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

Developmet Board Prototype

Raoul Rubien
rubienr@sbox.tugraz.at

Institute for Technical Informatics
Graz University of Technology

5th August 2016

Underlying work

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

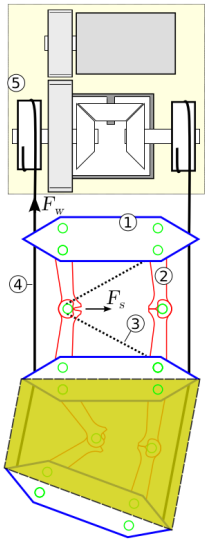

Figure 1: *particle chain*

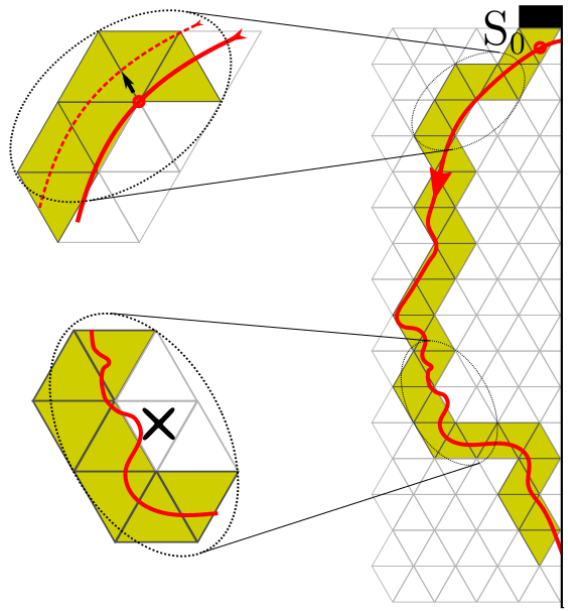

Figure 2:
folding a shape

chain is stretched in natural state
shape Memory Alloy used as actuator (3)
joints unlocked by actuators (2)
force F_s folds chain

Approach & limitation

current particle implements 1-Wire via power supply wires
energy must be buffered before communication starts
power must be switched off/on
automatic chain position detection is costly

Idea

decouple communication from power supply (4)
using a daisy-chain protocol, and
actuator wires (3)

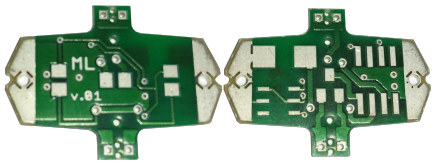

Figure 3: *particle PCB*

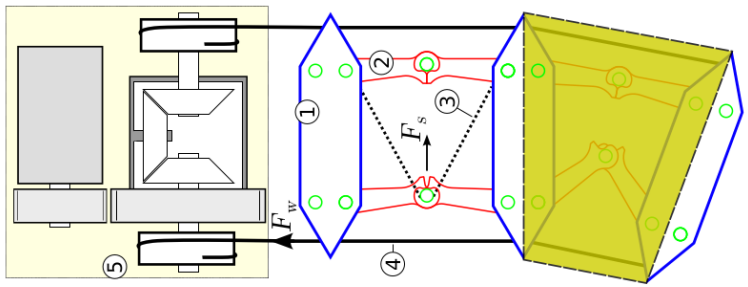

Figure 4: *particle chain*

Project extent

particle board development
easy accessible test points and transmission wires
flexible and fast network assembly
integrate simulation

Project constraints

reasonable low level MCU
minimize number of components on particle PCB
communication:
- exploit SMA wires
- decouple from power supply

small MCU package in final productive particle
single communication entry point to the network

Network approach

linear network

daisy chained participants

Advantages

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

Disadvantages

faulty particle:

