Verifying Embedded Graphics Libraries leveraging Virtual Prototypes and Metamorphic Testing

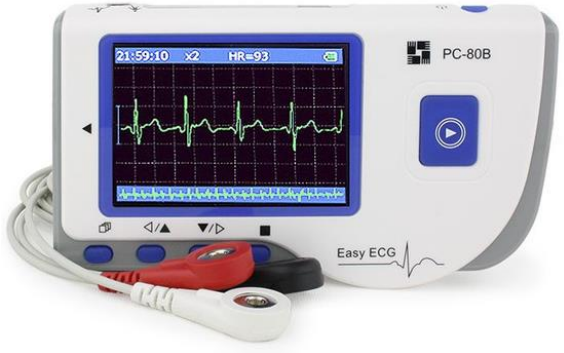
Christoph Hazott, Florian Stögmüller, Daniel Große
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: christoph.hazott@jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Motivation

- **Embedded SW**
  - Closely tied to the HW it runs on
  - FW libraries for low-level control of HW

- **Embedded Graphics Libraries**
  - Offer functionality to draw on embedded displays
  - Complex interaction (full SW-to-HW stack)
  - Challenges in verification
    - Physical HW
    - Test oracle
    - Automation



Agenda

- Virtual Prototypes, TFT_eSPI Library and Metamorphic Testing
- Definition of Metamorphic Relations
- Framework
- Evaluation
- Conclusion

Virtual Prototypes

A **Virtual Prototype** (VP) is an executable SW model of a HW system that runs on a host computer.

A VP is binary compatible to the physical HW.

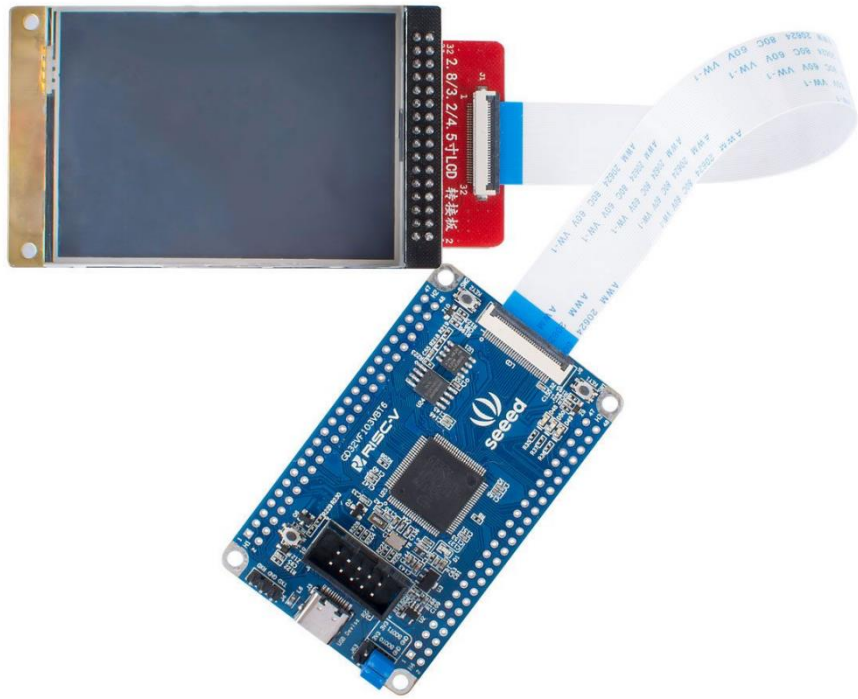
- **Industrial-proven, widely used by semiconductor global players**
- **VPs are modeled in SystemC**
  - C++ class library
  - IEEE1666-2011 Standard
  - Transaction Level Modeling (TLM) for abstraction

