Zinnia

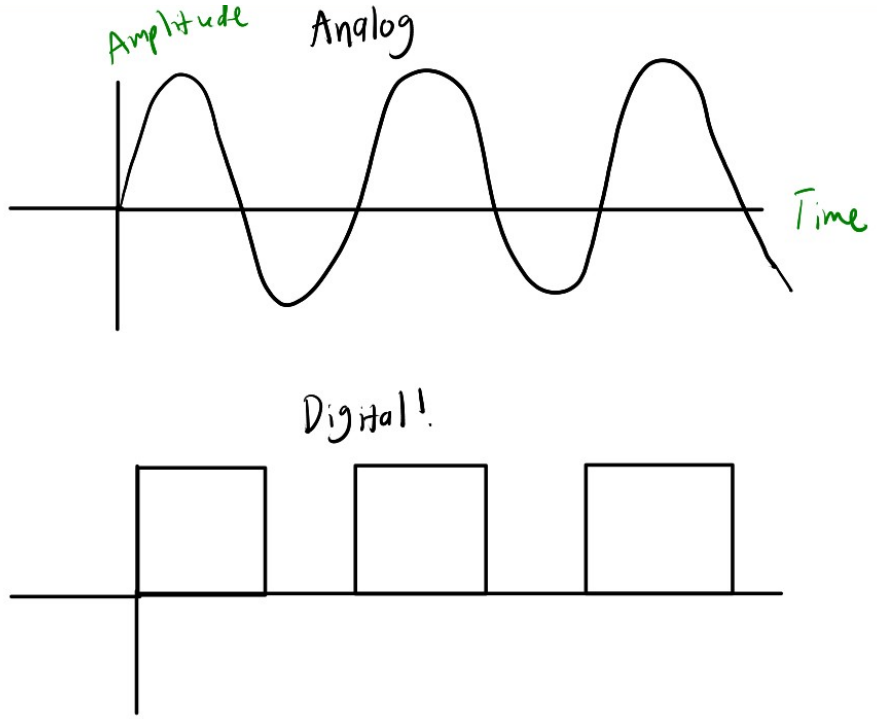
A (questionable) experiment in DSL design and parallel algorithms for signal processing

Yusuf Bham
Arjun Vedantham

Outline

Digital Signal Processing (DSP)

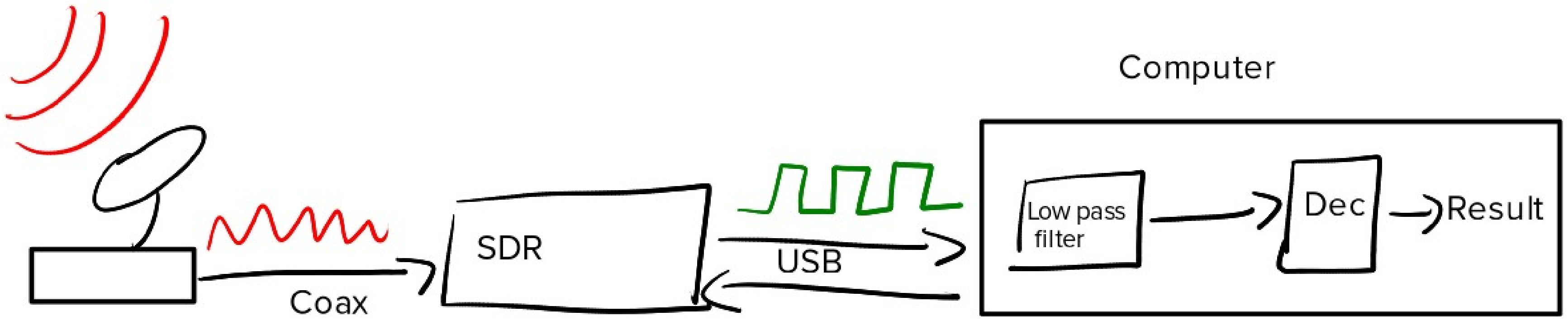
- In the past: discrete hardware components
- Nowadays: Process digital samples in software
- Example: Software defined radio



RTL-SDR

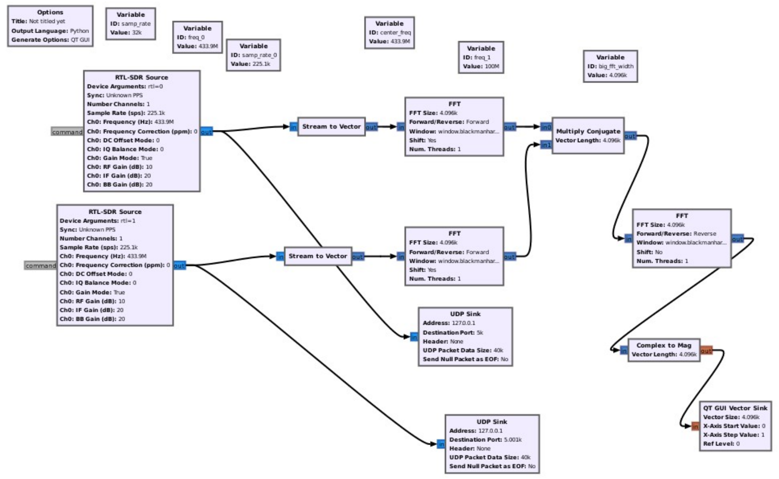
SDR Pipeline

Digital signal processing is great for flexibility and retargetability!



