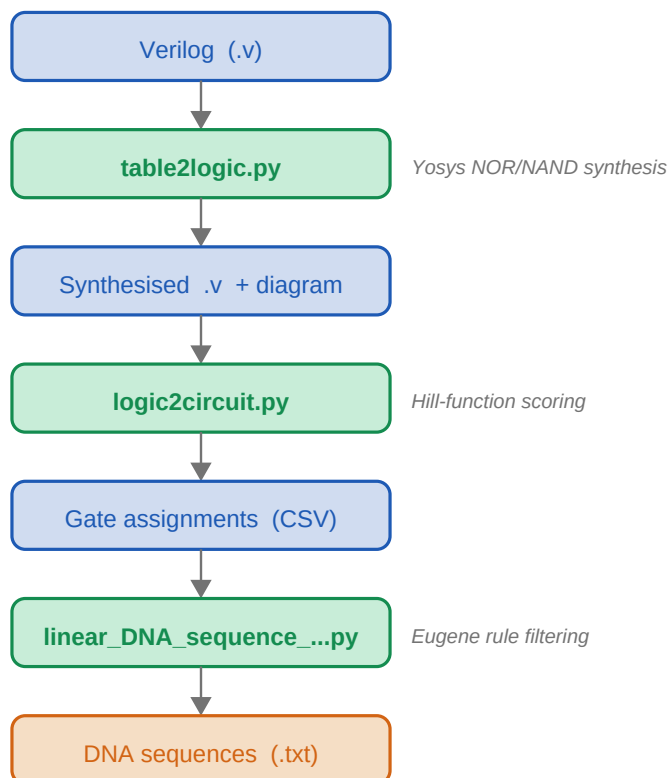


SynCAD

Synthetic Circuit Automated Design

User Guide & Pipeline Reference

An end-to-end automated pipeline converting Boolean logic specifications into physically constructible DNA sequences for expression in *E. coli*.



Author: Euijin Seo ufoooo1391@postech.ac.kr

Date: 5■ 21, 2026

Contents

1. Overview
2. Prerequisites & Installation
3. Quick Start
4. Step 1 — Logic Synthesis (`table2logic.py`)
5. Step 2 — Biological Gate Assignment (`logic2circuit.py`)
6. Step 3 — DNA Sequence Generation (`linear_DNA_sequence_from_circuits.py`)
7. Graphical User Interface
8. Worked Example
9. Biological Background
10. File Format Reference
- A. Appendix — Complete Gate Parameter Table

1. Overview

SynCAD (Synthetic Circuit Automated Design) converts abstract Boolean logic into a constructible DNA sequence. The three-stage pipeline is:

- **Logic synthesis** — maps a Verilog design onto NOR/NAND gates via Yosys.
- **Biological gate assignment** — scores all valid repressor-gate assignments using Hill-function kinetics and ranks by ON/OFF ratio.
- **DNA sequence generation** — assembles promoter + gene-body sequences, filtered by Eugene compositional design rules.

A Streamlit GUI (`synCAD_ver1.py`) integrates all three steps. Each can also run standalone from the command line.

2. Prerequisites & Installation

Required software

- Python ≥ 3.9
- Yosys (must be on `$PATH`)
- Python packages: streamlit, pandas, tqdm, networkx, matplotlib, openpyxl

Install Python packages

```
pip install streamlit pandas tqdm networkx matplotlib openpyxl
```

Verify Yosys

```
yosys --version # e.g. Yosys 0.38
```

Project layout

```
SynCAD_hompage/  
  table2logic.py           # Step 1  
  logic2circuit.py         # Step 2  
  linear_DNA_sequence_from_circuits.py # Step 3  
  synCAD_ver1.py           # Streamlit GUI  
  genetic_part_sequences.xlsx # built-in parts database
```

3. Quick Start

GUI (recommended)

```
cd SynCAD_hompage
streamlit run synCAD_ver1.py
# Open http://localhost:8501
```

Upload a Verilog file, choose gate types, set input signal levels, then click **Run SynCAD**.

Command-line (all three steps)

```
# Step 1 – synthesise
python3 table2logic.py --input my_circuit.v --gate NOR

# Step 2 – assign biological gates
python3 logic2circuit.py \
    -v output_my_circuit_NOR.v \
    -i in1:0.01,1.0 in2:0.01,1.0 \
    -k 3 --save-csv results.csv

# Step 3 – generate DNA sequences
python3 linear_DNA_sequence_from_circuits.py \
    --graph circuit_assigned.pkl \
    --parts genetic_part_sequences.xlsx \
    --gate-rule Eugene_device_rule.txt \
    --device-rule Eugene_part_rule.txt \
    --output DNA_sequence_result.txt
```

4. Step 1 — Logic Synthesis (table2logic.py)

What it does

- Parses and flattens the Verilog design hierarchy.
- Optimises the Boolean network.
- Maps all logic to NOR, NAND, or both.
- Writes synthesised Verilog and exports a PNG circuit diagram.