SYSTEMC™

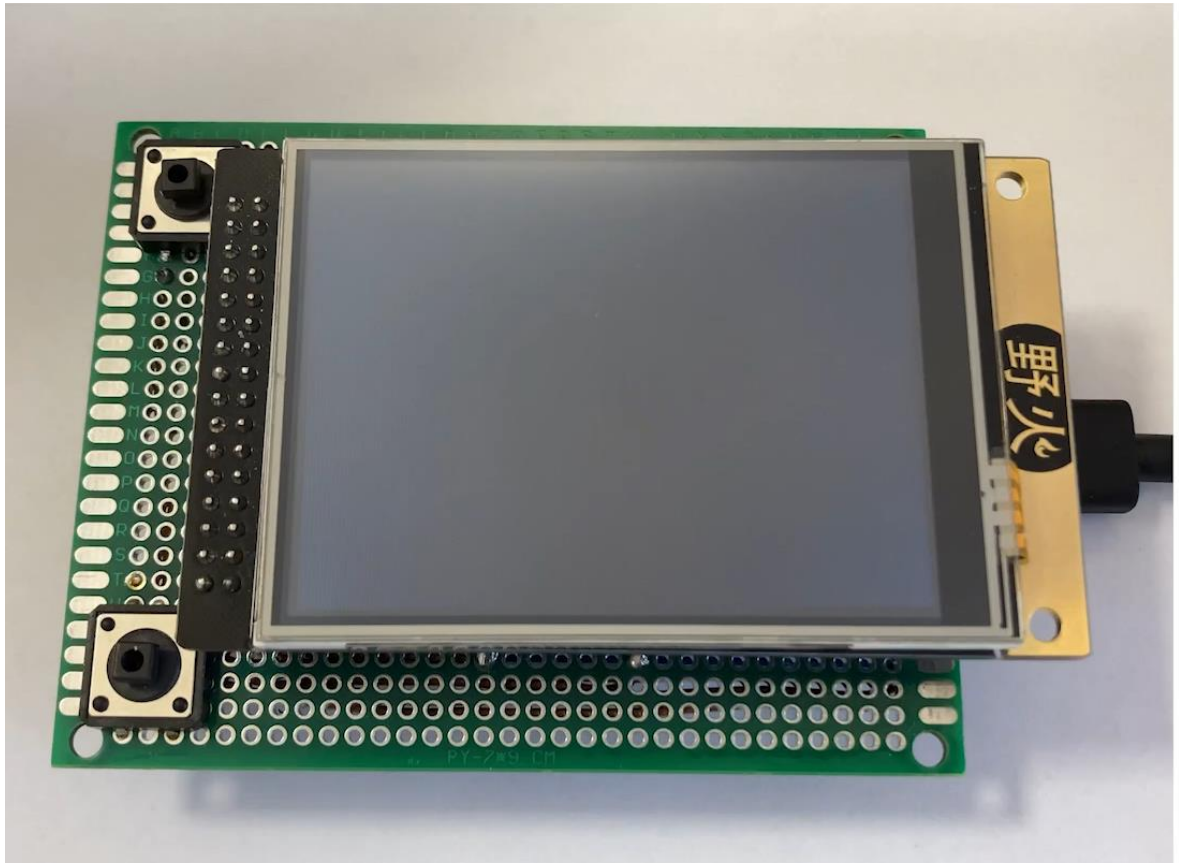
accellera SYSTEMS INITIATIVE™

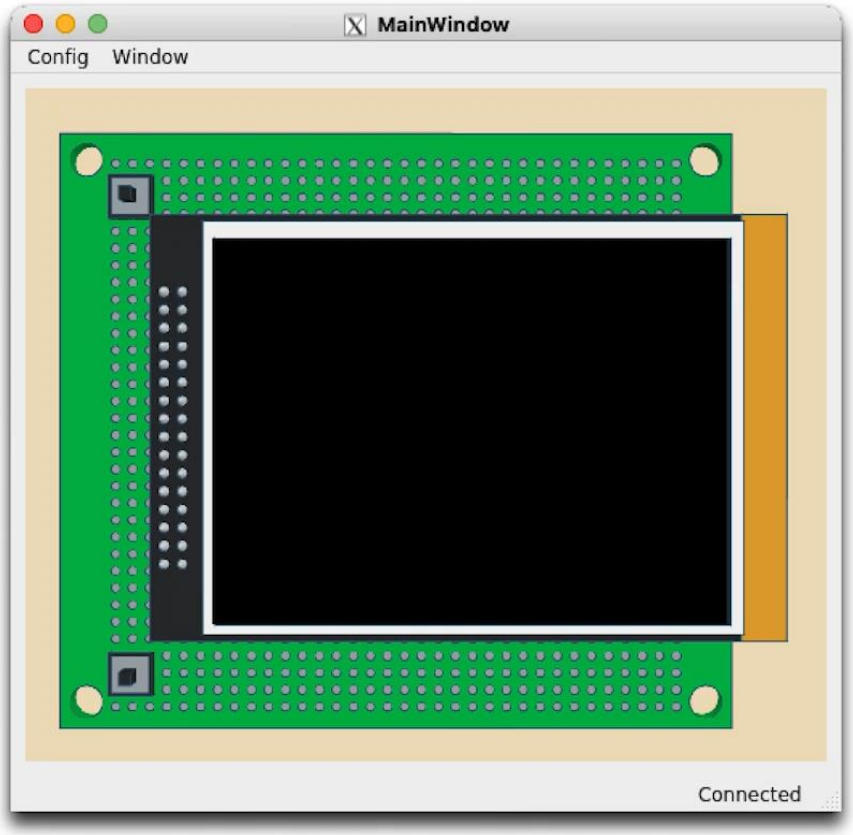
RISC-V VP++: GD32V Platform Configuration

- **Based on our open-source RISC-V VP++**
  - https://github.com/ics-jku/riscv-vp-plusplus
- **GD32VF103VBT6 microcontroller**
  - Nuclei N205 processor core
- **Implemented peripherals in RISC-V VP++**
  - GPIO, AFIO, EXTI, SPI, EXMC, RCU
- **UI**
  - ILI9341 display w XPT2046 touch controller



RISC-V VP++: GD32V Demo

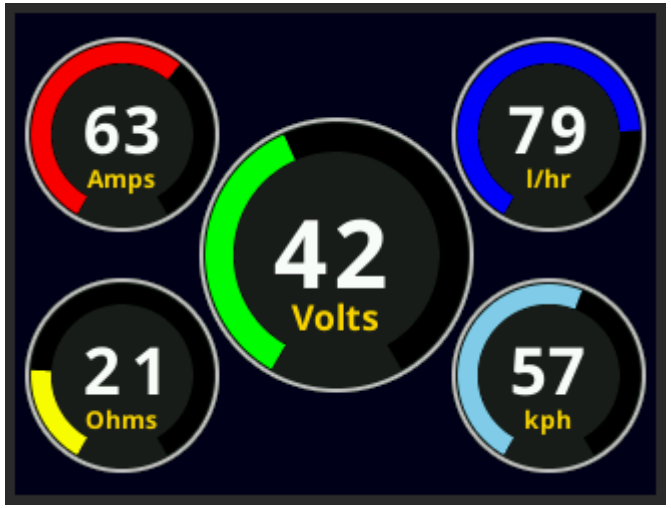




- Verification is highly automatable and no physical HW needed

TFT_eSPI Library

- **“A feature rich Arduino IDE compatible graphics and fonts library for 32-bit processors”[1]**
- **Software Interfaces**
  - Basic Geometric Objects
  - Sprites
  - Fonts
- **Hardware Interfaces**
  - Touch controller
  - 8-bit parallel or SPI interface for display control



[1]https://github.com/Bodmer/TFT_eSPI

TFT_eSPI Verification Challenge

- **Current verification:**
  - Use TFT_eSPI and visually check the result

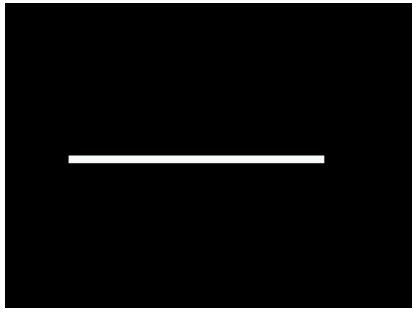
- **Our approach:**
  - Overcome need of test oracle via *Metamorphic Testing*
    - Relates multiple program executions via *Metamorphic Relations* (MRs)
    - If relation is violated, bug is found