DSP Programming Approaches

- Generates long, hard to read Python scripts
- CPU-bound!



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Variable
ID: freq_0
Value: 433.9M

Variable
ID: samp_rate_0
Value: 225.1k

Variable
ID: center_freq
Value: 433.9M

Variable
ID: freq_1
Value: 100M

Variable
ID: big_fft_width
Value: 4.096k

RTL-SDR Source
Device Arguments: rtl=0
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 225.1k
Ch0: Frequency (Hz): 433.9M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: True
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

RTL-SDR Source
Device Arguments: rtl=1
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 225.1k
Ch0: Frequency (Hz): 433.9M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: True
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Stream to Vector

Stream to Vector

FFT
FFT Size: 4.096k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

FFT
FFT Size: 4.096k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Multiply Conjugate
Vector Length: 4.096k

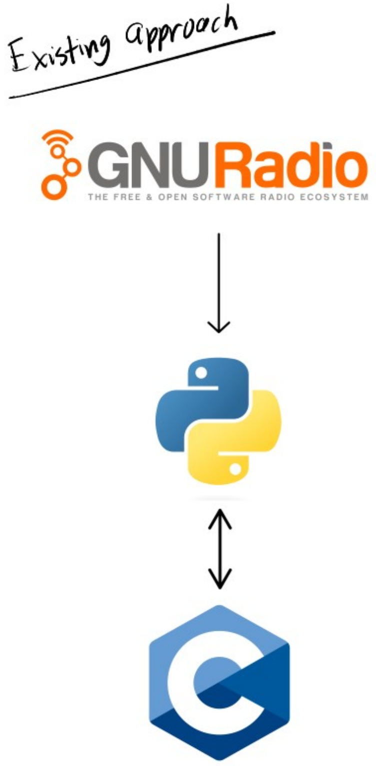
FFT
FFT Size: 4.096k
Forward/Reverse: Reverse
Window: window.blackmanhar...
Shift: No
Num. Threads: 1

Complex to Mag
Vector Length: 4.096k

QT GUI Vector Sink
Vector Size: 4.096k
X-Axis Start Value: 0
X-Axis Step Value: 1
Ref Level: 0

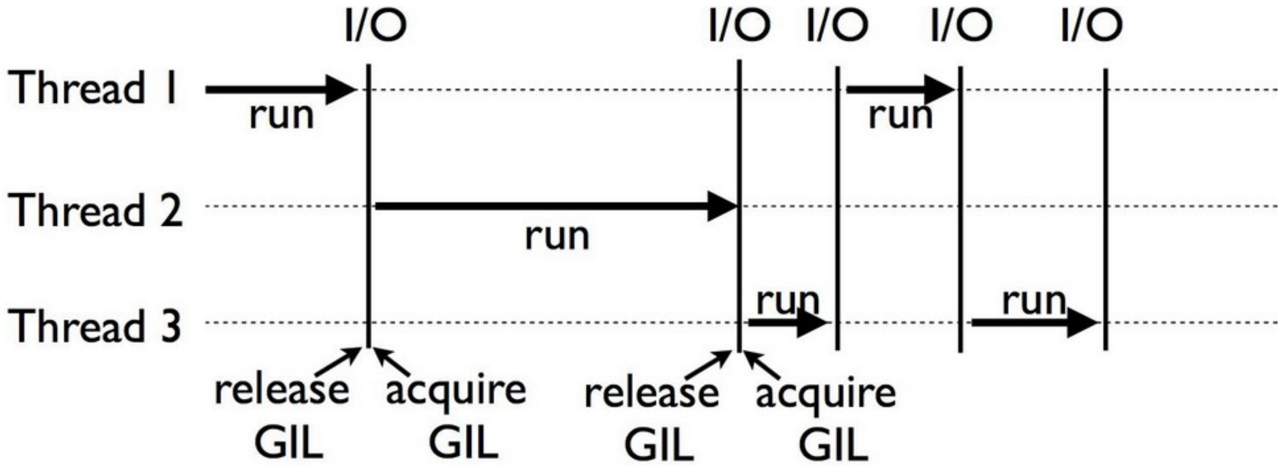
UDP Sink
Address: 127.0.0.1
Destination Port: 5k
Header: None
UDP Packet Data Size: 40k
Send Null Packet as EOF: No

UDP Sink
Address: 127.0.0.1
Destination Port: 5.001k
Header: None
UDP Packet Data Size: 40k
Send Null Packet as EOF: No



