Overview of the design components:

Submitted by:

Or Shaul

Noa Farkash

Stas Kogan

Idan Levy

Our design contains three main stages:

1. Input stage:

1.1. **Registers:** The input stage consists of two 4 bit registers that accept an input A [3:0] and B [3:0] and another register of 5 bit that will accept the OP Code from the user to determine the function to be performed by the ALU.
1.2. **Feedback:** At the input of the register storing A, there is a multiplexer which can choose between a User-supplied input and the output Y from the previous clock cycle.
1.2.1. **Mux 2:1:** Implements the feedback required, selects between the user input or Y output to be promoted to the logic stage, the Mux's decision is based on the Opcode[0] bit: $\begin{cases} 1, & A\ from\ user \\ 0, & Y\ from\ Output \end{cases}$
1.3. **2x DMux 4:1:** This stage also contains two DMUX gates which are using Opcode [3:4] to select between the logical stage we want to implement such that the input of each one is A or B only for the relevant stage, otherwise the input of the stage is zero, the purpose is to reduce dynamic power as required in the spec.

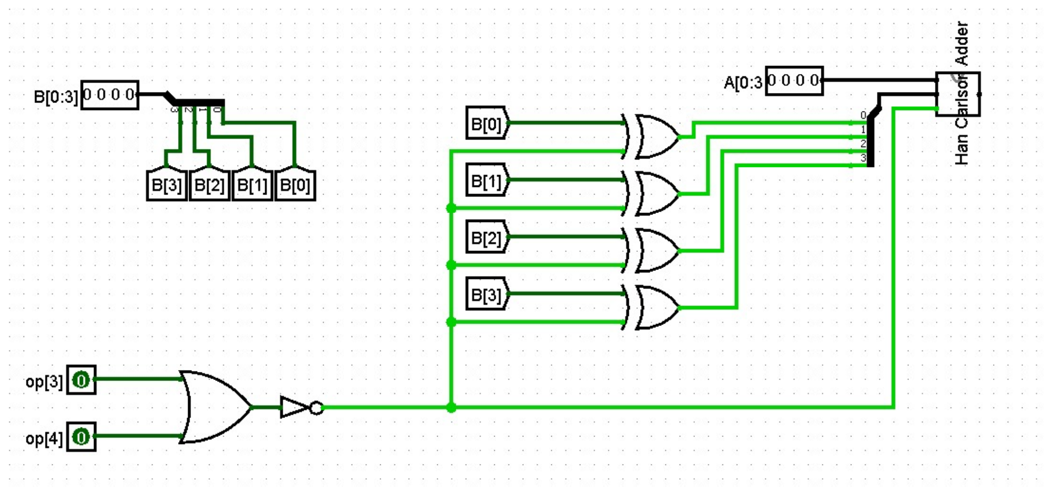
2. Function stage:

2.1. **Bitwise XOR:** Contains four XOR gates that will implement the bitwise XOR between two inputs that we are required to do in the spec.
2.2. **ADD/SUB:** In our design we will use the Han-Carlson adder the control of the function we will want to implement will be by a NOR gate that will receive Opcode [4] and Opcode [3] and decide the function according to his output in the following way: $opcede\ [3] = \begin{cases} 0, & SUB \\ 1, & ADD \end{cases}$
This output bit will also be the carry in to the adder.
The output bit of the NOR gate will also go to be one of two inputs to four XOR gates that will implement bitwise XOR with the input B.

