Daisy Chain Communication Protocol for Chains of Robotic Particles forming Shape-Shifting Displays

Raoul Rubien, BSc
Institute for Technical Informatics

Graz, January 16th, 2017

Part I - Hardware Implementation

Shape Shifting Display principle

- a display can approximate 3D surfaces
- a display consists of multiple foldable chains
- one chain outlines a slice

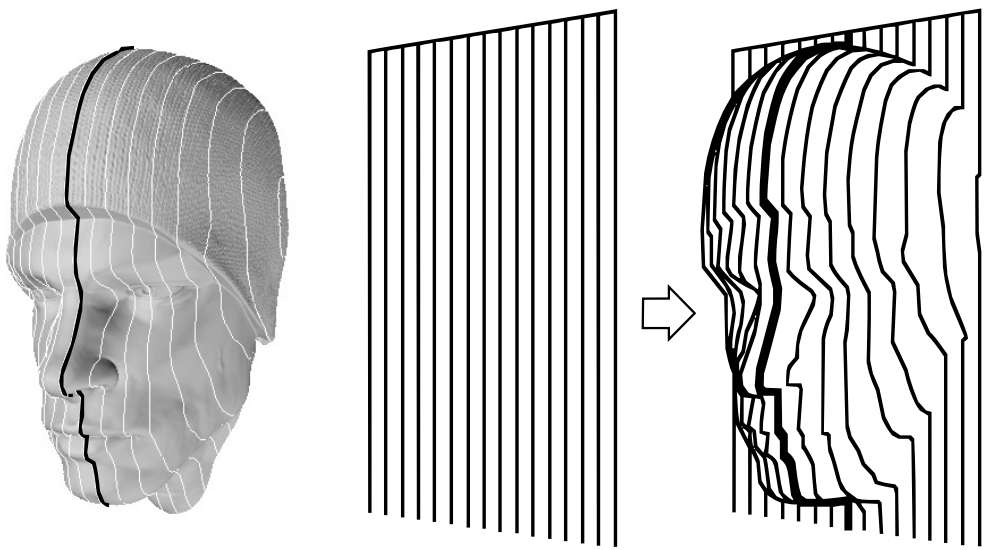


Figure 1: display principle [DOI 10.1145/2695664.2695932]

Motivation for communication protocol

- particles can actuate independently
- particles need to be coordinated among each other
  - in same chain
  - in whole system

$\Rightarrow$ need of network for communication

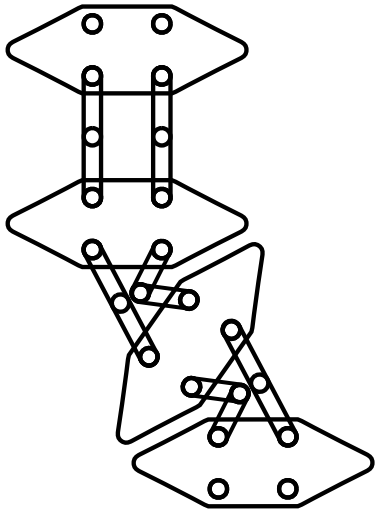


Figure 2: folded chain

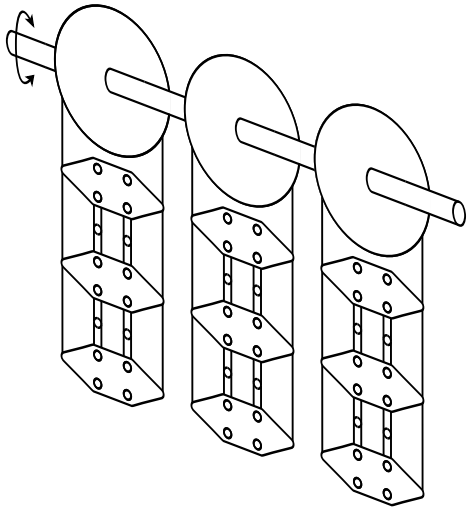


Figure 3: external force mechanism

[DOI 10.1109/IROS.2014.6943112]

Current implementation

- communication via power supply wires: 1-Wire protocol

Limitation

- when actuating no communication
- particle localization is costly (non scalable, brute force)
- 1-Wire communication limits load
- communication is limited to one particle at once
- local computation not possible

Our goal

Define a communication protocol that overcomes those limitations.

Requirements

- decouple communication and power supply
- without extra wires
  - exploit same actuator wires between consecutive particles
- simplify particle localization
  - introduce addressing mode: particle / particle range
- synchronous actuation
  - synchronous time among particles: deviation $\leq \pm 1$ sec.
- tiny particle
- cheap and small MCU for local computation

Constraints

- arising forces may break the system
  - actuation is planned globally
  - a target shape may be split into sequence of actuations
- particles are not aware of
  - others' state
  - network geometry
  - actuation plan
- Why is wireless or UART communication not applied?
  - makes the particle complex
  - localization not simple
  - no access to timings as with Manchester coding

Network design

- daisy chained participants

Advantages

+ simple discovery
+ simple to implement
+ no media access control
+ no loops
+ no dynamic routes

Disadvantages

- failure prone: 1 erroneous particle disconnects whole chain

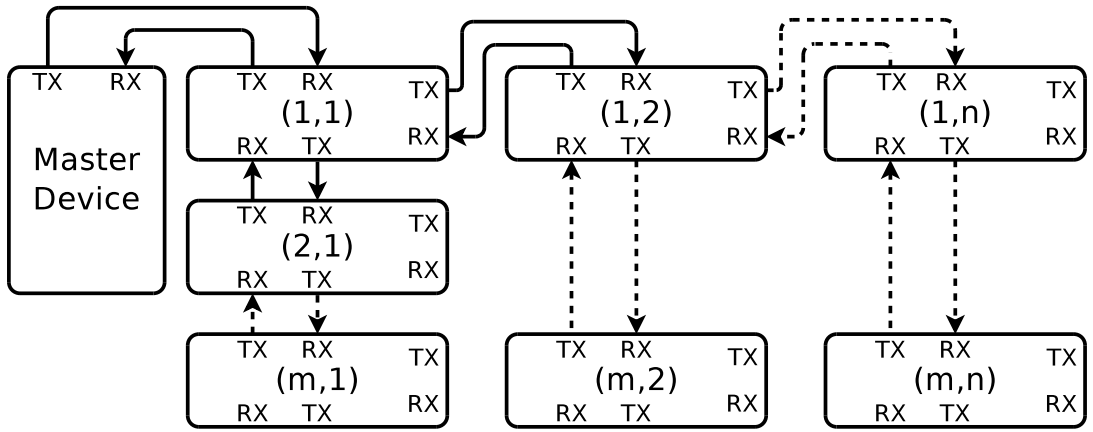


Figure 4: network topology

Hardware design

- particle hardware that applies actuators for communication
- applicable to the network structure

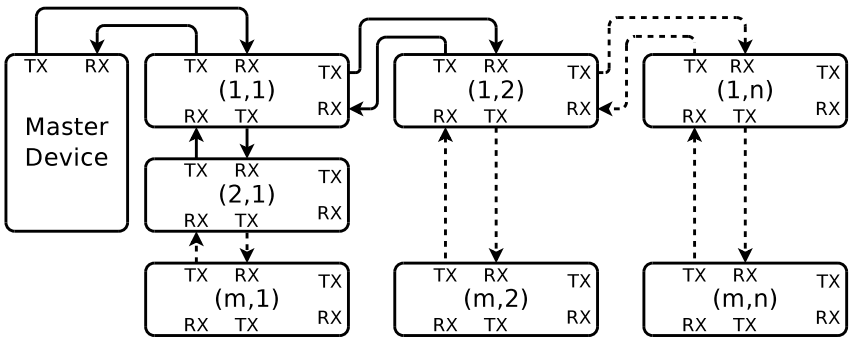


Figure 5: network topology

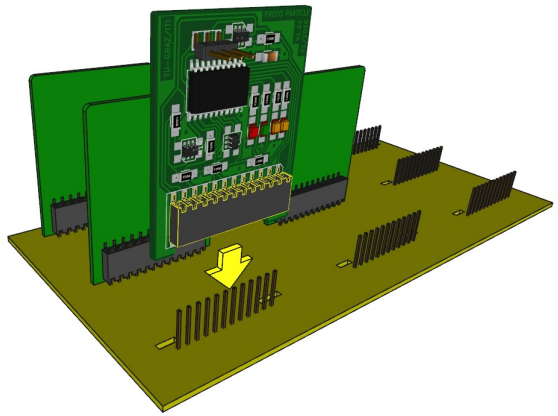


Figure 6: development hardware

Exploiting actuators

- using actuator wires for communication

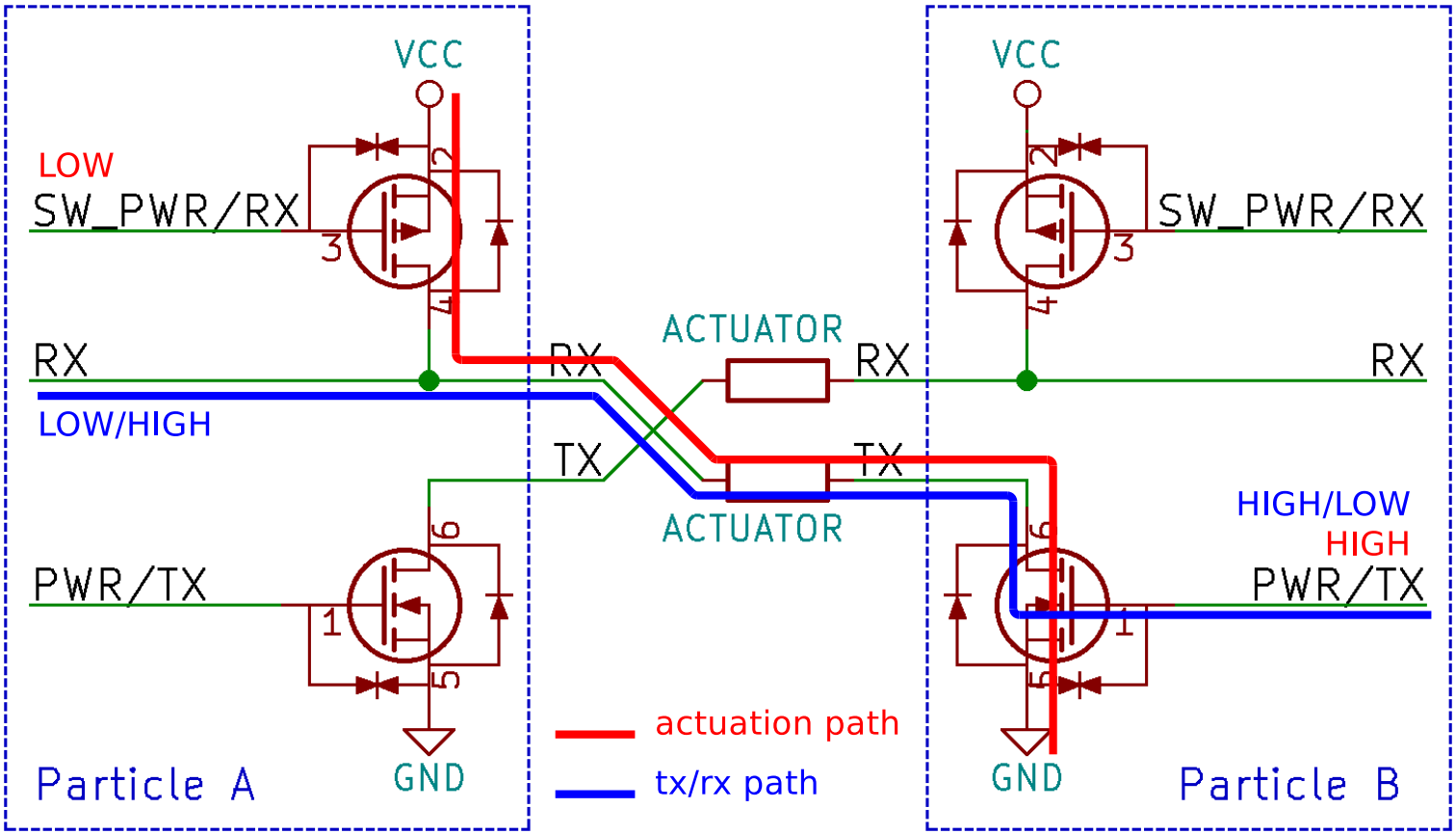


Figure 7: actuator and transmission/reception wiring

Requirements

- ☑decouple communication and power supply
- ☑without extra wires
  - ☑exploit same actuator wires between consecutive particles
- ☐simplify particle localization
  - ☐introduce addressing mode: particle / particle range
- ☐synchronous actuation
  - ☐synchronous time among particles: deviation $\leq \pm 1s$
- ☑tiny particle
- ☑cheap and small MCU for local computation