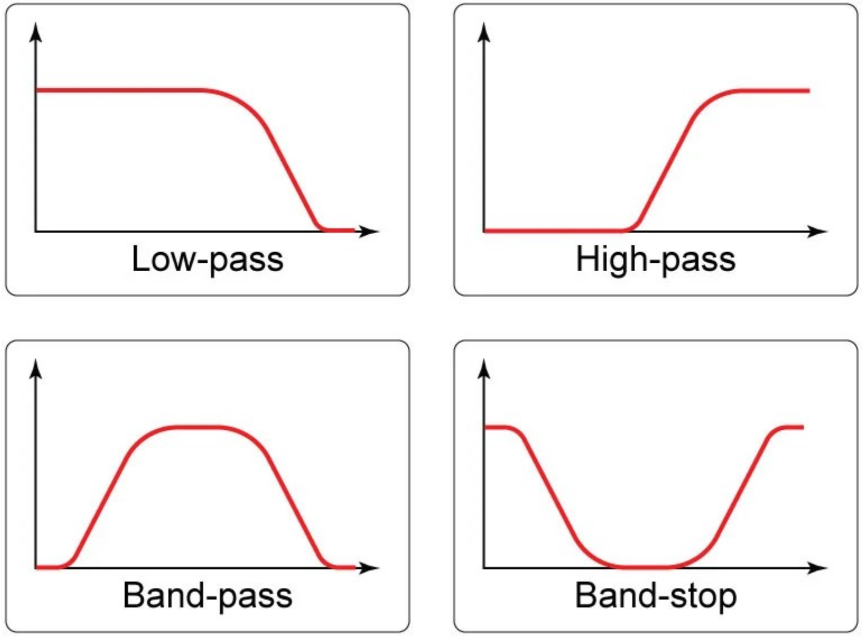
Issues

- CPU bound
- Python output stunts parallelism



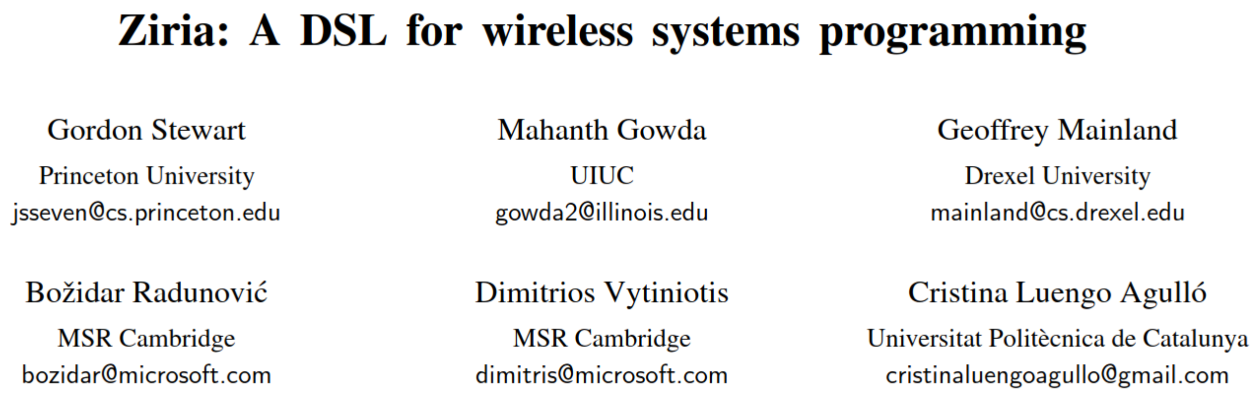
A New Approach?

- Hardware acceleration?
  - FPGAs
- Domain specific language?
  - Better parallelization?
  - Functional paradigm
    - Compose filters



Ziria

- Similar premise
- Targeted CPUs instead of FPGAs
- Aimed at wireless systems programming



Ziria: A DSL for wireless systems programming

Gordon Stewart
Princeton University
jsseven@cs.princeton.edu

Mahanth Gowda
UIUC
gowda2@illinois.edu

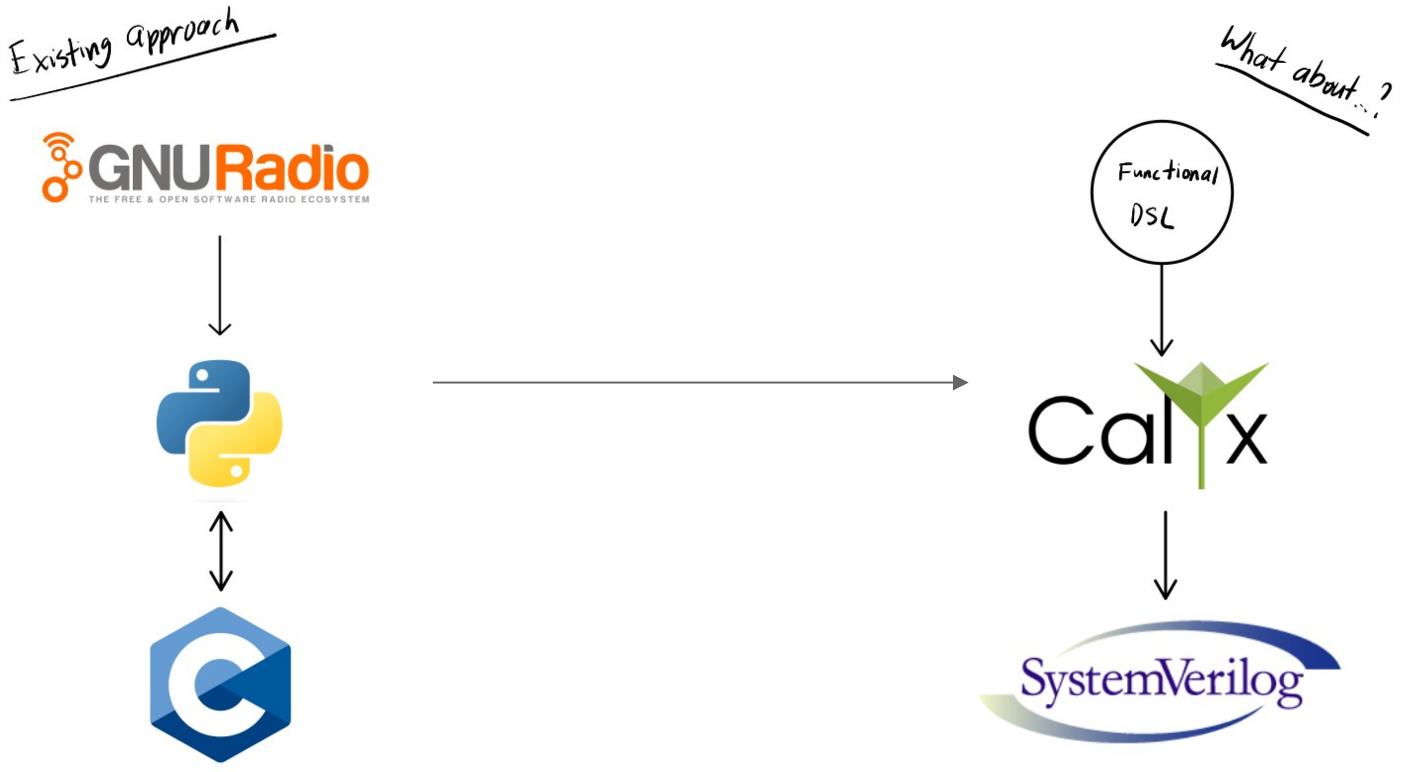
Geoffrey Mainland
Drexel University
mainland@cs.drexel.edu