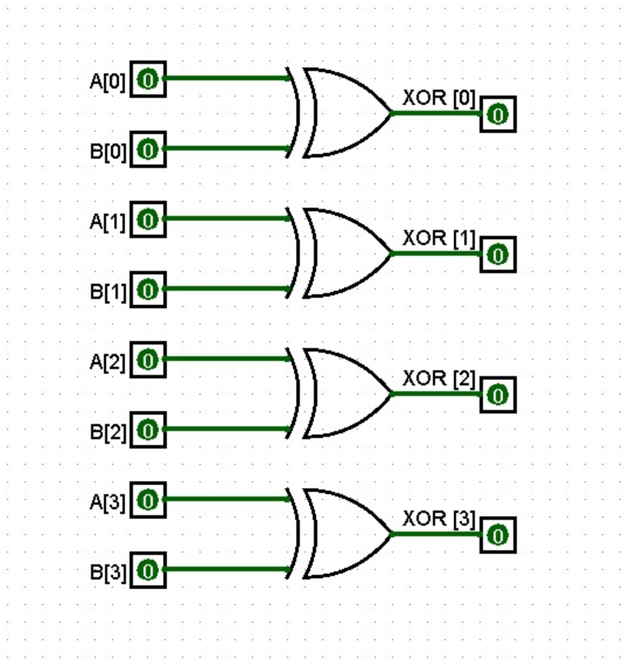
The Han-Carlson adder will receive input A,B and $Carry_{in}$.
2.3. **Barrel –Shift – Right (BSR):** This block receives two input A and i and preforms right shift to register A i times where i is {1,2,3,4}. This block will be designed using four 4:1 Mux to perform the shift of the register and will be controlled by Opcode [2:1].

3. Output Stage:
3.1. **4:1 Mux**: Gets 4 inputs (ADD/SUB result, Bitwise XOR result, BSR result and one that will be shorted to ADD/SUB). The mux is controlled by two select bits of Opcode [4:3].
3.2. **Registers:** The output stage consists of one four-bit register that accept the four-bit output as an input and holds it and sends it to the feedback block in the input.

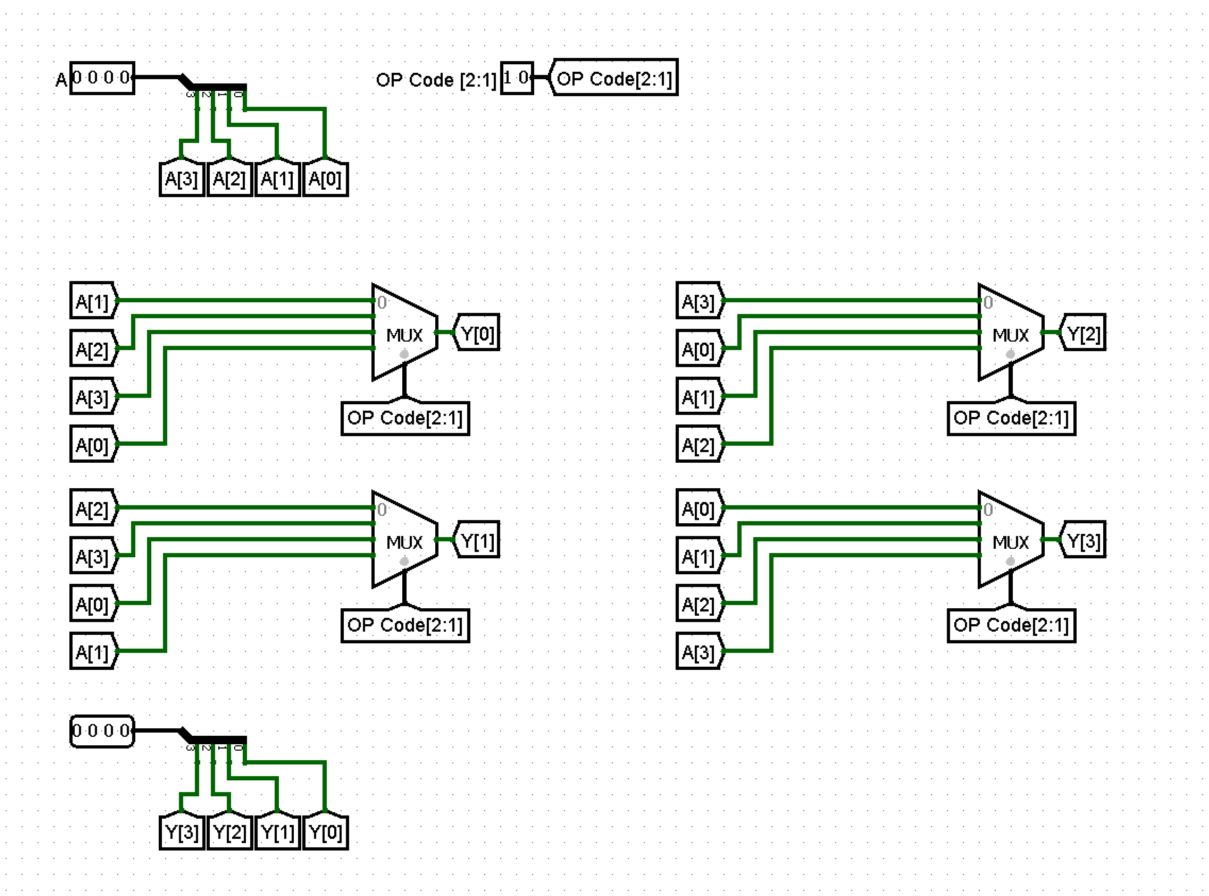
Design of the ADD\SUB stage:



Bitwise XOR:



BSR:


A
0 0 0 0
A[3]
A[2]
A[1]
A[0]
OP Code [2:1]
1 0
OP Code[2:1]
MUX
Y[0]
Y[1]
Y[2]
Y[3]
0 0 0 0
Y[3]
Y[2]
Y[1]
Y[0]

<u>Opcode Implementation:</u>

We divided the 5-bit opcode into 3 sections, each representing a different type of configuration:

- Opcode[0] - This bit represents the 1st input to the ALU:
  - If Opcode[0] = 0, we feedback the output into the input register A
  - If Opcode[0] = 1, we take the input from the user and feed it to the input register A.
- Opcode[2:1] – BSR number of shifts
- Opcode[4:3] – select between operations: arithmetic - Add/Sub, Logic: BSR, XOR.

Operation	Opcode[4]	Opcode[3]
ADD	0	0
SUB	0	1
XOR	1	0
BSR	1	1

- Opcode[2:1] - represents the number of shifts the BSR should do:
  - Opcode[2:1] = 00 => Shift by 1
  - Opcode[2:1] = 01 => Shift by 2
  - Opcode[2:1] = 10 => Shift by 3
  - Opcode[2:1] = 11 => Shift by 4 (no shifting actually)

Explicit opcodes representation:

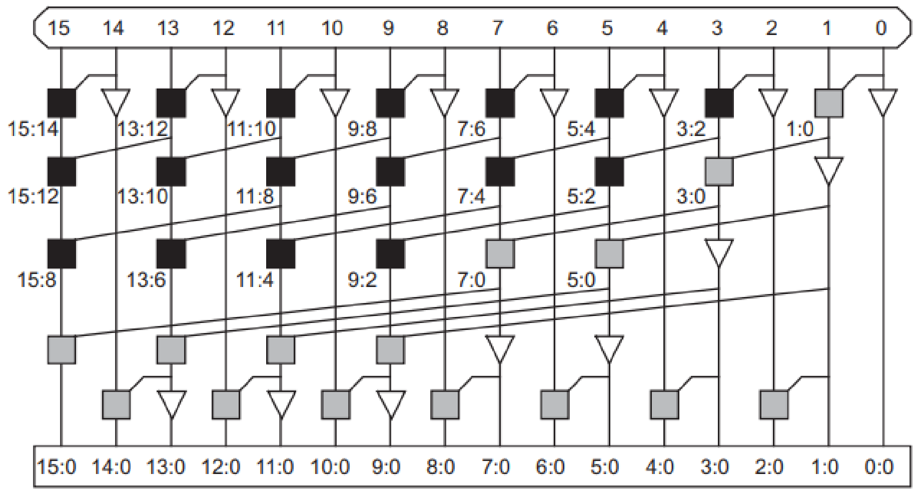
Operation	Opcode[4]	Opcode[3]	Opcode[2]	Opcode[1]	Opcode[0]
Output **ADD** B	0	0	X	X	0
Output **SUB** B	0	1	X	X	0
Output **XOR** B	1	0	X	X	0
Output **BSR** (1)	1	1	0	0	0
Output **BSR** (2)	1	1	0	1	0
Output **BSR** (3)	1	1	1	0	0
Output **BSR** (4)	1	1	1	1	0
A **ADD** B	0	0	X	X	1
A **SUB** B	0	1	X	X	1
A **XOR** B	1	0	X	X	1
A **BSR** (1)	1	1	0	0	1
A **BSR** (2)	1	1	0	1	1
A **BSR** (3)	1	1	1	0	1
A **BSR** (4)	1	1	1	1	1

Han-Carlson adder:

The Han-Carlson Adder is a type of adder circuit used in digital electronics for performing addition operations. It was proposed by C.K. Han and T.A. Carlson in their paper titled "High-speed VLSI adders using carry lookahead techniques" published in 1986.

