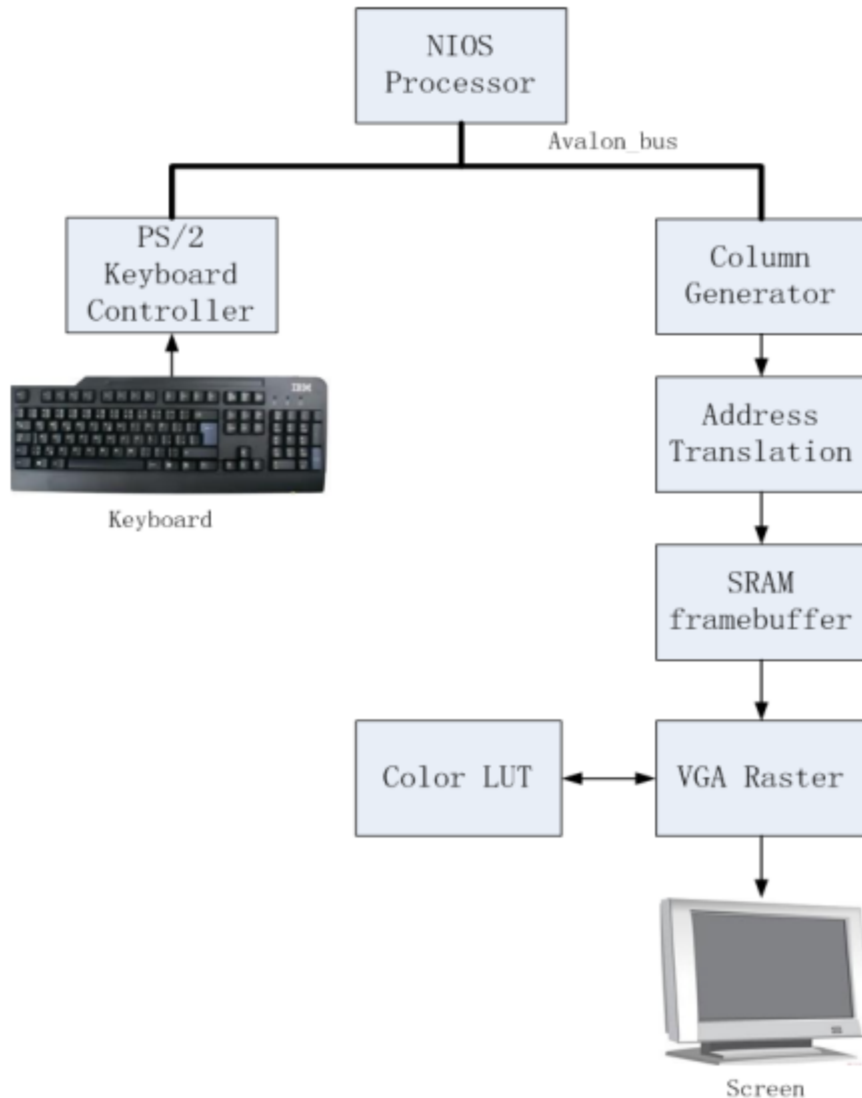
CUDoom: Raycasting Video Game

Alden Goldstein, Edward Garcia
Minyun Gu, Wei-Hao Yuan, Yiming Xu

Original Proposal



Milestone 1

- Implement raycasting algorithm in software
- Design several mazes

Milestone 2

- Integrate the algorithm with FPGA
- Realize hardware acceleration for the algorithm
- Display the world properly on screen

Milestone 3

- Add audio output to the game
- Complete game features, i.e. player movement, interface

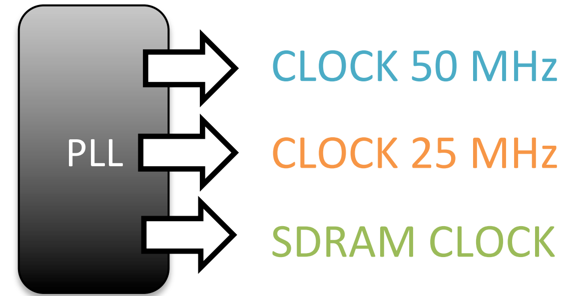
30 FPS Goal, Raycasting in software, SRAM Framebuffer

Actual Implementation

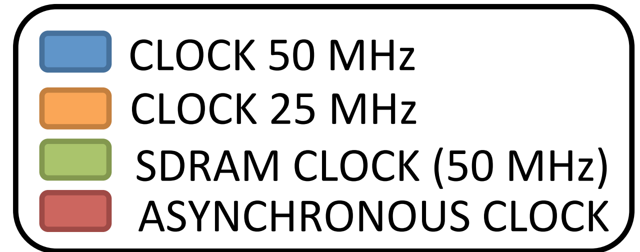
- All milestones complete
- 60 Frames per second
- Hardware raycasting acceleration
- Wall textures, floor textures, sky generation
- Multiple wall heights
- Background music from flash memory

System Overview

NIOS II
INTERRUPT
DE2WM8731
SOUND CONTROLLER
KEYBOARD
KEYBOARD CONTROLLER
AVALON BUS
SDRAM CONTROLLER
SDRAM
TRISTATE MASTER
CFI FLASH
NIOS INTERFACE
RAY FSM
FIFO
MEMORY
SRAM
SKY GENERATION
TEXTURE GENERATION
TEXTURE ROM
VGA RASTER
MONITOR



LEGEND



Software Overview

- Keeps track of player position
  - Local copy of world map
  - Polls keyboard and updates player direction /position
- Keeps track of casting rays from player's FOV
  - Calculates and stores angle measurements
  - Passes individual rays and player position to hardware
- Generates Music
  - Keeps track of sound generation through interrupts
  - Fetches new samples from flash memory

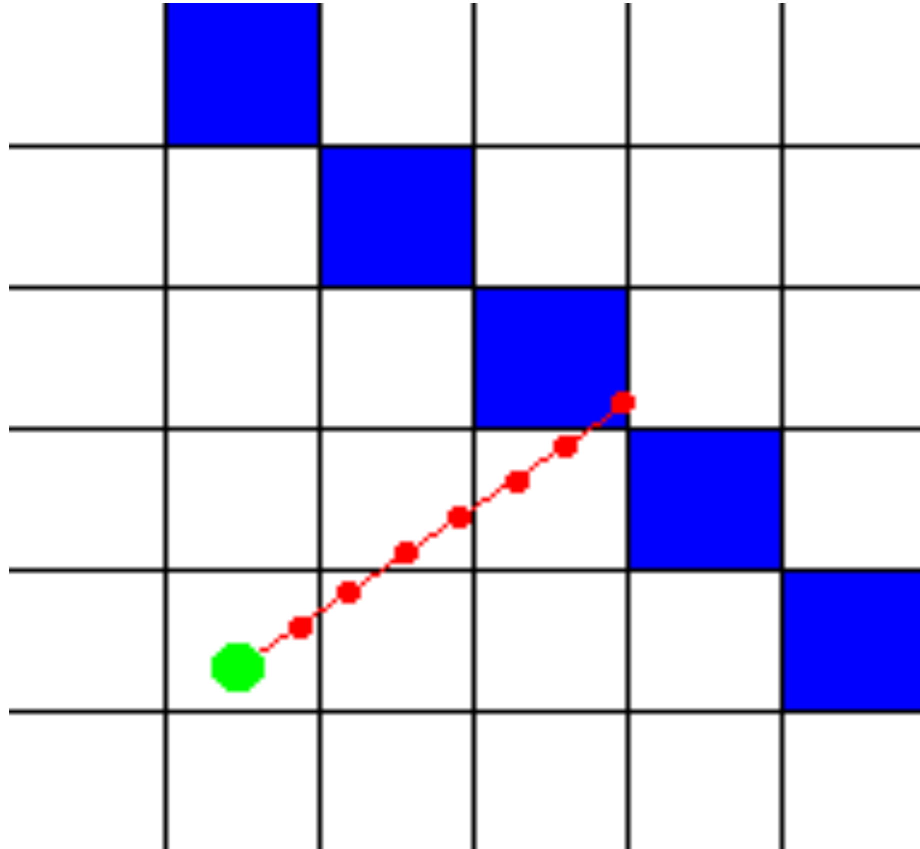
Hardware Overview

- Two main clock domains: Nios components (50 MHz), VGA Components (25 MHz)
- Raycasting acceleration calculates ray extension loop and generates intermediate variables such as wall heights
- Memory buffer for intermediate variables protected by dual clock FIFO
- Separate wall texture, floor texture, and sky components generate pixel calculations on VGA timings

Ray Casting in a Nutshell

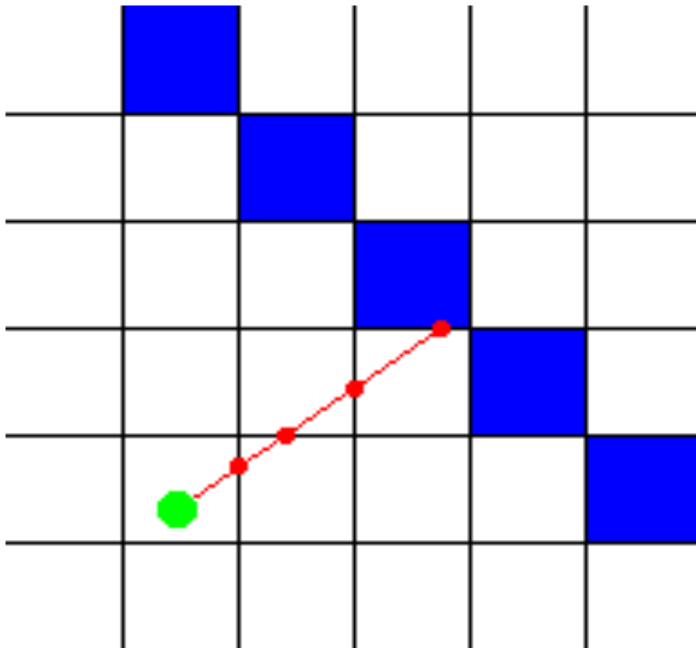
- We cast rays ….obviously!
- Based on perspective, farther walls appear smaller…more precisely, column height = inverse distance
- 2-D map layout, based on a matrix, thus all walls must be square (can be diagonal in more advanced ray casters)
- So, we cast rays to find wall on 2-D map, and used the distance to calculate the perceived column height

How to find walls



Lode Vandevenne, Lode's Computer Graphics Tutorial,
http://lodev.org/cgtutor/raycasting.html, 2007

DDA



- Modified Breshenham's
- Covers ALL Walls
- Used in LodeV's software template
- FAST and never misses a wall
- Seems ideal…right??

