Phase 1 & Phase 2

Fibonacci Sequence Number Identifier Circuit

Created by: Arsalan Khan

Student ID: 210862640

Course: CP220 - Digital Electronics

OVERVIEW

The project's core objective is to develop a digital circuit that calculates the nth term in the Fibonacci sequence, where the value of n is inputted in binary format. This circuit is designed to assist in computational tasks where quick retrieval of Fibonacci numbers is essential.

GOALS

A. General

1. **Professional Presentation:**

   The document is concise, clearly structured, and adheres to professional standards in presenting the project requirements.

2. **Timeliness:**

   The document is prepared and submitted as per the timeline provided.

3. **Grammar & Spelling:**

   The text is reviewed and ensured to be free of grammatical and spelling errors.

B. Content

1. **Background:**

   The project is tailored to create a circuit that, when provided with a binary input, calculates the corresponding number in the Fibonacci sequence.

2. **Inputs (a0 to a2):**

   The circuit will be designed to accept a 3-bit binary input (ranging from 0 to 7), representing the position in the Fibonacci sequence.

3. **Outputs (b0 to b3):**

   Four output lines, presenting the corresponding Fibonacci number in binary format.

4. **Error Conditions and Responses:**

   Specific error handling for inputs requesting Fibonacci terms outside the pre-defined range, resulting in an error output pattern.

5. **Eliminate Ambiguities:**

   A well-defined algorithm for calculating Fibonacci numbers ensures consistent and accurate outputs.

PROBLEM STATEMENT

The designed circuit aims to swiftly and accurately provide the nth Fibonacci number, corresponding to a 3-bit binary input, ensuring precision and consistency in outputs and error handling.

SPECIFICATIONS

Inputs:

a0 to a2: Three binary inputs capable of representing numbers from 0 to 7, indicating the position in the Fibonacci sequence.

Outputs:

b0 to b3: Four binary outputs showcasing the Fibonacci number corresponding to the provided input.

Error Conditions & Responses:

Specific error outputs for inputs that are not within the acceptable range or those that do not correspond to a Fibonacci term within the sequence's limit set for this circuit.

1. Out-of-Range Inputs:

Scenario: The circuit is designed to handle inputs corresponding to specific positions in the Fibonacci sequence. If an input represents a position beyond this limit, it should be treated as an error.

Error Response: A specific binary pattern, distinct from valid Fibonacci numbers, is outputted to signal this error.

2. Hardware Limitation:

Scenario: The circuit might have hardware limitations, like a maximum calculable Fibonacci number due to the bit width of the output or other hardware constraints.

Error Response: A unique error pattern in the output to indicate that the requested Fibonacci number exceeds the hardware's calculation capability.

3. Invalid Input Patterns:

Scenario: If the circuit receives an input pattern that doesn't correspond to any valid request (although this is mitigated by the 3-bit input width which inherently restricts the input to 0-7).

Error Response: The circuit could generate a unique error pattern distinct from valid outputs.

Ambiguous Cases:

1. Starting Index:

Alternative Interpretation: The sequence starts at the first term as 0 (0, 1, 1, 2, 3...).

Terms Generated: When input is ‘001’ (representing 1), output could potentially be ‘0000’ (representing 0 as the first term).

Standard to be Used: The project will adopt the convention that the sequence starts with the first term as 1 (1, 1, 2, 3...). Therefore, an input of ‘001’ will yield an output of ‘0001’, representing 1 as the first term.

2. Error Handling for Zero:

Alternative Interpretation: Input of ‘000’ could be treated as an error since there's no "zeroth" term in the sequence, or it could be assigned to a default value.

Output: The circuit could potentially yield an error signal or a default value, leading to an inconsistency in interpretation.

Standard to be Used: The project will treat an input of ‘000’ as an error, and a specific error output pattern will be generated. This ensures consistency and avoids the ambiguity of assigning a value to the "zeroth" term.

