Challenges for an FPGA hardware implementation of recurrence relations for real-world problems

Sergey VERLAN

Université Paris Est Créteil

Motivation

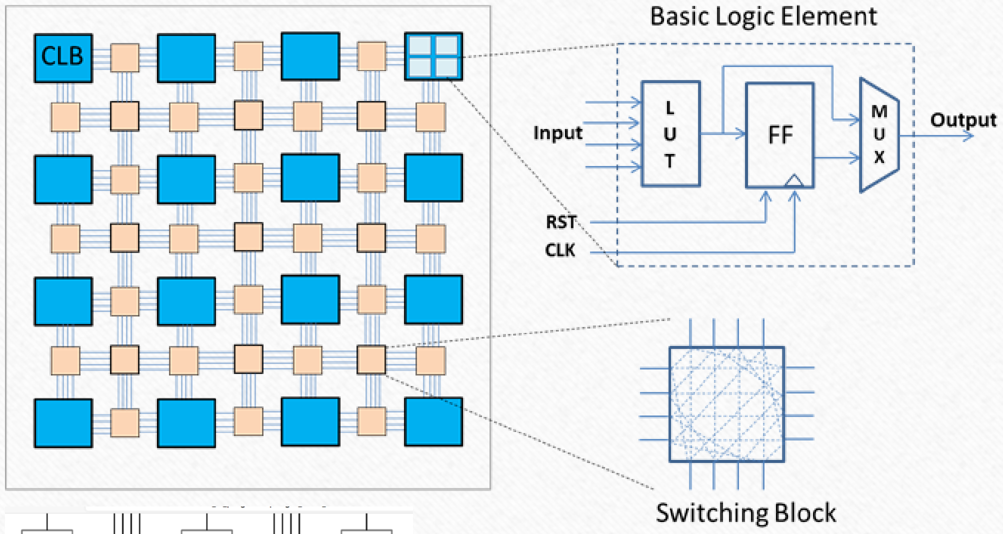
- Desire to design a uc model for problem solving **competitive** with current-day programming and computer hardware.
- Common point for uc: the models are synchronous, highly distributed and parallel.
- But this is just the current silicon architecture!
- However, the lack of tools (and the long tradition) pushes the designers to use microprocessor architectures.
- Obvious choice – custom silicon architecture using FPGA.
- We observed a lack of theoretical methods in the hardware design area.

Why FPGA?

- A technology used to prototype hardware circuits.
- Same device can be programmed to realize different circuits.
- FPGAs are responsible for a major shift in the way digital circuits are designed.

Field-Programmable Gate Arrays (FPGA)

- A technology used to prototype hardware circuits.
- A matrix of Configurable logic blocks (CLBs) with configurable interconnections.
- Each CLB contains several slices, each of them containing several logic cells



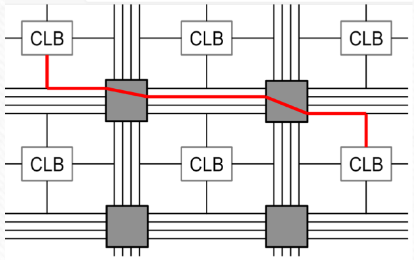
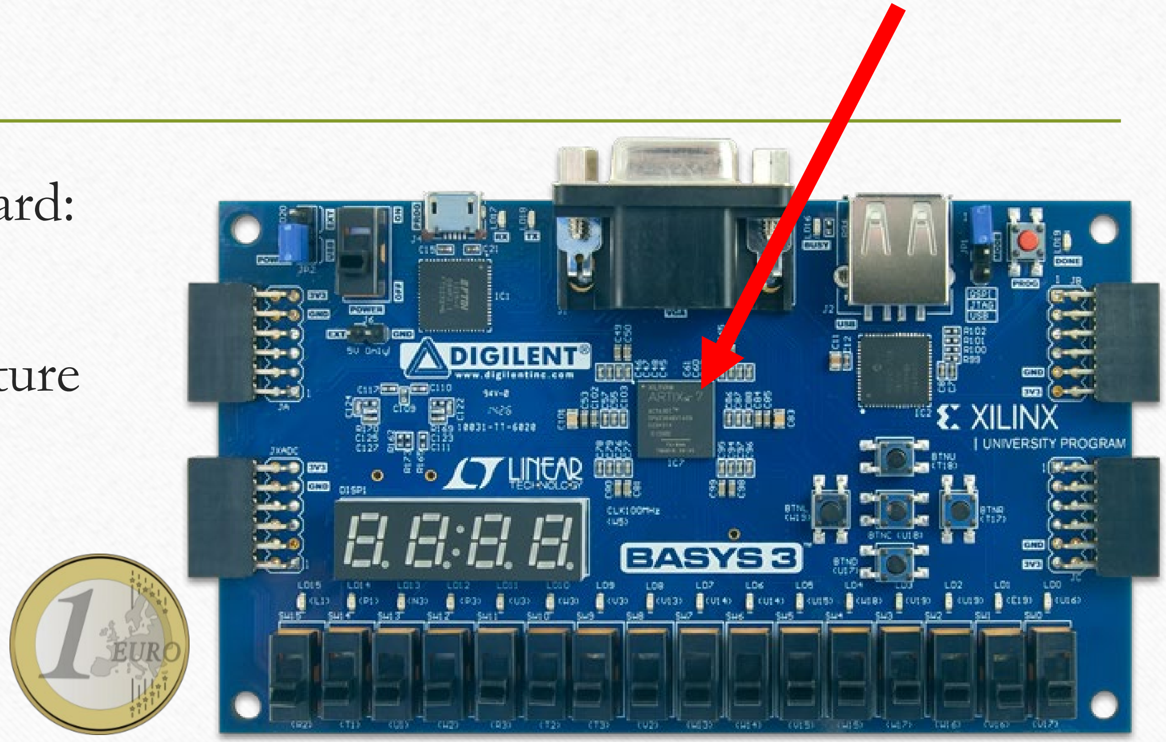


Image from http://evergreen.loyola.edu/dhhoe/www/HoeResearchFPGA.htm

Used FPGA

- We used an entry level FPGA board: Digilent Basys 3
- Based on Xilinx Artrix 7 architecture
- 33,280 logic cells in 5200 slices
- 90 DSP slices

