Daisy Chain Communication Protocol for Shape-Shifting Displays

Network Protocol Implementation

Raoul Rubien
rubienr@sbox.tugraz.at

Institute for Technical Informatics
Graz University of Technology

5th August 2016

Outline

1. Introduction
   - Principle
   - Limitation & Motivation
2. Methodology
   - Development Work Flow
   - Simulation
   - Protocol Implementation
3. Results
4. Future Work

Particle network principle

- Particle Chains for Shape-Shifting Displays [1]
  - network design
  - particle development board

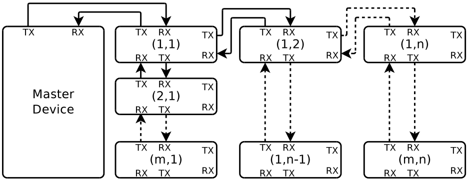


Figure 1: *network topology*

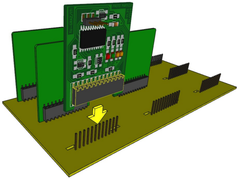


Figure 2: *development hardware*

[1] http://arxiv.org/abs/1402.2507

Limitation & Motivation

- No existent protocol for the given network topology allowing
  - scheduling of actuation commands,
  - network synchronization and
  - runtime drift compensation.

Conclusion

- Develop a lightweight network protocol tailored for the given structure.

Outline

Methodology

- daisy chain protocol development
  - Physical Layer
    runtime drift compensation
  - Data Link Layer
  - Network Layer
    time synchronization
    actuation scheduling
- simulation supported development process
  - automated testing

Project constraints

- use actuators for protocol communication
- single network communication entry point
- daisy chained network without global time awareness

Development work flow - I

- How to guarantee good **code quality**?
- How to **speed up** development?

Solution

- simulate
  - verify result - JUnit tests
  - visualize result: signals, variables, pins, interrupts, ...
  - inspect result - read log

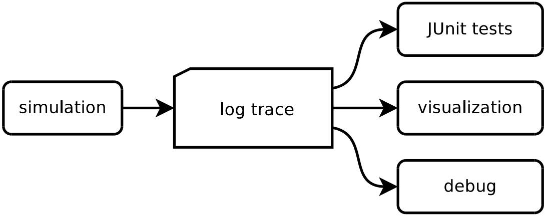


Figure 3: *work flow*

Development work flow - II

- IDE independent tool chain
  - integrating multiple projects
  - allows deployment to real MCU
  - starts simulation

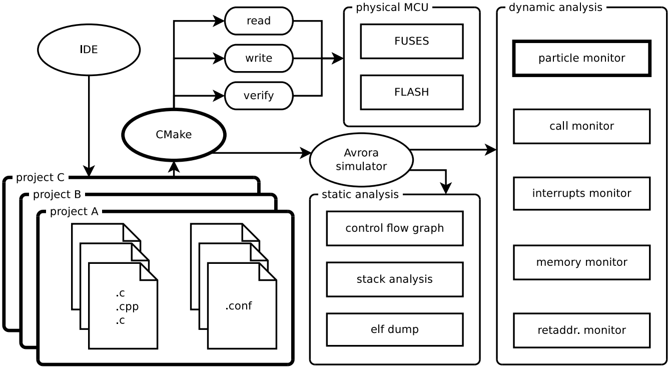


Figure 4: *tool chain overview*

Simulation

- framework simulates a whole network
- nodes are defined abstracted as platforms
- each platform is associated with a firmware
- firmware is compiled as usual for physical MCU - ATmega16

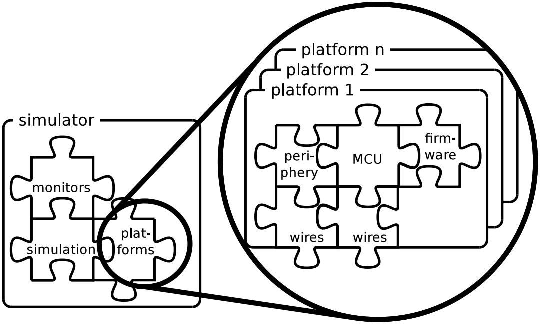


Figure 5: *simulator structure*

Simulating with Avrora[2]

- simulation result may also be used by other tools
  - friction simulation
  - network visualization

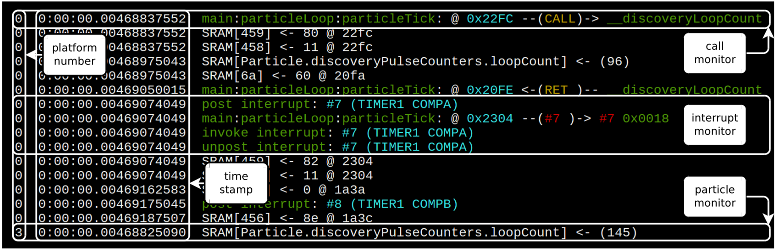


Figure 6: *simulation trace*

[1]http://compilers.cs.ucla.edu/avrora

Protocol implementation

- state machine
- permanently called

```
void main() {
    while(true) { process(); }
}
```

