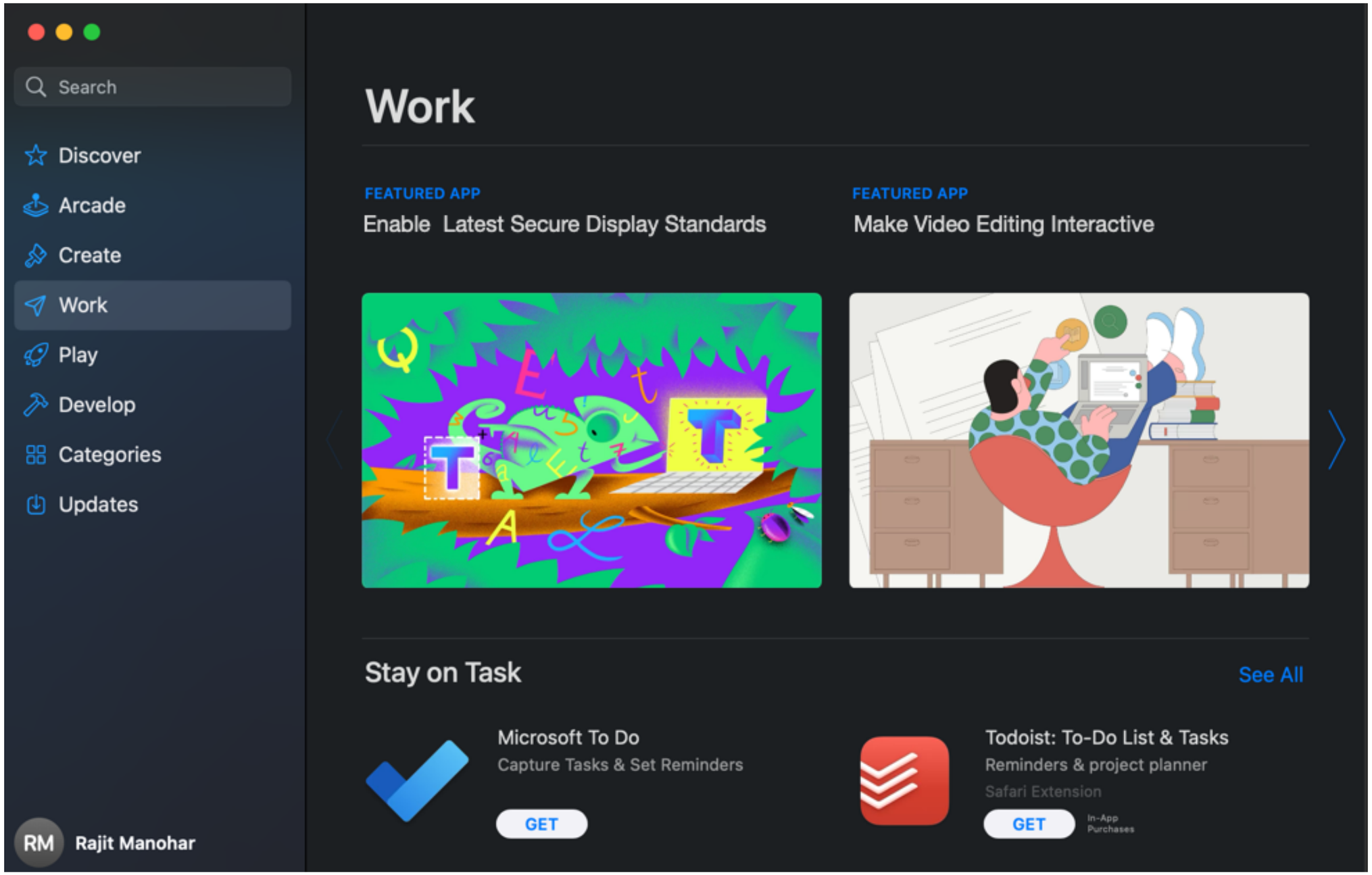
Using chiplets to enable a firmware app store

Rajit Manohar
Asynchronous VLSI and Architecture Group
Computer Systems Lab
Yale University
https://avlsi.csl.yale.edu/

Yale

AVLSI

What might an “app store” for firmware look like?