Lode Vandevenne, Lode's Computer Graphics Tutorial, http://lodev.org/cgtutor/raycasting.html, 2007

LoopBack

- Because hops in DDA are quantized, it can be prone to ugly, erratic, errors if not enough precision is used (such as fixed point software)
- A happy medium is to employ a loopback, in which edges are refined after iteration.
- The artifact from missing a wall is much more predictable and less ugly than those of DDA, and rest of the wall is smooth as with DDA. Slower, but more robust = less risky option for our project

Ray FSM

Motivation: Casting Rays is an iterative procedure… can be very slow as mentioned

To get across a 32 X 32 map using 1/32 of a square increments, can be as large as….

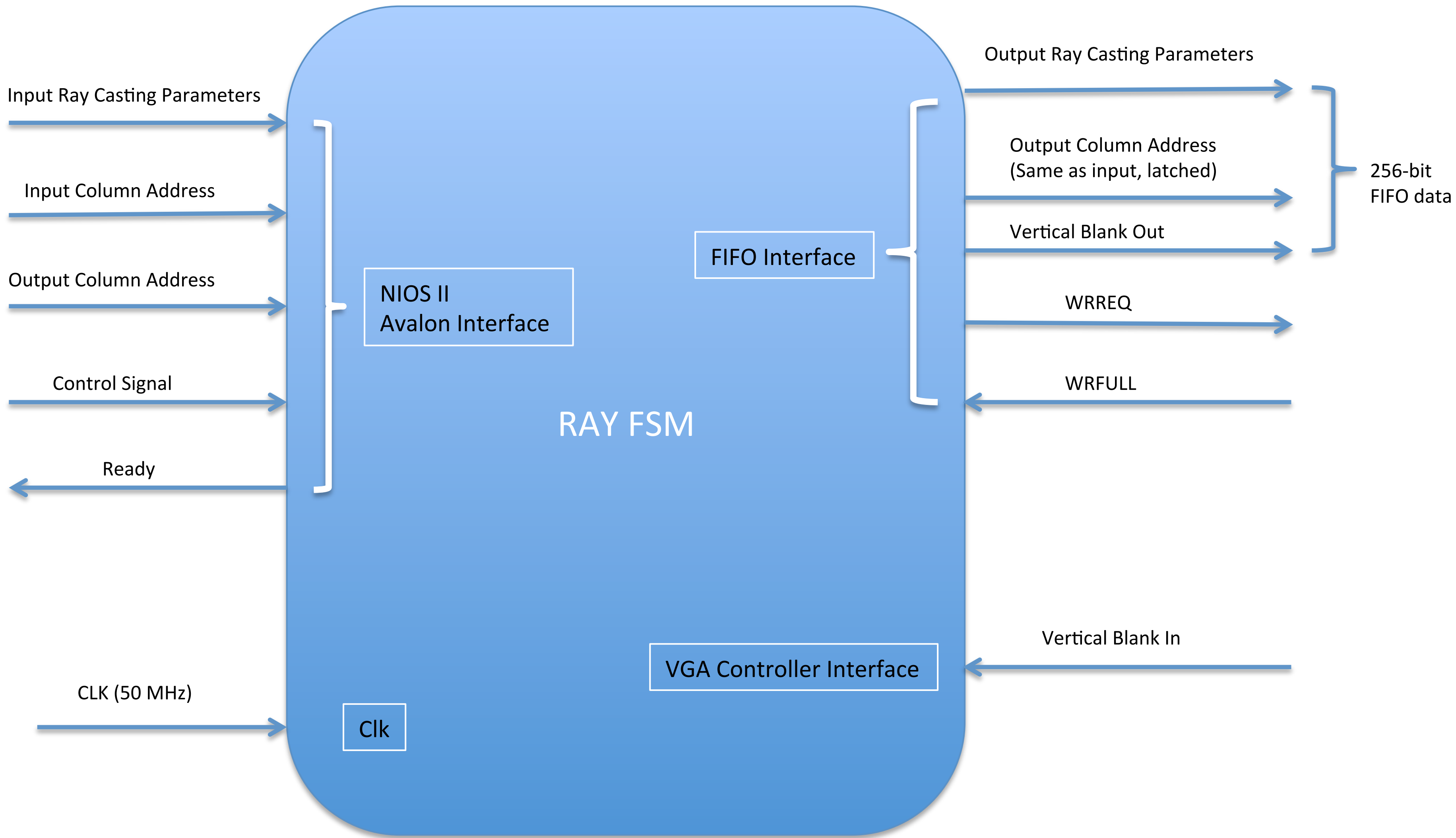
32 X 32 X sqrt(2) = 1500 iterations per column

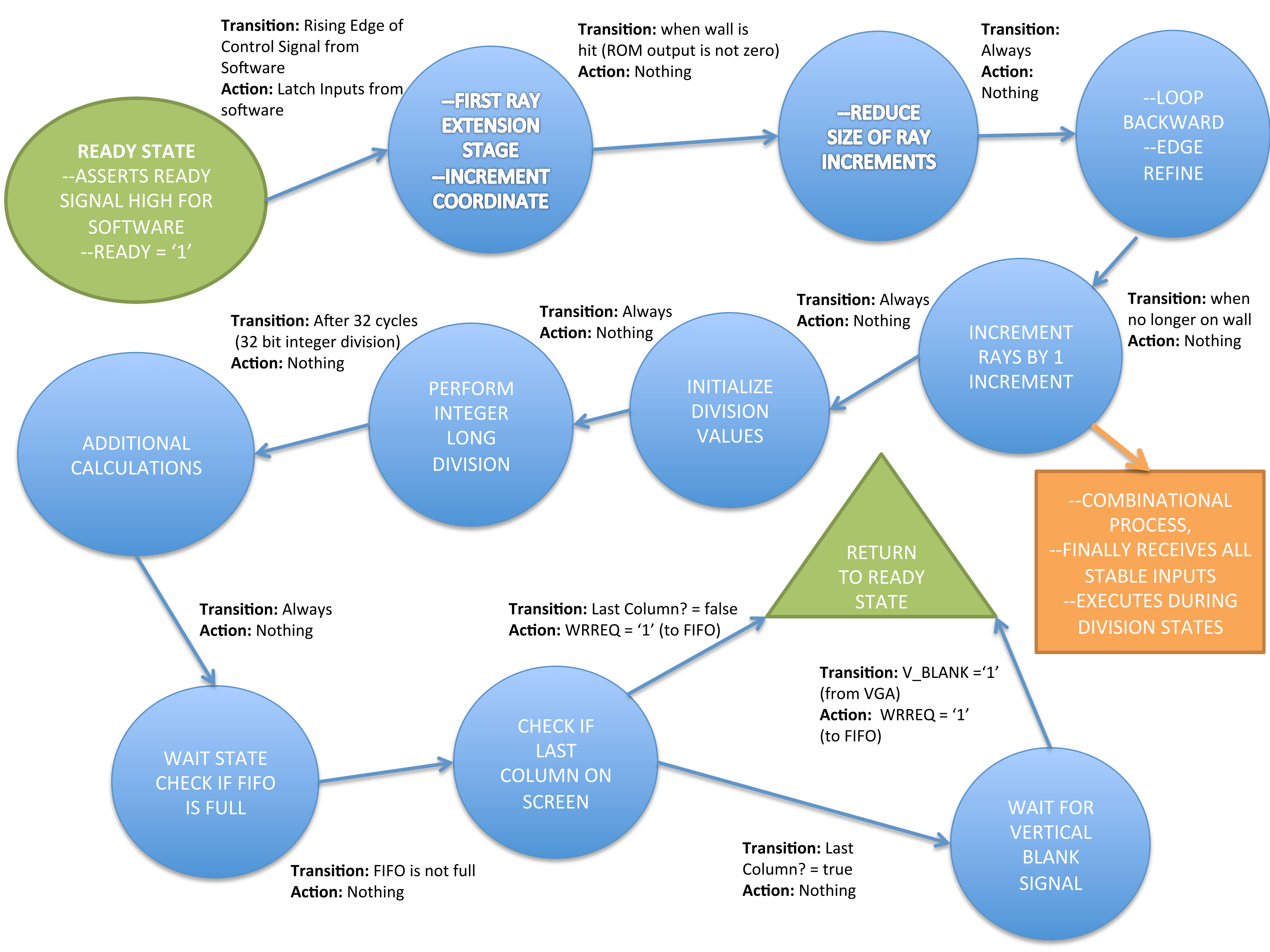
→1500 X 640 =

almost 1 million iterations per frame!

Ray FSM (continued)

- Loops in software carry large overhead + serial instructions within loop
- Why not increment at 50 MHz --> need hardware
- Share burden between hardware and software = efficient pipelining


Input Ray Casting Parameters
Input Column Address
Output Column Address
Control Signal
Ready
CLK (50 MHz)
NIOS II
Avalon Interface
Clk
RAY FSM
FIFO Interface
VGA Controller Interface
Output Ray Casting Parameters
Output Column Address
(Same as input, latched)
Vertical Blank Out
256-bit
FIFO data
WRREQ
WRFULL
Vertical Blank In



READY STATE
--ASSERTS READY SIGNAL HIGH FOR SOFTWARE
--READY = ‘1’
Transition: Rising Edge of Control Signal from Software
Action: Latch Inputs from software
--FIRST RAY EXTENSION STAGE
--INCREMENT COORDINATE
Transition: when wall is hit (ROM output is not zero)
Action: Nothing
--REDUCE SIZE OF RAY INCREMENTS
Transition: Always
Action: Nothing
--LOOP BACKWARD
--EDGE REFINE
Transition: when no longer on wall
Action: Nothing
INCREMENT RAYS BY 1 INCREMENT
Transition: Always
Action: Nothing
INITIALIZE DIVISION VALUES
Transition: Always
Action: Nothing
PERFORM INTEGER LONG DIVISION
Transition: After 32 cycles (32 bit integer division)
Action: Nothing
ADDITIONAL CALCULATIONS
--COMBINATIONAL PROCESS,
--FINALLY RECEIVES ALL STABLE INPUTS
--EXECUTES DURING DIVISION STATES
Transition: Always
Action: Nothing
WAIT STATE CHECK IF FIFO IS FULL
Transition: FIFO is not full
Action: Nothing
CHECK IF LAST COLUMN ON SCREEN
Transition: Last Column? = false
Action: WRREQ = ‘1’ (to FIFO)
RETURN TO READY STATE
Transition: Last Column? = true
Action: Nothing
WAIT FOR VERTICAL BLANK SIGNAL
Transition: V_BLANK =‘1’ (from VGA)
Action: WRREQ = ‘1’ (to FIFO)