Božidar Radunović
MSR Cambridge
bozidar@microsoft.com

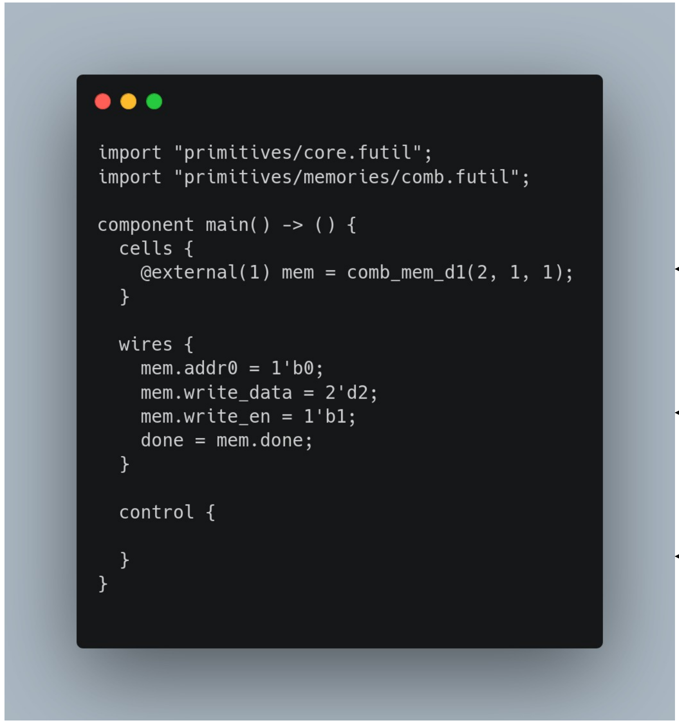
Dimitrios Vytiniotis
MSR Cambridge
dimitris@microsoft.com

Cristina Luengo Agulló
Universitat Politècnica de Catalunya
cristinaluengoagullo@gmail.com

A New Approach?



Calyx?



```
import "primitives/core.futil";
import "primitives/memories/comb.futil";

component main() -> () {
  cells {
    @external(1) mem = comb_mem_d1(2, 1, 1);
  }

  wires {
    mem.addr0 = 1'b0;
    mem.write_data = 2'd2;
    mem.write_en = 1'b1;
    done = mem.done;
  }

  control {

  }
}
```

Sample Calyx program

Intermediate representation (IR) for hardware generation

Memory cells

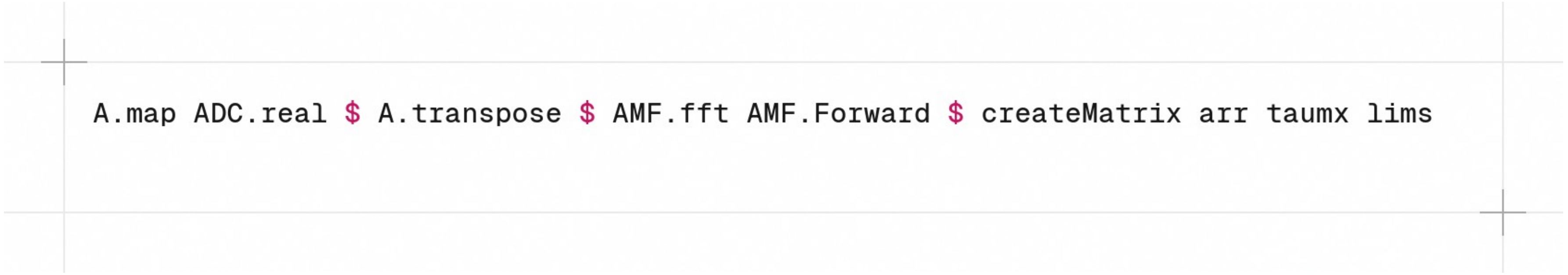
Wires (assignments)

Control schedule

Zinnia

Language Design - Overview

- Functional
  - Easy composition
- Modular
- Amenable to parallelization
- Target: Calyx IR -> synthesizable SystemVerilog



