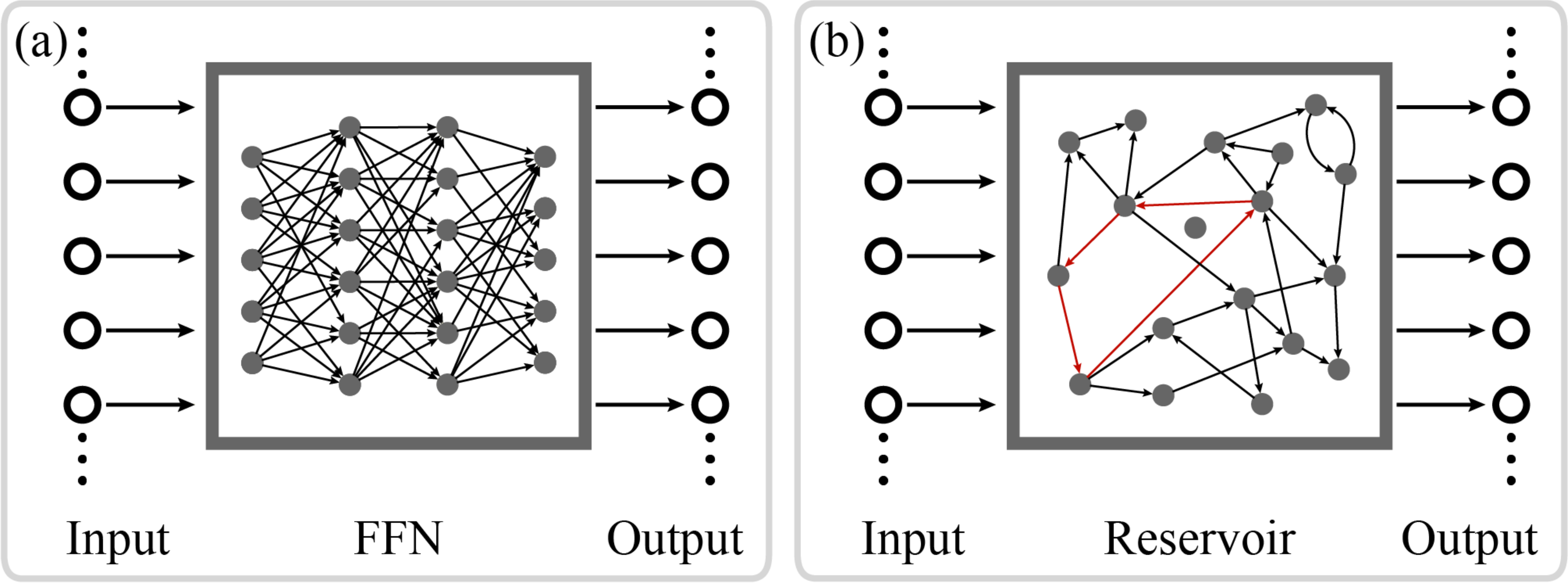
FPGA IMPLEMENTATION OF NGRC FOR PREDICTING DYNAMICAL SYSTEMS

By João Folhadela and Daniel Köglmayr – DLR KI-EXE

2nd FPGA Developers' Forum (FDF) meeting (CERN)

Introduction – Reservoir Computing (RC)



Reservoir Computing

Optimize $\boldsymbol{W}_{out}$ via Ridge regression. Minimize loss:

$$\sum_{t=-T}^{0} \| \boldsymbol{W}_{out}\boldsymbol{r}(t) - \boldsymbol{y}_{\boldsymbol{R}}(\mathrm{t}) \|^2 + \beta \| \boldsymbol{W}_{out} \|^2$$

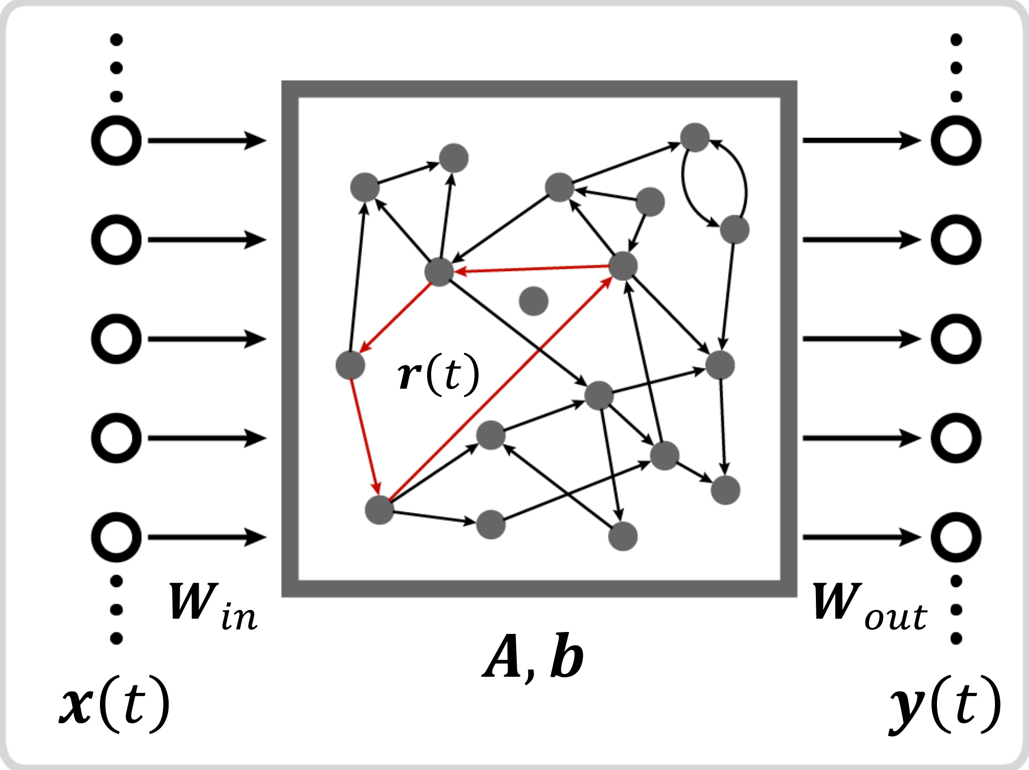
$\boldsymbol{y}_{\boldsymbol{R}}(\mathrm{t})$: known real output

