UNIVERSITY OF MICHIGAN

Phantasos: A Framework for FPGA based Hardware-in-the-Loop Testing

Karthik Karyamapudi

Introduction

Testing embedded systems firmware is hard

(citation needed)

Introduction

- With regular software
  - Can create test harnesses with code
  - Create stubs which emulate the behavior of production environment
  - Automate inputs and outputs
  - Define desired behavior
  - Repeat tests as many times as necessary

Introduction

- Why can’t we do the same for embedded systems?
  - Embedded systems firmware is, well, embedded
  - Placed in a potentially remote, physical environment with complicated behavior

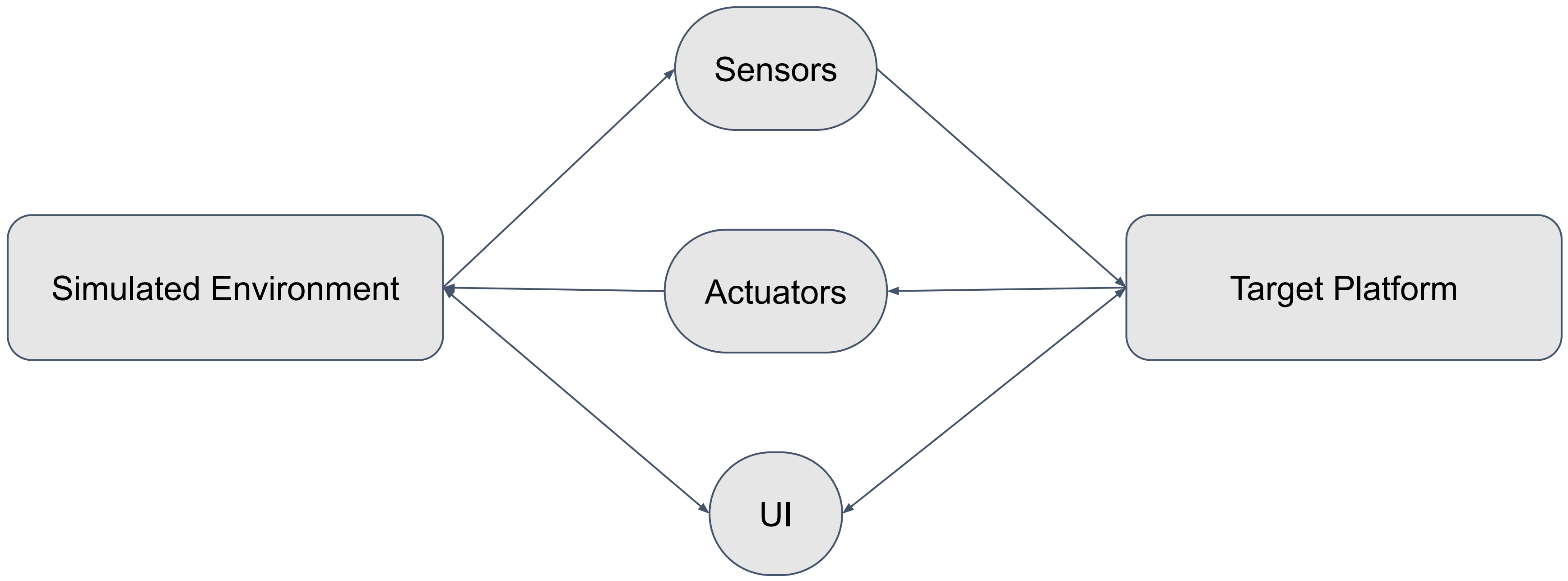
Options

- Test parts of the firmware independently offline
  - Does not model system-wide behavior
  - Need to structure firmware around this
- Use virtual machine to simulate the entire system at once
  - Needs accurate model of target core + interconnect + peripherals
  - Might not capture hardware behavior such as errata
- Hardware-in-the-Loop Testing
  - Runs firmware directly on the target hardware

