Making it Easier:
Improving Design Productivity

MARK HOROWITZ
Lessons Learned from the 80’s

DON’T TRY TO SOLVE THE PROBLEM

- The complete problem is much too large
  - No one has the resources to do it

THE PROBLEM IS REALLY MANY SMALLER PROBLEMS

- Need to find natural interfaces
  - Create tools and methodologies for subproblems
- Users can then pick build “tool” sets from this work

- Clean interfaces can be
  - Between different abstraction levels
  - Along domain lines
Themes

**WE ARE INTERESTED IN DOMAIN SPECIFIC TOOLS/LANGUAGES**

- Restricting application domain allows the tool to be more powerful
- Many of the tools are build around a domain

**WE ARE VERY INTERESTED IN BUILDING CONSTRUCTORS**

- If systems are expensive to build, want to reuse them
  - This means we want the systems to be flexible
- Rather than building a specific instance
  - Create a tool that understands local rules
  - And can create instances from parameters
APPLICATION FRAMEWORKS
Many recent IoT applications are based on an Embedded-Gateway-Cloud (MGC) pattern. Programming complete MCG-based applications for connected devices is complicated and requires expertise in different languages on each tier (e.g., Python, Java, C).
fabryq Solution

The Fabryq platform lets developers write web apps that can communicate with multiple Bluetooth Low Energy (BLE) devices.

Fabryq supports rapid MCG application prototyping by providing a reusable server and mobile “Smart Gateway”.
fabryq Results

Applications developed by pilot study participants and the research team demonstrated Fabryq’s utility.

Suggested a need for a smooth way to move from prototype to production.

Published at EICS ‘15
Ravel

A FRAMEWORK THAT IN SINGLE LANGUAGE SUPPORTS WRITING APPLICATIONS WITH A DISTRIBUTED MODEL VIEW CONTROLLER PATTERN IN A DATA FLOW ARCHITECTURE.

THE DEVELOPER WRITES A SINGLE MODEL WITH ONE DATA SCHEMA; RAVEL GENERATES ALL NECESSARY MECHANISMS FOR MODEL SYNCHRONIZATION ACROSS ALL TIERS OF THE IOT APPLICATION.
Leveraging Good Interfaces

existing iot frameworks simplifies development by providing a complete ecosystem for embedded, gateway, cloud, communications, and security. Consequently, the application and any future updates are tight up to that particular framework. Any investment in extending ecosystem and optimizing would not carry over to a different platform.

Instead of providing a runtime, Ravel translates code and uses templates to generate code for the target platforms. Such as Arduino, Intel, Nordic, Android, Amazon Cloud, Django, etc. Switching between supported frameworks and ecosystems becomes a push of a button.
Halide

**Language Created to Make Writing Image/Vision Apps Easier**

**Image Applications**
- Need to do tons of simple operations
- These operations have high locality
  - Can be done fast and efficiently

**Before Halide**
- Either wrote clean code
- Or wrote fast code that was hard to read/reuse

**Halide’s Big Idea**
- Separate schedule of code from function of code
Halide to Zynq FPGA

Halide App

- Halide Compiler
  - HW HLS C
  - HW Interface Config

- System Generator

- FPGA CAD tool
  - FPGA bitstream
  - Linux Drivers
  - Linux Program

- CPU/FPGA System (Zynq)

Existing CPU Back-end (Prototyping)

Heterogeneous Back-end (Implementation)

X86 Program

- C Testbench
  - C Compiler
  - HW Verification
Energy Results
HARDWARE PLATFORMS
WHILE SOLDERLESS BREADBOARDS ALLOW FOR QUICK CIRCUIT PROTOTYPING, NOVICE USERS CAN MAKE ERRORS THAT ARE DIFFICULT AND TIME-CONSUMING TO DIAGNOSE
Toastboard Solution

The Toastboard is an instrumented breadboard that can automatically sense the voltage on all of its rows.

It includes an extensible set of automated checker functions that can detect and explain circuit errors in a user-friendly way.
Toastboard Results

A PRELIMINARY EVALUATION DEMONSTRATED THAT THE TOASTBOARD CAN HELP NOVICE USERS CORRECT COMMON CIRCUIT ISSUES WITH CHECKER FUNCTIONS.
Imix:
Enabling Next-Generation Research for Secure IoT

STANDALONE IoT SYSTEM
- Crypto, wireless, power, sensors, software

MAXIMIZE SYSTEM VISIBILITY
- All internal signals exposed for debugging
- Power measurement header

USABILITY AND CONTROL
- Fully open source
- Software backbone: Tock
Imix: Features List