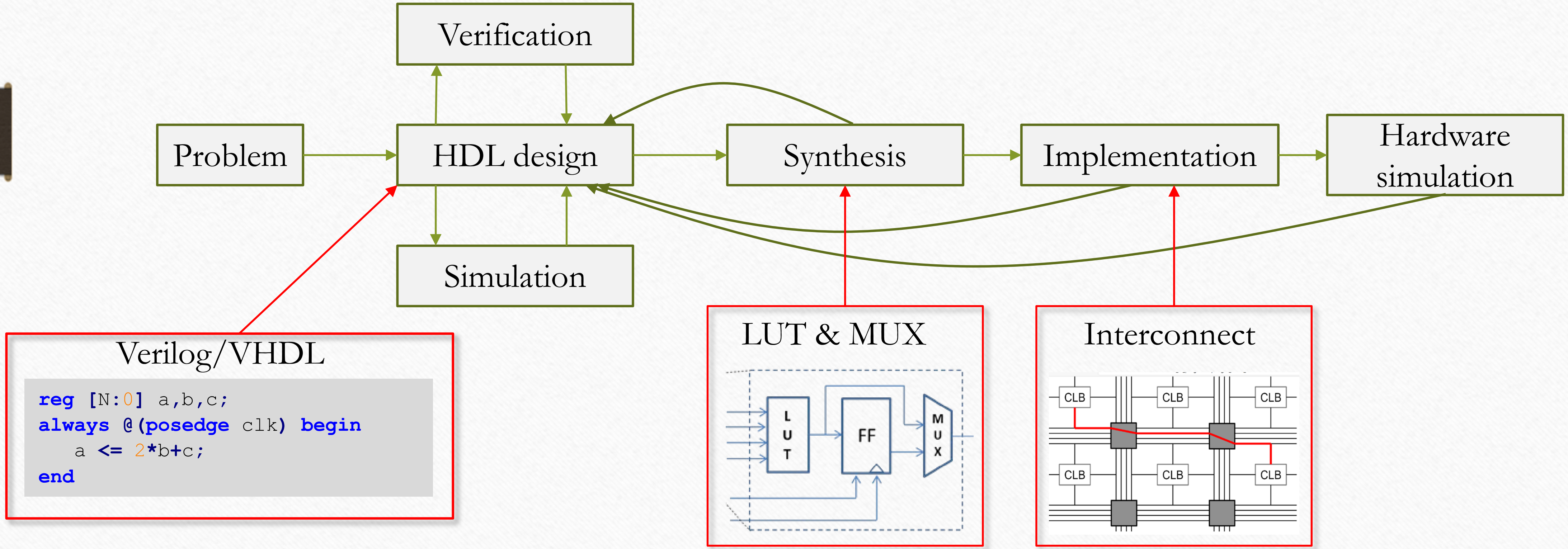


Some tests done on Xilinx VC707 board based on Virtex 7 architecture
485K cells & 2800 DSP slices

Circuit design using FPGA

- The circuits for FPGA are **designed** using a Hardware Description Language (HDL).
- The most common are VHDL and Verilog.
- Important: despite of similitudes circuit design is very different from software programming.
- Basically, it corresponds to the paradigm of data-flow programming.
- There are several levels of HDL design: transistor/gate level, RTL, behavioral.

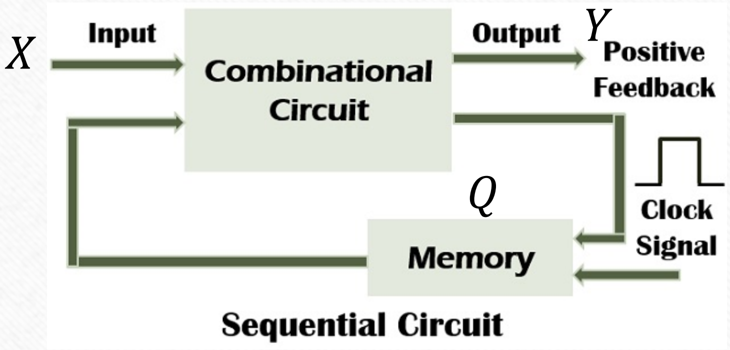
Development cycle



Remarks about hardware design

- Hardware design is a tedious task often unique for each project.
- Except for ready-to-use components, the designers operate at a very low level of abstraction.
- One of our main goals is to design a programming/specification language and a parallel hardware architecture allowing people to easily program different algorithms and obtain efficient FPGA designs for them.
- We start with traditional hardware design models and then discuss the extension.

Sequential and combinational circuits



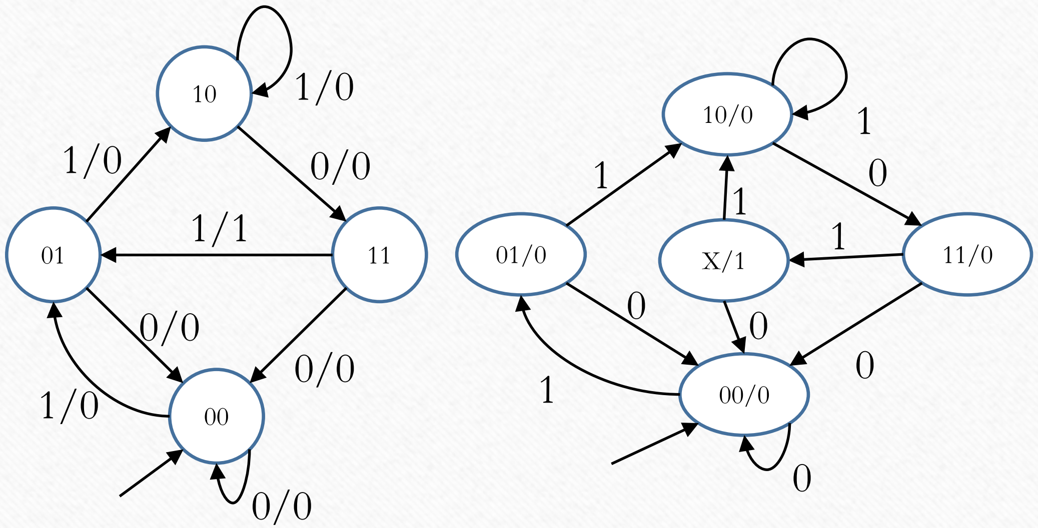
$$Y(t) = F(X(t))$$

$$Q(t+1) = F\big(Q(t), X(t)\big)$$
$$Y(t) = G\big(Q(t), X(t)\big)$$

Images from https://techdifferences.com/difference-between-combinational-and-sequential-circuit.html

Mealy/Moore machine

- Natural representation of sequential circuits.
- A sequential circuit or a Mealy/Moore machine can be directly translated to HDL (especially at the behavioral level).
- In fact, the HDL traditional synchronous design makes an explicit use of a Mealy machine.



