Quicksilver

A 2D accelerator for the AFTx03 SoC

Manik Singhal, Jake Stevens, Erik Swan
Purdue University

Team

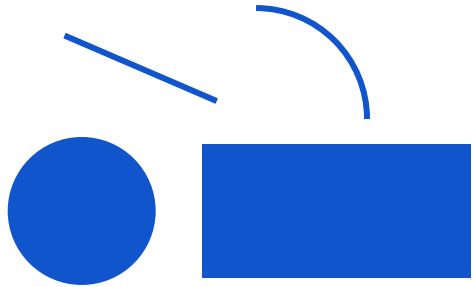
Manik Singhal
EE ‘16

Jake Stevens
CompE ‘16

Erik Swan
EE ‘16

What is Quicksilver?

- A 2D Accelerator for the AFTx03 SoC
- Draws basic 2D primitives
  - Lines
  - Filled Rectangle
  - Octant Arc
  - Filled Circle



- Frees processor from complex graphics operations, adds easy visual output option to the SoC.

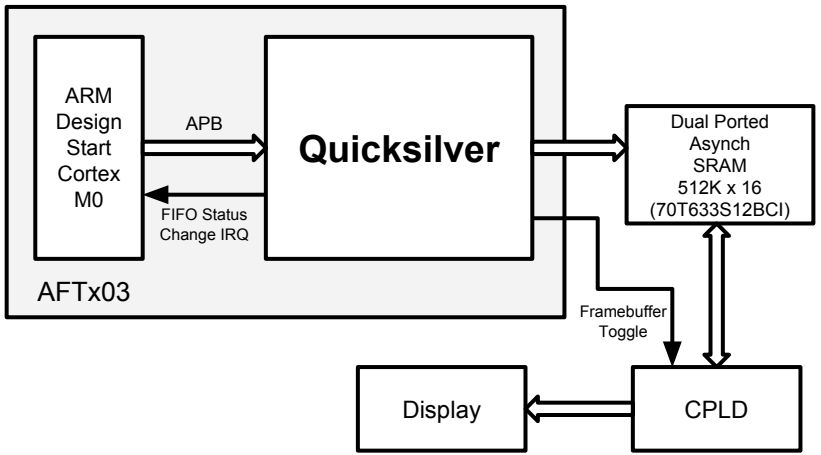
Quicksilver Advantages

- ➔ Freedom for the CPU = lots of untapped performance!
- ➔ Efficient and effective rendering algorithms (1px/cycle)
- ➔ Beautiful demos for the SoC