Metamorphic Testing for TFT_eSPI

Relates multiple program executions

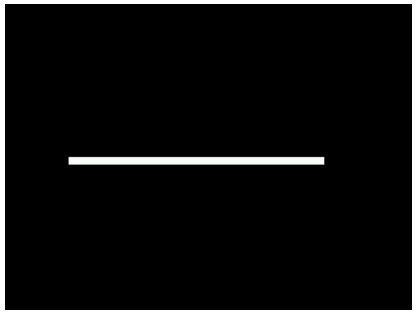
Source testcase

`drawLine(A, B)` → compileAndRun →



Follow-up testcase

`drawLine(B, A)` → compileAndRun →



=?

Metamorphic Testing FW function

$$F \equiv \hat{F} \implies \text{compileAndRun}(F) = \text{compileAndRun}(\hat{F})$$

$$F \not\equiv \hat{F} \implies \text{compileAndRun}(F) \neq \text{compileAndRun}(\hat{F})$$

- **Derive follow-up testcase from source testcase**
  - Semantic preserving (or breaking) input transformation
    - Modify method parameters
    - Add, remove or reorder method calls
- **No test oracle needed**

Definition of MR: Pixel

- **Transformation**
  - Reorder method calls
- **Function**

```
void drawPixel(int32_t x, int32_t y, uint32_t color);
```

- **MR**

$$drawPixel(x_0, y_0, c_0), drawPixel(x_1, y_1, c_1), \ldots, drawPixel(x_{n-1}, y_{n-1}, c_{n-1})$$
$$\equiv$$
$$drawPixel(x_{n-1}, y_{n-1}, c_{n-1}), \ldots, drawPixel(x_1, y_1, c_1), drawPixel(x_0, y_0, c_0)$$

Definition MR: Line

- **Transformation**
  - Modify Method Parameters
- **Function**

```
void drawWedgeLine(float ax, float ay, float bx, float by,
                   float aw, float bw,
                   uint32_t fg_color, uint32_t bg_color=0x00FFFFFF);
```

- **MR**

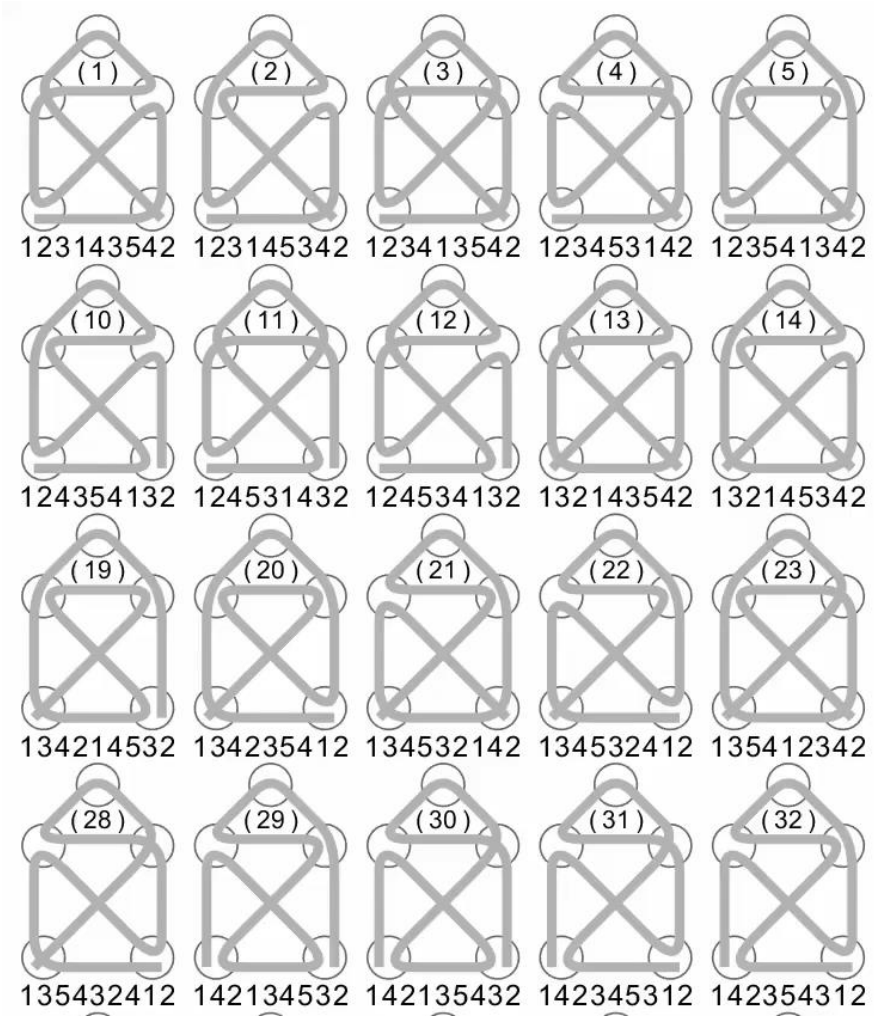
drawWedgeLine(ax,ay,bx,by,aw,bw,c_{fg},c_{bg})