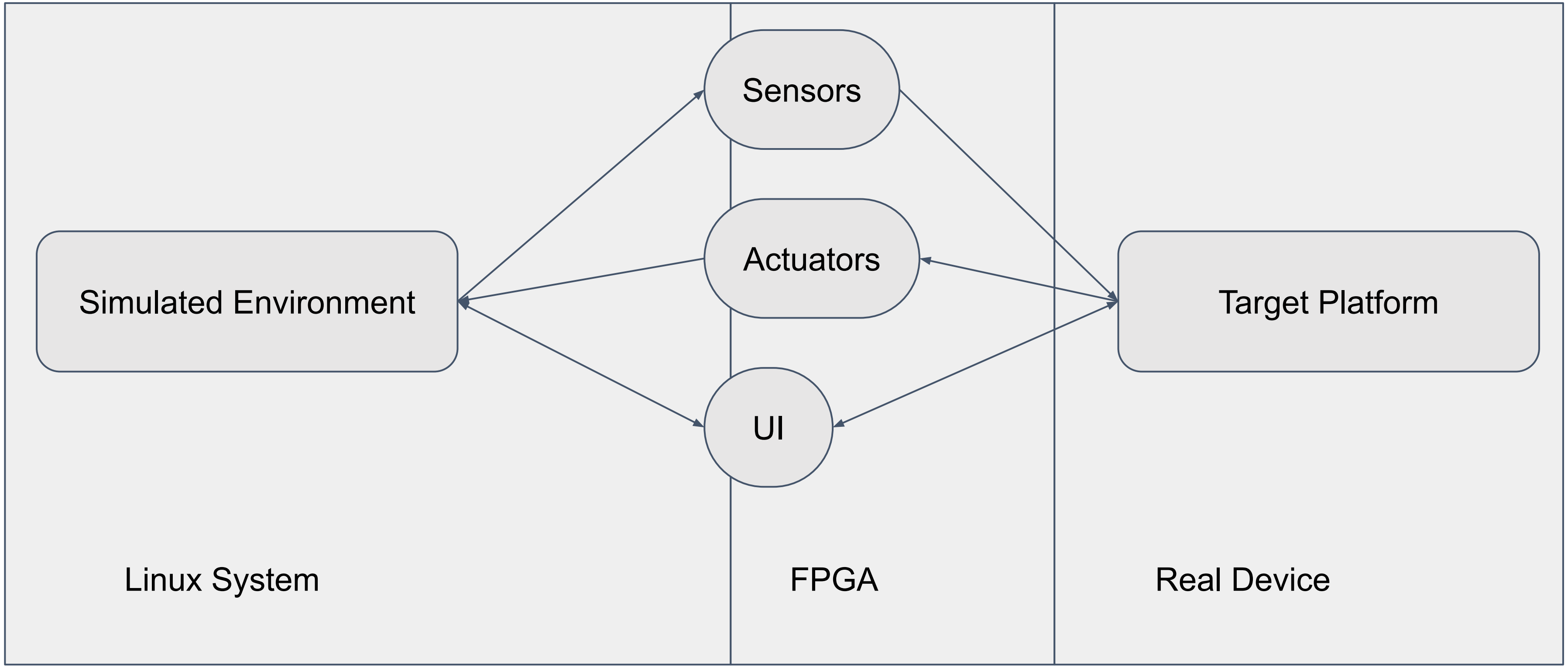
Hardware-in-the-Loop Testing

- Already used in automotive, power management, aerospace, etc.
- Typically expensive, specialized test harnesses
- Connected to custom hardware
- Not practical for simpler applications
- Generally focuses on control systems with Simulink

