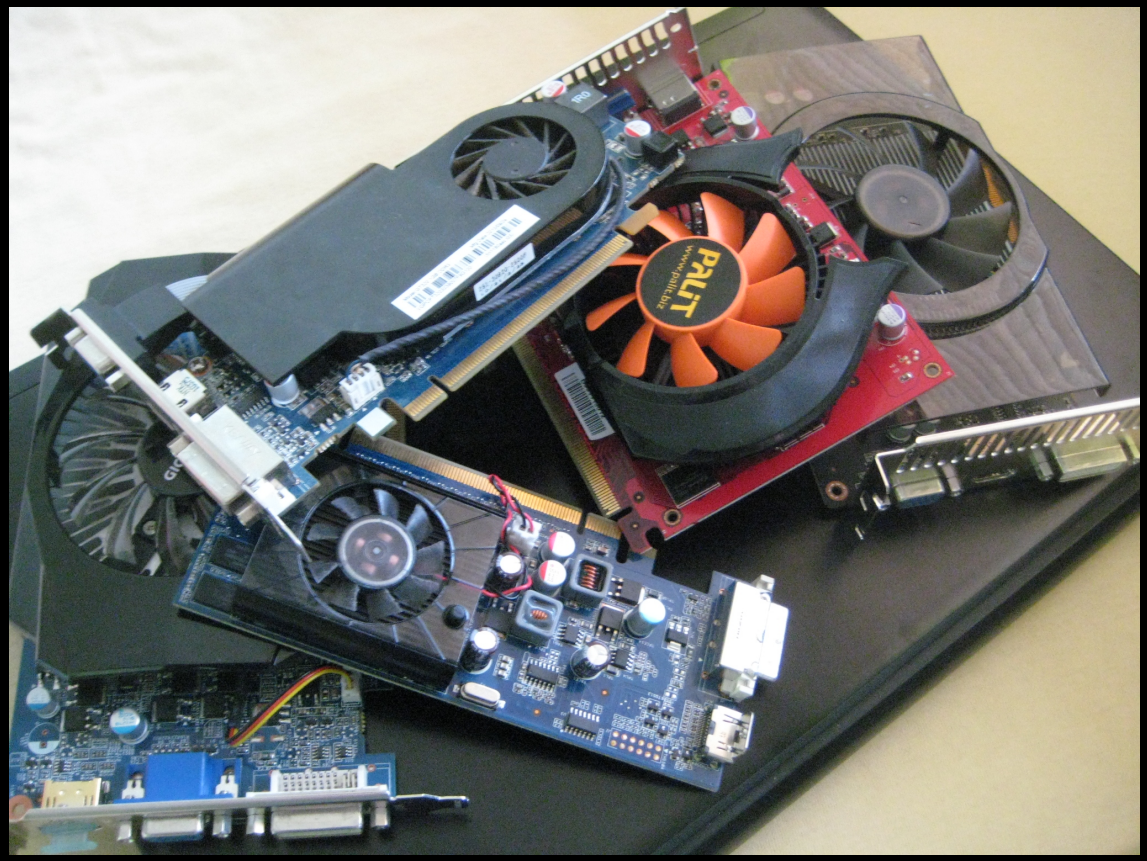
Challenge

Process

Results

Future work

Results

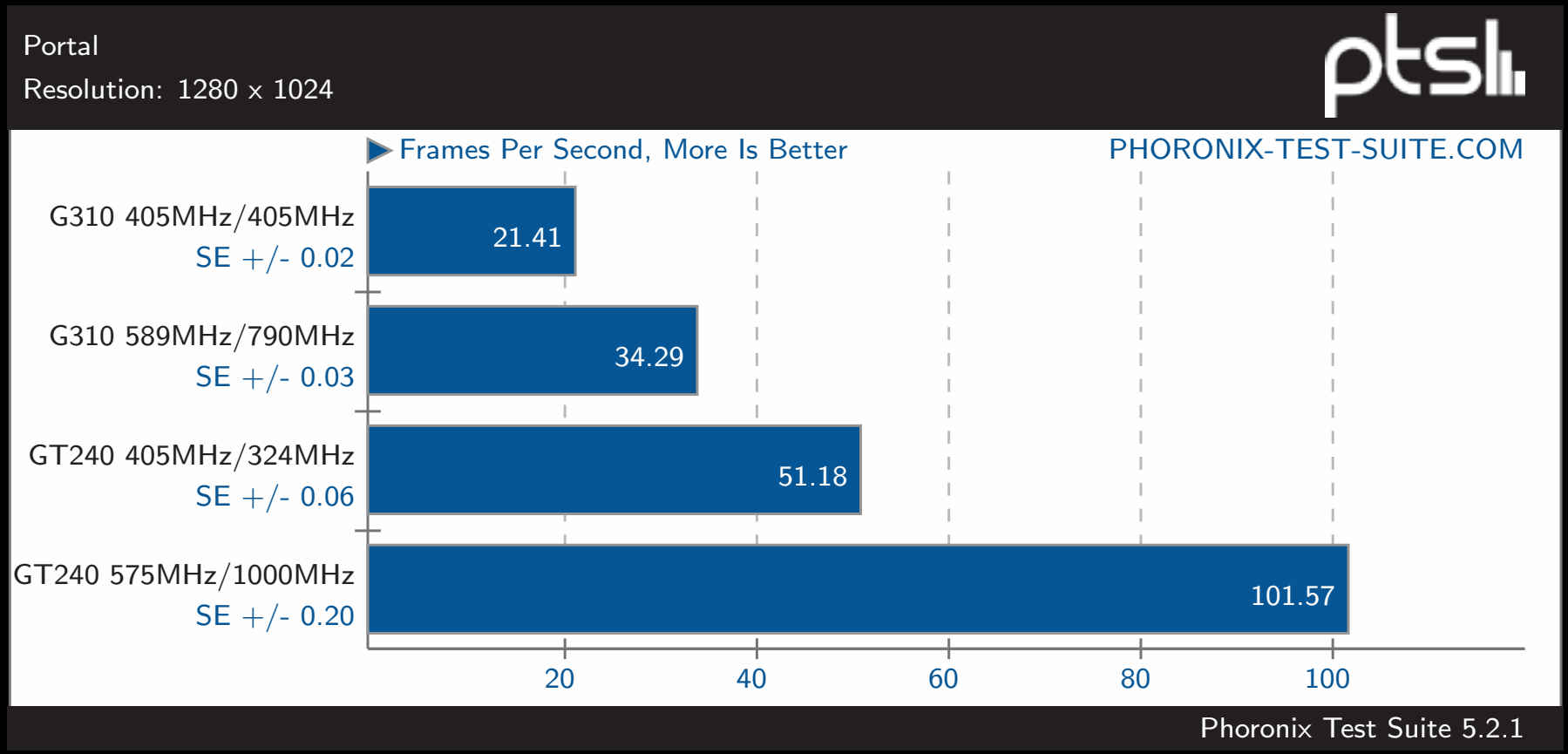


Results - Test system

- AMD FX-6300, 6-core @ 3,5GHz
- 8GB DDR3 @ 1,83GHz
- NVIDIA GPU
  - GT310, DDR3
  - GT240, GDDR3

- Fedora 20 x86_64
- Mesa 10.1.5
- xorg-x11-drv-nouveau 1.0.9-2

Results - Portal



Results - Xonotic

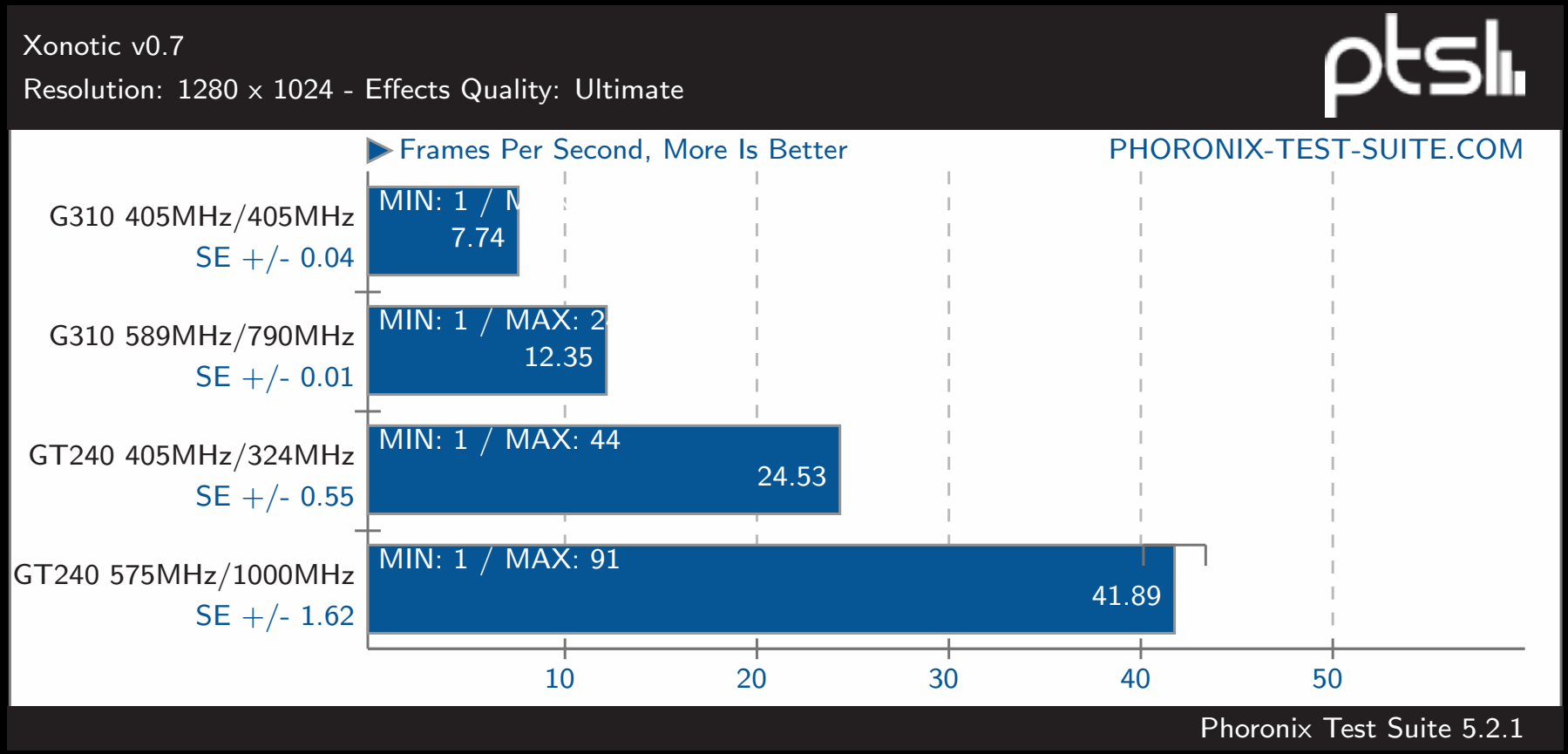


Table of Contents

Challenge

Process

Results

Future work

Future work

- Reliability
  - GPU doesn't pause nicely, changing clocks hangs under stress
  - Some DDR2 cards show corruption in highest perf lvl, bandwidth problem?
  - Link training should be done on boot, now postponed
- GDDR5 reclocking
- Load-based automatic clock/voltage adjustment
- Fermi, Kepler, Maxwell

Conclusion

- Clock scaling mandatory for speed
- Understandable process
  1. Understand the clock tree
  2. Generate parameters
  3. Execute
  4. Generalise
- Very rewarding in terms of FPS!

Code drops soon in a kernel near you... Or try
https://github.com/RSpliet/kernel-nouveau-nva3-pm

Oh, one more thing...

Hire me!

See http://roy.spliet.org

Questions

Questions?