Embedded Device Generation

Turning Software into Hardware

Rohit Ramesh and Prabal Dutta

Speaker’s Notes:

- I’m Rohit Ramesh, a PhD Student at the University of Michigan
- I’ve been working on Embedded Device Generation with Prof. Prabal Dutta
Embedded Device Generation

1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

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

Speaker’s Notes:

- Our goal for device generation is to turn the problem of hardware development into a problem of software development.
- Phil talked about how building a web application used to be a task that took a dozen people a year, and now it’s a process which take two people a few months.
- We want to replicate that vast speedup for embedded hardware development.
- The goal is to build tools that automatically turn high-level application logic, expressed as code, into complete designs for embedded devices.
- This will allow hobbyists to build devices faster, with much less time spent learning the minutiae of embedded development.
- We also want to improve development tools for professionals, allowing them to work faster and more effectively.
Brewing Beer at Home

1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker’s Notes:

- So, I want you all to imagine someone whose hobby is brewing beer.
- They’ve got a simple benchtop setup, meant for really small batches.
Hobbyist wants a bespoke temperature controller.

1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker's Notes:

- Now, in order to make really good beer you need to control the temperature of the fermentation process precisely.
- Off the shelf temperature controllers are costly, and ludicrously overpowered for the job at hand.
- So our hobbyist decided to make their own temperature controller.
- The problem is they don’t know embedded development, and learning to develop hardware takes time and money they don’t have.
They write some Application Logic.

1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker’s Notes:

- However, they have an ace up their sleeve.
- They know how to program and have device generation tools handy.
- So they write a piece of code, describing what their temperature controller should do. The application logic.
Device Generation turns code into design files.

Speaker's Notes:

- Device generation tools can then take this code, and turn it into a complete design for the hardware and firmware of their temperature controller.
Hobbyist sends the design to a fabricator.

**Speaker's Notes:**

- The hobbyist can then send the design files to a fabricator like Seeed.
A box arrives in the mail ...

Speaker’s Notes:

- Who, a few days later, will send back a box ...
Speaker’s Notes:

- … that contains the device our hobbyist designed.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

---

Speaker's Notes:

- Our work, is on this portion, Embedded device generation.
- Which will allow people without embedded development skills to create an embedded devices that suit their individual needs.
Embedded Device Generation

1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker's Notes:

- The traditional embedded development starts with some specification for a device, usually pseudo-code or quick sketches, and turns that into the hardware and firmware for that device.
- This process takes a lot of knowledge, effort, and time to get right.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker’s Notes:

- Device generation tries to achieve the same outcome through a somewhat different path.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker’s Notes :

- At its core, device generation has a formalism.
- A rigorous formal model of embedded development that allows us to reason about device designs algorithmically.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker's Notes:

- Likewise, there is also a library of components, that provide the pool of parts used during device generation.
- It captures the information that modern developers have to collate from datasheets, manuals, and numerous other sources.
Speaker’s Notes:

- Together, the formalism and component library allow us to turn the hard problem of embedded development into a problem that’s much easier to solve automatically.
- There’s three steps to the device generation process.
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker's Notes:

- Analysis, which turns the high-level application logic a user writes into a specification within the formalism.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker's Notes :

- Synthesis, which works within the formal domain to turn that specification into a formal schematic for the device.
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker's Notes:

- and, Reification, which turns formal schematic back into the same design files people use today.
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker's Notes:

- I want to take a deeper look at this process, starting with Analysis.
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

---

```javascript
// 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();
        }
    }
}
```

---

**Speaker's Notes:**

- This, is what we expect the code for a simple temperature controller to look like.
- It consists of two major sections, the component declarations and control logic.
Speaker's Notes:

- The component declarations let the user specify what components the device must have.
- The thing is, people don’t usually care exactly which thermometer or heater their device uses. They care that they can measure the temperature and heat a liquid.
- In this case, the user cares to measure the temperature underwater, and over a particular range of values.
- So we allow them to say that they need an immersion thermometer, with a range of between 0 and 100 degrees.
- This means that device generation tools can do the specific work of choosing a thermometer, a heater, or a cooler, for the user.
1. Introduction
a. Goals
b. Workflow

2. Description
a. Overview
b. Analysis
c. Synthesis
d. Reification

3. Conclusion

Speaker’s Notes:

- The control logic, here, looks much like modern embedded code.
- Albeit without specific library declarations, initialization blocks, and other hardware-specific components.
- Again, this given device generation the freedom to make those choices automatically.
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

---

Speaker's Notes:

- Together, the component declaration and control logic are the application logic for a device.
- They form a specification for the device as a whole, telling what the major components are and what they do during runtime.
- The cool thing about this is that the application logic is also a component *within* the device.

```rust
// 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();
      }
   }
}
```
### 1. Introduction

a. Goals
b. Workflow

### 2. Description

a. Overview
b. Analysis
c. Synthesis
d. Reification

### 3. Conclusion

---

Speaker’s Notes:

- If, during runtime, `thermometer.temp()` and `cooler.on()` read from an actual thermometer and turn on an actual cooler, then the device as a whole will work.
- Or at the very least satisfy the specification.
- This logic then can be viewed as a software component, one with a set of needs.

```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();
        }
    }
}
```
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

---

Speaker’s Notes:

- In particular, access to a thermometer, heater, and cooler.
- If this software is part of the device and these needs are met, then the device as a whole will satisfy the specification.
- This component is, in fact, the formal version of the specification.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker's Notes:

- With this we can move to synthesis, and turn that formal specification into a formal schematic for an tire device.
Start with the Application Logic.