1101 sequence detector

Boolean networks & reaction systems

- A B.N. is a particular kind of sequential dynamical system, used to describe biological genetic regulatory networks.
- E.g. $\mathrm{x}(t+1) = x(t) \wedge z(t) \vee y(t)$
  $y(t+1) = y(t) \wedge \neg x(t)$
- The basic variant follows:
  $$Q(t+1) = F\big(Q(t), X(t)\big)$$
  $$Y(t) = G\big(Q(t), X(t)\big)$$
- Which is exactly sequential digital circuits update scheme.

- A R.S. is a different unconventional computing view on B.N.
- It uses a chemical reaction style:
- $x + z \rightarrow x$
  $y + \neg x \rightarrow x + y$
- With the meaning the lhs is a reactant which is consumed, producing an output (the rhs) at the next time step

Using hardware to simulate B.N.: 20K variables and 500K operators on an entry-sized board, with 10^5 speedup.

Example: 2-bit binary counter

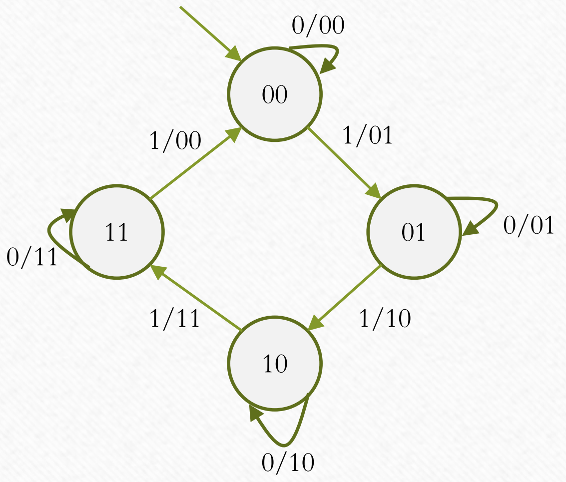
Reaction system:

$x_1 + \neg s \to x_1$
$x_2 + \neg s \to x_2$
$x_2 + \neg x_1 \to x_2$
$s + \neg x_1 \to x_1$
$s + x_1 + \neg x_2 \to x_2$

Boolean (sequential) circuit:

$x_1(t+1) = x_1(t)\ xor\ s(t)$
$x_2(t+1) = \big(s(t) \& x_1(t)\big)\ xor\ x_2(t)$

Mealy machine
input: s, output: $x_2 x_1$



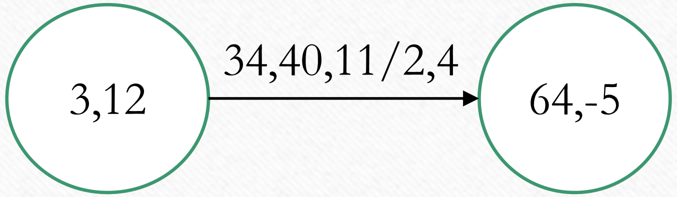
Next step: Integer Mealy/Moore machine

- Use vectors instead of bits for input/output/state:

$$Q(t+1) = F\big(Q(t), X(t)\big)$$
$$Y(t) = G(Q(t), X(t))$$

where X,Y and Q are vectors of vectors of bits (= vectors of numbers)

- Mealy machine:



- Can be directly implemented in hardware.
- F & G ?

The model

Reaction-like	Equational (BN-like)	Automata-like
?	?	Integer Mealy machines

Numerical P systems

- The chemical-style counterpart of equations defined by integer Mealy machines.
- Input/output real-valued variables.
- A model having a graph structure (in case of a tree structure a Venn diagram is used).
- Each node contains:
  - Real-valued variables (x_i)
  - Programs (rules) of form:

$$\mathrm{Pr} = F(x_1, \dots, x_k) \rightarrow c_1|v_1 + \cdots + c_n|v_n$$

Production function

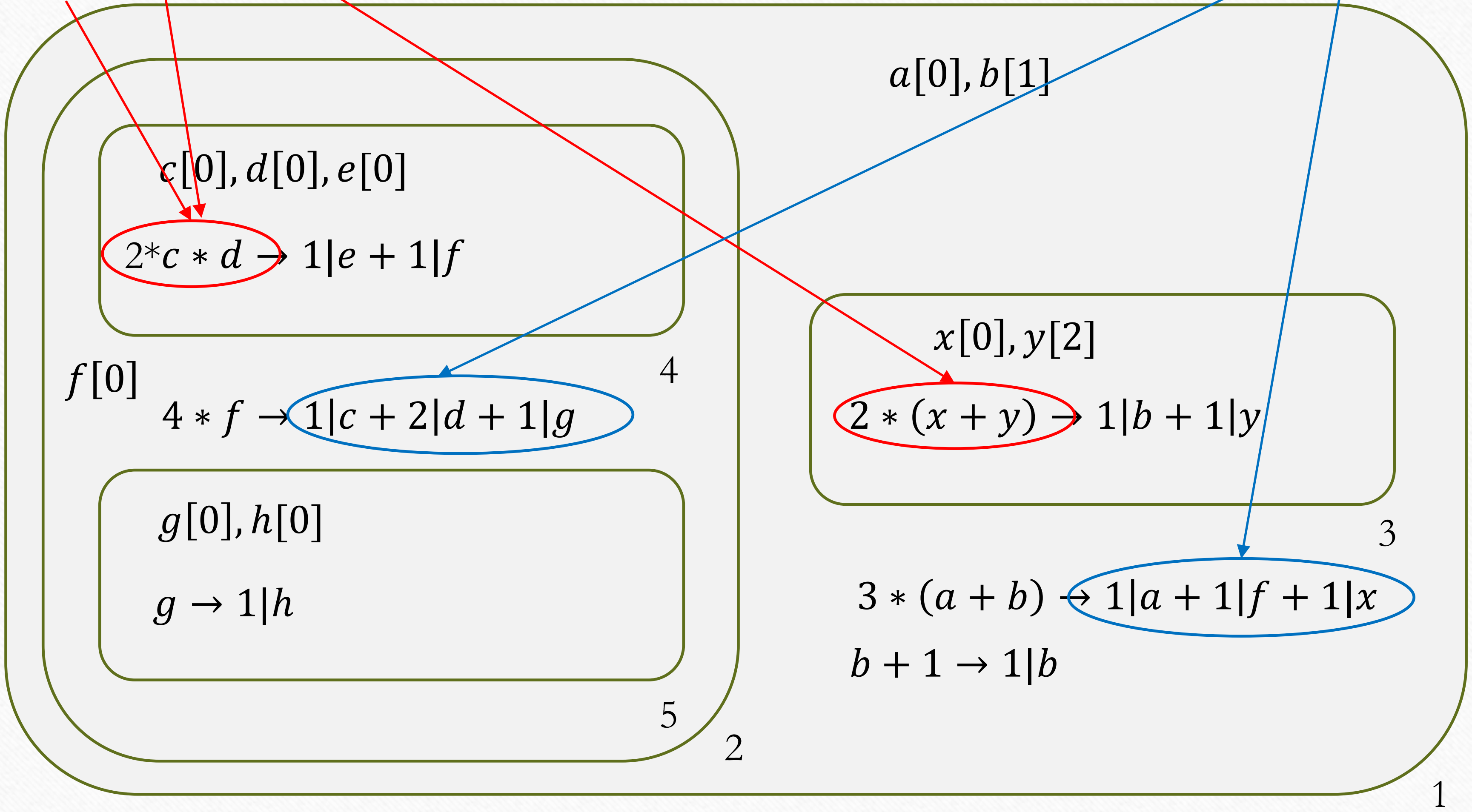
Repartition protocol

Example

F can use only same node variables

F is generally a polynomial
with integer coefficients

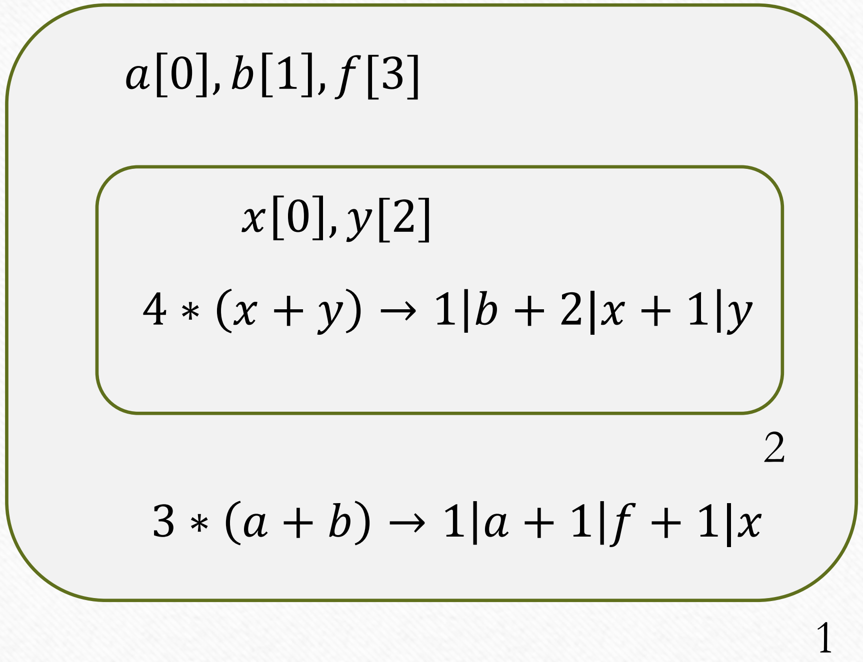
The repartition protocol has natural coefficients
Can use only local or neighbor variables



Computation

- It is done in three steps.
  - The value of F is computed:
    - F_1=3∗(0+1)=3
    - F_2=4∗(0+2)=8
  - The value of used variables (a, b, x, y) becomes 0.
  - Then the repartition protocol specifies which quantity of the F value goes to each variable. For example, for F_2:
  - $F_2 * \frac{1}{1+2+1} = \frac{F_2}{4} = 2$ goes to b
  - $F_2 * \frac{2}{4} = 4$ goes to x
  - $F_2 * \frac{1}{4} = 2$ goes to y
- For F_1: a, f, x each get value 1
- Finally

a	b	f	x	y
1	2	3+1	1+4	2



The model

Reaction-like	Equational (BN-like)	Automata-like
Numerical P systems	?	Integer Mealy machines

Numerical P systems as discrete time series

- $a(t+1) = a(t) + b(t)$
- $b(t+1) = x(t) + y(t)$
- $f(t+1) = f(t) + a(t) + b(t)$
- $x(t+1) = 2 * \big(x(t) + y(t)\big) + a(t) + b(t)$
- $y(t+1) = x(t) + y(t)$

$a[0], b[1], f[3]$

$x[0], y[2]$

$4 * (x + y) \rightarrow 1|b + 2|x + 1|y$

2

$3 * (a + b) \rightarrow 1|a + 1|f + 1|x$

1

DTS= recurrences = difference equations

Discrete time series
Systems of recurrences

- A recurrence relation (of order k):

$$u(t) = \varphi\big(u(t-1), \dots, u(t-k)\big), t \geq k, t \in \mathbb{N}$$

- System of recurrences: a relation may also depend on other relations.
- Canonical form (of order 1):

$$u_i(t) = \varphi_i\big(u_1(t-1), \dots, u_m(t-1)\big), 1 \leq i \leq m$$

- Recalls a discrete fixed-point:

$$\mathrm{Y}(t+1) = f(Y(t))$$

Relation with difference equations

- $(\Delta f)(t) = f(t+1) - f(t)$
- $(\Delta^2 f)(t) = f(t+2) - 2f(t+1) + f(t)$
  etc
- This relation can be inverted, hence they are equivalent (satisfied by same sequences). E.g.
- $3\Delta^2 f + 2\Delta f + 7f = 0$ is equivalent with
- $3f(t+2) = 4f(t+1) - 8f(t)$