Source: adafruit.com

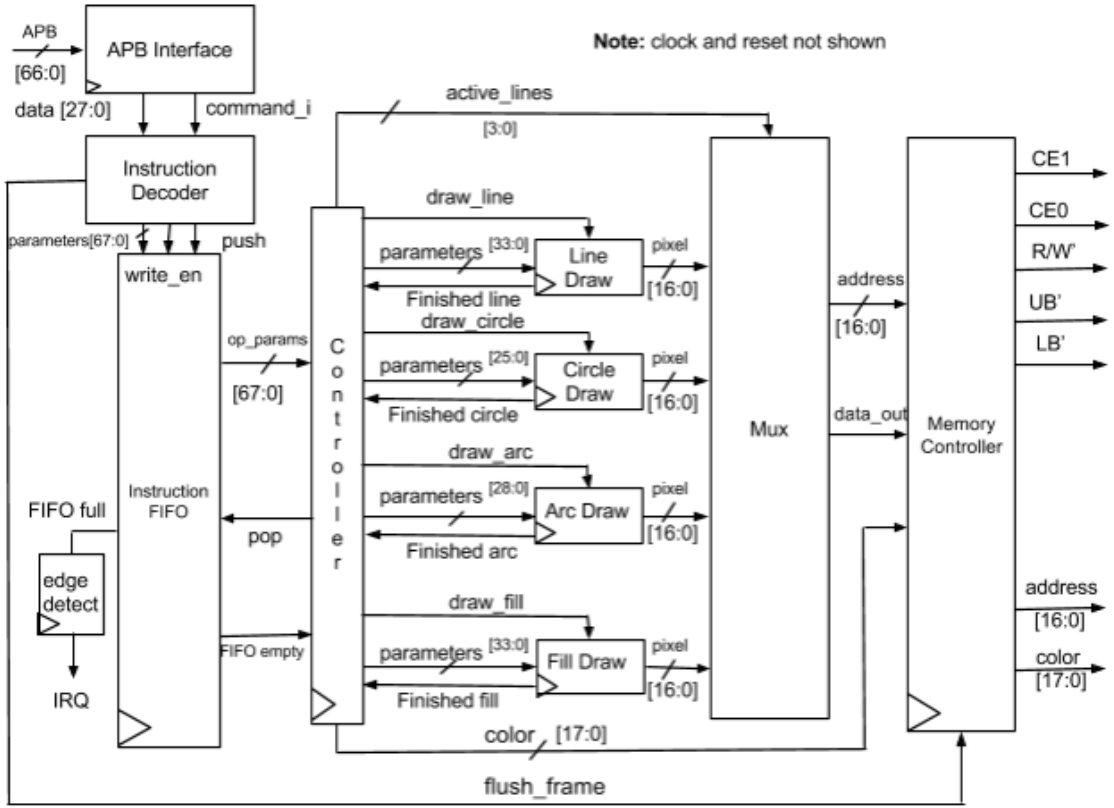
Interfacing with Quicksilver

- **ARM Cortex M0**
  - APB for commands/parameters
  - IRQ for change in FIFO full status
- **Dual Port SRAM**
  - Data bus for X, Y, R, G, B
    - X, Y (address, 17 bits)
    - R,G,B (data, 18 bits)
- **External CPLD**
  - Frame buffer toggle via SRAM for dual buffering



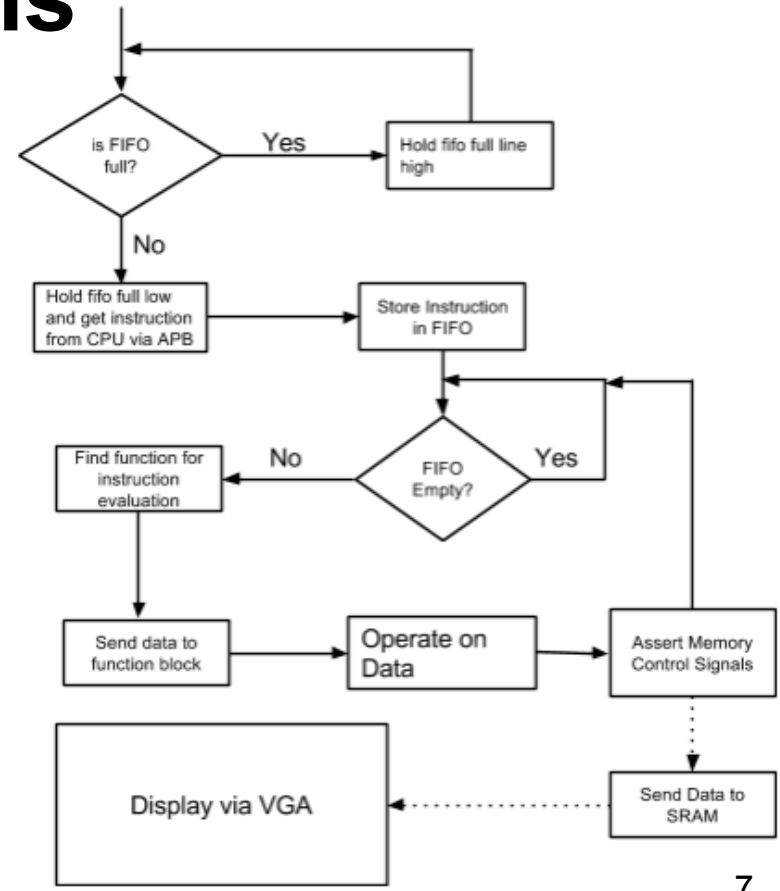
The Architecture of Quicksilver

- ➔ Independent functional blocks
- ➔ Instruction FIFO throttles CPU commands
- ➔ Memory controller output to SRAM