≡

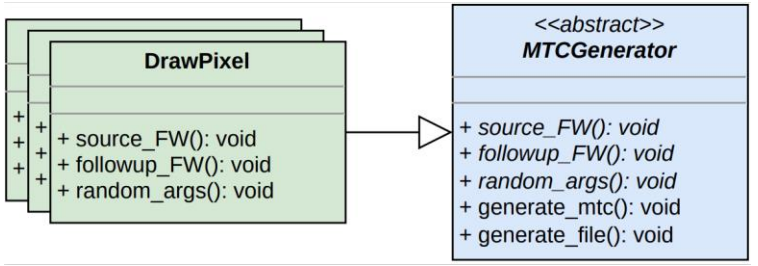
drawWedgeLine(bx,by,ax,ay,bw,aw,c_{fg},c_{bg})

Definition MR: Nikolaus

- **Background**
  - Haus vom Nikolaus (House of Santa Clause)
  - Multiple solutions
  - Eulerian path finding problem
- **Methodologies**
  1) Reorder Method calls according to 44 solutions
  2) Create sprites and push to display
  3) Draw house and
     move in different directions on display



MTC Generators

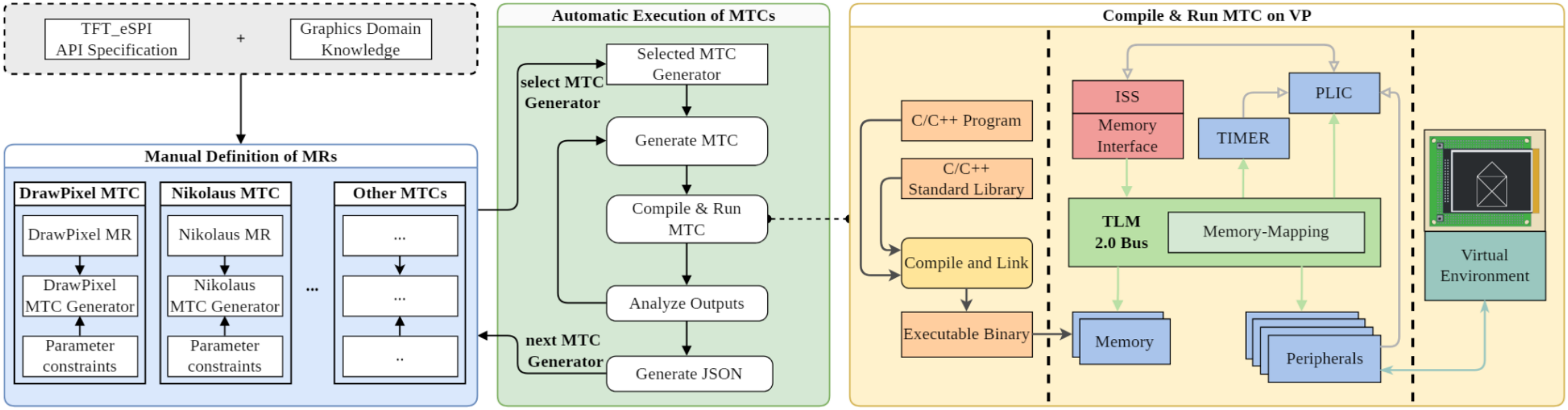


- *Metamophic Testcase Generator* (MTC) as abstract implementation
- Generate source FW and follow-up FW

```
// Source FW
#include "TFT_eSPI.h"
int main() {
   TFT_eSPI tft = TFT_eSPI();
   tft.init();
   tft.drawPixel(34,66,47586);
   tft.drawPixel(357,21,60570);
   tft.drawPixel(203,326,54515);
   tft.writecommand(0xFF);
   tft.writedata16(1);
   return 0;
}
```

```
// Follow-up FW
#include "TFT_eSPI.h"
int main() {
   TFT_eSPI tft = TFT_eSPI();
   tft.init();
   tft.drawPixel(203,326,54515);
   tft.drawPixel(357,21,60570);
   tft.drawPixel(34,66,47586);
   tft.writecommand(0xFF);
   tft.writedata16(2);
   return 0;
}
```