Differential equations (1)

- Discretization using Euler method yields a recurrence:

$$y'(t) = f\big(t, y(t)\big), y(t_0) = y_0$$

$$y(n+1) = y(n) + hf\big(t(n), y(n)\big), \qquad t(n) = t_0 + nh$$

Differential equations (2)

- Isomorphism of real-valued sequences and power series:

$$i\big((a_n)_n\big) = \sum_{n\in\mathbb{N}} \frac{a_n}{n!} x^n \ \ and\ hence\ \ \frac{d}{dx}\big((a_n)_n\big) = (a_{n+1})_n$$

- E.g.

$$\frac{d^2y}{dx^2} - 5\frac{dy}{dx} + 6y - 4xe^{2x} + 2e^{2x} = 0$$

$$\frac{d^2}{dx^2}(a_n)_n = 5\frac{d}{dx}(a_n)_n - 6(a_n)_n + 4(2^n n) - 2(2^n), \qquad y = i(a_n)_n$$

$$a_{n+2} = 5a_{n+1} - 6a_n + 2^n(4n - 2)$$

Differential equations (3)

- In (linear) homogeneous case it is even simpler:
  - $y''' + y' + y = 0 \Leftrightarrow f(n+3) + f(n+1) + f(n) \; = 0$
  - $xy'' + 2y' - y = 0 \Leftrightarrow nf(n+2) + 2f(n+1) - f(n) = 0$

The model

Reaction-like	Equational (BN-like)	Automata-like
Numerical P systems	Recurrences/ Difference equations	Integer Mealy machines

Real numbers

- NPS and difference equations use real numbers.
- But real numbers do not exist in practice: they are replaced by fixed-point or floating-point approximations, which are indeed integers.
- So, it can be assumed that their implementations use "integers". However, arithmetic operations are a bit different for floating-point encodings.

NPS to Verilog

- Two-steps process:
  - Transform NPS to recurrences.
  - Transform recurrences to Verilog.
- Encode real values using a fixed-point integer encoding.
- The resulting code runs at the clock speed.

$2 * b + c \rightarrow 1 | a$

transforms to

$a(t+1) = 2 * b(t) + c(t)$

that transforms to

```
reg [N:0] a,b,c;
always @(posedge clk) begin
   a <= 2*b+c;
end
```

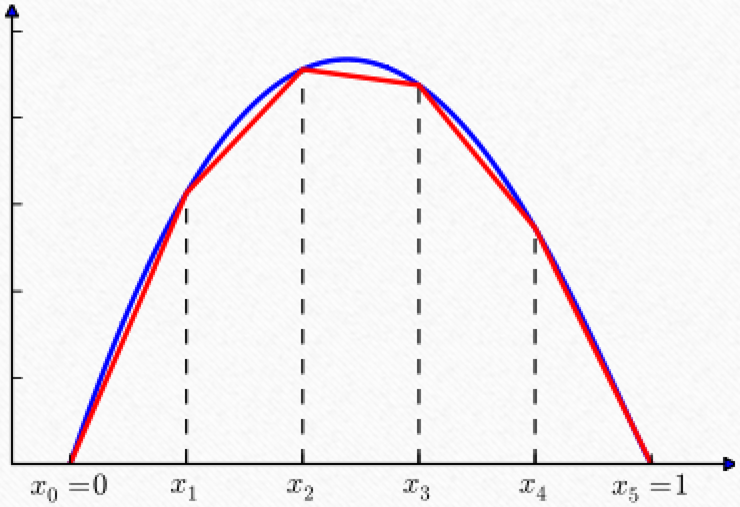
Some problems

- Update functions can be arbitrary.
  However, even * is costly to implement in hardware.
- The computation of update functions is too long.
- Communication paths are too long.

- Use linear functions.
- Bound the depth and split the function or slow down the clock.
- Enforce the locality for updates – as in NPS model.

Non-linearity

- Not everything can be linear.
- One way to tackle the non-linearity is to use piecewise-linear functions to approximate it.



Generalized NPS (GNPS)

- To implement piecewise linearity we consider that the applicability of a rule is controlled by a predicate from the FO theory $(\mathbb{R}, +, >)$. For example:

  $$P(x, y, z; E, F) = E > x \;\&\&\; ((F > y * 2 + 0.3 * z) \;||\; (E + F > x + 2 * y + z))$$

- Then a rule would be just

  $$P(x_1, \ldots, x_k; E_1, \ldots, E_m);\; F(x_1, \ldots, x_k) \rightarrow c_1|v_1, \ldots, c_n|v_n$$

- Where F is linear.

Example

$u[-]$

$x[0]$
$(\mathrm{u} = 1)|\mathrm{x} + 1 \rightarrow x$
$(u = 0)|x \rightarrow x$

$\mathrm{y}[0]$
$(\mathrm{u} = 0)|\mathrm{y} + 1 \rightarrow y$
$(u = 1)|y \rightarrow y$

Global function:

$$x(t+1) = \begin{cases} x(t) + 1, & if\ u = 1 \\ x(t), & if\ u = 0 \\ 0, & otherwise \end{cases}$$

$$\mathrm{y}(t+1) = \begin{cases} y(t) + 1, & if\ u = 0 \\ y(t), & if\ u = 1 \\ 0, & otherwise \end{cases}$$

$$x(0) = 0, \qquad y(0) = 0$$

Conditional recurrences

- The translation of GNPS to equations yields conditional recurrences.
- E.g. $P_1(\dots); F_1(\dots) \to \cdots k|a \dots$ and $P_2(\dots); F_2(\dots) \to \cdots k'|a \dots$
- This yields the following conditional recurrence:
- $a(t+1) = if\ P_1(\dots)\ \&\&\ P_2\ (\dots)\ then\ \ \dots$
  $else\ if\ P_1(\dots)\&\&\,!\,P_2(\dots)\ then\ \ \dots$
  $else\ if\ !\,P_1(\dots)\&\&P_2(\dots)then\ \ \dots$
  $else\ \ \dots$
- Conditional recurrences can be constructed directly by attaching conditions to a recurrence. E.g.

$$a(t+1) = if\ \big(E(t) > a(t)\big)\ then\ 2 * b(t) + c(t)\ else\ a(t)$$

Transforming to Verilog (2)

- $a(t+1) = if\ \big(E(t) > a(t)\big)\ then\ 2 * b(t) + c(t)\ else\ a(t)$ transforms to a conditional expression.

```
reg [N:0] a,b,c;
always @(posedge clk) begin
    a <= E>a ? 2*b+c : a;
end
```

- Any predicate as previously discussed transforms directly to Verilog.