Part II - Protocol Implementation

Particles discovery

- transmit / receive mechanism
- simple position discovery
  - top-left, in chain, last of chain
- top-down enumeration
- bottom-right most reports feedback

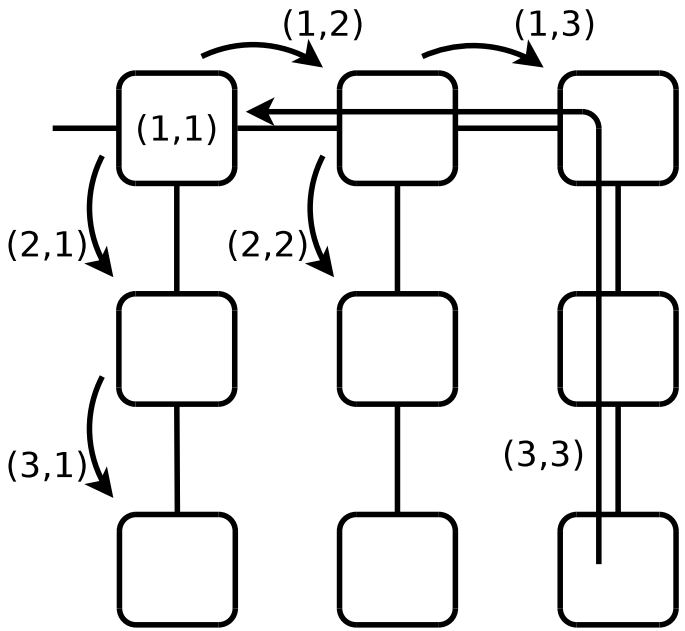


Figure 8: addressing sequence

Addressing mode

- single particle
- particle range

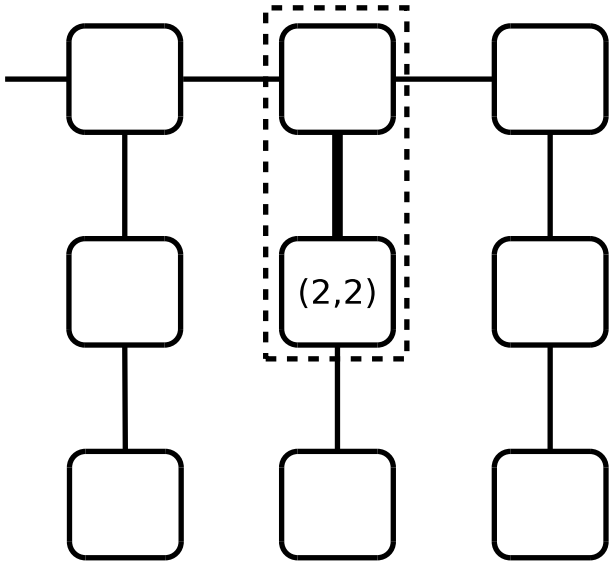


Figure 9: single particle

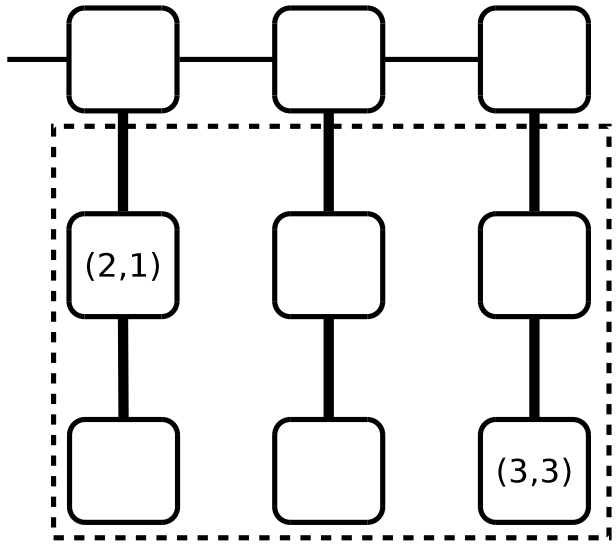


Figure 10: particle range

Synchronization

- time synchronization
  - global network time
  - local time counting
- clock skew compensation
  - internal RC clock drift / inaccuracy

$\Rightarrow$ exploit line code for timing adjustment

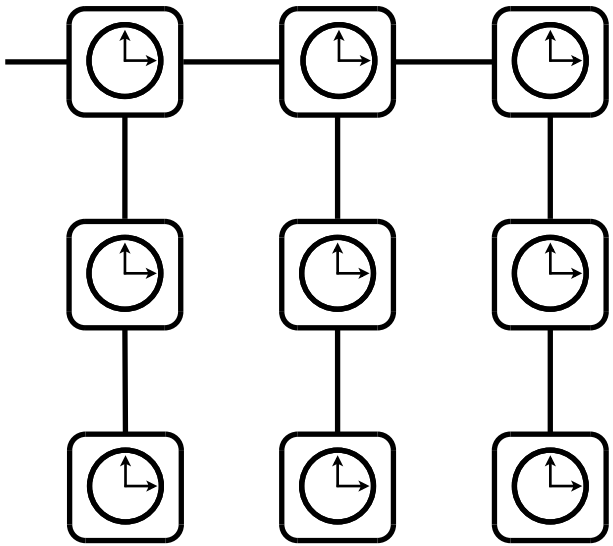


Figure 11: synchronized network time

Time synchronization

- top-down approach
- time is propagated to consecutive particles

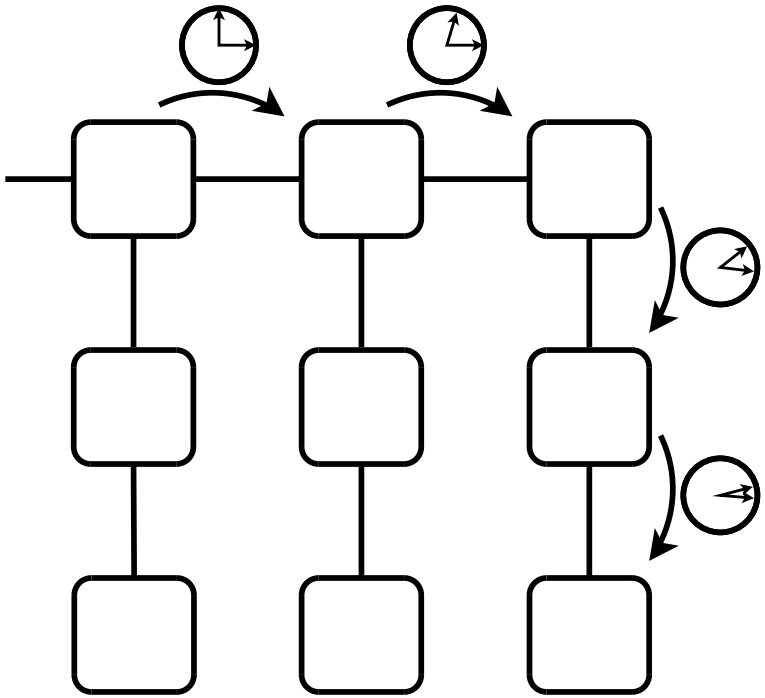


Figure 12: synchronization process

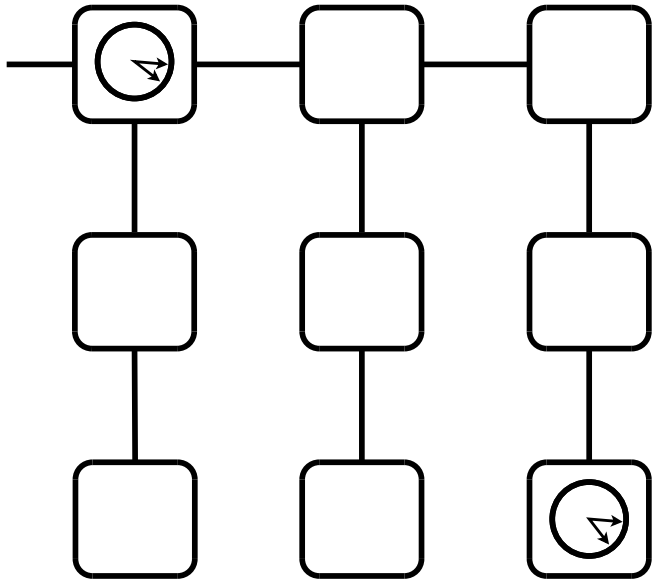


Figure 13: synchronized time

Line code - clock skew compensation

- opted for Manchester coding
- no additional clock wire needed
- can be exploited to adjust timings
- simple to implement

