Course Project Report

Course Name: ESS 102, Digital Design
Student Name: Abhirath Adamane, Lesin
Student ID: (IMT2023029), (IMT2023565)
Group ID: Group-43

30 November 2023

Abstract

This project aims to understand and apply core digital design concepts through the creation of a ”Date to Day Converter” [1] and a ”Serial Adder”. The serial adder, a critical component in digital arithmetic, aims to deliver fast and accurate addition operations, enhancing computational efficiency for a broad spectrum of applications in digital systems. The creation of a date-to-day converter helps to swiftly and accurately determine the corresponding day of the week for a given calendar date and can be used in applications requiring translation between date representations and days of the week in digital environments.

1 Date to Day Converter

1.1 Description

- The date-to-day converter converts any date in 2023 such as January 31 into its respective day such as Tuesday.
- It takes the month as a 4-bit input (for example 0011(m 3:0) is March) and the date as a 5-bit input (for example 01000(d 4:0) is 8th).
- The circuit first checks if the month and date are valid (31st April is an example of an invalid input). If the user input is invalid then the “Erroneous input” (e) output line becomes 1 and the day output is set to 000 (w 2:0).
- When a valid input is entepurple the “Erroneous input” (e) output line becomes 0 and the day output is set to represent a day using a 3-bit output bus.
- The output representation is shown in the table below.

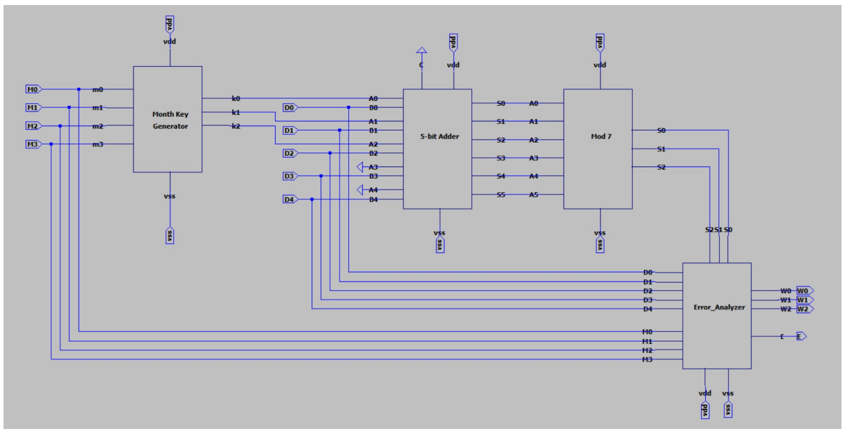
binary output	date it represents
000	Sunday
001	Monday
010	Tuesday
011	Wednesday
....	......

1.2 Implementation

- 1. The month input is fed into a sub-circuit called the “Month Key Generator” which takes a 4-bit input and gives a 3-bit output which represents the number of excess days up till the start of that month (in other words it is the day before the first of a given month).
- 2. With the help of a “5-bit Adder” we add the user-input date to the generated “month key”. This gives a 6-bit output (including the final carry).

- 3. The 6-bit input is sent to a sub-circuit called “Mod 7” which calculates input%7 and gives a 3-bit output between 000 to 110 (inclusive).
- 4. The user-input month and date along with the output is sent into a sub-circuit called Error Analyzer which checks if the input is valid and passes the output of the “Mod 7” or gives a high “Erroneous input ” as needed.

1.3 Circuits Used

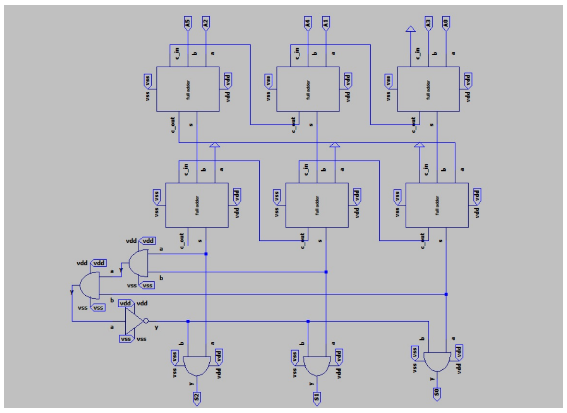


1.3.1 MOD 7 Converter:

This works on the principle that a 6-bit binary number (abcdef) has the same remainder when divided by 7 as a 4-bit binary number (wxyz = abc + def) has with 7.

