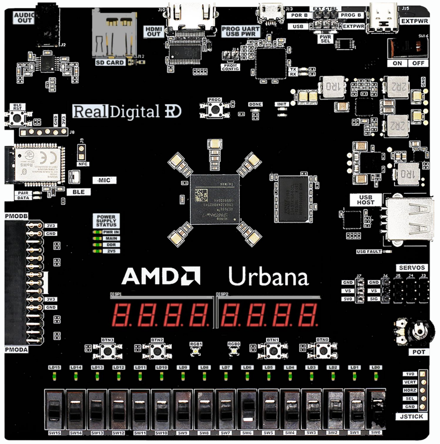
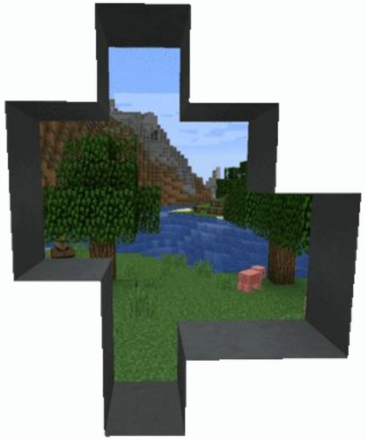
MINECRAFT FPGA EDITION





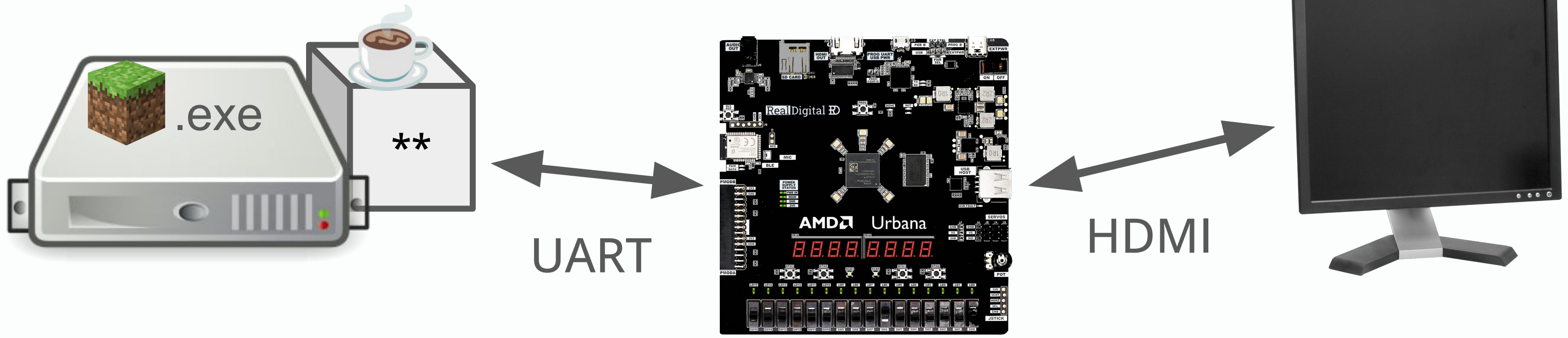
6.2050 Project Proposal Presentation

Yohan Guyomard, Win Win Tjong

November 5, 2024

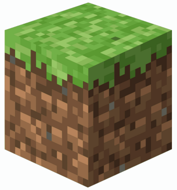
Project Overview

- FPGA client that connects to Minecraft worlds
  - Interact with the same world as other players*
- Renderer uses a ray-tracing approach, ideally with several bounces for realistic effects like shadows, reflections and refraction.



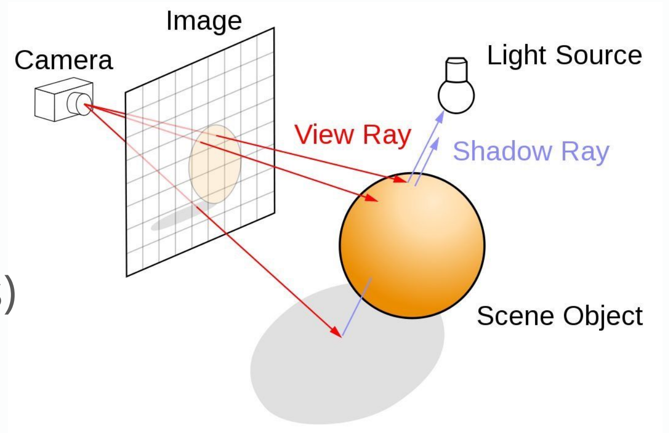
**custom plug-in is used to replace Minecraft's protocol (and TCP) with UART