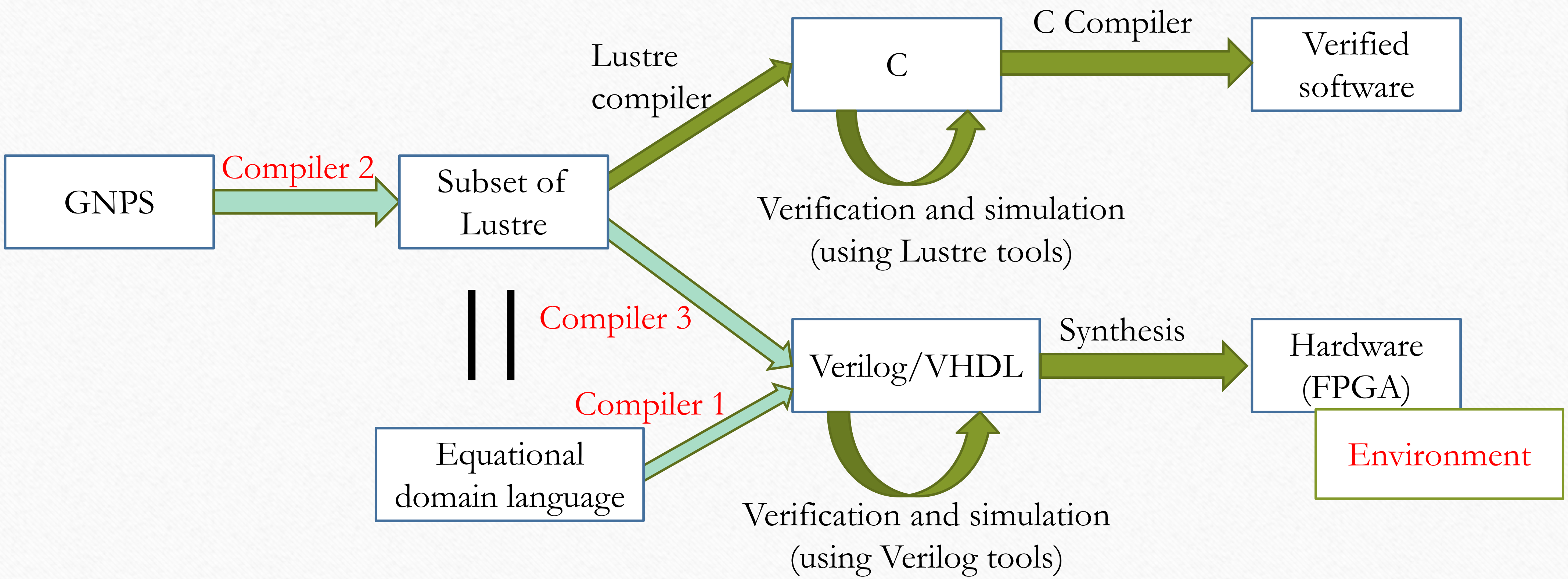
Generalizing: Numerical Networks of Cells

- Cells containing variables (discrete time series) over a monoid $(G, +, 0)$.
- Rules $g \mid f \rightarrow x$, with guard g and update function f for variable x.
- Following natural restrictions can be used:
  - Locality – all variables of g and f are from the same or neighbor cells.
  - g from the FO theory $(G, +, 0, <)$ (suppose that G has a total order relation <).
  - f member of the vector space $(G, +, 0)$ over $\mathbb{N}$ ("linearity").
- Capture (G)NPS, BN, mvBN, PN, D0L and various fuzzy-based computational models.

Synchronous languages

- The equation view of (G)NPS can be directly translated to synchronous programming languages, more specifically to LUSTRE.
- It is a family of languages (SIGNAL, ESTEREL, LUSTRE) aiming to make the same abstraction for programming languages as the synchronous abstraction in digital circuits.
- Allows easy verification for concurrent conditions.
- Used for the design of a critical software (nuclear plants, aeronautics, satellites).

