西南交通大学 Southwest Jiaotong University

UPEC UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE Connaissance - Action

FPGA Implementation of Numerical P Systems

Reporter: Zeyi Shang

2019.2.6

Content | Index

The Brief Introduction of Numerical P Systems (NPS)

Comparing to the symbol object P systems, the NPSs contain only one program in each membrane region. The left side of the program is called production function involving real-valued variables and the right side is call distribution protocol, as shown bellow:

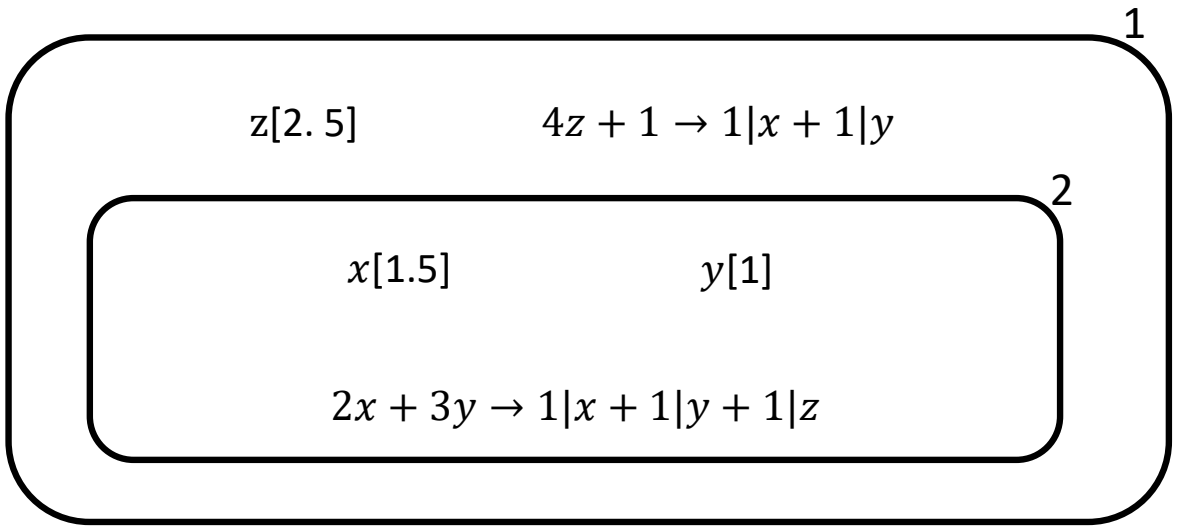
Program: $2 * x + 3 * y \rightarrow 1|x + 2|y$

production function

distribution protocol

The integer coefficient in front of the vertical line denotes the percentage of the outcome of production function, which will be assigned to the variable behind the vertical line.

A example of NPS



	Initial value	After Step 1	After Step 2
x	1.5	7.5	17
y	1	7.5	17
z	2.5	2	12.5

The Brief Introduction of Enzymatic Numerical P Systems (ENPS)

The ENPS contains **multiple** programs in one membrane region. There are several special variables working as the enzyme to catalyze corresponding programs, if the value of a enzyme meets a quantitative condition. A program of ENPS can be depicted as below:

Program: $2 * x + 3 * y(e \rightarrow) \ 1|x + 2|y$

enzyme variable

production function

distribution protocol

Assume that the quantity condition is: $e \geq \min\{x, y\}$. This program will happen when $e \geq \min\{x, y\}$, and will not happen if e fails to meet. Note that different programs can happen in parallel.

A example of ENPS

ComputeQuadraticofOutputError

$sg[u_k]$ $E_{sg}[0]$ $\mathrm{sgn}[0]$ $ac[0]$ $w_i[\omega_i]$ $x_i[e_i]$

$\mathrm{Pr}_{1,accumulate_i}$: $\xi_{pk\_i} * x_1 * x_i(E_i \to)1 \mid ac$

$\mathrm{Pr}_{2,accumulate_i}$: $0 * sg - 1(E_{sg} \to)1 \mid \mathrm{sgn}$

$\mathrm{Pr}_{3,accumulate_i}$: $2 * sg \to 1 \mid E_{sg}$

$\mathrm{Pr}_{4,accumulate_i}$: $0 * sg + 1(E_{sg} \to)1 \mid \mathrm{sgn}$

$\mathrm{Pr}_{5,accumulate_i}$: $2 * ac * \mathrm{sgn}(E_i \to)1 \mid Sum_i + 1 \mid Sum_{all}$