Frame Sync

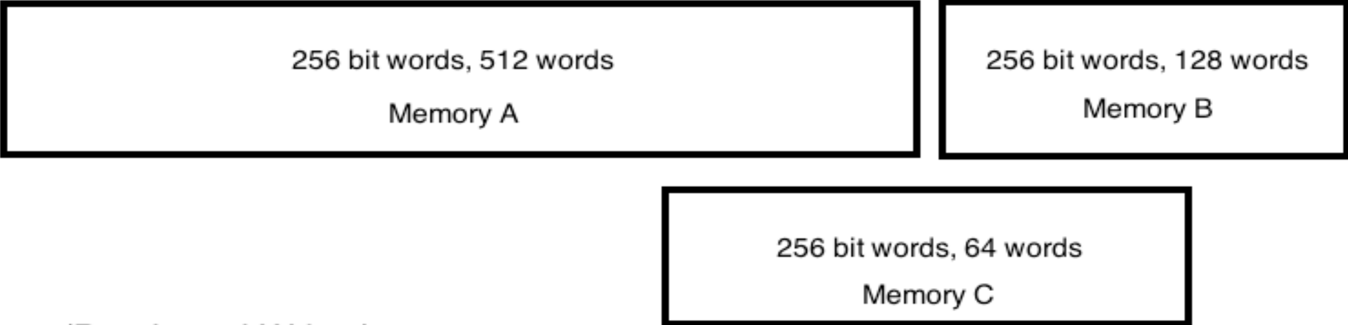
- Edwards wanted Frame Sync, so naturally we put it in…
- David wanted Anti-Aliasing, so naturally we left it out…

Why Frame Sync was annoying

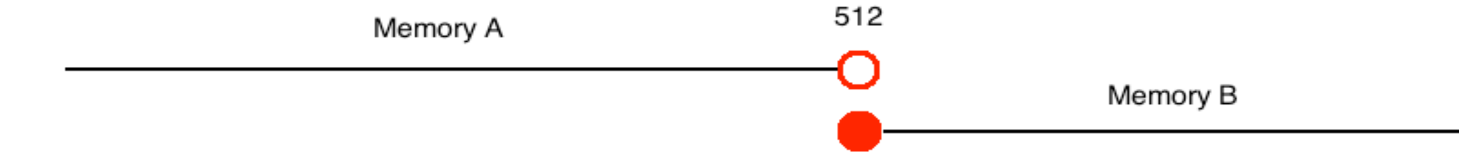
- Not to big of a deal…just double memories, have software wait on V-blank (via FSM), and toggle memories on V-blank… so why is it hard?
- 224 bits of parameters per column X 640 columns, naturally lends itself to 256 bits of parameters and 1024 addresses…
- But this is too big when we double it!!

Memory

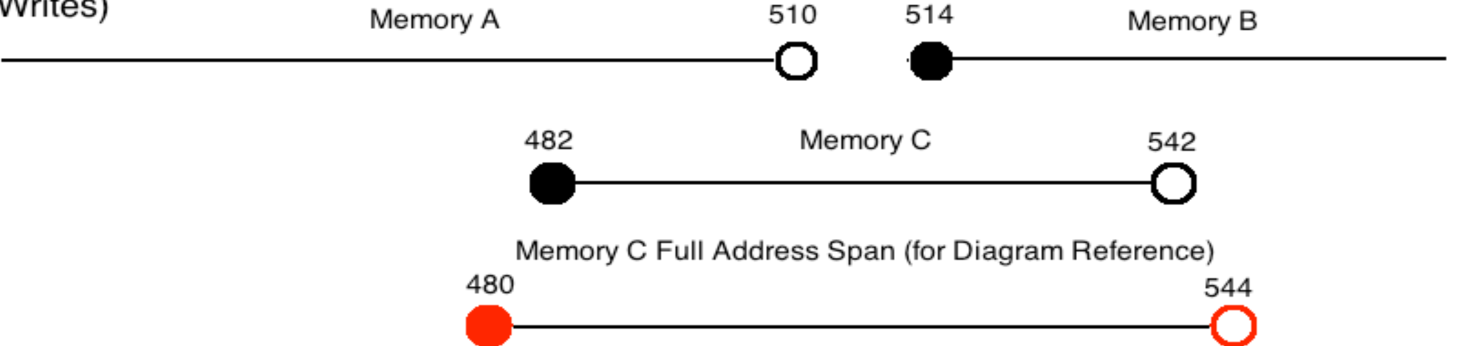
- Two Solutions: Cut bits or cut addresses
- Cut down bits to 192, and splice a 128 bit by 1024 memory, with a 64 bit by 1024 memory. Requires 96 M4K blocks according to Megafunction Wizard.
- Use a full 256 bits, but have a memory with 512 addresses and a memory with 128 addresses. Requires 74 M4K blocks according to Megafunction Wizard.
- We used second scheme, to preserve memory and bits (help image more), however, switching addresses lead to a strange ugly line where switch was. We fix this by using memory addressing scheme on the next page



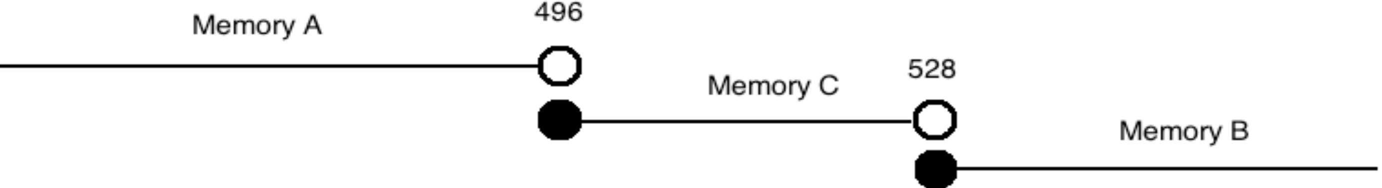
Old Scheme (Reads and Writes)

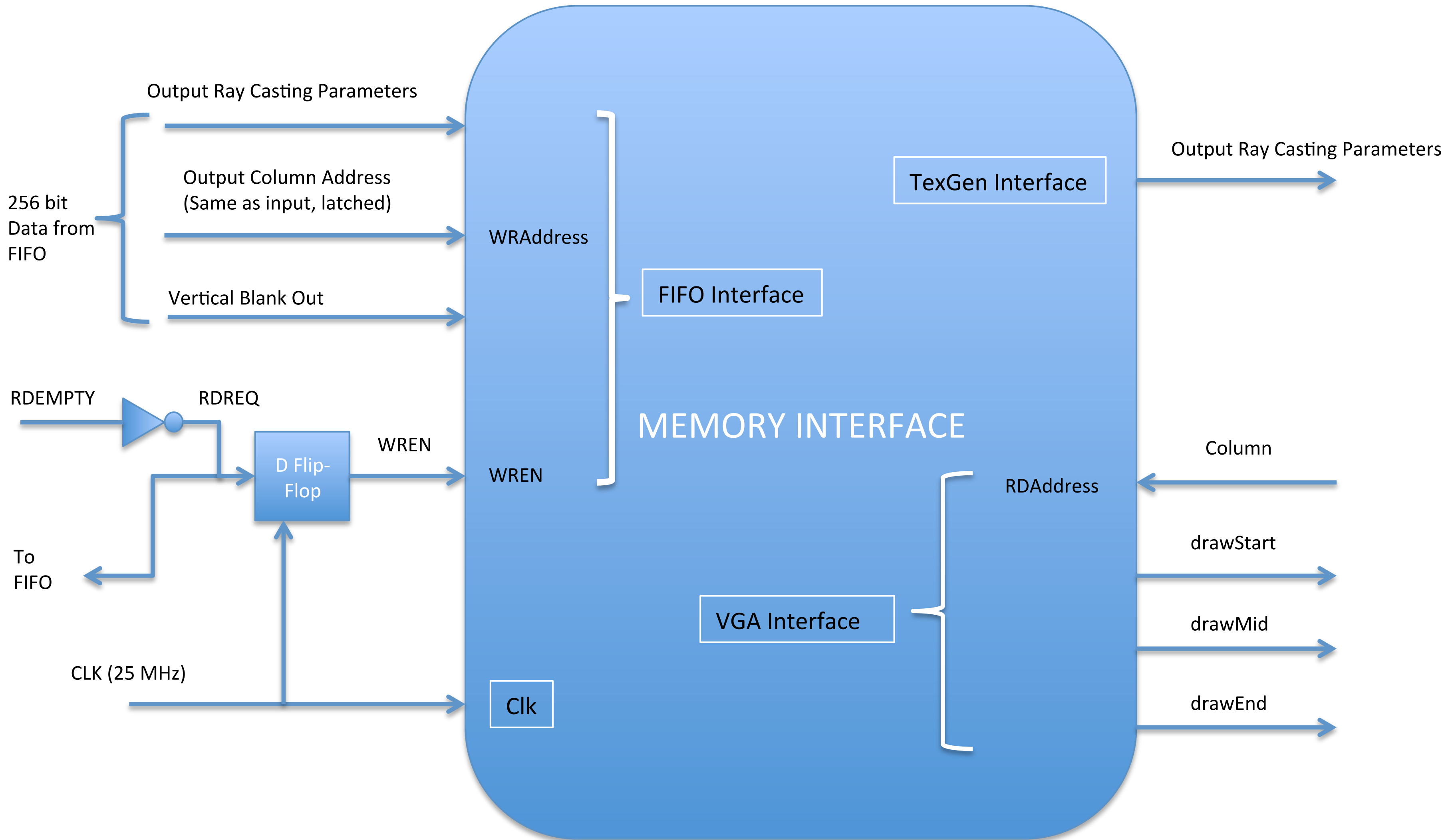


New Scheme (Writes)



New Scheme (Reads)





