Scala Symposium 2015

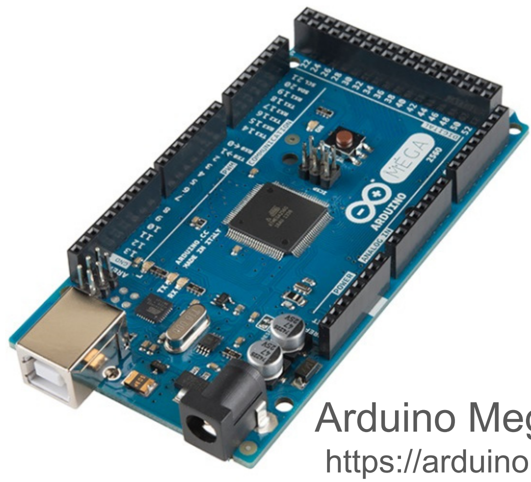
ESPeciaL: an Embedded Systems Programming Language

Portland, OR
June 2015

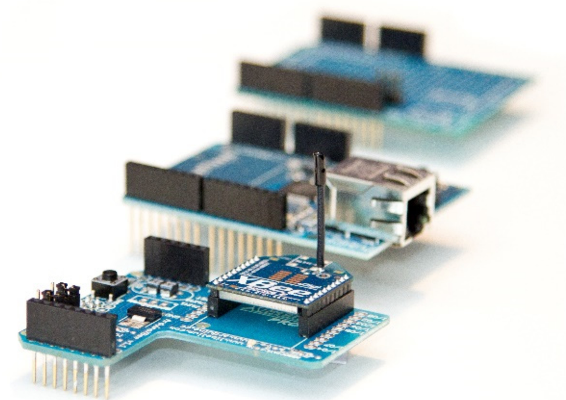
Christopher Métrailler, Pierre-André Mudry

Embedded systems for teaching

- Arduino
  - Low cost
  - Many peripherals available
  - Usage of libraries
  - Bare metal



Arduino Mega
https://arduino.cc



Programming principles: a simpler C

- Two functions
  - Setup function
  - Infinite loop
- Simple library calls
- «No» interrupts
- No pointers
- Simple types

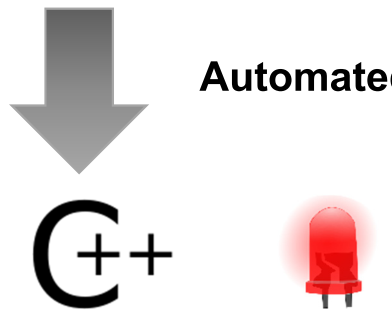
Blink

```
/*
  Blink
  Turns on an LED for one second, then off for one second.

  This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Can this be improved?

- C is fine, most of the time
- Proximity of C to hardware not always required / does not bring much.
- ESPeciaL:

```
val cst = Constant(bool(true))
val led = DigitalOutput(Pin('C', 12))