Terminology & Preliminary Stuff



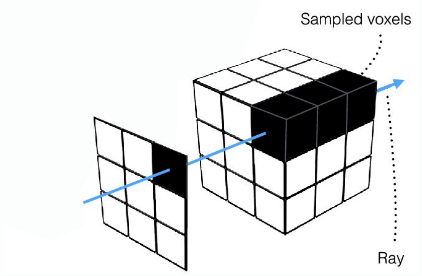
voxel (or block)

Ray-tracing



chunk

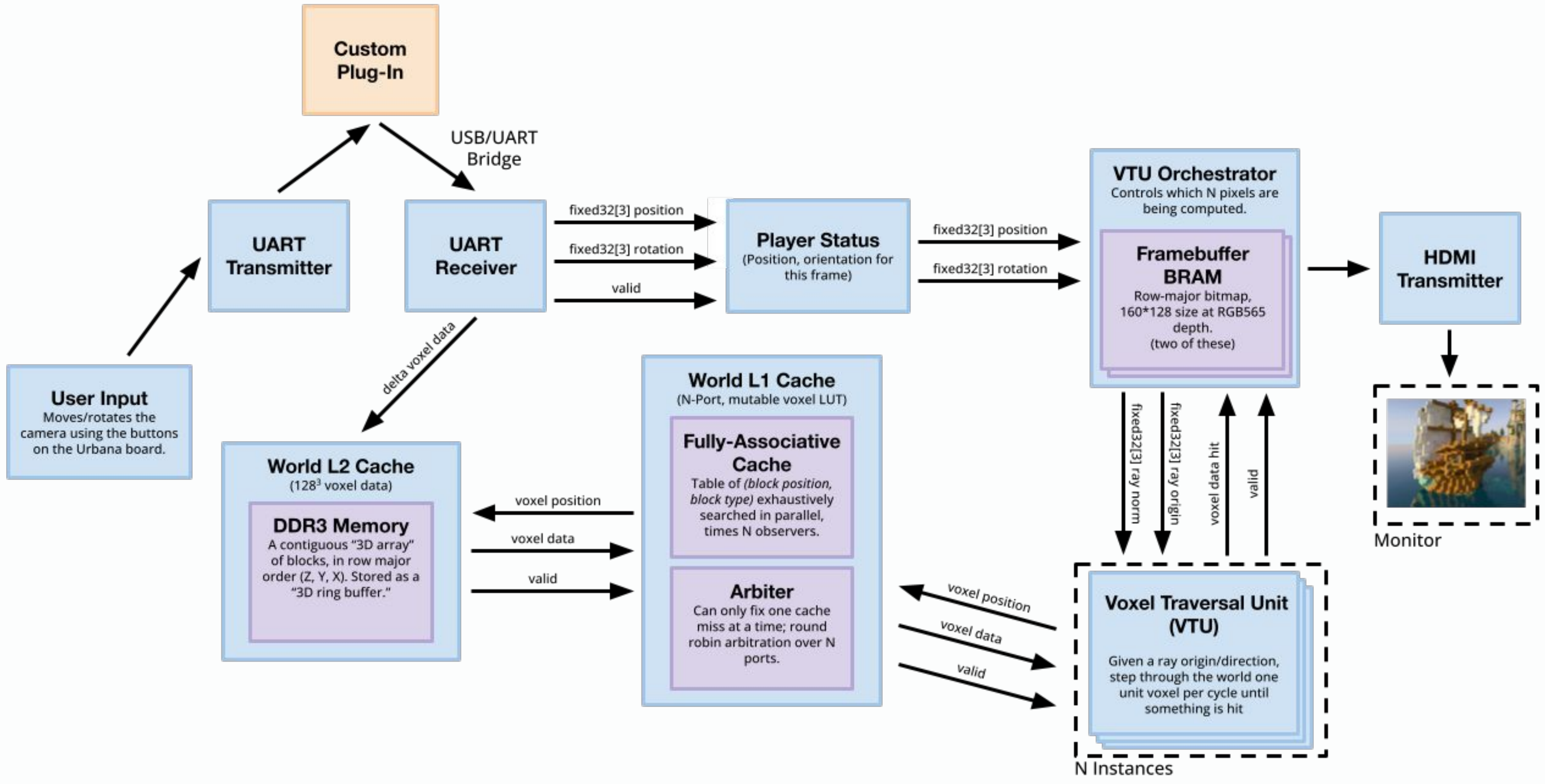
Segment of world stored on the FPGA (128^3 voxels)