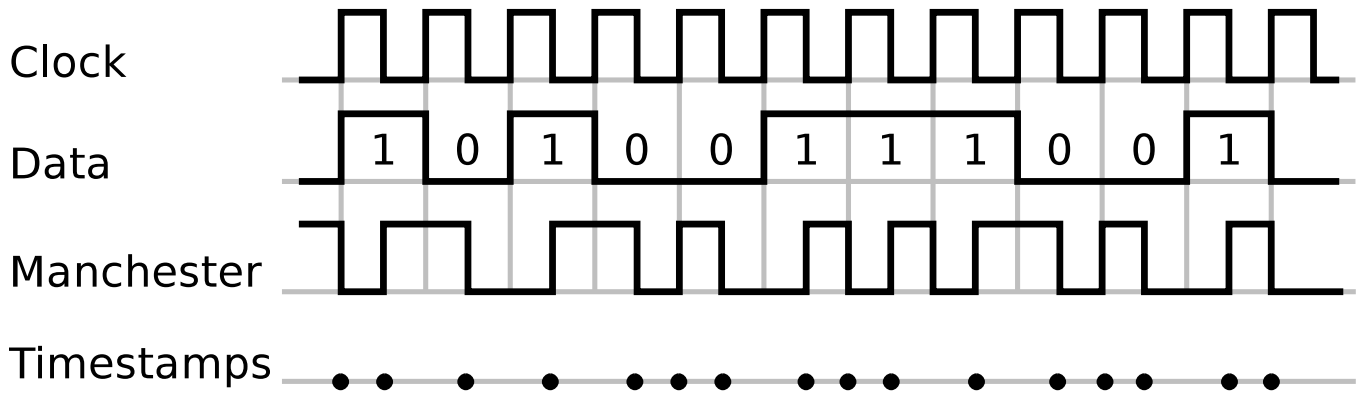


Figure 14: Manchester coding: $data = clock \oplus manchester$

Requirements

- ☑decouple communication and power supply
- ☑without extra wires
  - ☑exploit same actuator wires between consecutive particles
- ☑simplify particle localization
  - ☑introduce addressing mode: particle / particle range
- ☑synchronous actuation
  - ☑synchronous time among particles: statistical dev. $\leq \pm 1s$
- ☑tiny particle
- ☑cheap and small MCU for local computation

Actuation example

1st	phase:	neighbor discovery
2nd	phase:	address assignment
3rd	phase:	enumeration finished feedback
4th	phase:	time synchronization (synchronous)
5th	phase:	send ”heat wires at specific time” command
6th	phase:	synchronous command execution

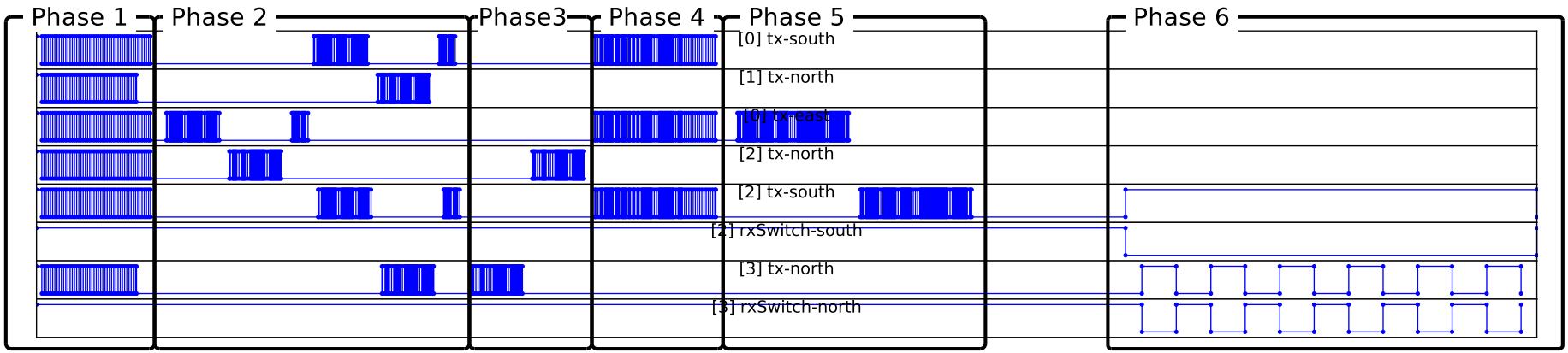


Figure 15: network visualization of simulated test case

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

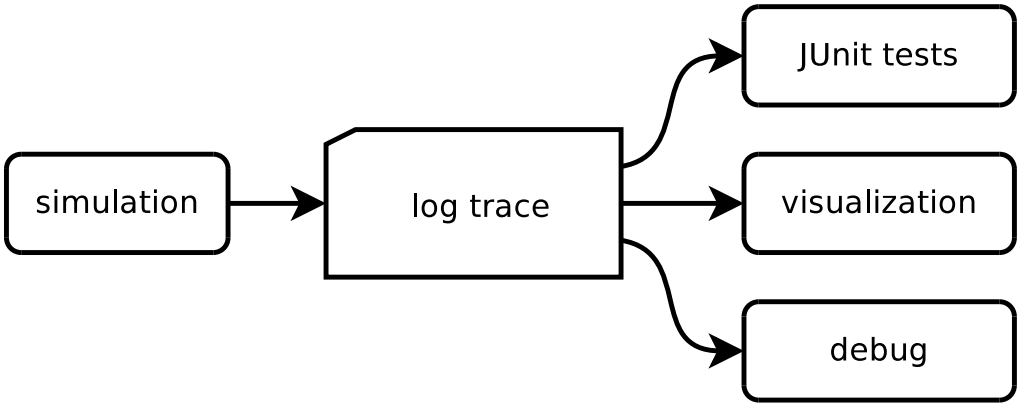


Figure 16: development work flow

Development work flow - II

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

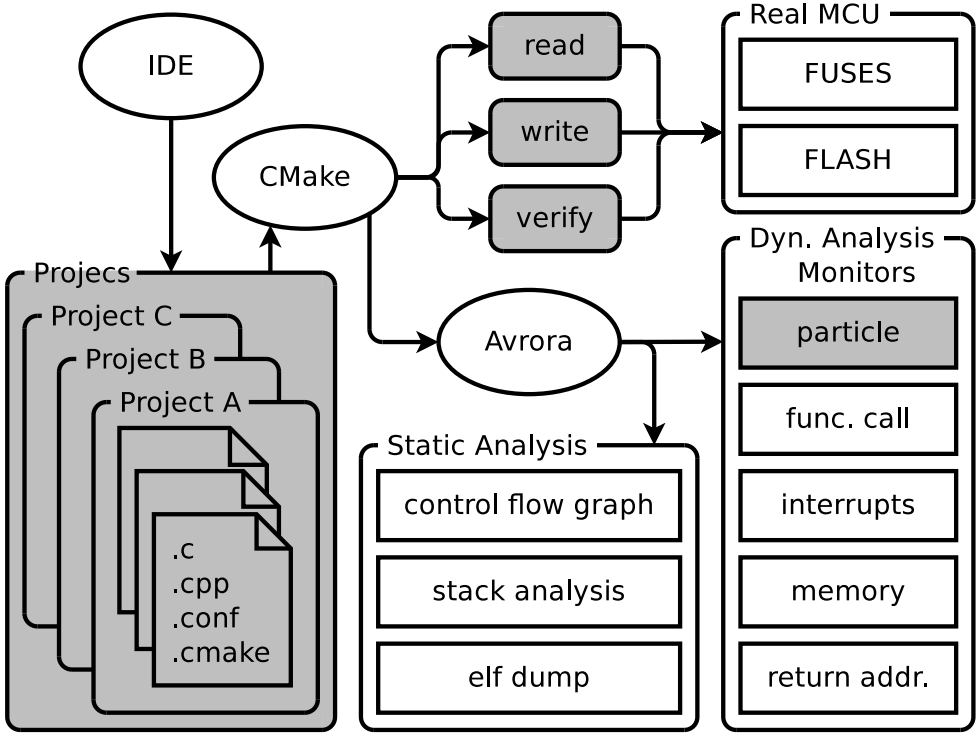


Figure 17: tool chain overview

The Avrora simulator

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

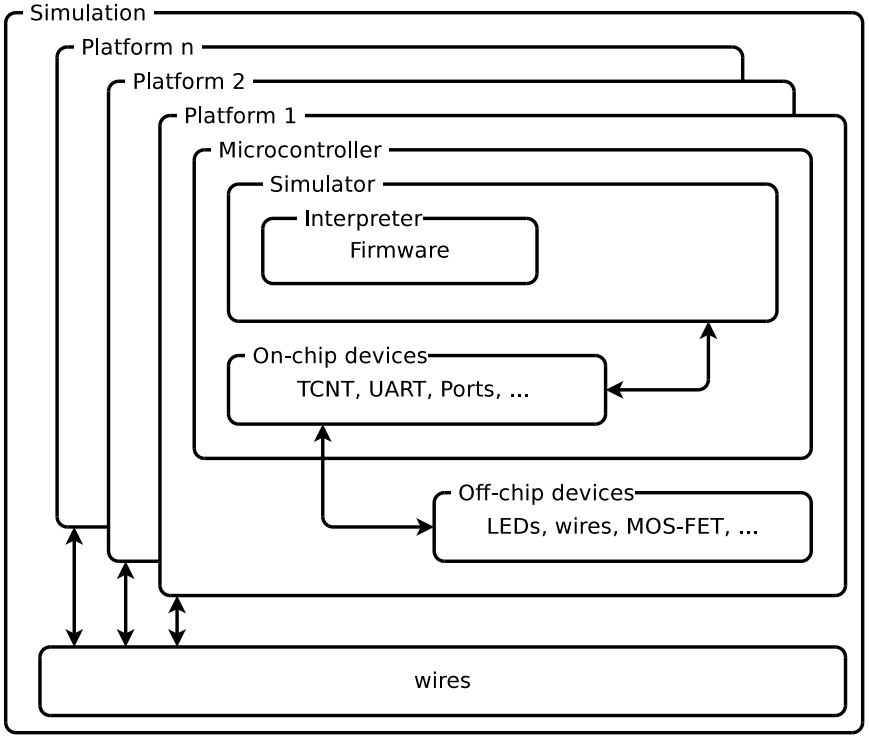


Figure 18: simulator structure

Simulating with Avrora

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

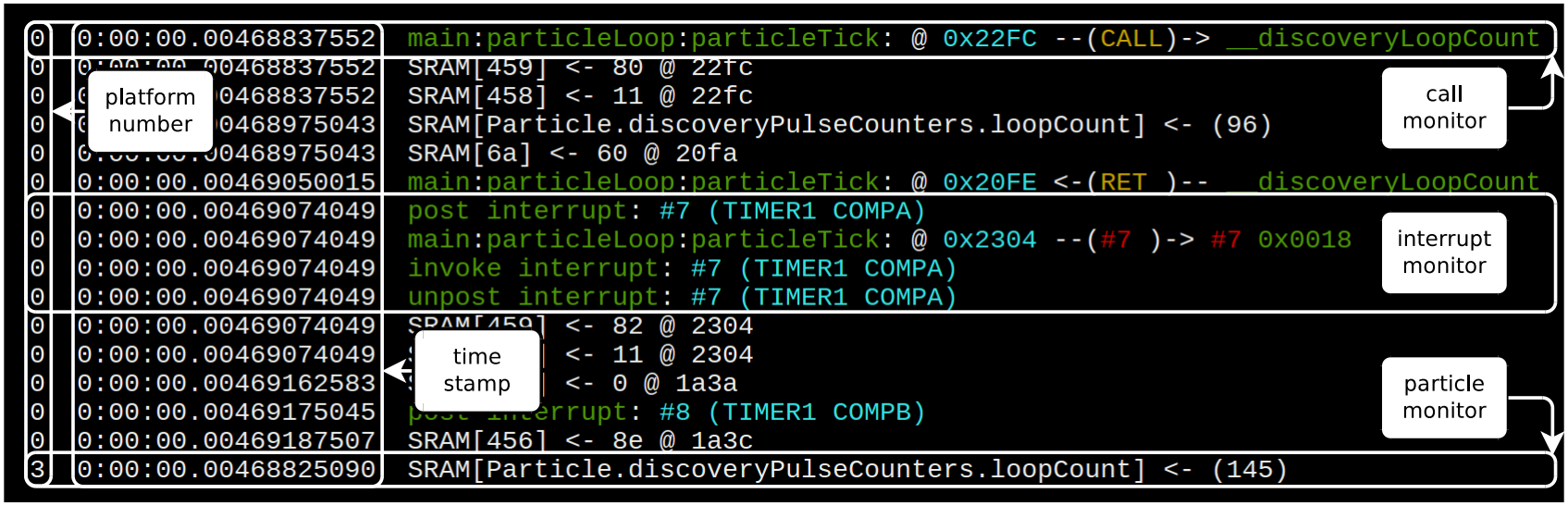


Figure 19: simulation trace

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

Work flow example video (1:20min)

