



Bitfiltrator



Dan Ivanovich, Izzy Friedfeld-Gebaide,
Noam Hirschorn



Introduction

What is a Bitstream?

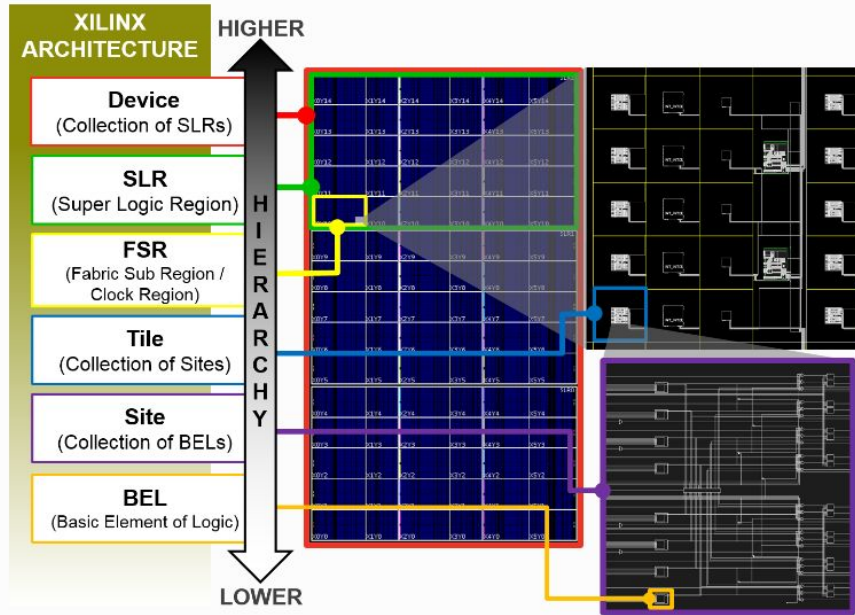
- Configure FPGA hardware to implement code
 - I.e. Configure the data stored in LUTs
- Why reverse engineer?
 - For Ultrascale, need Vivado to generate Bitstream
 - Can bypass Vivado
 - Vivado runs time consuming safety checks
 - Can edit bitstream during operation
 - Quicker and Easier for Fault injection testing
 - Can modify as a form of patching
 - Can check if bistream has been modified
 - Can move sensitive LUTs to hide from attacker

Why is this Difficult?

- FPGAs only documented on high level
 - Structure covered in next session
- Other papers only go in so much depth
- Also might only deal with smaller devices
 - Larger devices have a larger hierarchy than addressing as been documented for

Device Structure

Levels of Ultrascale* Xilinx FPGA



Levels of architectural hierarchy in Xilinx FPGAs.

* Also applies to
Ultrascale+

Credit: Rapidwright.io

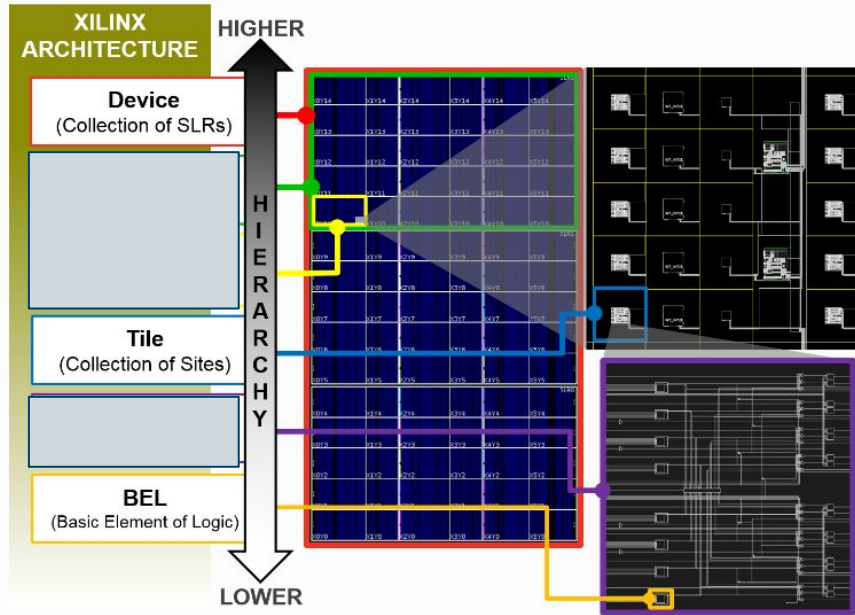
Note on Overall Design

- Regular design
 - Easy to manufacture at scale
 - Chip design needs to be cheap to mass produce
 - Easy to integrate different components
 - Can abstract away different resources as the same at high level
 - Easy to design
 - Can split up design to various levels of hierarchy in a modular fashion
 - Easy to error check/prevent bugs
 - Many operations need to be enabled, less chance for design errors if simple and repeatable design
- Works on overall grid system, column based

