Apps with Hardware
Enabling Run-time Architectural Customization in Smart Phones

Michael Coughlin, Ali Ismail, Eric Keller
University of Colorado Boulder
Mobile Devices

Devices are designed around certain restrictions

This leads vendors to make tradeoffs

What if users and developers could choose?
Vision: Smart Phone with an FPGA
Software-defined Radio

Picasso: Flexible RF and Spectrum Slicing

Steven Hong
Stanford University
hsyng@stanford.edu

Jeffrey Mohlman
Stanford University
jmehlman@stanford.edu

Sachin Katti
Stanford University
skatti@stanford.edu

ABSTRACT
This paper presents the design, implementation and evaluation of Picasso, a novel radio design that allows simultaneous transmission and reception on scenario and arbitrary spectrum fragments using a single RF front end and antenna. Picasso leverages this capability to flexibly partition fragmented spectrum into multiple slices that share the RF front end and antennas, yet operate concurrent and independent IEEE802.11 protocols. We show how this capability provides a novel and clean abstraction of the fragmented spectrum in Wi-Fi networks and handles coexistence in dense deployments. We prototype Picasso, and demonstrate experimentally that a Picasso radio
High-performance Computing

Cryptography

<table>
<thead>
<tr>
<th>Technology</th>
<th>Throughput (GBytes/Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>E5503 Xeon Processor</td>
<td>0.01 (Single core)</td>
</tr>
<tr>
<td>AMD Radeon HD 7970</td>
<td>0.33</td>
</tr>
<tr>
<td>PCIe385 FPGA Accelerator</td>
<td>5.20</td>
</tr>
</tbody>
</table>


Analytics

<table>
<thead>
<tr>
<th>Ryft ONE Primitive</th>
<th>Analytics throughput of a single, fully-populated 1U Ryft ONE device</th>
<th>Equivalent Spark cluster size (to match Ryft ONE performance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>~10GB/sec</td>
<td>&gt; 100 nodes(^1)</td>
</tr>
<tr>
<td>Fuzzy Search</td>
<td>~10GB/sec</td>
<td>100-200 nodes(^2)</td>
</tr>
<tr>
<td>Term Frequency</td>
<td>~2.5GB/sec</td>
<td>100 nodes(^1)</td>
</tr>
</tbody>
</table>

http://www.datanami.com/2015/03/10/fpga-system-smokes-spark-on-streaming-analytics/
Architectural Enhancements

Copilot Integrity Protection (SEC 04)

Somniloquy (NSDI 09)
Why is now the right time?

**SoCs** with Programmable Logic coupled with ARM Cortex A9 (same as iPhone 4 and many other smartphones)

High-level Synthesis
Write C / C++ / SystemC / OpenCL code
Fundamental Problem:

Sharing the FPGA between applications
What we can already do

App loads: software runs on processor, FPGA configured with hardware
What we can already do

App loads: software runs on processor, FPGA configured with hardware

This is currently possible – run-time reconfiguration

Sort of
What we can’t do

What if we have two apps?

AppY
- AppY Software
- AppY Hardware

AppX
- AppX Software
- AppX Hardware

Processor
What we can’t do

**What if it’s a single chip** (and some I/O goes through the FPGA)
Why hasn’t this been solved before?

• Over a decade of research has proposed two main solutions:
  – Run-time place-and-route
  – Slot-based reconfiguration
Approach 1: Run-time Place/Route

- There is free space in the FPGA
- Place a new module there
Approach 1: Run-time Place/Route

- Routing can fail
- Routing is also very time consuming
- Therefore, is not practical
Approach 2: Slot-Based Reconfiguration

- Identical empty regions are reserved in FPGA
- Constrain tools to:
  - Not use wires/logic inside of slots
  - Use exact same wires for interface
Approach 2: Slot-Based Reconfiguration

- Hardware is loaded into slots
- Problem: if other logic exists, wire routing becomes very constrained
- Therefore, is also not practical
Previous Research

• Run-time Place and Route
  – Is very computationally expensive
  – Can possibly fail

• Slot-base Reconfiguration
  – Constrained routing is very restrictive and not applicable generally

• Therefore, previous research is not practical
Introducing Cloud RTR

- Allows for sharing of the FPGA between general apps
- Uses existing vendor technologies
- Adopts the idea of slots from previous research
- Cloud RTR makes existing vendor technology work for general apps
The App Deployment Model
• Creates a static design
  – All logic that does not change

• Design includes areas reserved for slots

• Sends this to the cloud compiler
• Create an app using existing tools

• Create a hardware definition in C

```c
bool example(ap_uint<32> *in, ap_uint<32> *out, bool *enabled,
            )
```
App Store (Cloud Compiler)

- Compiles hardware for each app
  - For each device variant
  - For each slot in each variant

[device1: 
  [slot1: a.bit, 
  slot2: b.bit, 
  slot3: c.bit]]

[device 2: 
  [slot1: d.bit, 
  slot2: e.bit]]
User (Operating System)

- A system service manages slots
- Downloaded apps include slot hardware
- The system service loads app hardware for apps

```
.apk:
[device 1:
 [slot1: a.bit,
  slot2: b.bit,
  slot3: c.bit]]
```
Security Considerations

• The slot manager enforces access to hardware

• However, FPGAs can theoretically directly access sensitive resources (while bypassing the OS)

• A secure loading system ensures that apps cannot access sensitive resources
Secure loading system

How does the secure loader work?