voxel traversal

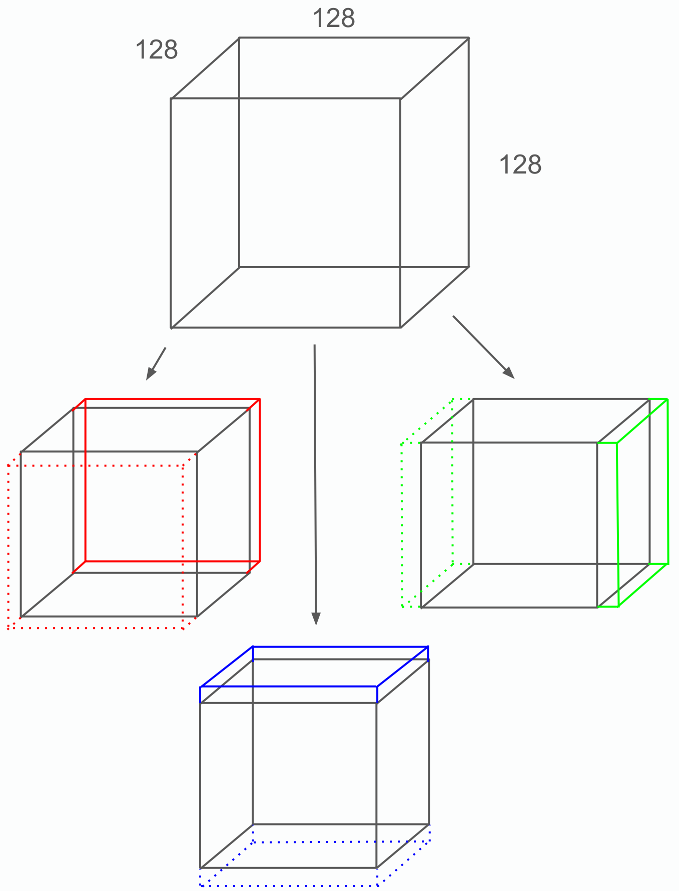
Steps one voxel at a time (air=continue; block=halt) (Amanatides & Woo)

Block Diagram


Custom Plug-In
USB/UART Bridge
UART Transmitter
UART Receiver
fixed32[3] position
fixed32[3] rotation
valid
Player Status
(Position, orientation for this frame)
fixed32[3] position
fixed32[3] rotation
VTU Orchestrator
Controls which N pixels are being computed.
Framebuffer BRAM
Row-major bitmap, 160*128 size at RGB565 depth.
(two of these)
HDMI Transmitter
Monitor
User Input
Moves/rotates the camera using the buttons on the Urbana board.
delta voxel data
World L2 Cache
(128³ voxel data)
DDR3 Memory
A contiguous "3D array" of blocks, in row major order (Z, Y, X). Stored as a "3D ring buffer."
voxel position
voxel data
valid
World L1 Cache
(N-Port, mutable voxel LUT)
Fully-Associative Cache
Table of (block position, block type) exhaustively searched in parallel, times N observers.
Arbiter
Can only fix one cache miss at a time; round robin arbitration over N ports.
voxel position
voxel data
valid
fixed32[3] ray norm
fixed32[3] ray origin
voxel data hit
valid
Voxel Traversal Unit (VTU)
Given a ray origin/direction, step through the world one unit voxel per cycle until something is hit
N Instances

L2 Cache

- Choice of data structure to store voxel data.
  - Update happens when player status changes
    - Direction depends on whichever direction the player moves towards
  - Stored in DDR3
  - Each voxel is represented with 5 bits
  - Total size is $128^3 * 5 \approx 10$Mb