- (3 × 3) network simulation → JUnit test → visualization

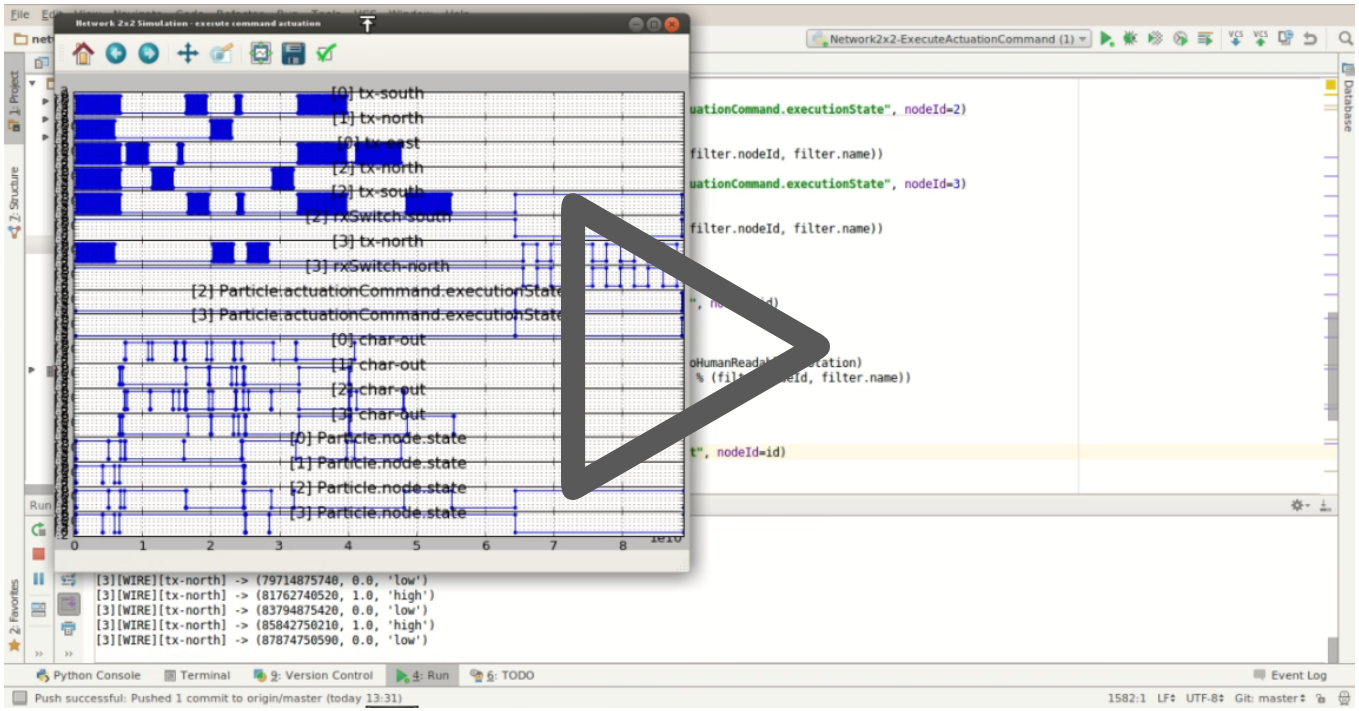


Figure 20: video of simulation, JUnit testing and visualization

Video `https://github.com/ProgrammableMatter/avrora-particle-platform`

Results

- real time network protocol
- initializes particle network
- time synchronization
- looses about 2ms accuracy per particle
  - in maximum network size $\leq \pm 1s$
- accuracy can be still optimized
  - stabilize *Vcc*
- communication speed $0.98 kByte/s$

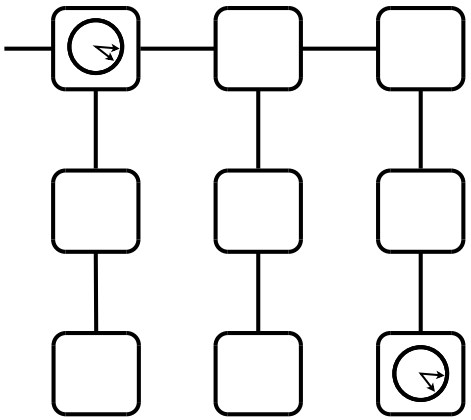
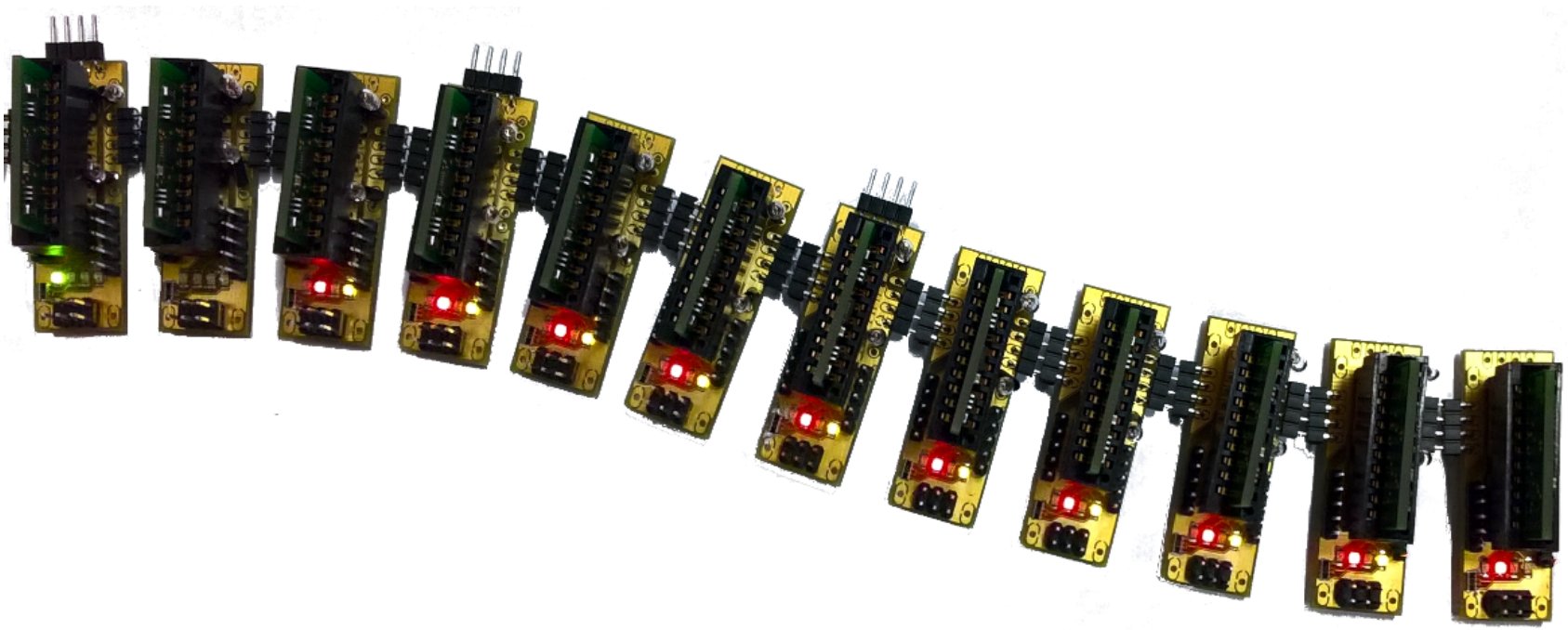
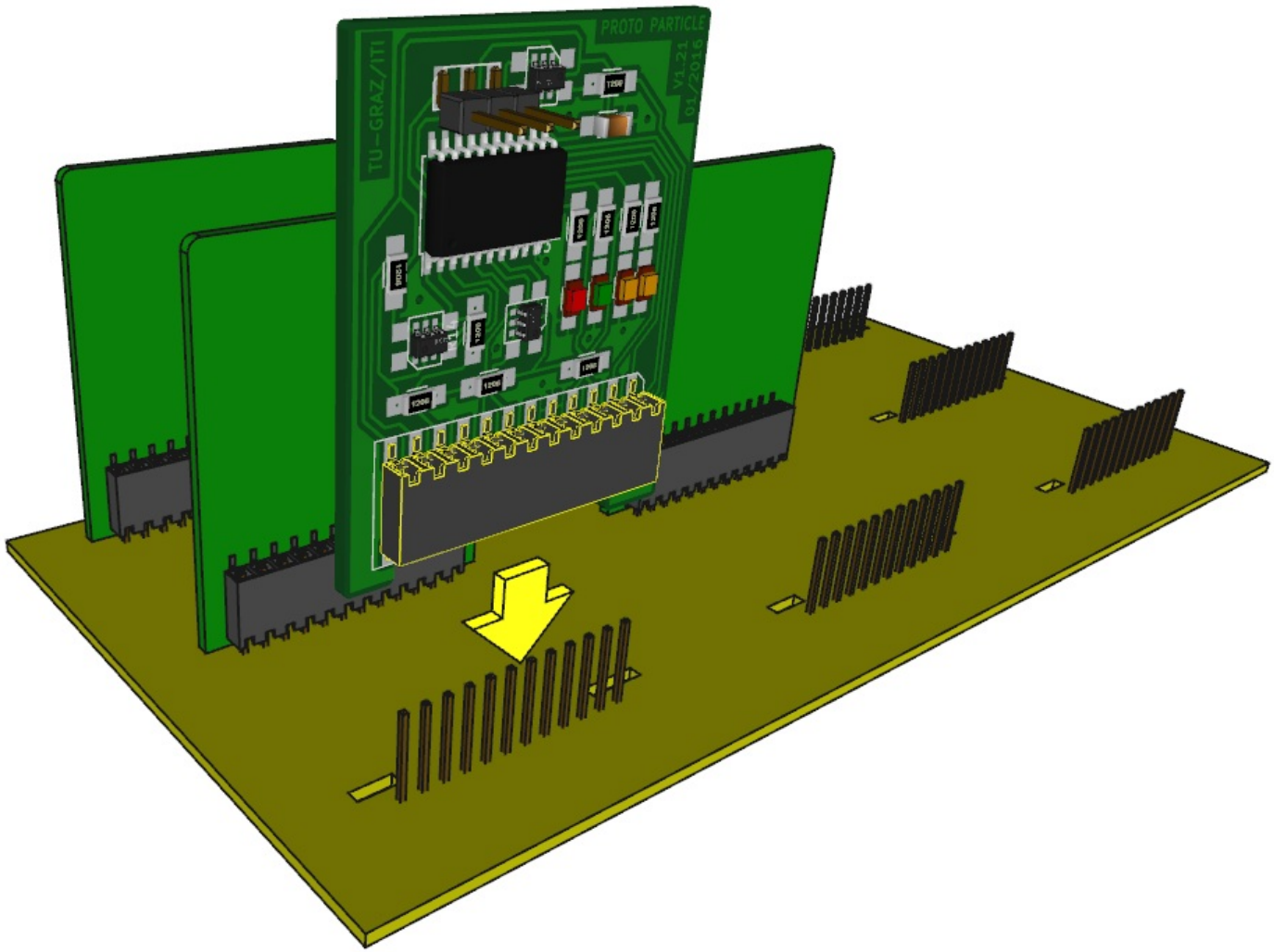