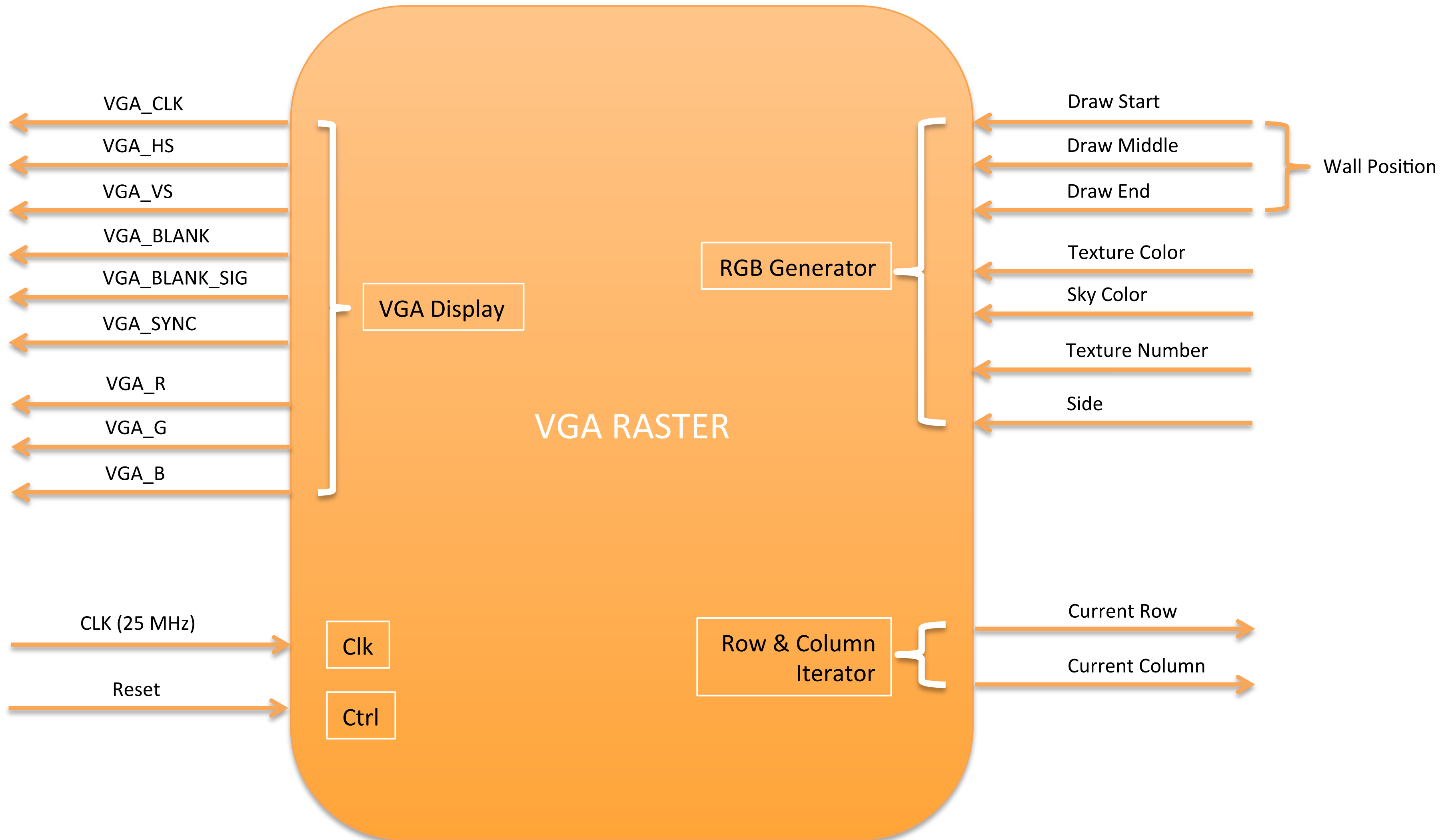
Output Ray Casting Parameters
256 bit
Data from
FIFO
Output Column Address
(Same as input, latched)
Vertical Blank Out
RDEMPTY
RDREQ
D Flip-
Flop
WREN
To
FIFO
CLK (25 MHz)
WRAddress
WREN
FIFO Interface
TexGen Interface
MEMORY INTERFACE
VGA Interface
Clk
RDAddress
Output Ray Casting Parameters
Column
drawStart
drawMid
drawEnd

FIFO

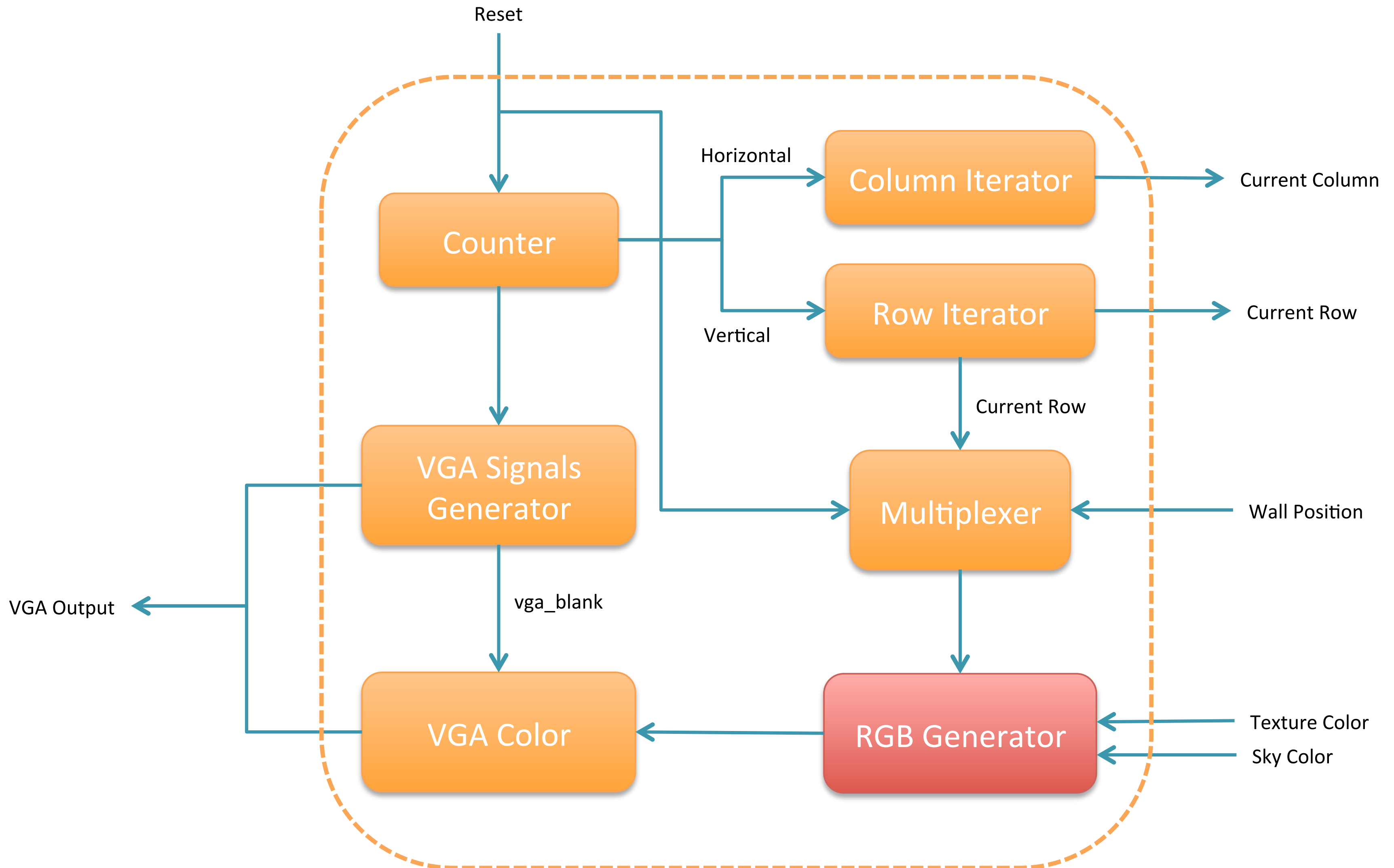
Motivation: System crashes, seemed to be result of corrupted M4K memory. M4K was interacting with two clocked domains, read addresses and outputs went to 25 MHz domain, writes came from 50 MHz domain (writes come from FSM)

Solution: Make M4K run on 25 MHz, add FIFO to allow Ray FSM to write to M4K Ram. Column addresses (and blank signal), appended to data, M4K constantly reads from FIFO and writes to address encapsulated within data