The Han-Carlson Adder utilizes a combination of carry lookahead techniques and parallel processing to achieve high-speed addition operations. It is particularly notable for its efficiency in terms of both speed and power consumption compared to other adder designs.

One of the key features of the Han-Carlson Adder is its use of carry-save adders (CSAs) in combination with carry lookahead logic. This allows for parallel processing of the carry propagation, which significantly reduces the propagation delay associated with carry propagation in traditional adder designs.



(d) Han-Carlson

In traditional binary addition, a carry bit is propagated through each bit position, causing a serial chain of carry propagation. This adder reduces the delay in the carry propagation by introducing a parallel-prefix structure. Faster adders look ahead to predict the carry-out of a multibit group. This is usually done by computing group PG signals to indicate whether the multibit group will propagate a carry-in or will generate a carry-out.

Long adders use multiple levels of lookahead structures for even more speed.The adder is composed of several stages, each of which performs a specific function. At each stage, the bits of the operands are divided into groups, and within each group, a prefix operation is performed to generate intermediate carry values. These intermediate carries are then used in the next stage to compute the final sum and carry bits.

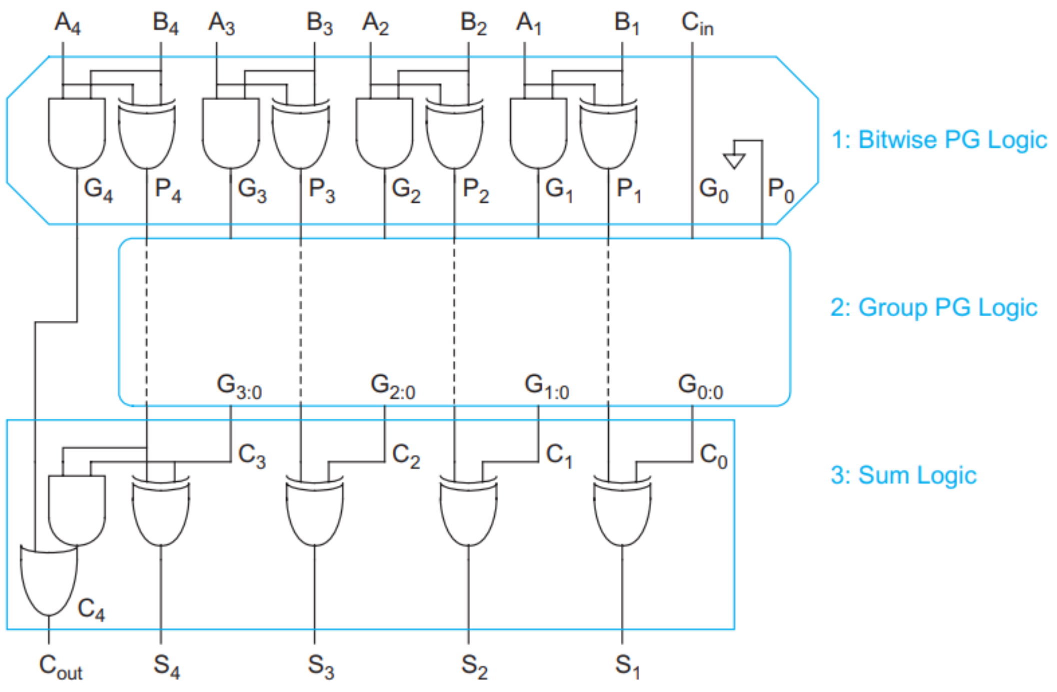
Addition can be reduced to a three-step process:

1. Computing bitwise generate and propagate signals.

2. Combining PG signals to determine group generates.

3. Calculating the sums.

These steps are illustrated in Figure 11.12. (taken from the course book) The first and third steps are routine, so most of the attention in the remainder of this section is devoted to alternatives for the group PG logic with different trade-offs between speed, area and complexity.