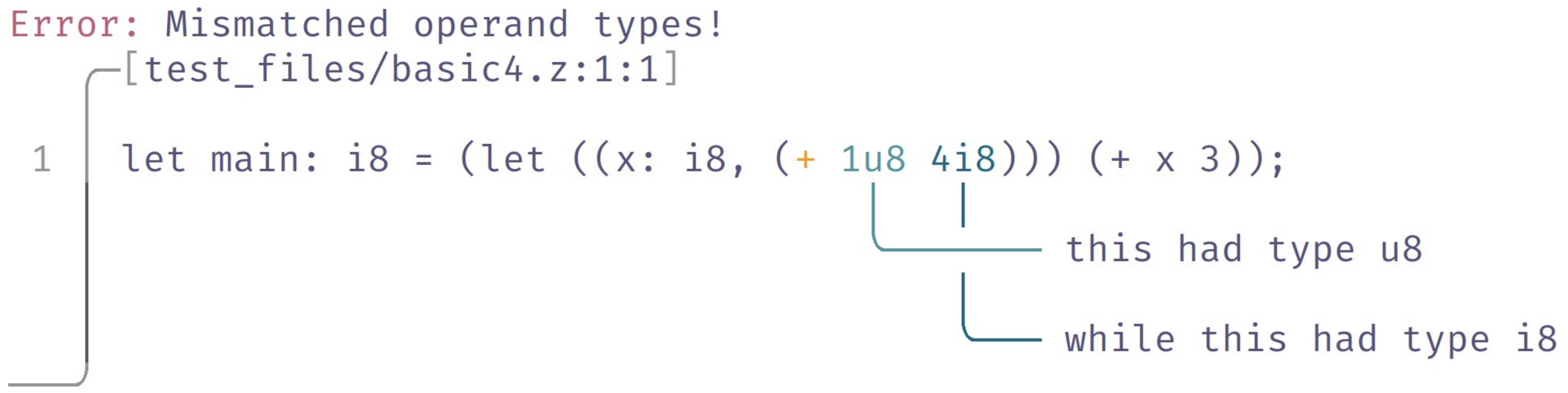
```
A.map ADC.real $ A.transpose $ AMF.fft AMF.Forward $ createMatrix arr taumx lims
```

Language Design - Typing

- Bidirectional type checker
- Inspired by “Complete and Easy Bidirectional Typechecking for Higher-Rank Polymorphism”
- Supports high levels of type inference
- Support for generics

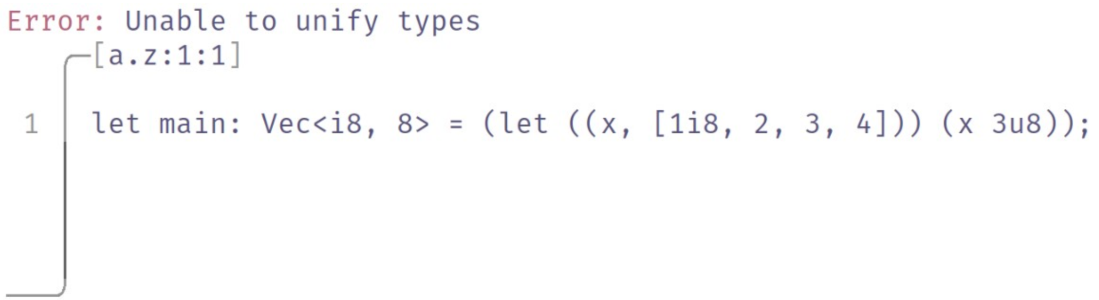
$$\frac{\Psi \vdash A \qquad \Psi \vdash e \Leftarrow A}{\Psi \vdash (e : A) \Rightarrow A}$$

```
scan :: (Num n, Int c) => Vec c n -> Vec c n
```



```
Error: Mismatched operand types!
   ╭─[test_files/basic4.z:1:1]
   │
 1 │ let main: i8 = (let ((x: i8, (+ 1u8 4i8))) (+ x 3));
   │                                  │   │
   │                                  ╰───┼──── this had type u8
   │                                      │
   │                                      ╰──── while this had type i8
───╯
```

Type Checking

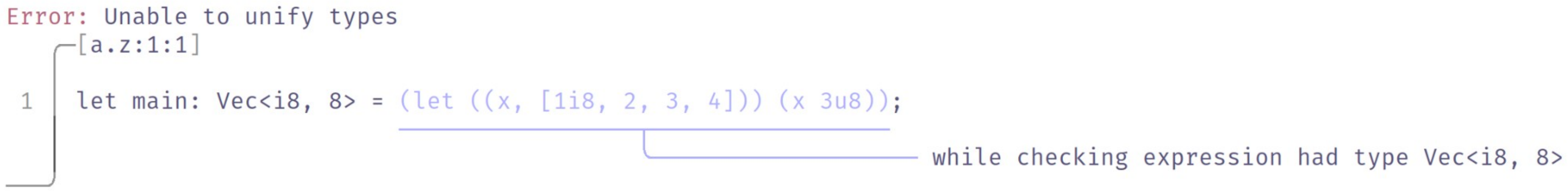


```
Error: Unable to unify types
    ╭─[a.z:1:1]
    │
  1 │ let main: Vec<i8, 8> = (let ((x, [1i8, 2, 3, 4])) (x 3u8));
    │
    │
    │
    │
    │
────╯
```