β: regularization parameter

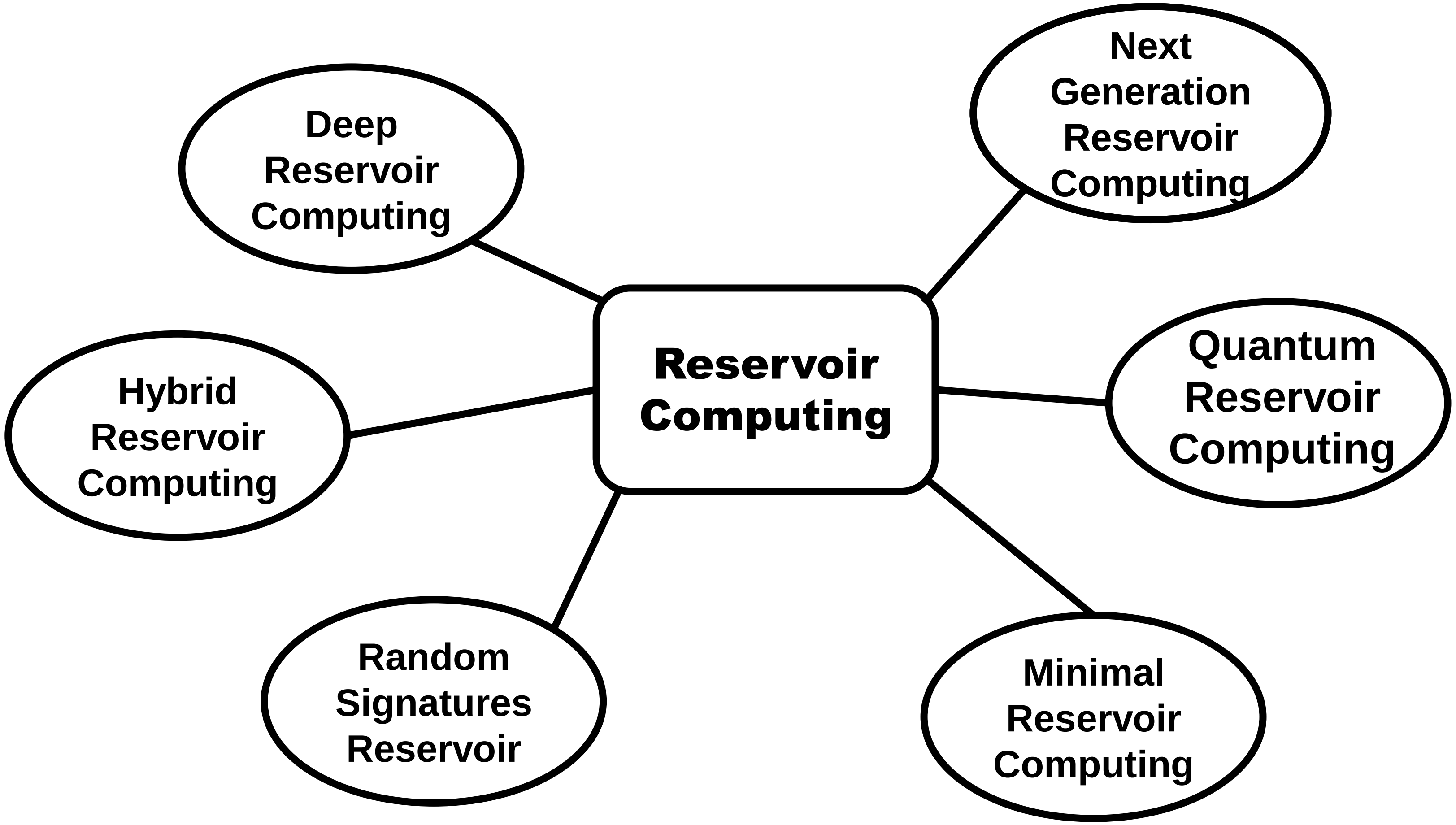
$\boldsymbol{W}_{out}$ is then obtained via:

$$\boldsymbol{W}_{out} = (\boldsymbol{r}^{\intercal}\boldsymbol{r} + \beta\, \boldsymbol{I})^{-1}\, \boldsymbol{r}^{\intercal} y$$

An analytic solution is usually much faster to compute than gradient descent.



RC SOTA



Reservoir Computing

Deep Reservoir Computing

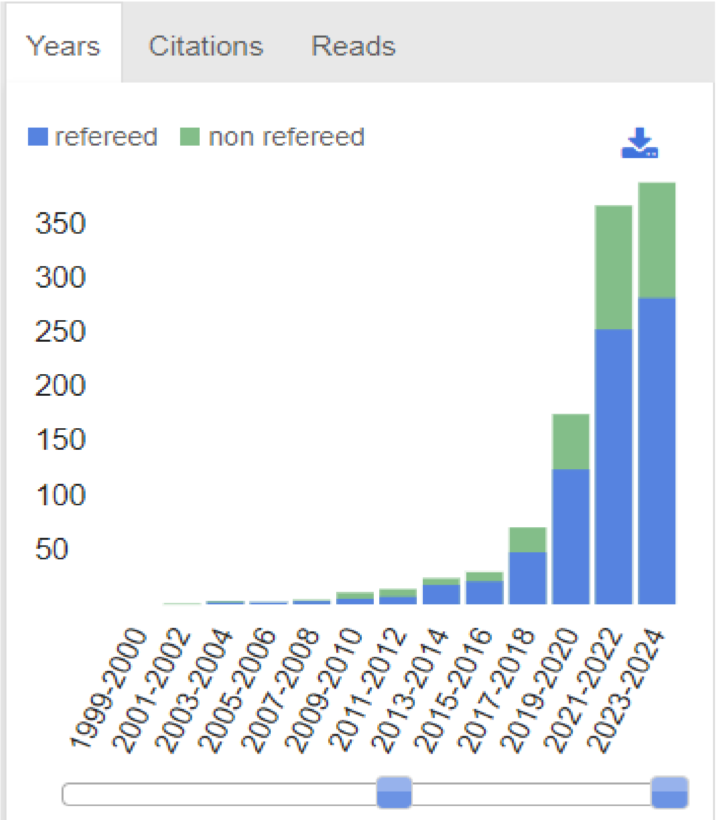
Next Generation Reservoir Computing

Hybrid Reservoir Computing

Quantum Reservoir Computing

Random Signatures Reservoir

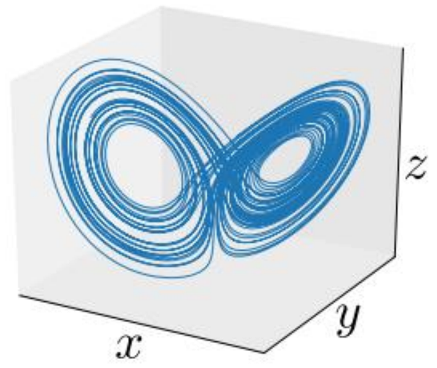
Minimal Reservoir Computing



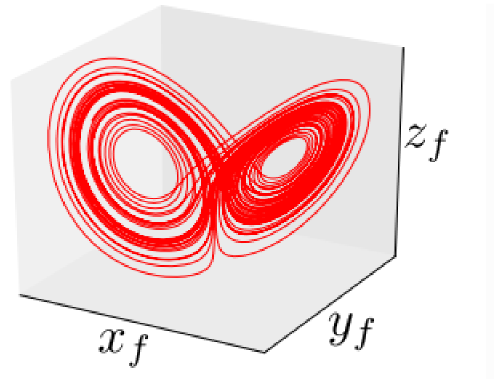
All methods have two basic architectural components:

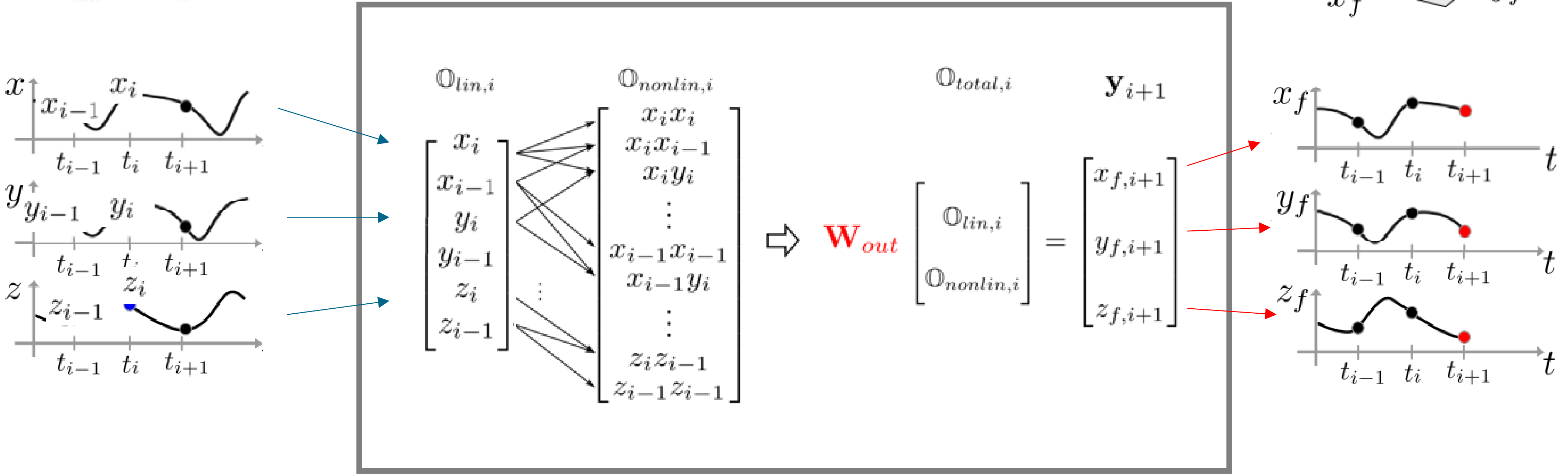
- Nonlinear dimensionality expansion of the input data
- Regression based mapping on the target data

Introduction – Next Generation Reservoir Computing (NGRC)



- Efficient
- Less training data required





$$\begin{bmatrix} x_i \\ x_{i-1} \\ y_i \\ y_{i-1} \\ z_i \\ z_{i-1} \end{bmatrix} \quad \begin{bmatrix} x_i x_i \\ x_i x_{i-1} \\ x_i y_i \\ \vdots \\ x_{i-1} x_{i-1} \\ x_{i-1} y_i \\ \vdots \\ z_i z_{i-1} \\ z_{i-1} z_{i-1} \end{bmatrix} \Rightarrow \mathbf{W}_{out} \begin{bmatrix} \mathbb{O}_{lin,i} \\ \mathbb{O}_{nonlin,i} \end{bmatrix} = \begin{bmatrix} x_{f,i+1} \\ y_{f,i+1} \\ z_{f,i+1} \end{bmatrix}$$

D. Gauthier et al., Nat. Comm., 2021

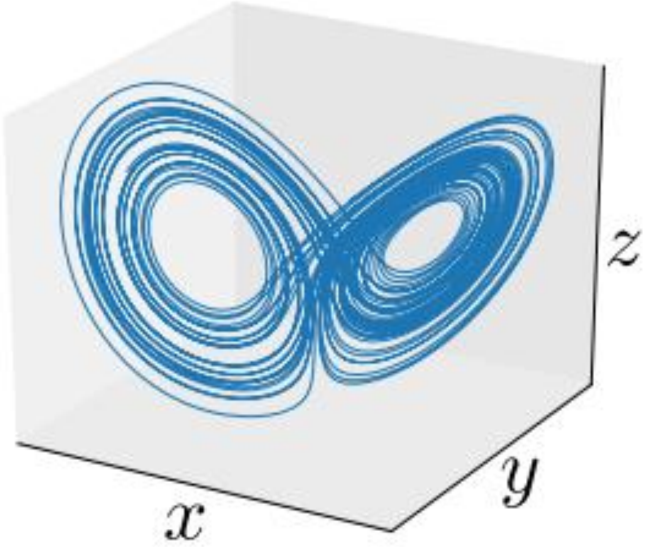
Introduction – NGRC on Lorenz Attractor

- We train with Python‘s numpy‘s linalg solver $\boldsymbol{W}_{out} = (\boldsymbol{r}^{\top}\boldsymbol{r} + \beta\,\boldsymbol{I})^{-1}\,\boldsymbol{r}^{\top} Y_{train}$
- Some parameters:

Train Steps	k	β	$\boldsymbol{W}_{out}$ shape
400	2	9×10^{-2}	27x3

k – timesteps β - regularization parameter

- Then inference is performed on FPGA
    - Transfer $\boldsymbol{W}_{out}$ and first linear state to FPGA memory
    - Prediction loop