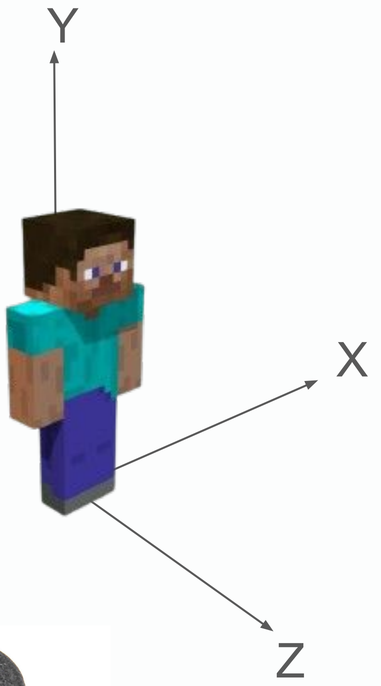


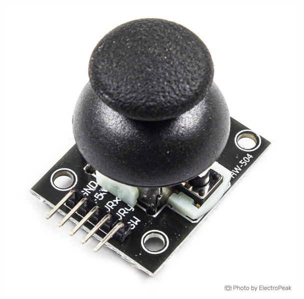
UART

- Communication protocol between the game’s server to the FPGA.
  - The asynchronization between transmission and receiving allows for flexibility when the two are happening at different rates.
  - Straightforward to use and allows for easier debugging.
  - Receives data from the game plug-in and stores it directly into the L2 cache.
    - The plug-in used for this project should be a simple implementation and should not hinder other parts of the project.
  - To run a smooth 30 fps, rate of transfer exceeds $128^2 * 5 * 30 \approx 2$ Mb.
    - The UART module in the Urbana board supports a transfer rate up to 2Mb, so it should be fine.

Player Input & Status

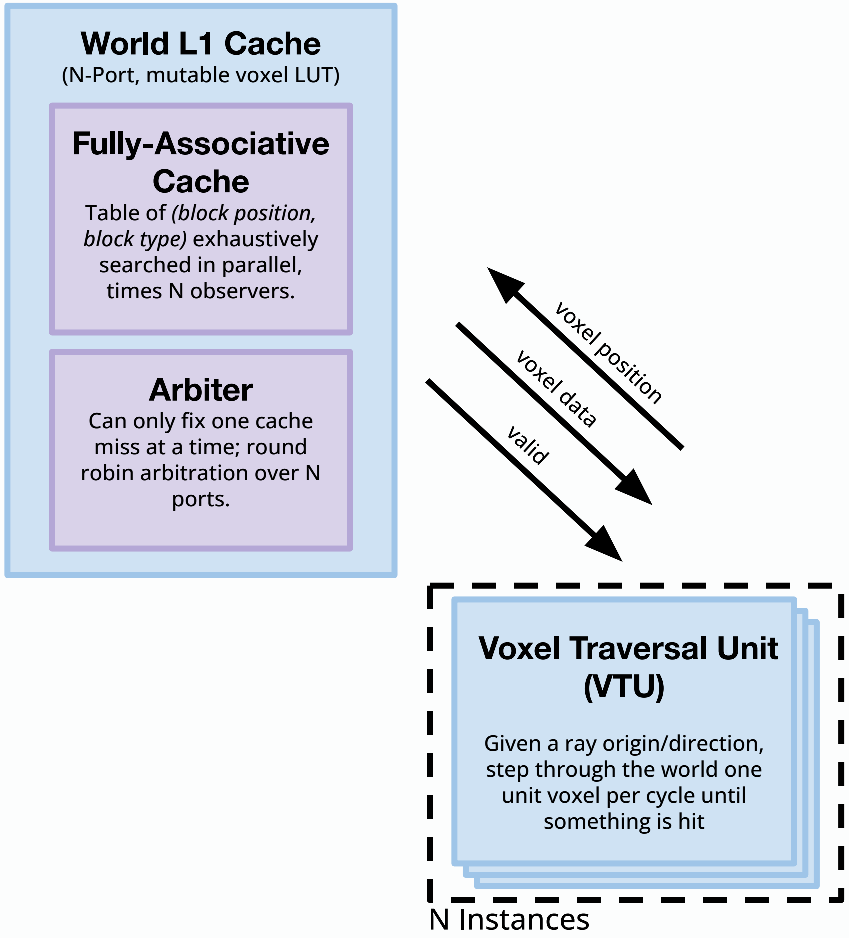
- The input of a joystick connected to the board are sent to the game via UART transmitter, which will update player status in-game.
  - This then is transmitted into the UART receiver and into the L2 cache
  - At the same time, this will update player's position, which will be accessed by the frame buffer.