$\mathrm{Pr}_{6,accumulate_i}$ $2 * w_i(E_i \to)1 \mid Sum_i + 1 \mid Sum_{all}$

$\mathrm{Pr}_{7,accumulate}$: $1.1 * E_i \to 1 \mid E_H$

ComputeQuadraticofControlIncrement

$S_{um}[0]$

$\mathrm{Pr}_{1,wight_i}$: $w_i * x_i(E_i \to)1 \mid S_{um}$

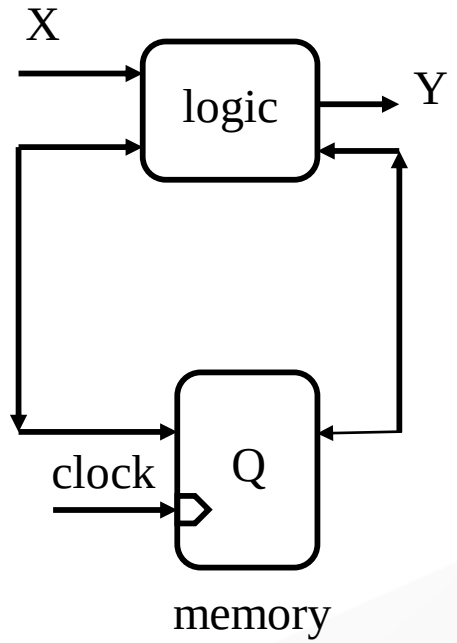
$\mathrm{Pr}_{2,wight_i}$: $-2 * \zeta_{qk\_i} * x_i * S_{um}(E_i \to)1 \mid Sum_i + 1 \mid Sum_{all}$

$\mathrm{Pr}_{3,wight_i}$: $1.1 * E_i \to 1 \mid E_H$

Sequential Circuits

Digital processing systems consist of sequential circuits. Assume X(t) is the input at time t, Y(t) the output and Q(t) the internal state at time t. X(t), Y(t) and Q(t) are Boolean vectors. A sequential circuit is driven by the following system of equations (t+1 means next time step):

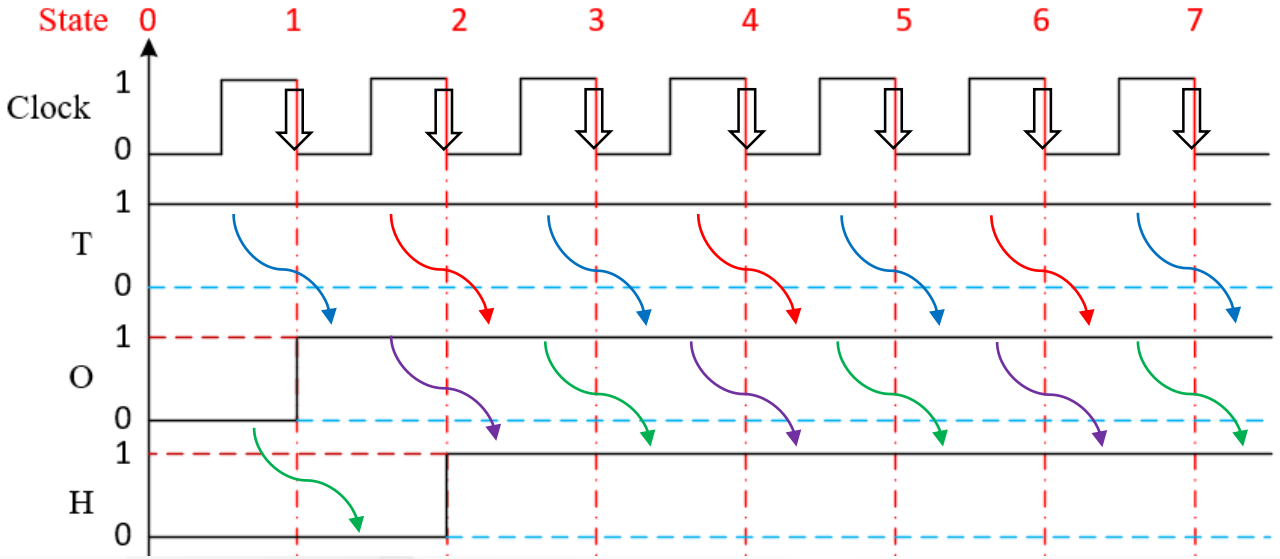
$$\begin{cases} Q(t+1) = F(Q(t), X(t)) \\ Y(t) = G(Q(t), X(t)) \end{cases}$$



$$2a + 3b \to 1|a + 2|b \Longrightarrow \begin{cases} \mathrm{a(t+1)} = \frac{1}{3}[2\mathrm{a(t)} + 3\mathrm{b(t)}] \\ \mathrm{b(t+1)} = \frac{2}{3}[2\mathrm{a(t)} + 3\mathrm{b(t)}] \end{cases}$$

Sequential Circuits

For a sequential circuit, a clock generated from on-board oscillator is the indispensable component, because the clock gives the “time” which can be felt by the circuit. In fact, sequential circuits are triggered by the rising edge or falling edge of the global clock. Particularly, when a clock edge arrives, the output and state of the circuit change. When the clock not arrives, the output and state of the circuit just hold.