APPENDIX – TABLES/SUPPORTING DIAGRAMS

Appendix A:

Binary Input (a2 a1 a0)	Decimal Input	Fibonacci Sequence Number (Decimal)	Binary Output (b3 b2 b1 b0)
0	0	Error - Zeroth Term	0
1	1	1	1
10	2	1	1
11	3	2	10
100	4	3	11
101	5	5	101
110	6	8	1000
111	7	13	1101
1000	8	Error - Out of Range	1110
1001	9	Error - Out of Range	1110
1010	10	Error - Hardware Limitation	1111

Phase 2

Fibonacci Sequence Number Identifier Circuit

Created by: Arsalan Khan

Student ID: 210862640

Course: CP220 - Digital Electronics

OVERVIEW

The project's core objective is to develop a digital circuit that calculates the nth term in the Fibonacci sequence, where the value of n is inputted in binary format. This circuit is designed to assist in computational tasks where quick retrieval of Fibonacci numbers is essential.

GOALS

A. General

1. **Professional Presentation:**

   The document is concise, clearly structured, and adheres to professional standards in presenting the project requirements.

2. **Timeliness:**

   The document is prepared and submitted as per the timeline provided.

3. **Grammar & Spelling:**

   The text is reviewed and ensured to be free of grammatical and spelling errors.

B. Content

1. All Outputs Analyzed (Truth Table):

Binary Input (a2 a1 a0)	Decimal Input	Fibonacci Sequence Number (Decimal)	Binary Output (b3 b2 b1 b0)
0	0	Error - Zeroth Term	0
1	1	1	1
10	2	1	1
11	3	2	10
100	4	3	11
101	5	5	101
110	6	8	1000
111	7	13	1101
1000	8	Error - Out of Range	1110
1001	9	Error - Out of Range	1110
1010	10	Error - Hardware Limitation	1111

Truth Table Analysis:

The truth table provides a mapping between 3-bit binary inputs and their corresponding numbers in the Fibonacci sequence, represented as 4-bit binary outputs.

Binary Inputs (a2 a1 a0): This is the 3-bit binary representation of numbers ranging from 0 to 7.

Decimal Input: This translates the 3-bit binary input into its decimal equivalent, essentially numbering the rows from 0 to 7.

Fibonacci Sequence Number (Decimal): For each binary (and its corresponding decimal) input, this column indicates the value in the Fibonacci sequence. Notably:

The zeroth term is flagged as an error since Fibonacci sequences typically start from the first term.

Inputs of 8 and 9 are considered out of the table's range and thus labeled as "Error - Out of Range."

A binary input representing the decimal value 10 is marked as "Error - Hardware Limitation."

Binary Output (b3 b2 b1 b0): This is the 4-bit binary representation of the Fibonacci number corresponding to each input. It's designed to handle larger Fibonacci numbers, hence the necessity for 4 bits.

Key Observations:

Error Handling:

When the binary input is 000 or 0 in decimal, it corresponds to the "zeroth" term in the Fibonacci sequence, which is not standard and hence is termed as an error.

When the binary input exceeds 7 or 111 in binary, it falls out of the range of the Fibonacci sequence that the system is designed to handle. Specifically, the numbers 8 (1000 in binary) and 9 (1001 in binary) are termed as "out of range".

A binary input of 10 (1010 in binary) triggers a "hardware limitation", implying that this specific system wasn't designed to compute the Fibonacci sequence for such inputs.

Standard Mapping:

For valid binary inputs between 1 (001 in binary) and 7 (111 in binary), the output correctly represents the Fibonacci sequence numbers in their binary form. For instance, an input of 2 (010 in binary) correctly produces an output of 1 (001 in binary) as the Fibonacci number.

Binary Sequence Order:

The binary input sequence follows a standard count-up pattern from 0 to 7. The output, however, doesn't follow this pattern, as it's based on the Fibonacci sequence. This is expected since the Fibonacci sequence doesn't grow linearly.

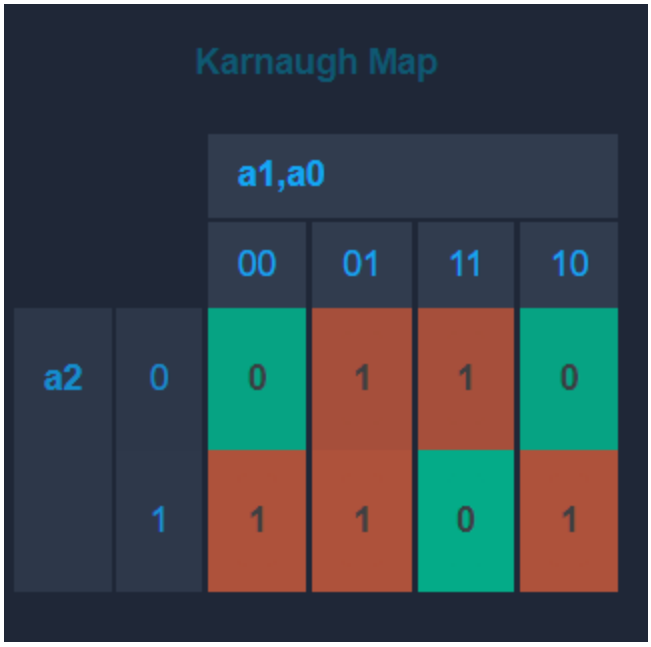
Increased Output Width:

While the input is only 3-bits wide, the output is 4-bits wide. This allows the system to represent larger Fibonacci numbers, such as 13 (1101 in binary), which is the largest Fibonacci number represented in this truth table.

2. Karnaugh Maps and Equations

We must prove this by showing the truth tables and the k-maps that correspond to the outputs:

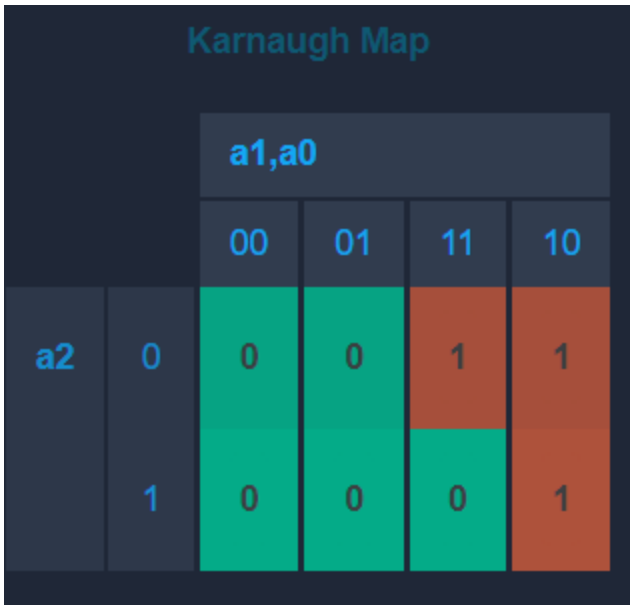
Karnaugh Map for Output b0:



Truth Table Representation of b0:

Binary Input (a2	a1	a0)	b0
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

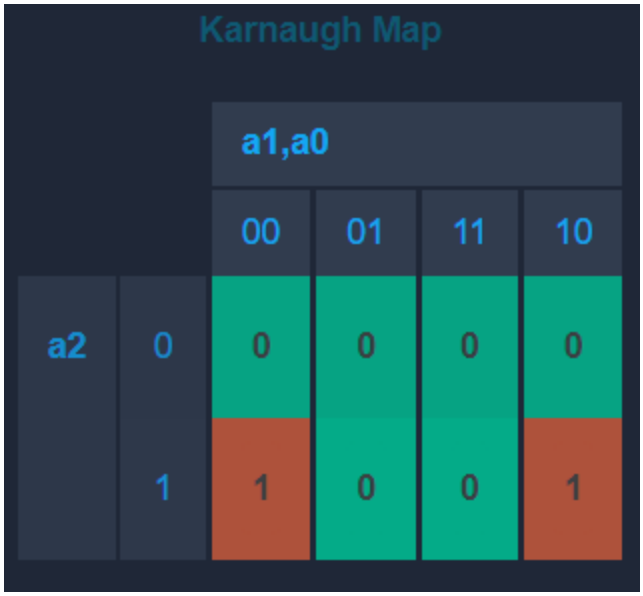
Karnaugh Map for Output b1:



Truth Table Representation of b1:

Binary Input (a2	**a1**	**a0)**	**b1**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