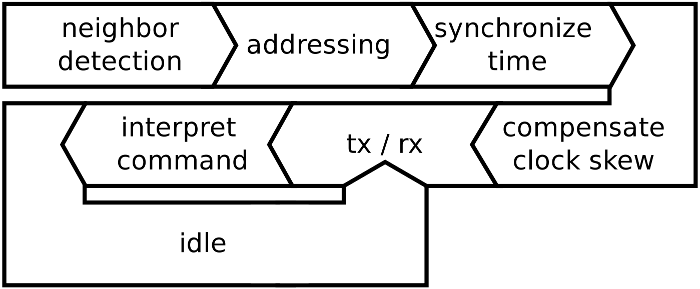


Figure 7: *protocol process*

Firmware sequence diagram

- reception, transmission and processing are independent
- may occur effectively concurrent

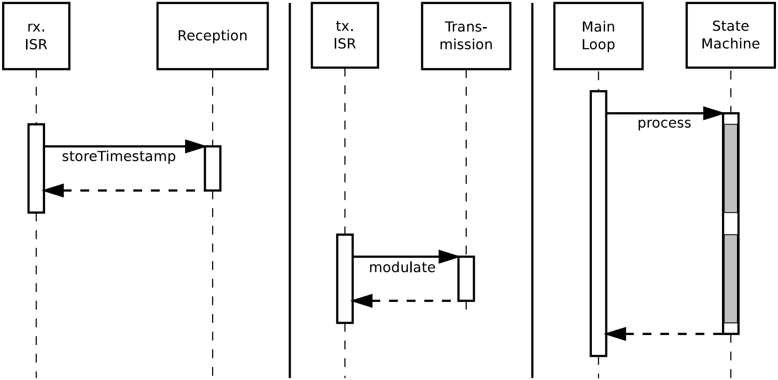


Figure 8: *firmware sequence diagram, gray states are blocking*

Encoding

- opted for Manchester coding
  - no clock wire needed
  - can be exploited to calculate clock skew
  - simple to implement
    1^{st} event buffering
    2^{nd} decoding

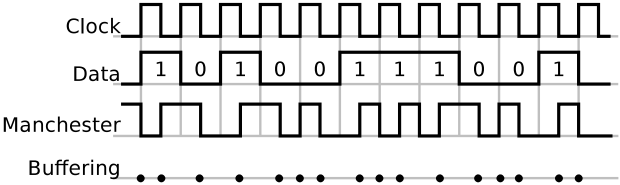


Figure 9: *Manchester coding: data = clock ⊕ manchester*

Reception implementation

- store timestamp of signal edge to circular buffer
  - 16*bit* timer-counter value
- decoder consumes from buffer and converts to bit
- interpreter interprets decoded buffer interpret-able

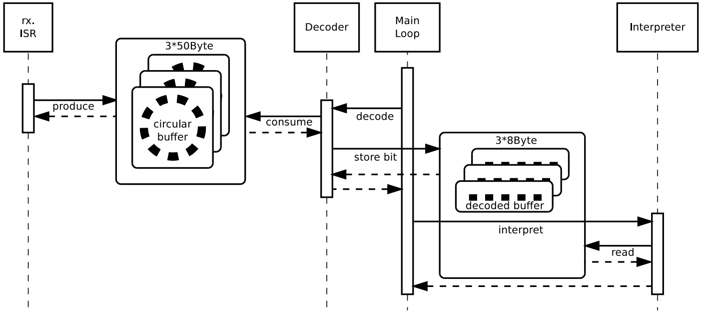


Figure 10: *decoding sequence diagram*

Actuation example

1^{st}	phase:	neighbor discovery
2^{nd}	phase:	address assignment
3^{rd}	phase:	enumeration finished
4^{th}	phase:	time synchronization
5^{nd}	phase:	send "heat wires at specific time" command
6^{rd}	phase:	execute command

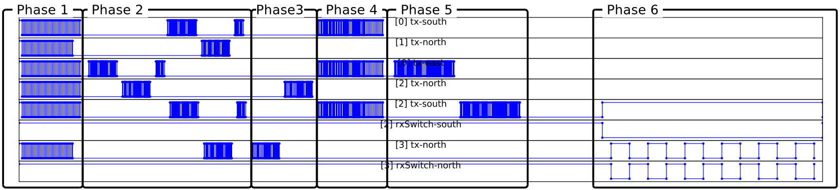


Figure 11: *network visualization*

Outline

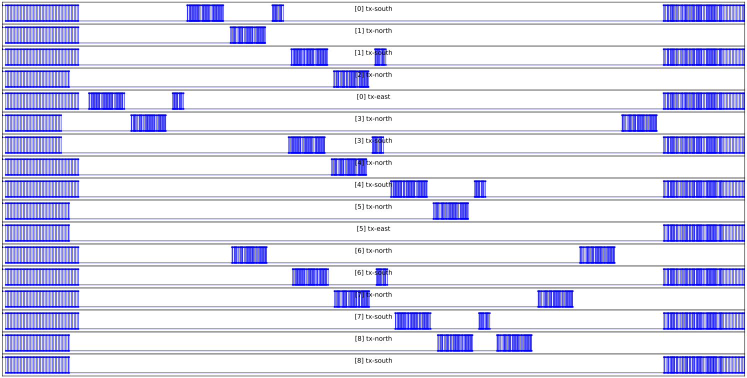
Results

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

Outline

Future work

- Physical Layer
  - considering partially implemented parity bit
- Network Layer
  - fault detection, fault tolerance
- time compensation
  - evaluate calibration accuracy
- remote programming
  - customize boot loader (firmware replication)
- forward/backward shaping of 1^{st} row



Interactive video example:
https://github.com/ProgrammableMatter