cst.out --> led.in // Set LED C#12 on
```



Automated code generation

Objective of this contribution

Provide a simpler and more flexible way to program (simple) embedded systems

1. State of the art
2. Presentation of the framework
3. Experimental results

State of the art

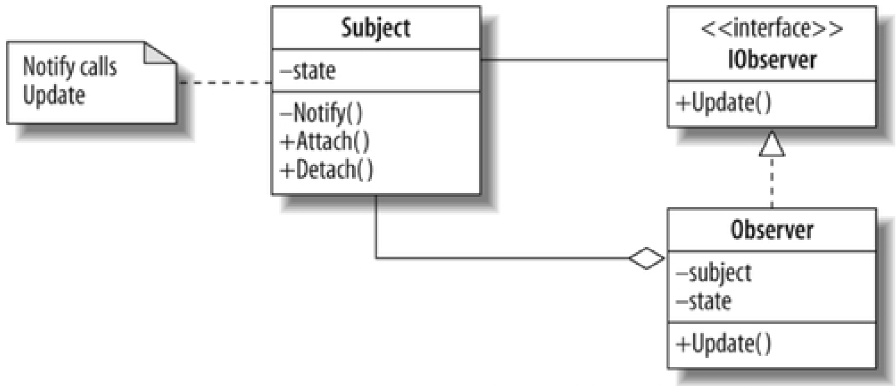
1. VISUAL PROGRAMMING LANGUAGES

High-level program representations

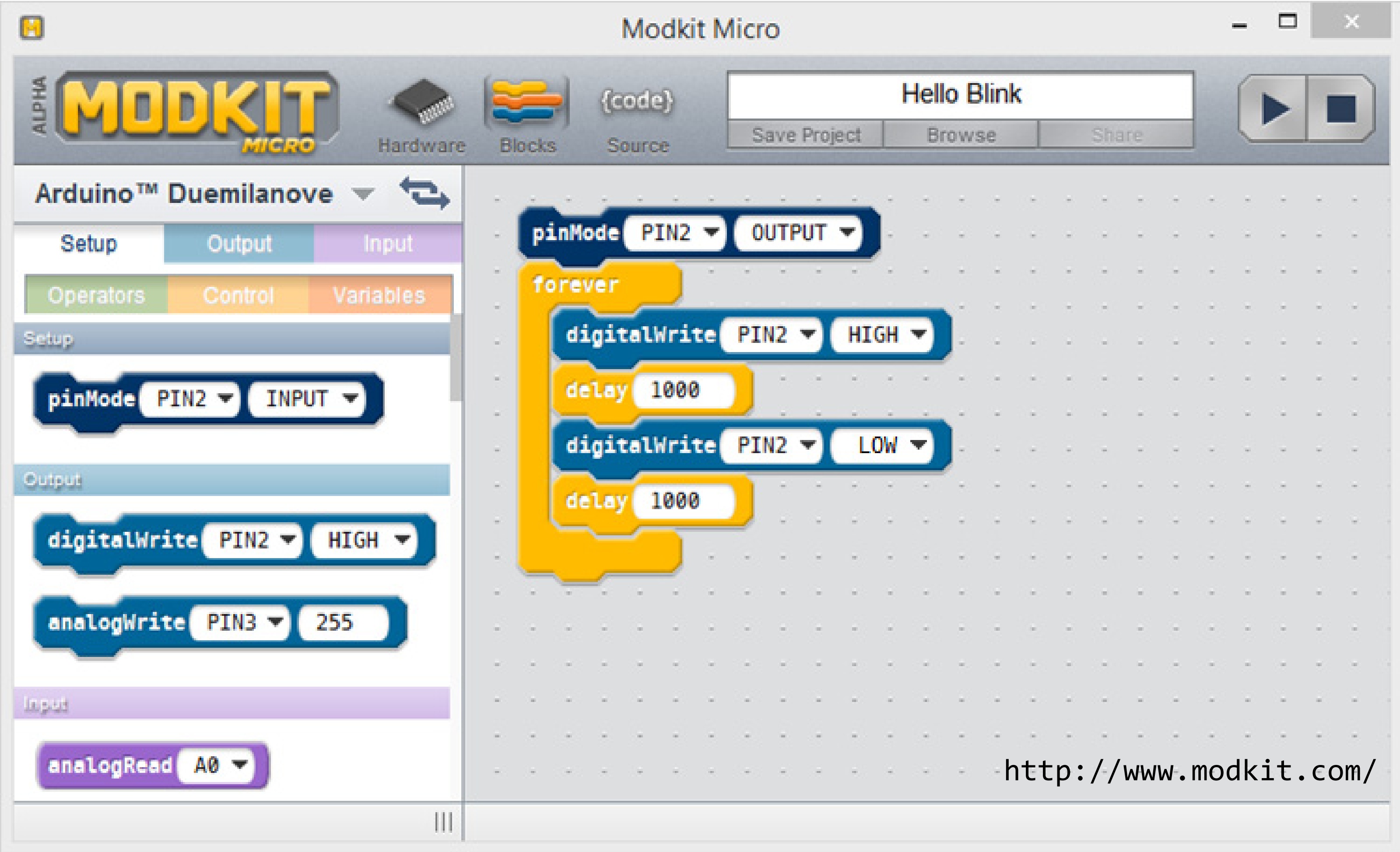
- Visual programming languages
  - Application model described visually
  - Code is emitted from this description
  - Abundant literature on the subject