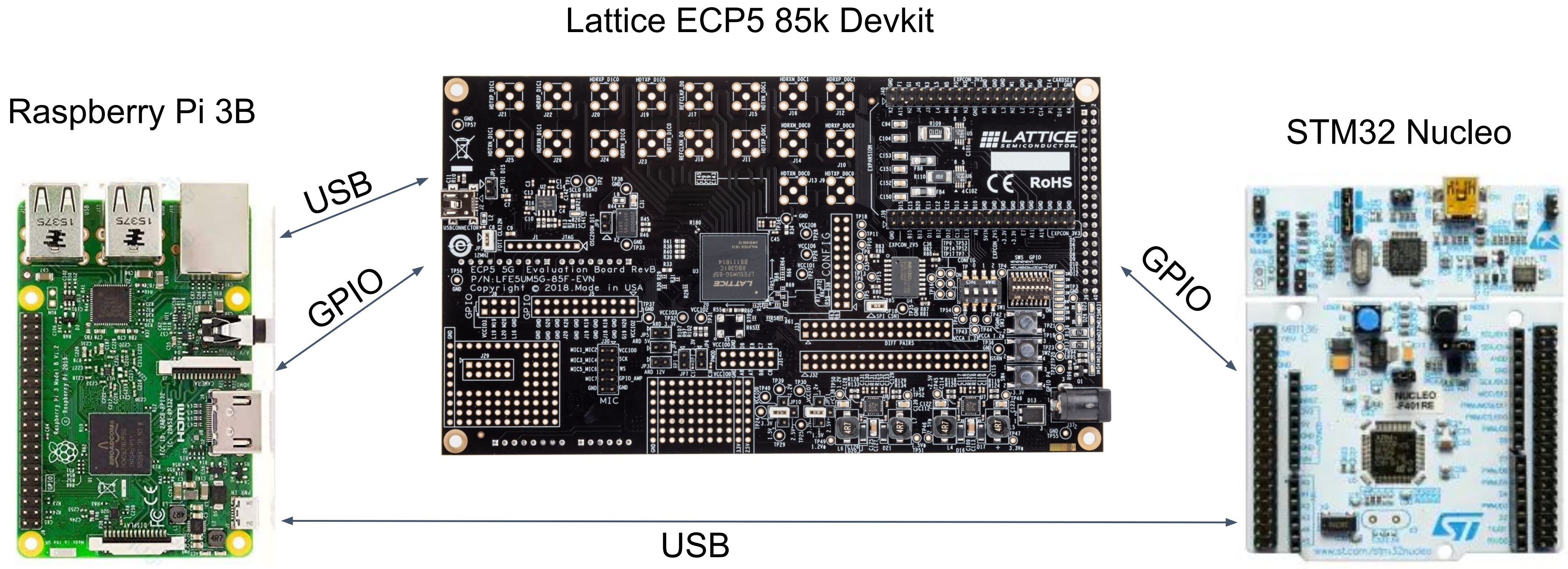
Hardware-in-the-Loop Testing



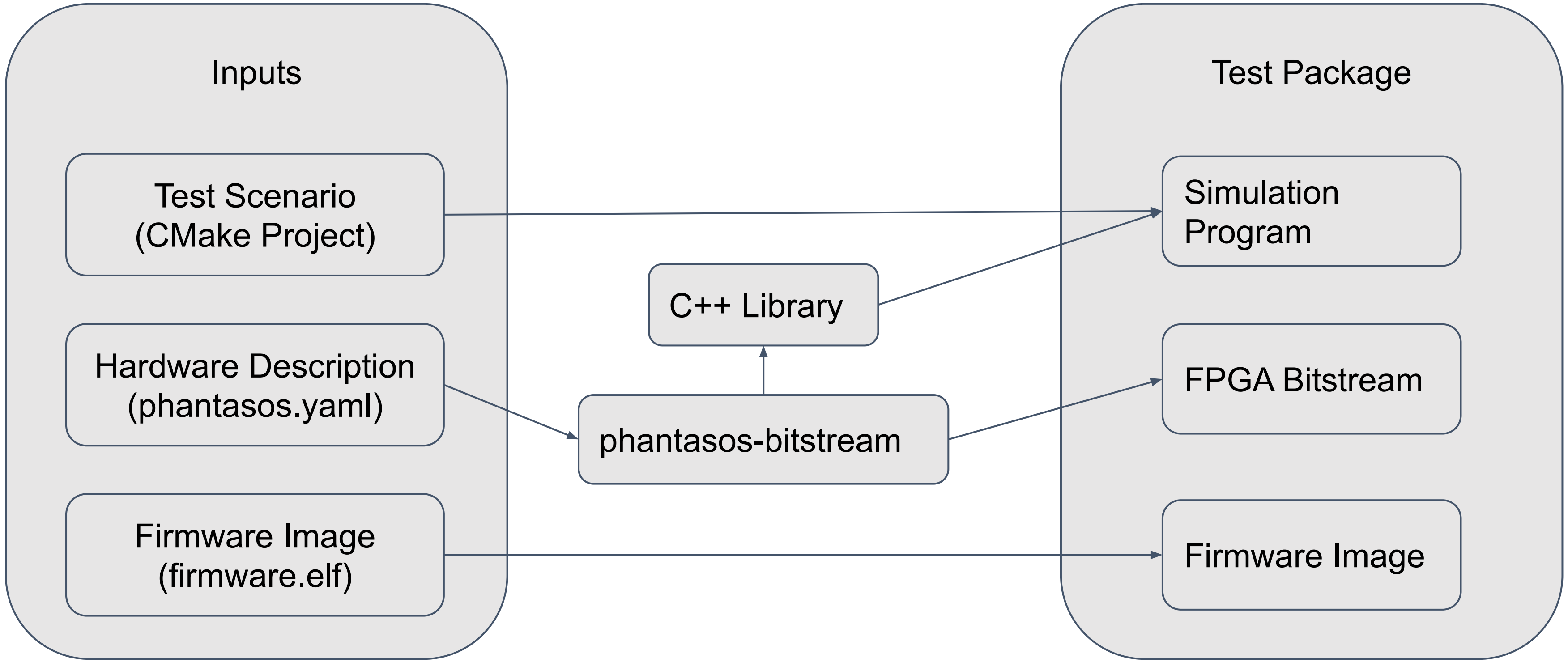
HiL Testing w/ Phantasos



Hardware Setup



Phantasos Architecture


Inputs
Test Scenario (CMake Project)
Hardware Description (phantasos.yaml)
Firmware Image (firmware.elf)