It should look the same….

Yale

AVLSI

Some requirements

- Instant gratification
  - ❖ Download update over the network and *immediately* see the benefit
    - ‣ Reconfigurable accelerator, like an FPGA

- “30 day free trial”
  - ❖ Security mechanism needed

- Software support
  - ❖ Existing applications should be “upgradable”
  - ❖ Enable software developers to create accelerators
    - ‣ US hardware engineers (US): ~65K*
    - ‣ US software engineers (US): ~ 1.48M (> 22x)

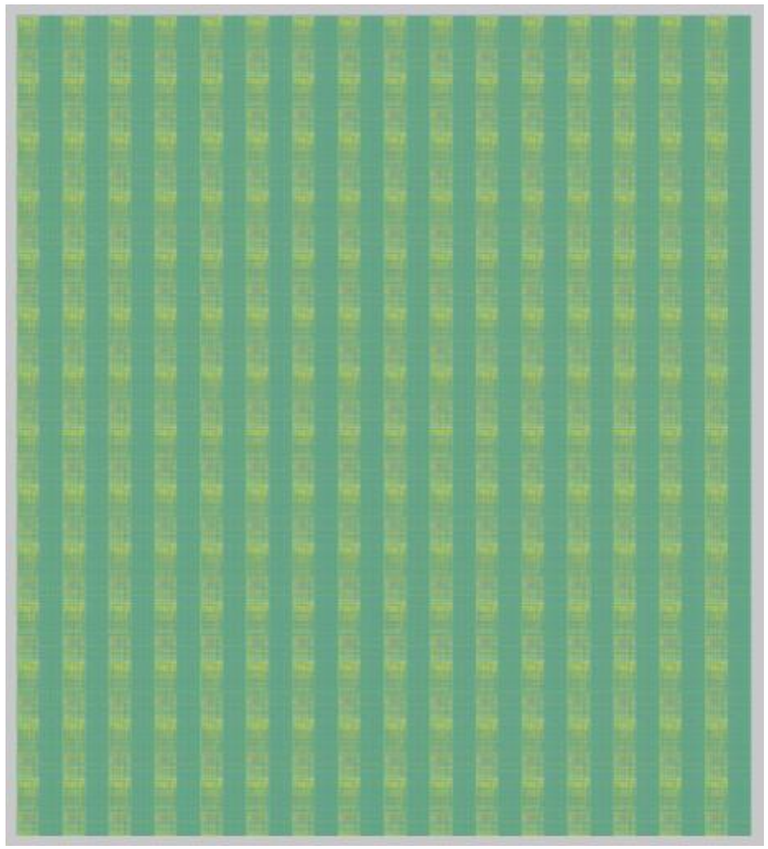
Yale

**Source: US Bureau of Labor Statistics*

AVLSI

Completed project: processor-FPGA integration

- Tight integration permits…
  - ❖ Greater opportunities for acceleration thanks to reduced latency
  - ❖ Leveraging the combination of FPGA strengths and CPU strengths