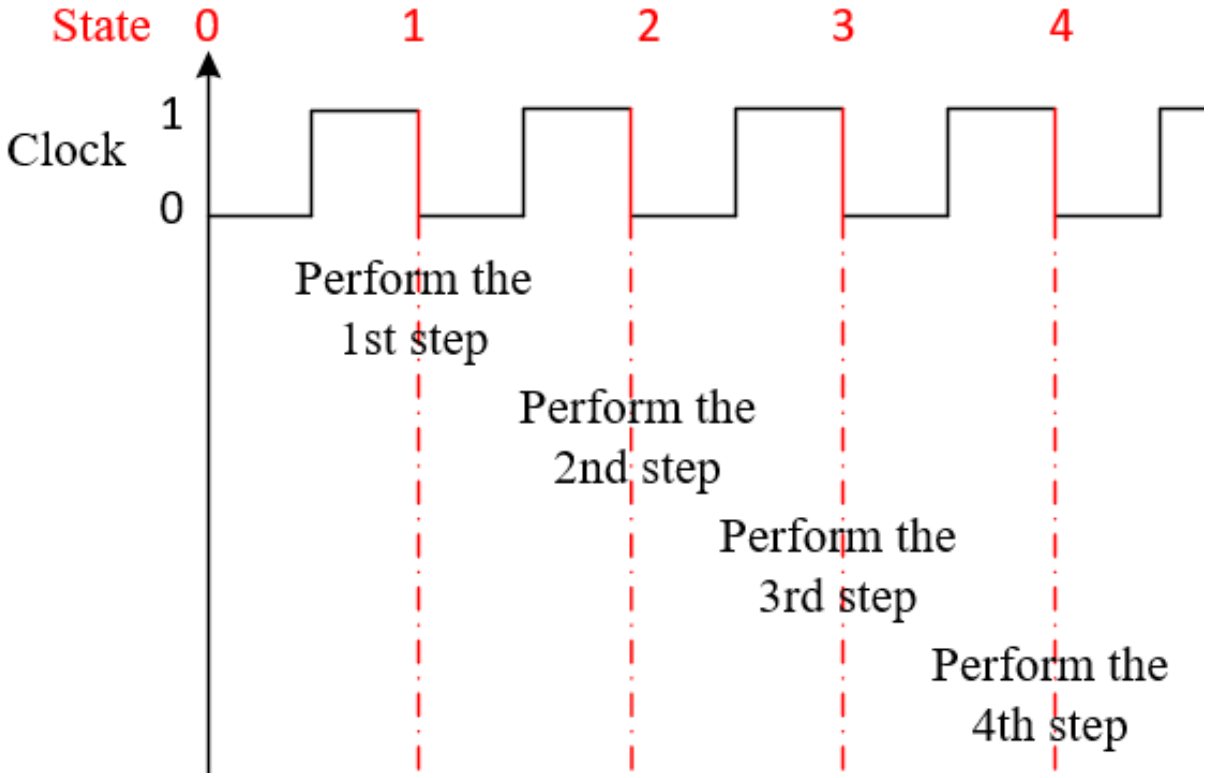
always do at the falling edge
begin
$O \Leftarrow T$;
$H \Leftarrow O$;
end

System Translation

In theory, there is no stop between two computing step. But a sequential circuit works in a pulse mode: the state and value are transmitted only when the clock arrives. Alternately, the clock edge can be regarded as a driver for the sequential circuit.

One possible solution: a computing step of NPS is executed at clock edge. By this way, the continuous computation of NPS is transformed to the pulse mode of sequential circuit.

System Translation



Membrane Representation

The function of membrane is to delimit compartments to provide a space for variables and rules. There is a injective relationship among membranes and their inside regions. Similarly, a set of inclusions (variables and rules) correspond to the membrane containing this set.

According to this fact, the inclusions are programmed in hardware description language (HDL), instead of the membranes.

Program Translation

There are 2 sides in a program. The left-hand-side is an algebraic expression, involving real number four fundamental arithmetic operations. The right-hand-side involves real number division operation.

For performance reasons, lhs just has multiplication and rhs has only one variable so division is not needed. An example is shown bellow:

$$sval_i * w_i \rightarrow 1 | rw$$

The multiplication operator "$*$" in Verilog, the HDL used to design, is used to perform the multiplications.

Fixed Point Real Number

The operands are real numbers. But, what a pity, FPGA cannot handle real numbers but just integers. A solution for this problem is to transform real numbers to integers by multiplying a radix exponent. For instance,

$$1.3075 \times 10^4 = 13075$$

1.3075 is a decimal real number and the radix of decimal number is 10.This method call “fixed point real number”. In FPGA, numbers are represented in binary. So all the operands should be transformed to binary.

Fixed Point Real Number

Suppose 24-bit registers are assigned to variable. The first bit is the sign bit. The next 10 bits store the integer part and the left 13 bits store the fractional part.

First, perform $1.3075 \times 2^{13} = 10711.04$, take the round number 10711.

Represent10711in binary:

0000_0000_0010_1001_1101_0111

or in hexadecimal:

0029d7

The inaccuracy is 0.04, which is omitted during the transforming. But it is small enough to neglect.