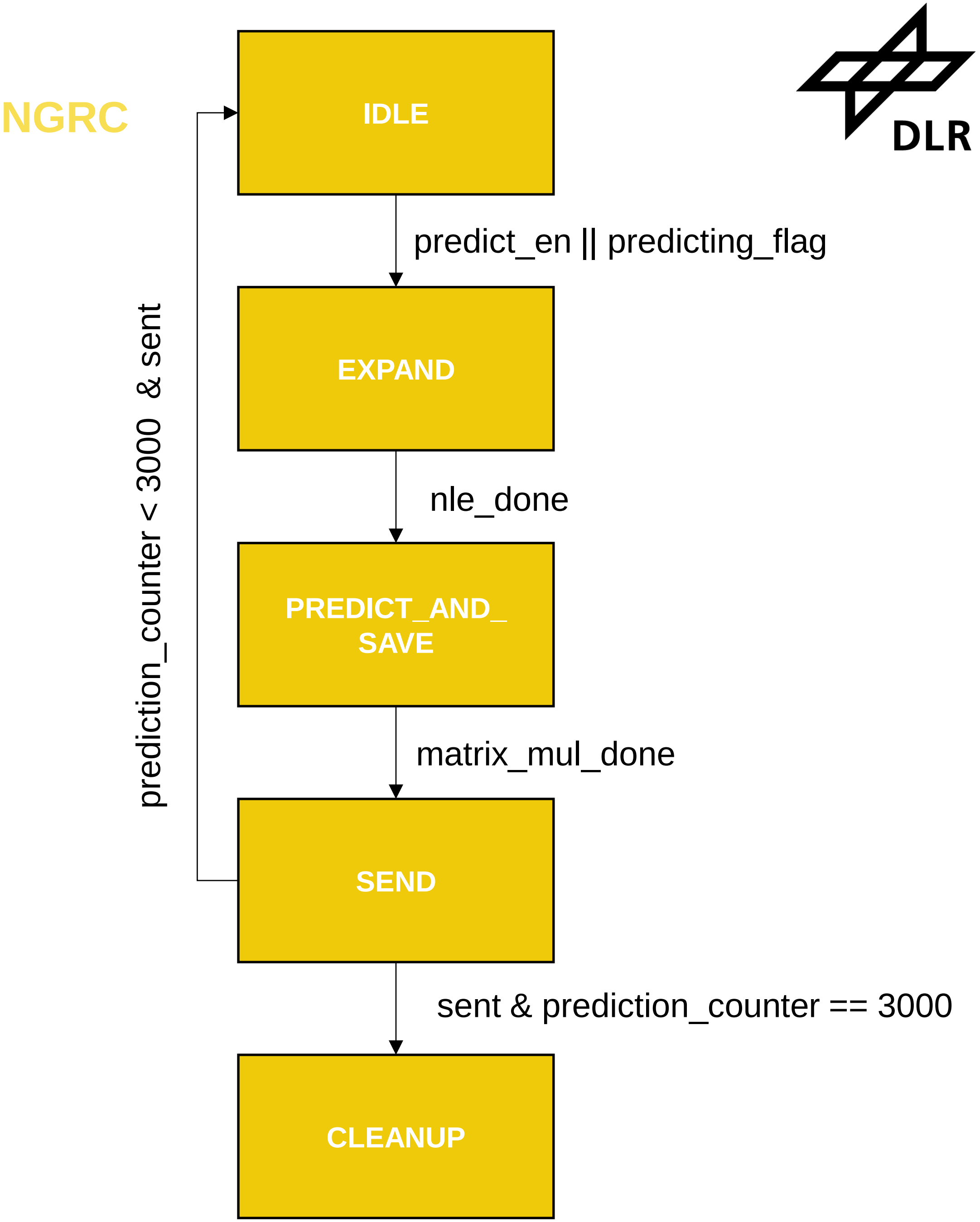
Implementation – The NGRC prediction loop

1. Creating the Reservoir
   - Read linear state from memory – current and previous timestep
   - Non-linear expansion – multiplying and powering the linear state
2. Predicting
   - Matrix multiplication, several vector dot products – Multiply and Accumulate (MACs)
3. Save prediction

In red – arithmetics are performed in float 32 with Vivado floating-point IP

Implementation – The FSM

- IDLE
  - Sets linear state
  - Waits for prediction loop to begin/continue
- EXPAND
  - Performs the non-linear expansion
- PREDICT_AND_SAVE
  - Calculates predicton
  - Saves it
- SEND
  - Send the current prediction via UART
- CLEANUP
  - Resets all counters and variables


NGRC
IDLE
predict_en || predicting_flag
EXPAND
nle_done
PREDICT_AND_SAVE
matrix_mul_done
SEND
prediction_counter < 3000 & sent
sent & prediction_counter == 3000
CLEANUP

