Sleptsov Net Computing

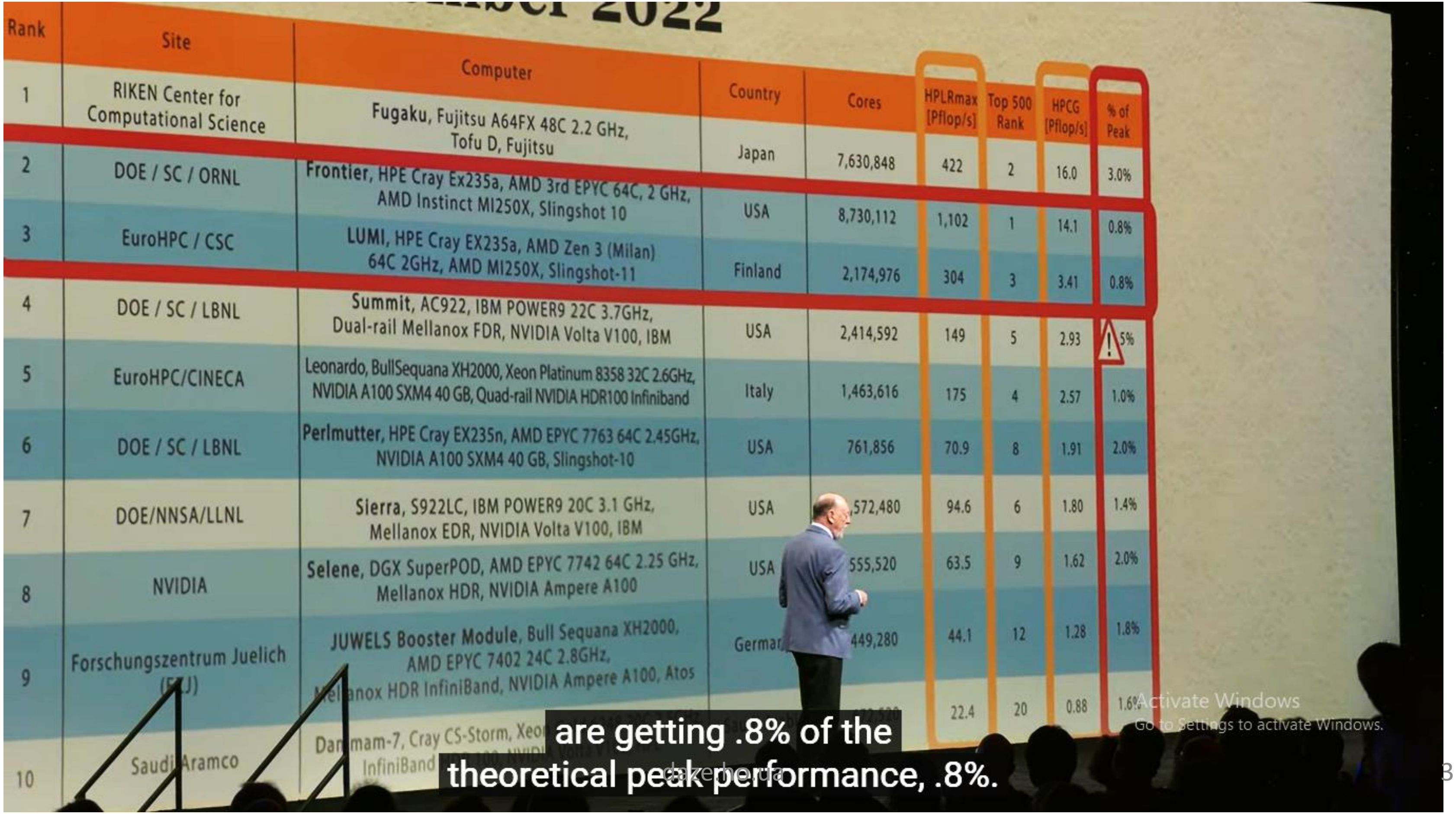
Dmitry Zaitsev

https://dima.zaitsev.github.io

Most powerful computers: http://Top500.org

System	Year	Vendor	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Efficiency (HPCG)
Supercomputer Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11	2021	HPE	8,730,112	1,102.00	1,685.65	0.8 %
Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D	2020	Fujitsu	7,630,848	442.01	537.21	3.0 %

Jack Dongarra Turing Award Talk



Frontier


FRONTIER
OAK RIDGE
National Laboratory
ENERGY
Hewlett Packard
Enterprise
AMD

Architecture and programming tools

- Multicore processor node – shared memory, 32-64 cores – *OpenMP*
- Massively parallel computations – GPU – 6D structure of threads (3D grid, 3D block), ~10000 cores – *NVIDIA CUDA* *or* *OpenCL*
- Distributed nodes, fast (multidimensional) interconnect – *MPI*

Benchmarks of computer performance

- Since 1977, LINPACK – to solve a dense system of linear equations (LU factorization with partial pivoting)
- HPL (High Performance Linpack), 1991
- The High Performance Conjugate Gradients (HPCG) Benchmark, 2015 – Sparse matrix-vector multiplication and solvers

Think! 0.8% of efficiency

- LINPACK, for a half of century, shaped computer architecture to be inefficient
- Processor-memory bottleneck of Von-Neumann architecture
- Sequential programming languages
- Heterogeneous conglomerate of concurrent programming techniques: OpenMP, MPI, OplenCL (CUDA)

Dilemmas

- Why we need separate Processor and Memory?
- Write Programs or Draw Programs?
- Is heterogeneous framework inevitable?
- Correctness Proof or Testing for Concurrent Programs?

Approaching a uniform concept

- Textual programming
- Graphs loaded by textual language
- **Pure graphical programming - nothing save graphs**
- Sleptsov net – fast universal language of concurrent programming
- Massively parallel computations
- Fine granulation
- Computing memory implementation

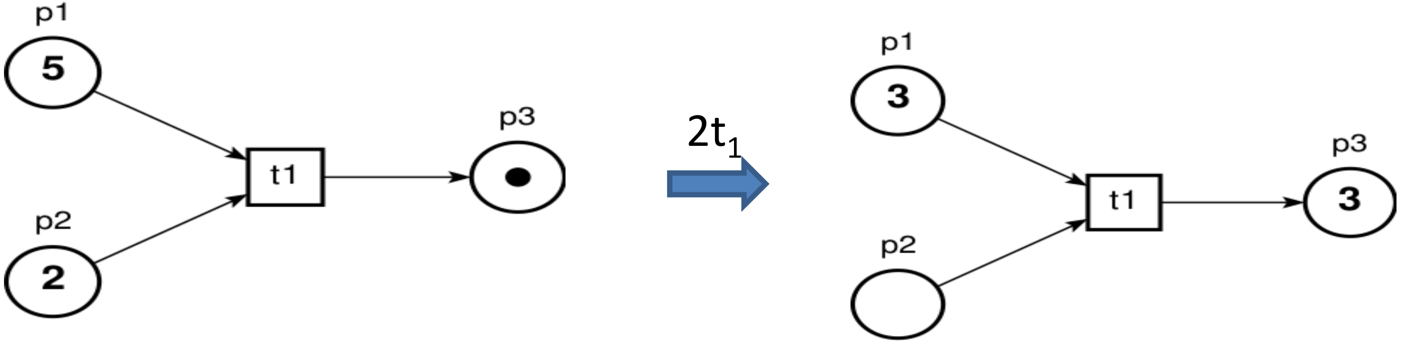
Place-transition nets

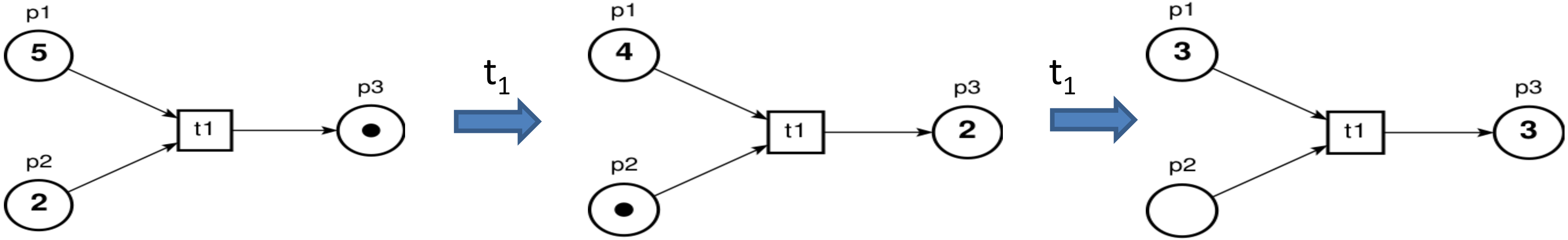
- Bipartite directed graph in CS:
- S.Gill, 1958 – parallel program schemata
- C.Petri, 1962 – added dynamical elements – tokens
- T. Agerwala, M.Hack, 1974-1976 – Turing-complete extensions: inhibitor and priority nets

Transition firing strategy

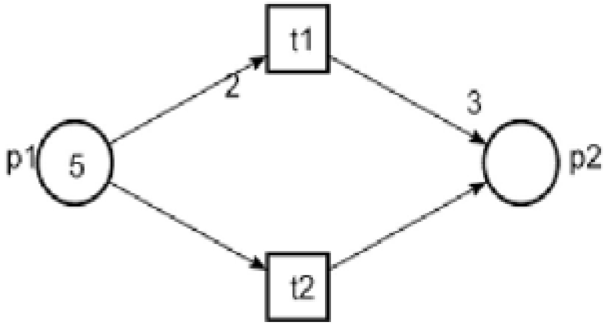
- ***Petri***
  - a single transition at a step
- ***Salwicki***
  - the maximal firing strategy
- ***Sleptsov***
  - the multiple firing strategy

Sleptsov net vs Petri net

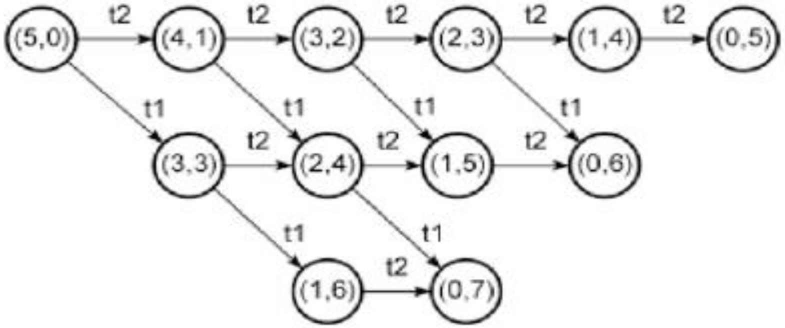




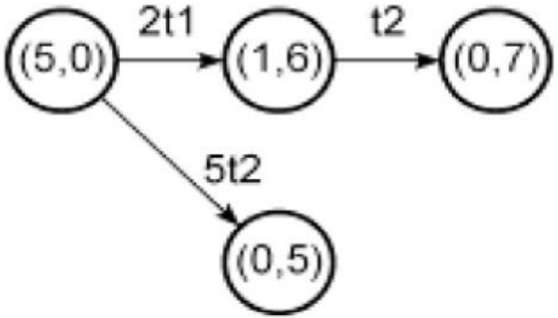
Sleptsov Net – Multiple Firing



Reachability graphs



Petri net



Sleptsov net

Inhibitor Priority Sleptsov Net

$N = (P, T, A, R, \mu_0)$

Places $p \in P$

Transitions $t \in T$

Arcs $A : P \times T \to \mathbb{Z}_{\geq 0} \cup \{-1\}, T \times P \to \mathbb{Z}_{\geq 0}$

Transition priority arcs $R : T \times T$, R^+ is a strict partial order

Marking $\mu : P \to \mathbb{Z}_{\geq 0}$

- $A(p,t) = 0,\ A(t,p) = 0$ no arc;
- $A(p,t) > 0,\ A(t,p) > 0$ regular arc of specified multiplicity;
- $A(p,t) = -1$ inhibitor arc.

Transition Firing Rule

Firing multiplicity of arc:

$$c(p,t) = \begin{cases} \mu(p)/A(p,t), & A(p,t) > 0, \\ 0, & A(p,t) = -1 \wedge \mu(p) > 0, \\ \infty, & A(p,t) = -1 \wedge \mu(p) = 0. \end{cases}$$

Firing multiplicity of transition:

$$c(t) = \min_{A(p,t)\neq 0} (c(p,t)).$$

Firing transition choice:

$$(c(t') > 0) \wedge (\forall t \in T, t \neq t', c(t) > 0 : (t,t') \notin R^{+}).$$

Next marking:

$$\mu^{\tau+1}(p) = \mu^{\tau}(p) - c^{\tau}(t')\cdot A(p,t) + c^{\tau}(t')\cdot A(t,p),\ p \in P.$$

Sleptsov Nets Run Fast

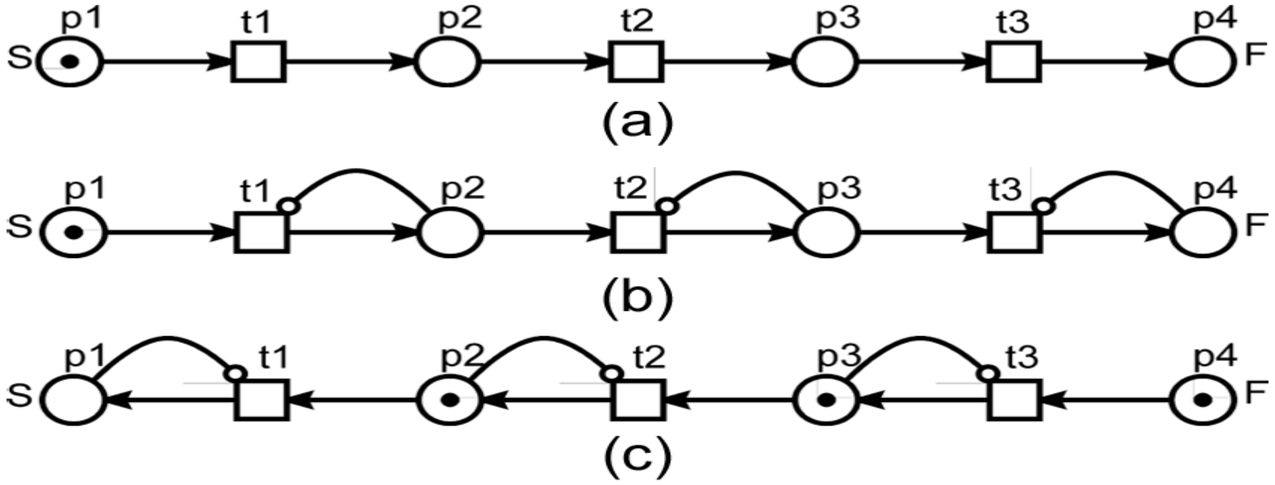
COMPARING TIME COMPLEXITIES OF OPERATIONS (LINEAR SCALE - NUMBER OF STEPS)

Operation	PN	SN
CLEAN	$x+2$	2
MOVE	$x+2$	2
COPY	$2\cdot x+3$	4
ADD	$x+y+2$	3
SUB	$\max(x,y)+3$	3
GT	$\max(x,y)+3$	4
MUL	$y\cdot(2\cdot x+3)+x+3$	$11\cdot\log_2 y+3$
DIV	$(x/y)\cdot(2\cdot y+2)+(x\%y)+y+4$	$39\cdot(\log_2 x-\log_2 y)+19$

Concurrent programming in SNs

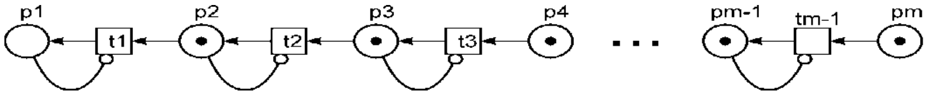
- <u>Control Flow Approach – an explicit control flow starting-finishing transformation of data</u>
- Data Flow Approach – start computations on readiness of data
- Hybrid Approach
- Unstructured approach

Peculiarities of programming in SNs

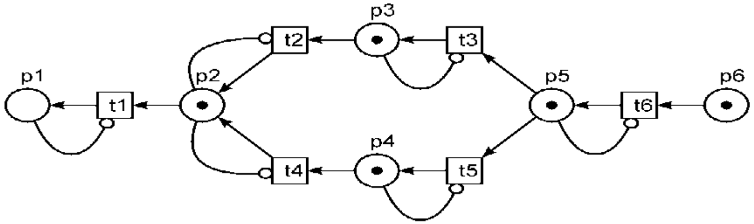


- Reversed control flow (c)
- Using inhibitor arcs to control transitions’ firing
- Infinite firing multiplicity on a valid inhibitor arc

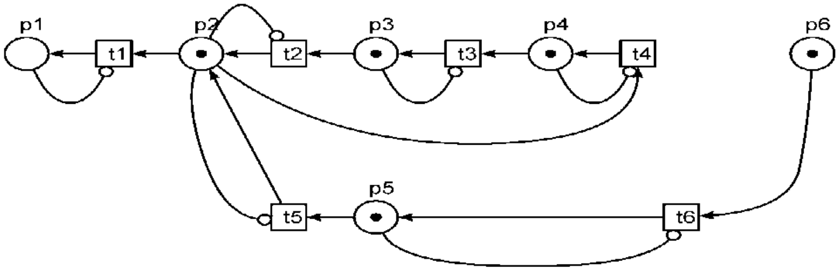
Basic statements



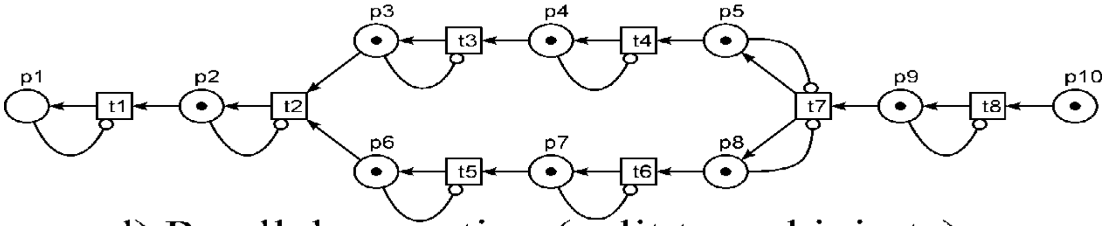
a) sequence;



b) branching;

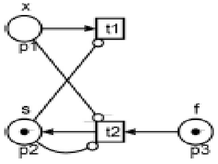


c) cycle;

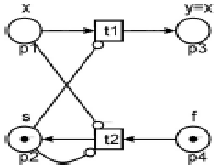


d) Parallel execution (split t_2 and join t_7)

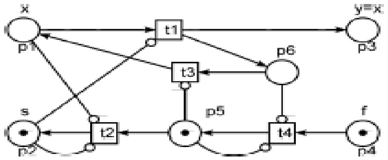
Basic subnets (subroutines)



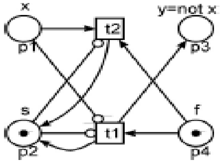
CLEAN:
$x := 0$



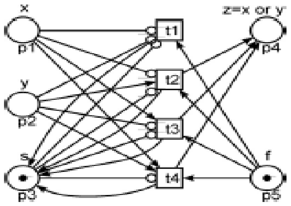
MOVE:
$y := x,\ x := 0$



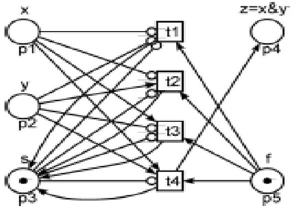
COPY:
$y := x$



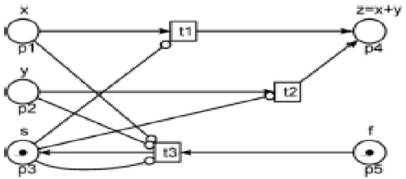
NOT(x):
$y := \neg x,\ x := 0$



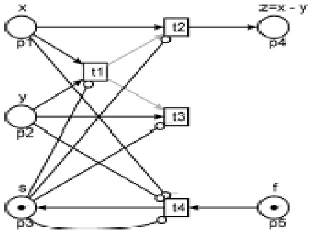
OR(x,y)
$z := x \vee y, x := 0, y := 0$



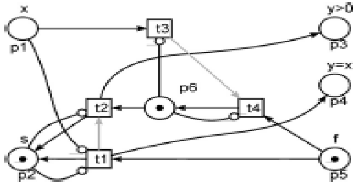
AND(x,y):
$z := x \wedge y, x := 0, y := 0$



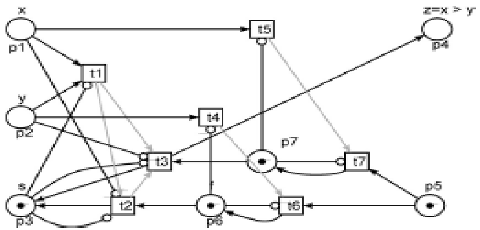
ADD(x,y):
$z := x + y,\ x := 0,\ y := 0$



SUB(x,y):
$z := x - y,\ x := 0,\ y := 0$

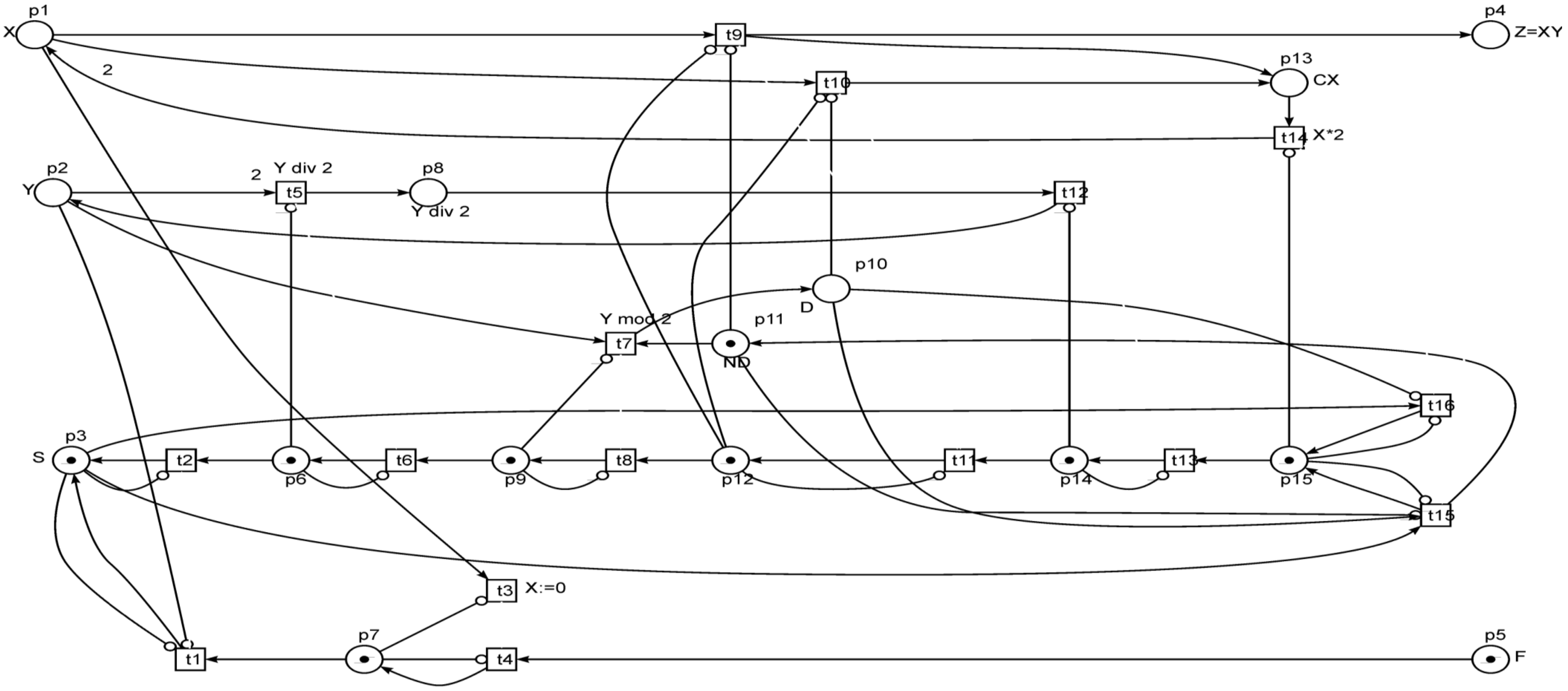


GT0(x):
$y := (x > 0),\ z := (x = 0)$

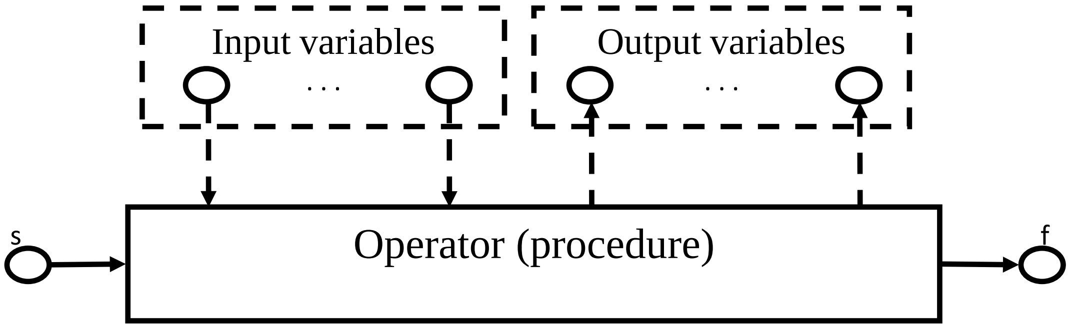


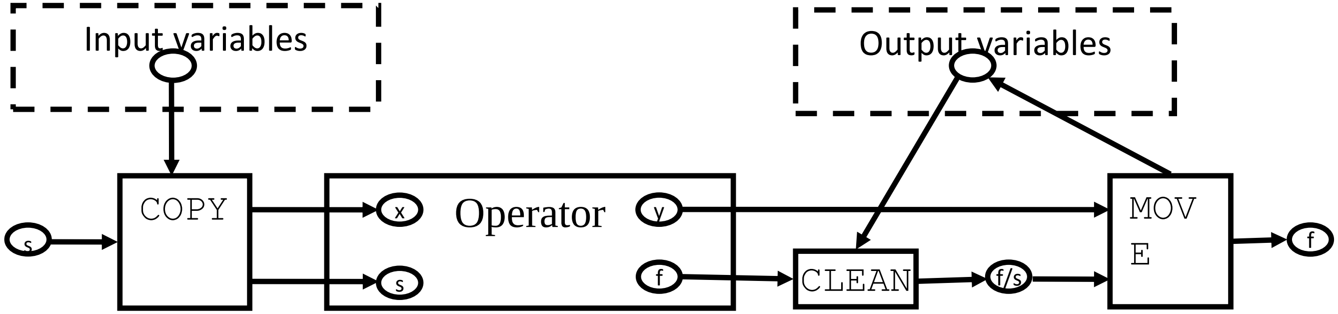
GT(x,y):
$z := (x > y),\ x := 0,\ y := 0$

Fast multiplication in Sleptsov nets

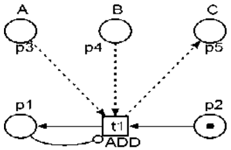


Work with variables

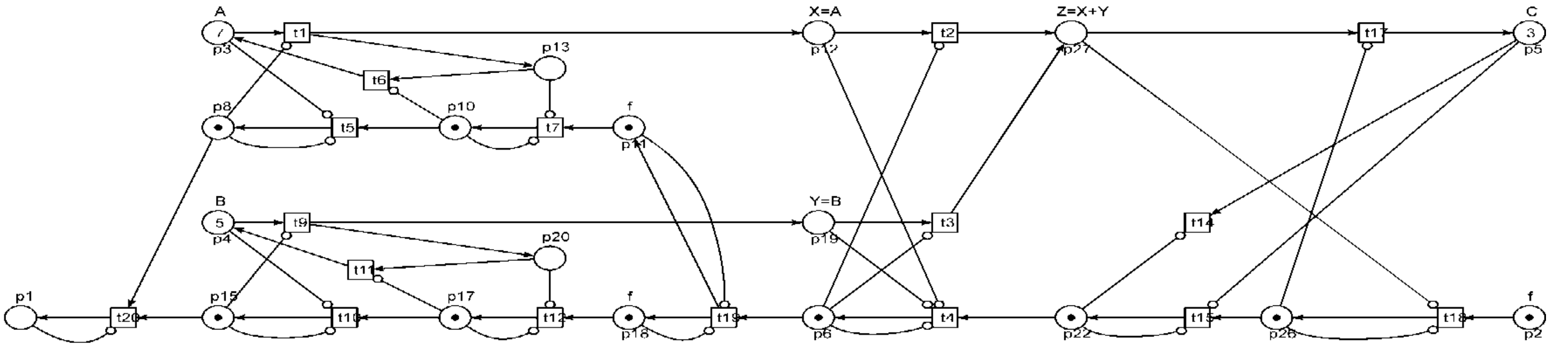




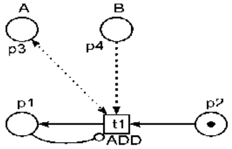
Expansion of dashed/dotted arcs



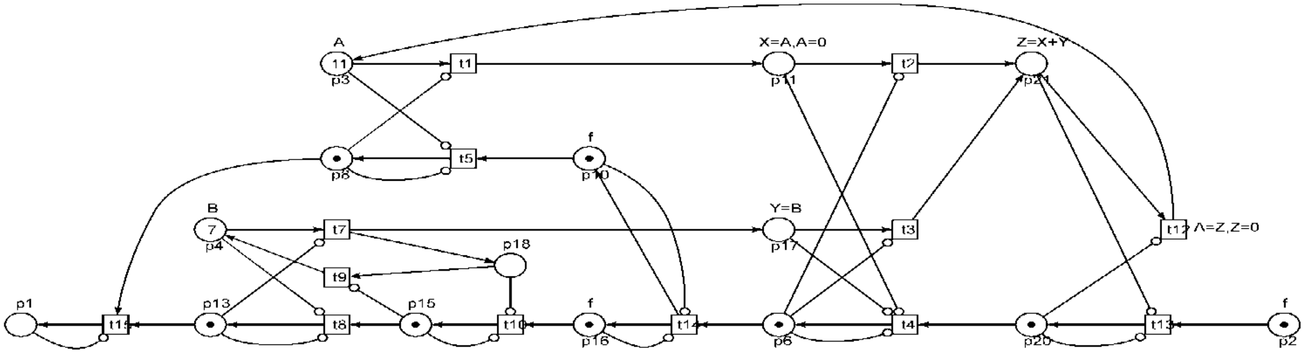
a)
c=add(a,b)



b) expansion of a)

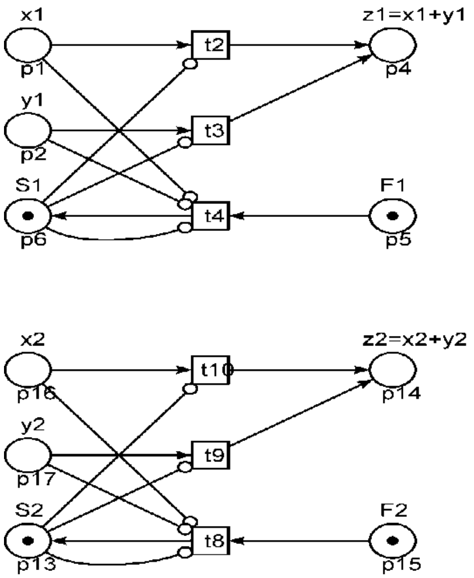


c)
a=add(a,b)

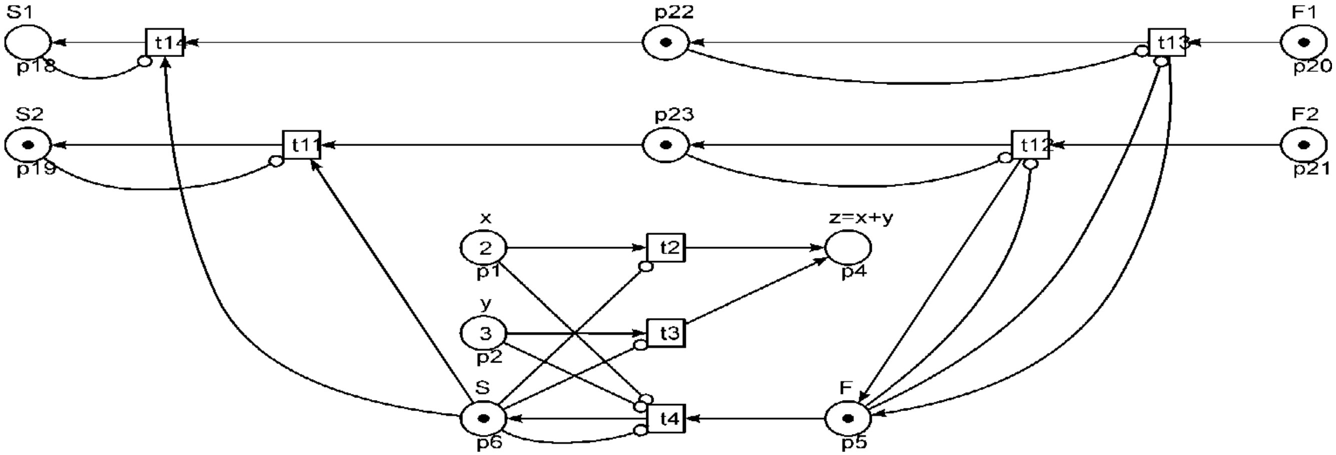


d) expansion of c)

Subnets (routines) calls



a) inline

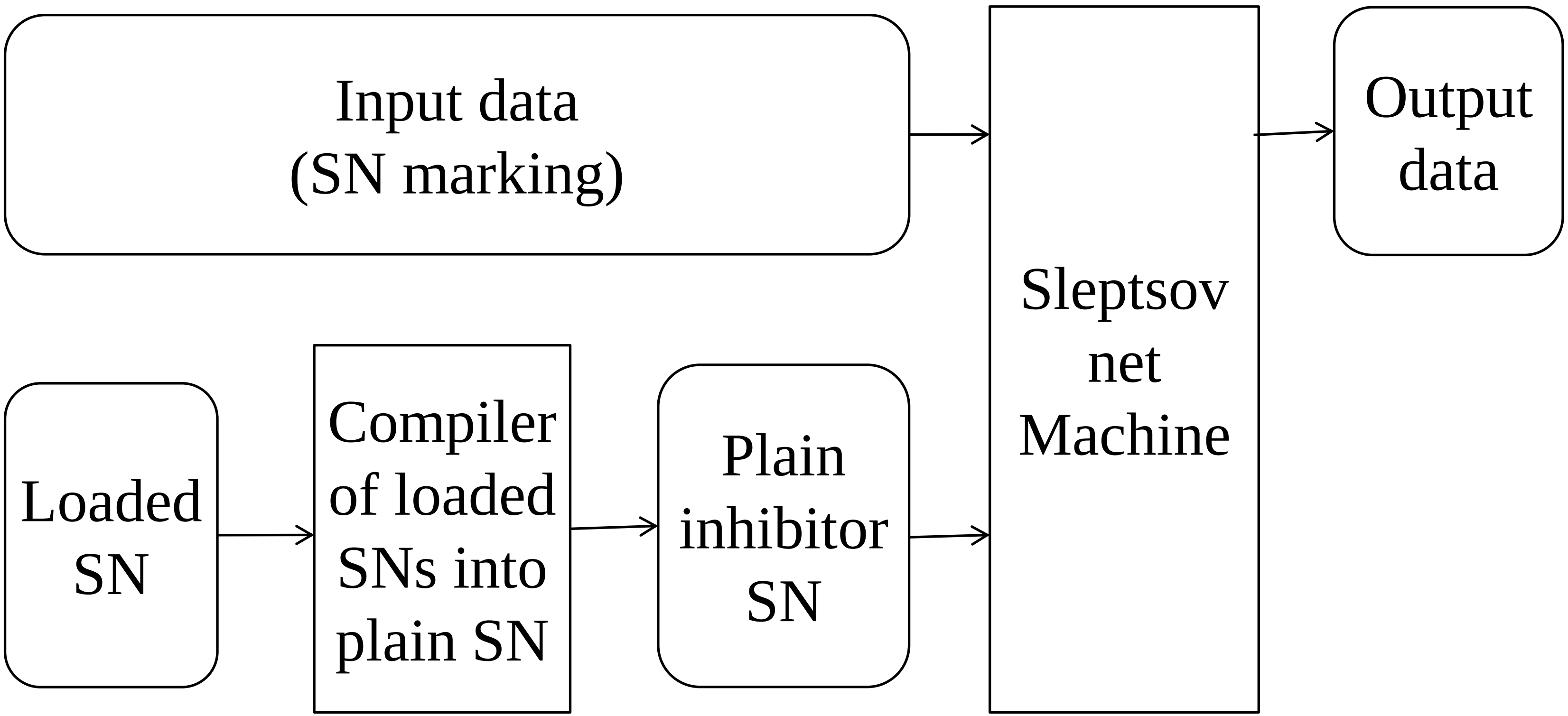


b) call-return

Verification of SN programs

- Boundedness and conservativeness
- Liveness (absence of deadlocks)
- Proof of program function
- Methods:
  - solving Diophantine systems in nonnegative numbers for linear invariants
  - solving systems of logic equations for finding siphons and traps
  - symbolic state space techniques
  - reduction and decomposition (into clans)

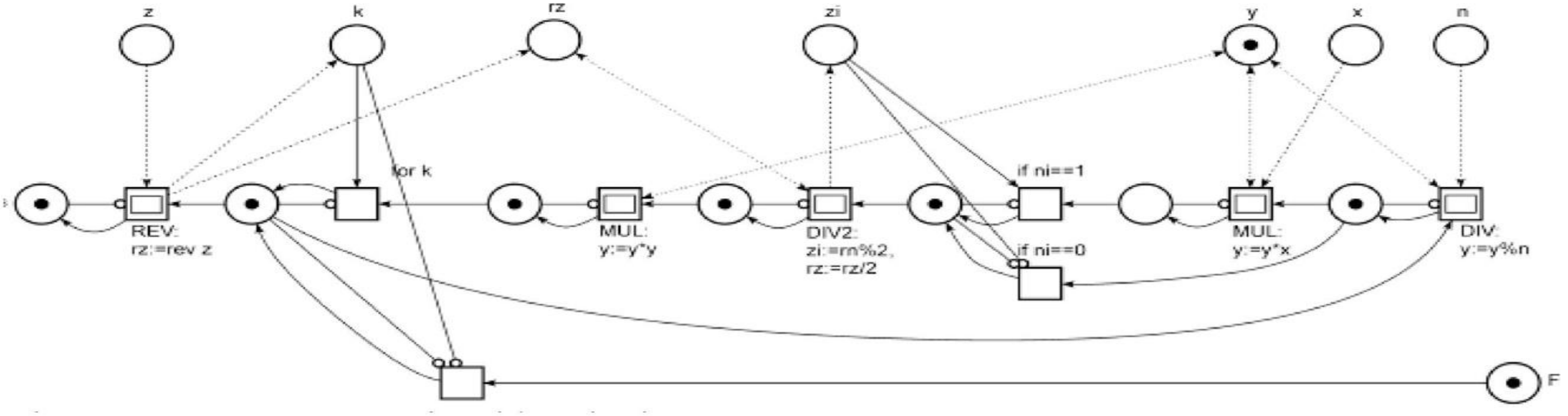
Sleptsov Net Paradigm of Computing



Sleptsov net hardware machine

- Reconfigurable multidimensional sparse matrices of connected transitions and places
- Each transition represents a separate asynchronous process that repeats the following stages:
  - lock the set of incidental places or wait their availability to lock them;
  - compute the firing multiplexity *c*;
  - if $c=0$, unlock incidental places and wait update (increment);
  - if $c>0$, fire in *c* instances and unlock incidental places.
- A transition implements Sleptsov firing rule using division, multiplication, subtraction, addition, and minimum

Examples of SN programs: RSA encoding/decoding



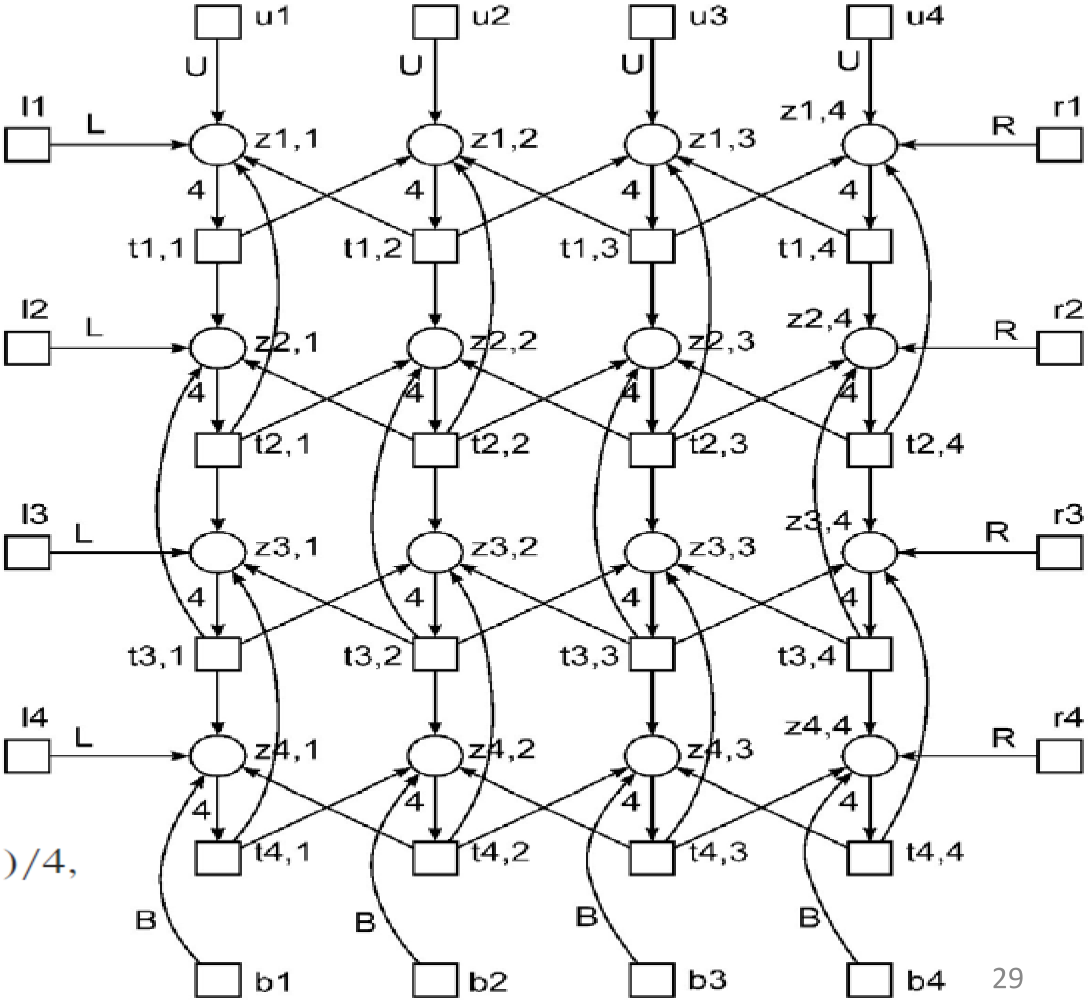
$$y = x^z \mod n$$

$$x^z = x^{(\ldots((z_k \cdot 2 + z_{k-1}) \cdot 2 + z_{k-2}) \ldots + z_2) \cdot 2 + z_1}$$
$$= \left(\ldots \left(\left(\left(x^{z_k} \right)^2 \cdot x^{z_{k-1}} \right)^2 \cdot x^{z_{k-2}} \right)^2 \cdot \ldots \cdot x^{z_2} \right)^2 \cdot x^{z_1}$$

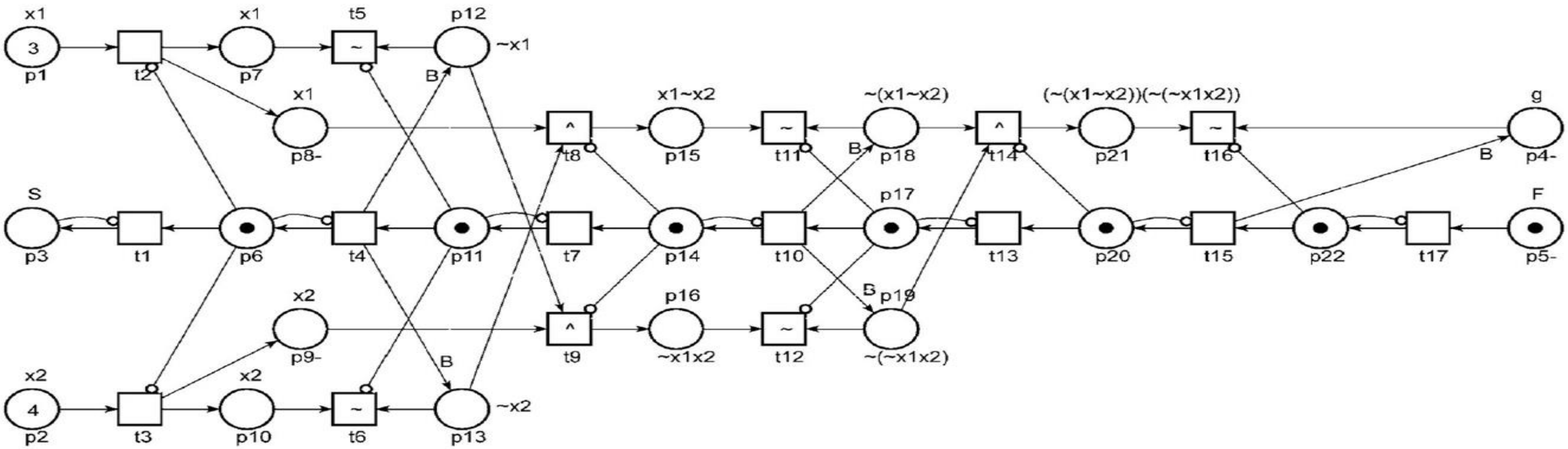
Examples of SN programs: Solving Laplace equation

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = 0$$

$$\varphi_{i,j} = (\varphi_{i-1,j} + \varphi_{i+1,j} + \varphi_{i,j-1} + \varphi_{i,j+1})/4,$$
$$\varphi_{i,j} = \varphi(x_i, y_j).$$

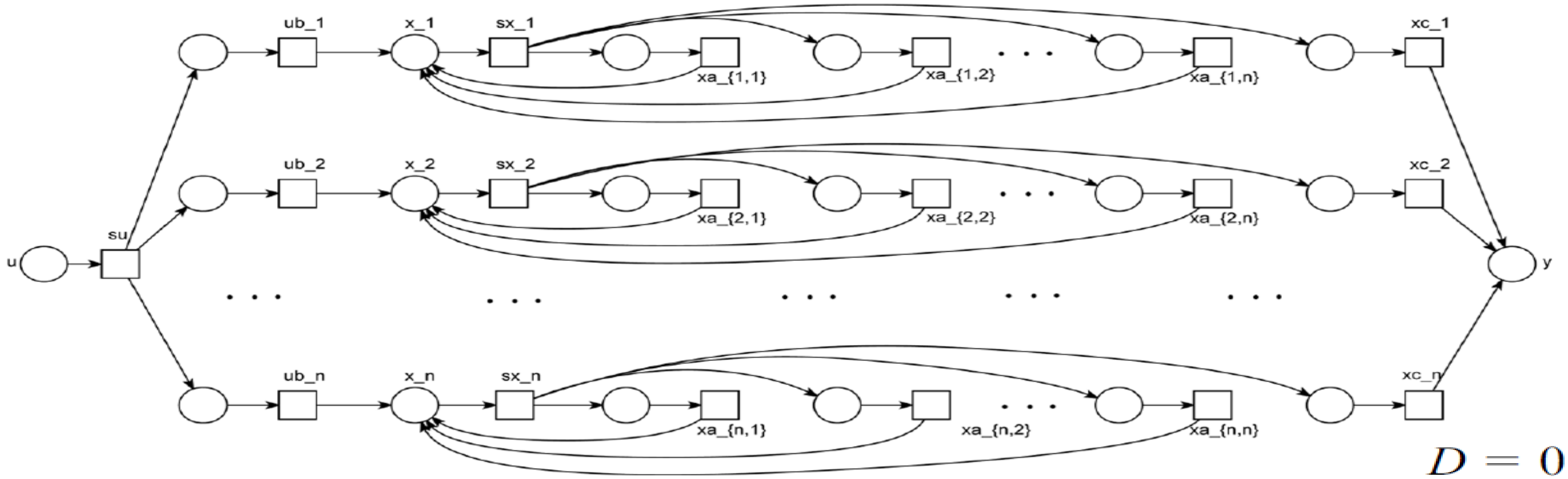


Examples of SN programs: Computing Fuzzy Logic Function



$$\varphi = x_1\bar{x}_2 \vee \bar{x}_1 x_2$$

Examples of SN programs: Discrete-Time Linear Control in Two Ticks



$$D = 0$$

$$\begin{cases} x(k+1) = Ax(k) + Bu(k) \\ y(k) = Cx(k) + Dy(k) \end{cases}$$

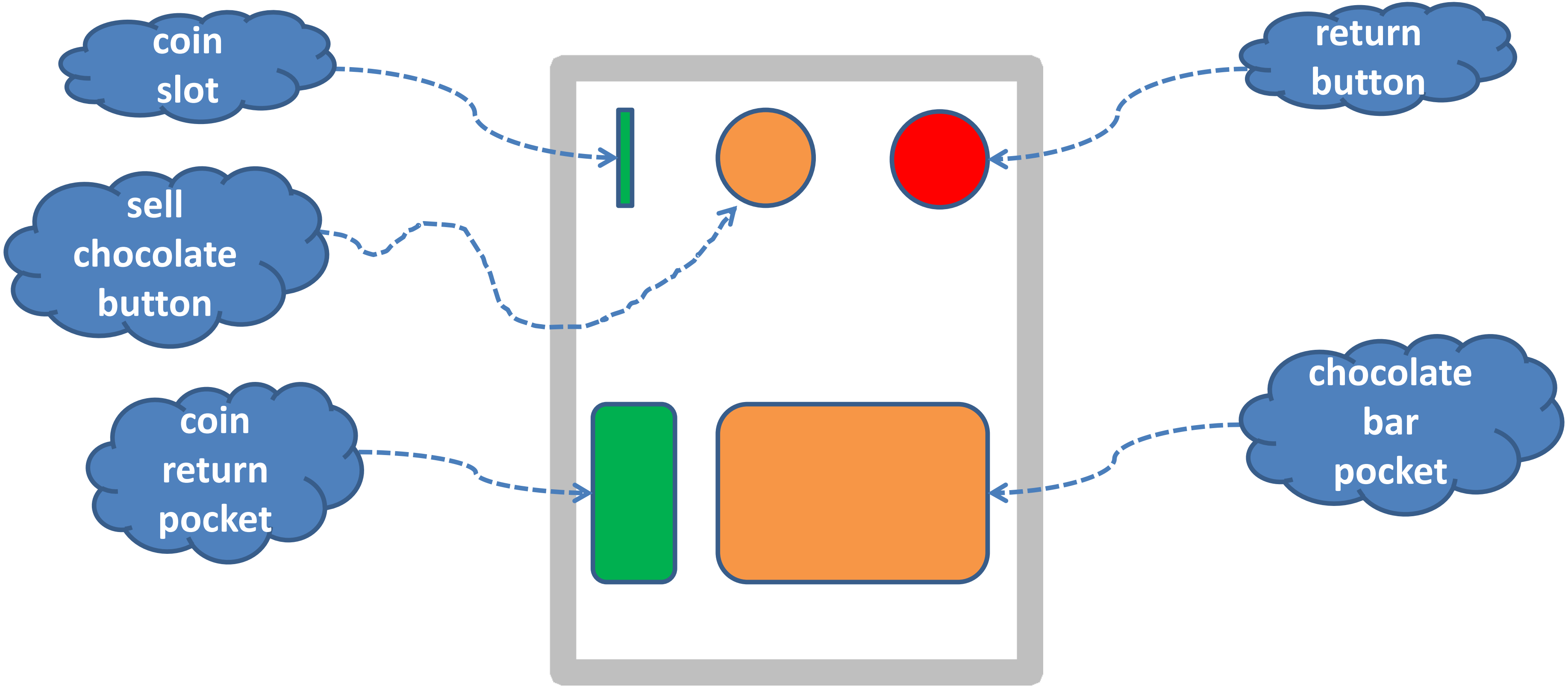
General Purpose Software Design in Sleptsov Net Machine

- SN Virtual Machine on multicore CPU – Qing Zhang, 2023
- SN Virtual Machine on GPU – Tatiana R. Shmeleva, 2023
- Compiler-linker of SN for hierarchical design – Hongfei Zhao, 2023
- Modeling system Tina for graphical design, LAAS, CNRS, France

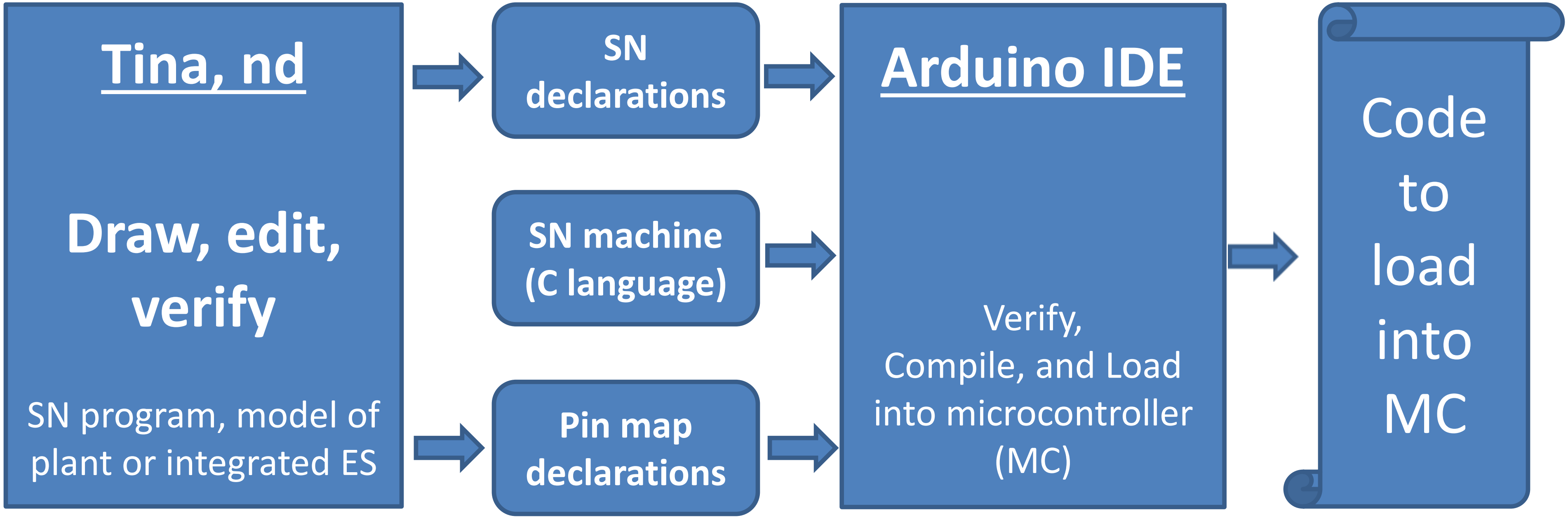
Reliable Embedded System Graphical Design in Sleptsov Net Machine

- **SN machine on microcontrollers (in Arduino IDE) – Ruiyao Xu, 2024**
- **SN machine on FPGA (in Gowin FPGA Designer) – Si Zhang, 2024**
- **Modeling system Tina for graphical design, LAAS, CNRS, France**

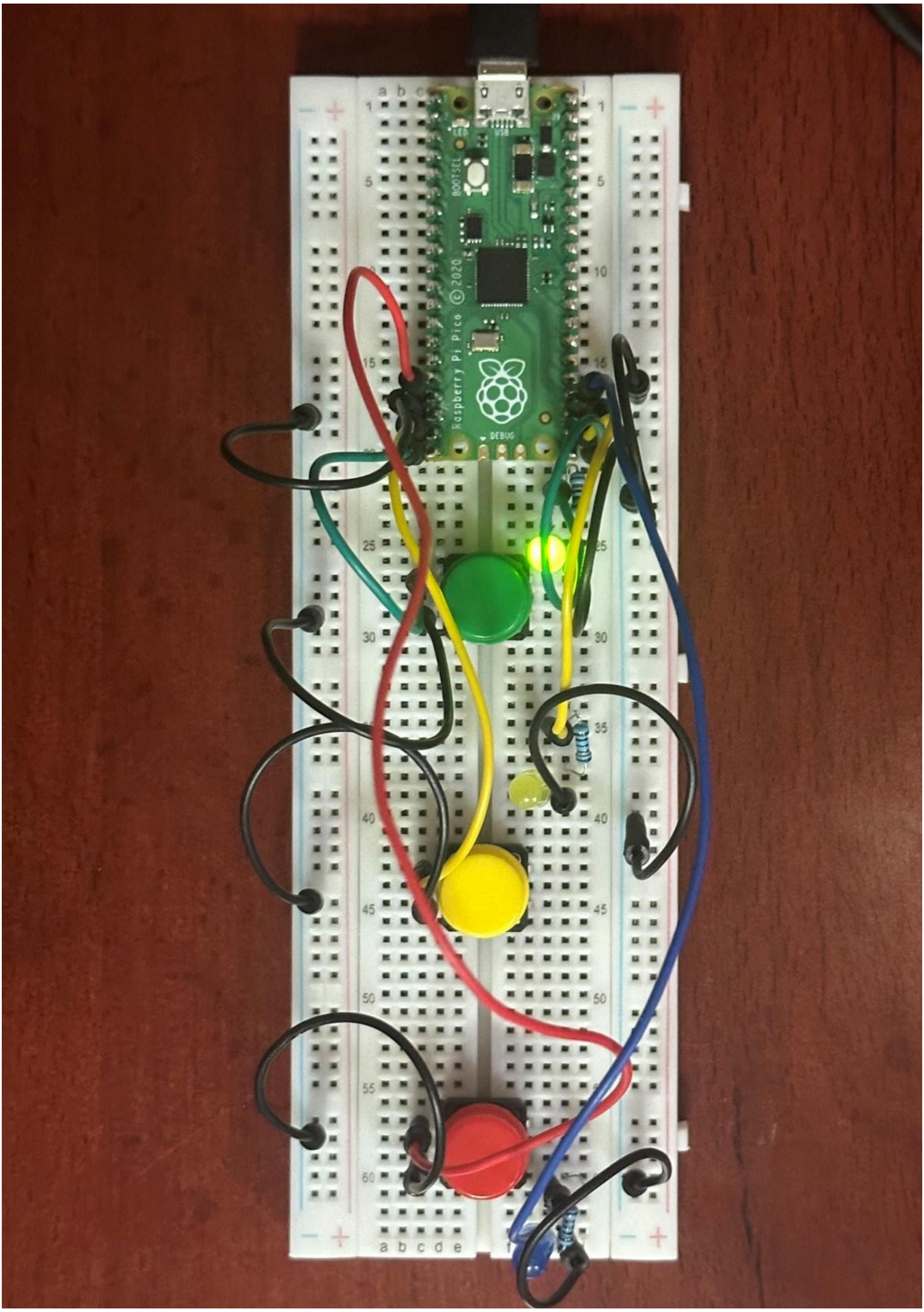
Vending machine example



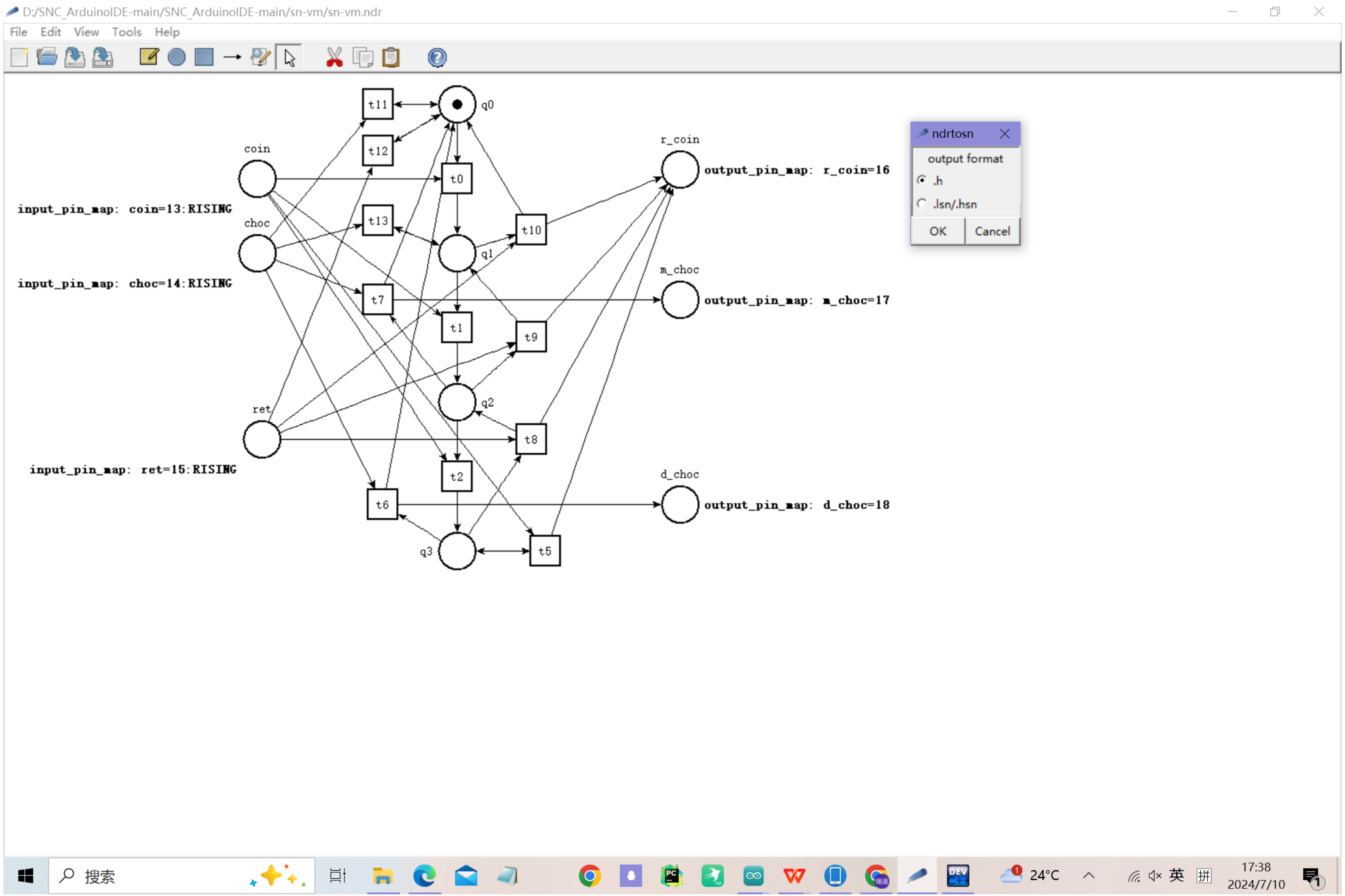
General workflow of SN machine design for microcontrollers



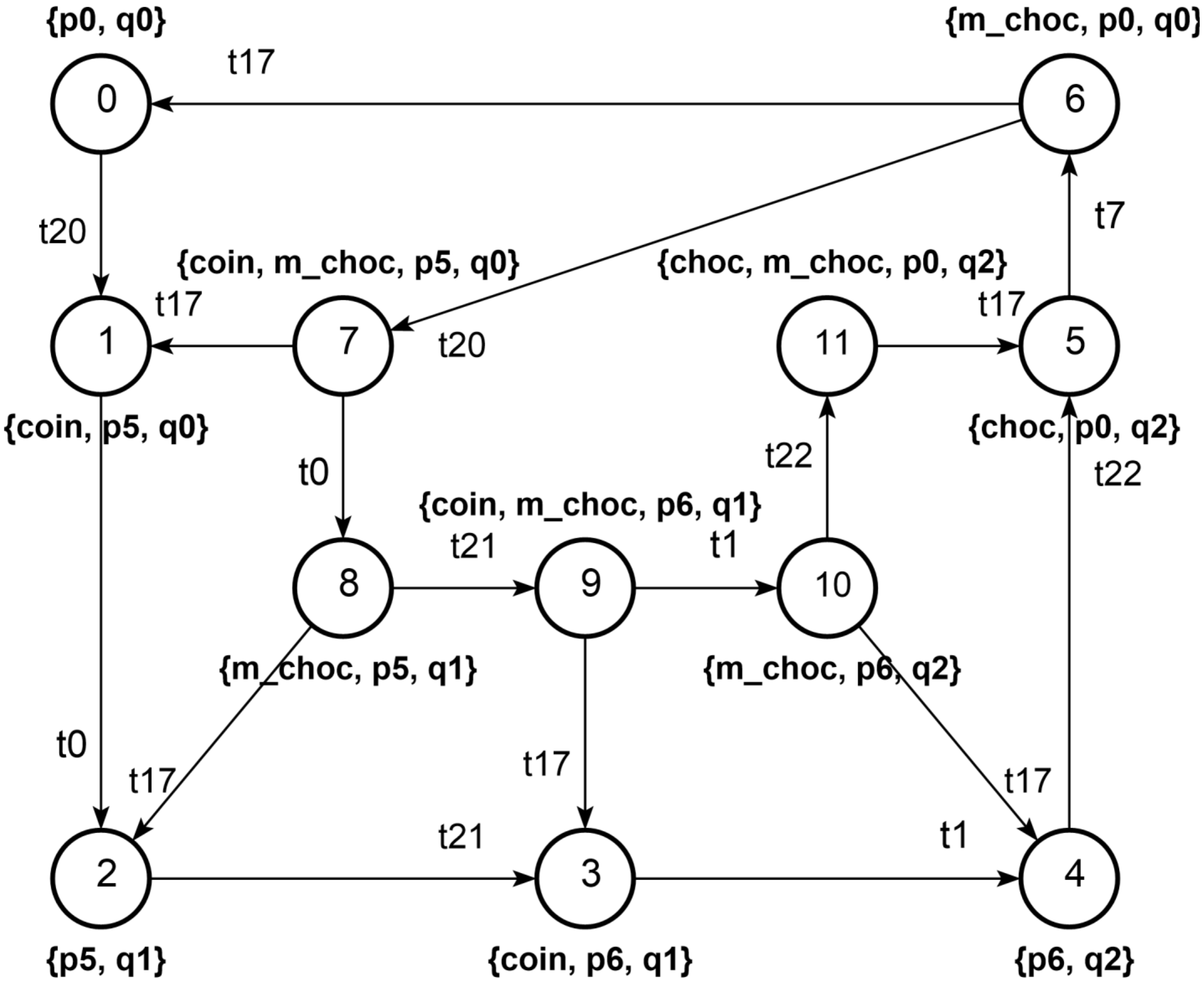
Prototyping with Raspberry Pi Pico



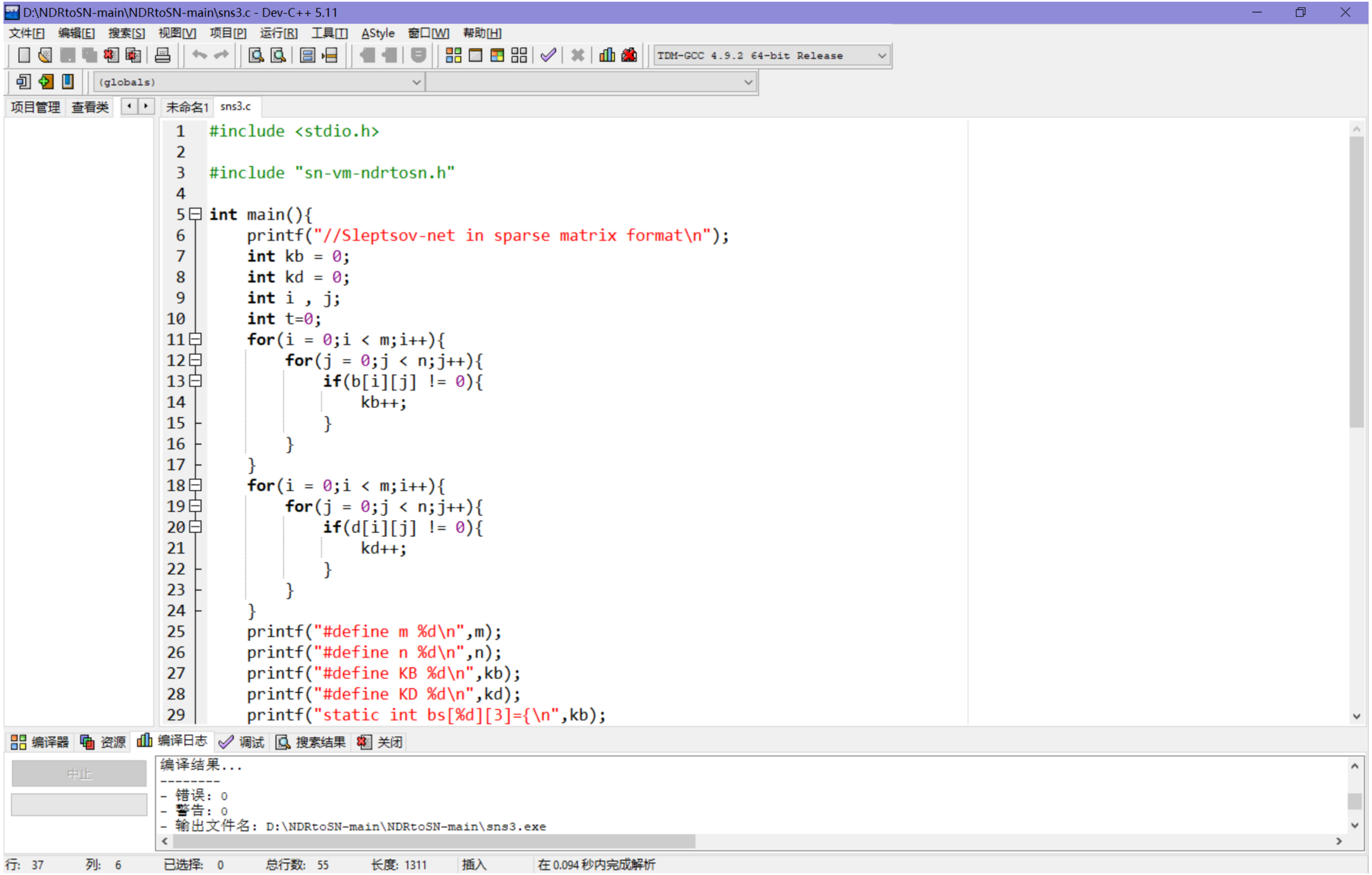
VM control SN program in Tina



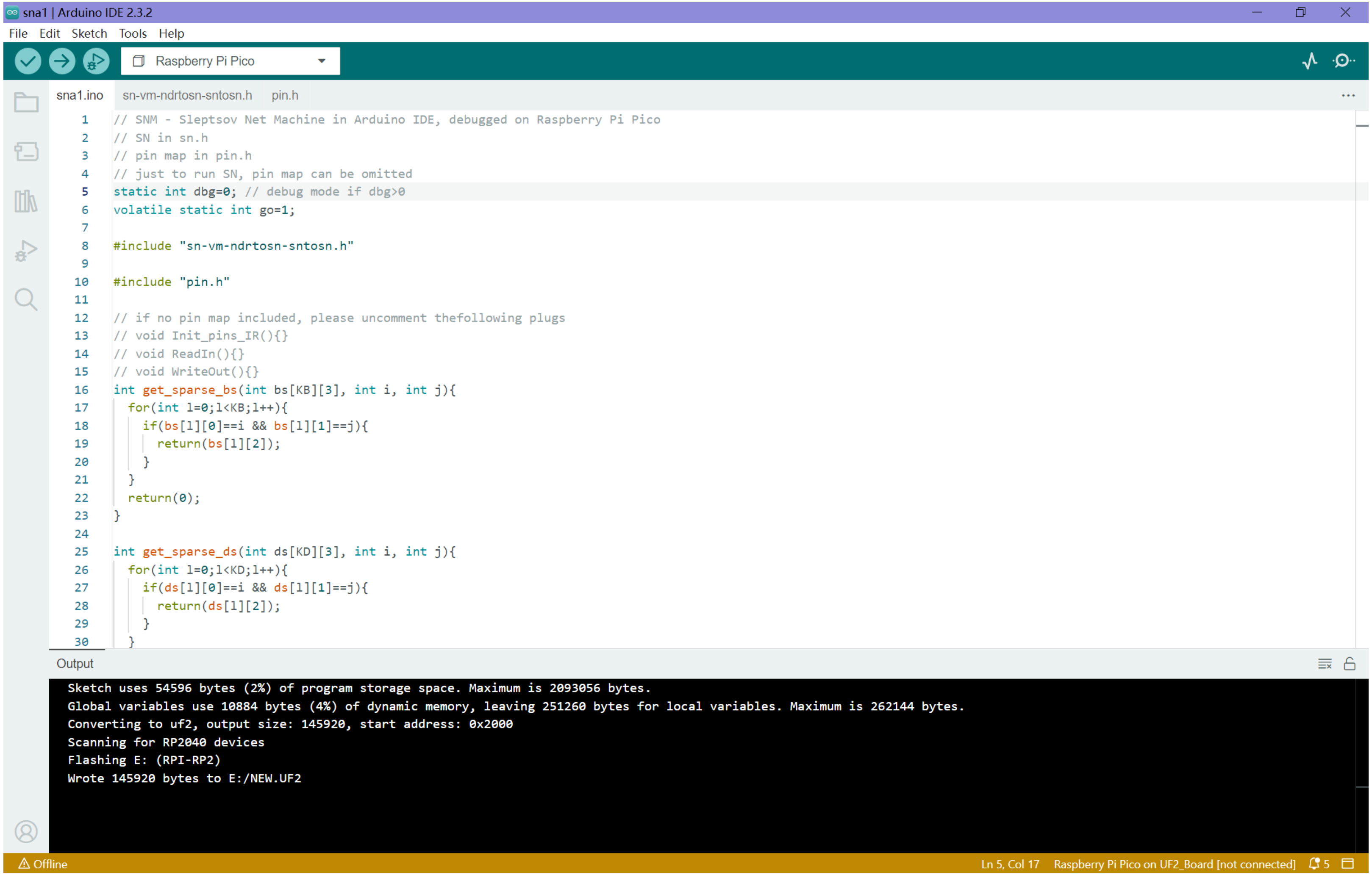
Formal verification: State Space analysis