- How to represent a sequential program graphically?

Main approaches used

1. UML model
2. **Block-based**
3. **Flow-based**
4. **Dataflow**

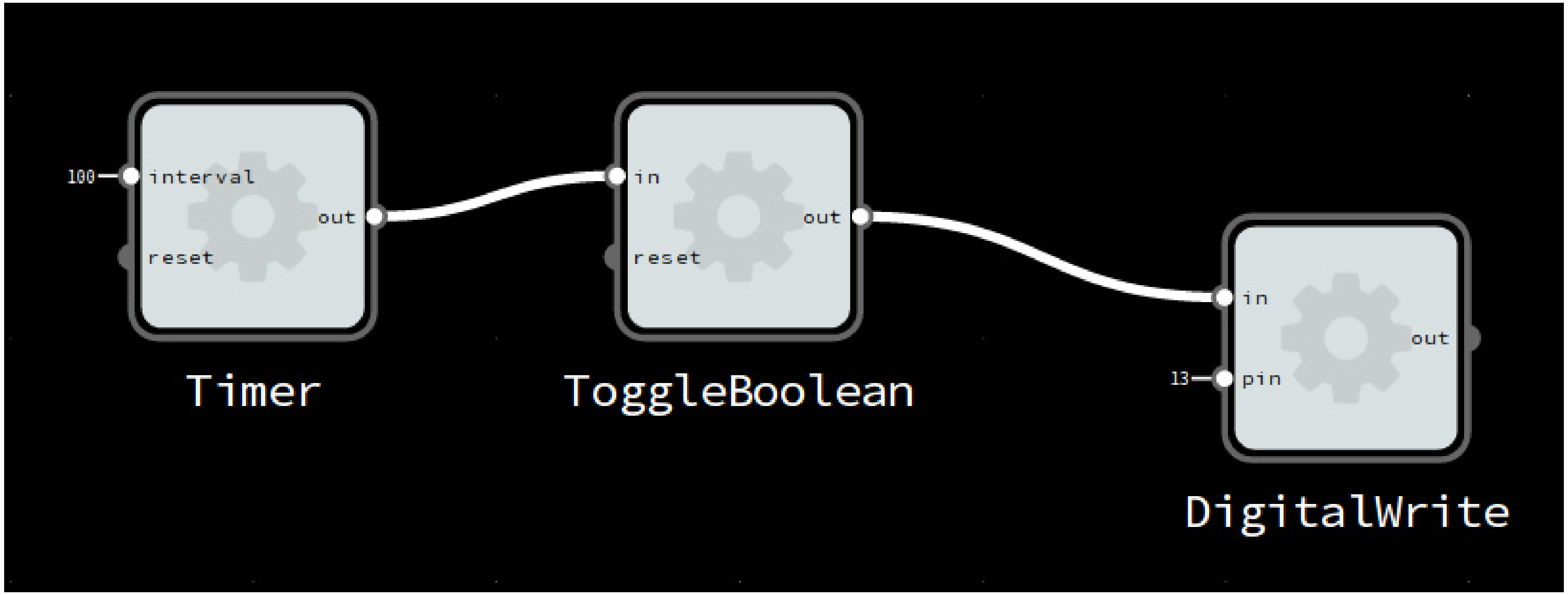


Block-based programming



- Code generation for Arduino
- Other projects: Blitbloq, Ardublock and Scratch

Flow-based programming

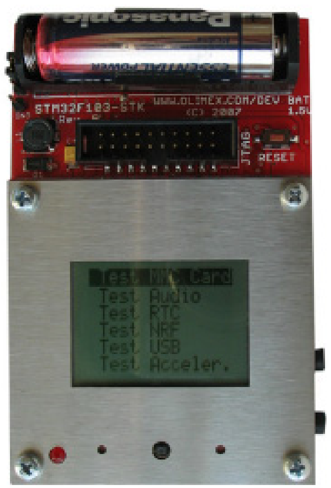


- NoFlo, MicroFlo for embedded systems
- Node-RED for IoT
- LabView (dataflow)

2.1. FRAMEWORK OVERVIEW

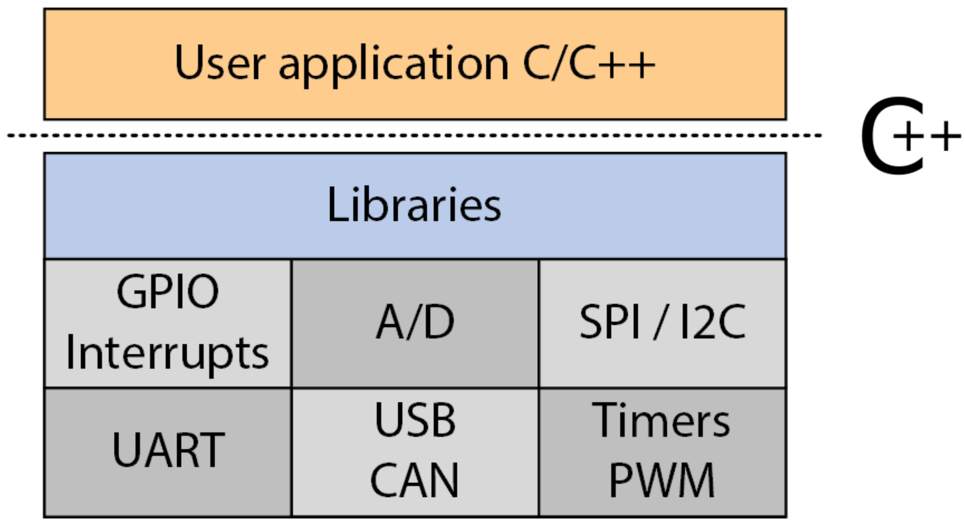
Embedded systems programming

- When not using Arduino-like systems
  - Low level C/C++ code



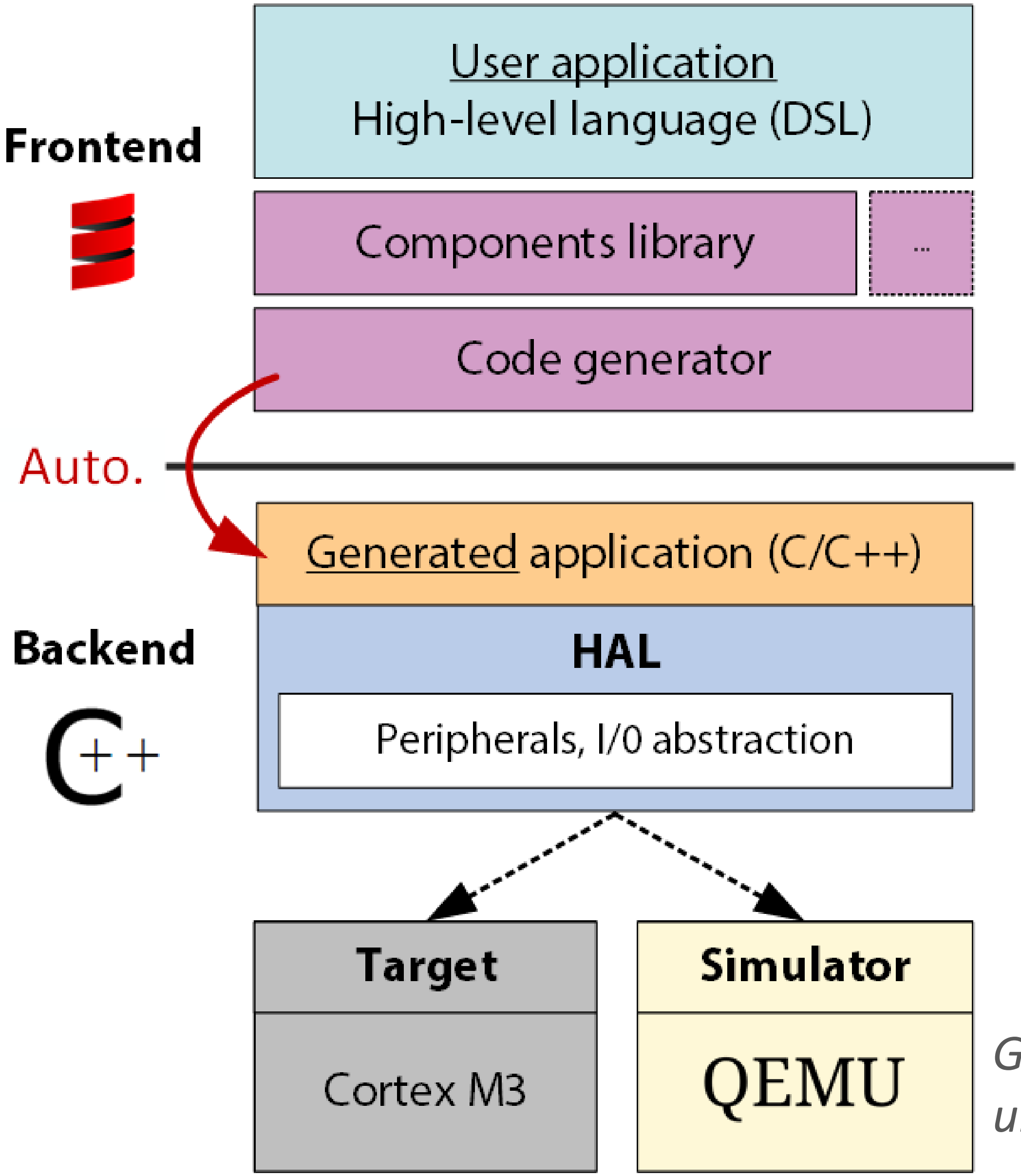
ARM 32 bit
CORTEX M3™

STM32F103
72 MHz



Standard approach

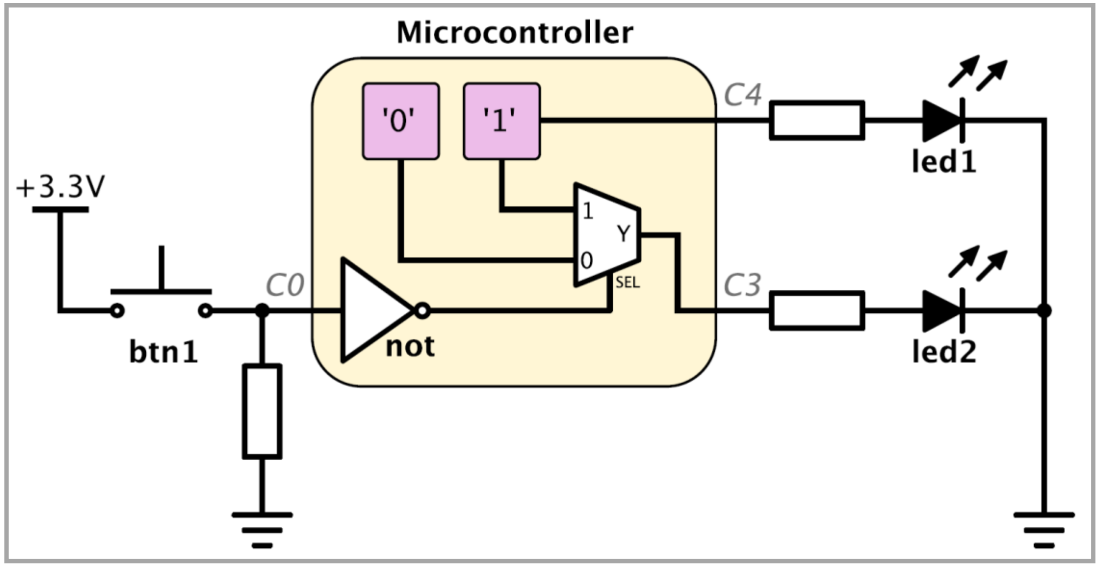
ESPeciaL framework architecture

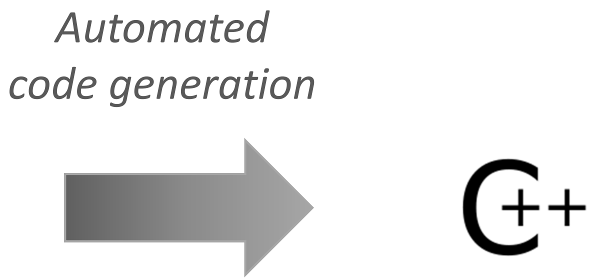


Great for unit testing

Workflow 1/4

- Sample application
  - Led1 is ON
  - Led2 is OFF when the button is pressed
  - Use constant values, digital I/O, a not gate and a 2-input multiplexer



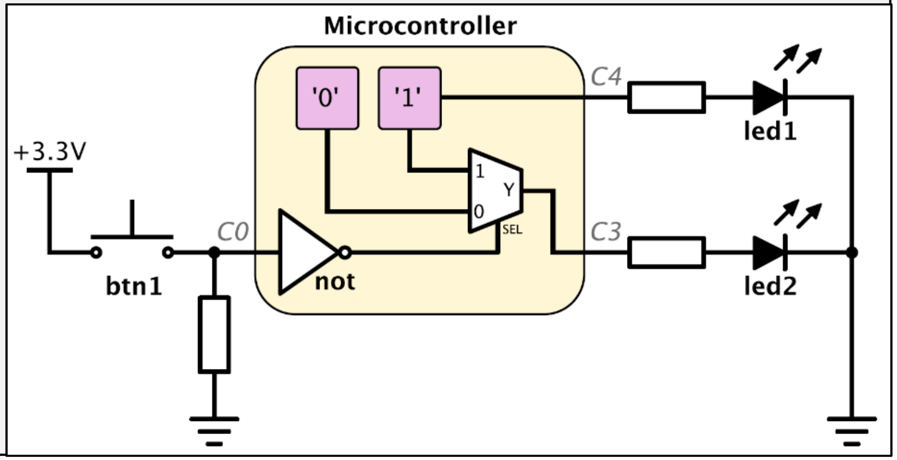


Workflow 2/4

Dataflow specification

SCALA + DSL

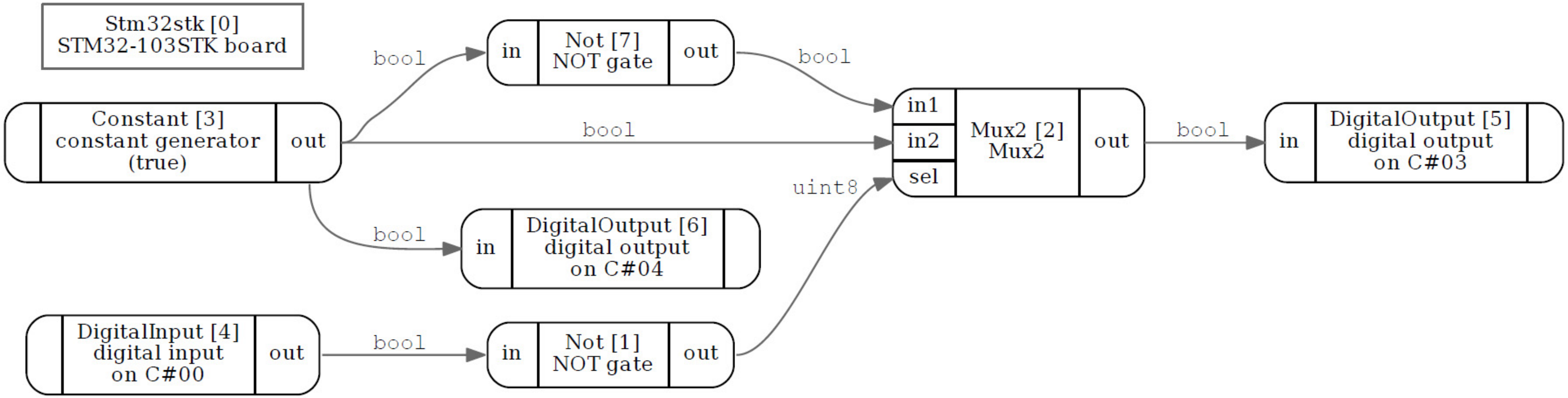
```
val not = Not()
val mux = Mux2[bool]()
val cst1 = Constant(bool(true)).out

IO.btn1.out --> not.in

not.out --> mux.sel
!cst1 --> mux.in1
cst1 --> mux.in2

mux.out --> IO.led2.in
cst1 --> IO.led1.in
```



- Use components available in the framework
- Wires --> have a direction, a type (short circuit)

Workflow 3/4

Directed acyclic graph (DAG)



- Resolving the graph to generate a sequential C++ code
- Input-Process-Output (IPO) model

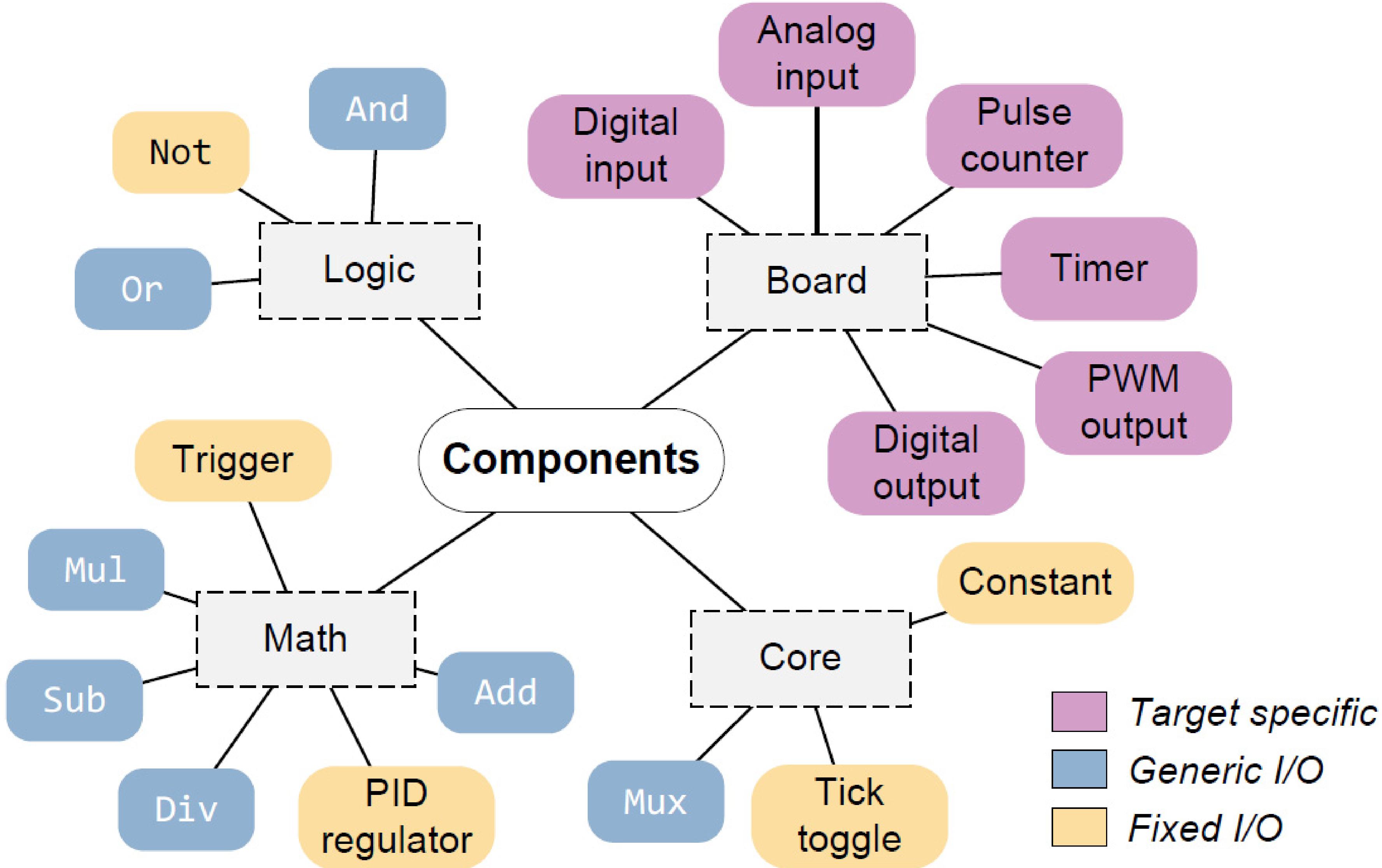
Workflow 4/4

C++ code generation from the DAG based on the HAL

```
// ...
while(1) {
  // 1) Read inputs (I)
  bool in_C0 = in_cmp04.get();          →  DigitalInput in_cmp04('C', 0);

  // 2) Process (P)
  uint8_t out_cmp01 = if(in_C0) ? 0:1;  →  Not gate

  uint8_t sel_cmp02 = out_cmp01;
  bool out_cmp02;
  if(sel_cmp02 == 0)                     →  Mux with 2 inputs
    out_cmp02 = false;
  else
    out_cmp02 = true;

  // 3) Update outputs (O)
  out_cmp05.set(out_cmp02);              →  DigitalOutput out_cmp05('C', 3);
  out_cmp06.set(true);                   →  DigitalOutput out_cmp06('C', 4);
}
```

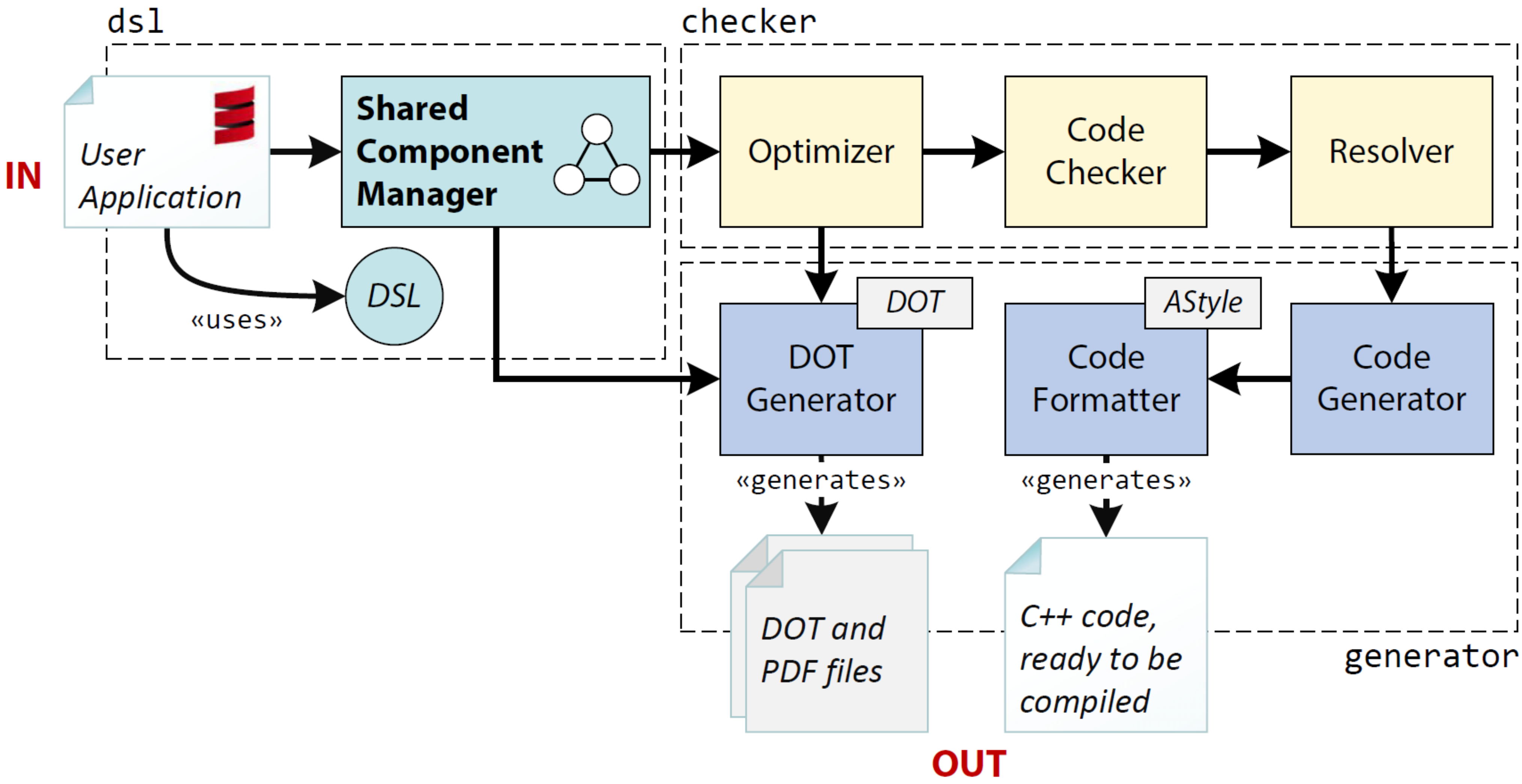
Component library

Generic or target specific components



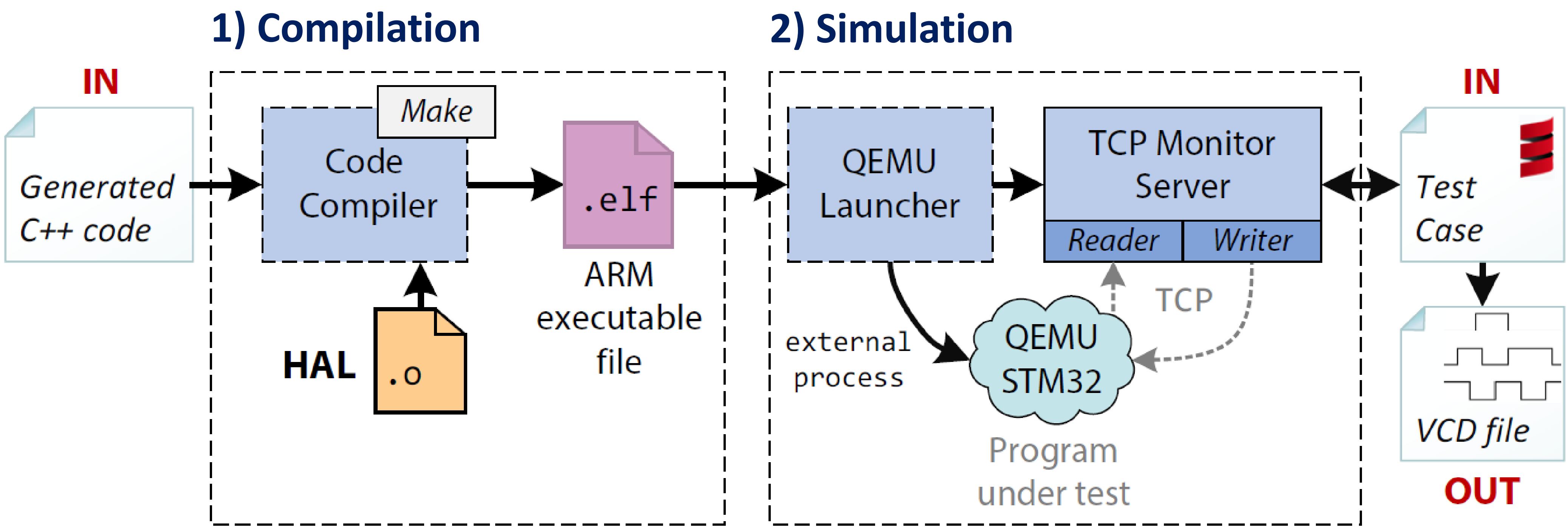
2.2. FRAMEWORK IMPLEMENTATION

Code generation pipeline



Code compilation

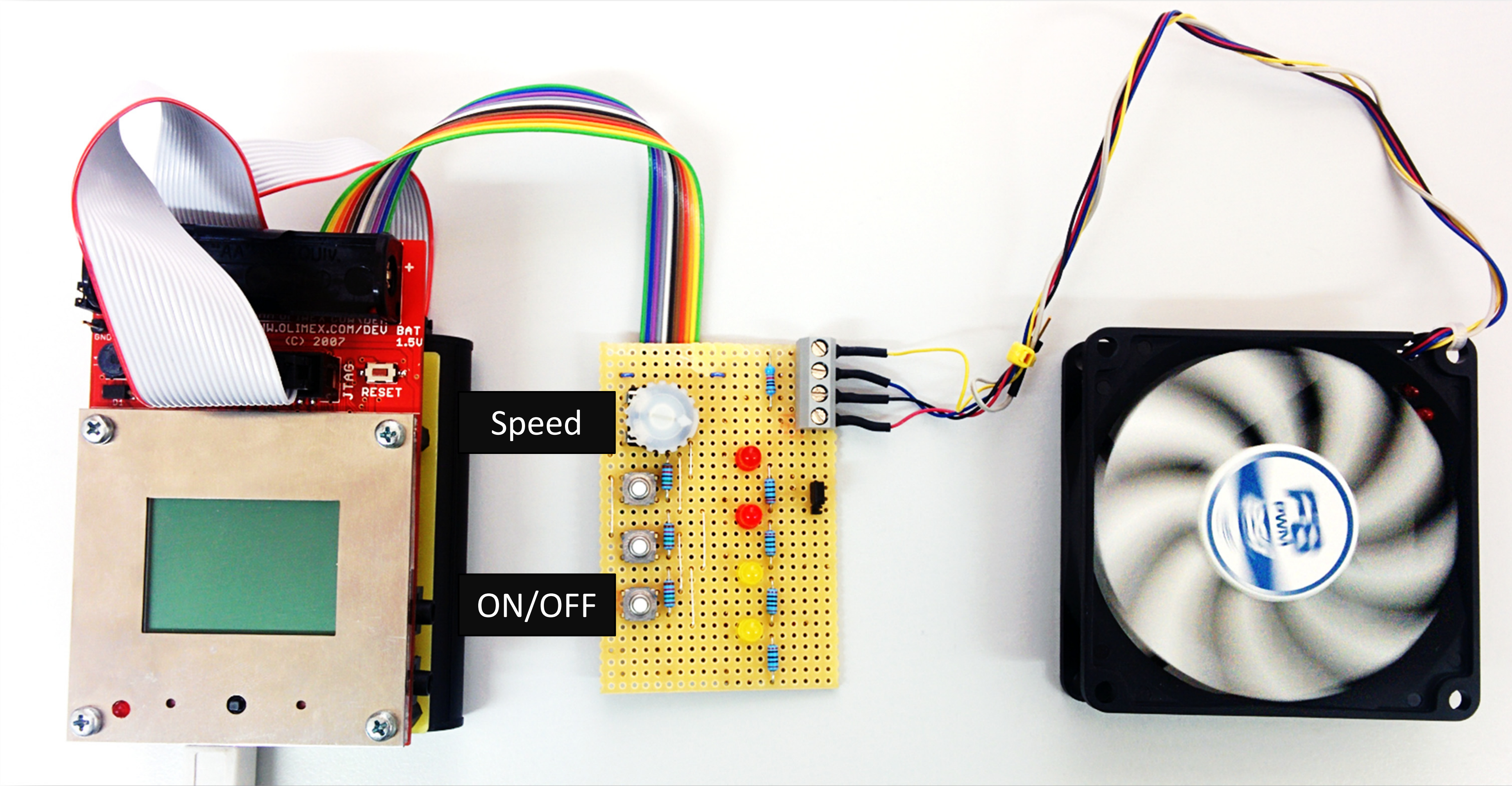
1. Compile the generated code for target using our HAL
   - GNU ARM toolchain (`make`, `GCC`, …)
2. Compile and simulate the code in QEMU
   - Fully automated unit tests in Scala (Scalatest)



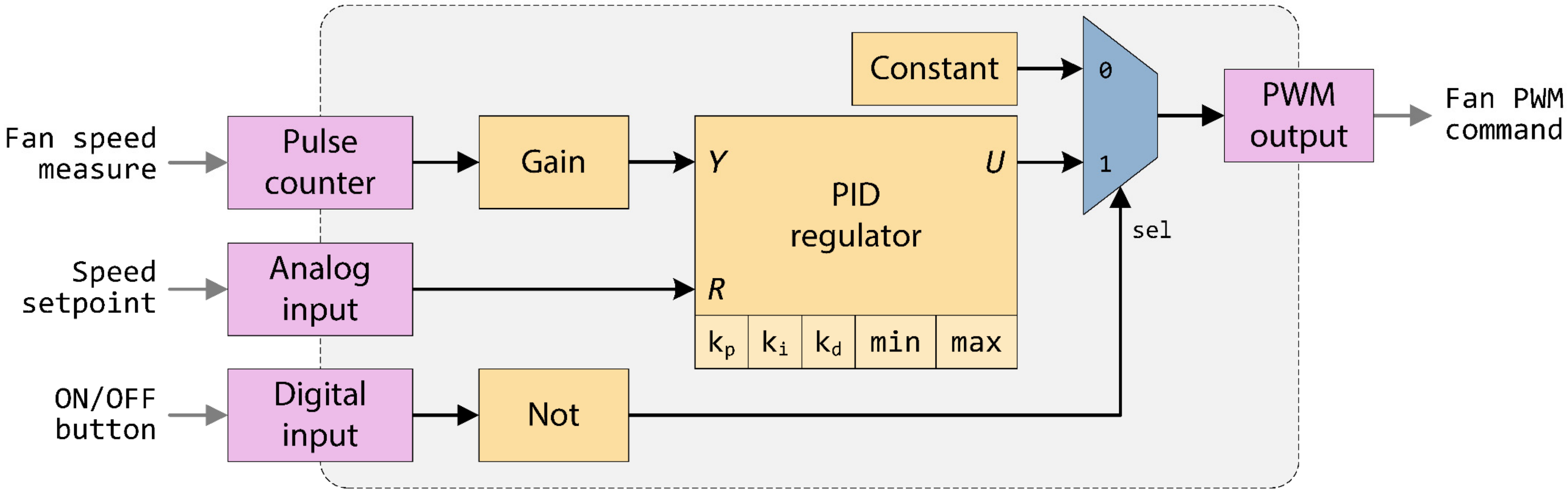
3. EXPERIMENTAL RESULTS

More complex application

Speed regulation of a fan



Real world application
Speed regulation of a fan



Regulation application

SCALA + DSL

```
val pid = PID(1.0, 0.5, 0, 50, 4000) // Inputs
val pulse = PulseInputCounter(Pin('B', 9)).out
val setpoint = Stm32stkIO.adc1.out

val speedGain = SpeedGain(4000*45) // Logic
val mux = Mux2[uint16]()
val not = Not()

val pwm = Stm32stkIO.pwm3 // Output

// PID input measured from the pulse counter
pulse --> speedGain.in
speedGain.out --> pid.measure

setpoint --> pid.setpoint // Setpoint from the potentiometer

Constant(uint16(50)).out --> mux.in1 // Stop the fan
pid.out --> mux.in2
Stm32stkIO.btn1.out --> not.in
not.out --> mux.sel

mux.out --> pwm.in  // Fan PWM command
```

Synthesis

- Embedded systems language prototype
  - DSL dataflow (block diagram, model)
  - No low-level C/C++ code required
- Several components available in the framework, extensible
- New hardware targets can be easily added
- Test applications working as expected

Limitations

- Sequential execution model can be too restrictive
  - Dataflow applications cannot have cycles

    Solution

    ⇨ Upgraded to an event-driven asynchronous dataflow
      - More complex scheduler
      - Execute the graph in a network of processes
- Simulation in QEMU is limited
  - Not all MCU peripherals are available yet

Future work

- More complex and complete components
- Multi-tasking OS to remove IPO limitations
- New hardware targets
- Lightweight-modular staging
- Web-based graphical editor

Thank you for your attention !

https://github.com/hevs-isi/especial-frontend