This can be mathematically proven by writing the 6-bit number as 8*m+n where m and n are 3-bit numbers. We can see that 8*m+n can be written as 7*m + m + n so the remainder of 8*m+n when divided by 7 is the same as the remainder given by m+n.

Hence this 6-bit input can be converted into a 3-bit remainder with the help of a 3-bit Adder and a K-map.



1.3.2 5-Bit Adder:

This is a standard 5-bit Ripple Carry Adder.

1.3.3 Error Analyzer:

This looks at 4 cases:

- When the month is 0000.
- When the day is 00000.
- When 11111 (31) is given as a date input for a month like 0100 (April) with atmost 30 days.

- When the date input is greater than 11100 (28) for 0010 (February). It is constructed with a few K-maps and a logic similar to a multiplexer can be used for displaying the final output.

1.3.4 Month Key Generator:

In this sub-circuit, we give 4-bit input as mentioned and it returns a 3-bit key. This can be constructed using 3 K Maps.

month (input)	key (output)
0001	110 (6 excess days)
0010	010 (2 excess days)
0011	010 (2 excess days)
....	......

2 Sequential Circuit: Serial Adder

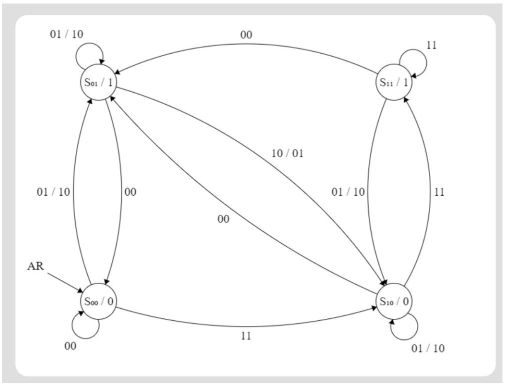
2.1 Description

A serial adder is a digital circuit that adds two binary numbers sequentially, bit by bit. It's commonly used in applications where space or resources are limited, as it requires fewer components compared to parallel adders. Serial adders are often found in low-power devices, simple processors, or applications where speed isn't a critical factor. [2]

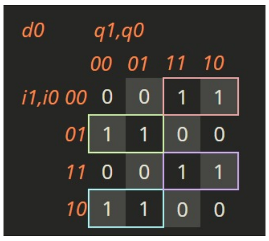
2.2 Implementation

- Initialization: The adder receives the binary numbers to be added, usually stored in registers. The initial carry bit is set to zero. In our case we are taking the input bit by bit in the function inputSchedule.
- Bit-by-bit addition: Starting from the least significant bit (LSB), the adder sequentially adds the bits of both numbers along with the carry from the previous addition. The sum and a carry-out are generated for each bit. This is performed by the NSL .
- Carry propagation: The carry-out from each bit addition becomes the carry-in for the next higher bit position. This data is held in the registers in the form of ”ns” and ”ps”.
- Propagation through clock cycles: This sequential process continues until all bits have been added, propagating carries through clock cycles. The simpy module takes care of the simulation aspect of this FSM.
- Result: Once all bits are processed, the final sum is obtained, stored in an output register or buffer. In our case we use string slicing to give the appearance of an output buffer for better visualization.