Diagram showing the secure loading system with the following components:
- Processor
- Memory Controller
- Operating System
- FPGA
- Slot 1
- Slot 2
- Signature Verification
- Reconfiguration Module
- ICAP
The OS wants to reconfigure Slot 1
The signature of the module is verified
Secure loading system

The module is written to the ICAP
Secure loading system

The ICAP performs the reconfiguration
Evaluation

• Is there value in apps with hardware?

• Is the cloud-based compilation of Cloud RTR practical?
Micro benchmark 1: QAM demodulator

4 orders of magnitude
Micro benchmark 2: AES

FPGA is 3x vs. OpenSSL
Micro benchmark 3: Memory Scanner

• We also implemented a hardware memory scanner

• It can scan the entire address space transparently to the OS
  – 2.7% memory read performance hit
  – 5.5% memory write performance hit

• We tested this using the LMbench testbench
**Brute-force compilation**

<table>
<thead>
<tr>
<th>Google Play Store Figures</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Apps as of Dec 14</td>
</tr>
<tr>
<td>1.43 Million</td>
</tr>
<tr>
<td>Average Monthly App Growth</td>
</tr>
<tr>
<td>6.10%</td>
</tr>
<tr>
<td># of Apps for January 16</td>
</tr>
<tr>
<td>117,521</td>
</tr>
</tbody>
</table>

provided by AppFigures.
### Brute-force compilation

<table>
<thead>
<tr>
<th>Max # of Apps Compiled per day</th>
<th>2 Slots Requirements</th>
<th>% of April Apps that use Hardware (# of Apps Uploaded per Day)</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Slots</td>
<td>Apps</td>
<td>0.1 (3) 1 (34) 10 (347)</td>
</tr>
<tr>
<td>2</td>
<td>121</td>
<td># of Device Variants # of Machines Required to Compile Apps</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
<td>1 1 3</td>
</tr>
<tr>
<td>4</td>
<td>76</td>
<td>10 1 3</td>
</tr>
<tr>
<td>5</td>
<td>59</td>
<td>10 3 29</td>
</tr>
<tr>
<td>6</td>
<td>51</td>
<td>100 29 288</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000 29 288 2875</td>
</tr>
</tbody>
</table>

Reasonable for most scenarios
Brute-force compilation

<table>
<thead>
<tr>
<th>Max # of Apps Compiled per day</th>
<th>6 Slots Requirements</th>
<th>% of April Apps that use Hardware (# of Apps Uploaded per Day)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.1 (3) 1 (34) 10 (347)</td>
</tr>
<tr>
<td># of Slots</td>
<td># of Device Variants</td>
<td># of Machines Required to Compile Apps</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1 1 7</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>1 7 69</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>7 69 681</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>69 681 6809</td>
</tr>
<tr>
<td>6</td>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

Still reasonable for most scenarios
Reducing the numbers even more

- Compilation can be offloaded to manufacturers
- Manufacturers will likely reuse designs (Qualcomm, ARM chips are often reused)
- Developers will likely use libraries
Implementation Case Study: Orbot

- Tor on Android
- AES is on the critical path
- Examine AES as an integration study
What we found:

• Memory operations are the bottleneck
  – Data must be placed correctly in memory
  – Userspace I/O has high overhead
  – Many system calls are incompatible with UIO

• It is easier to build an application from ground-up
Conclusion

• We have presented our vision of apps with hardware

• Cloud RTR implements our vision by leveraging the mobile app deployment model

• We have demonstrated the value and practicality of our vision
Questions?

- Email: michael.coughlin@colorado.edu
- Source code: https://github.com/nsr-colorado/cloud-rtr
Vendor Supported Partial Reconfiguration

Goal: Space saving for customer

Vendor tools

- base.bit
- partial_1.bit
- partial_2.bit

(Partial bitstreams work in 1 location, and are just for base.bit)
Examples of Libraries

- **Crypto**
  - Asymmetric (RSA, ECDSA, etc...)
  - Symmetric (3DES, Twofish, Blowfish)
- **Soft processors**
- **Encoding**
  - Network encoding (Reed-Solmon, etc...)
  - Media encoding (JPEG, MPEG, etc...)
- **DSP**
  - FFTs, Filters, etc...
bool example(ap_uint<32> *in
    ap_uint<32> *out,
    bool *enabled,
)
typedef ap_uint<32> uint32_t_hw;
typedef hls::stream<uint32_t_hw> mem_stream32;

bool aes(volatile unsigned int m_mm2s_ctl[500],
         volatile unsigned int m_s2mm_ctl[500],
         volatile unsigned sourceAddress,
         ap_uint<128> *key_in,
         ap_uint<128> *iv,
         volatile unsigned destinationAddress,
         unsigned int numBytes,
         int mode,
         mem_stream32& s_in,
         mem_stream32& s_out
"
The problem

Let’s examine the problem

Processor

FPGA

AppX software

AppX hardware

I/O

I/O
The problem

First, there are various interconnects needed
The problem

Control signals and logic must also be placed

Diagram:
- Processor
- FPGA
- AppX software
- AppX hardware
- I/O signals
The problem

The app may have complex inputs, or need to interact with other logic.
Secure loading system

- A trusted system is booted with Secure Boot

- Included is a static module that reconfigures slots

- This module only allows signed modules into slots that access sensitive resources
Our solution

• Builds off of prior research...

• ...but in a way that is compatible with vendor tools

• To do this, we leverage the deployment model for mobile apps