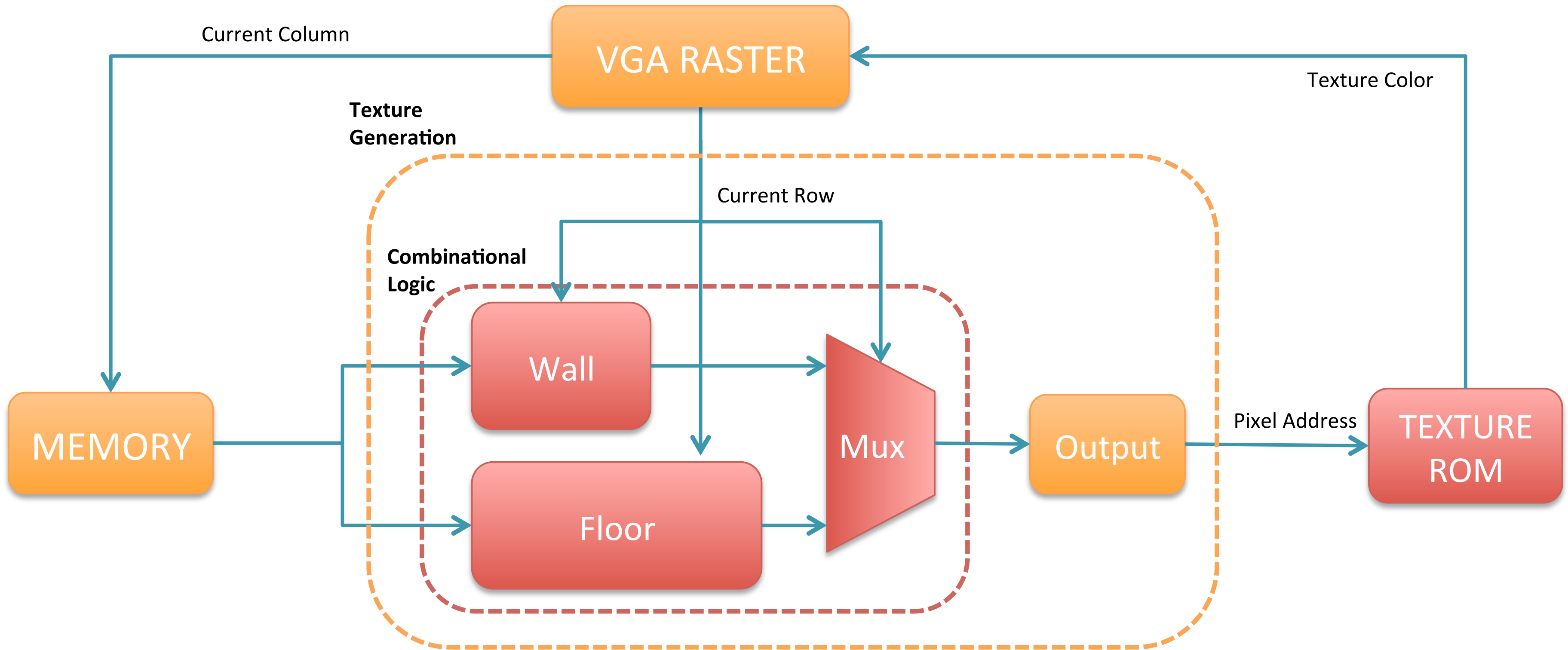
VGA Rastering



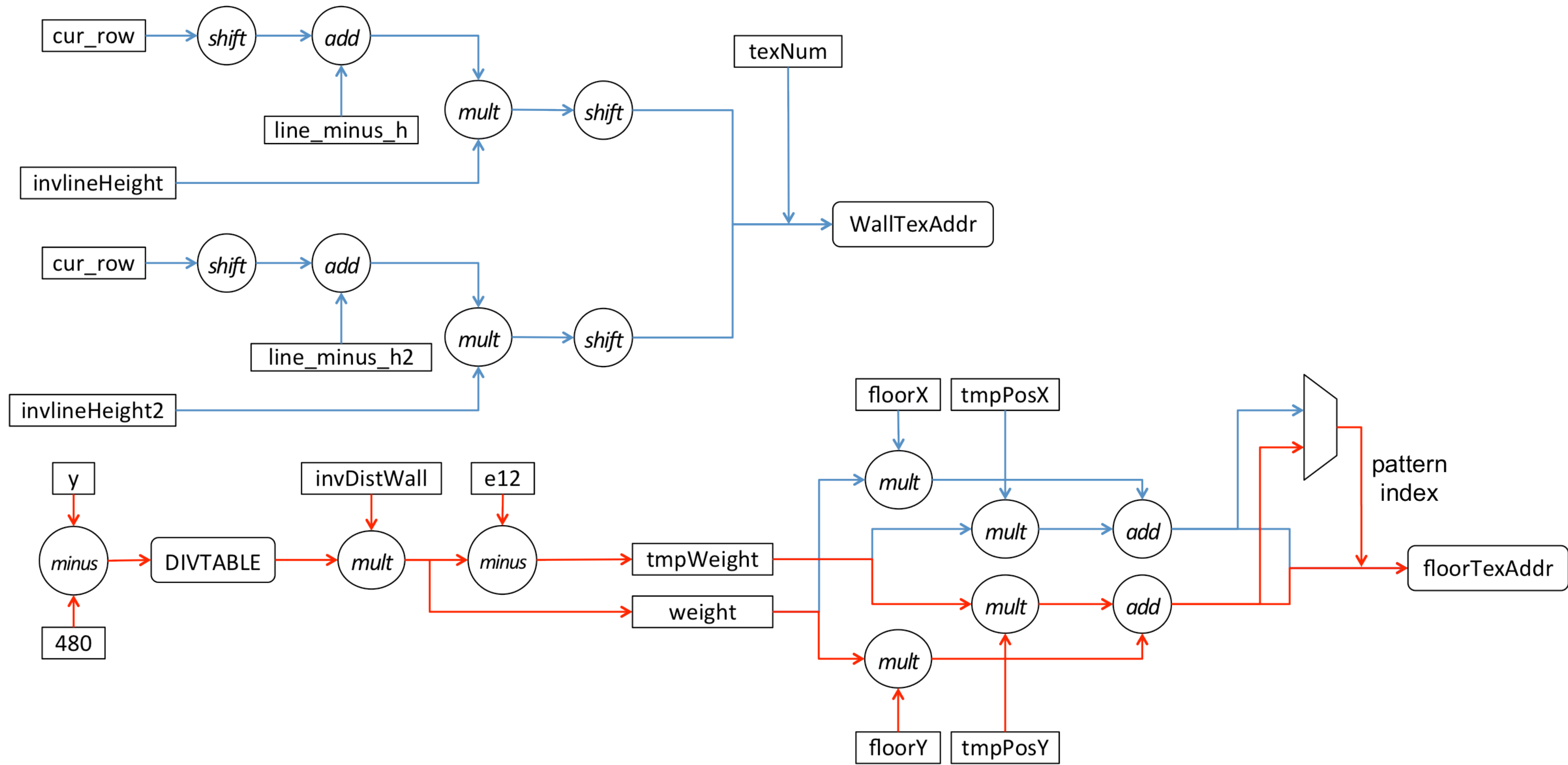
VGA Rastering



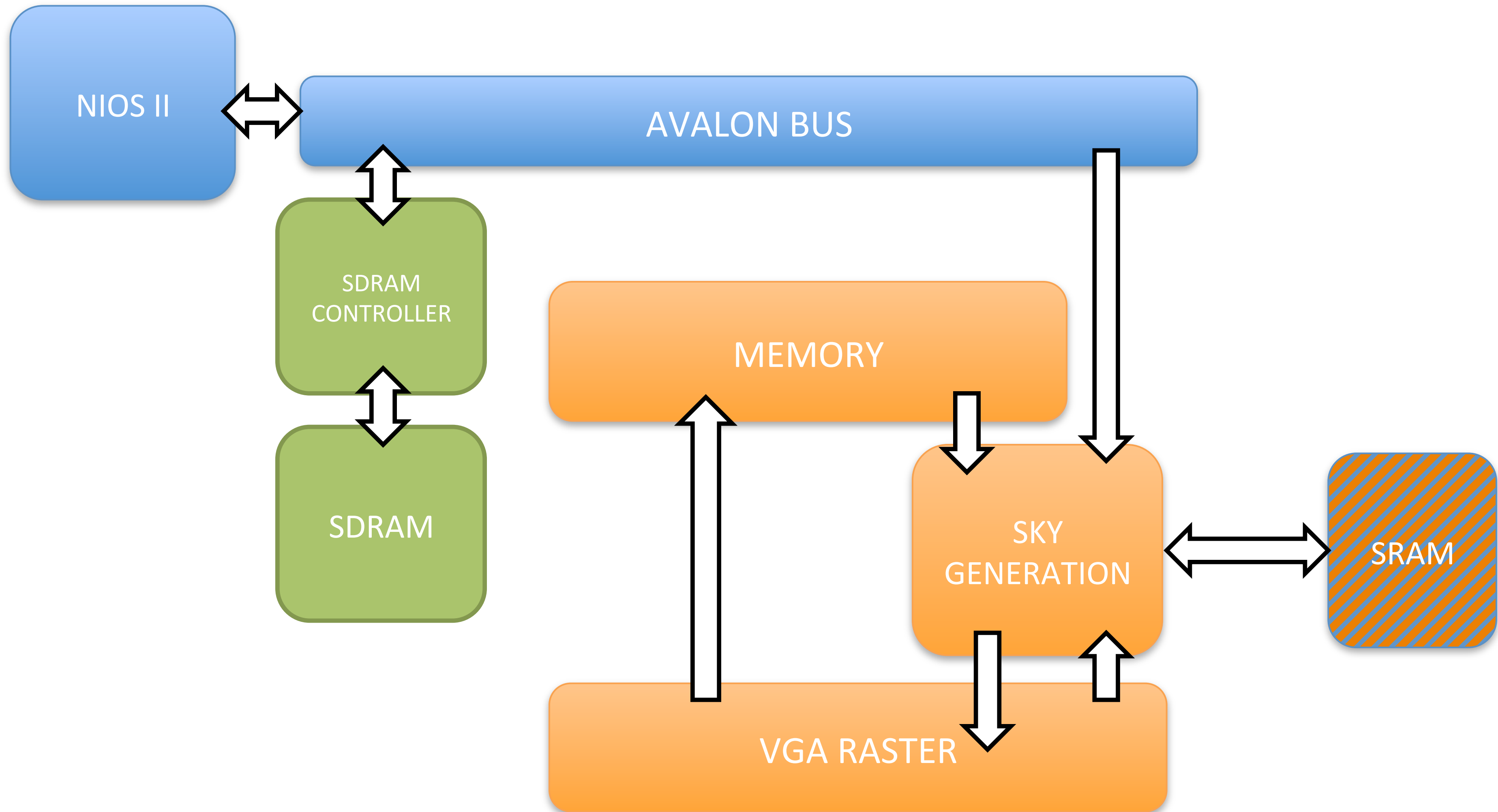
Texture Generation



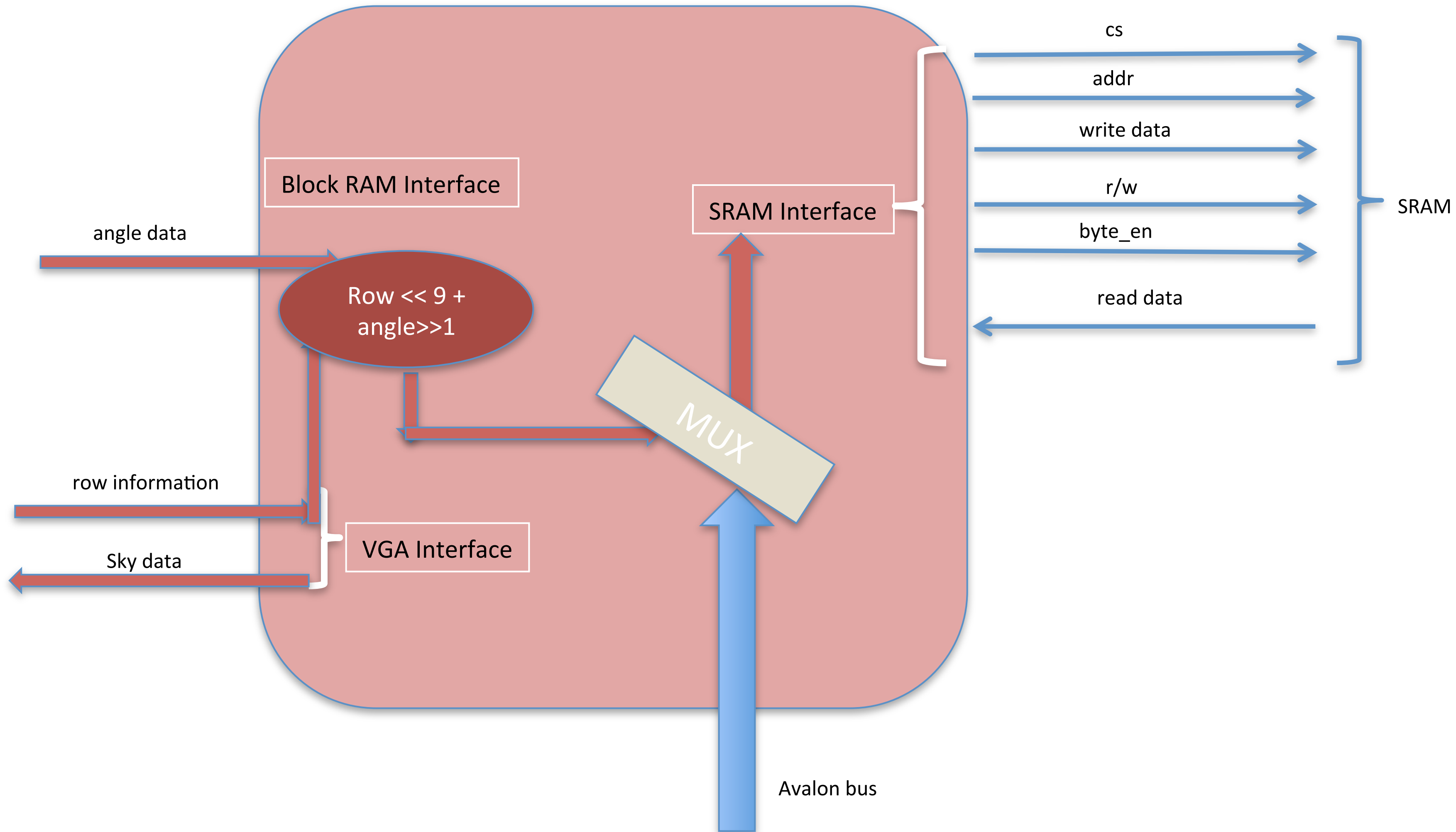
Critical Timing



Sky generator block diagram