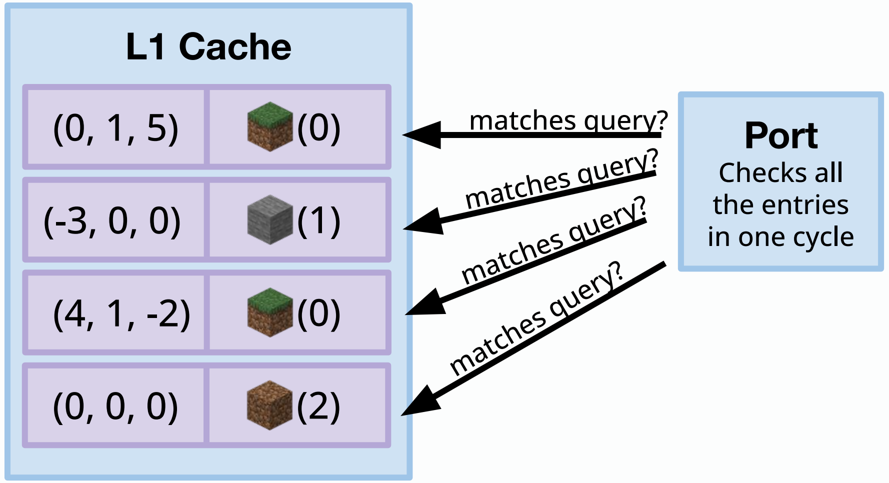
Memory Bottleneck

- We can compute the path of the ray really fast and in-parallel
- **But** *each* iteration of *each* VTU needs 1 memory access
  - Voxel is air? continue.
  - Voxel is not? halt.
- L2 cache has just **one** reading port, so really it's just 1 working VTU with deadtime for N - 1 VTUs


World L1 Cache
(N-Port, mutable voxel LUT)
Fully-Associative Cache
Table of *(block position, block type)* exhaustively searched in parallel, times N observers.
Arbiter
Can only fix one cache miss at a time; round robin arbitration over N ports.
voxel position
voxel data
valid
Voxel Traversal Unit (VTU)
Given a ray origin/direction, step through the world one unit voxel per cycle until something is hit
N Instances

World L1 Cache

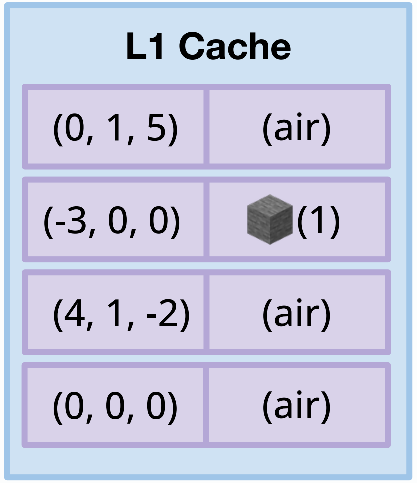
- Likely, all rays will query the same set of voxels*
- This data structure has **N ports**, each simultaneous single-cycle access if cache-hit!
- On cache miss, it's potentially answering many VTUs' requests all at once
  - If not, arbiter takes over
  - Continues to serve the non-blocked ports!



*educated speculation

World L1 Cache (Problem)

- Can’t make this very large*
  - For N ports and M entries, that’s N*M comparators
  - There’s a reason K-way set associative caches exist, but these don’t really apply here
- The bulk of the cache will just be “air” voxels*
  - Wasteful; “air” or ”not air” is 1 bit of information, not an entire voxel ID!



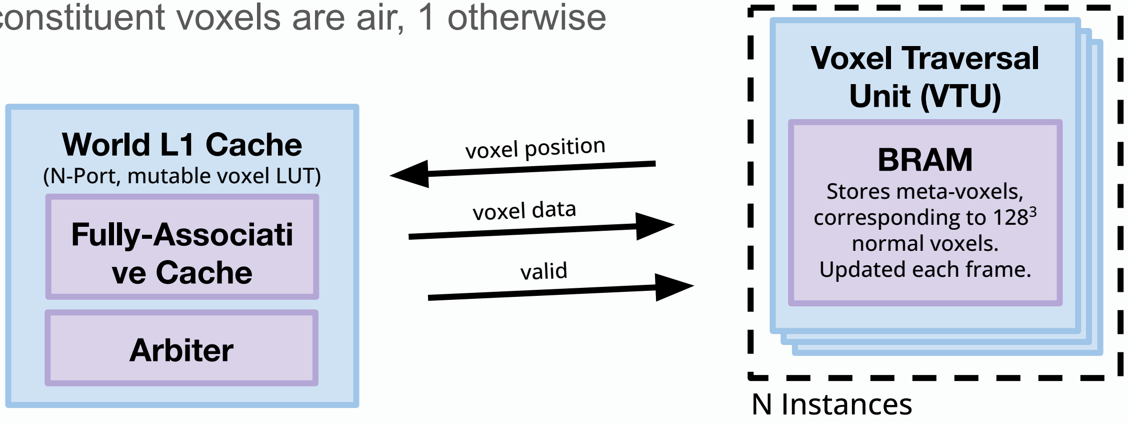
L1 Cache

(0, 1, 5)	(air)
(-3, 0, 0)	(1)
(4, 1, -2)	(air)
(0, 0, 0)	(air)

*educated speculation

Sparseness Cache

- Each VTU has its own cache of the entire chunk!
  - 1 BRAM per VTU
- Only care about air/not air, so 128^3 bits of data
  - Actually that doesn't fit… ~2 Mbits > 32 Kbits
- So, create "meta-voxels" and store that!
  - 1 meta-voxel = 4^3 voxels
  - Occupies 1 bit: 0 if all its constituent voxels are air, 1 otherwise


World L1 Cache
(N-Port, mutable voxel LUT)
Fully-Associative Cache
Arbiter
voxel position
voxel data
valid
Voxel Traversal Unit (VTU)
BRAM
Stores meta-voxels, corresponding to 128^3 normal voxels. Updated each frame.
N Instances

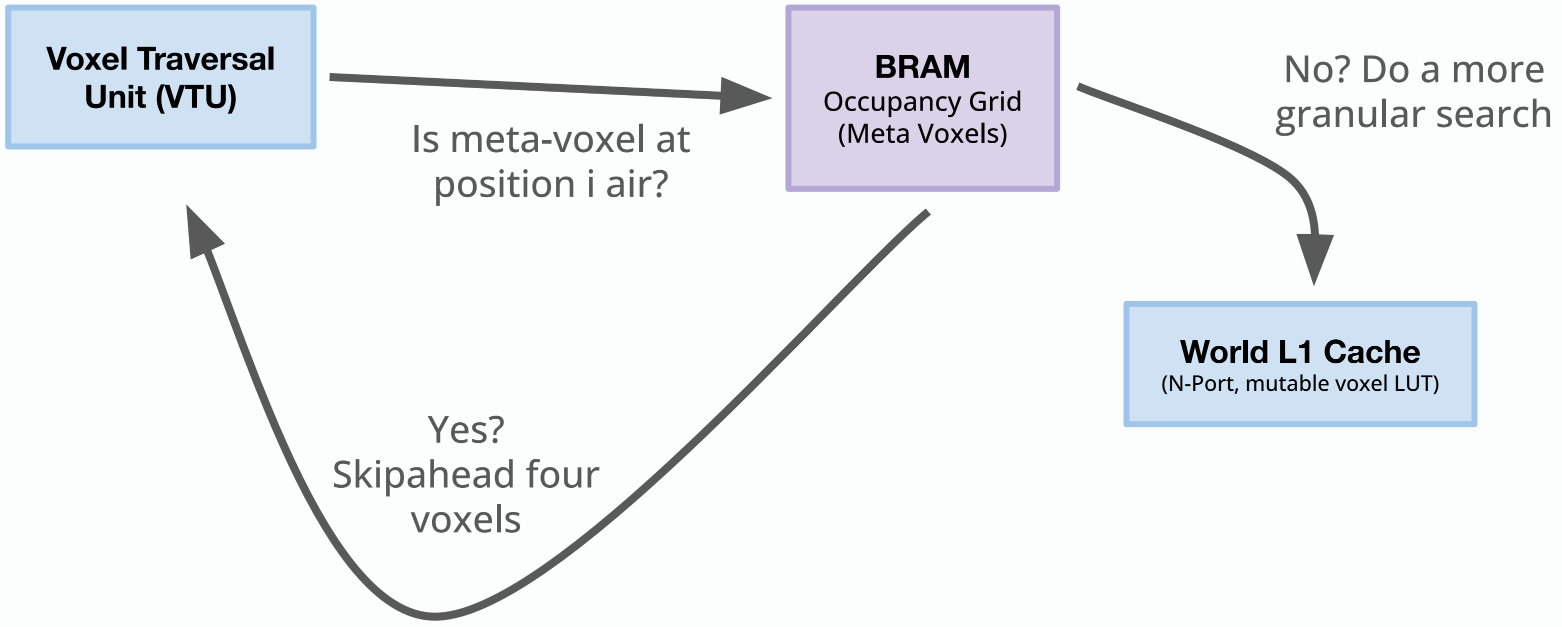
VTU Orchestrator

- One more memory thing: how do N VTUs draw to the same framebuffer?
  - The orchestrator will block a VTU until its pixel can be pushed to the framebuffer BRAM
  - Shouldn’t create much deadtime, the VTU are all asynchronous anyways

Project Checkoff List

- **The Commitment**
  - Implement a basic renderer using ray-tracing
  - Stream a static world from a Minecraft server onto the FPGA
- **The Goal**
  - Sufficiently optimize our renderer for higher framerates, (more) light bounces, textures; overall something more pleasing to the eyes.
  - Continuously update the FPGAs copy of the voxel world with incoming data; from a user’s point of view, the world should seem infinite.
- **The Stretch**
  - Higher resolution graphics (going beyond BRAM) and/or higher framerates
  - Rendering entities (players, mobs, etc.)

Sparseness Cache


Voxel Traversal Unit (VTU)
Is meta-voxel at position i air?
BRAM
Occupancy Grid
(Meta Voxels)
No? Do a more granular search
World L1 Cache
(N-Port, mutable voxel LUT)
Yes?
Skipahead four voxels

Timeline

	Computer Interface (Win Win)	**Rendering** (Yohan)
Week 0 (Nov. 6)	Implement L2 cache (ring)	Implement L1 cache
Week 1 (Nov. 11)	Integrate into L1 cache	Implement VTU
Week 2 (Nov. 18)	UART transceiver (protocol)	Add occupancy optimization
Week 3 (Nov. 25)	User input and player status	Create VTU orchestrator
Week 4 (Dec. 2)	Final integration	Final integration
Week 5 (Dec. 9)	Finalize report	Finalize report

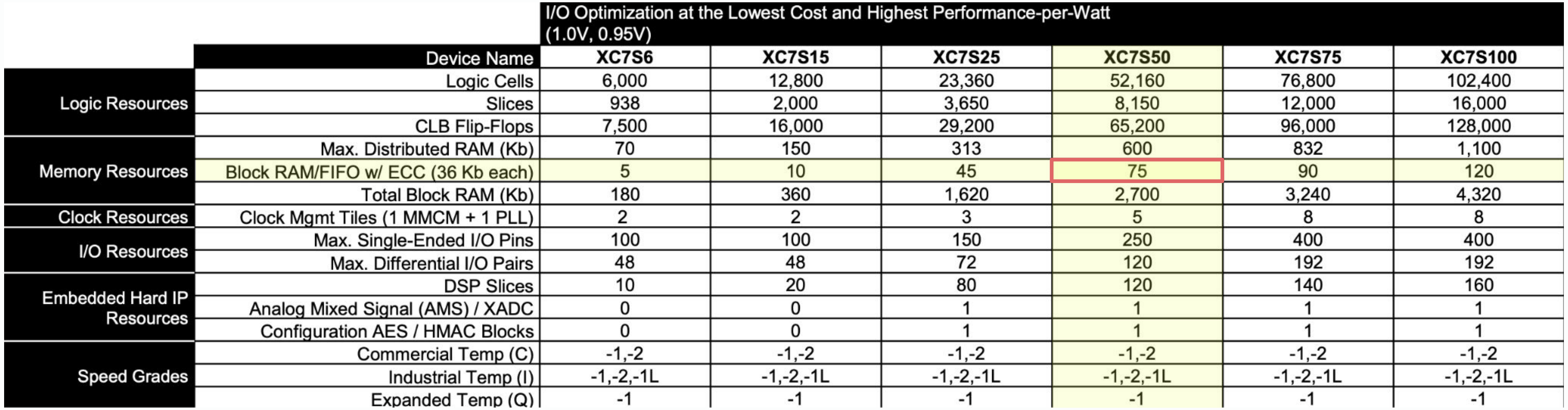
Thank you! Questions?