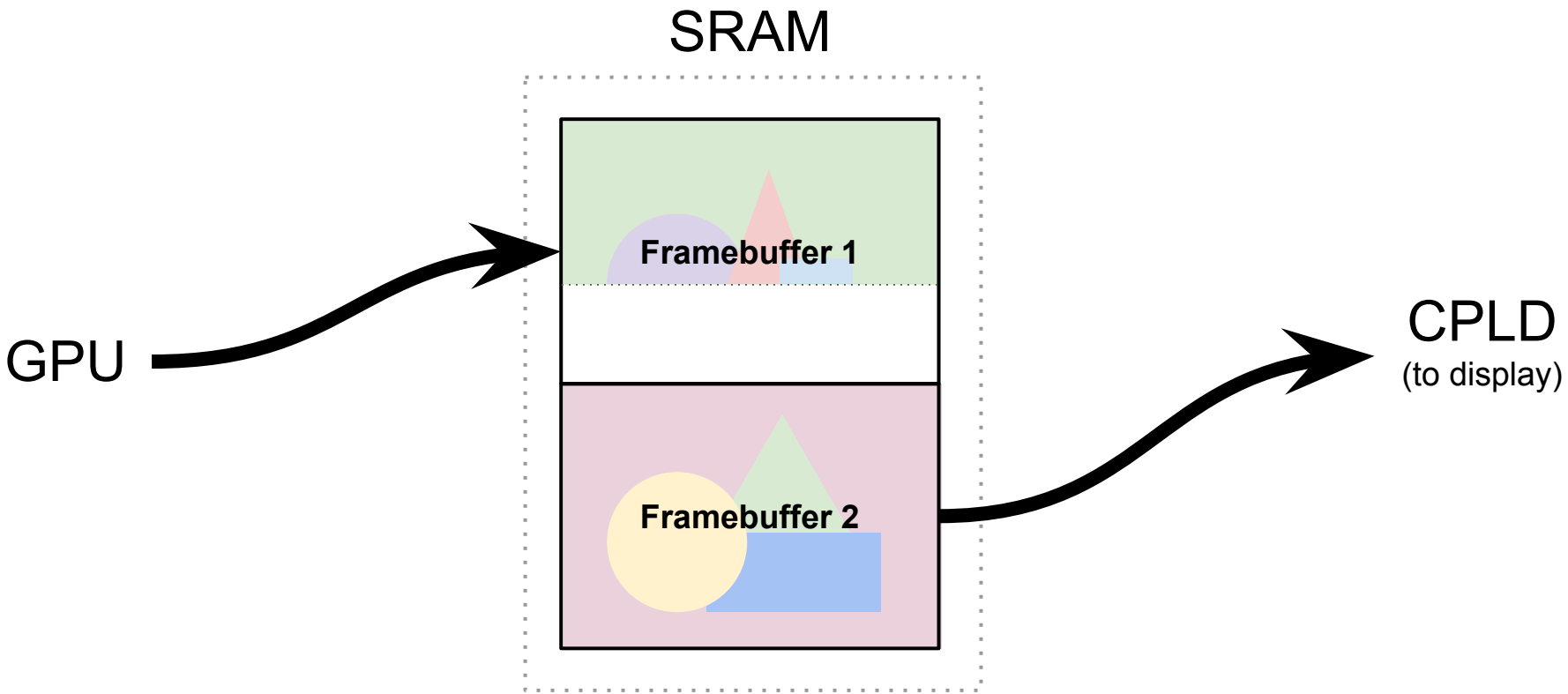


Sequence of Operations

- ➔ Two continuous cycles:
  - ◆ CPU → APB Decoder → FIFO
  - ◆ FIFO → Controller → Function Blocks → Memory Controller → SRAM
- ➔ Performance and flow determined by FIFO



Double Buffering



GPU and CPLD switch buffers when a frame is completed.

Packing Address: Need Fast Multiply

```
assign rtpaddy = addressy * `WIDTH;
```

Size (μm^2)	Delay (ns)
12,294	3.63

➔ Simple multiply

```
always @ (addressy)
begin
    reg_rtpaddy = 0;
    for (int i = 0; i < `HEIGHT; i = i+ 1)
    begin
      if (addressy == i)
        begin
          reg_rtpaddy = (i*(`WIDTH)) ;
        end
    end
end
```

Size (μm^2)	Delay (ns)
69,930	3.28

➔ Attempt at utilizing unrolling of for loop

```
case (addressy)
        `HEIGHT_BITS'd0: reg_rtpaddy = `SUM_BITS'd0;
        `HEIGHT_BITS'd1: reg_rtpaddy = `SUM_BITS'd320;
        `HEIGHT_BITS'd2: reg_rtpaddy = `SUM_BITS'd640;
        `HEIGHT_BITS'd3: reg_rtpaddy = `SUM_BITS'd960;
        `HEIGHT_BITS'd4: reg_rtpaddy = `SUM_BITS'd1280;
        `HEIGHT_BITS'd5: reg_rtpaddy = `SUM_BITS'd1600;
        `HEIGHT_BITS'd6: reg_rtpaddy = `SUM_BITS'd1920;
        `HEIGHT_BITS'd7: reg_rtpaddy = `SUM_BITS'd2240;
        `HEIGHT_BITS'd8: reg_rtpaddy = `SUM_BITS'd2560;
        `HEIGHT_BITS'd9: reg_rtpaddy = `SUM_BITS'd2880;
        `HEIGHT_BITS'd10: reg_rtpaddy = `SUM_BITS'd3200;
        `HEIGHT_BITS'd11: reg_rtpaddy = `SUM_BITS'd3520;
        `HEIGHT_BITS'd12: reg_rtpaddy = `SUM_BITS'd3840;
```