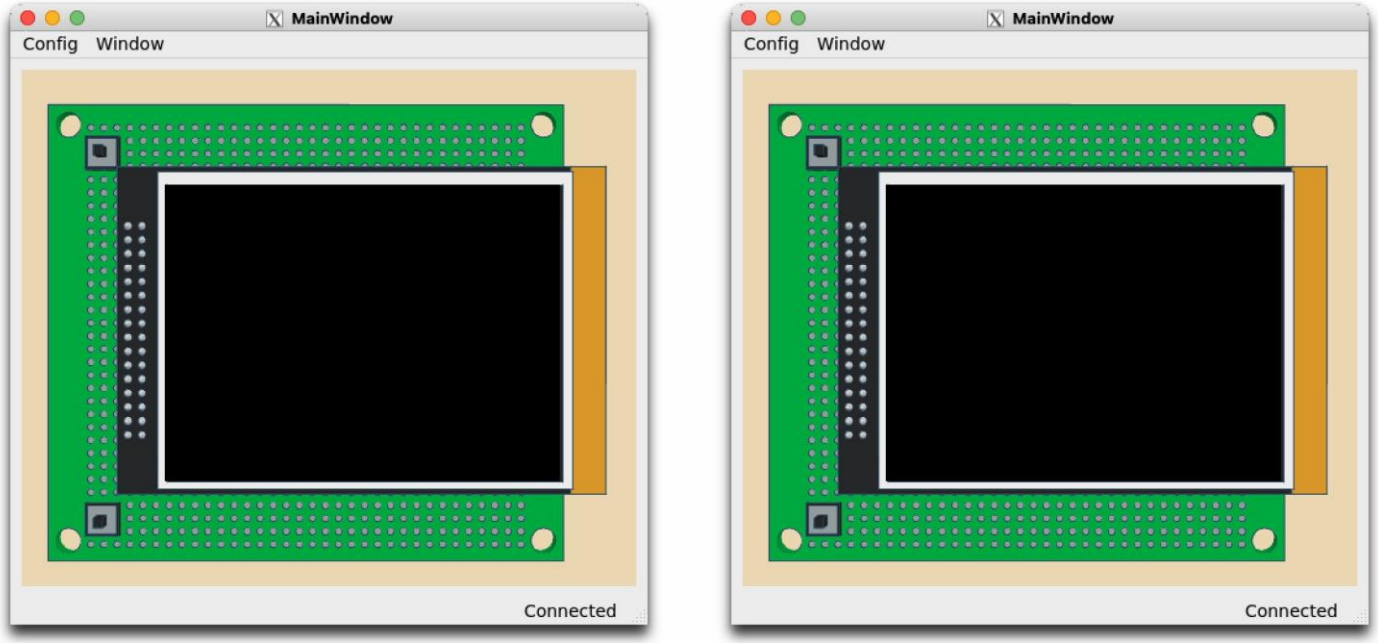
Framework Overview



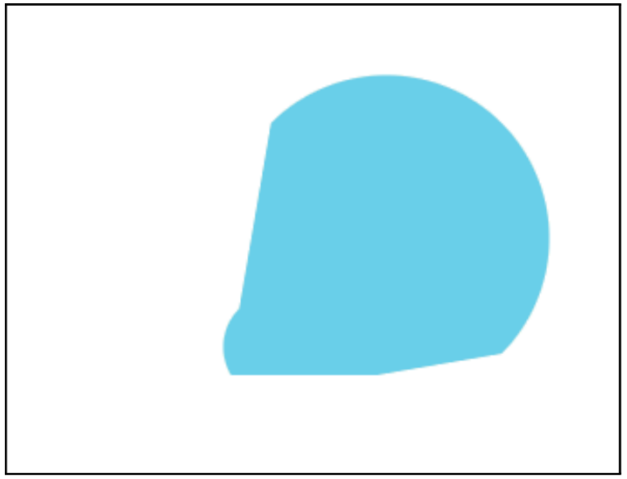
Open source: https://github.com/ics-jku/mt-graphlib-framework

Results

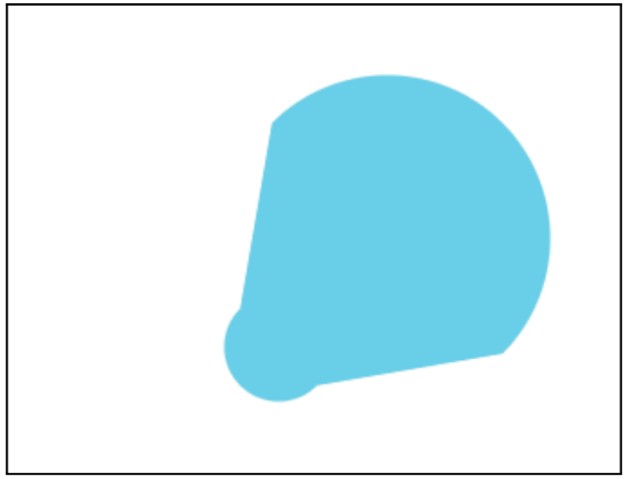
Number of MRs	21
Timeout per MR	4h
ø Testcases per MR	4300
Failed MRs	11
Total Bugs	15