Target FPGA

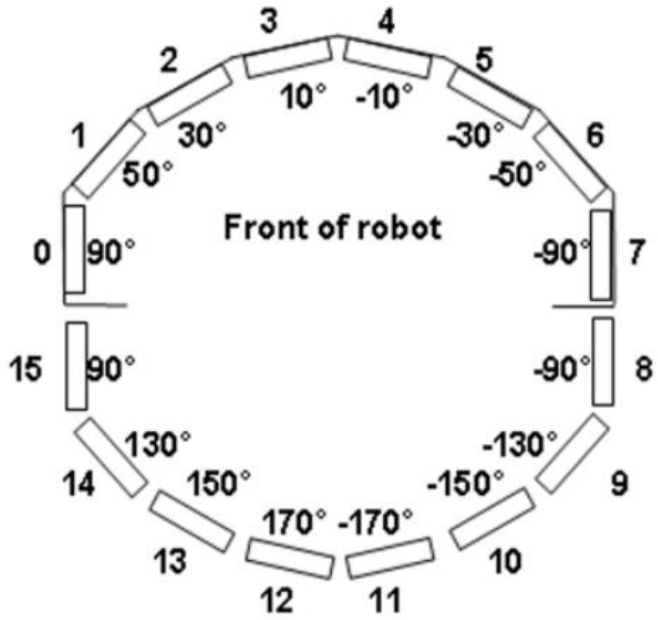
Development board: BASYS 3

On-board FPGA: Xilinx Artix-7 XC7A35T-1CPG236C

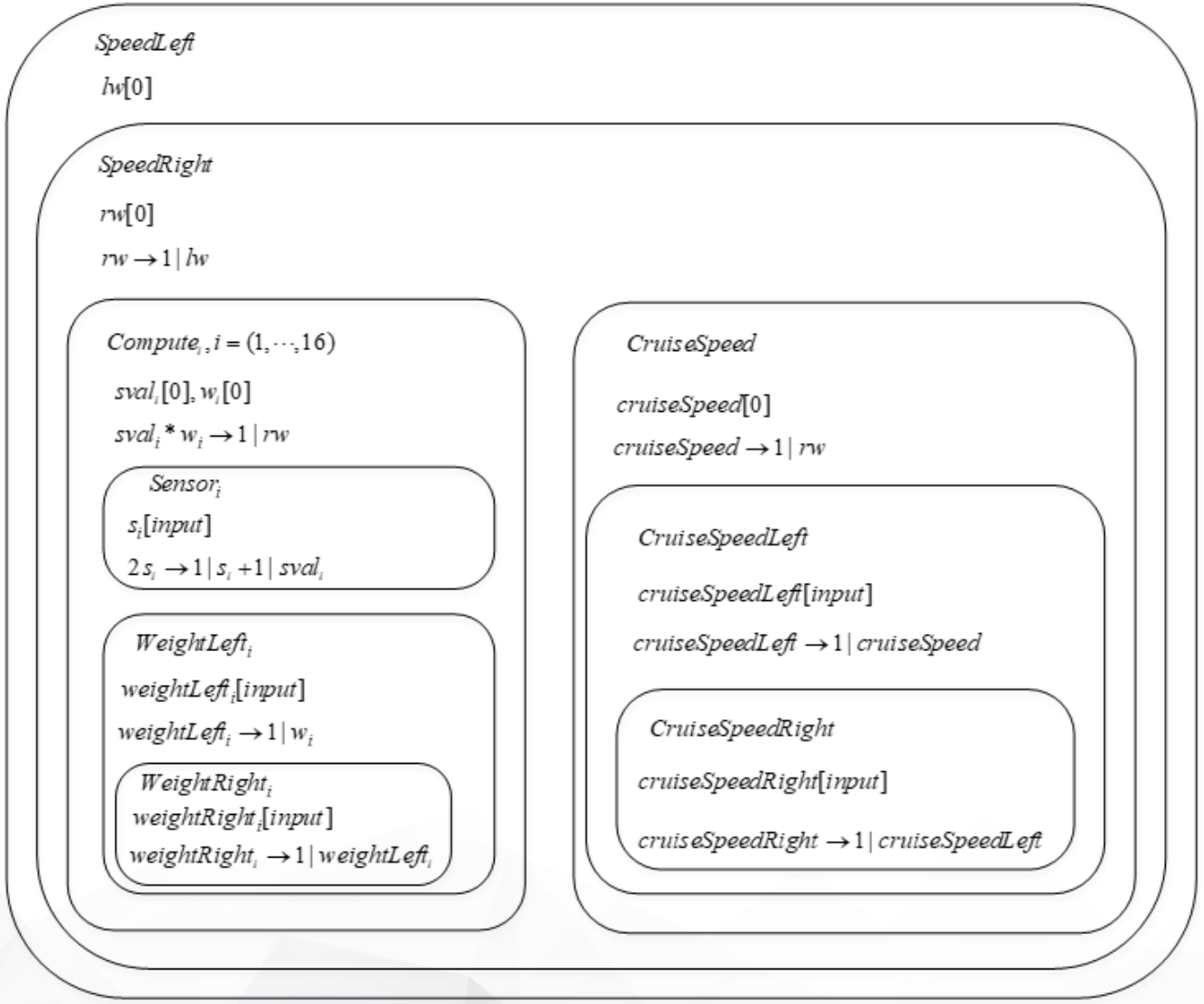
Application

Pioneer 3-DX

16 sonar sensors