- disrupts segment
- no recovery

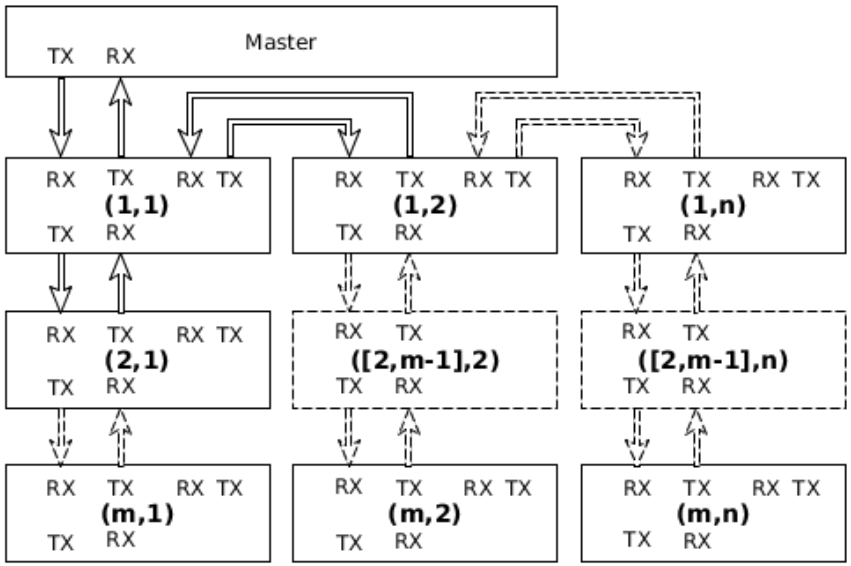

Figure 5:
network topology

The prototype

- not satisfying new requirements
- too small/unhandy for development

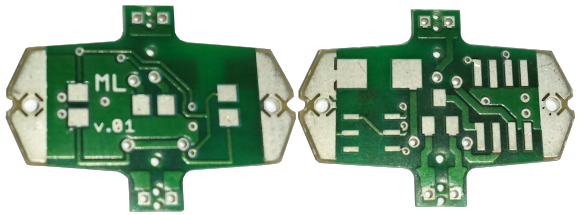

Figure 6: *prototype*

Version 1.0

linear chain of development particles
+ not mounted in chain mechanics
- but still time consuming assembly