Input Verilog format

A module with input/output ports and assign statements. Top-level module must be named `example`.

```
module example(input a, input b, output out);  
    assign out = ~(a | b);  
endmodule
```

CLI reference

```
python3 table2logic.py --input <FILE.v> [--gate NOR] [--gate NAND]
```

Flag	Default	Description
--input / -i	(required)	Path to input Verilog file
--gate / -g	NOR	Gate type(s): NOR, NAND, OR, AND

Outputs

- `output_{stem}_{GATES}.v` — synthesised Verilog
- `{stem}_{GATES}.png` — circuit diagram

5. Step 2 — Biological Gate Assignment (logic2circuit.py)

What it does

Maps each logic-gate node to a biological repressor characterised by Hill-function kinetics. Exhaustively enumerates valid assignments (no duplicate gene family) and ranks them by ON/OFF ratio.

CLI reference

```
python3 logic2circuit.py \
  -v <synthesised.v> \
  -i name:low,high [name:low,high ...] \
  [-k 3] [--quiet] [--save-csv <path>]
```

Flag	Default	Description
--verilog / -v	(required)	Synthesised Verilog from Step 1
--inputs / -i	(required)	Space-separated name:low,high pairs
--topk / -k	3	Number of top results to show
--quiet	off	Suppress tqdm progress bar
--save-csv	none	Save results table to CSV

Input level specification

```
-i in1:0.0042,2.0082 in2:0.0022,5.0543
#      ^^^^ low  ^^^^^ high (in REU)
```

Output CSV format (when --save-csv is used)

rank	score	_0_	_1_
1	0.8532	AmeR_F1	BetI_E1
2	0.7214	PhIF_P2	SrpR_S2
3	0.6891	QacR_Q1	LmrA_N1

Note: Column names (_0_, _1_, ...) match the internal wire names in the synthesised Verilog.

6. Step 3 — DNA Sequence Generation

`linear_DNA_sequence_from_circuits.py`

What it does

- Determines valid transcriptional unit (TU) orderings from the circuit graph.
- Filters orderings via Eugene BEFORE compositional design rules.
- Assembles promoter + gene-body sequences from the parts database.
- Writes all valid linear DNA sequences to a text file.

Input files

circuit_assigned.pkl — NetworkX DiGraph (Python pickle). Internal nodes must carry 'RBS' and 'Repressor' node attributes. Input nodes have in-degree 0; the output node has out-degree 0.

genetic_part_sequences.xlsx — parts database (3 sheets):

Sheet	Contents
gate	Promoter + gene-body sequences for each repressor
input	Promoter sequences (pBAD, pTet, pTac, ...)
output	Reporter sequences (YFP, ...)

Eugene rule files — *_Eugene_device_rule.txt and *_Eugene_part_rule.txt

CLI reference

```
python3 linear_DNA_sequence_from_circuits.py \
  --graph      circuit_assigned.pkl \
  --parts      genetic_part_sequences.xlsx \
  --gate-rule  Eugene_device_rule.txt \
  --device-rule Eugene_part_rule.txt \
  [--output DNA_sequence_result.txt]
```

Flag	Default	Description
--graph	(required)	Gate-assigned NetworkX pickle
--parts	(required)	Genetic parts Excel workbook
--gate-rule	(required)	Eugene gate rule file
--device-rule	(required)	Eugene device rule file
--output	DNA_sequence_result.txt	Output path

Output format

```
##### Genetic circuit sequence #####
```

```
1.
```

```
Logic : //__ pBAD / pTet__pA1__AmtR ... //  
ATGCGTAAGTCG...
```

```
Output : //__ pAmtR__YFP //  
ATGCGT...
```

7. Graphical User Interface

Launch

```
streamlit run synCAD_ver1.py
# -> http://localhost:8501
```

Workflow

Section 1 — Input design

- Upload a .v Verilog file.
- Select gate types (NOR, NAND, or both).
- Set low/high REU values for each detected input signal.

Section 2 — Run Steps 1 & 2

Click **Run SynCAD**. On completion: logic diagram is displayed, synthesised Verilog and gate-assignment CSV are available for download.