Lower Levels of Ultrascale Xilinx FPGA

- Slices
 - Basic Element of Logic (BEL) - Building blocks of FPGAs
 - Make up slices which combine to perform the basic resources
 - BRAM, DSP, CLBs
 - Two Slices per CLB
- FSR/Clock Region
 - Makes up a column of resources
 - Each same size by having set amounts of different resources
 - 60 CLBs, 24 DSPs, and 24 18k BRAMs
 - CLB - perform logic, DSPs - specialized, parallel MACs, BRAM - store and transfer data

Levels of Ultrascale* Xilinx FPGA

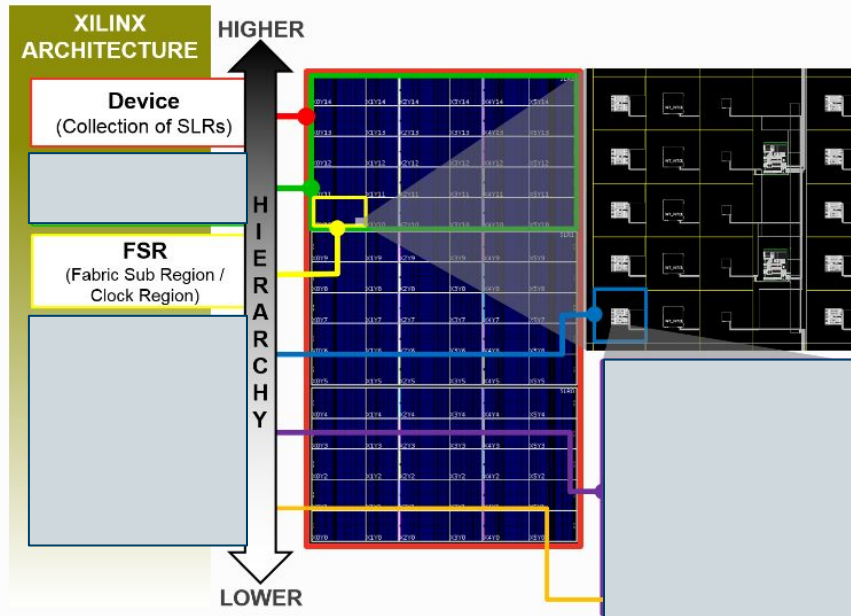


Levels of architectural hierarchy in Xilinx FPGAs.

* Also applies to
Ultrascale+

Credit: Rapidwright.io

Levels of Ultrascale* Xilinx FPGA



Levels of architectural hierarchy in Xilinx FPGAs.

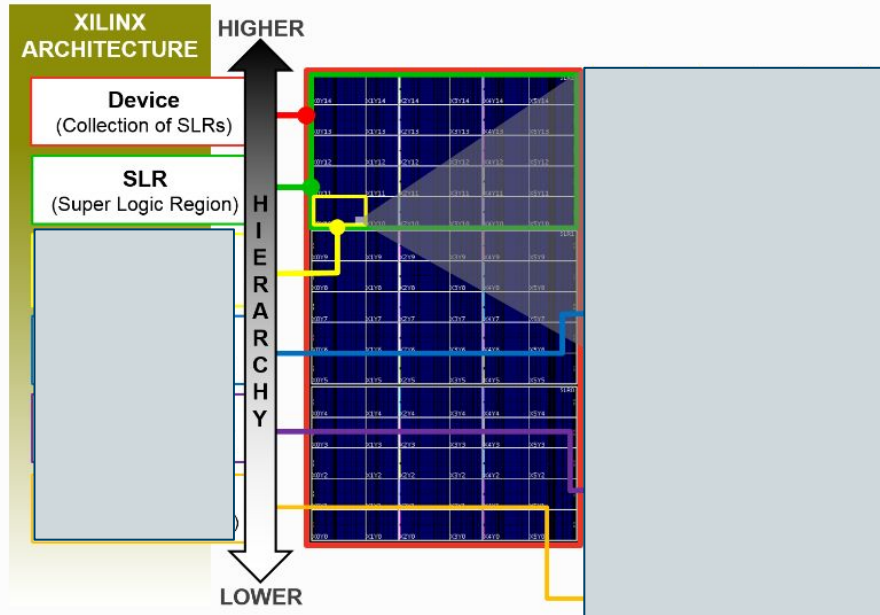
* Also applies to
Ultrascale+

Credit: Rapidwright.io

Higher Levels of Xilinx Ultrascale FPGA

- Super Logic Region (SLR)
 - Only present on some large devices with UltraScale
 - Use SSIT (Stacked Silicon Interconnect Technology) to communicate
 - Special interconnect between multiple SLRs for more efficient communication
 - Is 2D array of FSRs (clock regions)
- Device - The actual FPGA Device
 - Ex - xc7z010clg400-1
 - Device name is info about characteristics