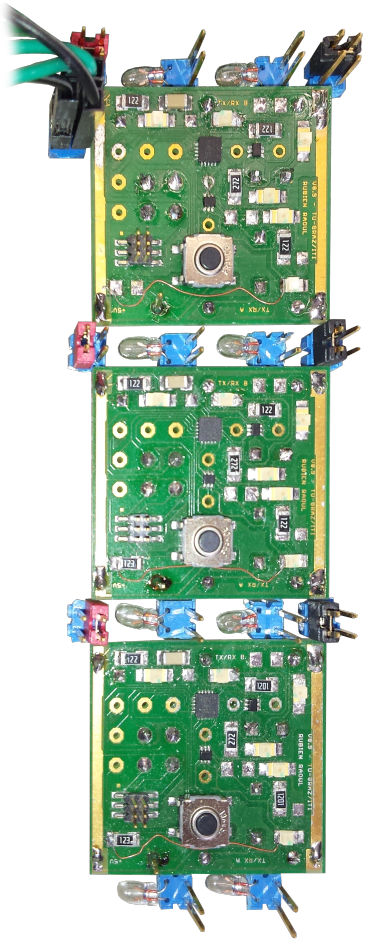

Figure 7: *Version 1.0*

Version 1.1

Advantages

+ repetitive design
+ configurable network shape

Disadvantages

- costly soldering
- expensive connectors
- one faulty particle breaks whole PCB

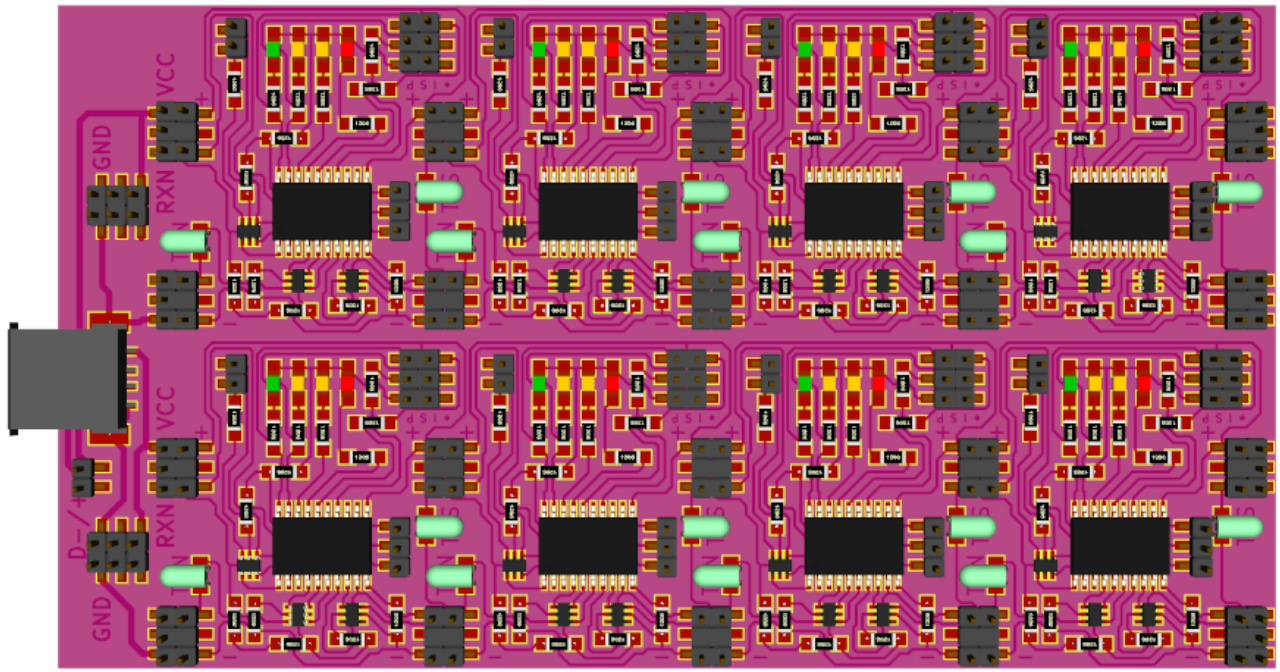

Figure 8: *Version 1.1 - particle array PCB*

Version 1.21

configurable network dimension
easy extensible network
faulty particles can be replaced
cheaper
higher particle density

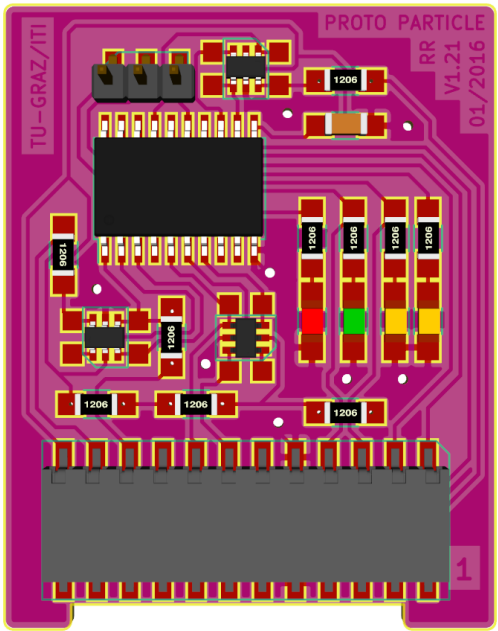

Figure 9: *pluggable particle*

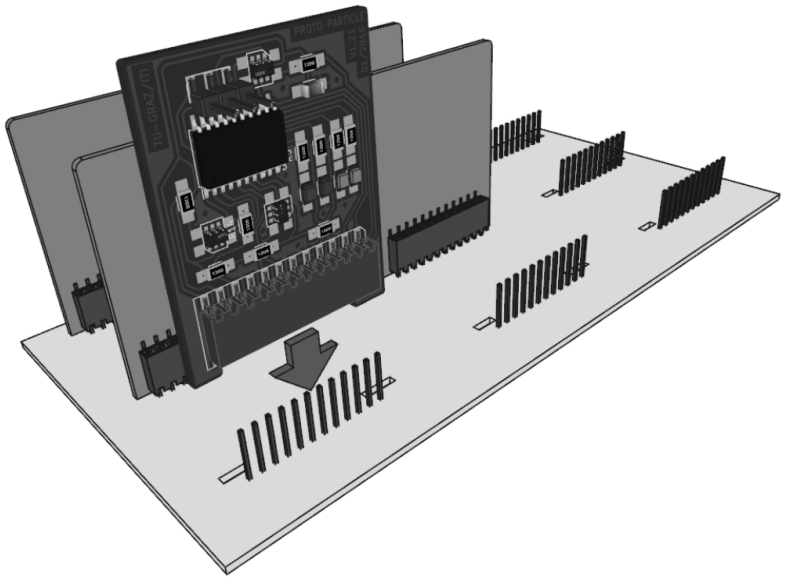

Figure 10: *grid board*

MCU selection

three separate ext. interrupt necessary
self programmable flash: firmware replication
small package for productive particle
optional bigger package for development board