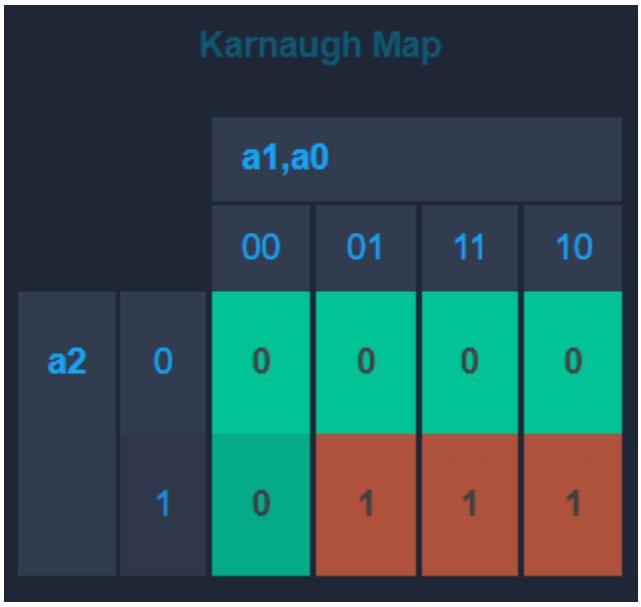
Karnaugh Map for Output b2:



Truth Table Representation of b2:

Binary Input (a2	**a1**	**a0)**	**b2**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Karnaugh Map for Output b3:



Truth Table Representation of b3:

Binary Input (a2	a1	a0)	b3
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Analyzing the results we can reduce equations to the following

1. b0 = a0
2. b1 = a1
3. b2 = a0 ^ a2
4. b3 = a1 ^ a2

3. **Equations Tested**

Maxima Code:

```
INPUT

/* Define the Boolean functions */
b0(a0, a1, a2) := a0;
b1(a0, a1, a2) := a1;
b2(a0, a1, a2) := a0 and a2;
b3(a0, a1, a2) := a1 and a2;

/* Test each combination and display the results */
print("For (0,0,0) -> ", [b0(0,0,0), b1(0,0,0), b2(0,0,0), b3(0,0,0)]);
print("For (0,0,1) -> ", [b0(0,0,1), b1(0,0,1), b2(0,0,1), b3(0,0,1)]);
print("For (0,1,0) -> ", [b0(0,1,0), b1(0,1,0), b2(0,1,0), b3(0,1,0)]);
print("For (0,1,1) -> ", [b0(0,1,1), b1(0,1,1), b2(0,1,1), b3(0,1,1)]);
print("For (1,0,0) -> ", [b0(1,0,0), b1(1,0,0), b2(1,0,0), b3(1,0,0)]);
print("For (1,0,1) -> ", [b0(1,0,1), b1(1,0,1), b2(1,0,1), b3(1,0,1)]);
print("For (1,1,0) -> ", [b0(1,1,0), b1(1,1,0), b2(1,1,0), b3(1,1,0)]);
print("For (1,1,1) -> ", [b0(1,1,1), b1(1,1,1), b2(1,1,1), b3(1,1,1)]);
```

```
OUTPUT

(%i1)

/* Define the Boolean functions */
b0(a0, a1, a2) := a0;
(%o1) b0(a0,a1,a2):=a0;

(%i2)

b1(a0, a1, a2) := a1;


(%o2) b1(a0,a1,a2):=a1;

(%i3)

b2(a0, a1, a2) := a0 and a2;


(%o3) b2(a0,a1,a2):=a0anda2;

(%i4)

b3(a0, a1, a2) := a1 and a2;


(%o4) b3(a0,a1,a2):=a1anda2;

(%i5)

/* Test each combination and display the results */
print("For (0,0,0) -> ", [b0(0,0,0), b1(0,0,0), b2(0,0,0), b3(0,0,0)]);

For (0,0,0) ->  [0, 0, 0 and 0, 0 and 0]
(%o5) [0,0,0∧0,0∧0]

(%i6)

print("For (0,0,1) -> ", [b0(0,0,1), b1(0,0,1), b2(0,0,1), b3(0,0,1)]);
For (0,0,1) ->  [0, 0, 0 and 1, 0 and 1]
(%o6) [0,0,0∧1,0∧1]

(%i7)
```

```
print("For (0,1,0) -> ", [b0(0,1,0), b1(0,1,0), b2(0,1,0), b3(0,1,0)]);
For (0,1,0) ->  [0, 1, 0 and 0, 1 and 0]
(%o7) [0,1,0∧0,1∧0]

(%i8)

print("For (0,1,1) -> ", [b0(0,1,1), b1(0,1,1), b2(0,1,1), b3(0,1,1)]);
For (0,1,1) ->  [0, 1, 0 and 1, 1 and 1]
(%o8) [0,1,0∧1,1∧1]

(%i9)

print("For (1,0,0) -> ", [b0(1,0,0), b1(1,0,0), b2(1,0,0), b3(1,0,0)]);
For (1,0,0) ->  [1, 0, 1 and 0, 0 and 0]
(%o9) [1,0,1∧0,0∧0]

(%i10)

print("For (1,0,1) -> ", [b0(1,0,1), b1(1,0,1), b2(1,0,1), b3(1,0,1)]);
For (1,0,1) ->  [1, 0, 1 and 1, 0 and 1]
(%o10)      [1,0,1∧1,0∧1]

(%i11)

print("For (1,1,0) -> ", [b0(1,1,0), b1(1,1,0), b2(1,1,0), b3(1,1,0)]);
For (1,1,0) ->  [1, 1, 1 and 0, 1 and 0]
(%o11)      [1,1,1∧0,1∧0]

(%i12)

print("For (1,1,1) -> ", [b0(1,1,1), b1(1,1,1), b2(1,1,1), b3(1,1,1)]);
For (1,1,1) ->  [1, 1, 1 and 1, 1 and 1]
(%o12)      [1,1,1∧1,1∧1]
```