Size (μm^2)	Delay (ns)
26,109	0.32

➔ Prepopulated case structure from Python

Success Criteria

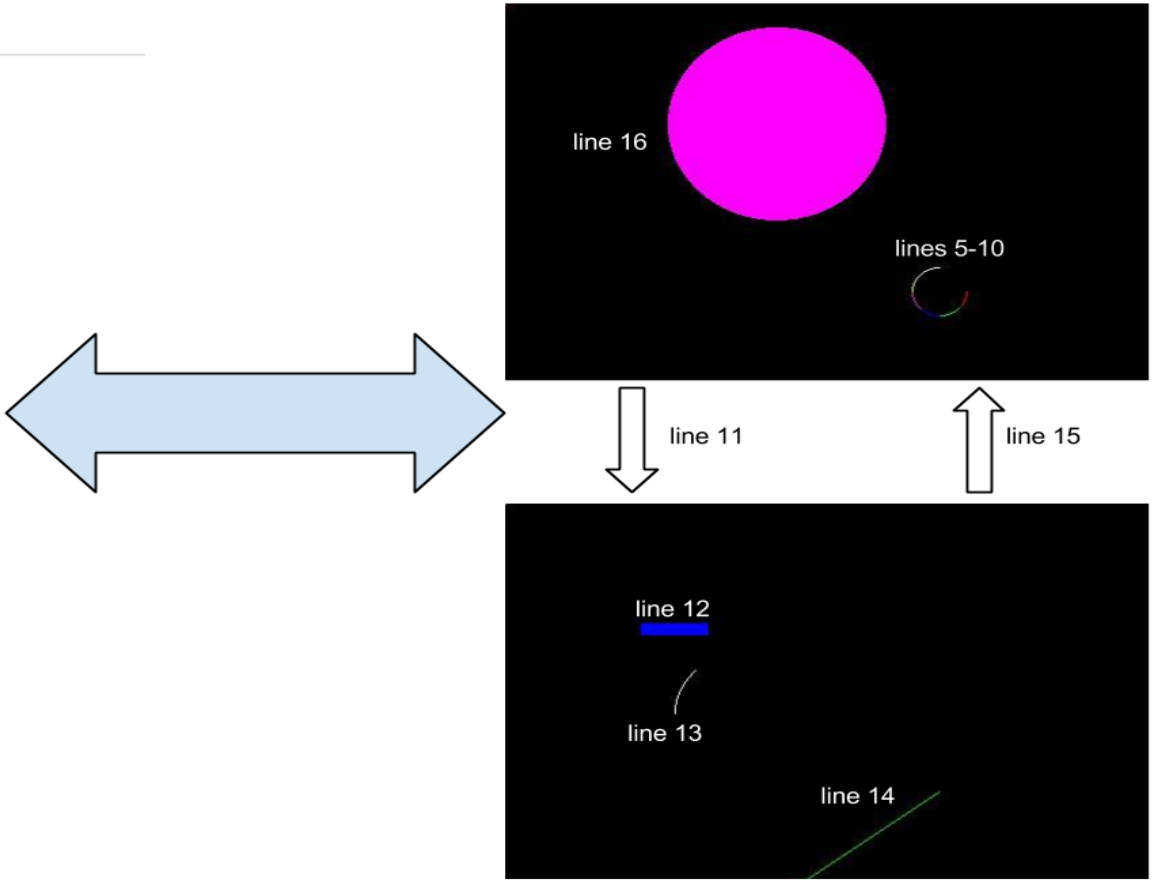
Success Criteria	
All Instructions Can Be Processed Correctly	✓
Correct Output of Primitive Function Blocks (Specifically Arc)	✓
Use of Interrupts to Prevent GPU Overload	✓
FPGA Simulation of A Complete Frame	✓
Functioning Memory Controller	✓
SoC with Quicksilver Integrated Passes LVS, DRC, and ERC	✓/ ✗

Process All Commands

```c
#include "gpu.c"

int main(void)
{
    GPUDrawArc(20,320,240,0,255,0,0); //R
    GPUDrawArc(20,320,240,1,0,255,0); //G
    GPUDrawArc(20,320,240,2,0,0,255); //B
    GPUDrawArc(20,320,240,3,235,31,228); //purple
    GPUDrawArc(20,320,240,4,255,255,0); //yellow
    GPUDrawArc(20,320,240,5,255,255,255); //white
    GPUFlush(); //drawing on two
    GPUDrawFilledRect(100,100,50,10, 0,0,255);
    GPUDrawArc(50, 175,175,4,255,255,255);
    GPUDrawLine(0,480, 320, 240, 0, 255, 0);
    GPUFlush(); //drawing on one
    GPUDrawCircle(80,200,100, 255,0,255);

    return 1;
}
```