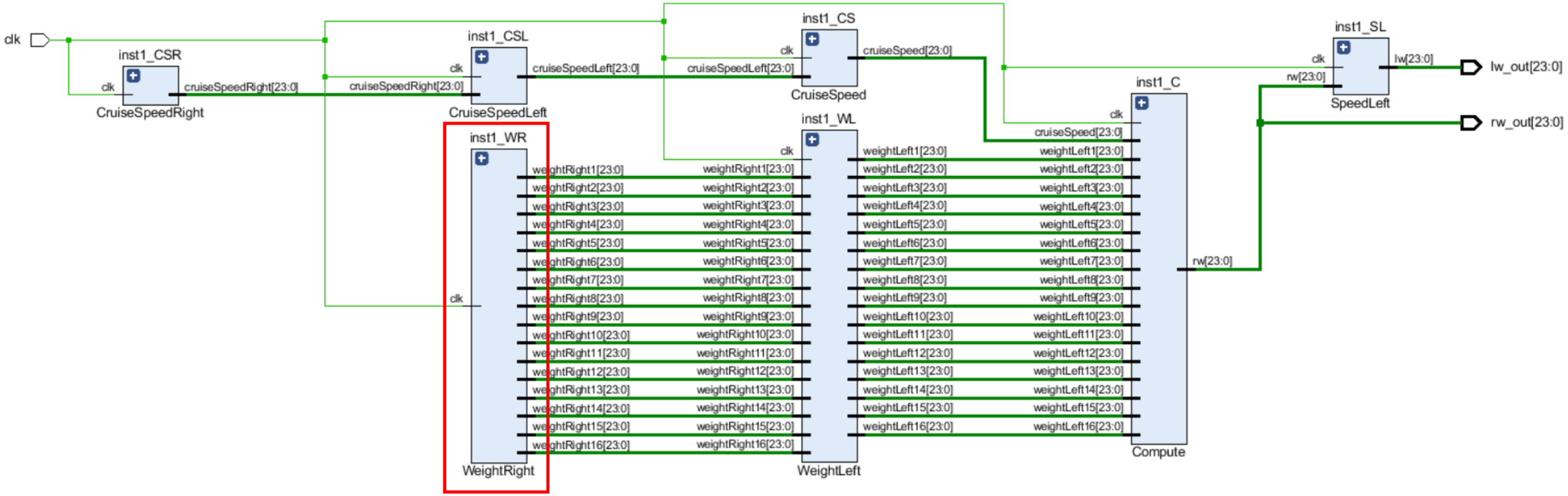
Target NPS (Buiu's algorithm)



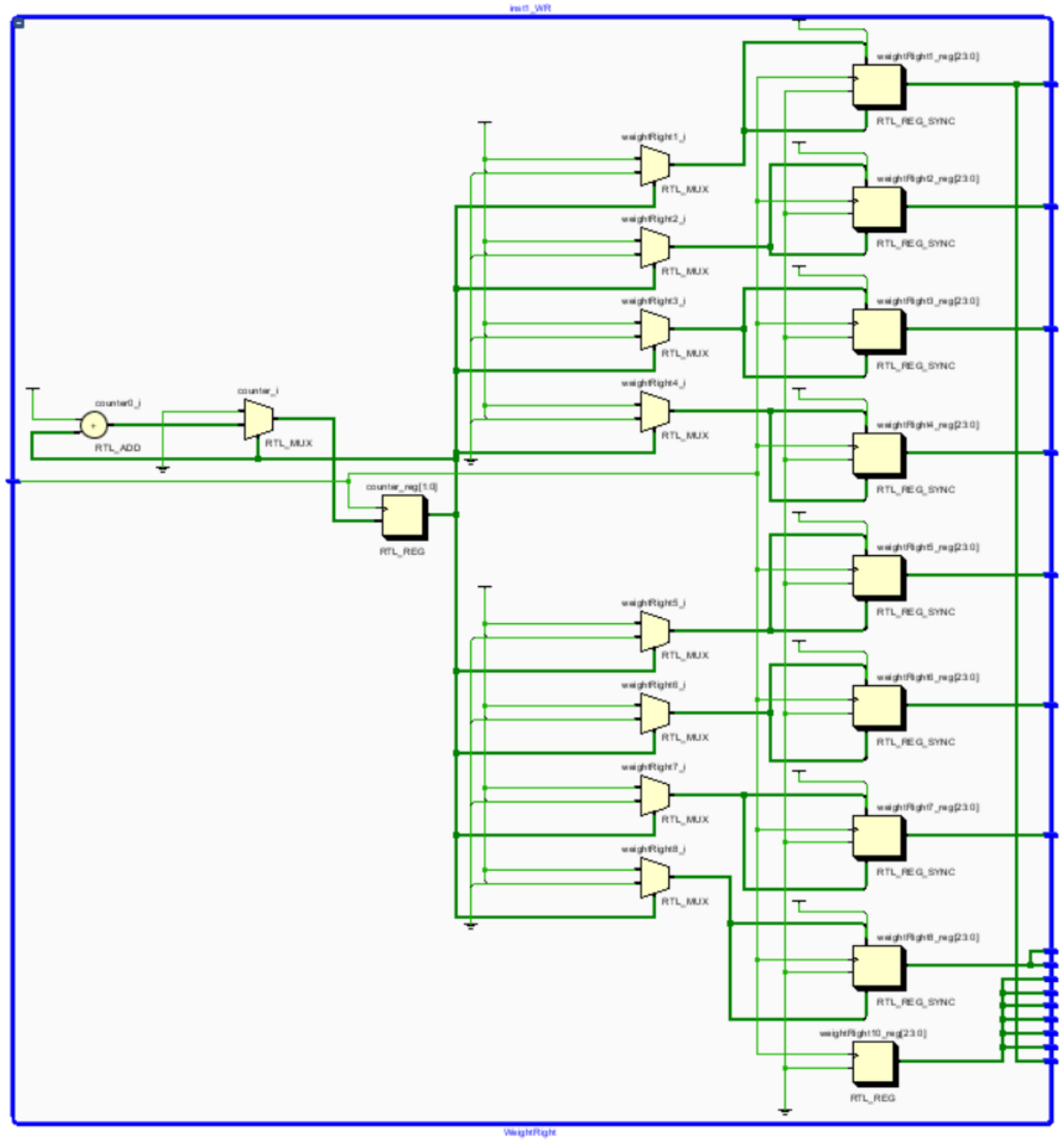
This NPS only computes 3 steps to output the results: the speed of right and left wheel of the robot.