Figure 21: expected statistical sync. deviance of $\leq \pm 1s$ within a (255×255) network



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


TU-GRAZ/ITI
PROTO PARTICLE
V1.21
07/2016

Particle version 1.0

- chain of development particles
- light bulbs replaced actuators

Advantages

+ not mounted in chain mechanics
+ adjustable length via jumpers

Disadvantages

- time consuming assembly
- ATiny20: too less SRAM and FLASH

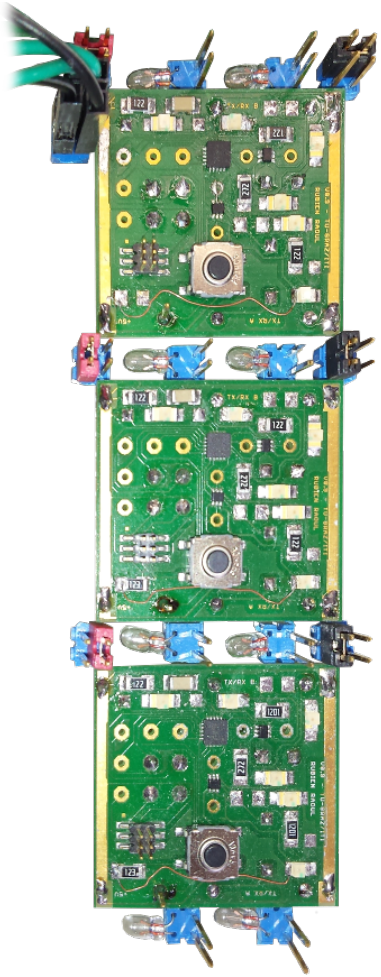


Figure 22: Version 1.0

Particle version 1.21

Advantages

+ simple to extend network
+ configurable network dimension
+ faulty particles can be replaced
+ higher particle density

Disadvantages

- bound to grid board size

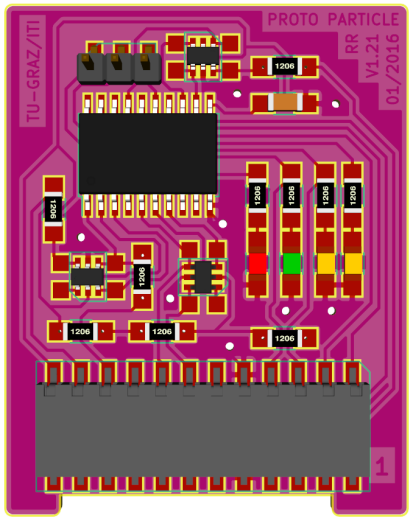


Figure 23: pluggable particle

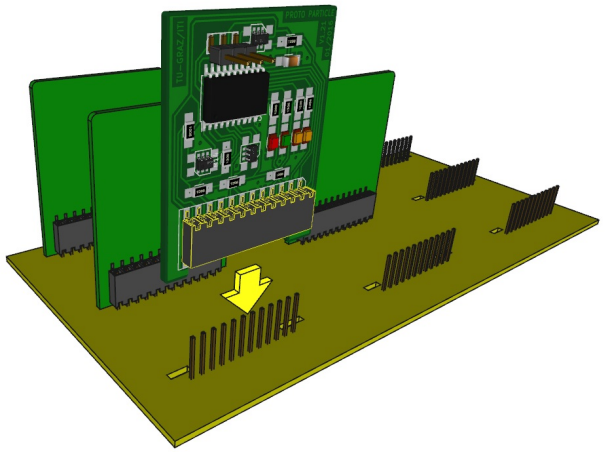


Figure 24: grid board

MCU requirements - capabilities

- three separate external interrupts
- self programmable EEPROM
  - remote programming (firmware replication)
- small MCU package

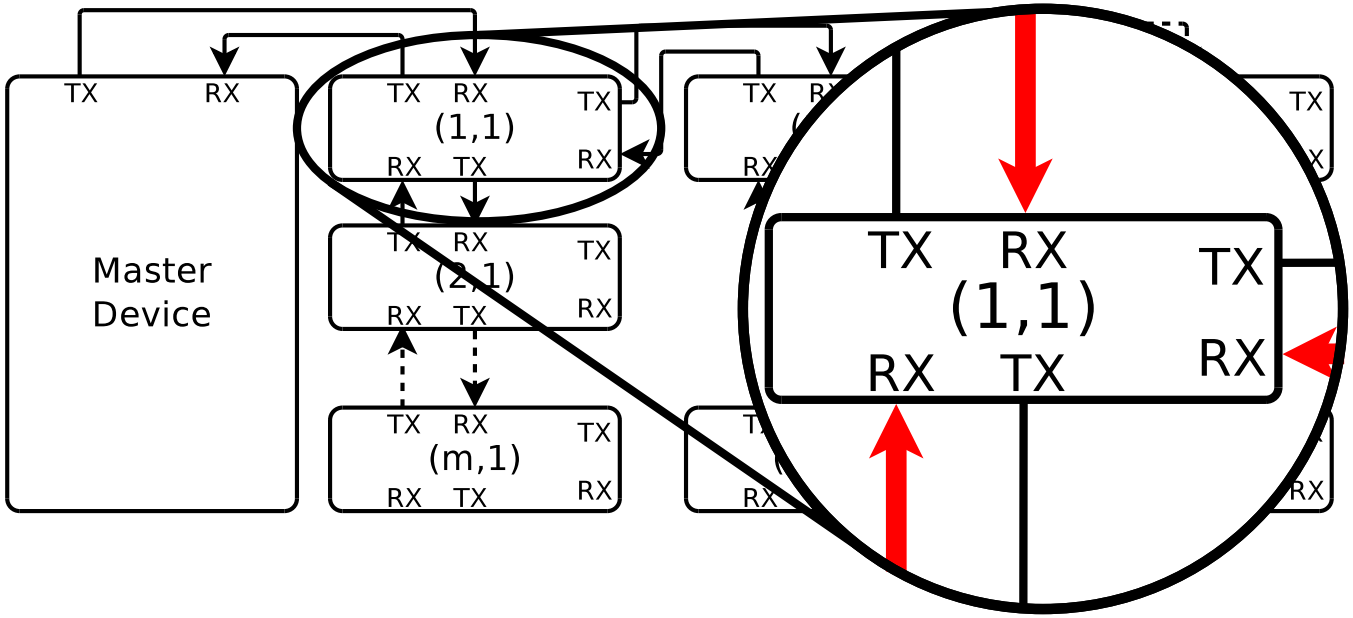


Figure 25: external interrupt inputs

MCU memory requirements - upper bound estimation

Flash

- expected max. firmware size: $\sim 4k$ SLOC
- estimated object code bytes per SLOC $\sim 4.0B$
- flash usage estimation:
  - $4k * 4B =$ **$\sim$16kB**

SRAM

- tx/rx buffers: 3 ports, 8byte
  - $3 * 8B * 2 =$ $16B$
- Manchester code decoding buffer
  - with 2 flank time stamps per bit
  - $3 * 8 * 8 * 2 * sizeof(uint16\_t)B * 0.75 =$ $576B$
- other global variables $200B$
- stack: max. 50 nested void function
  - calls with $\sim (1 * uint8\_t)$ argument
  - $50 * (1 + 2)B =$ $150B$
- SRAM estimation: **$\sim$950B**

SLOC [ISBN 0750686251]

Candidates

- candidates are ATTiny family MCUs having
  - $\geq 16kB$ flash and
  - $\geq 1kB$ SRAM

Comparison of used MCUs

	ATTiny20 (proof of concept)	**ATTiny1634**
# pin change int.	sufficient	sufficient
EEPROM	no	yes
flash	$2kB$	$16kB$
SRAM	$128B$	$1kB$
small package	$3mm \times 3mm$	$4mm \times 4mm$
alternative pkg.	no	yes, SOIC

