Bifröst

Visualizing and Checking Behavior of Embedded Systems across Hardware and Software

Will McGrath – wmcgrath@stanford.edu
Jeremy Warner – jeremy.warner@berkeley.edu
eMbedded Gateway Cloud [MGC]
eMbedded Gateway Cloud [MGC]

- embedded
- gateway
- cloud
Program logic bug?
Program logic bug?
Peripheral not configured?
Protocol error?
Wiring issue?
Sensor data out of range?

Program logic bug?

Peripheral not configured?

Protocol error?

Wiring issue?
Sensor data out of range?

Defective part?

Peripheral not configured?

Program logic bug?

Protocol error?

Wiring issue?
Toast Board

Measures analog voltages at all breadboard rows

Compares measured results to intended circuit to identify errors

No embedded software support
// variables will change:
int buttonState = 0; // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
    Serial.println("LED ON!");
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
    Serial.println("LED OFF!");
  }
}
```cpp
// variables will change:
int buttonState = 0; // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
    Serial.println("LED ON!");
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
    Serial.println("LED OFF!");
  }
}
```
William-McGraths-MacBook-Pro:~ wmcgrath\$ salamigdb
GNU gdb (GNU Tools for ARM Embedded Processors) 7.6.0.20140228-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-apple-darwin10 --target=arm-none-eabi".
For bug reporting instructions, please see:
0x000000858 in ?? ()
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x81000000 pc: 0x000000858 msp: 0x200023a0
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x81000000 pc: 0x000000858 msp: 0x200023a0
(gdb) b modfile.ino:27
Breakpoint 1 at 0x2164: file modfile.ino, line 27.
(gdb)
Existing Embedded Debugging Techniques

• Multimeter
  • Point to point
  • Obscures rapid changes

• Oscilloscope/Logic Analyzer
  • Fast, but expensive
  • Nontrivial configuration

• Print Debugging
  • Easy, but weak time information

• GDB
  • Powerful, but not always available
  • Complex configuration
In contrast with prior work which has focused on providing tools only for debugging software or circuits, we directly address the interconnected nature of embedded systems projects.
We believe that giving developers insight into **how hardware and software interact** in a richly-linked environment will enable more efficient debugging of embedded systems.
Software

```cpp
// variables will change
let buttonState = 0; // variable for reading the pushbutton status

void setup() { // initialize the LED pin as an output:
  pinMode(buttonPin, OUTPUT);
  digitalWrite(buttonPin, LOW);
}

void loop() { // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH) { // if it is, the buttonState is HIGH:
    digitalWrite(buttonPin, HIGH); // turn LED on:
    Serial.println("LED ON!");
  } else { // turn LED off:
    digitalWrite(buttonPin, LOW);
    Serial.println("LED OFF!");
  }
}
} // end of program
```

Hardware

Sensors

Actuators
Bifröst

```c
void setup() {
  pinMode(LedPin, OUTPUT);
  pinMode(potPin, INPUT);
  // For debug only:
  Serial.begin(3600);
}

void loop() {
  counter++;
  counter = counter % 15;
  int potVal = analogRead(potPin);
  if(potVal > 500) {
    digitalWrite(LedPin, HIGH);
  } else {
    digitalWrite(LedPin, LOW);
  }
  delay(10);
  if(counter > 10) {
    Serial.println(counter);
  }
}
```
Bifröst: IDE Overview

Digital Signals

Analog Signals

Variable Values

User Program

Code line at current time

Time-linked console
Instrumentation  Capture  Analysis
Instrumentation

Capture

Analysis

Hardware Instrumentation:
- Digital signals
- Analog signals

Software Instrumentation:
- Line numbers
- Variable values

Analysis
**Hardware Instrumentation:**
- Digital signals
- Analog signals

**Software Instrumentation:**
- Line numbers
- Variable values

**Capture**

**Analysis**

- Record with Logic Analyzer

---

**BiFröst**

---

**saleae**
**Hardware Instrumentation:**
- Digital signals
- Analog signals

**Software Instrumentation:**
- Line numbers
- Variable values

**Capture**
- Record with Logic Analyzer

**Analysis**
- Linked Visualization
  - Signals
  - Code
  - Console

**Automatic & User Authored Checks**
Bifröst: usage example
Bifröst: **configuration**
Bifröst: blinking LED
Bifröst: recording data
Bifröst: variable tracing
Bifröst: **trace ⇔ code linkage**
Bifröst: code-based navigation
Bifröst: console-based navigation
Data Capture

bifröst

```cpp
// pot.ino - control LED blinking speed with potentiometer
// (c) BotBook.com - Karvinen, Karvinen, Valtokari
int potPin= A0;
int ledPin= 13;
int x= 0; // 0..1023
int d= 0;
void setup() {
    pinMode(ledPin, OUTPUT);
}
void loop() {
    x= analogRead(potPin);
    d = x / 5;
    digitalWrite(ledPin,HIGH);
    delay(d);
    digitalWrite(ledPin, LOW);
    delay(d);
    if(x<340) {
        d=50;
    } else if(x<680) {
```

- Errors
- Warnings
- Passed
- digitalWrite to Hardware State Change
  - Line 15
    - t=0.000492375
  - t=0.360593125
  - t=0.72079025
  - Line 17

```cpp
```
Capture: setup
Capture: **setup**

- Device
- Configuration
- Instrumented Code
- Backend
- Program Code
- **bifröst**
Capture: execution