Transplantation

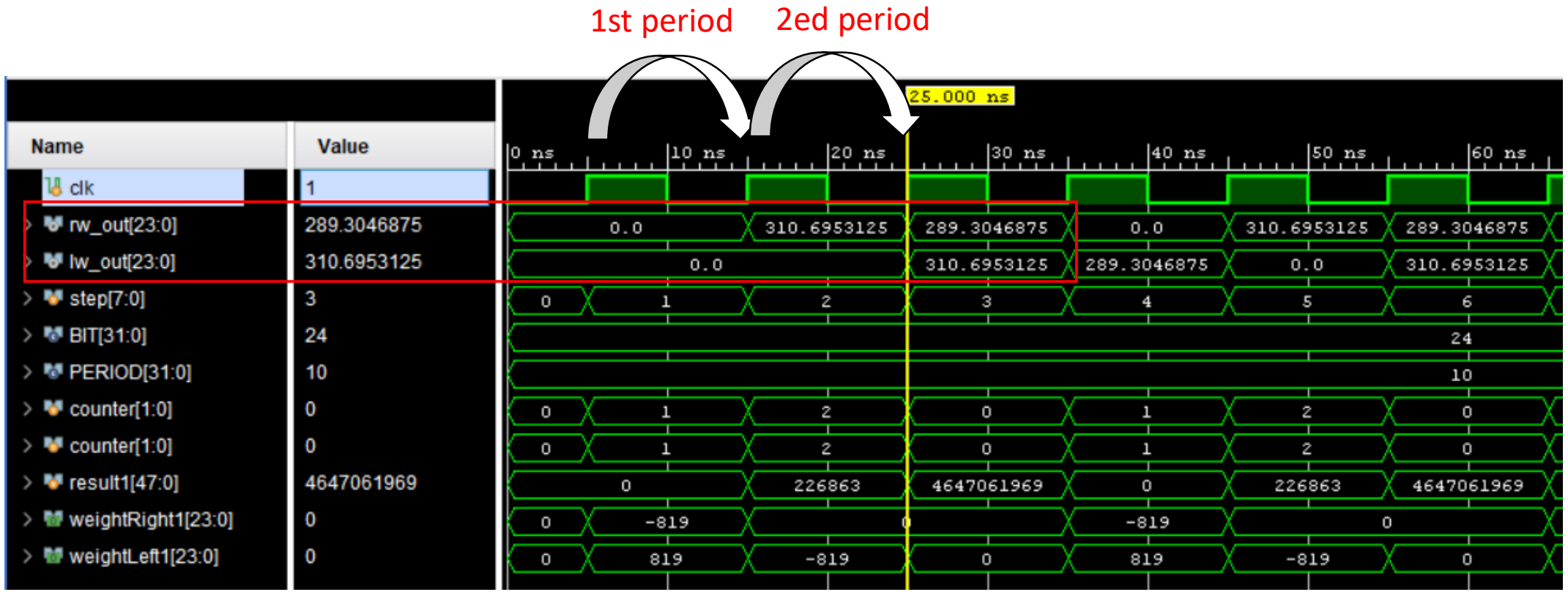
FPGA Design of Target NPS



The details of WeightRight



Performance Analysis of FPGA Implementation



The period of clock is 10 ns and NPS computes at rising edge of clock. The results, the speed of right wheel and left wheel, arise after the 3rd rising edge immediately. So, the FPGA NPS cost 20 ns to get the results.

Software simulation of Target NPS

As a comparison to FPGA implementation, the target NPS is simulated in a NPS simulation soft ware named *pep*, which is developed by Buiu's team. It costs 0.011703 s, which is 1.1703×10^7 ns , to get the results.