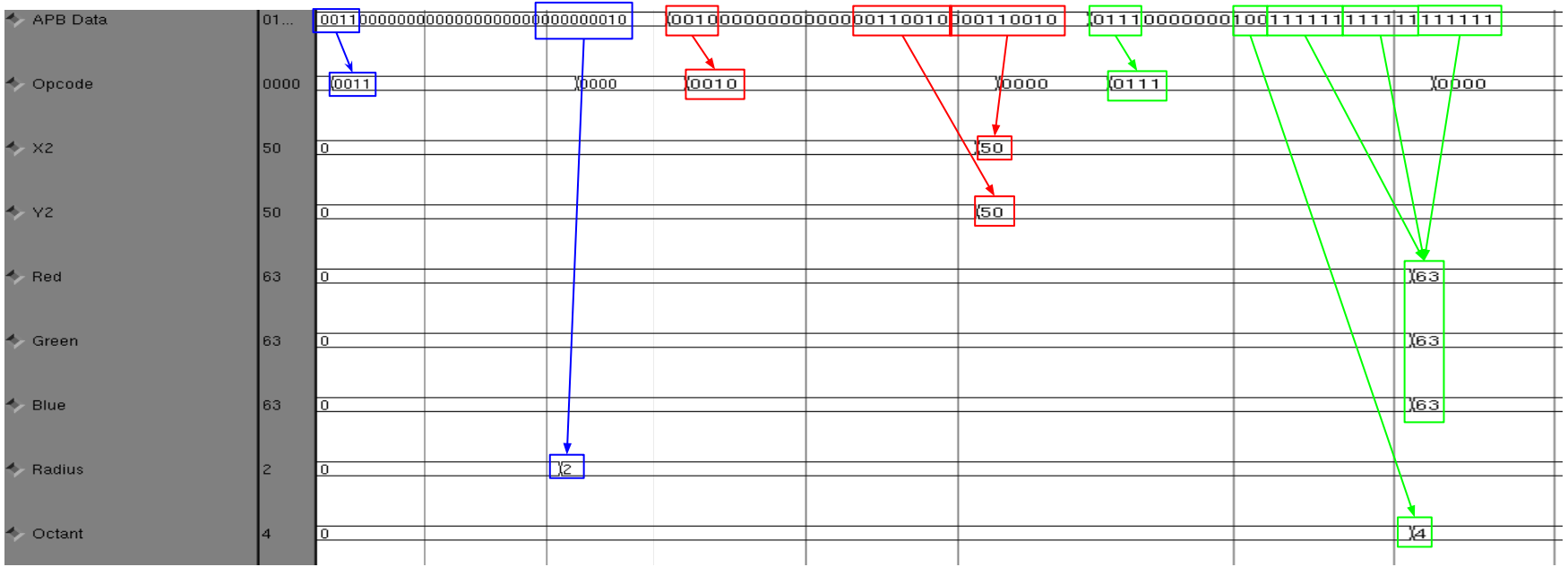


Processing: A Closer Look

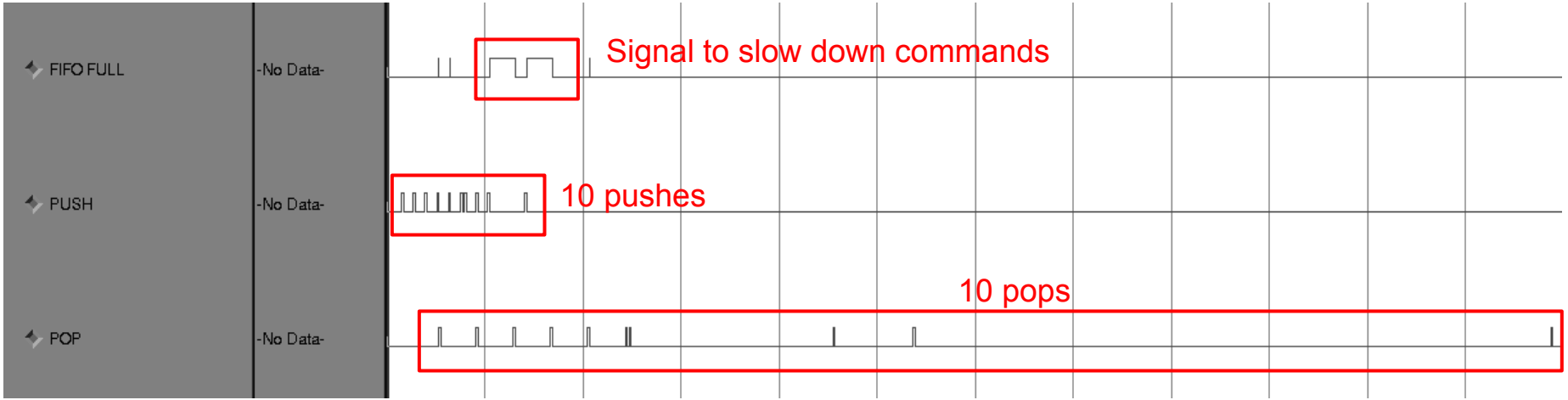
```
void GPUDrawArc(uint16_t radius, uint16_t x, uint16_t y, uint16_t oct,
                uint8_t r, uint8_t g, uint8_t b)
{
  GPUSetRad(radius);
  GPUSetXY2(x, y);
  GPUIssueInstruction(GPU_INST(GPU_CMD_DRAWARC, GPU_INST_ARC(oct, r,g,b)));
}
```

