PROJECT 3
RECURSION

Deadline: 23:55 on Sunday 8th January, 2023

Last updated*: 02:07 on Monday 26th December, 2022

Instructions

(1) The project must be implemented and submitted individually.
(2) Submissions in pairs or groups are not allowed. Plagiarism will not be tolerated.
(3) Submit exactly 1 Logisim (".circ") file under the appropriate submission box on Moodle.
The file must be named `ID_recursion.circ`, with `ID` replaced by your 9 digit ID number.
(4) Use the provided `template_recursion.circ` file as a template, and implement your designs in the respective circuits. *Do not* move or modify the input/output ports, the "blackbox" layout, and the names of the circuits!
(5) If you need to use a constant 0 or 1 as a component in your circuits, use the "Constant" component from the "Wiring" library.
(6) Use tunnels to keep your circuit clean and free of spaghetti wiring. Do not remove the tunnels provided in the template. Use tunnels with matching names to connect the inputs/outputs to your circuit. Note: There is no limit on the number of tunnels corresponding to a signal. As long as they have the same name, any number of tunnels can be used for a single signal.
(7) All submissions will be graded using an automated grading system. There will be *no manual grading*, and the grade you receive will be based *exclusively* on the functionality of your circuits. No credit will be given purely for "attempted" solutions.
(8) In order to ensure that your file is compatible with the automated grading system, and to get a preliminary evaluation of the functionality, you are provided access to a validation system. In order to use this validation system, you need to send an email to `DLSBodek@gmail.com` with your Logisim (".circ") file attached.
(9) If your file contains compatibility issues and/or errors, the validation system will send you a reply with a list of error which you need to fix.
(10) If your file contains no compatibility issues and errors, the validation system will run a small number of tests on your circuits, and will return the percentage of correct outputs observed. Note that this

*This file may be updated once the project has been released, in order to fix mistakes and add clarifications. It is recommended to always download and use the latest file.

validation system checks only a small fraction of the possible inputs to your circuit. If all of these outputs are as expected, it means that your implementation may or may not be correct. However, if some of these outputs are not as expected, it means that your implementation is certainly not correct. In other words, if the preliminary score given by the validation system is less than 100 %, you still need to fix your implementation; but a preliminary score of 100 % does not guarantee a final grade of 100 %.

(11) The project consists of a number of exercises, with grades which sum up to 11.8 points. The maximal grade for each exercise is mentioned at the beginning of the exercise. Your final grade will be the grade you receive (out of 11.8), expressed as a percentage.

(12) Before submitting the file to Moodle, make sure you have validated it, and have received a "No compatibility issues detected." remark. In addition, make sure that your preliminary score is as high as possible.

(13) You *may not* use gates with fan-in[†] larger than 2. Splitters and multiplexers with any fan-in are acceptable.

(14) Unless specified otherwise, you may use components only from the following libraries provided by Logisim:
   (a) Wiring
   (b) Gates (basic gates only[‡])
   (c) Base
   (d) Plexers (multiplexers and priority encoders only)

(15) Unless specified otherwise, your implementation of a circuit in Exercise m of Project n may use circuits you designed in Exercises $m - i$ in Projects $n - j$, for all $0 < i \leq m$, $0 \leq j \leq n$.

[†]The fan-in of a gate g is the number of input terminals of g, i.e., the number of bits in the domain of the Boolean function that specifies the functionality of g.

[‡]Basic gates include NOT, AND, OR, NAND, NOR, XOR, and NXOR. Parity gates, controlled buffers, and controlled inverters do not fall under basic gates.

1. Outline

This project deals with recursive implementations of combinational circuits. The general idea of the project is the same as the previous project (adjacency matrices, matrix multiplication, and paths in graphs). However, this project will use recursive implementations of the combinational circuits, rather than iterative ones.

Section 2 explains how matrix multiplication can be performed in a fully recursive manner. Sections 3 and 4 recap the basic circuits required for matrix multiplication, already covered in the previous project. Section 6 cover the design of circuits which were not covered in the previous project. In particular, these include the recursive calculation of matrix powers, and finding the existence of required paths. In addition, they also cover functions which were not part of the previous project.