Debugging information: code lines + function info
Capture: **execution**

- **Trace (w/ debug info) + decoded protocols**
- **Debugging information:** *code lines + function info*
- **Backend**

![Diagram](image-url)
Capture: **execution**

- Trace (w/ debug info) + decoded protocols
- Backend
- Debugging information: *code lines + function info*
- bifröst

Capture: **execution**

- Trace (w/ debug info) + decoded protocols
- Backend
- Debugging information: *code lines + function info*
- bifröst
Capture: **inspection**

incrementally *step, record, and visualize*
Code Instrumentation

```c
void readRegister(char registerAddress, int numBytes, signed char * values)
{
    // Since we're performing a read operation, the most significant bit of the register address should be set.
    char address = 0x80 | registerAddress;
    // If we're doing a multi-byte read, bit 6 needs to be set as well.
    if(numBytes > 1) address = address | 0x40;

    // Set the Chip select pin low to start an SPI packet.
    digitalWrite(CS, LOW);
    // Transfer the starting register address that needs to be read.
    SPI.transfer(address);
    // Continue to read registers until we've read the number specified, storing the results to the input buffer.
    for(int i=0; i<numBytes; i++)
    {
        values[i] = SPI.transfer(0x00);
    }
    // Set the Chips Select pin high to end the SPI packet.
    digitalWrite(CS, HIGH);
}
```
void readRegister(char registerAddress, int numBytes, signed char * values){
  //Since we're performing a read operation, the most significant bit of the register address should be set.
  Serial.write(45);
  char address = 0x80 | registerAddress;
  //If we're doing a multi-byte read, bit 6 needs to be set as well.
  Serial.write(47);
  if(numBytes > 1) address = address | 0x40;

  //Set the Chip select pin low to start an SPI packet.
  Serial.write(50);
  digitalWrite(CS, LOW);
  //Transfer the starting register address that needs to be read.
  Serial.write(52);
  SPI.transfer(address);
  //Continue to read registers until we've read the number specified, storing the results to the input buffer.
  for(int i=0; i<numBytes; i++){
    Serial.write(55);
    values[i] = SPI.transfer(0x00);
  }
  //Set the Chips Select pin high to end the SPI packet.
  Serial.write(58);
  digitalWrite(CS, HIGH);
}
void digitalWrite( uint32_t ulPin, uint32_t ulVal )
{
    Serial1.write(193); // Opcode for digitalWrite
    Serial1.write(ulPin); // Pin
    Serial1.write(ulVal); // Output Value

    // Handle the case the pin isn't usable as PIO
    if ( g_APinDescription[ulPin].ulPinType == PIO_NOT_A_PIN )
    {
        return;
    }

    // Enable pull-up resistor
    PORT->Group[g_APinDescription[ulPin].ulPort].PINCFG[g_APinDescription...
Checks

automatically inferred

user-parameterized
Checks: automatically inferred

“Writes should generate activity on a pin”

“Communication protocols adherence”

### Status

<table>
<thead>
<tr>
<th>Console</th>
<th>Checks</th>
<th>About</th>
</tr>
</thead>
</table>

#### + blinkyLED/D4 – Software Write to Hardware State Change
Your microcontroller wrote a value to a pin, but the pin did not change to that value. You should check whether the pin has the correct pinMode setup in the `setup()` function, and that there is no other connection to the pin which would stop it from changing.

#### - Warnings

1

#### + buttonLED/D5 – Software Write to Hardware State Change
Pin was already set to the specified value when written to.

#### - Passed

3

#### + buttonLED/D5 – Pin Mode Initialization Before Use OK

53
Checks: **user-parameterized**

"when D1 rises expect line 11 in 100 ms"
Checks: user-parameterized

"when D1 rises expect line 11 in 100 ms"
Checks: **user-parameterized**

"when D1 rises expect line 11 in 100 ms"

*pin* \(\rightarrow\) \textbf{rises} \(\rightarrow\) *event*
Checks: **user-parameterized**

"when D1 rises **expect line 11** in 100 ms"

*pin*  *event*  *source code*
Checks: \texttt{user-parameterized}

"when D1 rises expect line 11 in 100 ms"

\textit{pin \ event \ source code \ timing}
Checks: **user-parameterized**

![Diagram: Hardware → Trigger → Software](image)

“When D1 rises, expect Line 17 in 20 ms”
Checks: **user-parameterized**

Hardware → **Trigger** → Software

“When D1 rises, expect Line 17 in 20 ms”
Checks: **user-parameterized**

**Hardware** → **Trigger** → **Software**

“When D1 rises, expect Line 17 in 20 ms”
Evaluation

How effectively does Bifröst aid embedded debugging?
Evaluation: **issues**

Users identified issues such as:

- pin misconfiguration
- thresholding issues
- switch bouncing
- timing issues
Evaluation: strategies

Users identified these issues using:

- variable/signal data comparison
- code statement execution monitoring
- automatic and user-inferred checks
Limitations

**performance:**
instrumentation overhead  
(full speed to under 10x)

**scope:**
only instruments user code  
and Arduino core functions
Future Work

- Support for continuous capture
- Support for multiple devices
- Checking & visualizing networked behavior
Thanks!

Will McGrath – wmcgrath@stanford.edu
Jeremy Warner – jeremy.warner@berkeley.edu

With: Daniel Drew, Mitchell Karchemsky, Majeed Kazemitabaar, David Mellis, and Björn Hartmann