C++ Library
phantasos-bitstream
Test Package
Simulation Program
FPGA Bitstream
Firmware Image

Software Tools

- openOCD - Debugging tool used for embedded systems

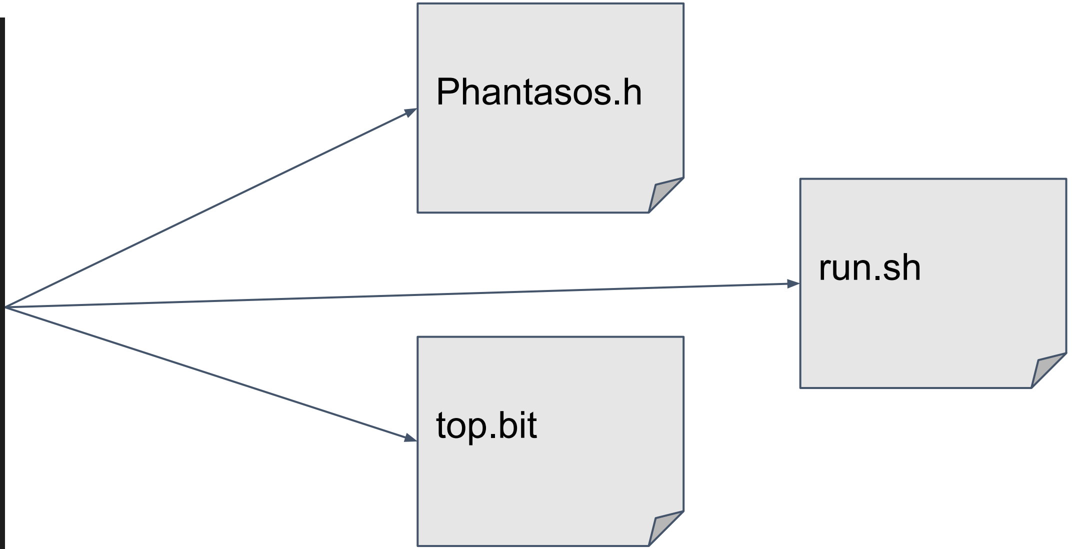
  Able to flash target system with image being tested