2. Matrix Multiplication

Theorem 1. *Recall the matrix multiplication defined over AND and OR in the previous project. The implementation used in the previous project can be expressed using repeated multiplication, i.e.*

$$A^\ell = \overbrace{A \cdot \dots \cdot A}^{\ell \text{ instances of } A}$$

Prove that the same can be expressed recursively using the recursive rule

$$A^\ell = \begin{cases} \left(A^2\right)^{\frac{\ell}{2}} & ; \quad \ell \text{ is even} \\ A \cdot \left(A^2\right)^{\frac{\ell-1}{2}} & ; \quad \ell \text{ is odd} \end{cases}$$

for $\ell > 2$, and the base cases

$$A^2 = A \cdot A$$

$$A^1 = A$$

3. Bitwise and Matrix Logical Operations

Preliminary Task 1. *Not for submission, repeated from the previous project*
Consider a combinational circuit *matrix_and*(n) defined as follows.
Input: $x\_row\_i \in \{0,1\}^n$, $\forall 0 \le i < n$
Output: $z \in \{0,1\}$
Functionality:

$$\begin{aligned} z &= \bigwedge_{i,j} x_{i,j} \\ &= \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} x\_row\_i[j] \end{aligned}$$

for all $0 \le i < n$ and $0 \le j < n$.

Design (on paper) an implementation of *matrix_and*(n) using only AND gates. What are the asymptotic cost and delay of your implementation?

4. Matrix Multiplication in Hardware

Preliminary Task 2. *Not for submission, repeated from the previous project*
Consider a combinational circuit *matmul_step_1*(n) defined as follows.

Input: $row \in \{0,1\}^n$, $col \in \{0,1\}^n$
Output: $output \in \{0,1\}$
Functionality: *output* represents the inner product of the inputs, over AND and OR, i.e.

$$output = \bigvee_{i=0}^{n-1} row[i] \wedge col[i]$$

Design (on paper) an implementation of *matmul_step_1*(n) using only AND and OR gates. What are the asymptotic cost and delay of your implementation?

Preliminary Task 3. *Not for submission, repeated from the previous project*
Consider a combinational circuit *matmul_step_2*(n) defined as follows.

Input: $row \in \{0,1\}^n$, $col\_i \in \{0,1\}^n$, $\forall 0 \leq i < n$
Output: $product\_row \in \{0,1\}^n$
Functionality:

$$product\_row[n-j-1] = \bigvee_{i=0}^{n-1} row[i] \wedge col\_j[i]$$

Design (on paper) an implementation of *matmul_step_1*(n) using at most n instances of *matmul_step_1*(n) (and optionally other basic logic gates). What are the asymptotic cost and delay of your implementation?

Preliminary Task 4. *Not for submission, repeated from the previous project*
Consider a combinational circuit *matmul*(n) defined as follows.

Input: $x\_row\_i \in \{0,1\}^n$, $y\_row\_i \in \{0,1\}^n$, $\forall 0 \leq i < n$
Output: $z\_row\_i \in \{0,1\}^n$, $\forall 0 \leq i < n$
Functionality: Let x, y, and z be matrices defined by the corresponding rows x_row_i, y_row_i, and z_row_i. Then, the outputs z_row_i are such that $z = x \cdot y$, with multiplication defined over AND and OR. Formally,

$$z\_row\_i[n-j-1] = \bigvee_{k=0}^{n-1} x\_row\_i[k] \wedge y\_col\_j[k]$$

where

$$y\_col\_i = y\_row\_0[n-i-1] \circ \cdots \circ y\_row\_(n-1)[n-i-1]$$

Design (on paper) an implementation of *matmul*(n) using at most n instances of *matmul_step_2*(n) (and optionally other basic logic gates). In addition, you may also use one instance of the combinational circuit *transpose*(n) which accepts as input an $n \times n$ binary matrix and outputs its transpose.

5. Powers of Matrices

Preliminary Task 5. *Not for submission*
Consider a combinational circuit *pow*(k, n) defined as follows.

Input: $x\_row\_i \in \{0,1\}^n$, $\forall 0 \leq i < n$
Output: $z\_row\_i \in \{0,1\}^n$, $\forall 0 \leq i < n$
Functionality: Let x, and z be matrices defined by the corresponding rows x_row_i, and z_row_i. Then, the outputs z_row_i are such that $z = x^k$, with powers of matrices defined using multiplication over AND and OR.

(1) Design (on paper) an implementation of *pow*(k, n) for $k = 0$. Use the minimal number of logical components.
(2) Design (on paper) an implementation of *pow*(k, n) for $k = 1$. Use the minimal number of logical components.
(3) Design (on paper) an implementation of *pow*(k, n) for $k = 2$, using at most one instance of *matmul*(n).
(4) Design (on paper) a *fully recursive* implementation of *pow*(k, n) for $k = \{4, 6, 8, \dots\}$, following the recursive expression given in Theorem 1. Your design must use at most two instances of *pow* $\left(\tilde{k}, n\right)$ where $\tilde{k} < k$. You may not use any other circuits or gates.
(5) Design (on paper) a *fully recursive* implementation of *pow*(k, n) for $k = \{3, 5, 7, \dots\}$, following the recursive expression given in Theorem 1. Your design must use at most one instance of *pow* $\left(\tilde{k}, n\right)$ where $\tilde{k} < k$, and at most one instance of *matmul*(n). You may not use any other circuits or gates.
(6) What are the asymptotic cost and delay of your implementation? How do these compare to the repeated multiplication-based implementation from the previous project?