Convert to sparse matrix format



Compose SN machine in Arduino IDE



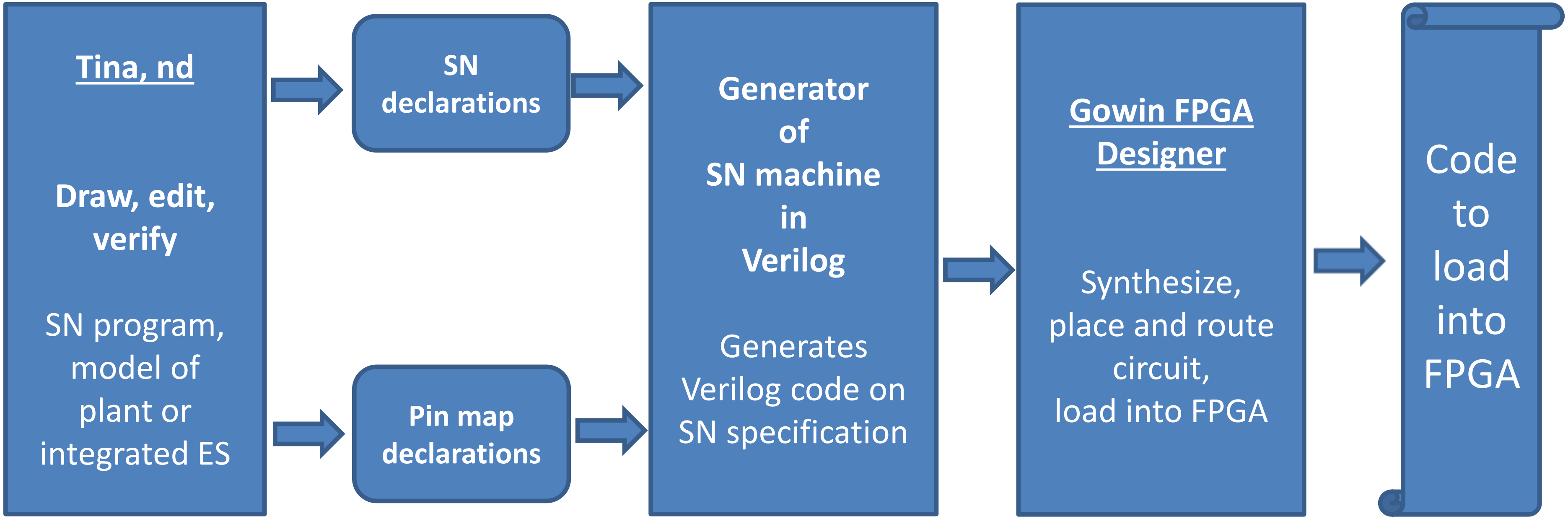
Video-presentation of SN machine work on Raspberry Pi Pico microcontroller

- Vending Machine on Raspberry Pi Pico

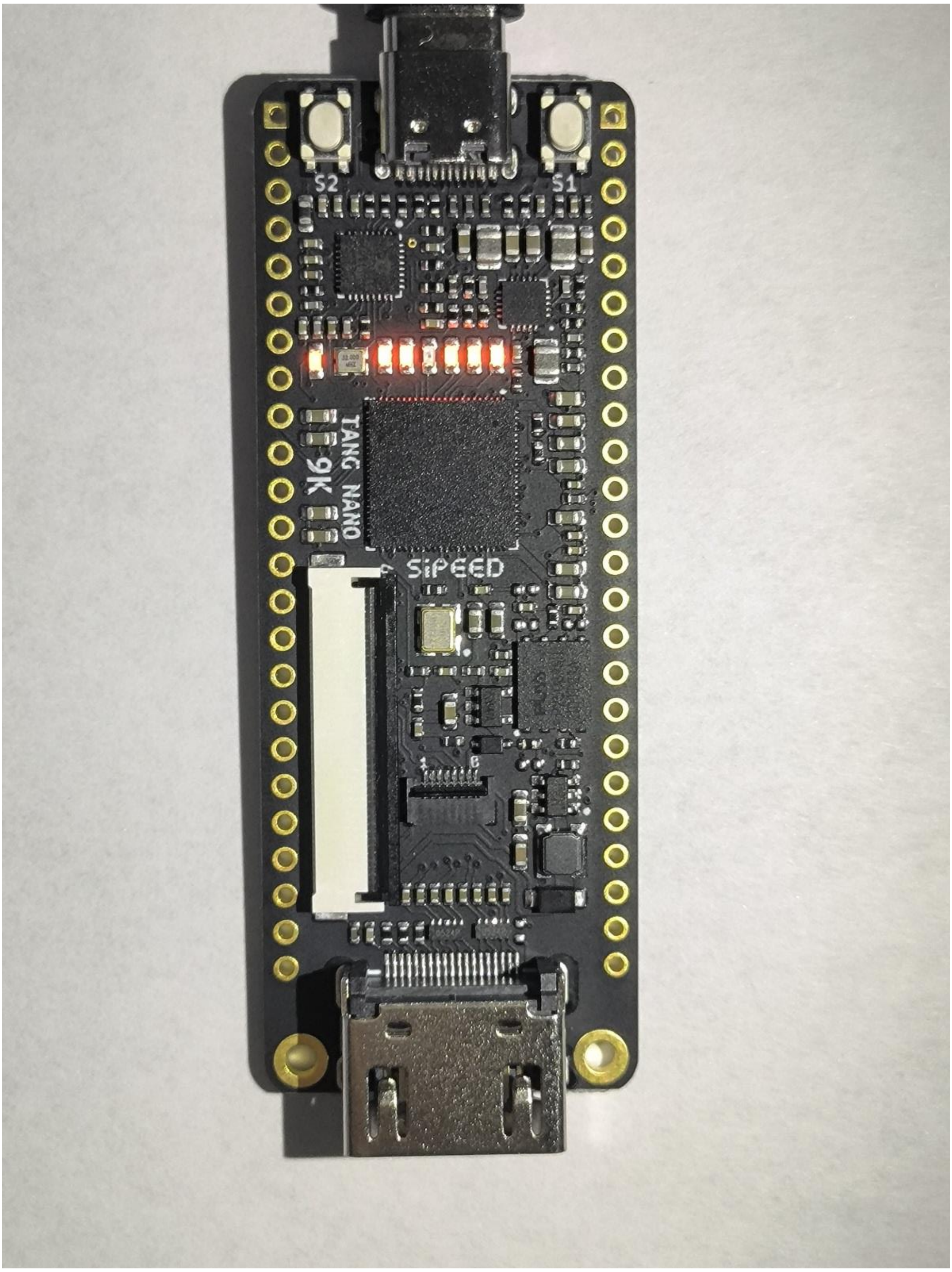
  (哔哩哔哩)

https://b23.tv/Ed21ze3

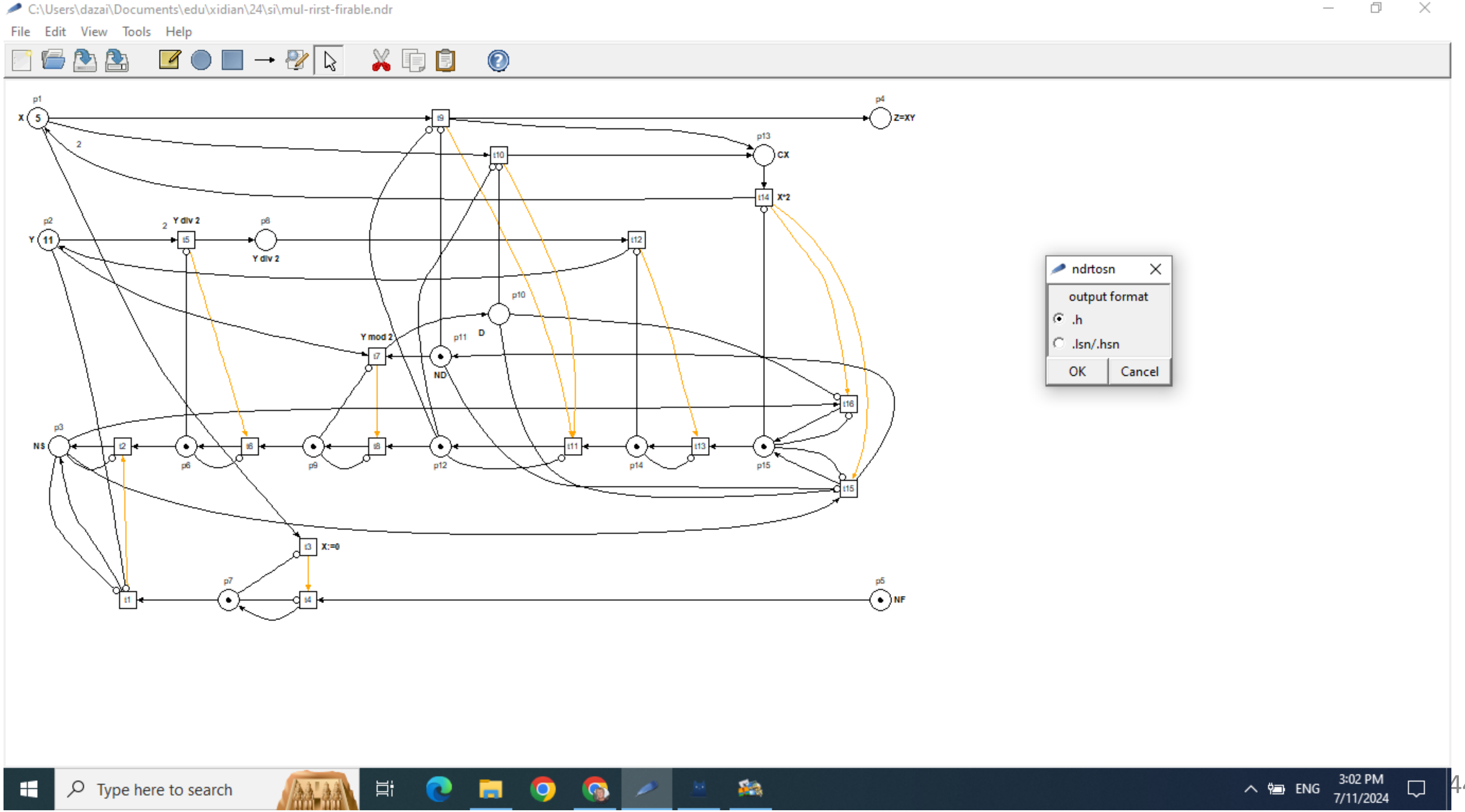
General workflow of SN machine design for FPGAs



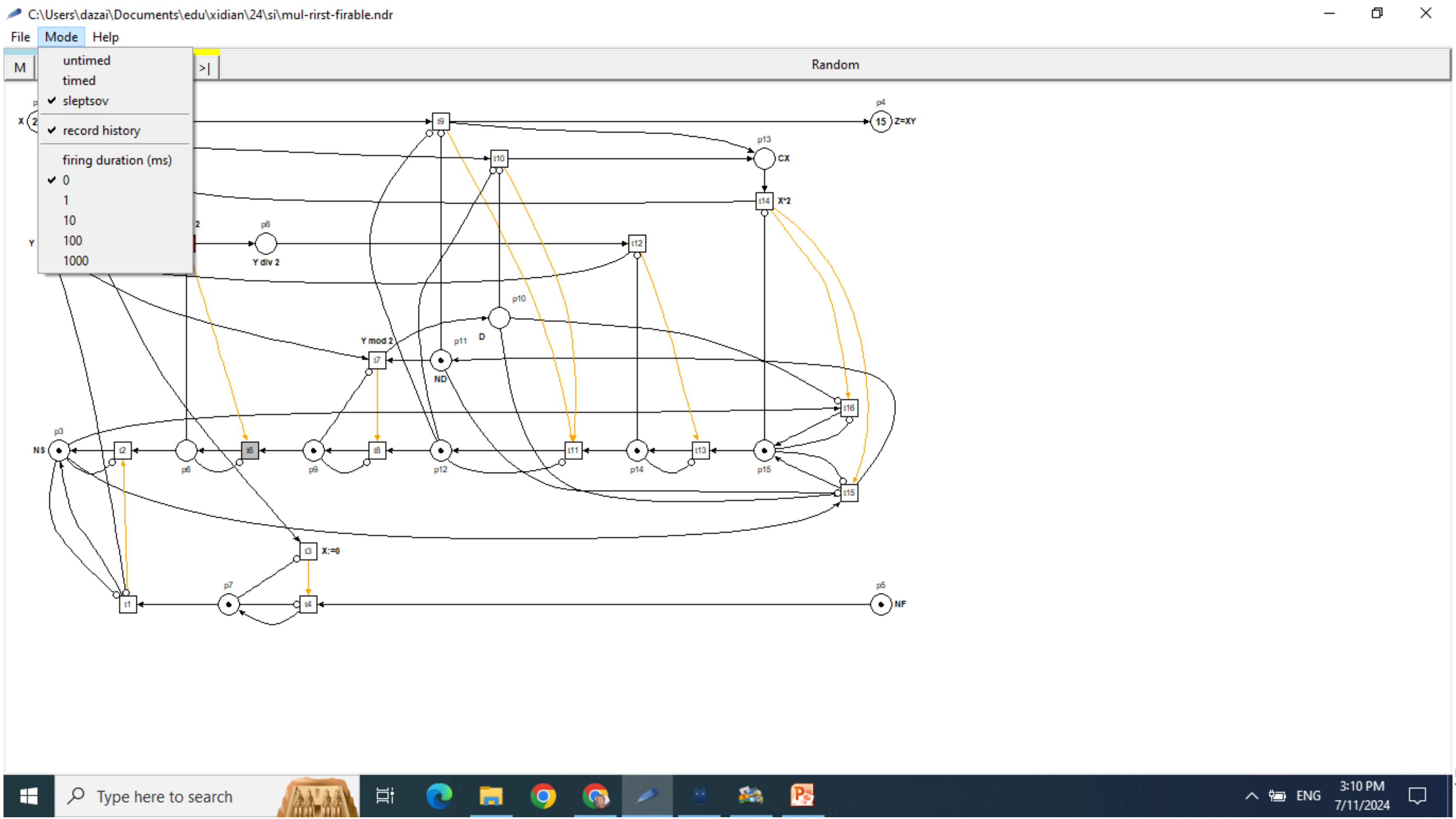
Run SN program on [FPGA Tang Nano 9K]