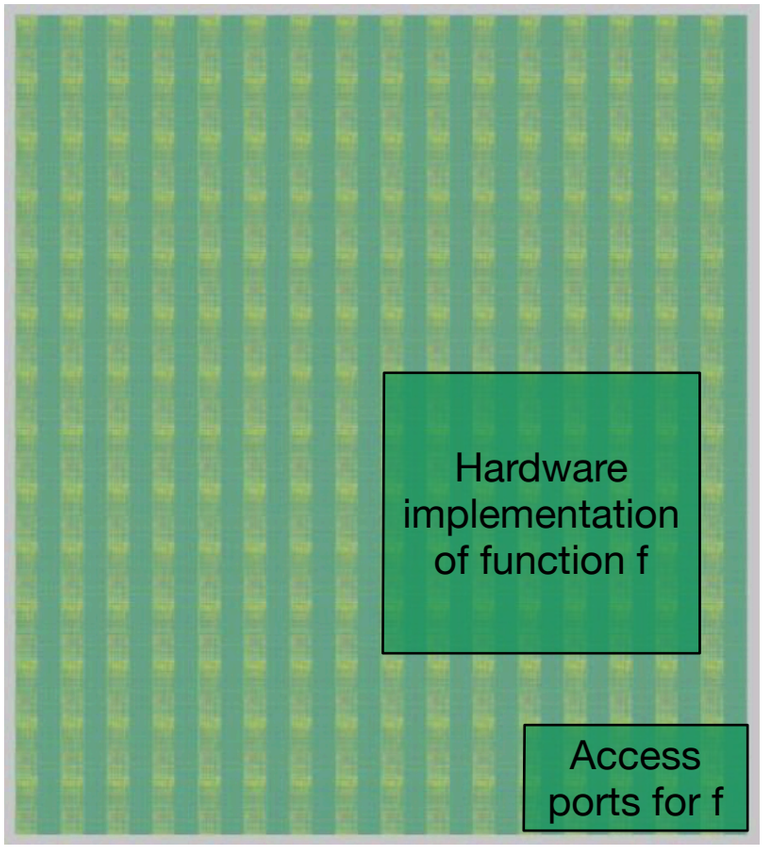


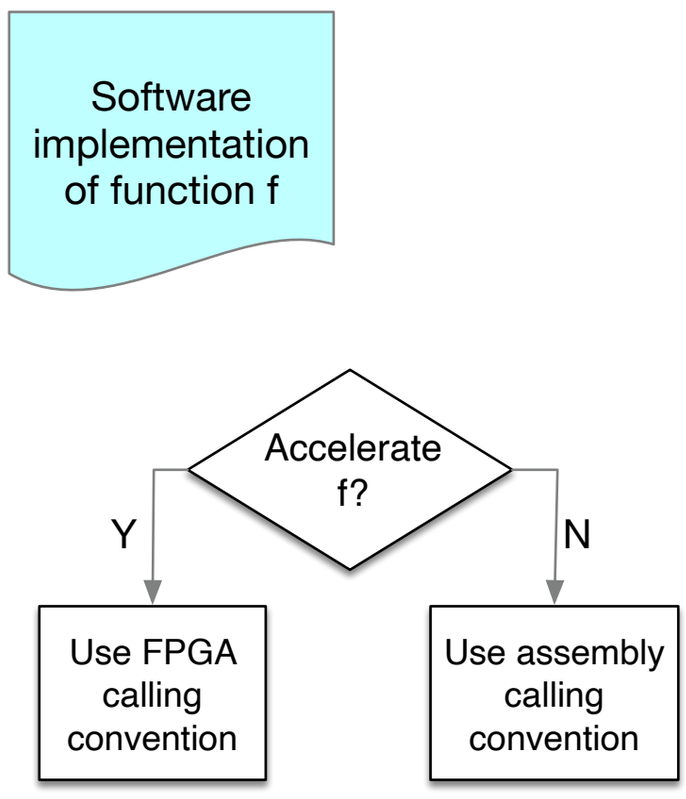
Asynchronous FPGA fabric (65nm) ~2012

- FPGA features
  - ❖ Configuration protected via CPU protection mechanisms, permits multi-programmed workloads
  - ❖ Runtime fine-grained reconfiguration supported
  - ❖ Polling and interrupt-based interface to the host (clocked) CPU
    - ‣ NULL accelerator (loopback) latency < 10 cycles

How does the software work?