Levels of Ultrascale* Xilinx FPGA



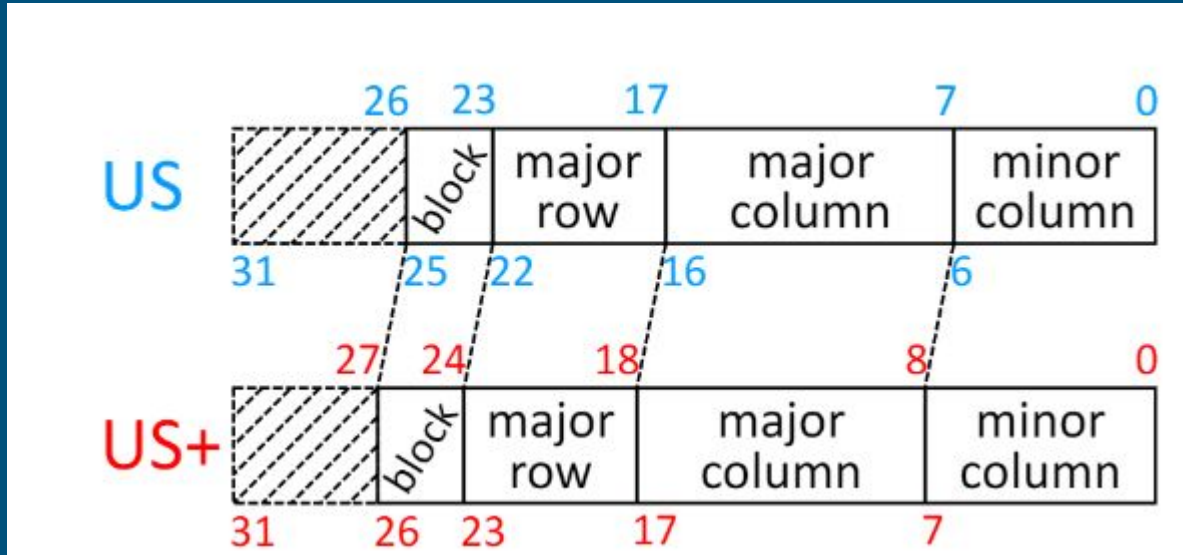
Levels of architectural hierarchy in Xilinx FPGAs.