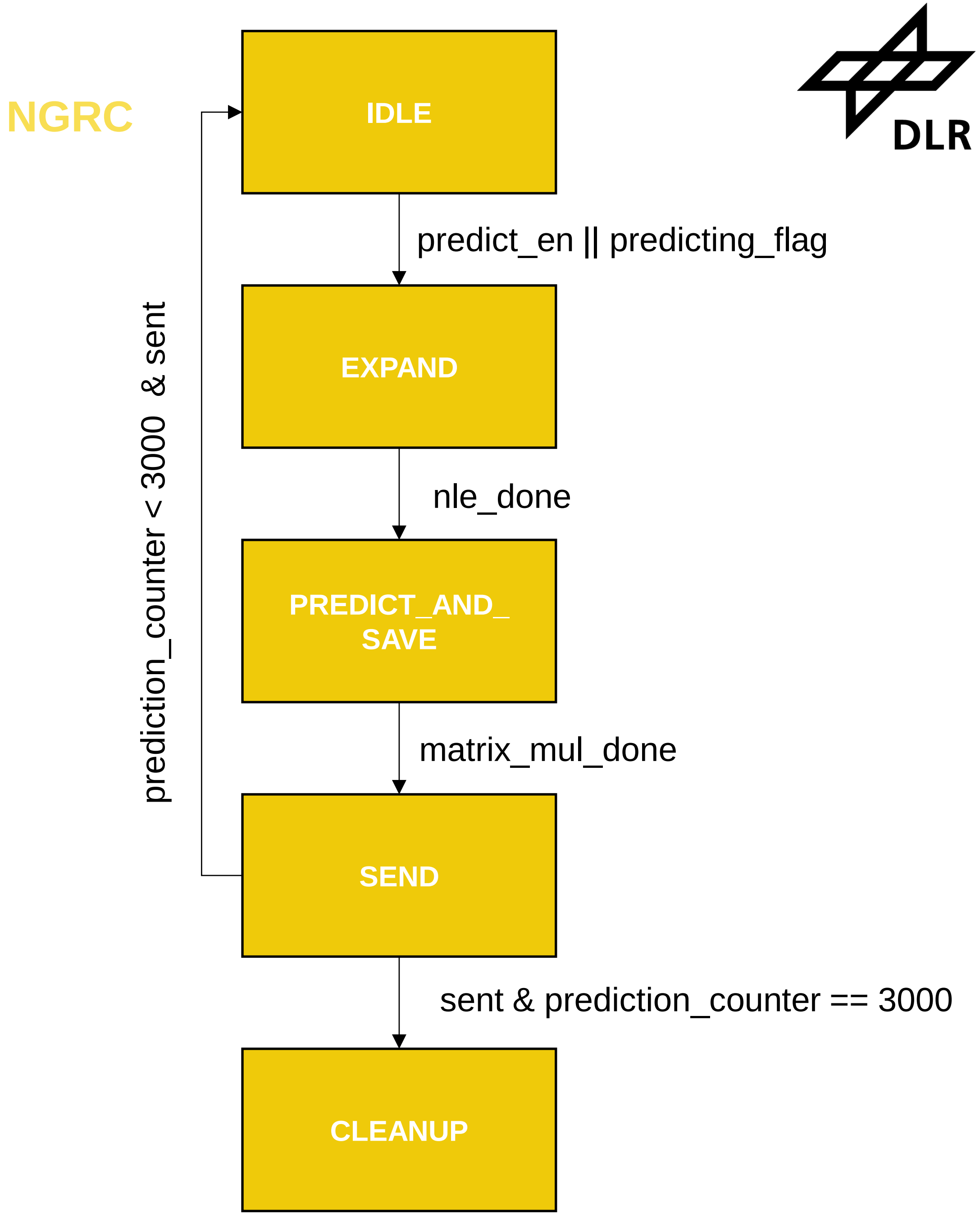
Implementation – The FSM

EXPAND

- Implemented as a module that expands a vector of size 6 into size 27 (6+21)
- Each multiplication is computed in parallel, with 32-bit float numbers:
    - Vivado Floating-point IP
        - *Multiply* operation
        - *Full DSP usage* (21 multiplications x 2 DSP = 42 DSPs)
        - Maximum Latency used (8 clock cycles)
- Concatenate the input vector with each of the multiplication results.

Implementation – The FSM

- IDLE
  - Sets linear state
  - Waits for prediction loop to begin/continue
- EXPAND
  - Performs the non-linear expansion
- PREDICT_AND_SAVE
  - Calculates predicton
  - Saves it
- SEND
  - Send the current prediction via UART
- CLEANUP
  - Resets all counters and variables


NGRC
IDLE
predict_en || predicting_flag
EXPAND
nle_done
PREDICT_AND_SAVE
matrix_mul_done
SEND
sent & prediction_counter == 3000
CLEANUP
prediction_counter < 3000 & sent

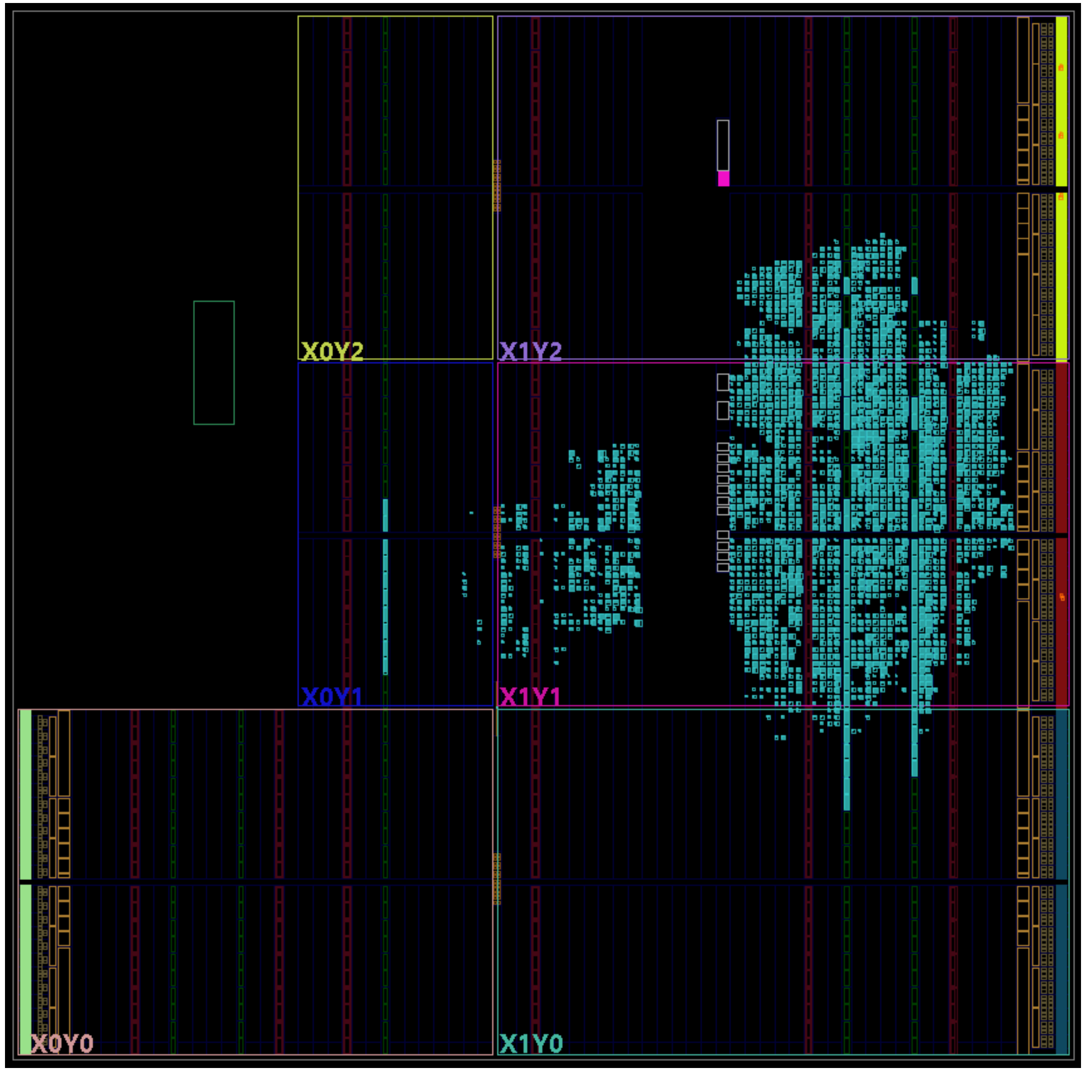
Implementation – The FSM

PREDICT_AND_ SAVE

- Perform matrix multiplication of the reservoir state (a vector) with the weights, $\boldsymbol{W}_{out}$:
  - 3 MACs implemented in parallel with Vivado Floating-point IP that compute:
    - *Fused Multiply-Add* operation
    - *Full DSP usage* (3 MACs x 4 DSP = 12 DSPs)
    - 10 clock cycles latency used
  - 10 latency x 27 accumulations = 270 clock cycles in total
- Save prediction (to update next linear state)

