Pong Trairatvorakul & Wissem Gamra

## EENG 425 FINAL PROJECT: Karplus-Strong String Synthesis

This chip implements the Karplus-Strong String Synthesis algorithm for 10-bit signals with a variable buffer size of 2 to 128. The algorithm consists of two phases (1) load in noise and (2) iterate. During the iterate phase, an output is produced every clock cycle.
Additionally, the chip also has two additional functionalities:

(1) Shift register with programmable depth
(2) Finding the mean of two numbers

Minimum clock period: 60ns

**STATES**

| State | Inputs | | Description |
|---|---|---|---|
| | load | iterate | |
| Ready | L | L | Does not do anything. Shift register not affected. Ready signal high. |
| Load | H | L | Loads samples from input into register.<br>Outputs first-in value in shift register |
| Iterate | L | H | Iterations of Karplus-Strong algorithm. Add unit takes in first-in and second-in values in shift register then outputs and stores in shift register the result of the add unit shifted right by 1. |
| Sum | H | H | Loads samples from input to shift register.<br>Outputs results of add uni shifted right by 1 directly and does not store in the shift register again. |

Any state can jump to any other state with the inputs above.

**TYPICAL USE**

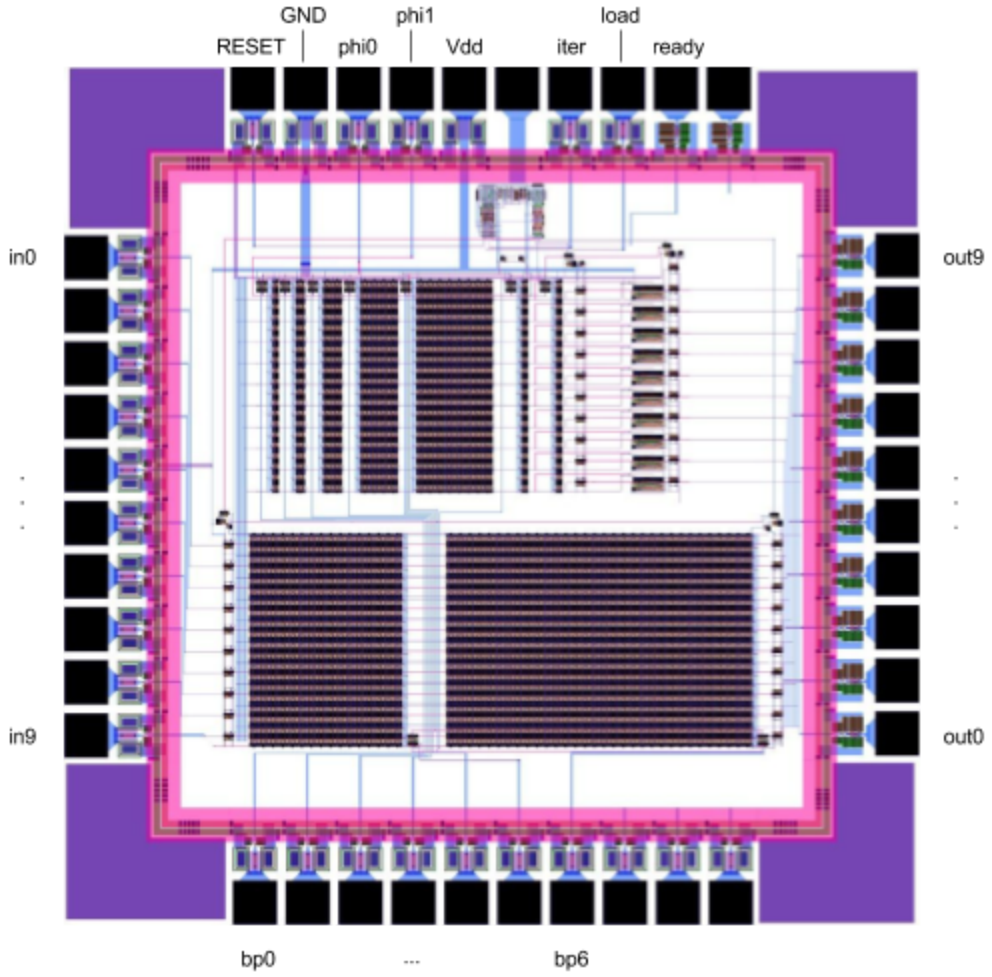
(1) Reset.
(2) Set *bypass* based on prefered size (*bypass = 129 - size*)
where *size* can be any value from 2 to 128 (inclusive)
(3) Set *load* to high
(4) Load in *size* samples to input, one at each clock cycle
(5) Before last load, change *load* to low and *iterate* to high
(6) Iterate through as many times as desired.
Result is outputted every clock cycle

**USE AS SHIFT REGISTER**

(1) Reset.
(2) Set *bypass* based on prefered size
(3) Set *load* to high
(4) Load in *size* samples to input, one at each clock cycle. The ‘first-in’ is outputted after buffer fills up.