Compiler chains



The goal

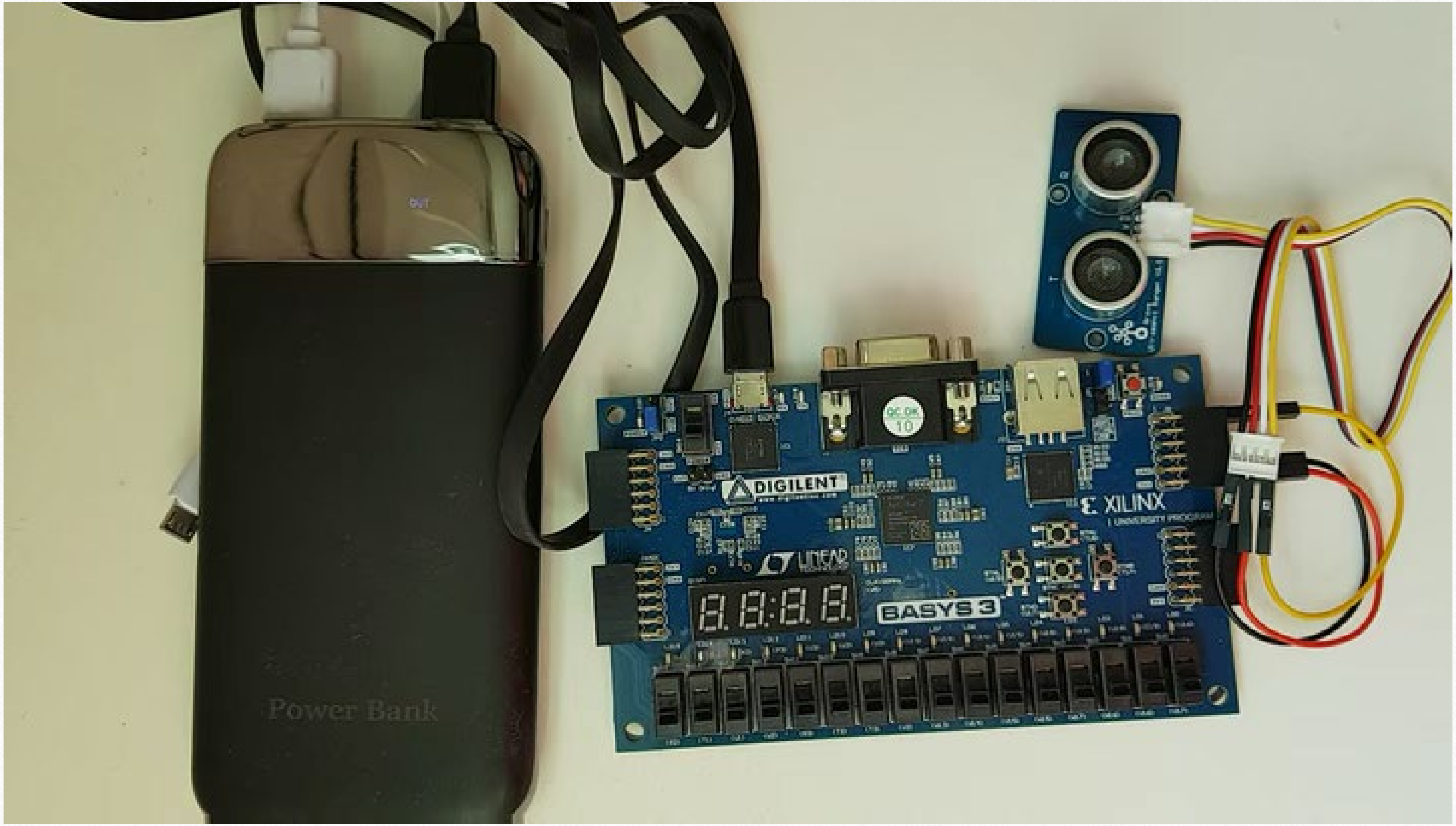
- Create a compiler toolchain that will compile from (G)NPS to circuit description.
- Having standard IO libraries, this allows to code an algorithm in (G)NPS, translate it to an FPGA circuit, run it and get the result/interaction over serial port or IO pins, without any FPGA/electronics knowledge!
- As a side effect can transform the algorithm to a verified C code.
- Another side effect – can compile and run some Lustre programs on FPGA.

Verification

- SystemVerilog has important verification (assertion-based) constructs (SVA).
- It is more-or-less equivalent to a temporal logic specification.
- We provide a verified version of support I/O libraries.
- We are working on a property language specification that would translate to SVA or other languages.

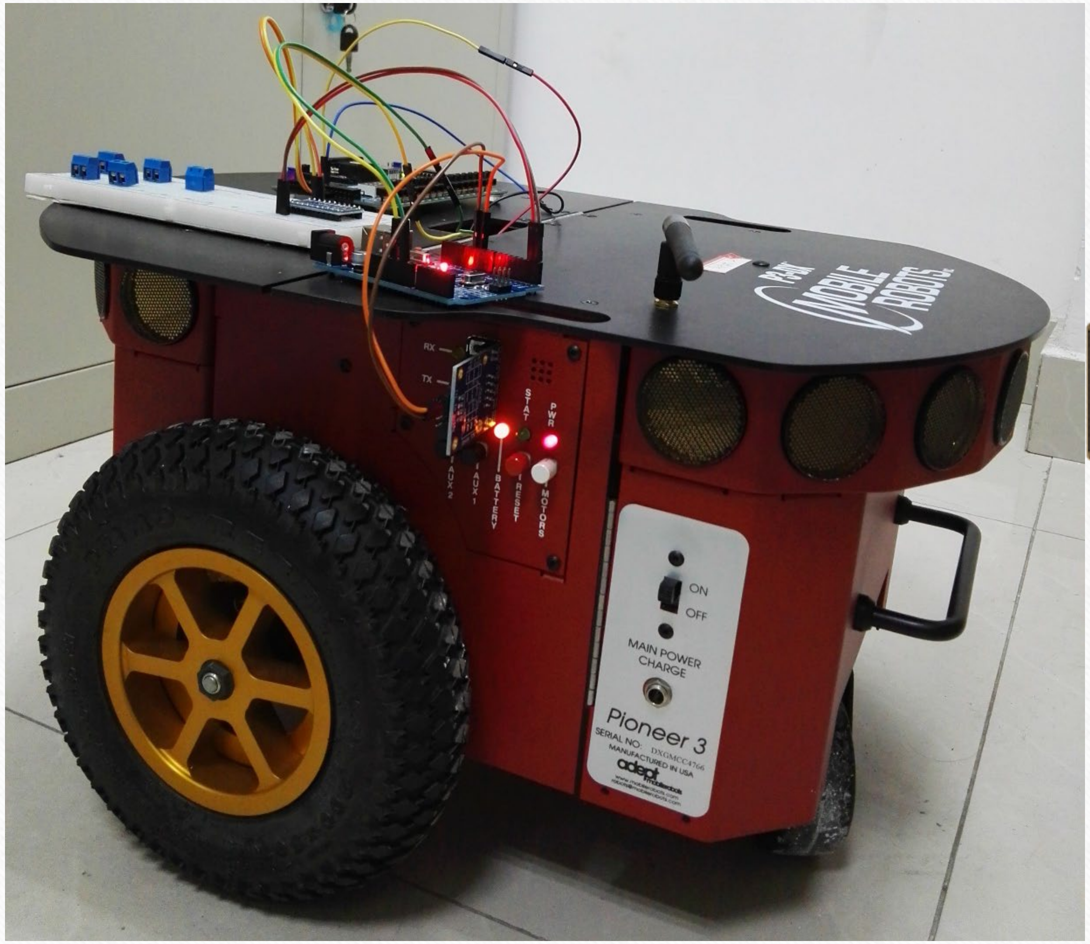
Example of the eq domain language program