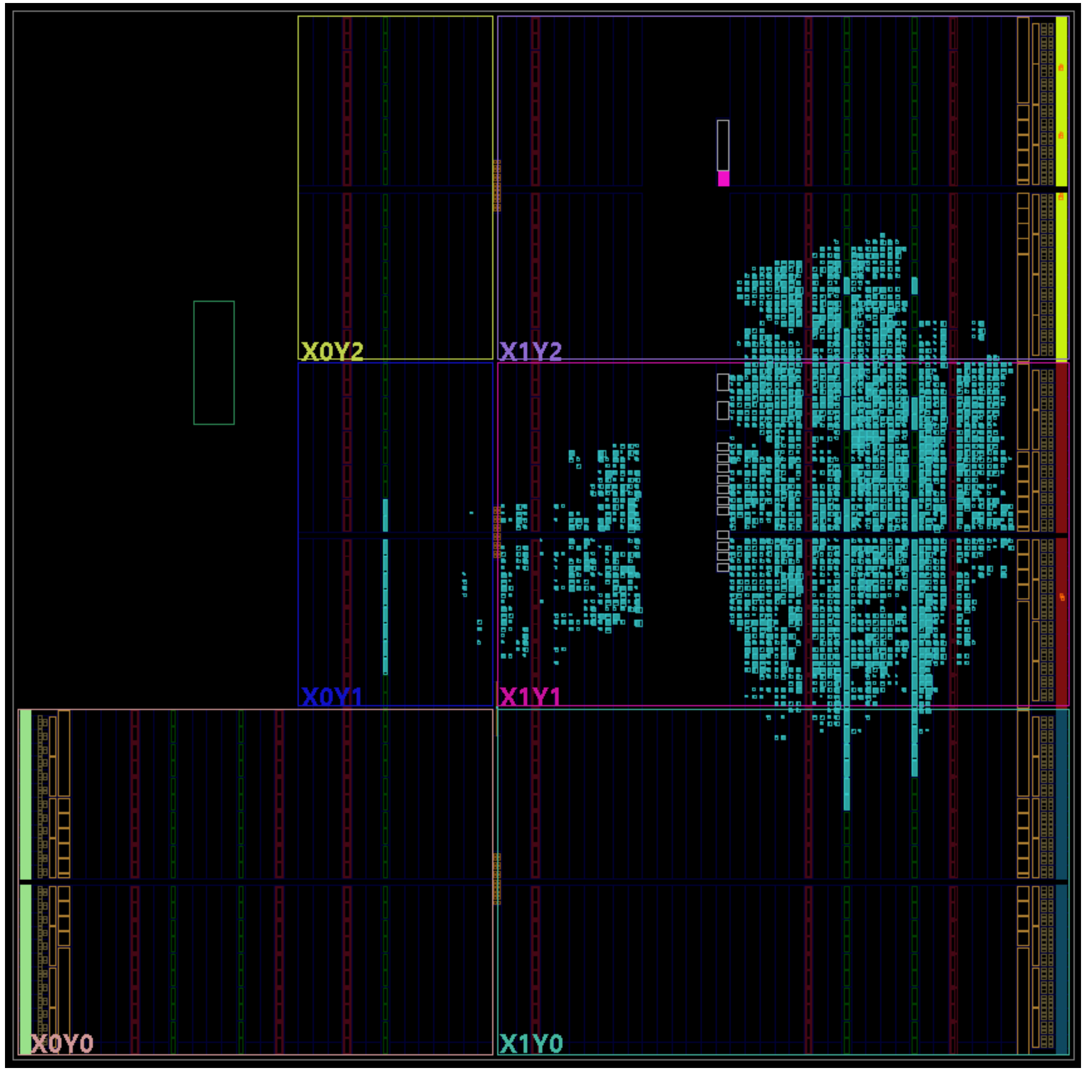
Resource usage

Resource	Used / Available
LUT	5,132 / 53,200
LUTRAM	270 / 17,400
FF	6,475 / 106,400
DSP	54 / 220