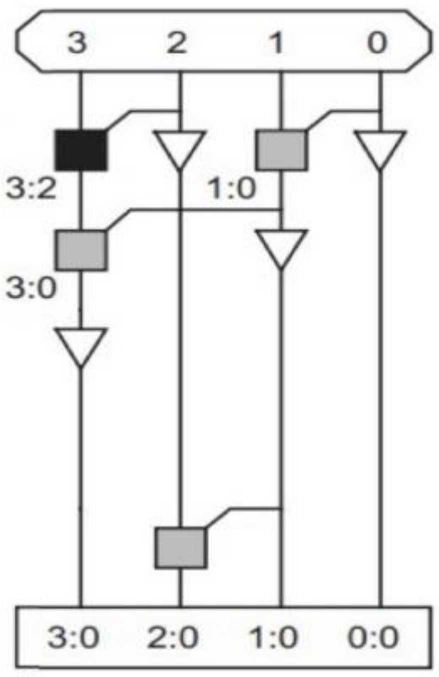
***In our implementation we will not use $C_4, G_4 and\ C_{out}$ as shown in the Figure above. The reason is we have registers A, B and Y with length 4 only.**

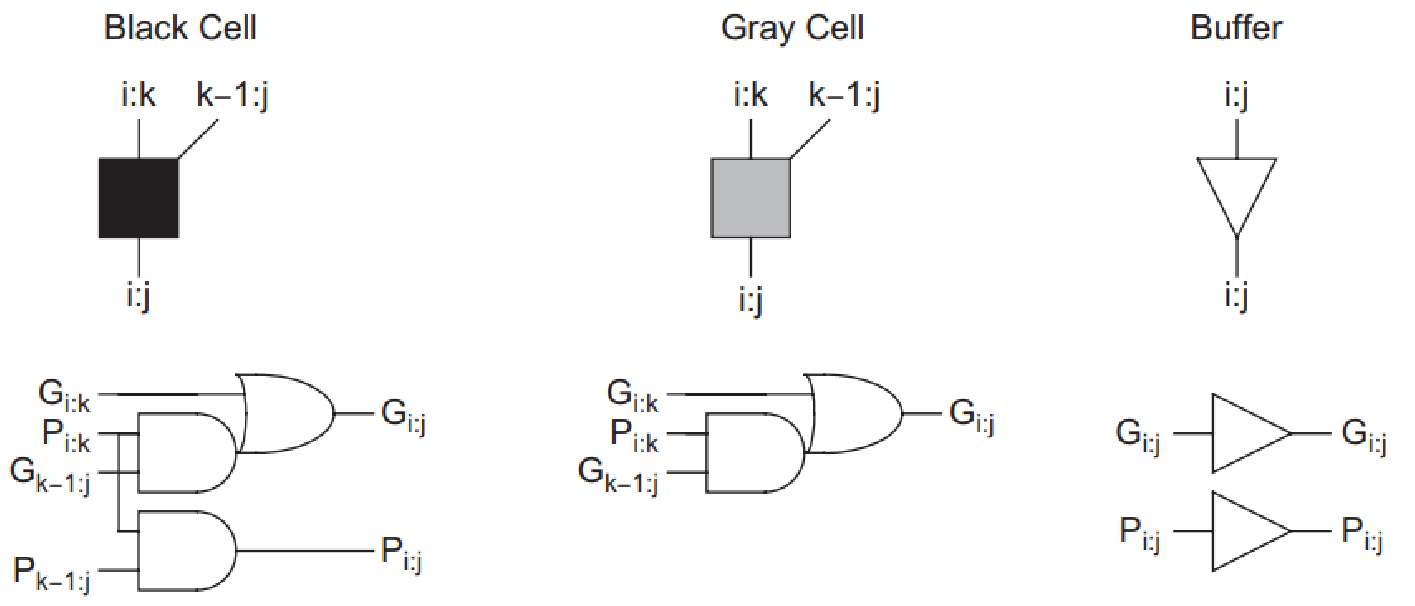
This adder takes advantage of the fact that the carry bits can be computed in parallel using prefix computations. This allows for faster carry propagation and reduces the overall delay in the addition process. By distributing the carry computation across multiple stages, the adder can achieve better performance compared to a traditional ripple-carry adder.