Type Checking

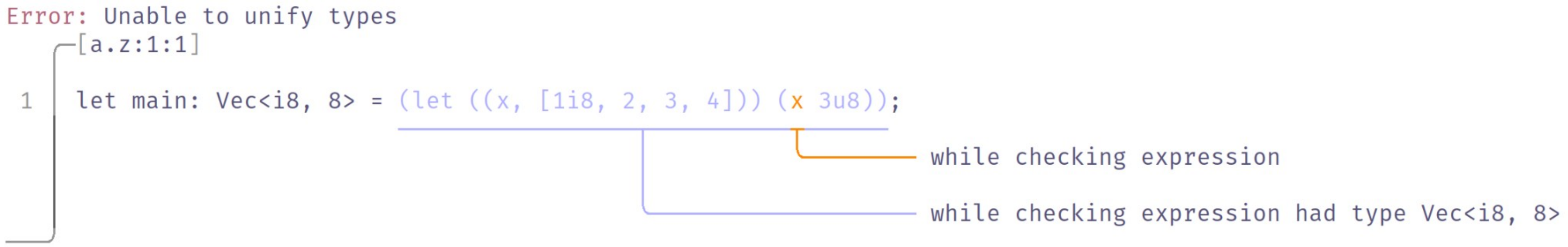
```
Error: Unable to unify types
   ╭─[a.z:1:1]
   │
 1 │ let main: Vec<i8, 8> = (let ((x, [1i8, 2, 3, 4])) (x 3u8));
   │
───╯
```

Type Checking



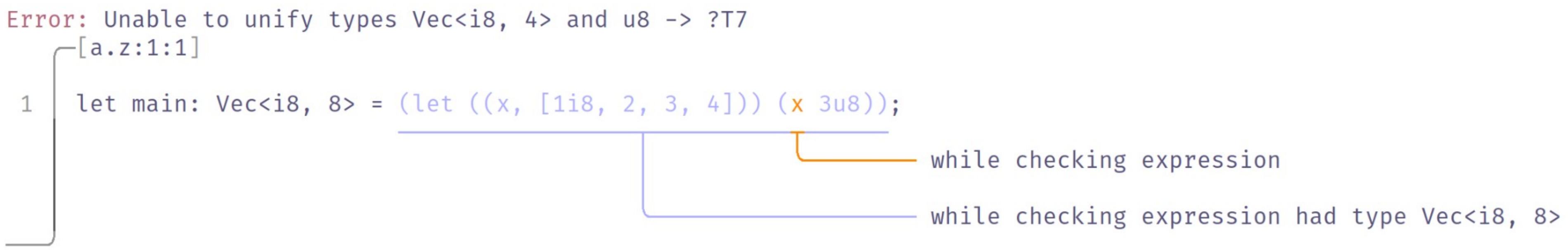
```
Error: Unable to unify types
    ╭─[a.z:1:1]
    │
  1 │ let main: Vec<i8, 8> = (let ((x, [1i8, 2, 3, 4])) (x 3u8));
    │                        ──────────────────┬──────────────────
    │                                          ╰──────────────────── while checking expression had type Vec<i8, 8>
────╯
```

Type Checking



```
Error: Unable to unify types
    ╭─[a.z:1:1]
    │
  1 │ let main: Vec<i8, 8> = (let ((x, [1i8, 2, 3, 4])) (x 3u8));
    │                        ───────────────────┬──────────┬─────
    │                                           │          ╰────── while checking expression
    │                                           │
    │                                           ╰───────────────── while checking expression had type Vec<i8, 8>
────╯
```

Type Checking

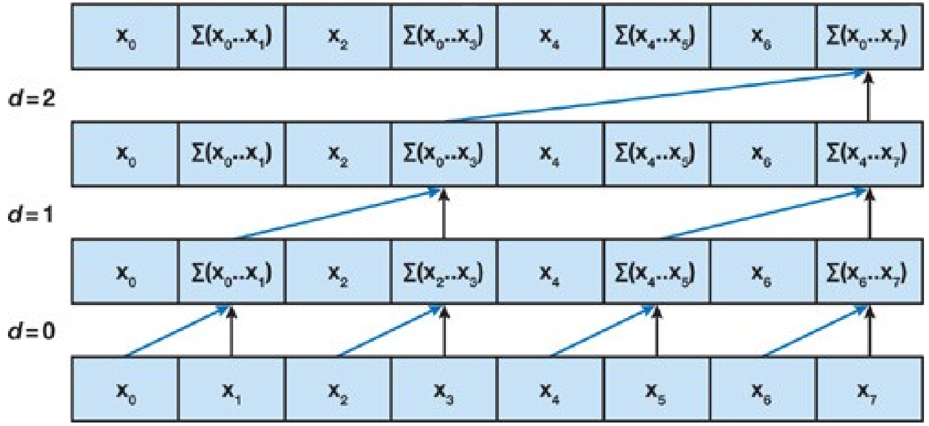


```
Error: Unable to unify types Vec<i8, 4> and u8 -> ?T7
   ╭─[a.z:1:1]
   │
 1 │ let main: Vec<i8, 8> = (let ((x, [1i8, 2, 3, 4])) (x 3u8));
   │                        ─────────────────┬──────────┬──────
   │                                         │          ╰──────── while checking expression
   │                                         │
   │                                         ╰─────────────────── while checking expression had type Vec<i8, 8>
───╯
```