Pong Trairatvorakul & Wissem Gamra

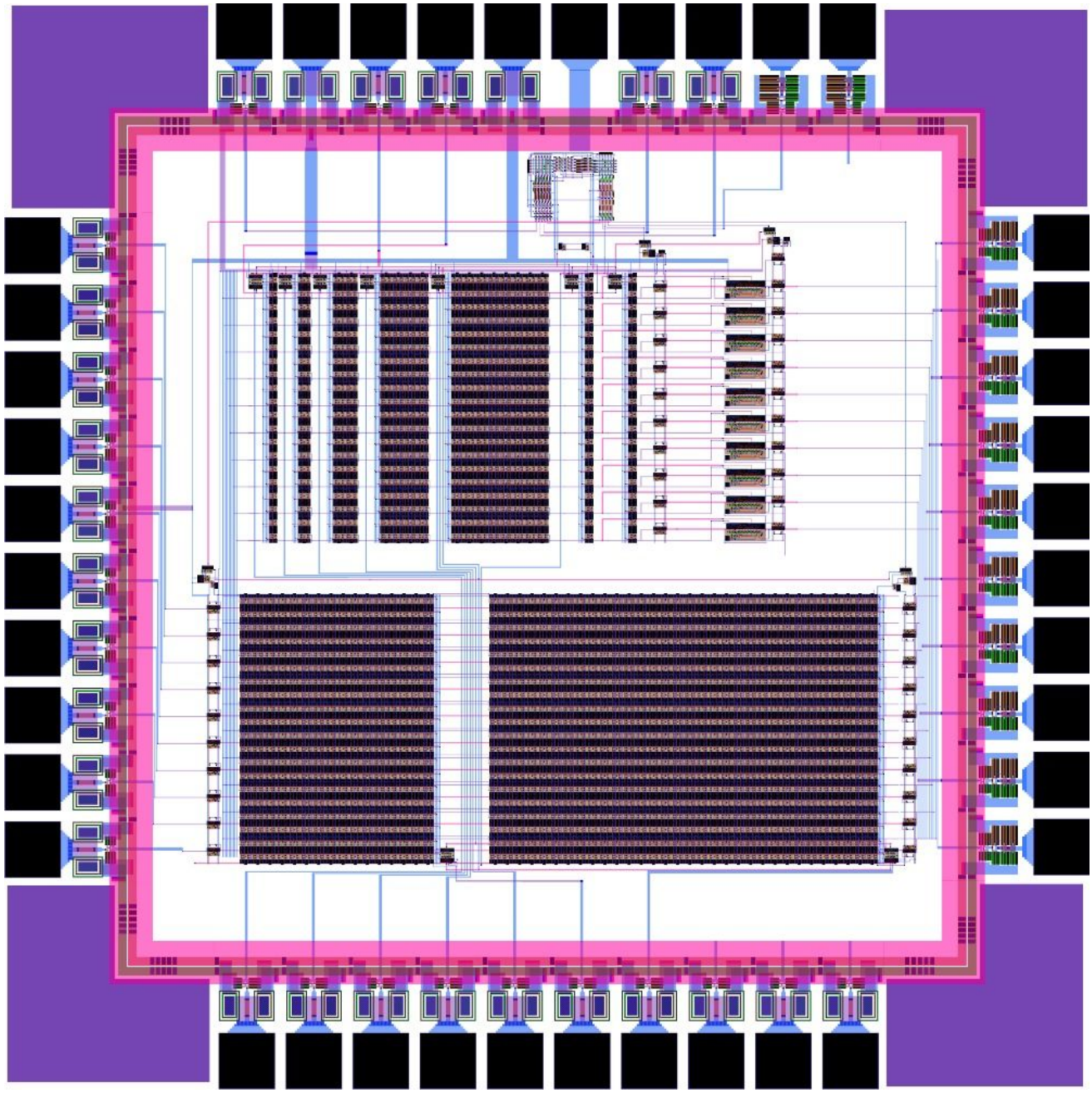
**INPUT/OUTPUT**



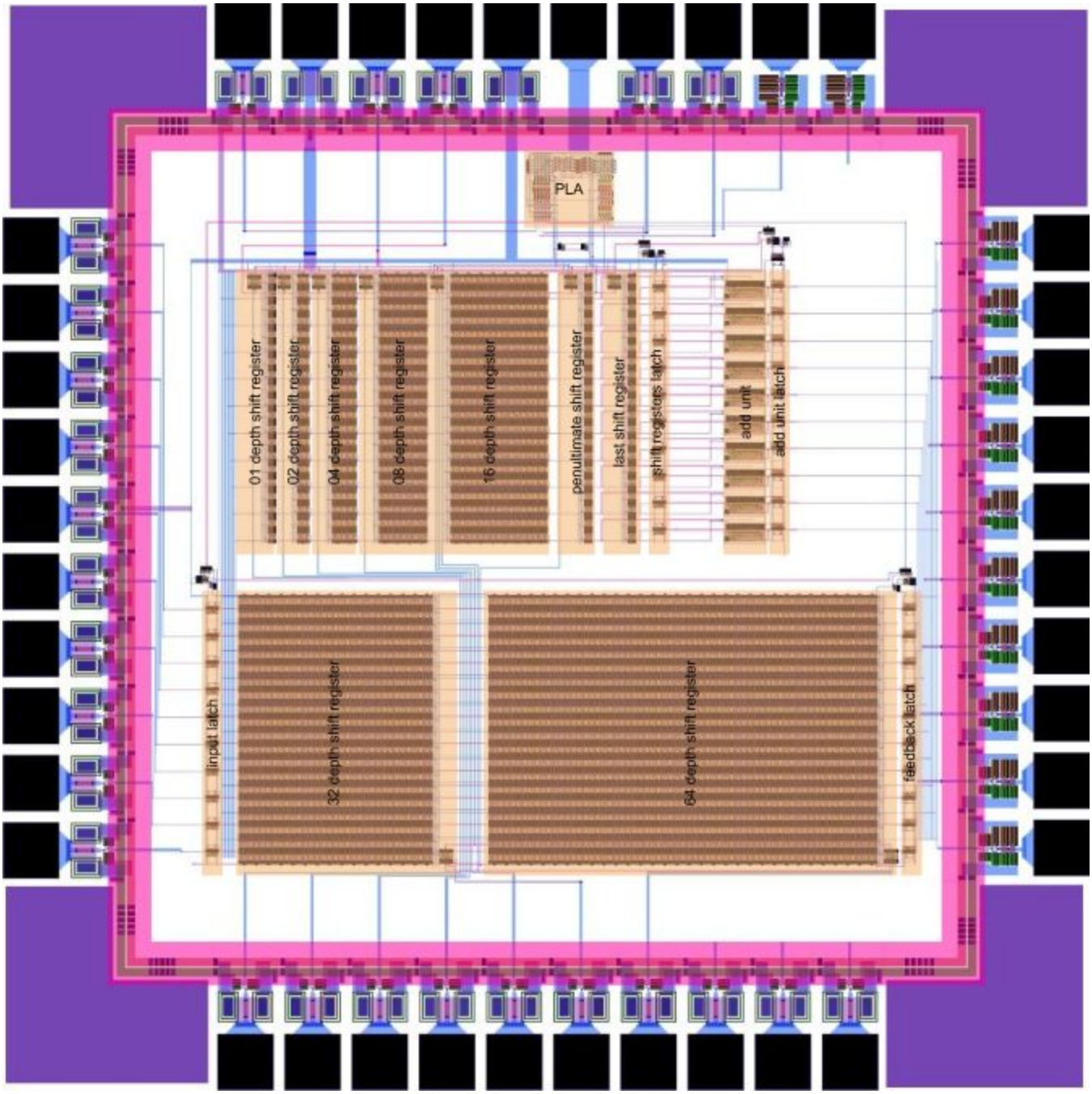

| Label | Name | Description |
|---|---|---|
| GND / Vdd | Ground and Vdd | Powers the chip |
| phi0 / phi1 | Clock | Two-phase non-overlapping clock |
| RESET | RESET control | Resets fsm to state 00 (ready state) |
| load | Load control | High to load in0 - in9 to shift register |
| iter | Iterate control | High to output average of last two in shift register |
| ready | Ready signal | Outputs high when chip is not doing anything |
| in0 - in9 | Inputs | 10-bit input signal |
| out0 - out9 | Outputs | 10-bit output signal |
| bp0 - bp9 | Bypass | 7-bit bypass to set the depth of the shift register |

Pong Trairatvorakul & Wissem Gamra

**PICTURE OF LAYOUT**



**A GUIDED TOUR OF MAJOR BLOCKS**



Pong Trairatvorakul & Wissem Gamra

**A NOTE ON TEST SCRIPTS**

**Individual Scripts**

(1) test_sreg.cmd
    Test the functionality of the shift register
(2) test_add.cmd
    Test the functionality of the add and halve unit (eg. average unit)
(3) test_iterate.cmd
    Test the iterate functionality (first load, then iterate). Essentially the algorithm

**Automatically generated scripts**

Due to the nature of the Karplus-Strong algorithm that requires a large input, we wrote a python script that can generate the irsim test script for us. The python script can be found in python_scripts

An example of a generated script is *auto1*
The expected values are *expected1*
The output from irsim is *out1.tcl* (note that it only provides the last 511 lines)
This is then converted to decimal numbers to match the format of *expected1.* (file *out1*)

**Usage:**

To automatically generate an irsim script and expected output, run:

% python python_scripts/full_test.py <buffer_size> <iterations> > <auto_filename> 2> <expected_filename>

For example:

% python python_scripts/full_test.py 127 10 > auto3 2> expected3

Then run the script in irsim as usual.
Once the script finishes execution, the results printed to stdout can be saved as a tcl (eg. out3.tcl)
We can convert that by running:

% python python_scripts/out_list.py out3.tcl > out3

This file can then be compared to the expected output.
Since irsim only saves the last 511 lines of the output, we can run diff on the two files. For example:

% diff out3 expected3

We simply have to see if the output file matches the end of expected file.