**Speaker’s Notes:**

- In the beginning our design consists of only the application logic.
Each component tells us about the device around it.

**Speaker’s Notes:**

- But the application logic tells us a lot about the device which it’s a part of.
Application Logic implies a microcontroller.

Speaker’s Notes:

- Being software, it tells us that it needs to be run on a microcontroller.
Speaker’s Notes:

- The various needs, tell us that the device must have peripherals with various properties.
- It tells us these peripherals have a software portion, and a hardware portion, with some, as yet unknown, bridge between software and hardware.
The order in which choices are made matters.

**Speaker's Notes:**

- From here there are number of choices synthesis tools can make.
- In fact synthesis is effectively a search for a set of design choices that lead to a complete device.
- And there are multiple choices that can be made right now.
- We can look the the component library to find versions of any of these parts.
- But the order we make our choices matters.
- At the moment we have a large number of microcontrollers to choose from, in contrast there's not as many thermometers.
- So let's try using a thermometer from the component library.
Sometimes choices don’t work ...

Speaker’s Notes:

- This first attempt ended up failing.
- The thermometer we chose didn’t cover the entire range of specified temperatures.
- The formalism gives us the tools to automatically detect mismatches like this and backtrack, so we can try another thermometer.
... and sometimes they do.

Speaker’s Notes:

- Like this one, which fulfills our spec.
- But we’ve also learned two new thing about our device from this choice.
New information creates new constraints.

Speaker’s Notes:

- Now that we’ve chosen a thermometer, we know that it communicates over an SPI bus.
- This means that we narrow the space of microcontrollers down a little more, removing every microcontroller without an SPI bus from the set of options.
New requirements appear that are a byproduct of design choices.

**Speaker’s Notes:**

- We’ve also created a new need for our device, a power supply.
- This is something that our device must have but which wasn’t a direct byproduct of the application logic.
Speaker’s Notes:

- Now we can choose a heater and cooler.
- Unlike the thermometer, our search has chosen to have the heater and cooler share a bus.
- There is enough information for our synthesis process to understand that an I2C bus can support multiple devices, and that these two components can share a bus.
- We know what busses the microcontroller must support, further reducing the space of choices.
Added constraints make search more tractable.

Speaker’s Notes:

- Collecting constraints like this allow us to make the search tractable.
- We can order choices, use additional metadata like cost or power consumption to make the synthesis process tractable and guide it towards solutions that prioritize properties we care about. (Speed, cost, etc…)
Added constraints make search more tractable.

Speaker's Notes:

- Now, we can finish out the design by adding a power supply.
No more unfulfilled requirements, the formal schematic is complete.

**Speaker's Notes:**

- This leaves us with a schematic where no components have unfulfilled needs, and a complete design where each component has the resources it needs to function.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker's Notes:

- With a complete schematic, we can move on to reification and transform that formal version of the design into the concrete hardware and firmware we need to build a device.
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker’s Notes:

- Let’s start with the software.
- Each of these components live within the software running on the microcontroller.
- Each of these components come from the component library, which contains both the formal representation of a piece of software, as well as the embedded C implementation.
Embedded Device Generation

1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker’s Notes:

- We can combine these implementations together, using information from the schematic to choose settings, wire together interfaces, and create the complete firmware needed for the device.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker’s Notes:

- Generating the hardware is very similar.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification
3. Conclusion

Speaker’s Notes:

- The connections between components in the formal schematic can be converted into an electronic schematic.
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker’s Notes:

- Creating a layout from the schematic can in three ways:
  - Completely automatic: Auto-place and auto-route tools generate the entire PCB layouts.
  - Completely manual: The user does everything.
  - Mixed initiative: The user chooses the important portion “This display goes here, these buttons go here” and everything else is done automatically.
1. Introduction
   a. Goals
   b. Workflow
2. Description
   a. Overview
   b. Analysis
   c. Synthesis
d. Reification
3. Conclusion

Speaker’s Notes:

- And that’s how we automate the entire embedded design process.
Conclusion

- Currently working on an early prototype.
- Integration with existing developer tool.
- Device generation will make embedded development faster, cheaper, and more accessible.
- We want your input.

Speaker’s Notes:

- Right now we’re working on an early implementation.
  - One that focuses on relatively simple devices,
  - which we can use to locate bottlenecks in the device generation process
  - Improve our formalisms
- We are also looking at how device generation can be incorporated into existing development tools.
  - The formalism can be used for verification, catching many common errors, and we want to bring those tools to developers as well.
- We think embedded device generation will be very impactful.
  - Useful for people at many different skill levels doing many different tasks.
  - Hobbyists building bespoke gear.
  - Scientists designing experimental apparatus cheaply and quickly.
  - Professionals working faster, making fewer mistakes.
- We also want your input.
  - We are still in very early development, and
  - we want to know what you think.
- What you would like to come out of our work.
- Where you believe our focus should be.
Questions?

Rohit Ramesh
rohitram@umich.edu
Backup Slides
1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
d. Reification

3. Conclusion

Speaker's Notes:

Reader's Notes:
- Foo
Embedded Device Generation

1. Introduction
   a. Goals
   b. Workflow

2. Description
   a. Overview
   b. Analysis
   c. Synthesis
   d. Reification

3. Conclusion

Speaker's Notes:

- 

Reader's Notes: