

Daisy Chain Protocol Development in Force-Guiding Particle Chains for Shape-Shifting Displays Network Protocol Implementation

Raoul Rubien
`rubienr@sbox.tugraz.at`

Institute for Technical Informatics
Graz University of Technology

5th August 2016

What this session covers

the development tool chain
simulation
protocol implementation

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Underlying work

Force-Guiding particle Chains for Shape-Shifting Displays[1]
development board
grid board

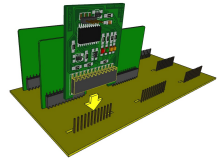
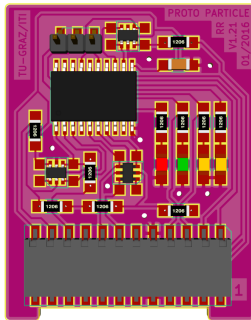
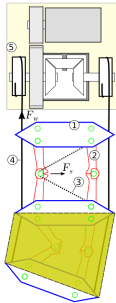


Figure 1: *underlying work*

Project extent

daisy chain protocol development (OSI)

Physical Layer, Data Layer, Network Layer

actuation scheduling and execution

time synchronization

runtime compensation of RC-oscillator discrepancy

Project constraints

exploit actuation wires for protocol communication

single communication entry point to the network

daisy chained network

Planning the development work flow

Aim

i) How to guarantee

good **code quality**?

ii) How to **test**?

iii) How how to **speed up**

development?

Solution

by continuous **testing**

by **simulation**

by using customized

tool chain,
i) and ii)

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

The Tool Chain

Tool chain overview

IDE independent

multiple projects can be integrated

CMake provides all necessary make targets such as deployment to real MCU and simulation targets

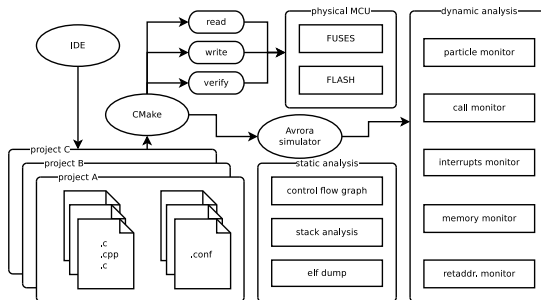


Figure 2: *tool chain overview*

CMake targets in detail

CMake implementation provides various targets

dynamic analysis (monitoring) of:

- wires

- any SRAM register

- I/O ports

- interrupts

- function calls

- stack overflow

- statistic reports (profiling): memory writes, function calls

static analysis:

- stack maximum size

- control flow graph (code optimization)

- inspecting CPU cycles per instruction (code opt.)

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

SimulationProtocol im-
plementationPhysical
Layer Coding

Results

Future Work

Simulation

How does a simulation framework help - I

the simulation framework simulates a whole network
 each node is served by a dedicated thread
 any node is defined by an abstract PCB model (platform)
 each platform is associated with the firmware, whereas
 the firmware is the same as for a real MCU

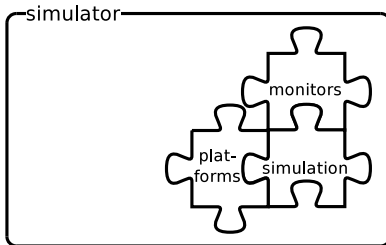


Figure 3: *simulator structure*

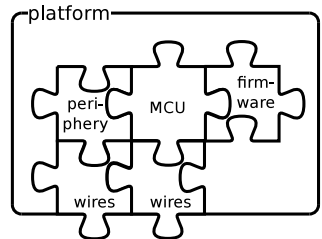


Figure 4: *platform structure*

How does a simulation framework help - II

the simulation output is

inspected by JUnit tests

interpreted by the developer

used to visualize signals, wires, variables, state changes and much more

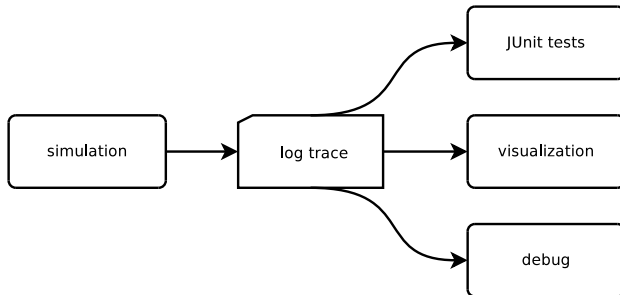


Figure 5: *work flow*

Avrora¹ simulation trace

may also be used in other tools such as:

- friction simulation
- network visualization

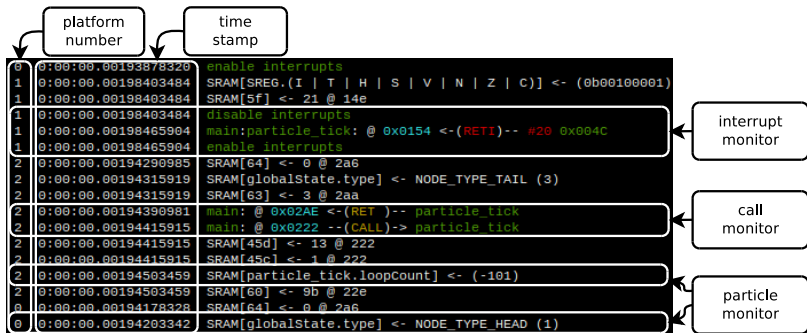


Figure 6: simulation trace

¹<http://compilers.cs.ucla.edu/avrora>

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Trace visualization

(2×2) network (interactive chart example [\[1\]](#) or [\[2\]](#)):
wire signals, arbitrary *uint8_t* debug output, internal global
variables

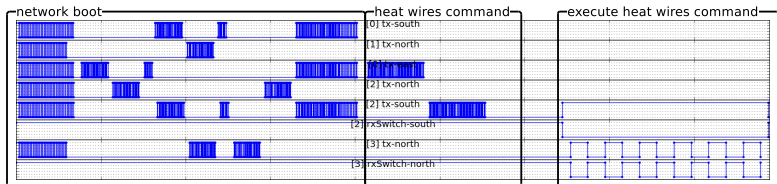


Figure 7: *node visualization*

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Control flow graph

code otimizing helper

inline vs. call

prove expected inlining result

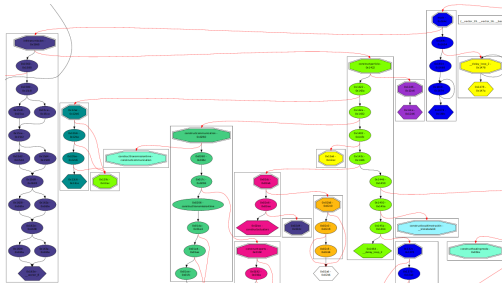


Figure 8: *control flow graph snippet*

square: interrupt
double octagon: procedure
hexagonal: blocks with return

edges: jumps, branches
red edges: calls
dotted: indirect calls or jumps

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Profiling

customized memory profiling
interrupt profiling

```

=={ Particle state profiling results for node 1 }=====
Address      Writes      Changes
-----
particle tick.loopCount  219/218
globalState.state  5/4
globalState.type  2/1
globalState.nodeId  1/0
globalState.northRxEvents  14/13
globalState.southRxEvents  14/13
globalState.flags( - | - | - | - | - | RECORD_RX_SOUTH | RECORD_RX_NORTH )  1/0
globalState.rxNorthByte1  1/0
globalState.rxNorthByte2  1/0
globalState.rxSouthByte1  1/0
globalState.rxSouthByte2  1/0
globalState.rxBitCounter  1/0
dirD.(D7 | D6 | STH_RX | D4 | D3 | NRTH_RX | D1 | D0)  0/0
portD.(D7 | D6 | STH_RX | D4 | D3 | NRTH_RX | D1 | D0)  1/1
MCUCR.(SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00)  1/1
dirA.(TP | STH_SW | A5 | STH_TX | LED | A2 | NRTH_TX | NRTH_SW)  1/1
portA.(TP | STH_SW | A5 | STH_TX | LED | A2 | NRTH_TX | NRTH_SW)  175/175
GCTR.(INT1 | INT0 | INT2 | - | - | - | IVSEL | IVCE)  2/2
SREG.(I | T | H | S | V | N | Z | C)  115/1

```

variable particle state:
5 writes, 4 changes

```

=={ Interrupt monitor results for node 0 }=====
Num  Name      Invocations  Separation  Latency  Wakeup
-----
1  RESET          0
2  INT0           0
3  INT1         43  353.5476  10.232558  0.0

```

interrupt statistics

Figure 9: Avrora profiling monitors example

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Code optimization helper

inspecting cycles per instruction example

```

case STATE_TYPE_PREPARE_FOR_SLEEP:
    if (ParticleAttributes.directionOrientedPorts.north.txPort->isTransmitting ||
26ca:    e0 91 be 01    [[LDS -> 2]]    r30, 0x01BE
26ce:    f0 91 bf 01    [[LDS -> 2]]    r31, 0x01BF
26d2:    85 85        [[LDD -> 2[2]]   r24, Z+13    ; 0x0d
26d4:    80 fd        [[SBRC -> 1/2/3]]  r24, 0
26d6:    37 c0        [[RJMP -> 2]]    .+110    ; 0x2746 <particleTick+0x288>
        ParticleAttributes.directionOrientedPorts.east.txPort->isTransmitting ||
26d8:    e0 91 cc 01    [[LDS -> 2]]    r30, 0x01CC
26dc:    f0 91 cd 01    [[LDS -> 2]]    r31, 0x01CD
        break;
  
```

SBRC @ 26d4 takes 1, 2 or 3 cpu cycles

Figure 10: *cycles per instruction inspection*

Protocol Implementation

Recall objectives

What is needed?

A **daisy chain** protocol capable of **executing commands synchronously** at a **given time** using **actuator wires**.

That means roughly

- 1) Layer 0
- 2) Layer 1
- 3) Layer 2

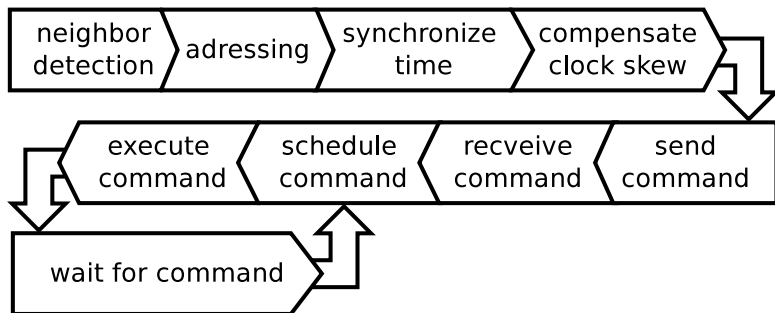


Figure 11: Layer 2: protocol process

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

State driven process implementation

```
void main() {
    // state machine call
    while(true) { process(); }
}
```

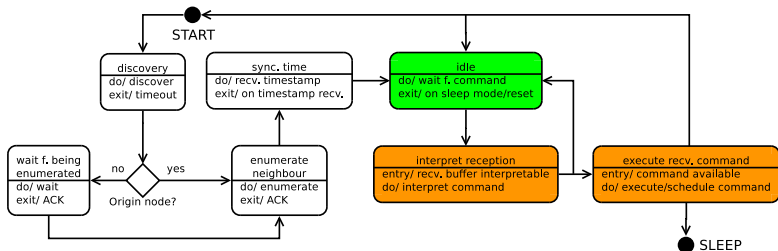


Figure 12: *protocol states*

Firmware sequence diagram

reception, transmission and processing are independent
may occur effectively concurrent

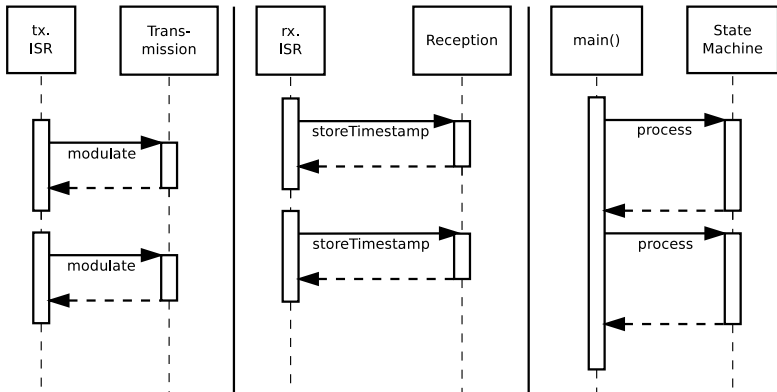


Figure 13: *firmware sequence diagram*

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Encoding

opted for Manchester coding

no common tx/rx clock needed

can be exploited to calculate clock skew

simple to implement

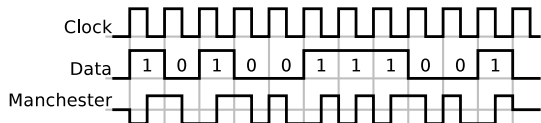


Figure 14: *Manchester coding: $data = clock \oplus manchester$*

Decoding implementation

store timestamp of signal flank to circular buffer

timestamp equals 16bit timer-counter value

decoder removes from buffer

interpreter: interprets on interpret-able decoded buffer

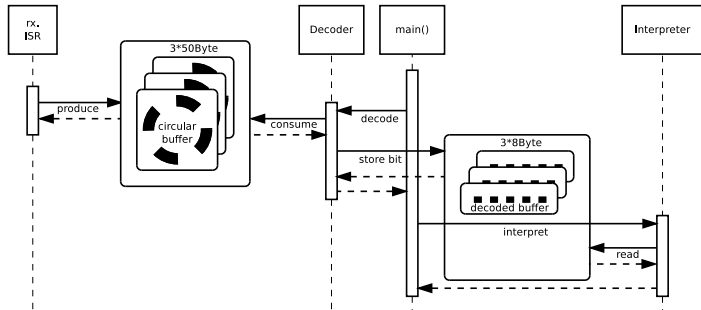


Figure 15: *decoding sequence diagram*

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Network boot example

- 1st phase: neighbor discovery
- 2nd phase: address assignment
- 3rd phase: enumeration finished
- 4th phase: time synchronization

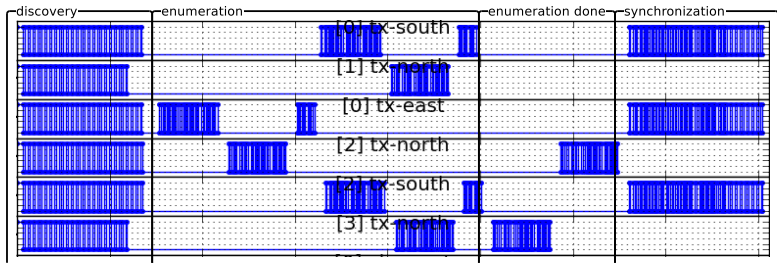


Figure 16: *network boot visualization*

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work

Addressing example

- 1st phase: network discovery and address assignment
- 2nd phase: send "heat wires at specific time" command
- 3rd phase: execute command

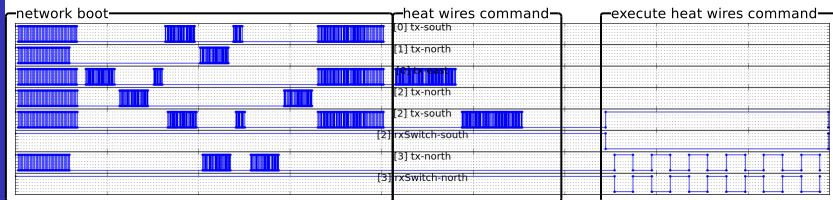


Figure 17: "execute command" network visualization

Results

- i) An extend-able **daisy chain network protocol** that
- ii) **executes** actuation **commands**
at a given time **synchronously**, and
- iii) a development **tool chain** that sustains:
 - iii.a) **simulation**,
 - iii.b) **debugging** and
 - iii.c) JUnit **testing**.



Because programmable matter matters!

Future work

Phy. Layer: considering partially implemented parity bit

Network Layer: fault detection, fault tolerance

actuation: power adjustment (PWM tuning)

time compensation: evaluation of calibration accuracy

firmware replication: customize boot loader

forward/backward shaping of 1st row

Introduction

Underlying
workUnderlying
work

Project extent

Approach

Tool Chain

Simulation

Protocol im-
plementationPhysical
Layer Coding

Results

Future Work



M. Lasagni and K. Römer, "Force-guiding particle chains for shape-shifting displays," *CoRR*, vol. abs/1402.2507, 2014.