Discovery - classification

- particle type classification
  - according to particle's connectivity

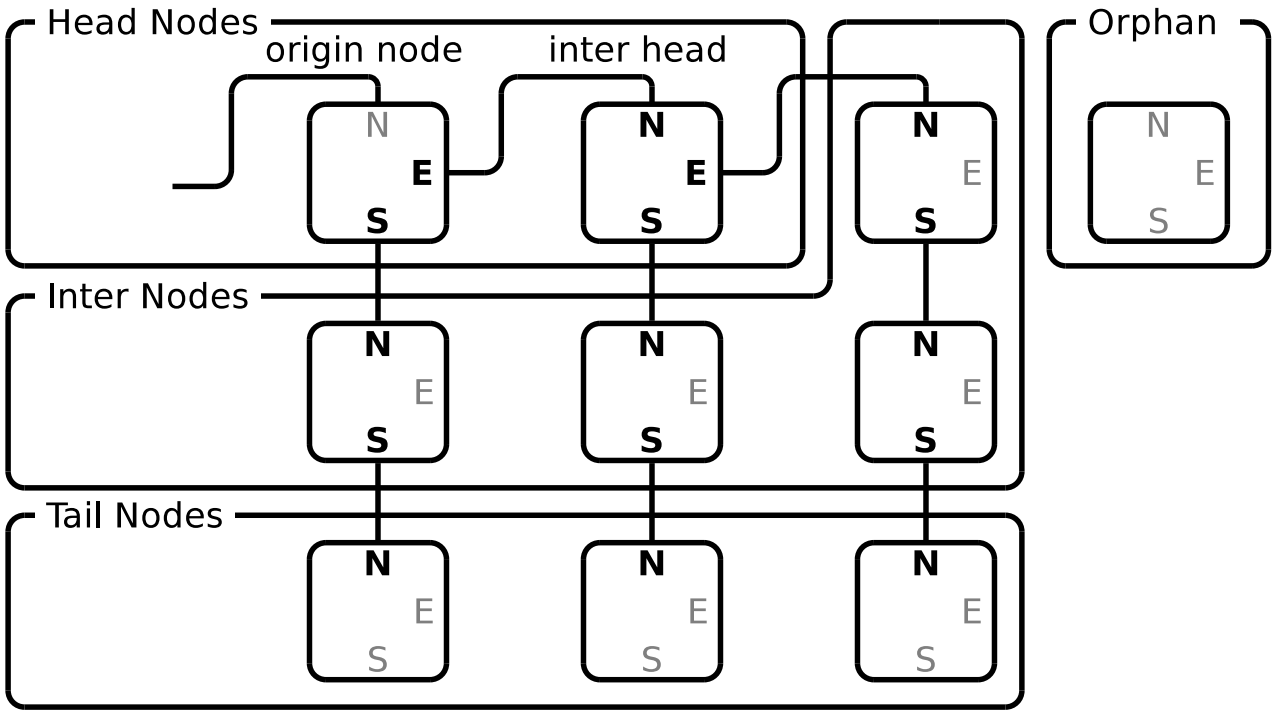


Figure 26: particle type classification matrix

Protocol implementation

- state machine
- permanently called

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

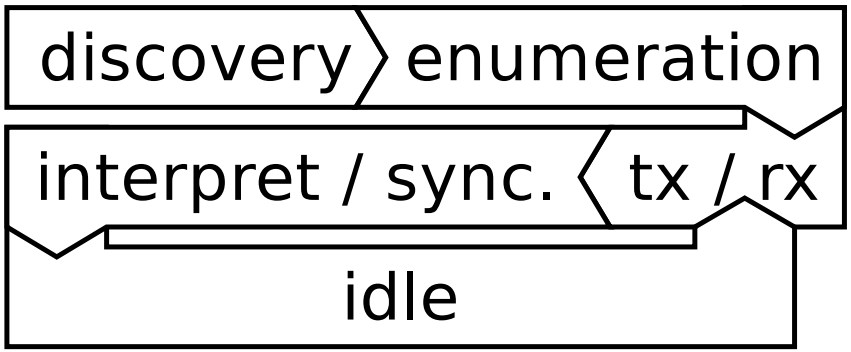


Figure 27: protocol process

Firmware sequence diagram

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

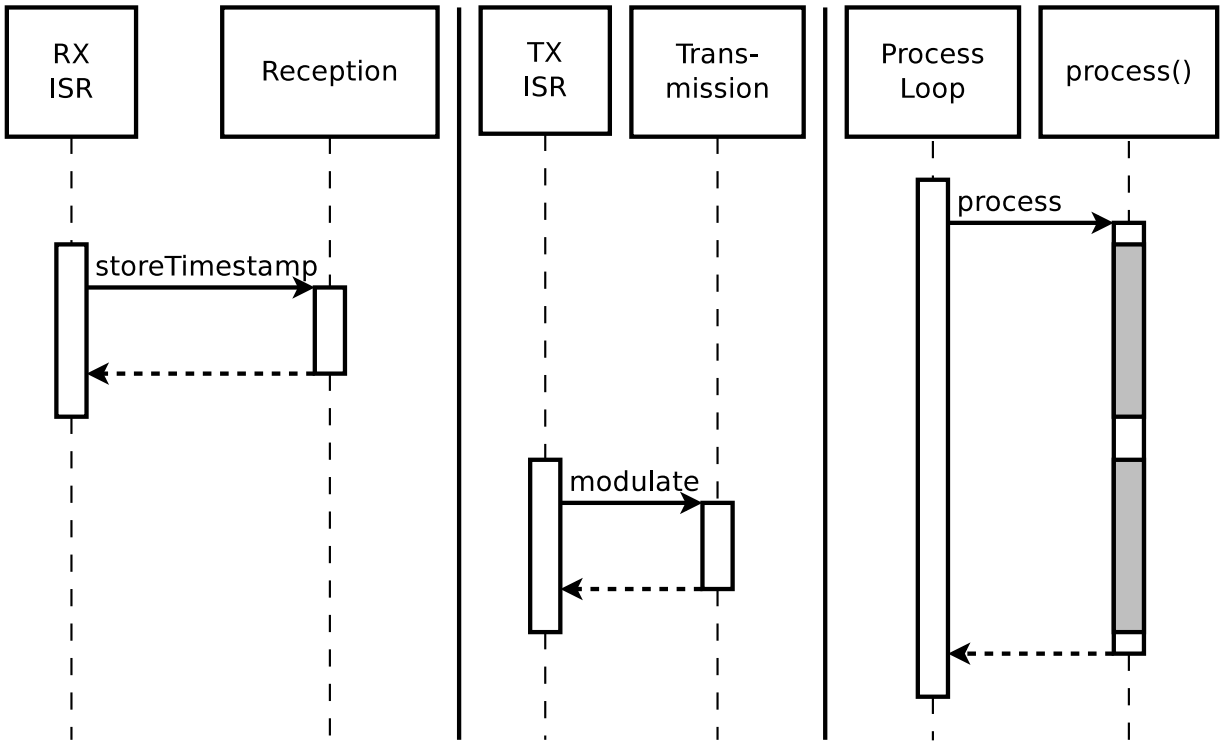


Figure 28: firmware sequence diagram, gray states are blocking

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

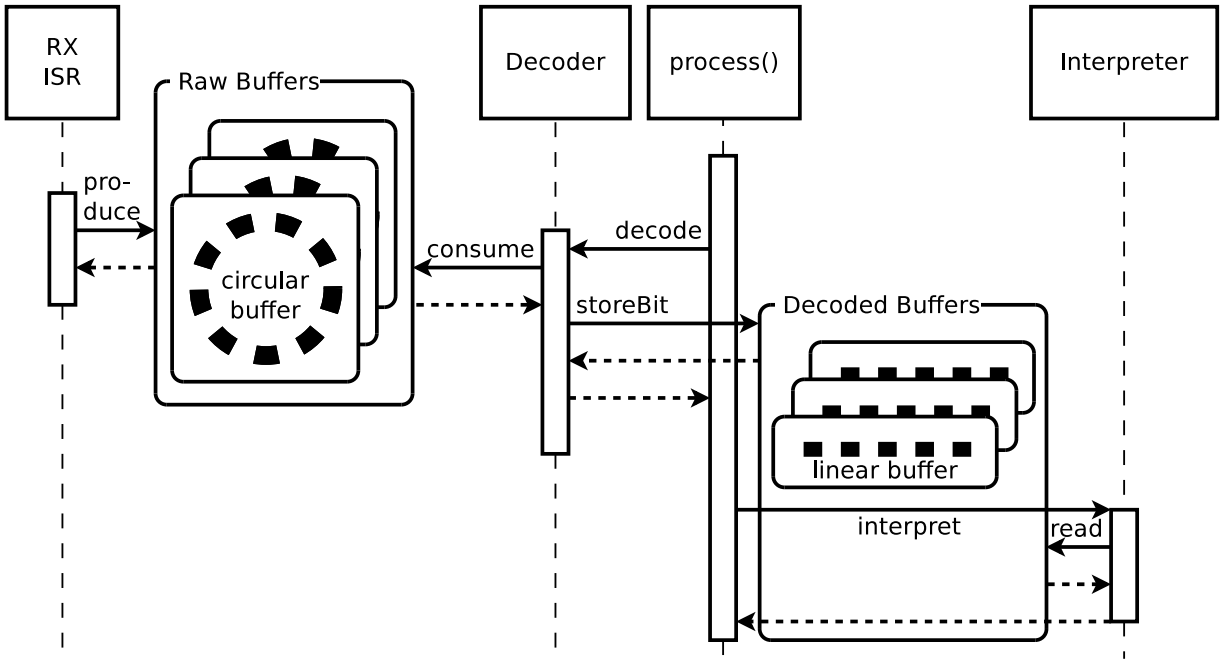


Figure 29: decoding sequence diagram

Integer discretization

- requirement:
  - reference time interval $d_r \triangleq$ Manchester clock delay
  - better approach $d_r \triangleq$ PDU delay