Relevant Opcodes

Set Radius	0011
Set XY2	0010
Draw Arc	0111

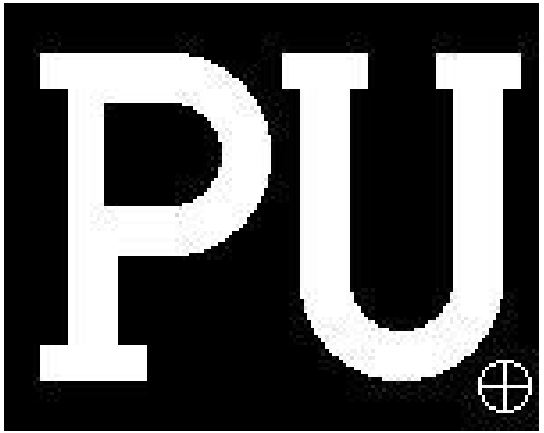


FIFO Interrupt



- ➔ FIFO signals full when there is one space left
  - ◆ Following instruction can be missed if full signaled at zero spaces
- ➔ Key: # of pushes == # of pops, pauses in pushes

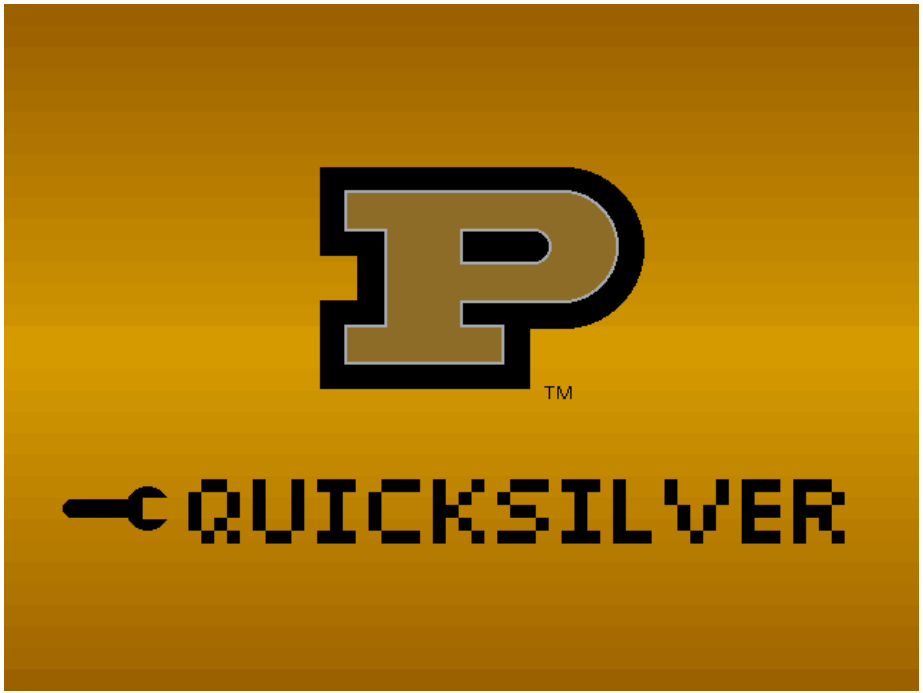
FPGA Simulation



- To fit on DE2 with SoC:
  - 1 bit of color
  - 150x120 Resolution

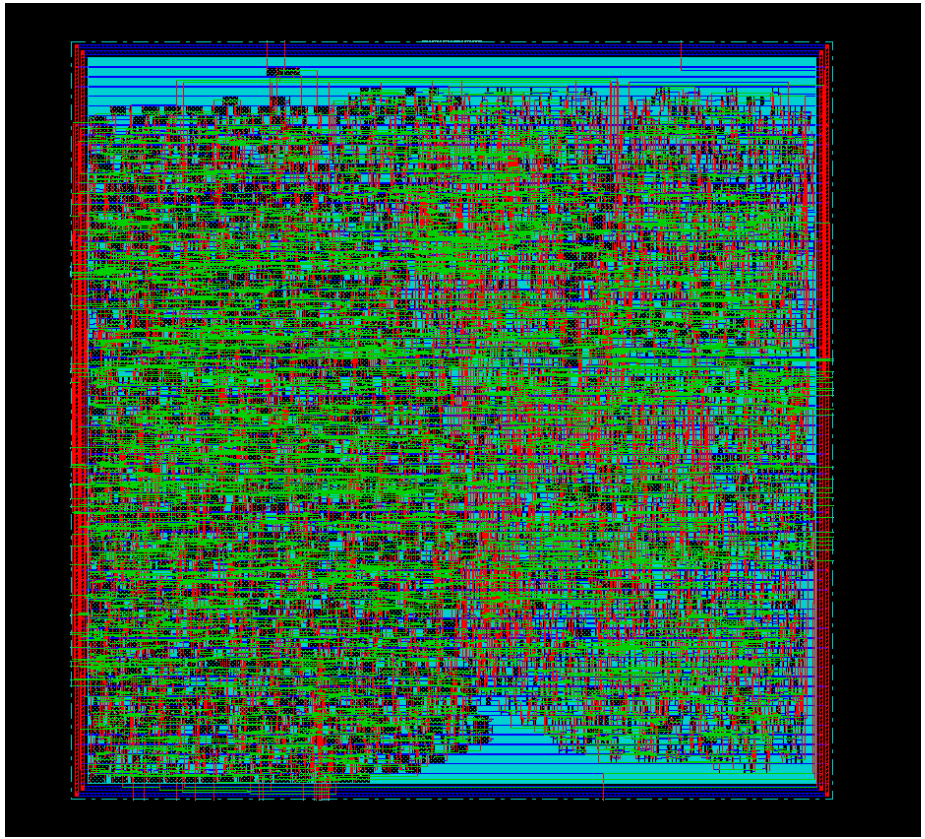
- Test program exercises:
  - Line, rectangular fill, circular fill, arc
  - Generation and handling of interrupts
    - First example of interrupts on any AFTx0x

A complex example



- ➔ Generated via mapped simulation
- ➔ 640 x 480, 24 bit color
- ➔ Neglecting CPU overhead, achieved 125 fps rendering over 32 frames
- ➔ **No bitmaps used!**

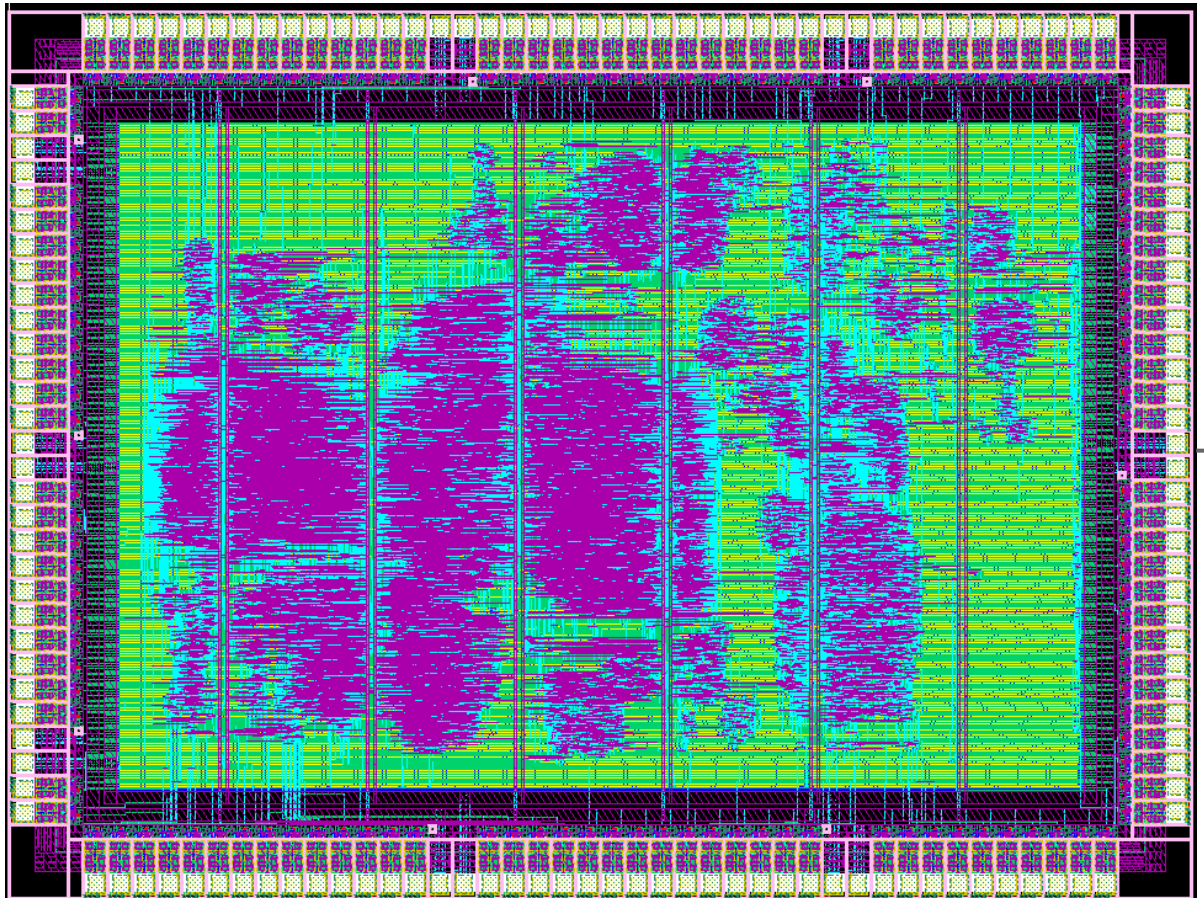
Quicksilver Alone at 0.5µm



	Critical Delay (ns)
Synthesized	17.17
Layout	13.034
Design Budget Estimate	13.5
Maximum Possible	20

