Turning Coders into Makers: The Promise of Embedded Design Generation

Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, Prabal Dutta

SITP Retreat 2017
Embedded Development is a key part of the maker movement.
Embedded Development is a key part of the personal fabrication movement.

More accessible and usable parts.
Embedded Development is a key part of the personal fabrication movement.

More accessible and usable parts.

Cheaper board fabrication.
Embedded Development is a key part of the personal fabrication movement. More accessible and usable parts. Cheaper board fabrication. Low entry requirements.
Embedded Development is a key part of the personal fabrication movement.

More accessible and usable parts.

Design is still a major bottleneck!

Cheaper board fabrication.

Low entry requirements.
We want to go from idea to finished device.

When the button is pressed, the light should blink.
Designing the circuit takes significant domain knowledge.

When the button is pressed, the light should blink.

Idea

Netlist-level Schematic
Designing the circuit takes significant domain knowledge.

- What should the circuit have in it?

When the button is pressed, the light should blink.
Designing the circuit takes significant domain knowledge.

When the button is pressed, the light should blink.

- What should the circuit have in it?
- What do I need to check in order to verify part compatibility?

- What I found challenging was to understand what are the components that have to be paired with the basic components that you are buying.

- Luisa, closing discussion [Mellis et al. 2016]

We also observed participants struggling to coordinate information from diverse sources, [Mellis et al. 2016]

When the button is pressed, the light should blink.
Designing the circuit takes significant domain knowledge.

What should the circuit have in it?

What do I need to check in order to verify part compatibility?

How do I even figure out what parts I want to use?

When the button is pressed, the light should blink.

Idea

Design

Verification

Part Selection

Netlist-level Schematic
This domain knowledge is often hard for novices to acquire.

“When I found challenging was to understand what are the components that have to be paired with the basic components that you are buying.”

- Luisa, closing discussion

[Mellis et al. 2016]

When the button is pressed, the light should blink.

**Diagram:**

- **Idea:**
  - Button press
  - Light turn on

- **Design:**
  - Resistance

- **Verification:**
  - $V_f = 1.2V$
  - $I_{th} \leq 20mA$

- **Part Selection:**
  - LED

- **Netlist-level Schematic:**
  - Arduino
  - 3.3V
  - D0
  - D1
  - GND
Automation makes PCB design much easier.

“The auto-router was magical. It was cool watching all the wires go bo-bo-bo-bop.”

- Tyler [Mellis et al. 2016]
# Working with the firmware is just coding.

```cpp
#import "arduino.h"
#import "button_driver.h"

BUTTON button;
GPIOLED light;

void setup() {
    button.init(GPIO3);
    light.init(GPIO4);
}

void loop() {
    if(button.isPressed()){
        light.toggle();
    } else {
        light.off();
    }
    delay(1000); //milliseconds
}
```

- The circuit has to be designed first so we have information about connectivity and interfaces.
- Arduino and other frameworks wrap low-level abstractions in nice, comprehensible interfaces.

---

**Embedded Firmware**

**Debug and deploy firmware**

**Completed Board**
We want to minimize the amount of knowledge needed to design electronics.
We want to minimize the amount of knowledge needed to design electronics.

Anyone who can program should be able to design an embedded device

- There are many more programmers than electrical engineers.
- This applies for professionals and hobbyists.
Embedded firmware requires domain knowledge to write.

```cpp
#include "arduino.h"
#include "button_driver.h"

BUTTON button;
GPIOLED light;

void setup() {
  button.init(GPIO3);
  light.init(GPIO4);
}

void loop() {
  if(button.isPressed()){
    light.toggle();
  } else {
    light.off();
  }
  delay(1000); //milliseconds
}
```

Requires finished circuit diagram, and the knowledge to create it.

Captures the function of the device at a high level.
We can make a version that doesn't need that domain knowledge.

Annotated Code
A mockup of the input format for EDG, designed to allow people to specify devices without needing to know electrical engineers.

```c
peripheral button = new MomentarySwitch();

peripheral light = new LED(color = Red);

void loop() {
  if(button.isPressed()){
    light.toggle();
  } else {
    light.off();
  }
  delay(1000); //milliseconds
}
```

Ask for properties that matter to you, don’t waste time picking parts.

Captures the function of the device at a high level.
EDG takes annotated code and a library of parts and produces SW and HW.

```c
import "arduino.h"
import "button_driver.h"

// Import libraries

BUTTON button;
GPIOLED light;

void setup()
{
  button.init(GPIO3);
  light.init(GPIO4);
}

void loop()
{
  if(button.isPressed())
    light.toggle();
  else
    light.off();
  delay(1000); //milliseconds
}
```

```c
// Annotated code

peripheral button = new MomentarySwitch();
peripheral light = new LED(color = blue);

void loop()
{
  if(button.isPressed())
    light.toggle();
  else
    light.off();
  delay(1000); //milliseconds
}
```

EDG-Based Dev Tool

Automatic Step → Manual Step →

Describe Hardware →

Datasheets → Library Creation →

Annotated Code → Parse →

Design Instantiation →

Deploy and Debug → Completed Board

Automatic Step

Manual Step

EDG takes annotated code and a library of parts and produces SW and HW.