Type Checking

Language Implementation - Scan

- Parallel scan/prefix sums
- Efficient for user-defined higher order functions
    - Examples: 1D convolution, rejecting HF samples



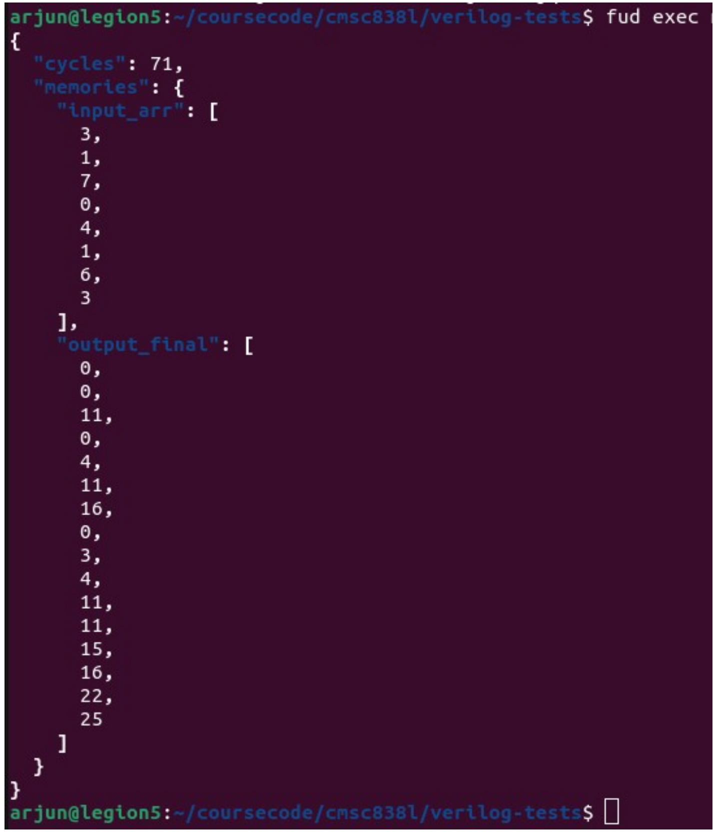
Upsweep in parallel scan

Evaluation Plan

- Scalability
- Correctness
- Hardware deployment [reach]

Evaluation - Scalability

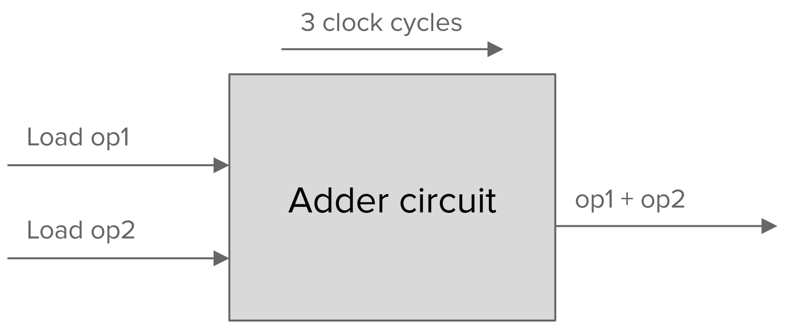
- Still evaluating
- Focus on cycle count in simulation
  - Show that HLS overhead is minimal
  - *Work should grow with O(n)*
  - Scalability currently limited

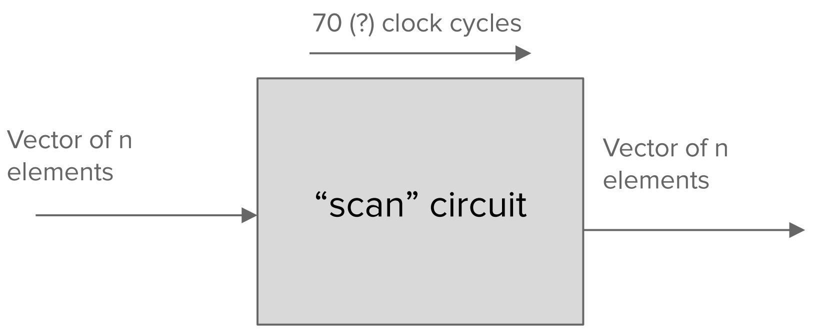


```
arjun@legion5:~/coursecode/cmsc838l/verilog-tests$ fud exec
{
  "cycles": 71,
  "memories": {
    "input_arr": [
      3,
      1,
      7,
      0,
      4,
      1,
      6,
      3
    ],
    "output_final": [
      0,
      0,
      11,
      0,
      4,
      11,
      16,
      0,
      3,
      4,
      11,
      11,
      15,
      16,
      22,
      25
    ]
  }
}
arjun@legion5:~/coursecode/cmsc838l/verilog-tests$
```

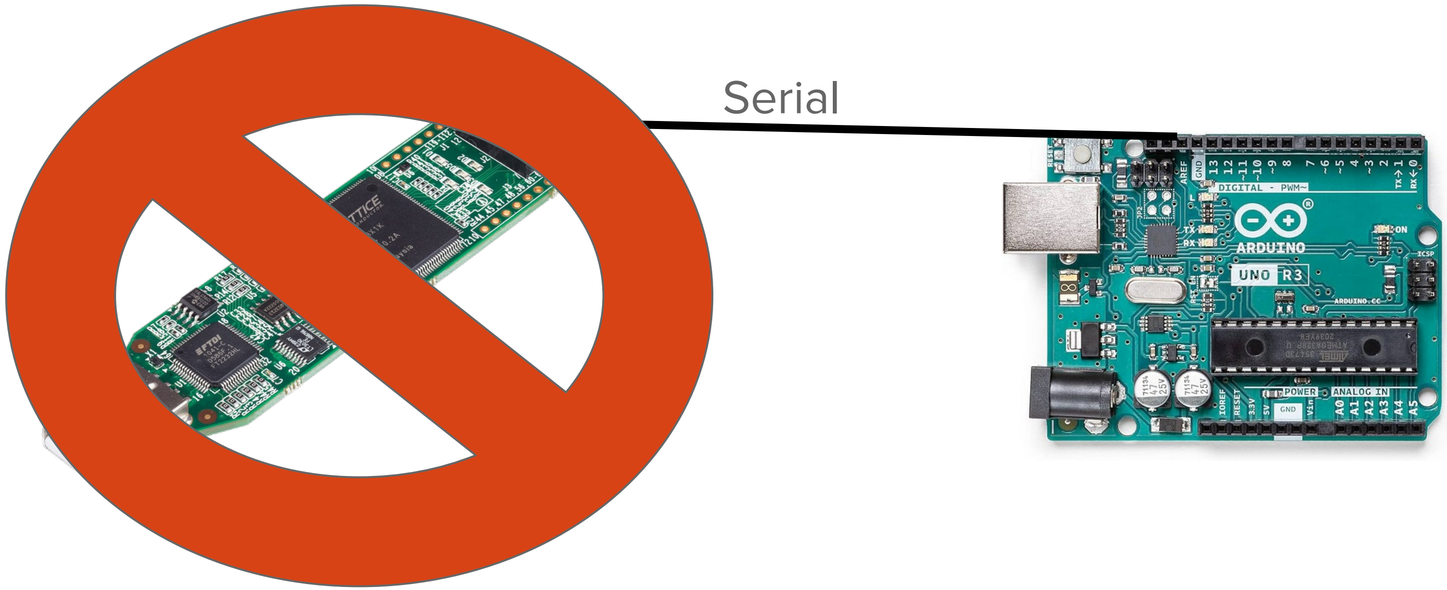
Evaluation - Correctness

Circuit testing with cocotb



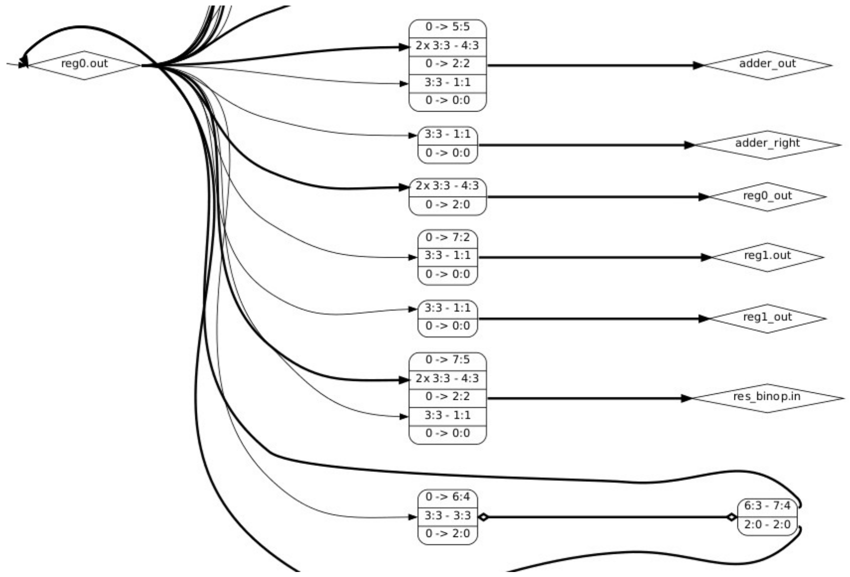


Evaluation - Hardware

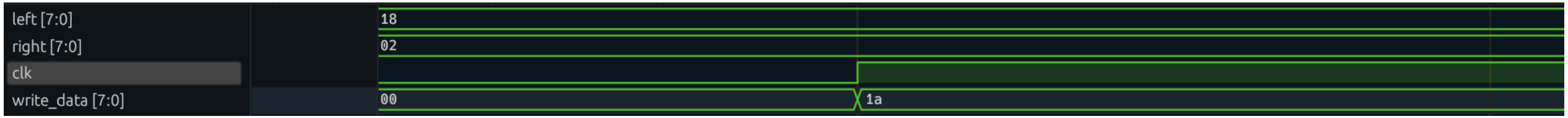


Reprogramming requires a stable clock signal

Demo



(Part of) the circuit DAG



Future Work/Wishlist

- Short term
  - Finish evaluation
  - Add functions and loops
- Long term
  - Improve scan performance with in-place algorithm
  - Greater parallelization (memory banking)
  - Incorporate latency into the control schedule
  - Linear typing
    - Reuse vector storage
  - Dependent types

Summary

- Functional, composable language for creating DSP circuits
- Implements bidirectional typing features
- Provides an easy way to parallelize workloads in hardware
    - Parallel scan primitive

References

- https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda
- https://chandrashek1007.medium.com/python-global-interpreter-lock-is-it-good-or-bad-634d1c82b4fd
- https://dl.acm.org/doi/10.1145/2786763.2694368