PROJECT 2
MATRIX MULTIPLICATION AND PATHS IN GRAPHS

Deadline: 23:55 on Thursday 15th December, 2022

Last updated*: 15:41 on Friday 16th 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_matmul.circ`, with `ID` replaced by your 9 digit ID number.
(4) Use the provided `template_matmul.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 13 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 13), 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 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 non-recursive implementations of combinational circuits. In particular, the project covers converting algebraic expressions defining the functionality of a circuit into hardware implementations.

The overarching theme of the project is analysing directed graphs using their matrix representations and basic logic gates. The circuits you will design in Section 8 will receive as input a graph (represented as a matrix), and will output a variety of properties of the graph, e.g. the existence of a path between a given pair of vertices, the connected-ness of the graph, etc. Sections 2 to 7 consist of a number of definitions, theorems, and preliminary tasks meant as stepping stones, intended to guide you towards the circuits' implementation.

Sections 2 and 3 cover the required basics of graph theory, including matrix representations of directed graphs. Section 4 explains how basic techniques from linear algebra can be used to obtain information about the existence of paths in graphs. Sections 5 and 6 explains how to implement required algebraic expressions using basic logic gates.

1.1. **Short Introduction to Multiplexers.**

The following is a very short introduction to multiplexers (MUXes). It is meant to provide you with the bare minimum information required for using MUXes as circuit elements. This topic will be covered in depth in class, in the coming weeks.

Definition 1. A $n : 1$ multiplexer (MUX) is a combinational gate with $n + 1$ inputs $D_0, D_1, \ldots, D_{n-1} \in \{0, 1\}$, and $S \in \{0, 1\}^{\log n}$, and one output $Y \in \{0, 1\}$ The functionality of a MUX is defined as

$$Y = \begin{cases} D_0 & ; \quad \langle S \rangle = 0 \\ D_1 & ; \quad \langle S \rangle = 1 \\ & \vdots \\ D_{n-1} & ; \quad \langle S \rangle = n - 1 \end{cases}$$

where $\langle S \rangle$ denotes the number represented by the binary string S.

In essence, a MUX allows you to select between the inputs $D_0, \ldots, D_{n-1}$, based on the number represented by S. This allows MUXes to be used to implement expressions with multiple cases, similar to if-conditions in software. For example, if $n = 8$, then $D_0, \ldots, D_7 \in \{0, 1\}$ and $S \in \{0, 1\}^3$. Hence, the output Y is "connected" to the input D_0 if $S = 000$, to D_1 if $S = 001$, to D_7 if $S = 111$, etc.

2. Directed Graphs

Definition 2 (Connected graph)**.** A directed graph $G = (V, E)$ is said to be connected if for any $v_i, v_j \in V$, there exists a path from v_i to v_j.

Definition 3 (Fully connected graph)**.** A directed graph $G = (V, E)$ is said to be fully connected if for any $v_i, v_j \in V$, there exists a path of length at most 1, (i.e. an edge) from v_i to v_j.

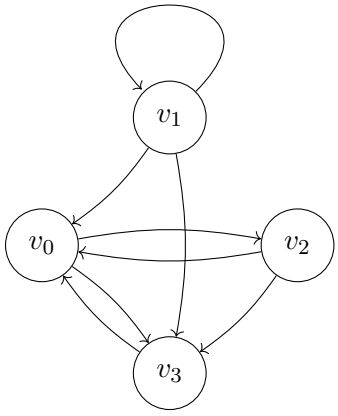
Definition 4 (Path of length zero)**.** For any vertex in a directed graph, there exists a path of length zero from the vertex to itself.

3. Adjacency Matrices

Definition 5 (Adjacency matrix)**.** Consider a directed graph $G = (V, E)$, such that $|V| = n$. The adjacency matrix A corresponding to G is defined to be a $n \times n$ matrix, such that each element in A is defined as

$$A_{i,j} = \begin{cases} 1 & ; (i,j) \in E \\ 0 & ; (i,j) \notin E \end{cases}$$

for all $0 \leq i < n$, $0 \leq j < n$.
Equivalently, the element in the ith row and jth column of A is 1 if and only if there exists an edge from the ith vertex in V to the jth vertex in V. Note that the rows and columns of A are numbered starting from 0, going top to bottom and left to right, respectively.