Comparison

	ATTiny20 (V1.0)	**ATTiny1634**
# pin change int.	sufficient	sufficient
self programing flash	no	yes
flash	$2kB$	$16kB$
SRAM	$128B$	$1kB$
small package	$3mm \times 3mm$	$4mm \times 4mm$
alternative package	no	yes (SOIC)

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

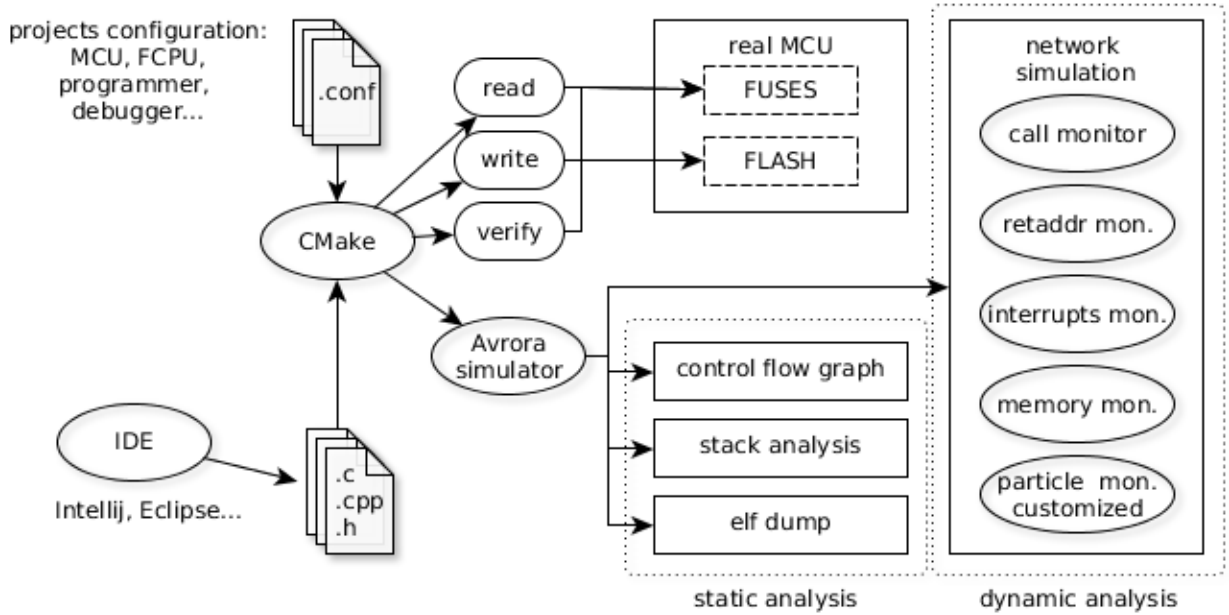

Figure 11: *tool chain overview*

CMake: Simulation and inspection targets

CMake implementation provides various targets

dynamic analysis (monitoring):
- wires
- any SRAM register
- I/O ports
- interrupts
- stack overflow

static analysis:
- stack: maximum size
- control flow graph (code optimization)
- inspecting CPU cycles per instruction (code opt.)

Avrora[1] control flow graph

square: entered from interrupt
double octagon: procedures entry points
hexagonal: blocks end with a return

edges: jumps, branches, fall-throughs
red edges: calls
dotted: indirect calls or jumps