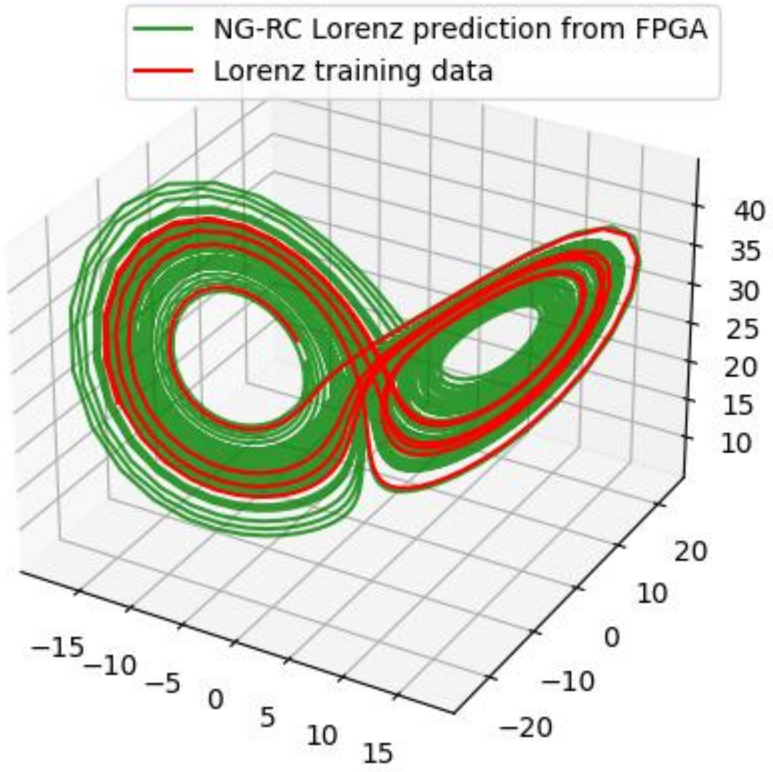
Other reports

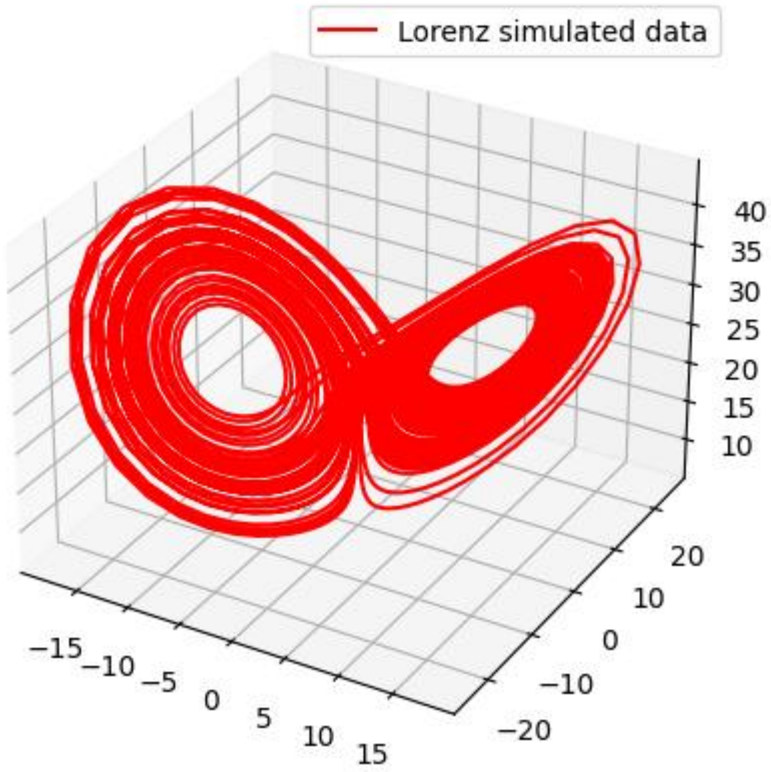
Property	Value
On-Chip Power	0.294 W
Dynamic	0.186 W
Static	0.109 W
Clock Speed	125 MHz
Worst Negative Slack	1.3 ns