Update: Plugin has been implemented

- We are able to query blocks in a Minecraft world
- From the same program, we have serial communication (currently tested with a microcontroller)

$\leftarrow$ Joe, right now

whaterweworkinwit



		I/O Optimization at the Lowest Cost and Highest Performance-per-Watt (1.0V, 0.95V)					
	Device Name	**XC7S6**	**XC7S15**	**XC7S25**	**XC7S50**	**XC7S75**	**XC7S100**
Logic Resources	Logic Cells	6,000	12,800	23,360	52,160	76,800	102,400
	Slices	938	2,000	3,650	8,150	12,000	16,000
	CLB Flip-Flops	7,500	16,000	29,200	65,200	96,000	128,000
Memory Resources	Max. Distributed RAM (Kb)	70	150	313	600	832	1,100
	Block RAM/FIFO w/ ECC (36 Kb each)	5	10	45	75	90	120
	Total Block RAM (Kb)	180	360	1,620	2,700	3,240	4,320
Clock Resources	Clock Mgmt Tiles (1 MMCM + 1 PLL)	2	2	3	5	8	8
I/O Resources	Max. Single-Ended I/O Pins	100	100	150	250	400	400
	Max. Differential I/O Pairs	48	48	72	120	192	192
Embedded Hard IP Resources	DSP Slices	10	20	80	120	140	160
	Analog Mixed Signal (AMS) / XADC	0	0	1	1	1	1
	Configuration AES / HMAC Blocks	0	0	1	1	1	1
Speed Grades	Commercial Temp (C)	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2
	Industrial Temp (I)	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L
	Expanded Temp (Q)	-1	-1	-1	-1	-1	-1

yohan's notes, delete later

- based on feedback from the block diagram and some results from our initial implementation, a large-enough, single-cycle useful L1 cache may be unrealistic.
- What if each VTU had its own copy of the voxel world?
  - Can't each have their own DDR RAM
  - Can't fit all the data we need in BRAM
  - But, for 99% of memory accesses we only care about "voxel exists or is it air?"
    - Reduce to 1-bit representation of the world
  - 128^3 = 2 Mbits is still too large
  - But, sample every four voxel (i.e. 1 if any of those 4^3 voxels are not air, 0 otherwise), one BRAM is *just* enough (32 Kbits)
- There are other, more compact ways to represent sparness (i.e. octree) but less trivial to implement
  - Maybe a stretch goal

basically, less memory accesses = less contention = less bad

yohan’s notes, delete later

can we rethink the L1 cache?

- now that each VTU has its own “approximate” copy of the world, is it better to change the L1 cache to be e.g. BRAM with more entries but slower access?
- We would need an arbiter, one VTU reads L1 cache at a time

basically, less memory accesses = less contention = less bad

yohan's notes, delete later

→ **can we race the beam, is it possible?**

- Without cheating, definitely not. Hell no. 720p @ 60FPS means 74.25 MHz clock and we can't trace an entire ray in one clock cycle
- *But*, our ray tracing algorithm is upper bounded
  - max(# cycles) = (# bounces) * (max travel distance)
  - (max travel distance) = 221 for 128^3 chunk
- so when we're displaying pixel i, we need to start rendering pixel i + 221, and i + 220, i + 219, etc. had started in prior cycles.
- so we need ~220 VTUs… not gonna happen

No, pretty much not possible

yohan's notes, delete later

→ **with a framebuffer, how large can it get?**

- (assuming storage, to/fro is a non-issue)
- 221 max ray distance (VTU steps one voxel at a time)
- assuming single-cycle step (not realistic):
  - 100MHz / 221 = ~452,000 pixels per VTU per second
  - Comfortably draw 480p@30 with 20 VTUs, 720p with 60 VTUs, etc.
- more realistically, two-cycle step average over 221/4 = 55 meta-voxels
  - Thrown in there are "meta-voxel misses" (thinks it hit solid, but had to go check that it was air) which can cost a lot of cycles if it's an L1 cache-miss. No idea how to estimate this, let's say it quadruples the worst case average.
  - 100MHz / (55 * 4) = ~454,000 pixels per VTU per second
- if we really limit the size of the frame buffer (e.g. a cute 128*160), we can *really* go crazy with light bounces
  - At this size, just one VTU is enough for everything (single-ray)

720p is possible, but choosing lower resolutions gives more room for creativity & artistic effects!!