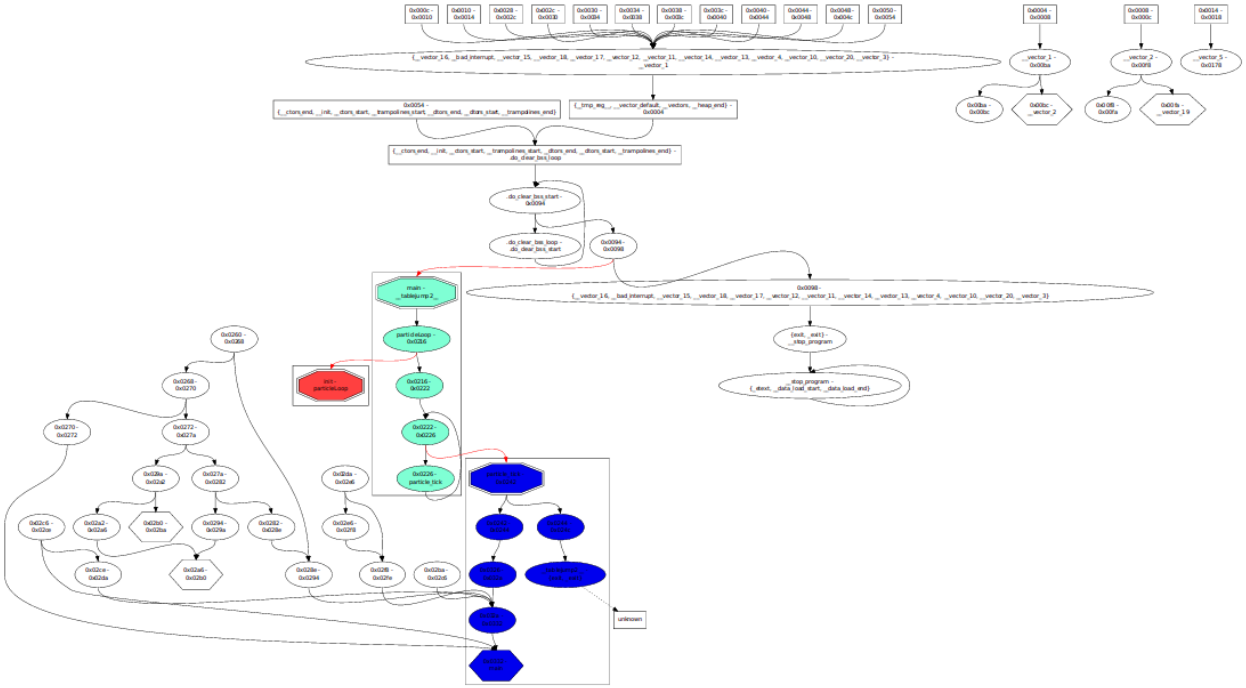

Figure 12: *control flow graph of a short program*

Avrora simulation trace

very informative for proving

a must have for unit testing and debugging

may be used in other visualization or simulation frameworks such as:

- friction simulation
- network visualization

highly customize-able

```
0  0:00:00.00193878320  enable interrupts
1  0:00:00.00198403484  SRAM[SREG.(I | T | H | S | V | N | Z | C)] <- (0b00100001)
1  0:00:00.00198403484  SRAM[5f] <- 21 @ 14e
1  0:00:00.00198403484  disable interrupts
1  0:00:00.00198465904  main:particle_tick: @ 0x0154 <-(RETI)-- #20 0x004C
1  0:00:00.00198465904  enable interrupts
2  0:00:00.00194290985  SRAM[64] <- 0 @ 2a6
2  0:00:00.00194315919  SRAM[globalState.type] <- NODE_TYPE_TAIL (3)
2  0:00:00.00194315919  SRAM[63] <- 3 @ 2aa
2  0:00:00.00194390981  main: @ 0x02AE <-(RET )-- particle_tick
2  0:00:00.00194415915  main: @ 0x0222 --(CALL)-> particle_tick
2  0:00:00.00194415915  SRAM[45d] <- 13 @ 222
2  0:00:00.00194415915  SRAM[45c] <- 1 @ 222
2  0:00:00.00194503459  SRAM[particle_tick.loopCount] <- (-101)
2  0:00:00.00194503459  SRAM[60] <- 9b @ 22e
0  0:00:00.00194178328  SRAM[64] <- 0 @ 2a6
0  0:00:00.00194203342  SRAM[globalState.type] <- NODE_TYPE_HEAD (1)
```

Figure 13: *simulation trace*

Avrora 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
  GCIR.(INT1 | INT0 | INT2 | - | - | - | IVSEL | IVCE)  2/2
  SREG.(I | T | H | S | V | N | Z | C)  115/1
```

```
=={ 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
```

Figure 14: *Avrora profiling monitors example*

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;
```

Figure 15: *cycles per instruction inspection*

Future work

daisy chain communication protocol
- Phy., Data and Network Layer (OSI)

addressing
- self enumeration

time synchronization
- task scheduling (TDM of comm. and tasks)

coding
- exploit Manchester coding for network synchronization
- runtime calibration of internal RC-oscillator firmware

replication
- customize boot loader

This project was supported by the Institute for Technical Informatics of Graz University of Technology and Matteo Lasagni who has always been sincere and helpful and assisted this project.


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