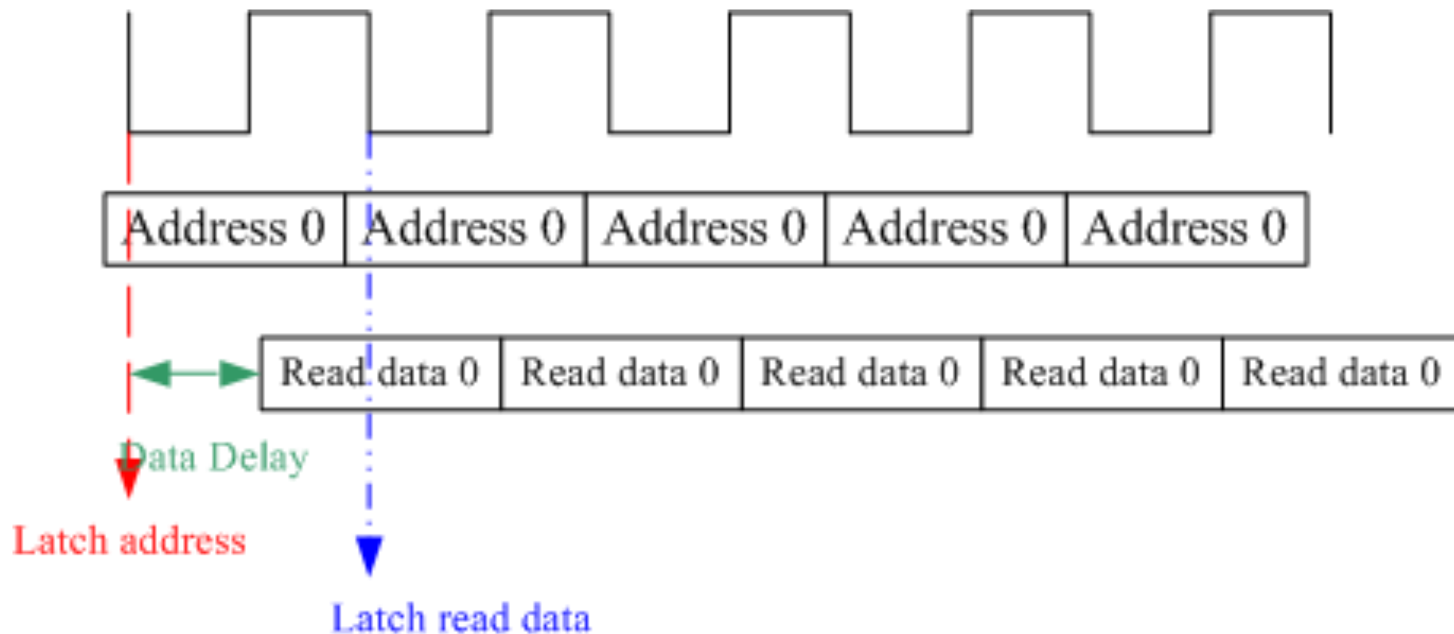


Sky Generation

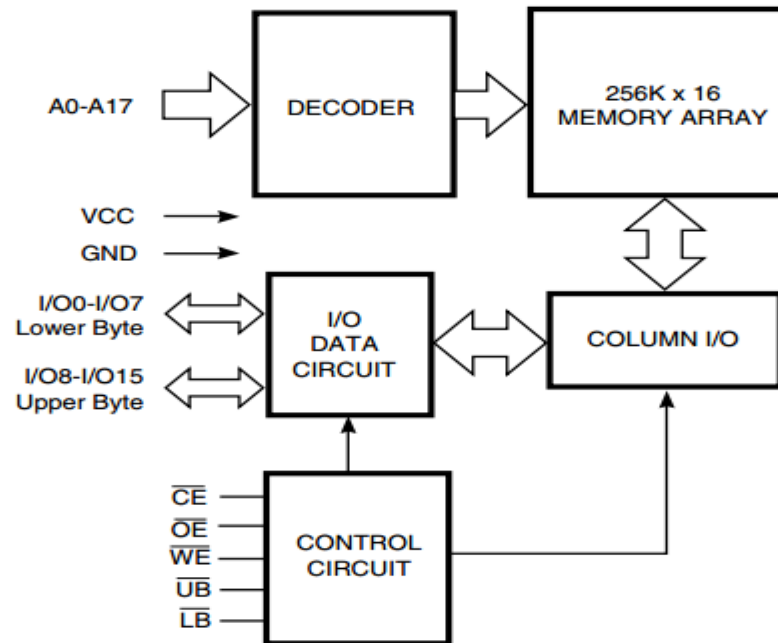


Sky Generation Timing

- There is no clock in SRAM, it is controlled by address
- Maximum Data Delay is 15 ns, whether the clock is 40ns or 20ns