- openFPGAloader - Loads bitstream into FPGA
- Amaranth HDL - enables hardware definition from Python
- Yosys + NextPnR - Synthesizes bitstream from hardware definition
- All open source!

phantasos-build

- Python tool developed for generating bitstream, software interface, and initialization script from hardware definitions

```yaml
%YAML 1.1
---
platform: ECP55GEVN
target: NucleoF4
peripherals:
- name: mastRange
  type: PulseWidthUltrasonic
  conn: ARDUINO
  pins:
    echo: D7
    trig: D6
```



phantasos-build

- Peripheral implementations are queried by the YAML file
- Have both C++ software library as well as FPGA logic
- Currently implemented:
    - HC-SR04 Ultrasonic Sensor
    - HD44780 Liquid Crystal Display Controller
    - General Purpose Input/Output
    - PWM Capture

Phantasos Data Flow

Define Peripheral

```yaml
- name: mastRange
  type: PulseWidthUltrasonic
  conn: ARDUINO
  pins:
    echo: D7
    trig: D6
```

Simulation uses Peripheral API

```cpp
// set ultrasonic distance
mastRange.setDistance(testDistance);
```

Peripheral Library communicates with FPGA fabric through Phantasos runtime

```cpp
void PulseWidthUltrasonic::setDistance(float cm){
    pulseWidthUs = cm*2*29.1545;
}
```

Peripheral logic receives value

```python
# control register
pulseWidthValue = Signal(16, reset=38000)
with m.If(self.pulseWidthUsCSR.w_stb):
    m.d.sync += pulseWidthValue.eq(self.pulseWidthUsCSR.w_data)
m.d.comb += self.pulseWidthUsCSR.r_data.eq(pulseWidthValue)
```

Firmware interacts with emulated device

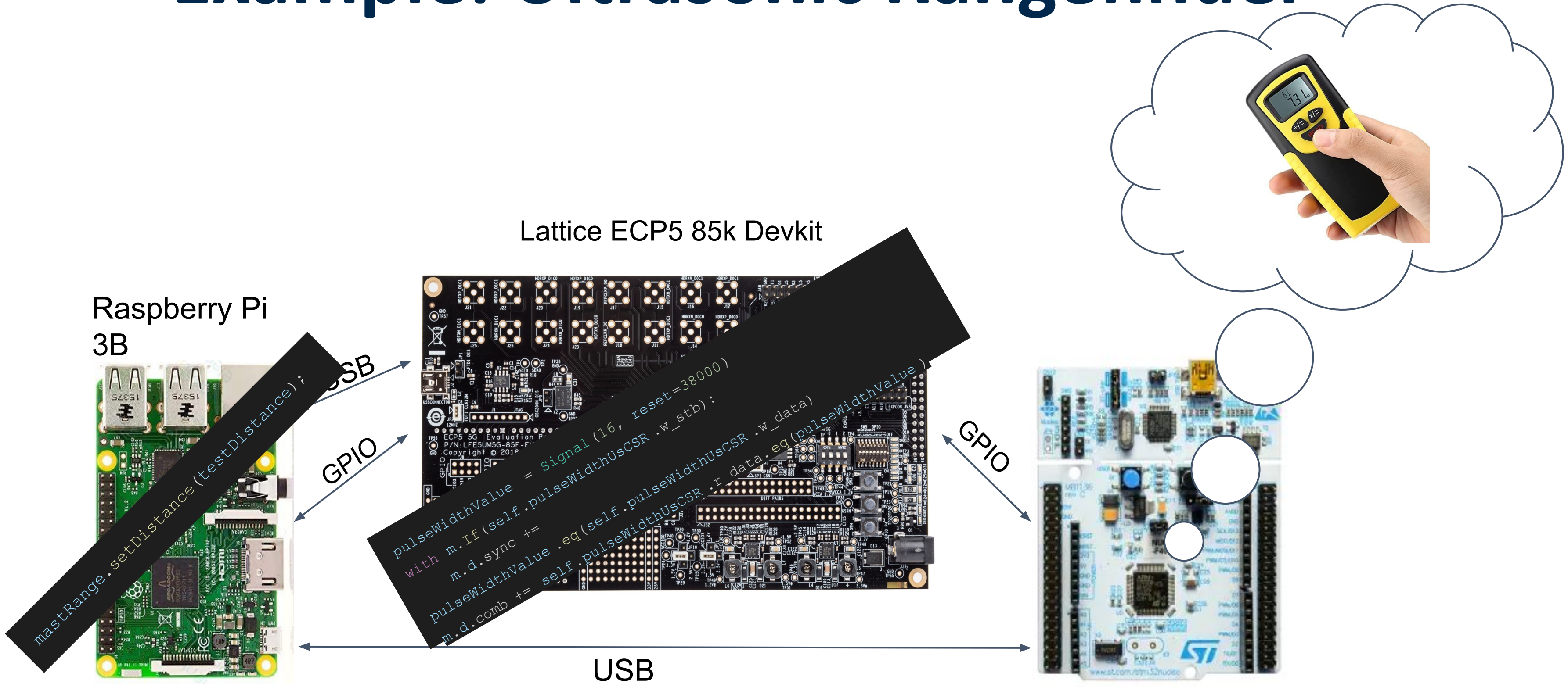
```cpp
ultrasonic.attach(6, 7);
ultrasonic.getDistanceCM();
```