6. Existence of Paths Using Matrix Multiplication

Preliminary Task 6. *Not for submission*
Consider a combinational circuit *path_in_exactly*(ℓ, n) defined as follows.

Input: $x\_row\_i \in \{0,1\}^n$, $\forall 0 \leq i < n$, *from* $\in \{0,1\}^{\log n}$, *to* $\in \{0,1\}^{\log n}$
Output: *path_exists* $\in \{0,1\}$

Functionality: Let

$$i = \langle \mathit{from} \rangle$$
$$j = \langle \mathit{to} \rangle$$

Then,

$$\mathit{path\_exists} = \begin{cases} 1 & ; \quad \exists \text{ path of length } \textit{exactly } \ell \text{ from } v_i \text{ to } v_j \\ 0 & ; \quad \text{otherwise} \end{cases}$$

Design (on paper) an implementation of $\mathit{path\_in\_exactly}(\ell, n)$ using exactly one instance of $\mathit{pow}(\ell, n)$, and multiplexers (with any fan-in). What is the asymptotic cost and delay of your implementation?

Preliminary Task 7. *Not for submission*
Consider a combinational circuit $\mathit{exactly}(n)$ defined as follows.
Input: $\mathit{x\_row\_i} \in \{0,1\}^n$, $\forall 0 \leq i < n$, $\mathit{from} \in \{0,1\}^{\log n}$, $\mathit{to} \in \{0,1\}^{\log n}$, $\mathit{num\_steps} \in \{0,1\}^{\log n}$,
Output: $\mathit{path\_exists} \in \{0,1\}$
Functionality: Let

$$i = \langle \mathit{from} \rangle$$
$$j = \langle \mathit{to} \rangle$$
$$\ell = \langle \mathit{num\_steps} \rangle$$

Then,

$$\mathit{path\_exists} = \begin{cases} 1 & ; \quad \exists \text{ path of length } \textit{exactly } \ell \text{ from } v_i \text{ to } v_j \\ 0 & ; \quad \text{otherwise} \end{cases}$$

Design (on paper) an implementation of $\mathit{exactly}(n)$ using at most n instances of $\mathit{path\_in\_exactly}(k, n)$, and multiplexers (with any fan-in).
What is the asymptotic cost and delay of your implementation? How do these compare to the implementation of $\mathit{exactly}(n)$ from the previous project?

Preliminary Task 8. *Not for submission*
Consider a combinational circuit $\mathit{atmost}(n)$ defined as follows.
Input: $\mathit{x\_row\_i} \in \{0,1\}^n$, $\forall 0 \leq i < n$, $\mathit{from} \in \{0,1\}^{\log n}$, $\mathit{to} \in \{0,1\}^{\log n}$, $\mathit{num\_steps} \in \{0,1\}^{\log n}$,
Output: $\mathit{path\_exists} \in \{0,1\}$
Functionality: Let

$$i = \langle \mathit{from} \rangle$$
$$j = \langle \mathit{to} \rangle$$
$$\ell = \langle \mathit{num\_steps} \rangle$$

Then,

$$path\_exists = \begin{cases} 1 & ; \quad \exists \text{ path of length } \textit{at most } \ell \text{ from } v_i \text{ to } v_j \\ 0 & ; \quad \text{otherwise} \end{cases}$$

Design (on paper) an implementation of $atmost(n)$ using at most n instances of $path\_in\_exactly(k, n)$, multiplexers (with any fan-in), and basic logic gates (with maximum fan-in of 2).
What is the asymptotic cost and delay of your implementation? How do these compare to the implementation of $atmost(n)$ from the previous project?

Preliminary Task 9. *Not for submission*
Consider a combinational circuit $min\_dist\_i\_j(n)$ defined as follows.
Input: $x\_row\_i \in \{0,1\}^n$, $\forall 0 \le i < n$, $from \in \{0,1\}^{\log n}$, $to \in \{0,1\}^{\log n}$,
Output: $path\_exists \in \{0,1\}$, $min\_dist \in \{0,1\}^{\log n}$
Functionality: Let

$$i = \langle from \rangle$$
$$j = \langle to \rangle$$

Then, $path\_exists = 1$ iff there exists a path from v_i to v_j. If there exists such a path, then the binary string $min\_dist[\log n - 1 : 0]$ represents the length of the shortest such path.