SN program for multiplication in Tina



Run SN program in Tina



SN machine generator (Dev-C)

D:\1yjsqj\zs\Task\Task4\generator\mul_snvg.cpp - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

Project Classes | mul_snvg.cpp | mul-ndrtosn.h

```
#include <stdio.h>
#include "mul-ndrtosn.h"    //mul

#define pres "[5:0]"
#define RES_PL 3

void int_to_binary(unsigned int num, char* binary) {
    int i;
    for (i = 5; i >= 0; i--)
    {
        binary[5 - i] = (num & (1 << i)) ? '1' : '0';
    }
}

int main()
{
    int i,j,t;
    char binary[6];

    printf("reg%s ",pres);
    for(i=0;i<m;i++)
    {
        printf("p%d",i);
        printf("%c",(i<m-1)?',':';');
    }
    printf("\nreg%s ",pres);
    for(j=0;j<n;j++)
    {
        printf("f%d",j);
        printf("%c",(j<n-1)?',':';');
    }
    printf("\nreg%s tf;",pres);
    printf("\nreg%s tc;",pres);
    printf("\nalways @(posedge sys_clk or negedge sys_rst_n) begin\n\
    if(!sys_rst_n) begin\n\
    \ttf  <= 6'b000000;\n\
    \ttc  <= 6'b000000;\n\
```

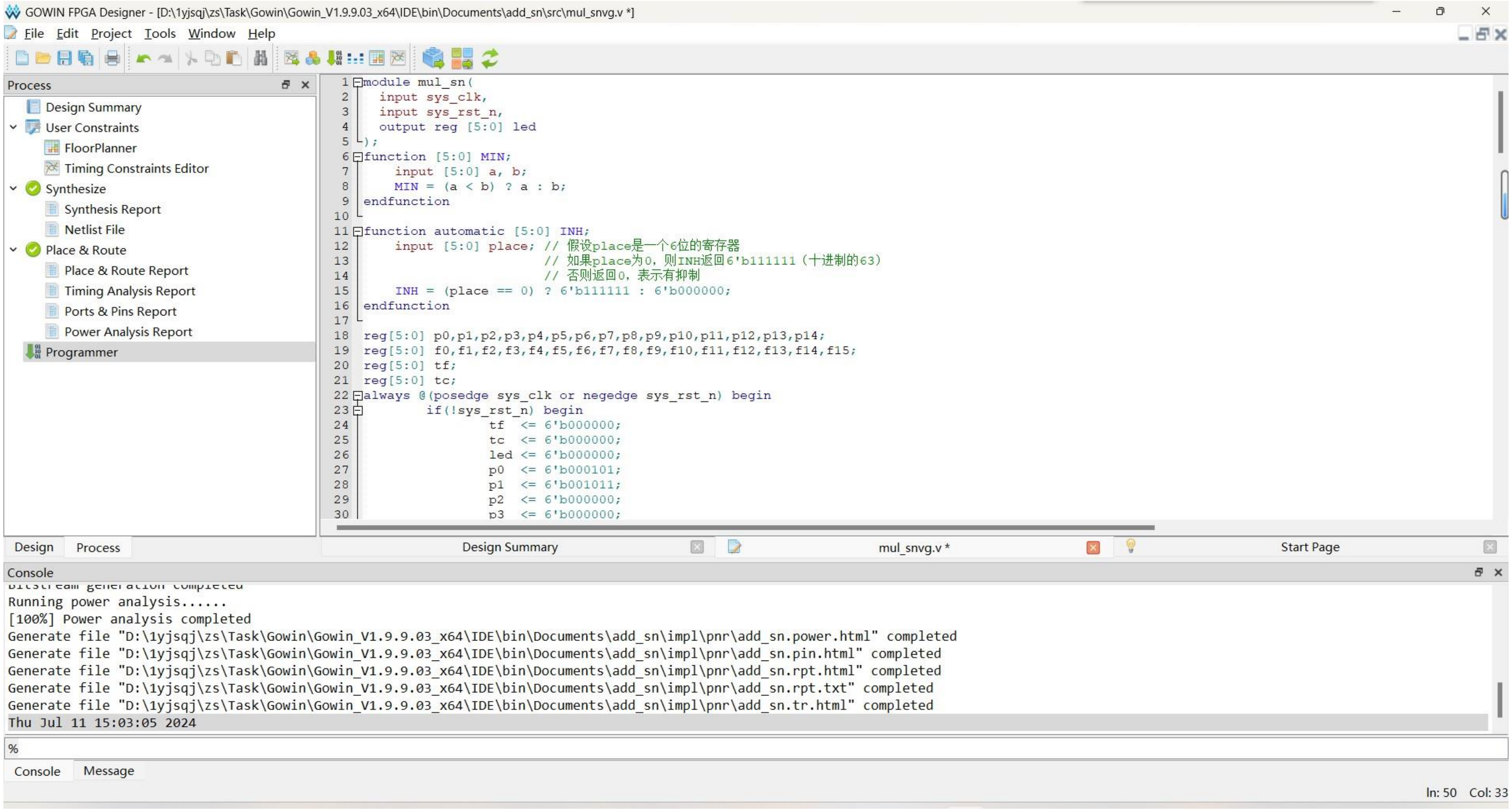
Compiler | Resources | Compile Log | Debug | Find Results | Close

Abort Compilation

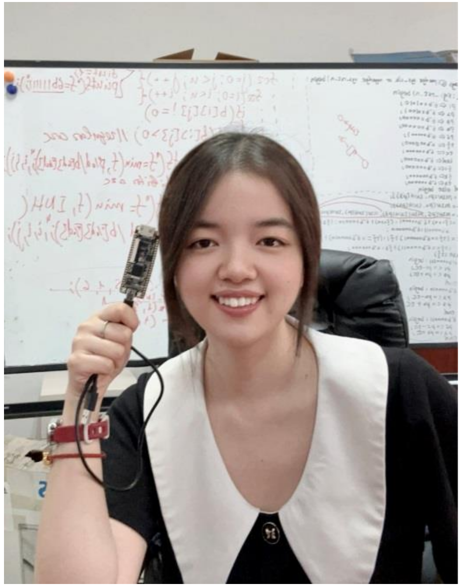
```
- Errors: 0
- Warnings: 0
- Output Filename: D:\1yjsqj\zs\Task\Task4\generator\mul_snvg.exe
- Output Size: 133.880859375 KiB
- Compilation Time: 0.64s
```

Line: 51 Col: 34 Sel: 0 Lines: 105 Insert Done parsing in 0.016 seconds

Verilog code in Gowin FPGA Designer



Video-presentation of SN machine work on FPGA Tang Nano 9k (Gowin IC)



- SN machine design for FPGAs

  (哔哩哔哩)

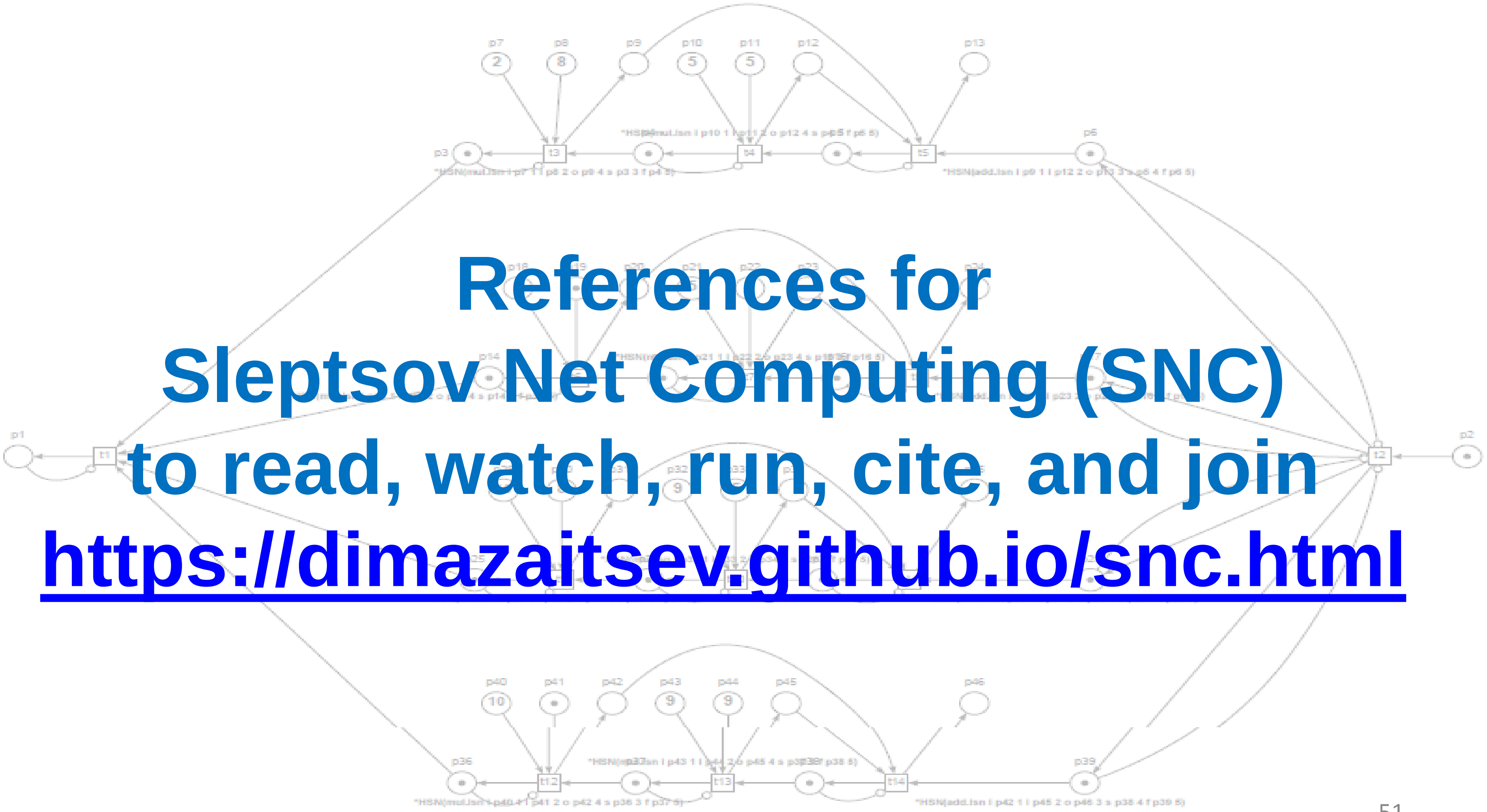
https://b23.tv/c8JBHiY

Conclusions

- Sleptsov net computing benefits:
  - graphical concurrent programming language,
  - formal verification of concurrent programs,
  - fine granulation of parallel processes,
  - massively parallel computations,
  - fast computing memory hardware concept
- Sleptsov net computing application area:
  - general purpose computing
  - reliable embedded system design

Basic References

- Zaitsev D.A. Sleptsov Nets Run Fast, IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2016, Vol. 46(5), 682–693.
- Dmitry A. Zaitsev, Strong Sleptsov nets are Turing complete, Information Sciences, Vol. 621, 2023, 172-182.
- Bernard Berthomieu, Dmitry A. Zaitsev, Sleptsov Nets are Turing-complete, Theoretical Computer Science, Volume 986, 2024.
- Dmitry A. Zaitsev, Tatiana R. Shmeleva, Qing Zhang, and Hongfei Zhao, Virtual Machine and Integrated Developer Environment for Sleptsov Net Computing, Parallel Processing Letters, Vol. 33, No. 03, 2350006 (2023).



References for Sleptsov Net Computing (SNC) to read, watch, run, cite, and join

https://dimazaitsev.github.io/snc.html