Evaluation – Simulated vs. Predicted Lorenz

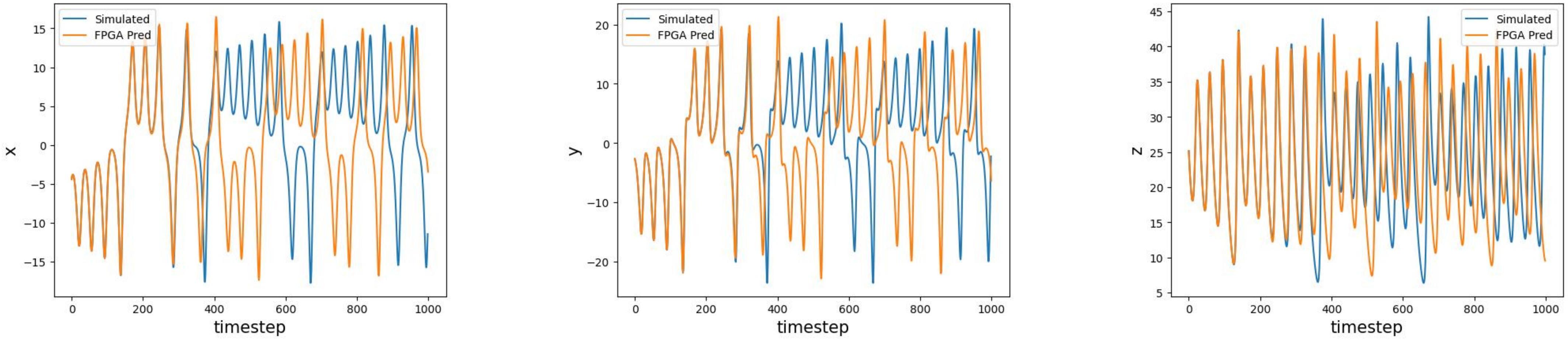
Prediction over 3000 timesteps



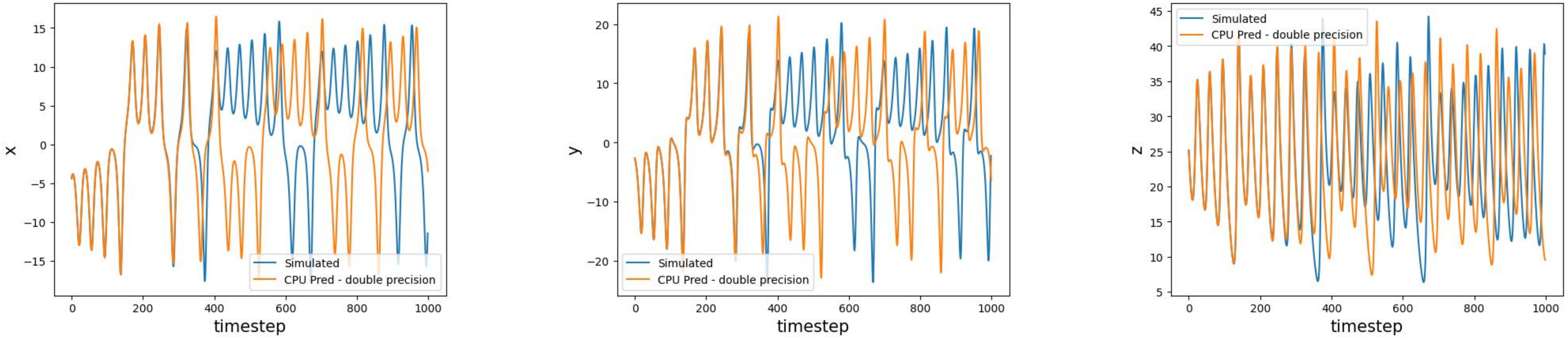


Evaluation – Simulated vs. Predicted Lorenz

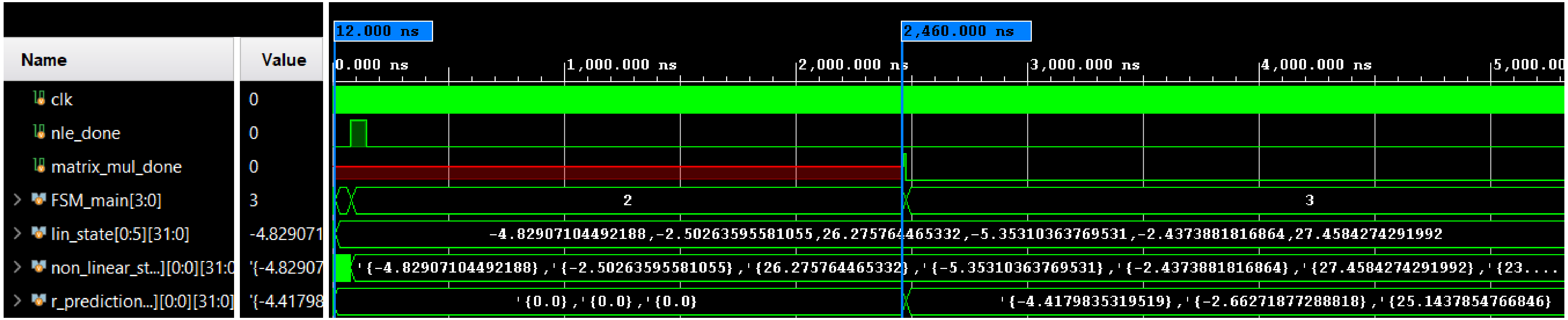
On FPGA



On CPU – with double precision



Latency Evaluation – Simulated vs. Predicted Lorenz



	FPGA	CPU
Device	Zynq7000-20	i7-1365U 13th Gen
Clock Speed	125 MHz	4.8 GHz
Inference time	2.5 µs	50 µs

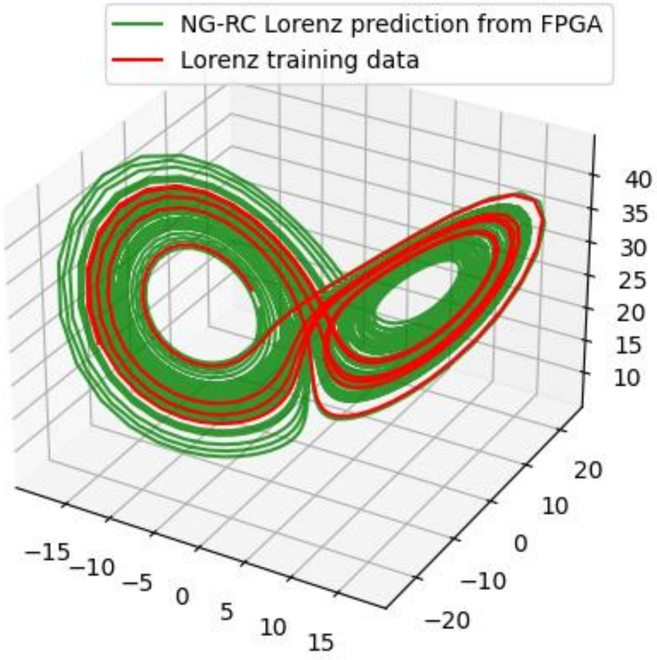
Evaluation – Predicting the Lorenz attractor

- Easy to implement

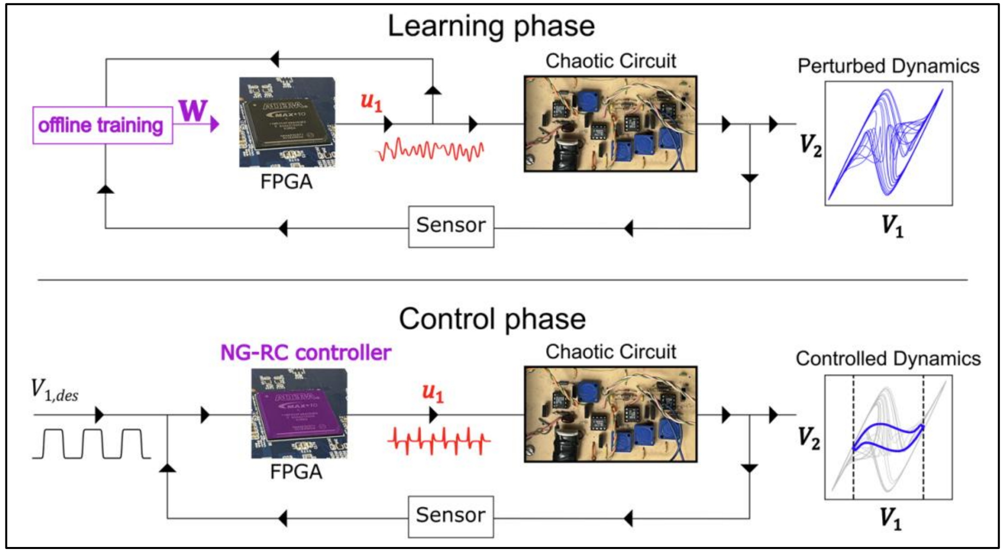
- Low power

- Fast

And real applications?



Applications of NGRC



- Control of chaotic circuit dynamics

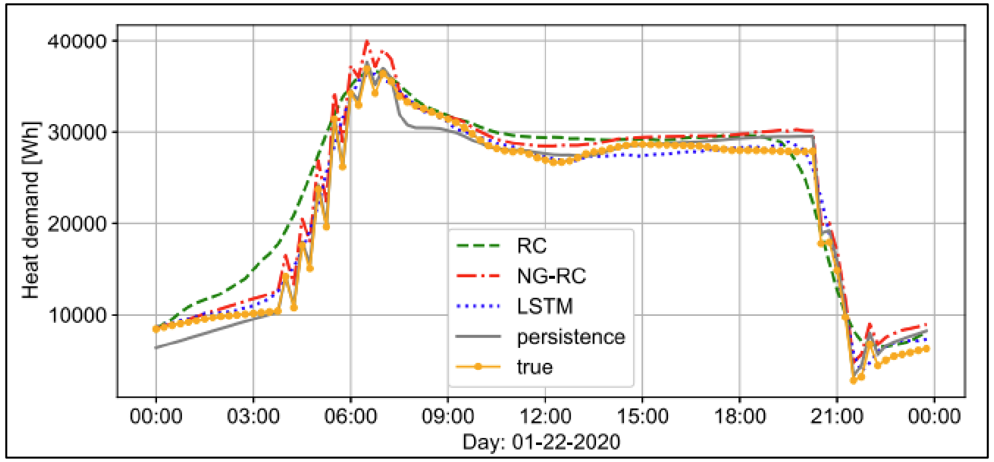
Kent, R.M. et. al *Controlling chaos using edge computing hardware*

Nat Commun 15, 3886 (2024)

- Energy demand forecasting

Karoline Brucke et. al *Benchmarking reservoir computing for residential energy demand forecasting*

Energy and Buildings 314 (2024)



- Financial markets, weather, costal line erosion, robotic control, etc.

THANK YOU!

Questions?