* Also applies to
Ultrascale+

Credit: Rapidwright.io

Bitstream Structure

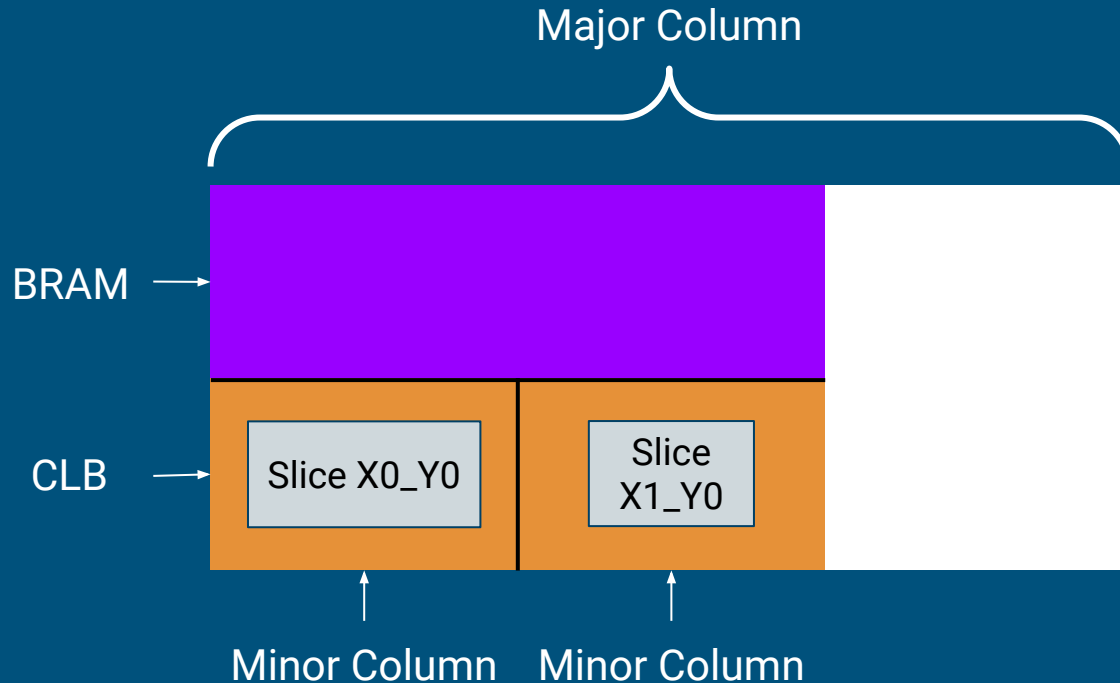
Basic Addressing



Frame Address

Bitfiltrator Paper Fig 2

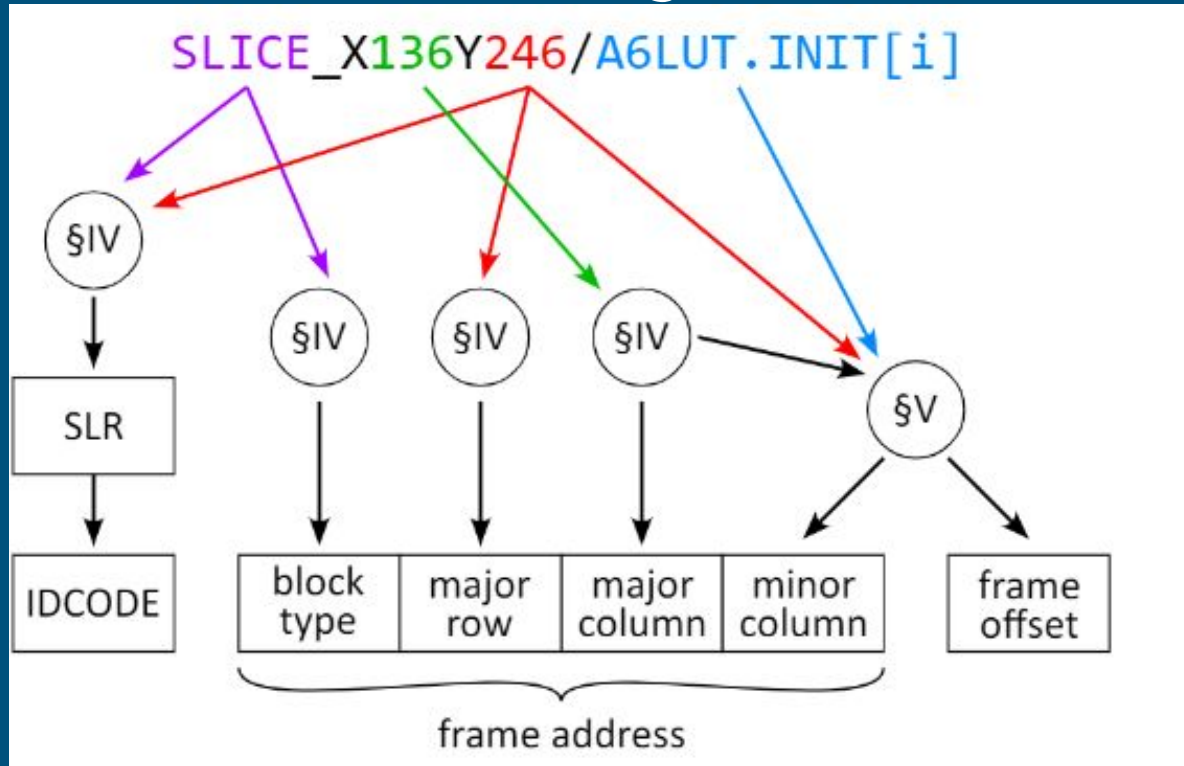
Levels of Ultrascale* Xilinx FPGA



Bitstream Configuration Registers

- Frame Address Register (FAR)
 - Stores Frame of FPGA
 - Basic address from previous slide
- Frame Data Input Register (FDRI)
 - Stores the actual frame data to configure the slice at the FAR address
 - Can start with base address at FAR and then just auto increment
 - Don't need distinct writes to FDRI and FAR for every frame
- IDCODE Register
 - Which SLR are writing to