Asynchronous SRAM interface

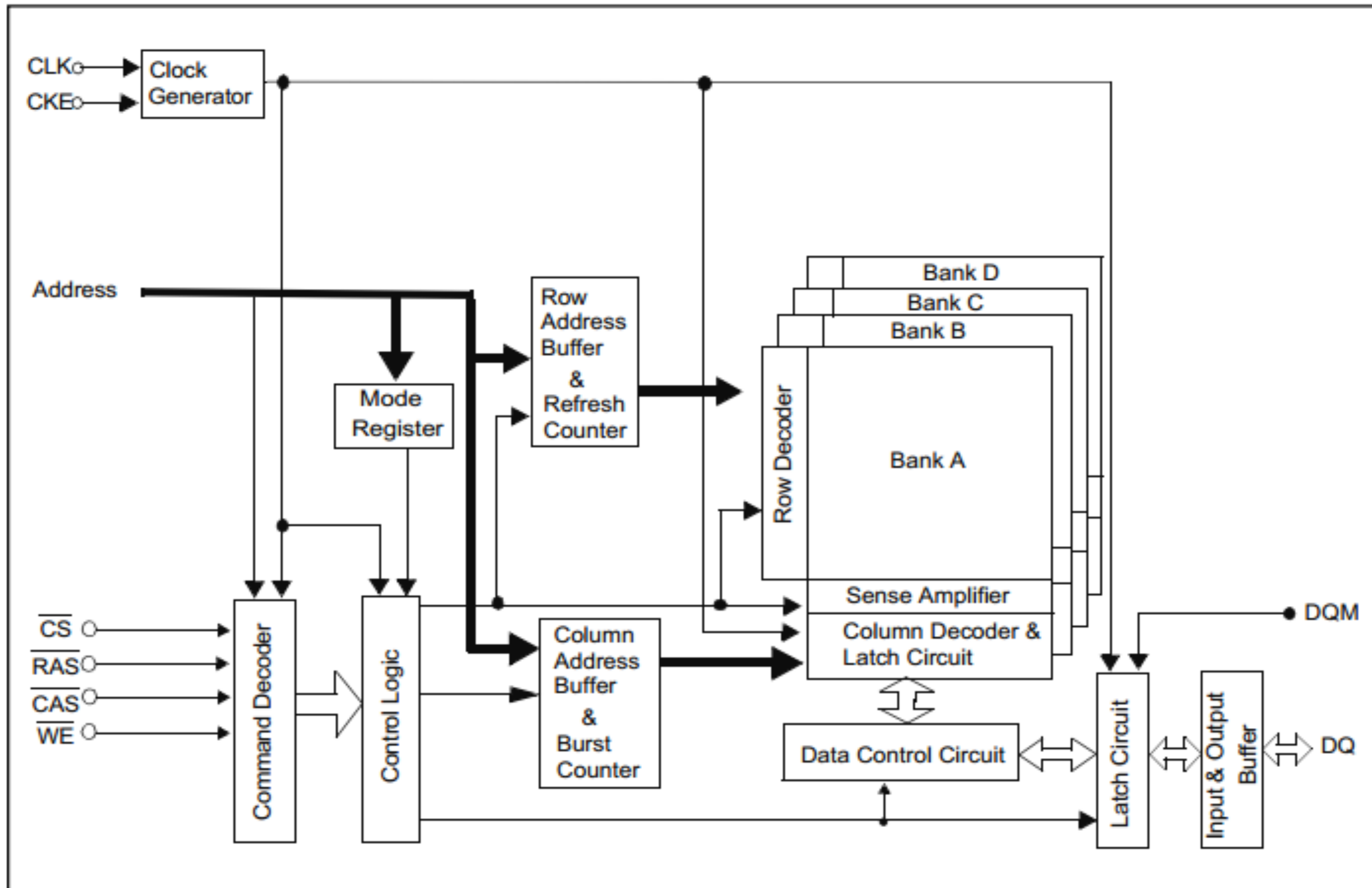


IS61LV25616 *ISSI®*

TRUTH TABLE

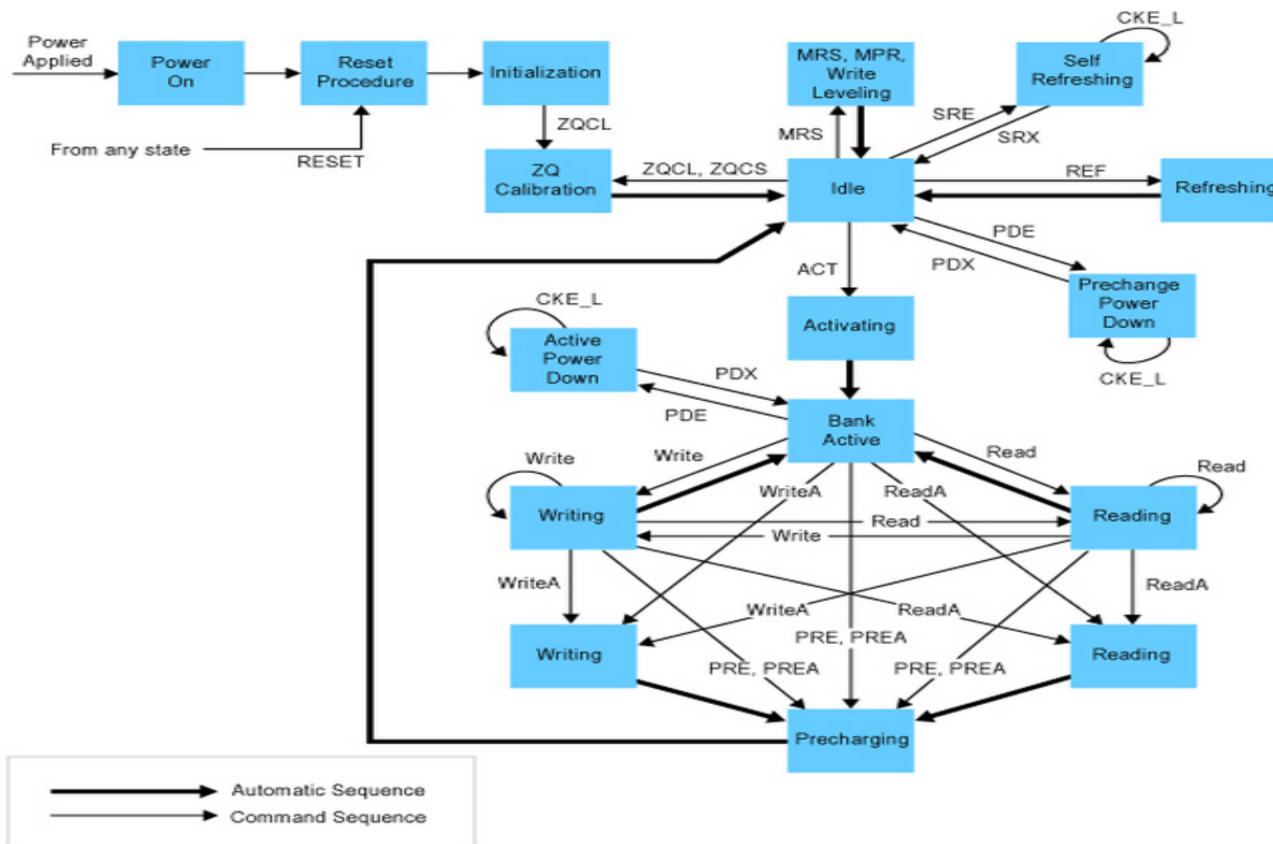
Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	$\overline{LB}$	$\overline{UB}$	I/O PIN I/O0-I/O7	I/O PIN I/O8-I/O15	Vcc Current
Not Selected	X	H	X	X	X	High-Z	High-Z	I_{SB1}, I_{SB2}
Output Disabled	H	L	H	X	X	High-Z	High-Z	I_{CC}
	X	L	X	H	H	High-Z	High-Z	
Read	H	L	L	L	H	D_{OUT}	High-Z	I_{CC}
	H	L	L	H	L	High-Z	D_{OUT}	
	H	L	L	L	L	D_{OUT}	D_{OUT}	
Write	L	L	X	L	H	D_{IN}	High-Z	I_{CC}
	L	L	X	H	L	High-Z	D_{IN}	
	L	L	X	L	L	D_{IN}	D_{IN}	

SDRAM architecture



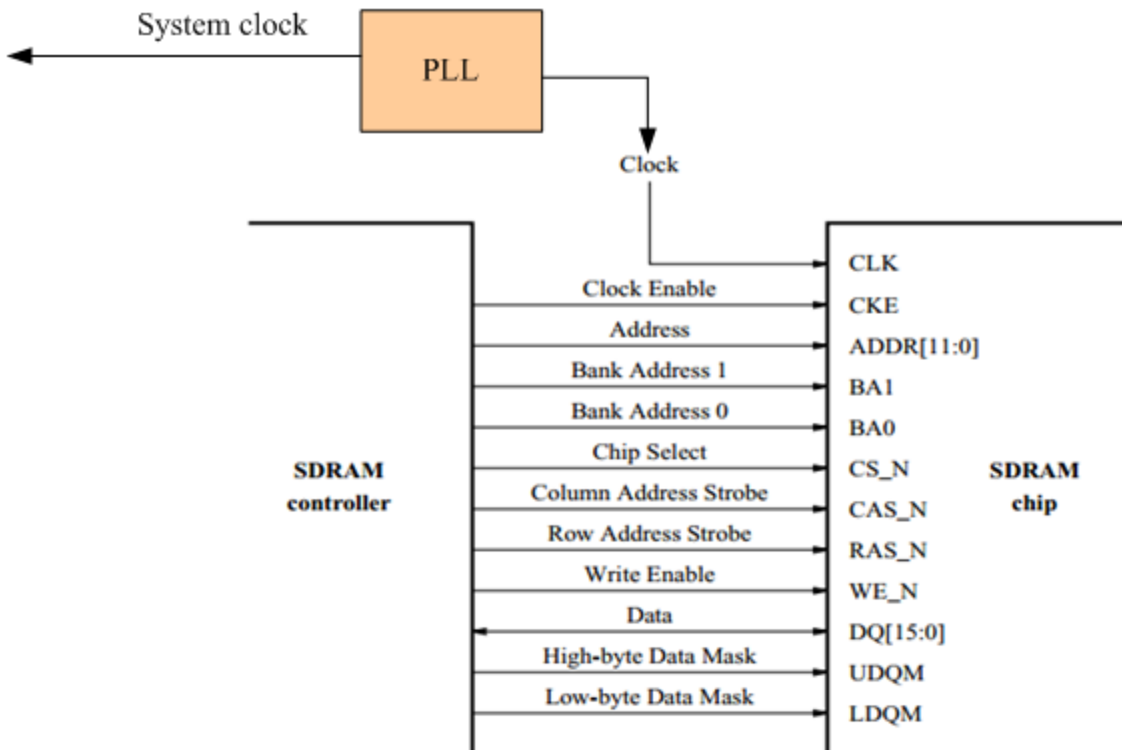
SDRAM controller diagram

- The SOPC generated controller would transmit data to DRAM according FSM

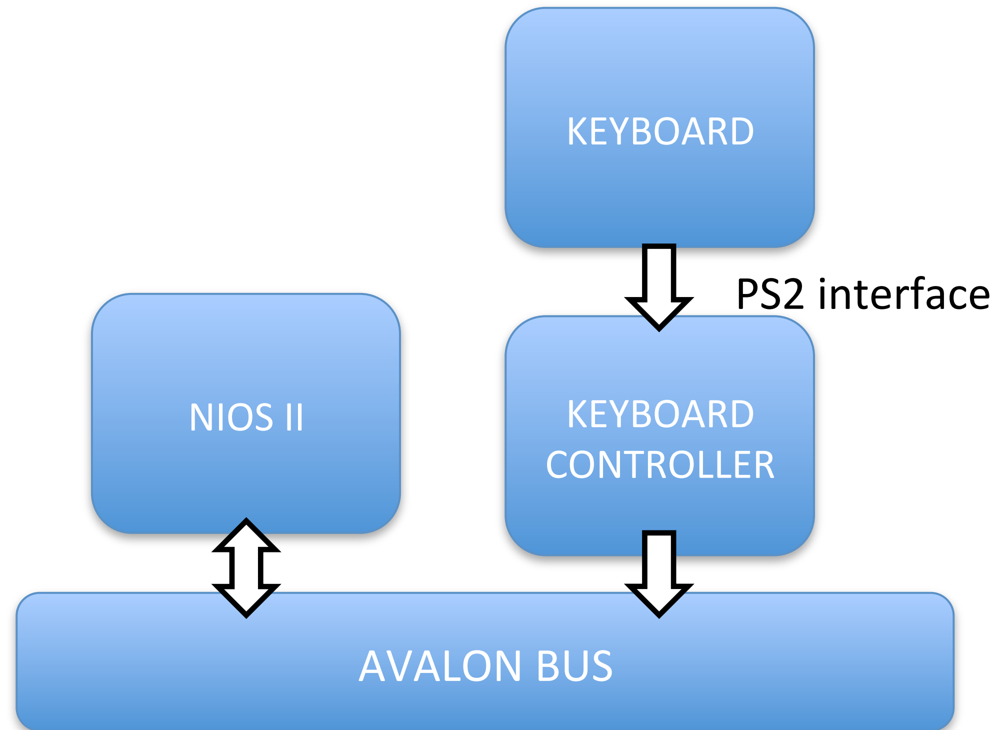


SDRAM interface

- After SOPC generating SDRAM controller, we integrated the controller according this diagram
- We also need to generate PLL for proper operation
  - To generate a DRAM clock ahead by 3 ns