The logic equations have been tested using Maxima, a computer algebra system. All equations passed the tests with expected outcomes.

Analysis:

B0 = a0: When a0 is 1, b0 is 1. This is consistent with the truth table
b1=a1: When a1 is 1, b1 is 1. This is consistent with the truth table
B2 = a0 ^ a2. For the AND operation, the output is true only when both inputs are true.

000 -> b2 = 0 (Matches the table)

001 -> b2 = 0 (Matches the table)

010 -> b2 = 0 (Matches the table)

011 -> b2 = 0 (Matches the table)

100 -> b2 = 0 (Matches the table)

101 -> b2 = 1 (Matches the table)

110 -> b2 = 0 (Matches the table)

111 -> b2 = 1 (Matches the table)

Therefore this equation matches the original truth table for b2 for every combination of inputs.

B3 = a1 ^ a2:

For the AND operation, the output is true only when both inputs are true.

The results for this equation match the original truth table for b3 for every combination of inputs.

4. **Summary Statement**

In Phase 2 of our project, we aimed to determine the most concise Boolean expressions to represent the outputs of a Fibonacci sequence generator with binary inputs. The system consisted of three binary inputs, a0, a1, and a2, and four binary outputs, b0, b1, b2, and b3.

Our initial step was to lay out the full truth table for each output based on the Fibonacci sequence. The truth table elucidated how the binary inputs mapped to the corresponding Fibonacci sequence values in binary output form.

With the truth table at our disposal, we moved on to the Karnaugh map method, a graphical tool utilized for simplifying Boolean functions. Four distinct Karnaugh maps were meticulously constructed, one for each output.

Subsequently, from the Karnaugh maps, we derived the simplest Boolean expressions for each output:

b0 = a0

b1 = a1

b2 = a0 AND a2

b3 = a1 AND a2

To ensure the correctness of our derived Boolean expressions I used Maxima. Through Maxima, we inputted and evaluated each Boolean expression for every potential combination of the inputs a0, a1, and a2. The results from Maxima were then cross-verified with our initial truth table to guarantee consistency and correctness.

In conclusion, through the systematic application of the Karnaugh map technique and computational verification using Maxima, we are confident in the accuracy and simplicity of the Boolean expressions derived for the Fibonacci sequence generator's outputs in Phase 2.

5. **Phase 1**

A well-defined algorithm for calculating Fibonacci numbers ensures consistent and accurate outputs. Phase 1 was attached to the top and edited to ensure correctness and validity.

CHECKLIST FOR SUBMISSION

A. General:

1. Professionally presented: ✔

Neat, etc.

2. Properly identified: ✔

e.g., name, id

3. On time: ✔

At the beginning of lab with checklist

4. Good grammar: ✔

e.g., complete sentences where required

5. Correct spelling ✔

B. Content:

1. All outputs analyzed: ✔

Truth table

2. Karnaugh maps (or equations): ✔

Match truth table

3. Equations tested: ✔

Maple, Maxima, etc.

4. Summary statement: ✔

All equations pass all tests

5. Previous phase included: ✔