Addressing a Slice



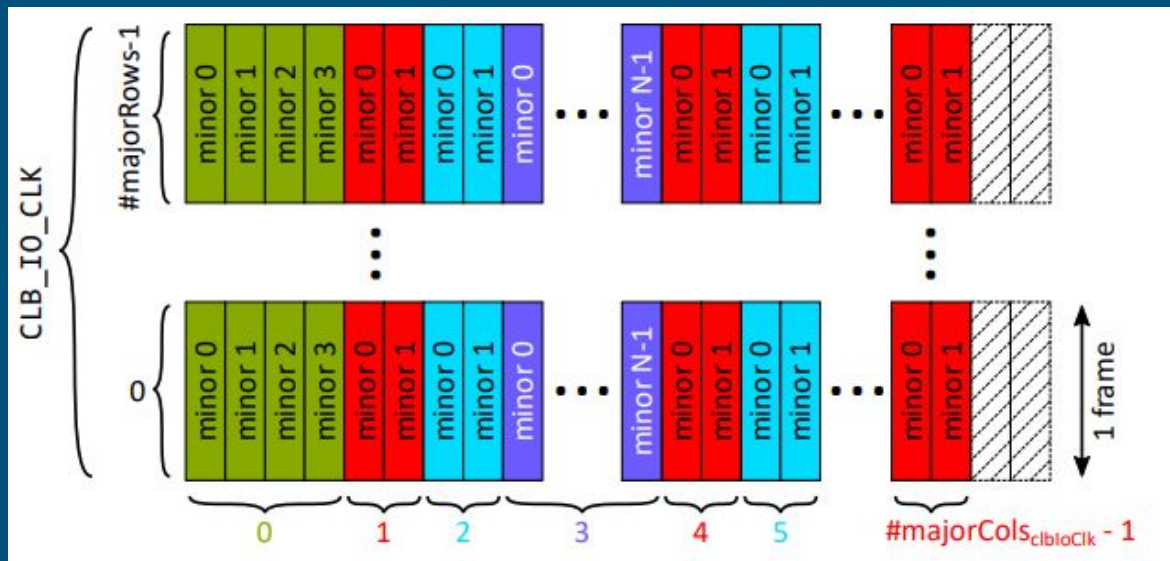
Basic Addressing

- General column based structure
 - Describe Clock Region as a column of resources
- Block Type
 - What type of resource (CLB or BRAM)
- Major row
 - Each will correspond to a certain block type
- Major Column
 - Each has 1 resource type
- Minor Column
 - Identifies slices within a resource

Extracting Device Parameters

Frame Indexing

- How do we locate a configuration frame in bitstream from its address?
 - # major rows, # major columns, # minors per major column



Enumerating Frame Addresses

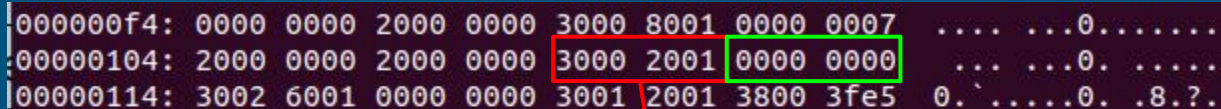
- We can generate a bitstream with a *per-frame* CRC!
- Now we can examine the packets:

80 B of padding

```
> xxd -s 0 -l 224 per_frame_crc_bitstream.bit
00000000: 0009 0ff0 0ff0 0ff0 0ff0 0000 0161 003d .....a.=
00000010: 656d 7074 795f 6465 7369 676e 3b43 4f4d empty_design;COM
00000020: 5052 4553 533d 4641 4c53 453b 5573 6572 PRESS=FALSE;User
00000030: 4944 3d30 5846 4646 4646 4646 463b 5665 ID=0xFFFFFFFF;Ve
00000040: 7273 696f 6e3d 3230 3232 2e31 0062 0015 rsion=2022.1.b..
00000050: 7863 6b75 3032 352d 6666 7661 3131 3536 xcku025-ffva1156
00000060: 2d31 2d63 0063 000b 3230 3234 2f30 342f -1-c.c..2024/04/
00000070: 3036 0064 0009 3031 3a30 333a 3239 0065 06.d..01:03:29.e
00000080: 00fe 1930 ffff ffff ffff ffff ffff ffff ...0.....
00000090: ffff ffff ffff ffff ffff ffff ffff ffff .....
000000a0: ffff ffff ffff ffff ffff ffff ffff ffff .....
000000b0: ffff ffff ffff ffff ffff ffff ffff ffff .....
000000c0: ffff ffff 0000 00bb 1122 0044 ffff ffff .....".D....
000000d0: ffff ffff aa99 5566 2000 0000 2000 0000 .....Uf ... ..
```

A WRITE command

```
000000f4: 0000 0000 2000 0000 3000 8001 0000 0007  ....  ...0.....
00000104: 2000 0000 2000 0000 3000 2001 0000 0000  ...  ...0. ....
00000114: 3002 6001 0000 0000 3001 2001 3800 3fe5  0.  ....0. .8.?.
```



001 10 000000000 00001 00 00000000001

Table 9-16: Type 1 Packet Header Format

Header Type	Opcode	Register Address	Reserved	Word Count
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	xx	RRRRRRRRRxxxxx	RR	xxxxxxxxxxxx

A WRITE command

```
000000f4: 0000 0000 2000 0000 3000 8001 0000 0007  ....  ...0.....
00000104: 2000 0000 2000 0000 3000 2001 0000 0000  ....  ...0.....
00000114: 3002 6001 0000 0000 3001 2001 3800 3fe5  0.  ....0.  .8.?.
```

Type 1 Header

1 Word of Data

001 10 000000000 00001 00 00000000001

WRITE

Register 1
(FAR)

Table 9-16: Type 1 Packet Header Format

Header Type	Opcode	Register Address	Reserved	Word Count
[31:29]	[28:27]	[26:13]	[12:11]	[10:0]
001	xx	RRRRRRRRRxxxxx	RR	xxxxxxxxxxxx

IDCODE and FAR Write

```
0000012c: 3001 8001 0382 4093 3000 8001 0000 0009 0.....@.0.....
```

IDCODE[31:0] = 0x03824093

```
0000018c: 3000 2001 0000 0000 3000 407b 0000 0000 0. ....0.@{....
```

Frame Address = 0x00000000

- Block Type: CLB_IO_CLK
- Major Row: 0
- Major Col: 0
- Minor Col: 0

FAR WRITE

0028ed98: 3000 2001 0002 0084 3000 0001 d3fe dae1 0.0.....

Frame Address = 0x0002084

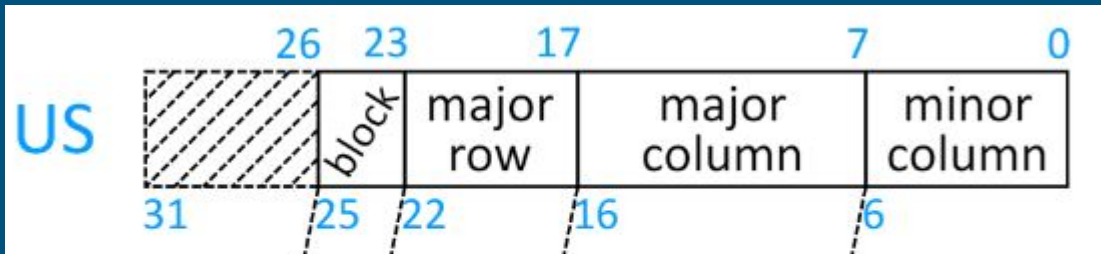
(CLB_IO_CLK)
Block
000000 000 000001 0000000001 0000100
Major Row (1) Major Col (1) Minor Col (4)

FAR WRITE

0028ed98: 3000 2001 0002 0084 3000 0001 d3fe dae1 0.0.....

Frame Address = 0x0002084

(CLB_IO_CLK)
Block
000000 000 000001 0000000001 0000100
Major Row (1) Major Col (1) Minor Col (4)



CRC WRITE

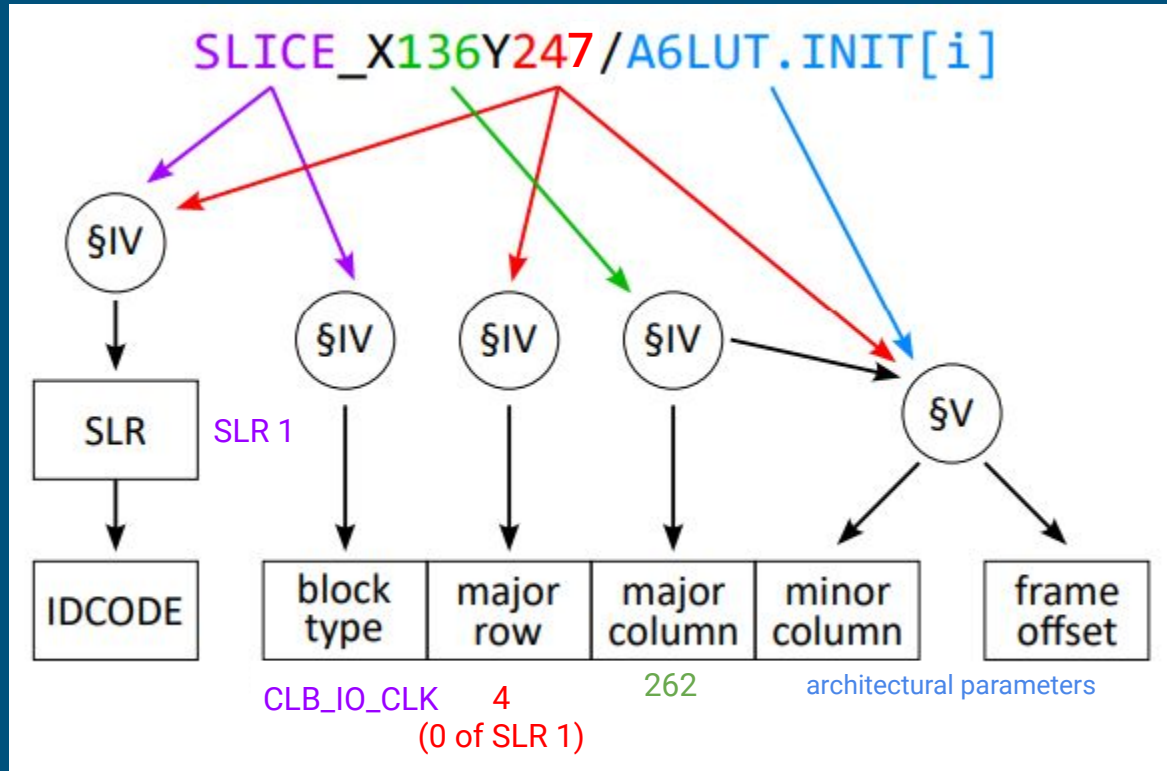
```
| 0000038c: 3000 0001 0348 86f2 3000 407b 0000 0000 0....H..0.@{....
```