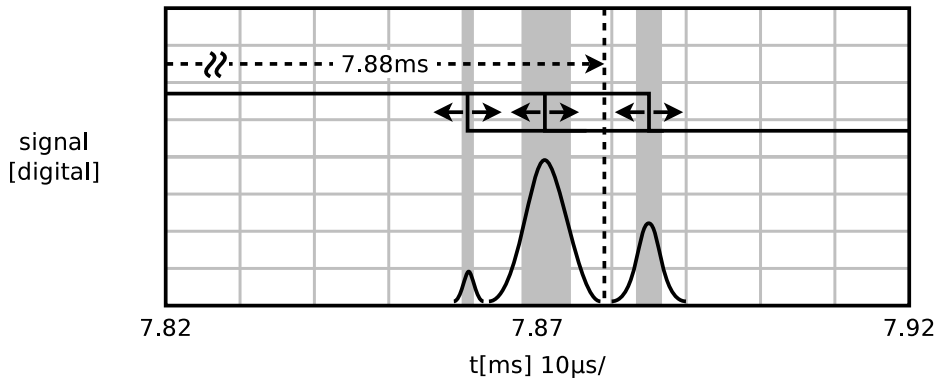


Figure 30: reference interval discretization

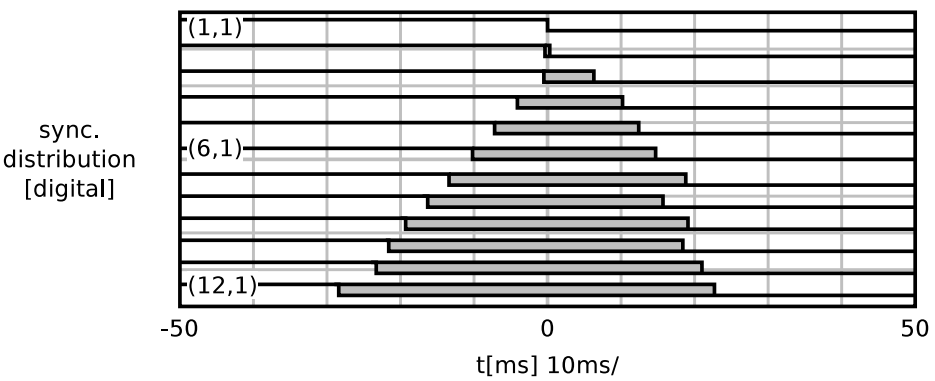


Figure 31: synchronization distribution

Observed ripple

- LED blinking influences *Vcc*
- *Vcc* ripple influences f_{mcu}

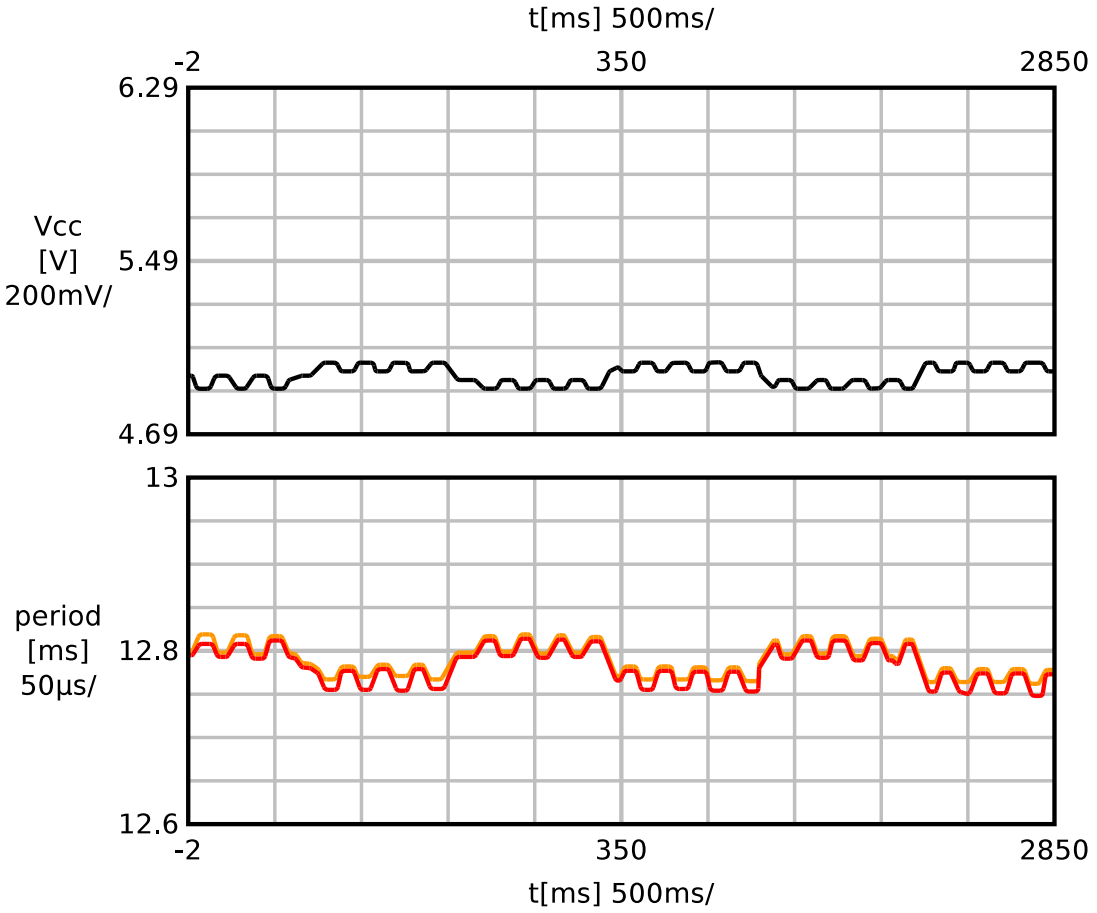


Figure 32: *Vcc* vs. reference periods of particles (1, 1) (beige) and (12, 1) (red)

Synchronization experiment using MCU oscillator calibration

- internal MCU osc. is changed on purpose
  - using minimum f_{mcu} change at each step
- particles automatically adjust themselves accordingly

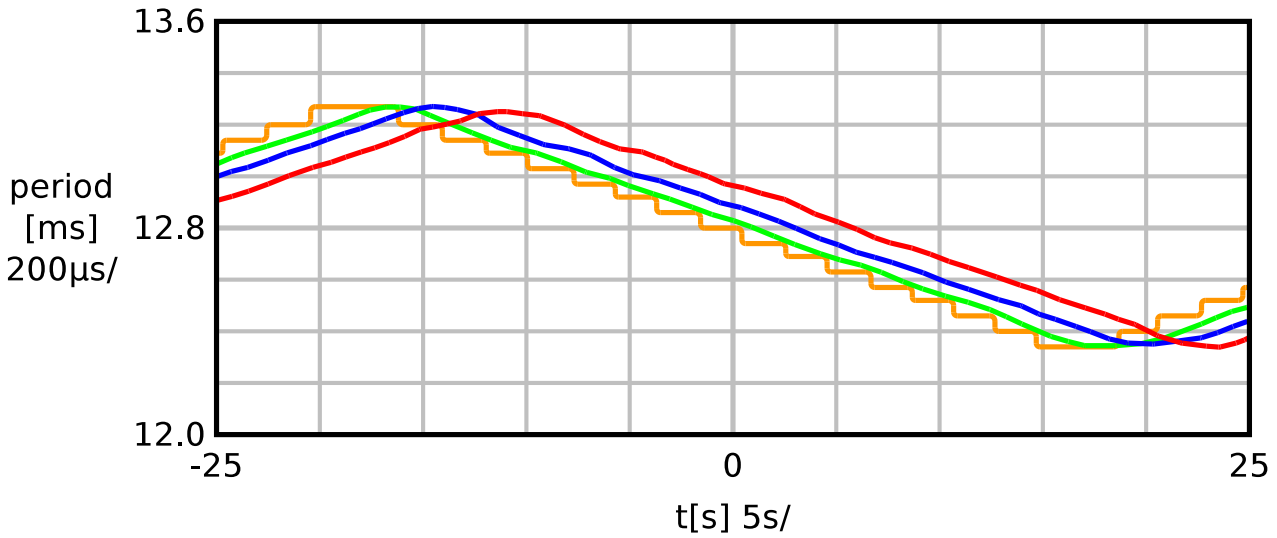


Figure 33: reference periods of particles (1, 1) (beige), (4, 1) (green), (7, 1) (blue) and (12, 1) (red)

Clock skew compensation smoothing - WMA

- weighted average of last and current observation
- reference periods compensation of nodes (1, 1), (4, 1), (7, 1) and (12, 1) as beige, green blue and red

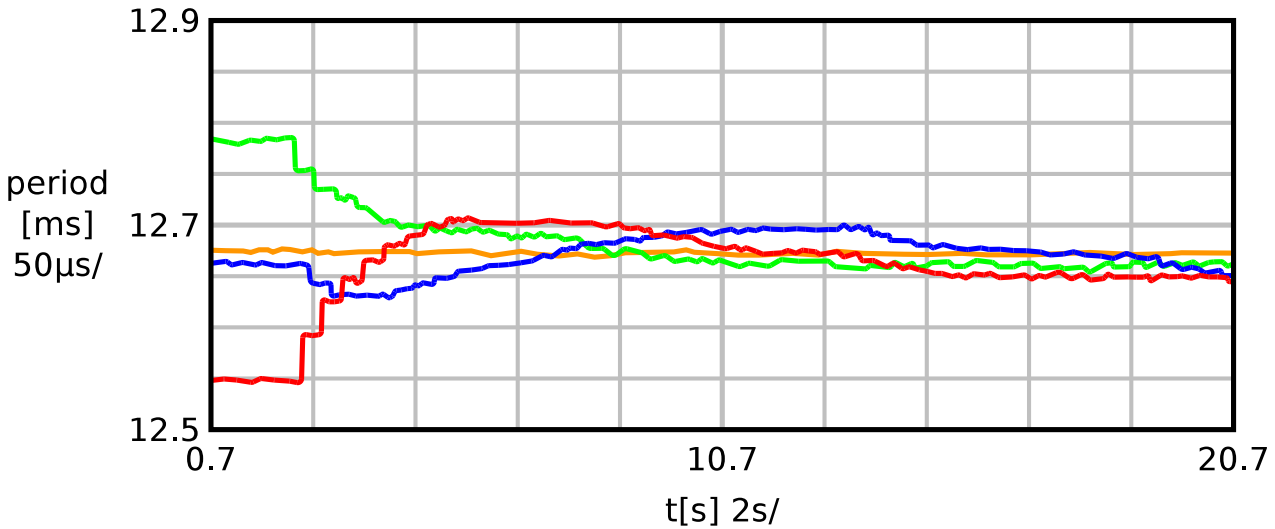


Figure 34: Weighted Moving Average

Clock skew compensation smoothing - SMAV

- average of buffered observations
- reference periods compensation of particles $(1, 1)$, $(4, 1)$, $(7, 1)$ and $(12, 1)$ as beige, green blue and red

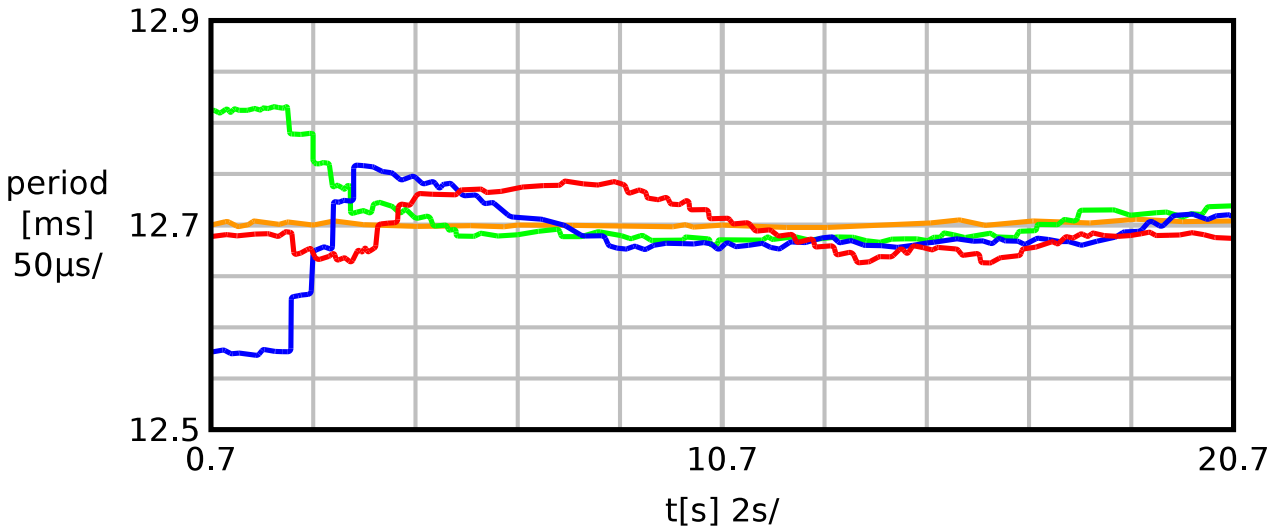


Figure 35: Simple Moving Average

Clock skew compensation smoothing - MLS

- linear regression using buffered observations
- reference periods compensation of particles $(1, 1)$, $(4, 1)$, $(7, 1)$ and $(12, 1)$ as beige, green blue and red

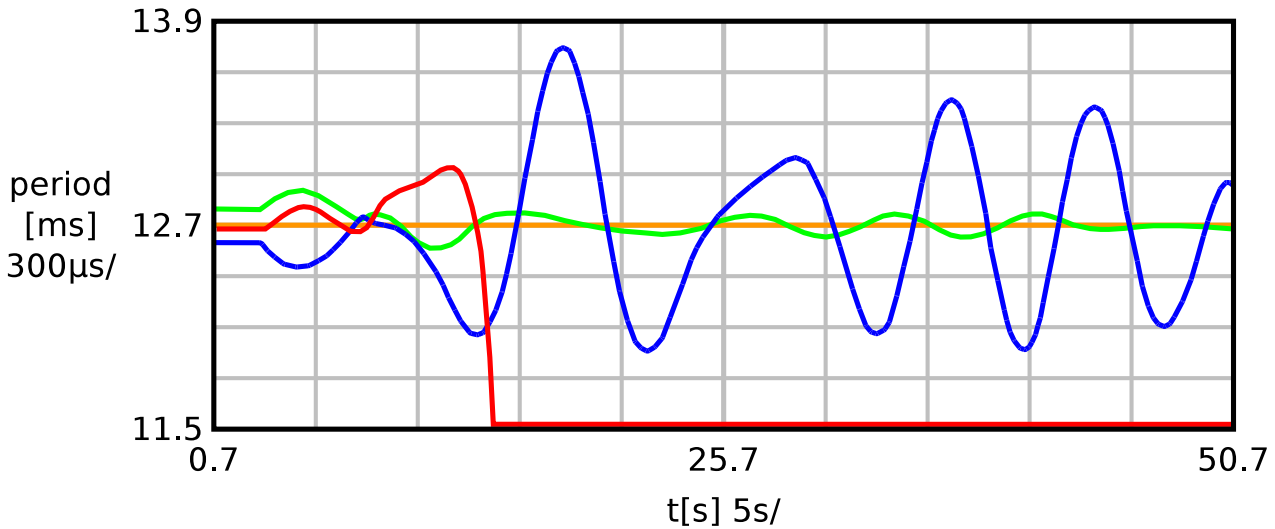


Figure 36: least squares linear regression (Moving Least Squares) overshooting

Clock skew compensation smoothing - Baseline

- only last observation is considered
- reference periods compensation of particles $(1, 1)$, $(4, 1)$, $(7, 1)$ and $(12, 1)$ as beige, green blue and red