Speedup: $(1.1703 \times 10^7) \div 20 = 5.5815 \times 10^5$

```
WARNING:Maximum number of simulation steps exceeded; Simulation stopped
INFO:Simulation finished succesfully after 3 steps and 0.011703 seconds; End state below:
num_ps = {
  SpeedLeft:
    var = { lw: 310.70, }
    E = {}
  SpeedRight:
    var = { rw: 289.30, }
    E = {}
```

Error Analysis

The software simulation results are

$$lw = 310.7, rw = 289.3$$

The FPGA implementing results are

$$lw = 310.6953125, rw = 289.3046875$$

The error of lw and rw

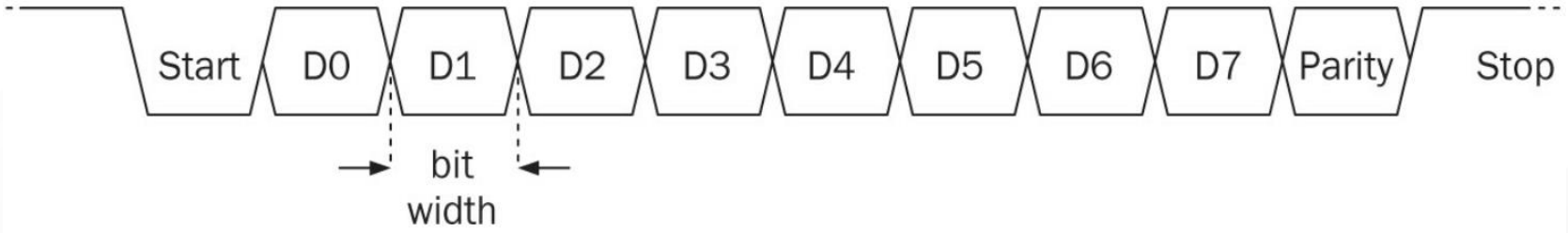
$$e_{lw} = \frac{310.7 - 310.6953125}{310.7} = 1.5087 \times 10^{-5}$$

$$e_{rw} = \frac{289.3046875 - 289.3}{289.3} = 1.6203 \times 10^{-5}$$

UART Communication

The computation results of FPGA can not be observed directly unless hardware debug cores are embedded in the design. Planting debug cores in designs is skillful process, needing specialized knowledge.

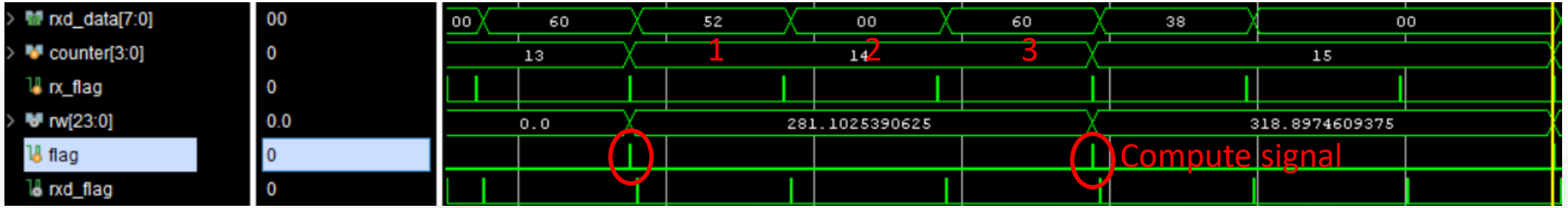
Transmit the results of FPGA to host PC to display is a convenient way to observe the results. But the communication mechanism should be built at first. Universal Asynchronous Receiver/Transmitter (UART) is a widely used digital communication protocol for two or more devices.