EDG takes annotated code and a library of parts and produces SW and HW.
Embedded design generation uses constraint solving tools to do this.

Annotated Code

```c
#pragma edg
led(red)
```

Datasheets

Embedded Firmware

```c
void loop {
  led = 1;
}
```

Netlist-level Schematic
Embedded design generation uses constraint solving tools to do this.

Space of Possible Designs

Annotated Code

```c
#pragma edg
led(red)
```

Datasheets

Embedded Firmware

```c
void loop {
  led = 1;
}
```

Netlist-level Schematic

Arduino

+3.3V

D0
D1
...
GND
Embedded design generation uses constraint solving tools to do this.

Space of Possible Designs

Annotated Code

```
#pragma edg
led(red)
```

Datasheets

Portion of designs that satisfy specification.

Embedded Firmware

```
void loop {
    led = 1;
}
```

Netlist-level Schematic

Arduino

+3.3V

DO

D1

GND
Embedded design generation uses constraint solving tools to do this.

Portion of designs that satisfy specification.
Embedded design generation uses constraint solving tools to do this.

Space of Possible Designs

Annotated Code

Datasheets

Portion of designs that satisfy specification.

Embedded Firmware

void loop {
  led = 1;
}

Arduino

+3.3V

D0

D1

... GND

Netlist-level Schematic
```c
import "arduino.h"
import "button_driver.h"

BUTTON button;
GPIOLED light;

void setup() {
    button.init(GPIO3);
    light.init(GPIO4);
}

void loop() {
    if(button.isPressed()){
        light.toggle();
    } else {
        light.off();
    }
    delay(1000); //milliseconds
}
```

```c
peripheral button = new MomentarySwitch();
peripheral light = new LED(color = blue);

void loop() {
    if(button.isPressed()){
        light.toggle();
    } else {
        light.off();
    }
    delay(1000); //milliseconds
}
```
Outline

- The EDG Methodology
- Proof-of-Concept and Evaluation
- Discussion and Future Work
The EDG Methodology
How do we represent the parts our tool has available?

We represent hardware and firmware components as blocks in a block diagram.
Blocks are made up of a number of sub elements.
Blocks are made up of a number of sub elements.

**Ports**

- **Digital Signal Input**
- **User Facing Button API and Hardware API**
Blocks are made up of a number of sub elements.

Ports

Implementation

Digital Signal Input

User Facing Button API and Hardware API

Code that wraps the HW API in a nice interface.
Blocks are made up of a number of sub elements.

**Ports**
- Digital Signal Input

**Implementation**
- LED Color, Input Voltage

**Metadata**
- User Facing Button API and Hardware API
- Code that wraps the HW API in a nice interface.
- API use and connectivity
Blocks are made up of a number of sub elements.

**Ports**
- LED
- Digital Signal Input

**Implementation**
- Button API
- Button Driver
- GPIO API
- Code that wraps the HW API in a nice interface.

**Metadata**
- LED Color, Input Voltage

**Conditions**
- User Facing Button API and Hardware API
- API use and connectivity
Conditions allow us to specify the operating conditions of a component.
Conditions allow us to specify the operating conditions of a component.
Conditions allow us to specify the operating conditions of a component.

LiPo Battery

LED

Mains Power

Input Voltage = 3.7v

Metadata

Input Voltage = 120v
Conditions allow us to specify the operating conditions of a component.

- **LiPo Battery**
  - Input Voltage = 3.7v
  - Input voltage is **above** 1.2v and **below** 5v

- **LED**

- **Mains Power**
  - Input Voltage = 120v
  - Input voltage is **above** 1.2v and **below** 5v

**Metadata**
Conditions encode the functional correctness of a block.

**Ports**
- Digital Signal Input

**Implementation**
- LED Color, Input Voltage
- Code that wraps the HW API in a nice interface.
  - User Facing Button API and Hardware API

**Metadata**
- LED Color, Input Voltage
- API use and connectivity

**Conditions**
- Input voltage is above 1.2v and below 5v
- If button API is used, then GPIO API must be active
The annotated code has to actually run on the device, and can be in a block.

```java
peripheral button = new MomentarySwitch();

peripheral light = new LED(color = Red);

void loop() {
    if(button.isPressed()){
        light.toggle();
    } else {
        light.off();
    }
    delay(1000); //milliseconds
}
```
The annotated code has to actually run on the device, and can be in a block.

Control Logic:
Block created by parsing annotated code, which will eventually run on the device.

```java
peripheral button = new MomentarySwitch();

peripheral light = new LED(color = Red);

void loop() {
  if(button.isPressed()){
    light.toggle();
  } else {
    light.off();
  }
  delay(1000); //milliseconds
}
```
The annotated code has to actually run on the device, and can be in a block.

```java
   peripheral button = new MomentarySwitch();
   peripheral light = new LED(color = Red);

   void loop() {
     if(button.isPressed()){
       light.toggle();
     } else {
       light.off();
     }
     delay(1000); //milliseconds
   }
```
The annotated code has to actually run on the device, and can be in a block.