CRC[31:0] = 0x034886f2

Identifying Major Columns

- We need to identify which major columns are CLBs, BRAMs, or DSPs
 - Iterate over all clockregions' columns
 - Place one instance of a resource in every column that has a corresponding resource
 - Generate a bitstream
 - Compare the bitstream against an “empty” bitstream
 - Major columns of differing frames contain the resource of interest
- Optimizations can be made for CLBs & BRAMs:
 - Readback capture: extract data from a programmed FPGA
 - Vivado-generated .ll (logic location) file shows frame addresses of each FF/BRAM

Putting it all together: finding a specific BEL



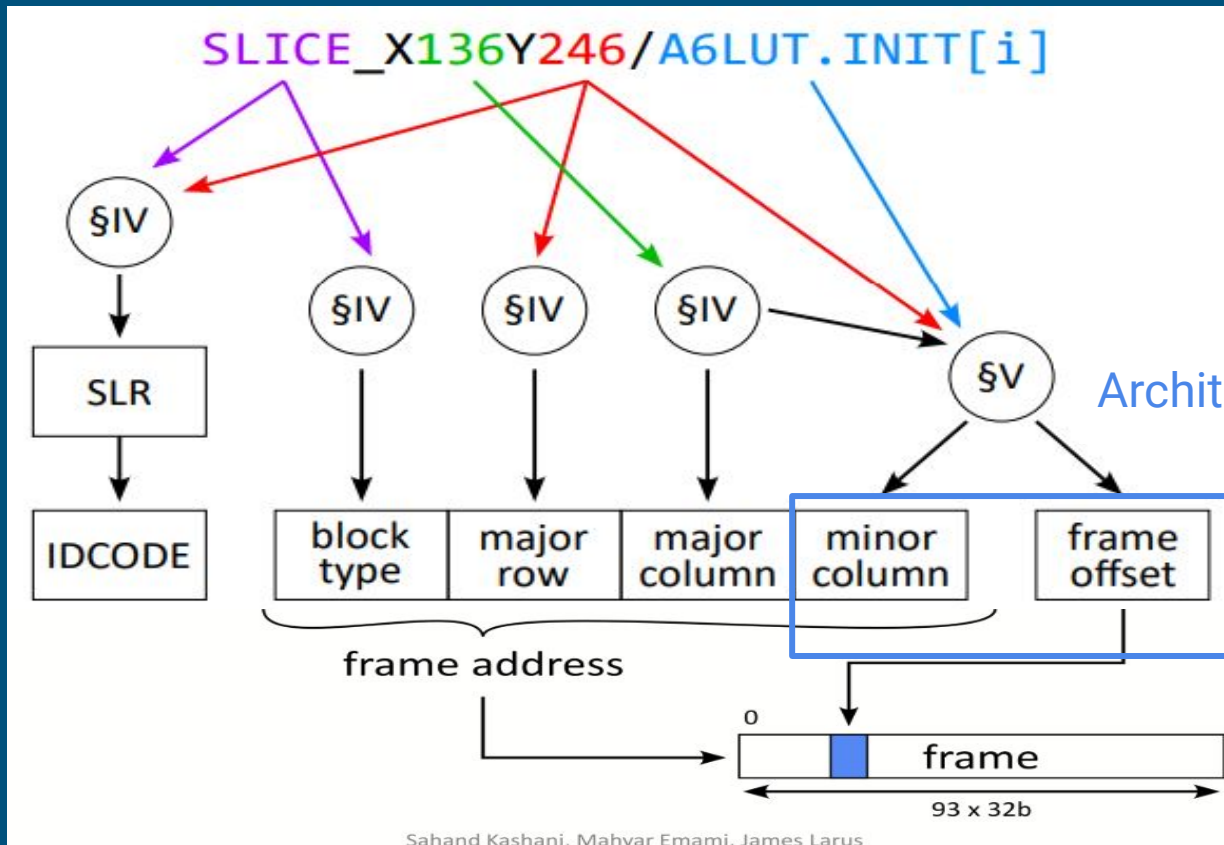
Extracting Architecture Parameters

BRAM Format

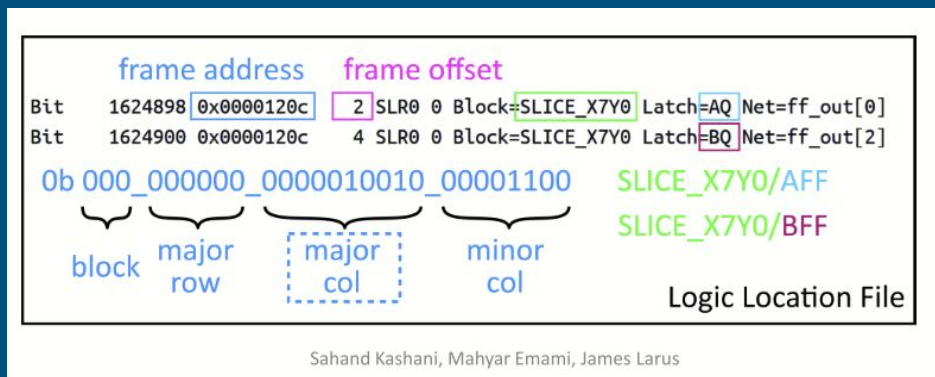
Device Specific

- We now have 3 of the 4 elements in a frame address for a BEL
 - Block type
 - Major row
 - Major column
- Need to find the minor column and frame offset of each bit in a BEL's init

Architecture Specific



BRAM Format



- Select a BRAM column and create a design that populates all its BELs
 - There are 24 18Kb BRAMs in a column with a total of 422 368 bits.
- Vivado can provide frame addresses and offsets of **user-state bits**.
 - Available in **logic location file** when generating bitstream
- Logic file describes the frame addresses and offsets of all 422 368 bits.
 - Parsing the logic location file reveals the minor addresses and frame offsets.

CLB register and LUT format

- Resource columns are formed of tiles.
 - BELs in each tile could have a different format in the bitstream
- BELs are NOT user-state bits and have no logic location file.
 - Complicated to determine the minors and frame offsets
- Impractical to fuzz 30, 720 configuration bits (480 64 bit-LUTS in a column)

THEORY: Hardware is highly regular. Only need to check a few formats and an individual tile in the CLB column

CLB register and LUT format

- Fuzzing the INIT of a CLB tile requires a baseline bitstream to compare to fuzzed variants.
- Fuzzed bitstreams are obtained by loading baseline and altering 1 bit in each
 - By manipulating the LUTs' INIT property in Vivado
- Comparing the bitstreams yields the minor and frame offset of config bits

Putting it together

Having discovered the minor and frame offset of each bit in the INIT properties.