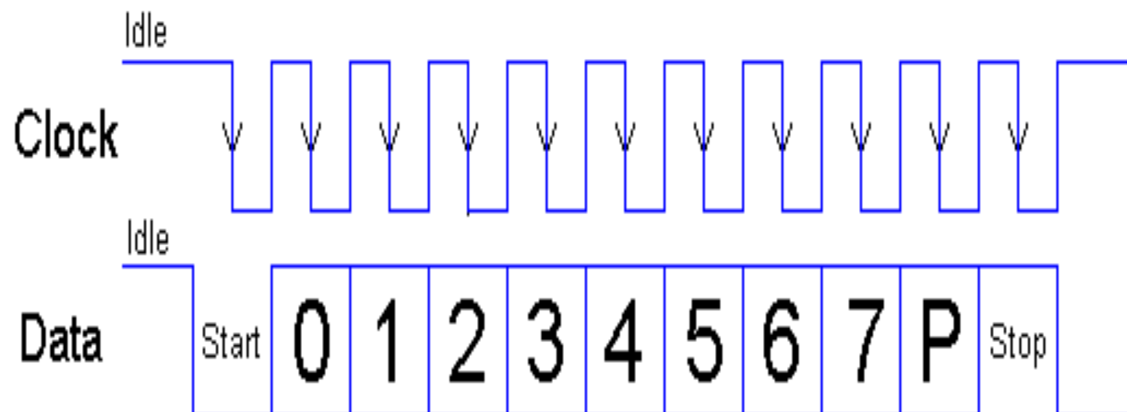


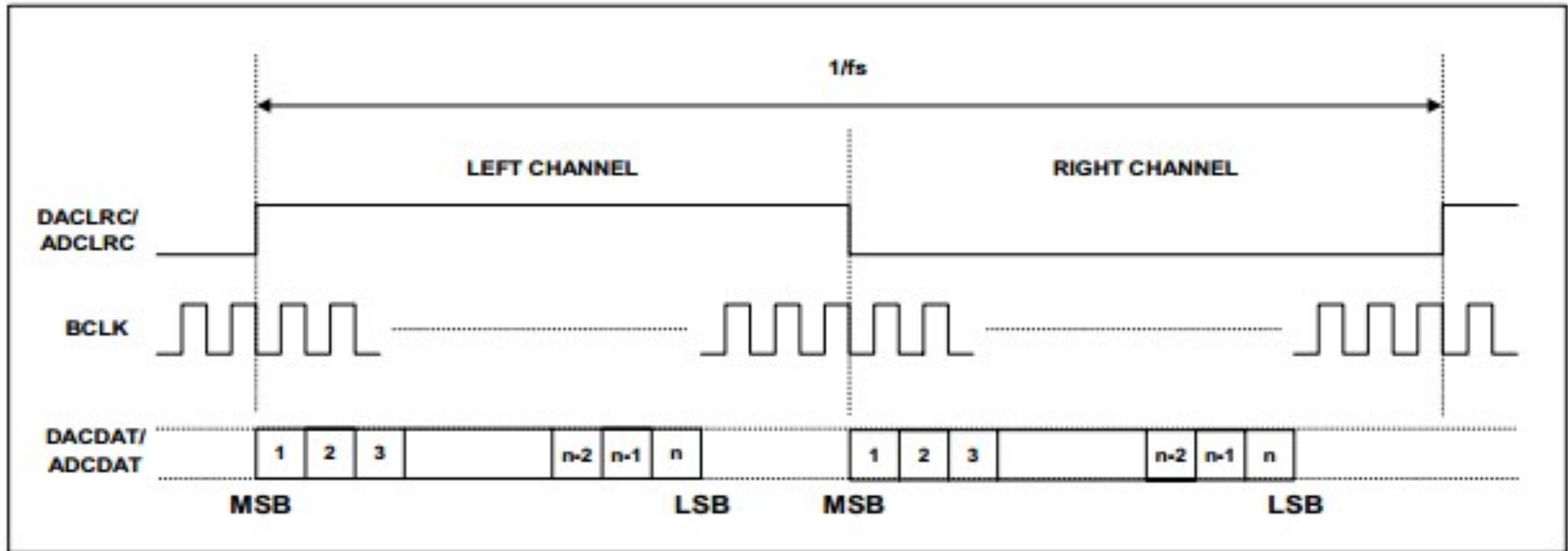
Keyboard



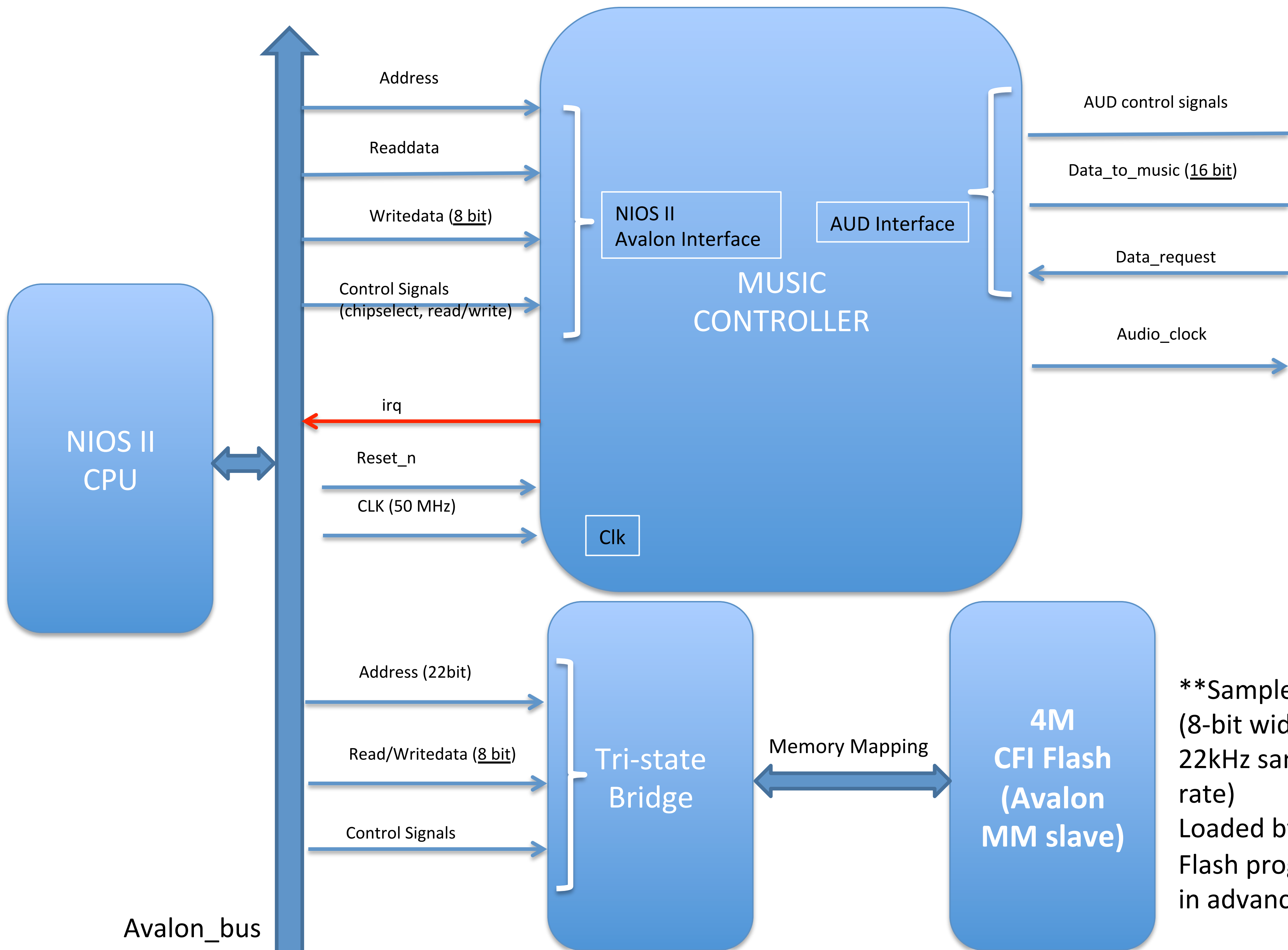
PS2 interface

- Keyboard controller from Lab3
- Serial interface with CK, DAT
- Data would be put to register for Nios polling

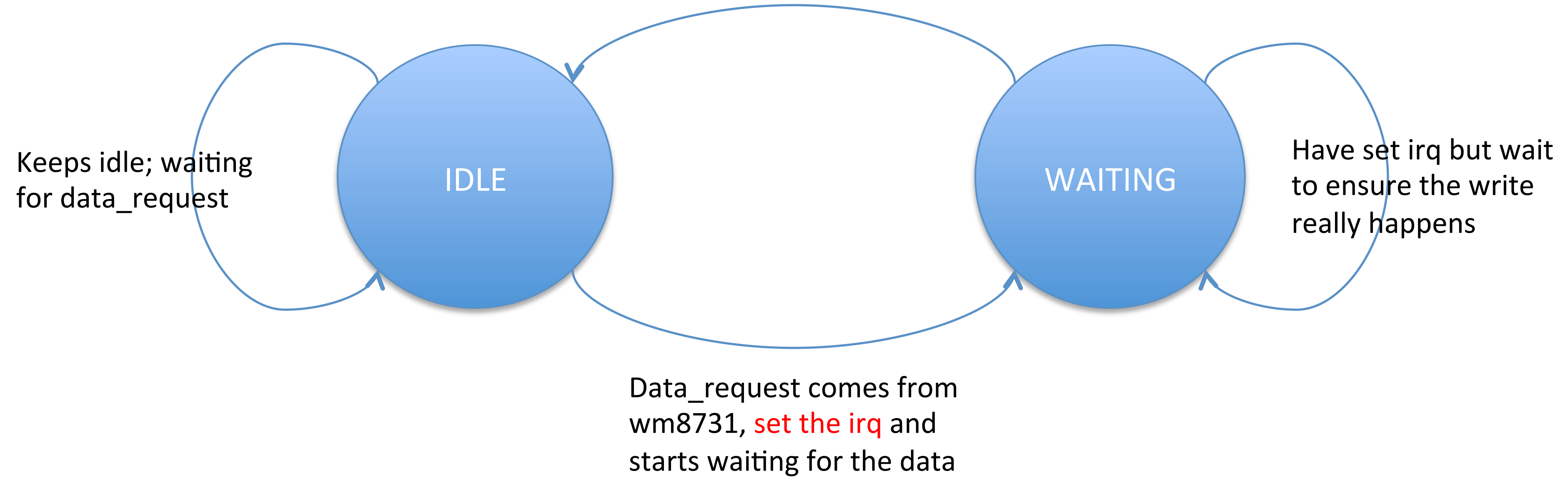




Data_request


Address
Readdata
Writedata (8 bit)
Control Signals
(chipselect, read/write)
irq
Reset_n
CLK (50 MHz)
NIOS II
Avalon Interface
AUD Interface
MUSIC
CONTROLLER
Clk
AUD control signals
Data_to_music (16 bit)
Data_request
Audio_clock
NIOS II
CPU
Address (22bit)
Read/Writedata (8 bit)
Control Signals
Tri-state
Bridge
Memory Mapping
4M
CFI Flash
(Avalon
MM slave)
Avalon_bus

As the sound controller can see:



Other issues: Why Flash? How about Buffering?
Choose the Sampling frequency

Key Difficulties

- Raycasting Speed
- SRAM Clock Domain
- SDRAM Clock Domain
- M4K Clock Domain
- Audio Interrupts
- Memory Division Screen Glitches
- Debugging with Unreliable Peripheral

Lessons Learned

- Pay attention to Clock Domains (Eddy)
- Hardware Debugging is as valuable as software debugging (Minyun and Wei)
- ModelSim is invaluable (Yiming)
- No Printf's in interrupts (Minyun)
- Persistance is key; You can accomplish anything if you have the patience to learn it. (Alden)