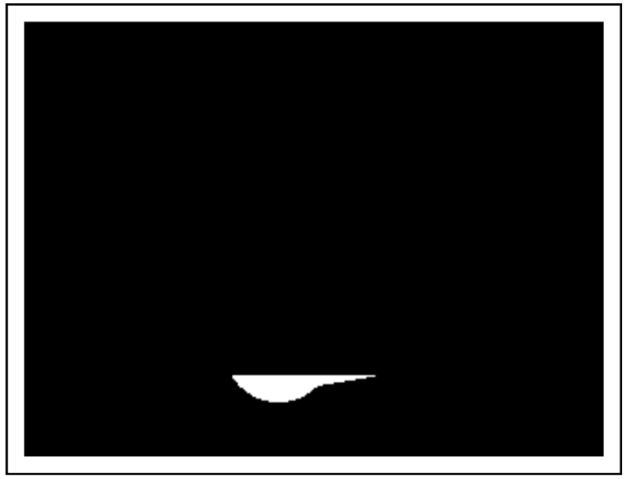
WedgeLine Bug



(a) Source Test Case

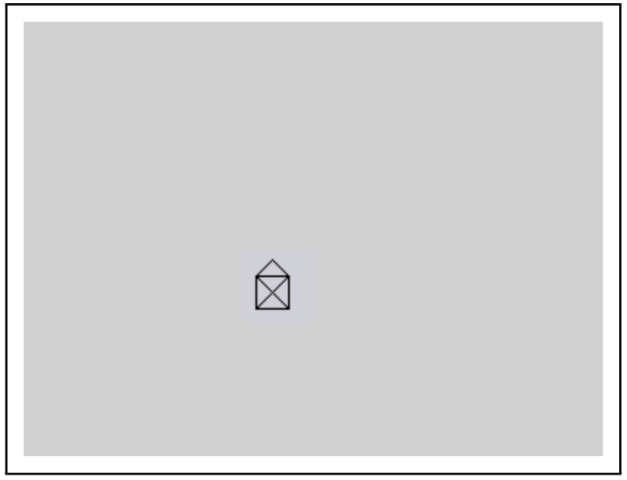


(b) Follow-up Test Case

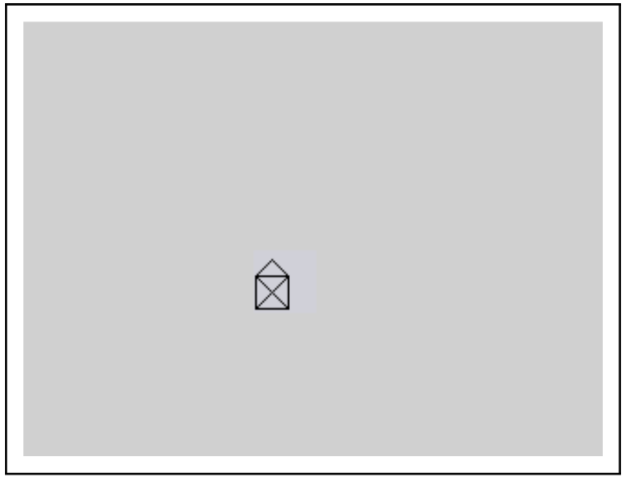


(c) Difference

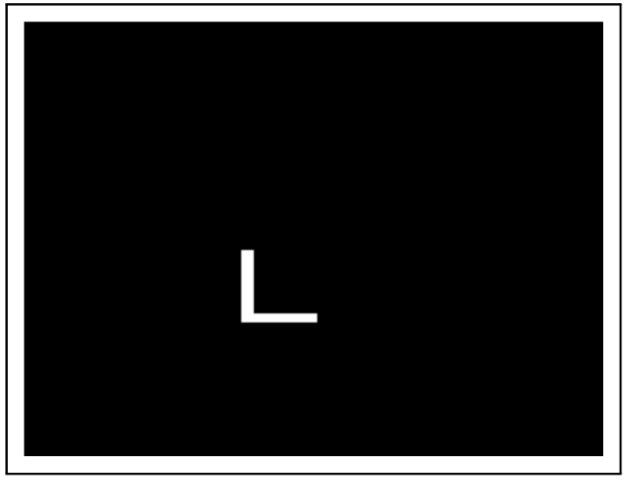
Nikolaus Move Bug



(a) Source Test Case



(b) Follow-up Test Case



(c) Difference

Conclusion

- **Effective firmware verification approach**
  - No need for physical HW
  - No need for a test oracle
  - Highly automated
- **Framework available on GitHub**
  - https://github.com/ics-jku/mt-graphlib-framework
- **Exposed 15 bugs in the TFT_eSPI library**

- **Outlook**
  - Combine with other techniques
  - Automate MR definition
  - Apply to other embedded applications

Verifying Embedded Graphics Libraries leveraging Virtual Prototypes and Metamorphic Testing

Christoph Hazott, Florian Stögmüller, Daniel Große
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: christoph.hazott@jku.at

JOHANNES KEPLER UNIVERSITY LINZ