We are now able to form a complete frame address for any such BEL and locate it in the bitstream.

TABLE I: Frame address of SLICE_X136Y247/A6LUT and various bits of its INIT property on a U50 FPGA. The minor column and frame offset of the configuration bits follow well-defined patterns.

INIT Index	SLR	Block Type	Major Row	Major Column	Minor Column	Frame Offset
0	SLR1	CLB_IO_CLK	0	262	11	351
1	SLR1	CLB_IO_CLK	0	262	10	351
2	SLR1	CLB_IO_CLK	0	262	9	351
3	SLR1	CLB_IO_CLK	0	262	8	351
4	SLR1	CLB_IO_CLK	0	262	11	350
5	SLR1	CLB_IO_CLK	0	262	10	350
6	SLR1	CLB_IO_CLK	0	262	9	350
7	SLR1	CLB_IO_CLK	0	262	8	350
...
60	SLR1	CLB_IO_CLK	0	262	11	336
61	SLR1	CLB_IO_CLK	0	262	10	336
62	SLR1	CLB_IO_CLK	0	262	9	336
63	SLR1	CLB_IO_CLK	0	262	8	336

SLICE_X136Y247/A6LUT is the 7th LUT in the CLB column when counting from the bottom of SLR1. Bit 0 in this LUT's INIT equation is in minor column 11, at frame offset 351

Evaluation

TABLE II: Summary of devices on which Bitfiltrator was tested. Devices are enumerated in Vivado and categorized by their architecture and family name. Bitfiltrator was only tested on devices for which bitstreams can be generated using the free Vivado “WebPack” license. No Virtex UltraScale or Zynq UltraScale+ RFSOC devices are available in the free version of Vivado.

Architecture	Vivado Name	Count
UltraScale	Kintex UltraScale	2 / 12
	Virtex UltraScale	0 / 7
UltraScale+	Kintex UltraScale+	3 / 10
	Virtex UltraScale+	8 / 31
	Zynq UltraScale+	21 / 38
	Zynq UltraScale+ RFSOC	0 / 16

Bitfiltrator Overview

- Input:
 - Target FPGA part number
- Output:
 - Number of visible and “hidden” major rows
 - Number of major columns in every row
 - Number of minor columns in every major column
 - Major columns of CLBs, BRAMs, and DSPs
- Validation:
 - Performed an additional test using a Tcl script setting INIT to a random value and attempt to recreate every initial value.
 - Supports our hypothesis that all tiles of the same type have the same format in the bitstream and are translated versions of one another.

Bitfiltrator Constraints

- Only used the smallest member of each device family as a proxy to extract information
 - Device-specific parameters and LUTs' INIT property are fuzzed in parallel over 24 cores
- End-to-end runtime is under 48h. Over 90% of the total execution time is spent fuzzing LUT INIT properties.

Conclusion

Questions?