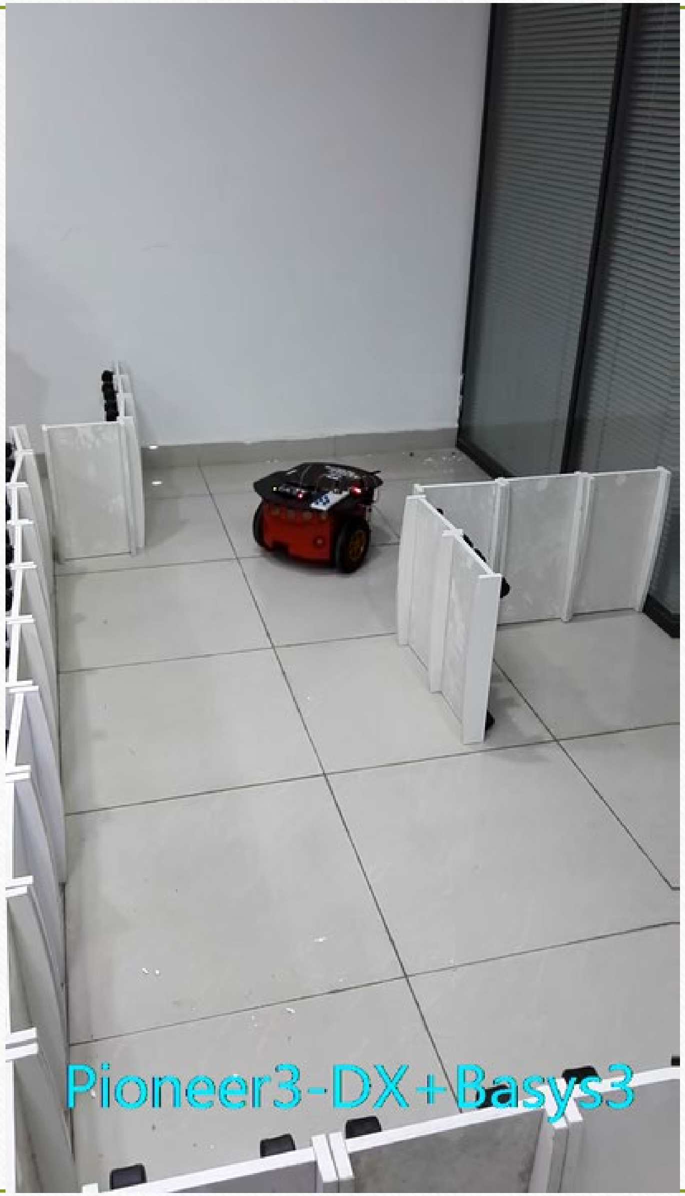
```
…
const{ START:=1; DIST_THRESHOLD := 30; …}
use ultrasonic as us { dataPin := V21; }
input { sw := pinIn(V17); us_distance := us.distance; }
output {res := pinOut(led,count); }
membrane m1{
count:=START;
count = if (sw!=0) then if (us_distance > DIST_THRESHOLD) then count + 1
                                                              else count - 1
                   else count;
}
```


Power Bank
DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3
LINEAR

Test bed

- Proportional–integral–derivative (PID) controller for the motion of Pioneer 3 DX robot.
- We tested several programs allowing to operate sensors and actuators directly by the circuit generated from (G)NPS.
- It allows to be very reactive (~15ns), so the approach is particularly interesting with hi-speed sensors.
- It uses a relatively low number of hardware resources.




Pioneer3-DX+Basys3

Non-linearity (revised)

- In some cases it is simpler to implement several non-linear functions directly.
- Theoretically, this means extending the basic system with a functional symbol in the signature of the theory, e.g. use predicates over FO theory $(\mathbb{R}, +, <, \frac{1}{\sqrt{\cdot}})$.
- The cost is slower or larger design as the implementation of corresponding functions can take several clock cycles or larger surface if using lookup tables.
- There are blocks for widely used functions like sin, cos, sqrt and sqr.

IEEE 754

- We did experiments on using IEEE 754 floating-point number representation in the implementation of the RRT path-planning algorithm on FPGA.
- We have used functional symbols (implemented as blocks) for +,-,* and $\frac{1}{\sqrt{\cdot}}$
- Feasible, but a lot of work with the resource allocation.

Summary

- (Conditional) recurrences translate simply to hardware circuits.
- The use of conditions simplifies a lot practical algorithm design.
- Natural applications: control theory, physics and biology.
- Theoretical challenges for the restriction of the model to map an efficient implementation.
- Functional extensions useful in practice.

Conclusions

- Unconventional (and membrane) computing gives new methods for distributed and parallel algorithm design with applications ranging from optimization to control theory.
- (G)NPS/conditional recurrences are good candidates for parallel algorithm specification to be implemented in hardware.
- They provide equivalences with well-known methods based on difference equations and ensure efficient HDL/hardware translation.
- There are several developed tools that show enormous ($> 10^5$) speedups with respect to software implementations, as well as high IO response time (~10ns).
- We started a big project aiming to develop a specification language and a compiler chain to transform GNPS algorithms into hardware description by non-specialists.

Further work

- Logic:
  - Further restrictions of the model (in order to minimize the depth of functions – on FPGA only some constant depth can be efficiently implemented).
  - Use different (full) signatures according to implementation facility (e.g. bit shifts instead of a constant multiplication).
  - Translation algorithms (from one theory to another).
- Verification:
  - Add verification constructs and facilitate the verification workflow.
  - Provide more verified I/O constructs.
- Algorithmic:
  - Search for conditions on algorithms yielding efficient implementations.
  - Elaborate a clean differential equations => hardware implementation workflow, including for non-linear and partial cases.
- Robotics
  - Implement and compare different PID algorithms.
  - Finish the implementation of path-planning algorithm.
- Finalize the UI for the compilers.