**Open Source**

**Two Wireless Protocols (802.15.4 and BLE)**

**Powered by USB, Li-ion Battery, or External Voltage Source**

**Integrated Battery Charger**

**Power Measurement Header for Each Subsystem**

**6-Axis Accelerometer/Magnetometer, Temperature Sensor, Light Sensor**

**Ultra Low Power ARM Cortex-M4 Processor with MPU**

**FTDI USB to Serial Link**

**Native USB**

**Arduino-Compatible**

**Compatible with Firestorm Bootloader**

**True Random Number Generator Based on Lampert Circuit**

**Runs Tock, a Secure Rust OS for IoT**
Going Meta: Embedded Device Generation

- **Compile high-level code into embedded hardware.**
- **Expand access to hardware development.**
- **Automate the development process for hobbyists**
- **Improve development tools for professionals.**

```java
// Component Declarations
component thermometer = new Thermometer(immersion, min-temp <= 0c, max-temp >= 100c);
component heater = new Heater(immersion, power > 10w);
component cooler = new Cooler(immersion);

// Control Logic
fn main()
while(true)
    if(thermometer.temp() < 20c){
        // Water temp too low
        cooler.off();
        heater.on();
    }
    else if(thermometer.temp() > 20c){
        // Water temp too high
        heater.off();
        cooler.on();
    }
    else {
        // Water temp just right
        heater.off();
        cooler.off();
    }
```
Brewing Beer at Home
Hobbyist wants a bespoke temperature controller.
They write some Application Logic.
Device Generation turns code into design files.
Hobbyist sends the design to a fabricator.
A box arrives in the mail ...
… with the finished device.
Overall Framework
# One Problem: How to Read Data Sheets

1. **There are a lot of components**
2. **Components are complicated**
3. **Existing search tools and online databases only store limited information**

**Embedded system designers need to choose between spending lots of time curating components and the optimality of their choices.**

<table>
<thead>
<tr>
<th>Component Family</th>
<th>Number of Entries on Parts.io</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connectors</td>
<td>35,835,399</td>
</tr>
<tr>
<td>Power Circuits</td>
<td>4,308,382</td>
</tr>
<tr>
<td>Diodes</td>
<td>2,200,889</td>
</tr>
<tr>
<td>Sensors/Transducers</td>
<td>1,429,819</td>
</tr>
<tr>
<td>Memories</td>
<td>1,319,651</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>818,329</td>
</tr>
<tr>
<td>Transistors</td>
<td>659,628</td>
</tr>
<tr>
<td>Drivers/Interfaces</td>
<td>121,955</td>
</tr>
<tr>
<td>Amplifiers</td>
<td>98,506</td>
</tr>
<tr>
<td>Transformers</td>
<td>86,021</td>
</tr>
</tbody>
</table>
Solution

CREATE A FRAMEWORK TO AUTOMATICALLY BUILD COMPONENT DATABASES FROM THEIR DATASHEETS. THIS COMPONENT LIBRARY WILL ENABLE A WAVE OF NEW APPLICATIONS THAT HAVE ACCESS TO DETAILED, ACCURATE COMPONENT INFORMATION.

• COMPONENT SEARCH

• DATASHEET ERROR CHECKING

• HARDWARE OPTIMIZATION

• EMBEDDED DEVICE GENERATION
### Results

#### Table Extractor

- **MAXIMUM RATINGS**
  - **Rating**
  - **Symbol**
  - **Value**
  - **Unit**

<table>
<thead>
<tr>
<th>Rating</th>
<th>Symbol</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collector - Emitter Voltage</td>
<td>$V_{CEO}$</td>
<td>65</td>
<td>Vdc</td>
</tr>
<tr>
<td>Collector - Base Voltage</td>
<td>$V_{CEO}$</td>
<td>80</td>
<td>Vdc</td>
</tr>
<tr>
<td>Emitter - Base Voltage</td>
<td>$V_{CEO}$</td>
<td>6.0</td>
<td>Vdc</td>
</tr>
<tr>
<td>Collector Current – Continuous</td>
<td>$I_C$</td>
<td>100</td>
<td>mAdc</td>
</tr>
<tr>
<td>Total Device Dissipation @ $T_A = 25^\circ$C</td>
<td>$P_D$</td>
<td>625</td>
<td>mW</td>
</tr>
<tr>
<td>Total Device Dissipation @ $T_J = 25^\circ$C</td>
<td>$P_D$</td>
<td>1.5</td>
<td>W</td>
</tr>
<tr>
<td>Operating and Storage Junction Temperature Range</td>
<td>$T_J$, $T_A$</td>
<td>-55 to +150</td>
<td>°C</td>
</tr>
</tbody>
</table>

#### THERMAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Symbol</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thermal Resistance, Junction-to-Ambient</td>
<td>$R_{JUA}$</td>
<td>200</td>
<td>°C/W</td>
</tr>
<tr>
<td>Thermal Resistance, Junction-to-Case</td>
<td>$R_{JUC}$</td>
<td>83.3</td>
<td>°C/W</td>
</tr>
</tbody>
</table>

Stresses exceeding Maximum Ratings may damage the device. Maximum Ratings are stress ratings only. Functional operation above the Recommended Operating Conditions is not implied. Extended exposure to stresses above the Recommended Operating Conditions may affect device reliability.

#### Process Cell Features

- **doc** | **part_num** | **storage_temp_min**
- X.pdf  | BC546       | -55     
- X.pdf  | BC547       | -55     
- X.pdf  | BC548       | -55     

- **Table Extractor**

- **Process Cell Features**
Validation
**Code Checkers**

**Tradition software checkers are complex software**

- Hard to write and add rules
Big Idea

IF YOU ARE GOING TO THROW STUFF AWAY, WHY PARSE IT IN THE FIRST PLACE

\[
A \rightarrow C \mid \ast \\
C \rightarrow \ast h \mid \ast k \mid f
\]