- Defined a set of “FPGA calling conventions”
  - ❖ Analogous to assembly language calling conventions in software
  - ❖ FPGA access ports (dedicated interface hardware)
- Stub functions + firmware management and scheduling
  - ❖ Lots of details in getting this to work properly…





Yale

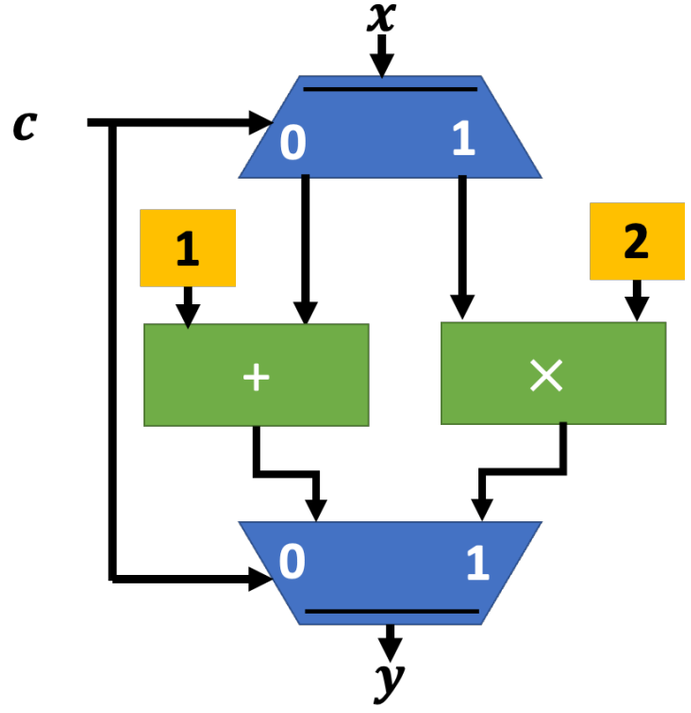
AVLSI

How do we map software to hardware?

- Current project: mapping unmodified software to asynchronous circuits
  ("High Level Synthesis" or HLS)

```
int x;
int y=0;
if(c){
  y=x*2;
} else {
  y=x+1;
}
```

- Our approach
  - ❖ Map programs into dataflow graphs
  - ❖ Translate dataflow components directly into asynchronous hardware
  - ❖ Note: standard HLS tools use state-machine + allocation, scheduling, binding



- Why asynchronous hardware?
  - ❖ Enable incremental improvement of the hardware
  - ❖ Closer match to software cost models

Yale

AVLSI

Fluid: an asynchronous HLS framework

- Implementation using LLVM compiler framework
  - ❖ C/C++ to dataflow graph to asynchronous hardware
  - ❖ Comparison against academic state-of-the-art HLS tools + commercial HLS tools
- Results
  - ❖ Raw translation is poor (reported by previous papers as well)
  - ❖ Translation + dataflow optimizations lead to significant improvements

	Academic	Commercial
Area	1.19x higher	1.95x higher
Energy	8x lower	3.58x lower
Latency	1.6x lower	1.15x higher*
Throughput	2.5x higher	1.34x higher

Yale

**due to one outlier benchmark*

AVLSI

Off-the-shelf FPGA prototyping

- Many of the techniques required asynchronous logic
  - … unsupported by commercial EDA flows

- Tool developments
  - Open-source asynchronous hardware description language

    `https://github.com/asyncvlsi/act/`
  - Automated translation to synthesizable, vendor-neutral Verilog model for FPGA prototyping
    - Current results: 15x-30x slowdown compared to ASIC in a similar feature size, if design fits on a single FPGA

- Next steps
  - Use a standard processor (RISC-V) to create a complete prototype system

Yale

AVLSI

Summary and acknowledgments

- A firmware app store is possible!
  - ❖ Tight coupling between high-quality CPUs and high-quality FPGAs is needed
  - ❖ Chiplets with tight FPGA/CPU integration are an enabling technology
  - ❖ More work is needed on pure software acceleration, rather than “Verilog written in C/C++”

- Thanks to my current and former students
  - ❖ John Teifel (early dataflow synthesis + FPGA)
  - ❖ Song Peng, David Fang (early dataflow synthesis)
  - ❖ Benjamin Hill (FPGA/CPU integration)
  - ❖ Rui Li (Fluid)

- Sponsors: AFRL, DARPA

Yale

AVLSI