Design (on paper) an implementation of $min\_dist\_i\_j(n)$ using $path\_in\_exactly(k, n)$, and other logic gates.
Hints:

(1) The circuit $path\_in\_exactly(k, n)$ can be used to find whether there exists a path from v_i to v_j of length exactly k.
(2) If there exists such a path with length k, then the existence of such a path with length $\tilde{k} > k$ is inconsequential. Equivalently, paths of short lengths must be *prioritized* when finding the shortest path.

Preliminary Task 10. *Not for submission*
Consider a combinational circuit $min\_dist\_anywhere(n)$ defined as follows.
Input: $x\_row\_i \in \{0,1\}^n$ for all $0 \le i < n$,
Output: $path\_exists \in \{0,1\}$, $min\_dist \in \{0,1\}^{\log n}$
Functionality: $path\_exists = 1$ iff $\exists \ell \in \mathbb{N}$, such that for any pair of vertices v_i and v_j, there exists a path of length ℓ between the vertices. In such a scenario, the binary string $min\_dist[\log n - 1 : 0]$ represents the minimum value of ℓ.
Equivalently, the binary string $min\_dist[\log n - 1 : 0]$ represents the minimum number of steps required to go from any vertex in the graph to any other vertex in the graph, provided it is possible.

Design (on paper) an implementation of $min\_dist\_anywhere(n)$ using $pow(k, n)$, and other logic gates.
Hints:

(1) It is possible to get from any vertex in a graph to any other vertex in the graph in ℓ steps if and only if there exist paths of length ℓ between *all* pairs of vertices.
(2) The (i, j)th element of the output of $pow(\ell, n)$ indicates whether there exists a path from v_i to v_j with length ℓ.

7. Exercises

Exercise 1. 0.1 P.
Using your design from Preliminary Task 1, complete the circuit `matrix_and_8_8` from the provided template to implement $matrix\_and(8)$.

You may reuse your implementation from the previous project, as long as it is contained in the Logisim file for the current project. It is recommended to copy and paste your implementation from the previous project, *ensuring that I/O ports from the old file are not copied over.*

Exercise 2. 0.1 P.
Using your design from Preliminary Task 2, complete the circuit `matmul_step_1` from the provided template to implement $matmul\_step\_1(8)$.

You may reuse your implementation from the previous project, as long as it is contained in the Logisim file for the current project. It is recommended to copy and paste your implementation from the previous project, *ensuring that I/O ports from the old file are not copied over.*

Exercise 3. 0.1 P.
Using your design from Preliminary Task 3, complete the circuit `matmul_step_2` from the provided template to implement $matmul\_step\_2(8)$.

You may reuse your implementation from the previous project, as long as it is contained in the Logisim file for the current project. It is recommended to copy and paste your implementation from the previous project, *ensuring that I/O ports from the old file are not copied over.*

Exercise 4. 0.1 P.
Using your design from Preliminary Task 4, complete the circuit `matmul` from the provided template to implement $matmul(8)$. You may use the circuit `transpose` from the provided template as a sub-block in your implementation.

You may reuse your implementation from the previous project, as long as it is contained in the Logisim file for the current project. It is recommended to copy and paste your implementation from the previous project, *ensuring that I/O ports from the old file are not copied over.*

Exercise 5. 0.2 P.
Using your design from Preliminary Task 5, complete the circuit `pow_0` from the provided template to implement $pow(0, 8)$.

Exercise 6. 0.2 P.
Using your design from Preliminary Task 5, complete the circuit `pow_1` from the provided template to implement $pow(1, 8)$.

Exercise 7. 0.5 P.
Using your design from Preliminary Task 5, complete the circuit `pow_2` from the provided template to implement $pow(2, 8)$.

Exercise 8. 2.5 P.
Using your design from Preliminary Task 5, complete the circuits `pow_i` for all $3 \leq i < 8$ from the provided template to implement the corresponding $pow(i, 8)$. Your designs must be *fully recursive*, as explained in Preliminary Task 5.

Exercise 9. 2 P.
Using your design from Preliminary Task 6, complete the circuits `path_in_exactly_i` for all $0 \leq i < 8$ from the provided template to implement the corresponding $path\_in\_exactly(i, 8)$.

Exercise 10. 1 P.
Using your design from Preliminary Task 7, complete the circuit `exactly` from the provided template to implement $exactly(8)$.

Exercise 11. 1 P.
Using your design from Preliminary Task 8, complete the circuit `atmost` from the provided template to implement *atmost*(8).

Exercise 12. 2 P.
Using your design from Preliminary Task 9, complete the circuit `min_dist_i_j` from the provided template to implement $min\_dist\_i\_j(8)$.

Exercise 13. 2 P.
Using your design from Preliminary Task 10, complete the circuit `min_dist_anywhere` from the provided template to implement $min\_dist\_anywhere(8)$.