Control Logic:
Block created by parsing annotated code, which will eventually run on the device.

```
peripheral button = new MomentarySwitch();
peripheral light = new LED(color = Red);
void loop() {
    if(button.isPressed()){
        light.toggle();
    } else {
        light.off();
    }
    delay(1000); //milliseconds
}
```
The annotated code has to actually run on the device, and can be in a block.

```java
peripheral button = new MomentarySwitch();
peripheral light = new LED(color = Red);

void loop() {
    if(button.isPressed()){
        light.toggle();
    } else {
        light.off();
    }
    delay(1000); //milliseconds
}
```

**Control Logic:**
Block created by parsing annotated code, which will eventually run on the device.

**Implementation**
Button API

**Conditions**
Both the ports must be provided a valid api to the corresponding physical component.
Solvers find variable assignments such that some set of constraints hold;

**Variables**

- A < 12
- B = 3 + A
- C > 0
- C = 3 - A

**Constraints**

- A = 2
- B = 5
- C = 1

**Solution**

- The solver finds *one* assignment such that every constraint is met.
- The solver is nondeterministic and might output any of the valid assignments.
We can combine every block in the library, and all possible potential links.
A pegboard connection is a block diagram
Mix with block conditions and control logic, and bake

- **A** < 12
- **B** = 3 + **A**
- **C** > 0
- **C** = 3 - **A**

The solver finds one assignment such that every constraint is met.

The solver is nondeterministic and might output any of the valid assignments.
We can instantiate a design from the solver output
We can instantiate a design from the solver output
We have gone from a high-level specification and a set of parts, to a design that can be realized.
Proof-of-Concept and Evaluation
Our prototype translates the EDG methodology onto available tools.

- The tool is built with Haskell on top of the Z3 constraint solver.
- Parsing and design instantiation is still manual to allow for faster iteration on the system internals.
Our test library has a wide variety of parts

Images from sparkfun.com, under a CC-BY 2.0 license
Can we generate designs at all?

```java
peripheral button = new MomentarySwitch();

peripheral light = new LED(color = Red);

void loop() {
    if(button.isPressed()){
        light.toggle();
    } else {
        light.off();
    }
    delay(1000); //milliseconds
}
```
Can we generate designs at all? Yes!

```java
peripheral button = new MomentarySwitch();
peripheral light = new LED(color = Red);

void loop() {
  if(button.isPressed()) {
    light.toggle();
  } else {
    light.off();
  }
  delay(1000); //milliseconds
}
```
We use symmetries to collapse the rest of our diagrams further.
We use symmetries to collapse the rest of our diagrams further.
Can the user specify exact parts? Handle complex power systems?

AdafruitChassisMotor motorL();
AdafruitChassisMotor motorR = duplicate motorL;
ReflectanceSensor lineSenseL(distance >= 1cm);
ReflectanceSensor lineSenseR = duplicate lineSenseL;
Battery power(duration >= 10mins);

const int baseSpeed = 100;
const int baseTurn = 25;
const int lightThreshold = 70;

int loop(){
    static int differential = 0;
    ...
}
AdafruitChassisMotor motorL();
AdafruitChassisMotor motorR = duplicate motorL;
ReflectanceSensor lineSenseL(distance >= 1cm);
ReflectanceSensor lineSenseR = duplicate lineSenseL;
Battery power(duration >= 10mins);

const int baseSpeed = 100;
const int baseTurn = 25;
const int lightThreshold = 70;

int loop(){
  static int differential = 0;
  ...
}
Can the user specify exact parts? Handle complex power systems?
Can the user specify exact parts? Handle complex power systems? Yes
Does our system preserve functionality even when the library changes?

- 4 colored, light-up buttons
- Play a pattern, ask the user to press the buttons in the same order.
Does our system preserve functionality even when the library changes?
Does our system preserve functionality even when the library changes?
Does our system preserve functionality even when the library changes?

Power amplifiers added to deal with complex power system.
Does our system preserve functionality even when the library changes? Yes
We also built other designs, including a datalogger and control system.
Performance is decent for small problem sizes.

For simple designs, our tool has taken around 3 hours in the most realistic tests. Mind, that’s still time you could be doing something else.
Discussion and Future Work
There are many things that need further research.

- Performance scales exponentially, and will never be a lower complexity class.
  - But there’s still immense room for optimization. Including library pruning, and more efficient encoding of constraints.
There are many things that need further research.

- Performance scales exponentially, and will never be a lower complexity class.
  - But there’s still immense room for optimization. Including library pruning, and more efficient encoding of constraints.

- We need some way for the user to figure out what their specifications actually mean, especially for large classes of wildly different parts.
There are many things that need further research.

- Performance scales exponentially, and will never be a lower complexity class.
  - But there’s still immense room for optimization. Including library pruning, and more efficient encoding of constraints.

- We need some way for the user to figure out what their specifications actually mean, especially for large classes of wildly different parts.

- We need to do research on making our system more expressive, as of now it cannot handle things like timing requirements or isolated power domains.
EDG can help automate optimization
EDG can help automate optimization
EDG can help automate optimization
EDG can help automate optimization
EDG can help automate optimization
EDG can help automate optimization
Questions?