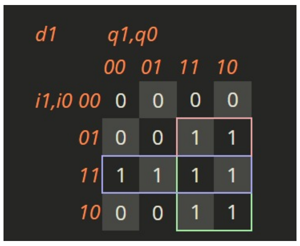
2.3 State Diagram, Logic Equations and K-Maps



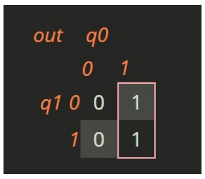
[3]



d0	q1,q0			
	00	01	11	10
i1,i0 00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

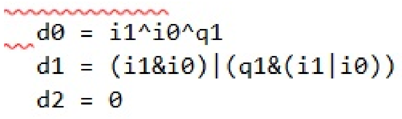


d1	q1,q0			
	00	01	11	10
i1,i0 00	0	0	0	0
01	0	0	1	1
11	1	1	1	1
10	0	0	1	1



out	q0	
	0	1
q1 0	0	1
1	0	1

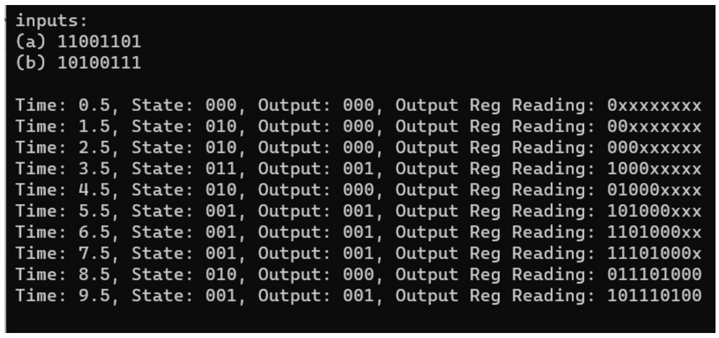
[4]



```
d0 = i1^i0^q1
d1 = (i1&i0)|(q1&(i1|i0))
d2 = 0
```

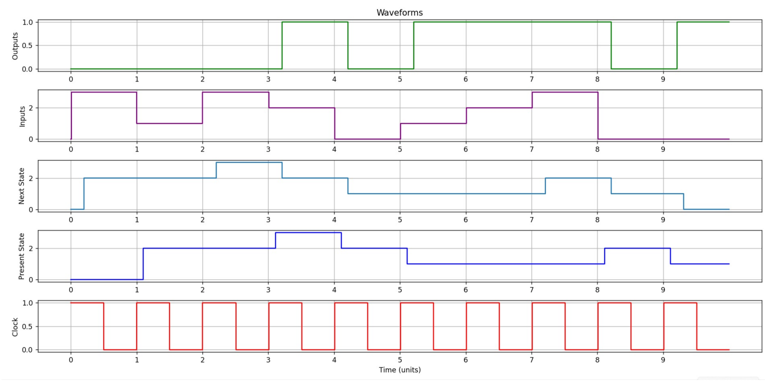
2.4 Results

We have tested the circuit against multiple streams of inputs of different sizes. The output is accurate. An example is:



```
inputs:
(a) 11001101
(b) 10100111

Time: 0.5, State: 000, Output: 000, Output Reg Reading: 0xxxxxxxx
Time: 1.5, State: 010, Output: 000, Output Reg Reading: 00xxxxxxx
Time: 2.5, State: 010, Output: 000, Output Reg Reading: 000xxxxxx
Time: 3.5, State: 011, Output: 001, Output Reg Reading: 1000xxxxx
Time: 4.5, State: 010, Output: 000, Output Reg Reading: 01000xxxx
Time: 5.5, State: 001, Output: 001, Output Reg Reading: 101000xxx
Time: 6.5, State: 001, Output: 001, Output Reg Reading: 1101000xx
Time: 7.5, State: 001, Output: 001, Output Reg Reading: 11101000x
Time: 8.5, State: 010, Output: 000, Output Reg Reading: 011101000
Time: 9.5, State: 001, Output: 001, Output Reg Reading: 101110100
```



References

[1] [Online]. Available: https://www.hrpub.org/download/20200930/MS14-13417288.pdf

[2] Mar 2022. [Online]. Available: https://en.m.wikipedia.org/wiki/Serial$_{b}inary_{a}dder$

[3] [Online]. Available: https://madebyevan.com/fsm/

[4] [Online]. Available: https://www.charlie-coleman.com/experiments/kmap/