Phantasos Data Flow

- Communication with FPGA fabric is done through RPi's SPI interface
- Each peripheral has registers and events connected through internal parallel bus
- Address space and interrupt numbers within FPGA are all tracked automatically and propagated to the software library

Example: Ultrasonic Rangefinder

Example: Ultrasonic Rangefinder

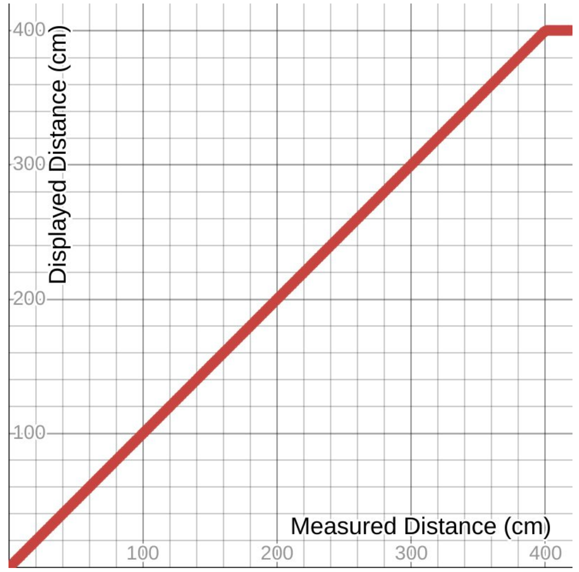


```
mastRange.setDistance(testDistance);
```

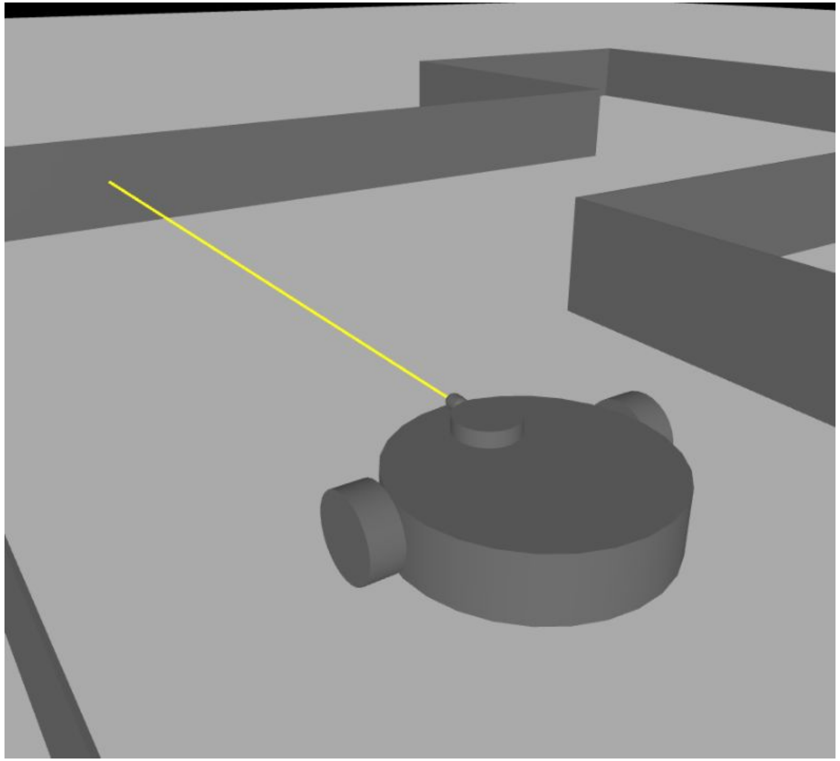
```
pulseWidthValue = Signal(16, reset=38000)
with m.If(self.pulseWidthUsCSR.w_stb):
    m.d.sync +=
pulseWidthValue.eq(self.pulseWidthUsCSR.w_data)
m.d.comb += self.pulseWidthUsCSR.r_data.eq(pulseWidthValue)
```

Example: Ultrasonic Rangefinder

- Uses the ultrasonic sensor and LCD display modules
- Sweeps the value of the sensor and reads the value on the display

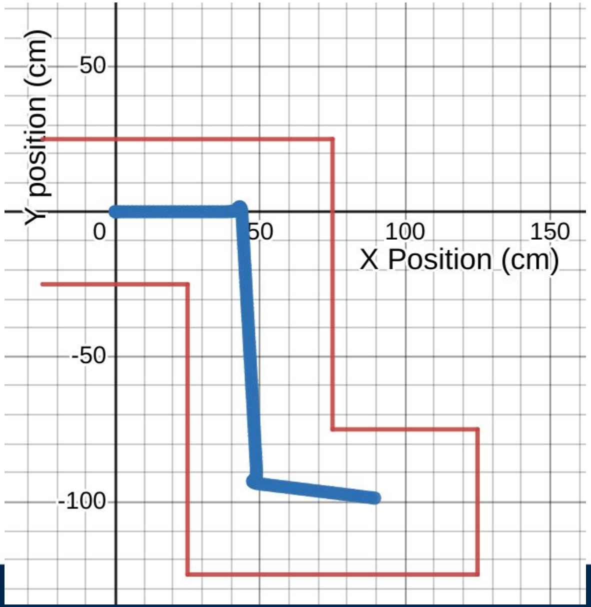


Example: Maze Navigation



Example: Maze Navigation

- Environment is simulated using MuJoCo library from DeepMind
- Uses same modules for range sensor and display
- Uses PWM and GPIO to model H-bridge motor drivers



Benefits

- Flexibility of hardware configuration
- One device per platform, no hardware development cost for custom test harness
- Could be rented out instead of purchased
- Phantasos modules can be reused without HDL or protocol knowledge

Applications

- Use as part of CI/CD flow for firmware projects
    - Initial intention for this project
- Prototyping platform for earlier in development process
- Educational/Competitive purposes, checking a firmware project automatically

Limitations

- Implementation Limitations
  - Small bandwidth across SPI
  - No analog signal support
- Inherent Limitations
  - No recreation of mechanical, thermal, electrical stresses
  - Non-deterministic behavior

Future Work

- Add support for a faster I/O, Xilinx Zynq or PCI card maybe
- Add analog I/O support to hardware
- Integration into testing frameworks, GitHub Actions
    - Network multiple boards for multi-agent tests
- Expand module library

Questions