Section 3 — DNA Sequence Generation (optional)

Upload the gate-assigned .pkl circuit graph and both Eugene rule files; a custom parts Excel is optional (the built-in file is used by default). Click **Generate DNA Sequences**.

Troubleshooting

Message	Likely cause & fix
Logic synthesis failed	Yosys not on \$PATH, or top module != "example". Check: yosys --version
Gate assignment: no results	Input low == high, or too many gates for available gene families.
DNA generation failed	.pkl nodes missing RBS/Repressor attrs, or unsupported input count (need 3–4 inputs, 1 output).

8. Worked Example

Save the following as `two_nor.v`:

```
module example(input a, input b, output out);
    assign out = ~(a | b);
endmodule
```

Step 1 — Synthesis

```
python3 table2logic.py --input two_nor.v --gate NOR
# -> output_two_nor_NOR.v
# -> two_nor_NOR.png
```

Step 2 — Gate assignment

```
python3 logic2circuit.py \
    -v output_two_nor_NOR.v \
    -i a:0.01,1.0 b:0.01,1.0 \
    -k 5 --save-csv results.csv
```

Expected output:

```
Searching gate combos: 100%|#####| 420/420 [00:03]
[INFO] Results saved -> results.csv

[1] Score = 0.8532
    _0_ -> AmeR_F1    _1_ -> BetI_E1
    ON/OFF ratio: 0.8532
```

9. Biological Background

Hill-function kinetics

Each repressor gate is modelled by a Hill equation:

```
out = y_min + (y_max - y_min) / (1 + ([in] / K)^n)

[in]  input protein concentration  (REU)
K     repression threshold          (REU)
n     Hill coefficient (cooperativity)
y_min / y_max  expression range    (REU)
```

ON/OFF ratio score

Circuit performance is the worst-case ON/OFF ratio across all outputs:

```
score = min_over_outputs(
    min(out | boolean == TRUE) /
    max(out | boolean == FALSE)
)
```

Higher score = better separation between logic-HIGH and logic-LOW, reducing biological read-out error.

Gene family constraint

No two gate nodes in the same circuit may share a gene family (e.g. two PhIF variants cannot coexist). This prevents transcriptional crosstalk.

10. File Format Reference

File	Format	Description
*.v (input)	Verilog	Boolean logic; top module = example
output_*.v	Verilog	Synthesised gate-level netlist
*.png	PNG	Circuit diagram (Yosys show)
gate_assignment_results.csv	CSV	Ranked gate assignments + scores
genetic_part_sequences.xlsx	Excel	Parts database (gate/input/output sheets)
*_Eugene_*.txt	Text	Eugene BEFORE ordering rules
*_assigned.pkl	Pickle	NetworkX DiGraph with RBS/Repressor attrs
DNA_sequence_result.txt	Text	Final linear DNA sequences

Appendix A — Complete Gate Parameter Table

Hill-function parameters for all 21 built-in biological gates (Cello v2 Eco1C1G1T1 dataset). REU = Relative Expression Unit.

Gate name	y_min	y_max	K	n
AmeR_F1	1.060	3.835	0.122	1.306
AmtR_A1	0.091	5.036	0.022	1.308
BetI_E1	6.5e-16	3.065	0.363	1.833
HlyIIR_H1	5.6e-22	2.420	0.094	1.537
IcaRA_I1	5.2e-14	3.558	0.186	1.085
LitR_L1	1.2e-20	5.833	0.017	0.689
LmrA_N1	0.145	2.045	0.090	1.397
BM3R1_B1	0.002	0.543	0.051	2.042
BM3R1_B2	7.3e-22	0.822	0.285	1.685
BM3R1_B3	1.4e-21	0.704	0.261	1.768
PhIF_P1	2.8e-23	6.814	0.013	2.160
PhIF_P2	4.0e-16	6.505	0.160	2.237
PhIF_P3	4.9e-19	6.794	0.167	2.010
PsrA_R1	2.1e-15	6.489	0.305	1.540
QacR_Q1	7.2e-23	3.744	0.126	1.706
QacR_Q2	3.0e-23	3.778	0.456	1.979
SrpR_S1	7.3e-16	1.737	0.009	1.186
SrpR_S2	5.2e-20	2.556	0.224	1.337
SrpR_S3	5.4e-19	2.547	0.400	1.569
SrpR_S4	8.2e-16	3.314	0.108	1.344
NANDSa_1	1.109	1.774	0.123	1.077

Note: Gates prefixed NAND* are assigned only to NAND-type nodes. All others are assigned to NOT/NOR nodes. A gene family (prefix before _) may appear at most once per circuit.