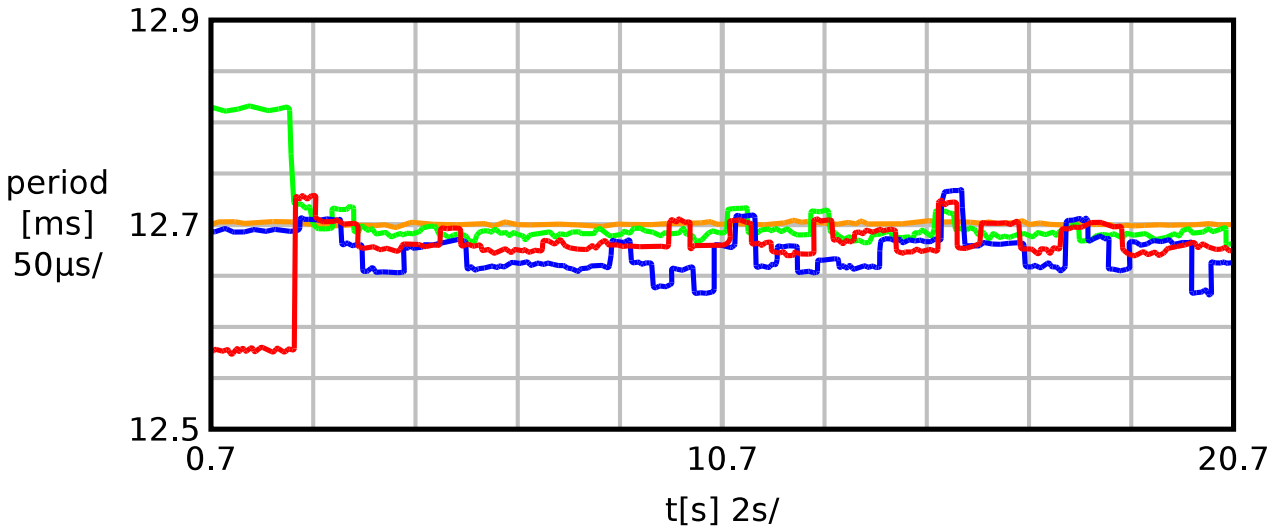


Figure 37: baseline (no smoothing)

Mechanical design

- mechanical design and curvature approximation example

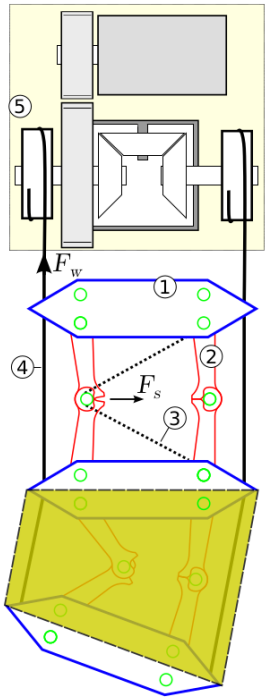


Figure 38: mechanical design of particle chain

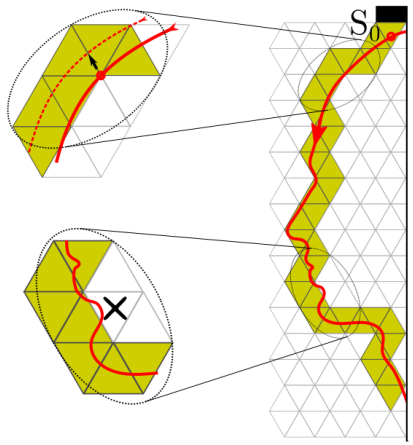


Figure 39: folding example

[DOI 10.1109/IROS.2014.6943112]

Simulation

- simulated network visualization

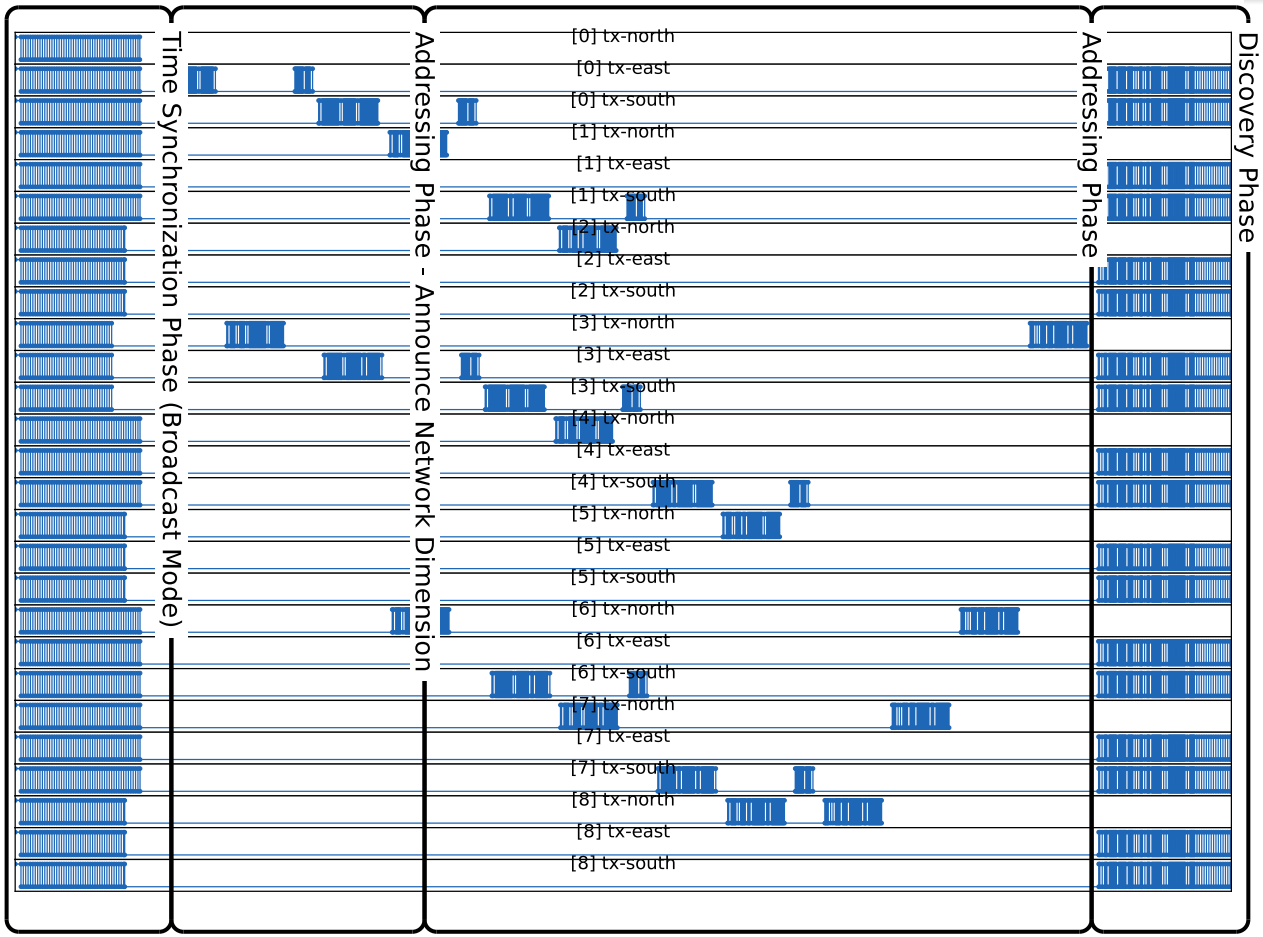


Figure 40: (3×3) network visualization

Supply voltage vs. oscillator frequency

- *Vcc* vs. f_{cpu}
- frequency more robust against *Vcc* ripple within $[2.7,\ 3.9]\,V$

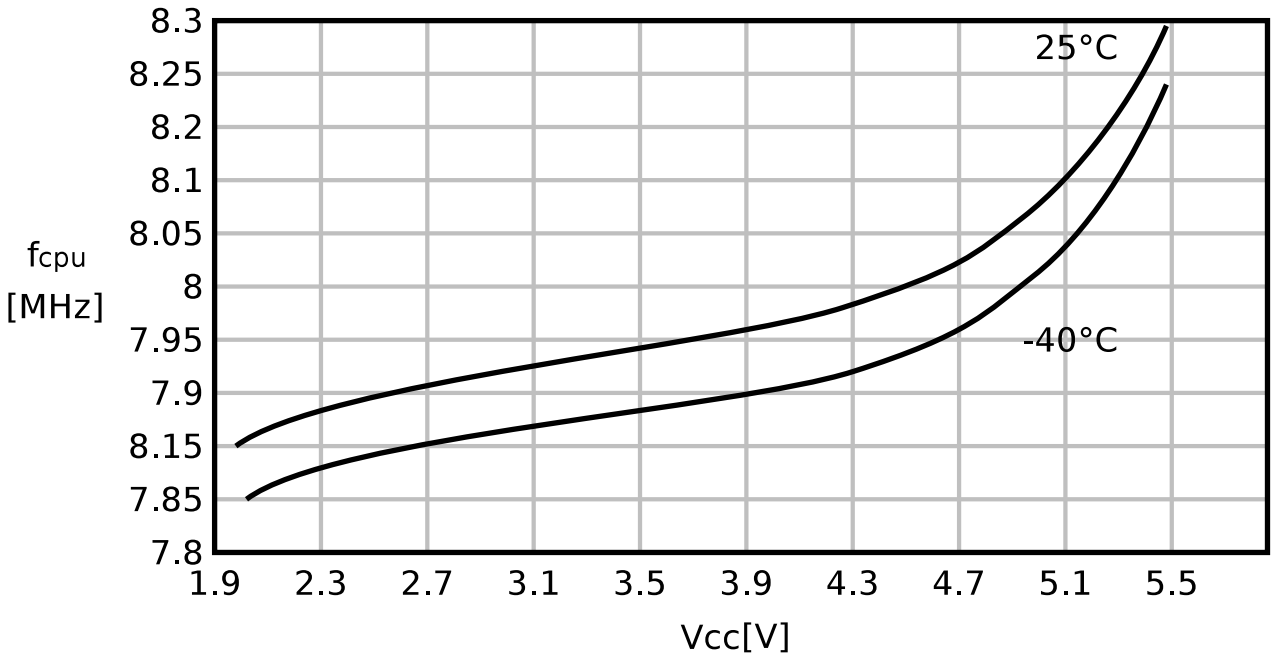


Figure 41: ATtiny1634 MCU clock frequency vs. supply voltage