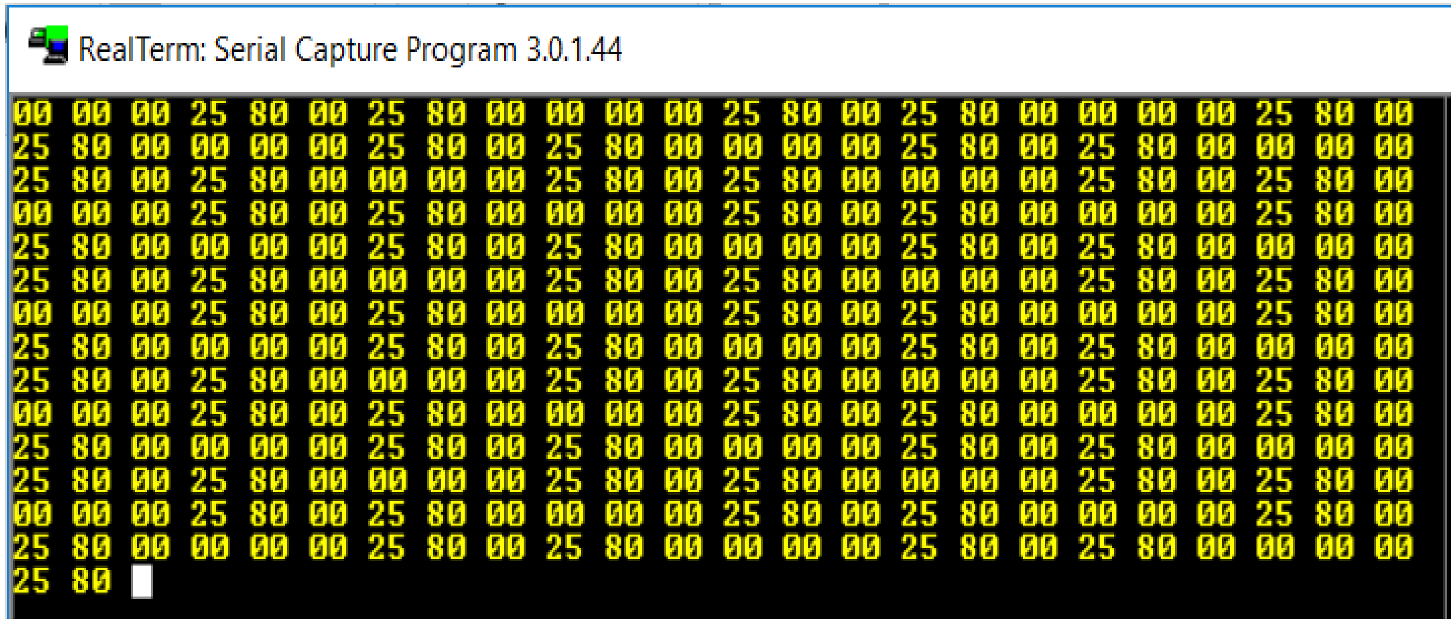
UART Communication

The transmission rate of UART is constant. What transmitted is a data frame, which contains 8 bits at most, not a single bit. Each bit I the frame will span some time, which is called “bit width”.

Each variable in target NPS has 24 bits. This implies that 3 data frames compose a variable. The sensor input includes 48 data frames. NPS will wait a long time to get all the input values. In order to output results faster, NPS computes one step once 3 data frames arrives.



UART Communication

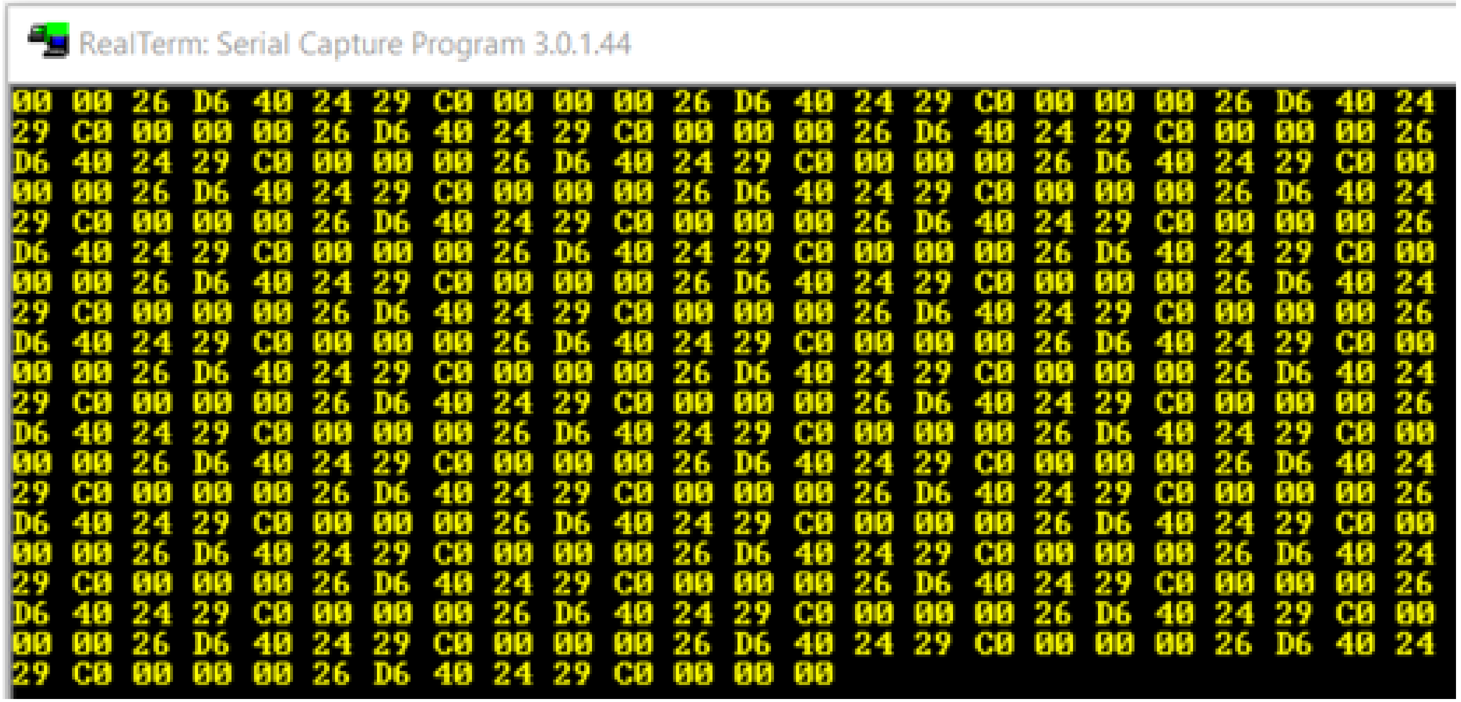


Hexadecimal: 258000

Decimal: 2457600

$$2457600 \div 2^{13} = 300$$

UART Communication



Hexadecimal: 26 D6 40

Decimal: 2545216

$2545216 \div 2^{13} = 310.6953125$

Hexadecimal: 24 29 C0

Decimal: 2369984

$2369984 \div 2^{13} = 289.3046875$

Future Work Plan

1. Translate other algorithms for robot control.
2. Use FPGA to directly control the robot using UART or I/O pins
3. Compare the consumption of hardware resources and power dissipation between NPS and corresponding ENPS.
4. Implement in hardware (E)NPS constructed for other fields.

THANKS