	Area (mm^2)
Layout	5.555
Design Budget Estimate	5.576

Quicksilver with AFTx03 at 0.25µm



AFTX02	Size (µm)
Core	3000 x 3000
Full Layout	3600 x 3600

AFTX03	Size (µm)
Core	4200 x 3000
Full Layout	4800 x 3600

LVS, DRC, and ERC

- ➔ The layout of the AFTx03 still has to be tested for DRC and ERC checks.
- ➔ The pad frame is currently undergoing some minor changes to meet DRC and LVS

Challenges

- ➔ Design decisions best for SoC lifecycle
  - ◆ Use APB or AHB? Both?
  - ◆ Use shared memory? Dedicated memory?
- ➔ Incorporating into SoC
  - ◆ FPGA implementation
  - ◆ ARM toolchain for compiling and running code for the target
  - ◆ Analog layout of pad frame

Improvements

- Graphics Processing
  - Rasterization
  - Fonts
  - Triangles
  - Bitmap Operations
- Software
  - Further develop drivers and API
- Verification
  - Implement in Python to create gold standards of output
  - Nanosim

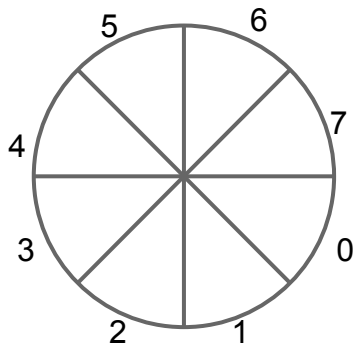
Questions?

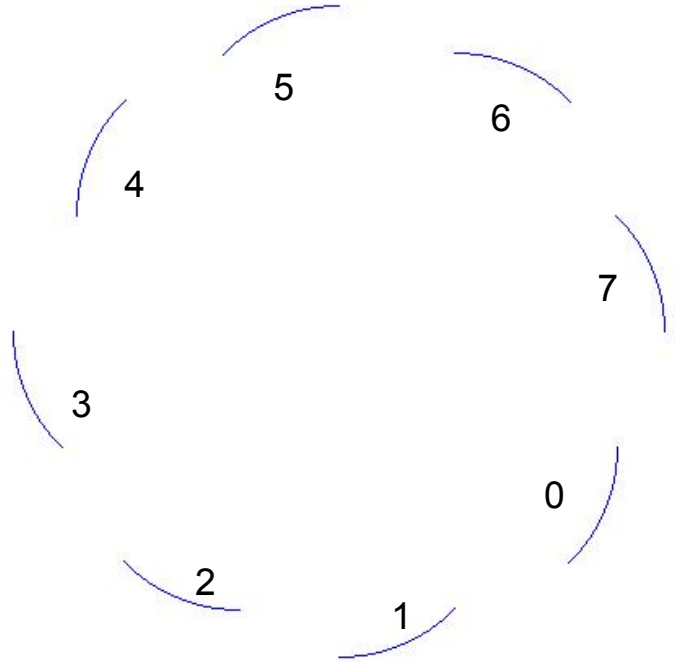
Arc functional block: validating output

Arc Draw Function Block draws octants

Arguments: (center, radius, octant, RGB)

Octants are labelled as following:





Memory Controller

From Data Sheet

$\overline{OE}$	$\overline{SEM}$	$\overline{CE_0}$	CE_1	$\overline{UB}$	$\overline{LB}$	$R/\overline{W}$	$\overline{ZZ}$	Upper Byte I/O_{9-17}	Lower Byte I/O_{0-8}	MODE
X	H	L	H	L	L	L	L	D_{IN}	D_{IN}	Write to Both Bytes

In FPGA code

```
gpu_ram gpu_ram(.address(gpu_ram_addr),
        .clock(HCLK),
        .data(color),
        .wren(CE1 & SEM & !CE0 & !UB & !LB & !R_W & !ZZ));
```

Result:

On Design Parallelization

- ➔ GPU's usually imply parallelisation
- ➔ Parallelization gives very little benefit due to memory constraint (1px/cycle)
- ➔ Functional blocks can attain 1px/cycle output!