$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 1. A Graph and the Corresponding Adjacency Matrix

4. Paths and Matrix Multiplication

Theorem 1. *Consider a directed graph $G = (V, E)$ and its corresponding adjacency matrix $A \in \{0,1\}^{n \times n}$. For every distinct $0 \leq i < n$ and $0 \leq j < n$, there exists a path of length 2 from $v_i \in V$ to $v_j \in V$ iff there exists at least one $0 \leq k < n$ such that there exists an edge $v_i \to v_k$ and an edge $v_k \to v_j$. Equivalently, there exists a path of length 2 from $v_i \in V$ to $v_j \in V$ iff*

$$\bigvee_{k=0}^{n-1} A_{i,k} \wedge A_{k,j} = 1$$

Theorem 2. *Consider a matrix $A \in \{0,1\}^{n \times n}$. Let*

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

where matrix multiplication is defined as usual, with multiplication replaced by AND and addition replaced by OR. Then,

$$A^{\ell}{}_{i,j} = \bigvee_{k_1=0}^{n-1} \left(\cdots \bigvee_{k_{\ell-1}=0}^{n-1} \left(A_{i,k_1} \wedge A_{k_1,k_2} \wedge A_{k_2,k_3} \cdots \wedge A_{k_{\ell-1},j} \right) \right)$$

Hence, there exists a path of length ℓ from $v_i \in V$ to $v_j \in V$ iff $A^{\ell}{}_{i,j} = 1$.

5. Bitwise and Matrix Logical Operations

Preliminary Task 1. *Not for submission*

Consider a combinational circuit *bitwise_or*(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 is the bitwise OR of x and y, i.e.

$$z\_row\_i[j] = x\_row\_i[j] \vee y\_row\_i[j]$$

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

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

Preliminary Task 2. *Not for submission*

Consider a combinational circuit *matrix_and*(n) defined as follows.

Input: $x\_row\_i \in \{0,1\}^n$, $\forall 0 \leq i < n$
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 $0 \leq i < n$ and $0 \leq j < n$.

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

6. Matrix Multiplication in Hardware

Preliminary Task 3. *Not for submission*

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 4. *Not for submission*

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_2*(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 5. *Not for submission*

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.

7. Existence of Paths Using Matrix Multiplication

Preliminary Task 6. *Not for submission*

Consider a combinational circuit *exactly*(n) defined as follows.

Input: *x_row_i* $\in \{0,1\}^n$, $\forall 0 \leq i < n$, *from* $\in \{0,1\}^{\log n}$, *to* $\in \{0,1\}^{\log n}$, *num_steps* $\in \{0,1\}^{\log n}$,
Output: *path_exists* $\in \{0,1\}$
Functionality: Let

$i = \langle from \rangle$

$j = \langle to \rangle$

$\ell = \langle num\_steps \rangle$

Then,

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

Design (on paper) an implementation of *exactly*(n) using at most $n-2$ instances of *matmul*(n), multiplexers (with any fan-in), and basic logic gates (with maximum fan-in of 2).
Hints:

(1) There exists a path of length ℓ from v_i to v_j iff $A^\ell{}_{i,j} = 1$.
(2) $A^\ell = A^{\ell-1} \cdot A$.
(3) A multiplexer can be used to select one out of a number of inputs.
(4) The (i,j)th element of a matrix can be selected by selecting the ith row of a matrix, and then selecting the jth element of the row.
(5) There exists a path of length 0 between any vertex and itself.

Preliminary Task 7. *Not for submission*

Consider a combinational circuit *atmost*(n) defined as follows.

Input: *x_row_i* $\in \{0,1\}^n$, $\forall 0 \leq i < n$, *from* $\in \{0,1\}^{\log n}$, *to* $\in \{0,1\}^{\log n}$, *num_steps* $\in \{0,1\}^{\log n}$,
Output: *path_exists* $\in \{0,1\}$
Functionality: Let

$i = \langle from \rangle$

$j = \langle to \rangle$

$\ell = \langle num\_steps \rangle$

Then,

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

Design (on paper) an implementation of *atmost*(n) using at most $n-2$ instances of *matmul*(n), multiplexers (with any fan-in), and basic logic gates

(with maximum fan-in of 2).
Hints:

(1) There exists a path of length at most ℓ between two vertices iff there exists a path of length p between the vertices, for at least one $0 \leq p \leq \ell$.
(2) There exists a path of length 0 between any vertex and itself.

Preliminary Task 8. *Not for submission*
Consider a combinational circuit *is_connected*(n) defined as follows.

Input: $x\_row\_i \in \{0,1\}^n, \forall 0 \leq i < n$
Output: $is\_connected \in \{0,1\}$
Functionality: Let x be the adjacency matrix defined by the corresponding rows x_row_i, and let G be the graph represented by x. Then,

$$is\_connected = \begin{cases} 1 & ; \quad G \text{ is connected} \\ 0 & ; \quad \text{otherwise} \end{cases}$$

Design (on paper) an implementation of *is_connected*(n) using at most $n-2$ instances of *matmul*(n), at most $n-1$ instances of *bitwise_or*(n), at most one instance of *matrix_and*(n), multiplexers (with any fan-in), and basic logic gates (with maximum fan-in of 2).
Hints:

(1) If a graph is connected, what can you say about the existence of paths between an arbitrary pair of vertices?
(2) If a graph with n vertices is connected, what is the maximal length of the shortest path between any pair of vertices?

Preliminary Task 9. *Not for submission*
Consider a combinational circuit *is_fully_connected*(n) defined as follows.

Input: $x\_row\_i \in \{0,1\}^n$ for all $0 \leq i < n$
Output: $is\_fully\_connected \in \{0,1\}$
Functionality: Let x be the adjacency matrix defined by the corresponding rows x_row_i, and let G be the graph represented by x.

$$is\_fully\_connected = \begin{cases} 1 & ; \quad G \text{ is fully connected} \\ 0 & ; \quad \text{otherwise} \end{cases}$$

Design (on paper) an implementation of *is_fully_connected*(n) using *matmul*(n), *bitwise_or*(n), *matrix_and*(n), and basic logic gates (with maximum fan-in of 2).
Hints:

(1) If a graph is fully connected, what can you say about the existence of paths between an arbitrary pair of vertices?

(2) If a graph with n vertices is fully connected, what is the maximal length of the shortest path between any pair of vertices?

8. Exercises

Exercise 1. 1 P.

Using your design from Preliminary Task 1, complete the circuit `bitwise_or_8_8` from the provided template to implement *bitwise_or*(8). You may use the LED matrices provided in the template as debugging tools in your implementation.

Exercise 2. 1 P.

Using your design from Preliminary Task 2, complete the circuit `matrix_and_8_8` from the provided template to implement *matrix_and*(8). You may use the LED matrix provided in the template as a debugging tool in your implementation.

Exercise 3. 1 P.

Using your design from Preliminary Task 3, complete the circuit `matmul_step_1` from the provided template to implement *matmul_step_1*(8).

Exercise 4. 1 P.

Using your design from Preliminary Task 4, complete the circuit `matmul_step_2` from the provided template to implement *matmul_step_2*(8).

Exercise 5. 1 P.

Using your design from Preliminary Task 5, 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 also use the LED matrices provided in the template as debugging tools in your implementation.

Exercise 6. 3 P.

Using your design from Preliminary Task 6, complete the circuit `exactly` from the provided template to implement *exactly*(8). You may use multiplexers

(of any fan-in) from Logisim's "Plexers" library. You may also use the LED matrix provided in the template as a debugging tool in your implementation.

Exercise 7. 2 P.

Using your design from Preliminary Task 7, complete the circuit `atmost` from the provided template to implement *atmost*(8). You may use multiplexers (of any fan-in) from Logisim's "Plexers" library. You may also use the LED matrix provided in the template as a debugging tool in your implementation.

Exercise 8. 2 P.

Using your design from Preliminary Task 8, complete the circuit `is_connected` from the provided template to implement *is_connected*(8). You may use the LED matrix provided in the template as a debugging tool in your implementation.

Exercise 9. 1 P.

Using your design from Preliminary Task 9, complete the circuit `is_fully_connected` from the provided template to implement *is_fully_connected*(8). You may use the LED matrix provided in the template as a debugging tool in your implementation.