The main advantage of the Han-Carlson adder is its ability to perform parallel addition with reduced delay. However, it typically requires more complex circuitry compared to a simple ripple-carry adder. The trade-off between speed and complexity depends on the specific requirements of the application.

Overall, the Han-Carlson adder is a parallel-prefix adder design that improves the efficiency of binary addition, an important contribution to the field of digital circuit design, especially in the realm of high-speed arithmetic operations, where minimizing delay and power consumption are critical considerations.

Group 2 of PG Logic is implemented using the following scheme:





Gates sizing:

Gate	Length	Hight	Total size
AND(2:2)	1.12μm	1.9μm	$2.128\mu m^2$
OR	0.92μm	1.9μm	$1.748\mu m^2$
XOR	1.72μm	1.9μm	$3.268\mu m^2$
INV	0.52μm	1.9μm	$0.988\mu m^2$
MUX 4:1	4.52μm	1.9μm	$8.588\mu m^2$
D-FF	3.52μm	1.9μm	$6.688\mu m^2$
MUX 2:1	1.52μm	1.9μm	$2.888\mu m^2$
AND 3:1	1.52μm	1.9μm	$2.888\mu m^2$
BUFFER	1.12μm	1.9μm	$2.128\mu m^2$

Input Stage:

PART	SIZE	Register 4 bit	OPCODE Register	MUX2X1 4 bit	DEMUX 4X16
AND 2X2	2.128				3x4x4=48
INV	0.988				2x4=8
OR	1.748				
XOR	3.268				
DFF	6.688	4	5		
MUX 4:1	8.588				
MUX 2:1	2.888			4	
AND 3:1	2.888				
TOTAL SIZE		26.752	33.44	11.552	110.048

Function Stage:

PART	SIZE	BSR	Bitwise XOR	ADDER				
				Bitwise XOR	Bitwise PG	Gray Cell	Black Cell	Buffer
AND 2X2	2.128				1x3	1x3=3	2	
INV	0.988							
OR	1.748					1x3=3	1	
XOR	3.268		4	4x2=8	1x3+1=4			

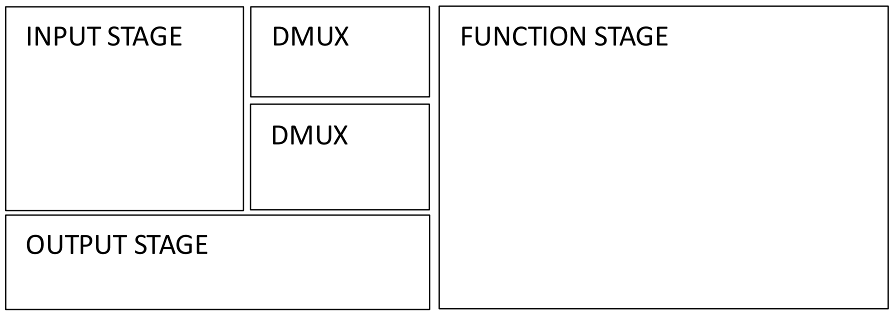
DFF	6.688							
MUX 4:1	8.588	4						
MUX 2:1	2.888							
AND 3:1	2.888							
BUFFER	2.128							5
TOTAL SIZE		34.352	13.072	26.144	19.456	11.628	6.004	10.64
				73.872				

Output Stage:

PART	SIZE	Register 4 bit	MUX 4X1 4 bit
AND 2X2	2.128		
INV	0.988		
OR	1.748		
XOR	3.268		
DFF	6.688	4	
MUX 4:1	8.588		4
MUX 2:1	2.888		
AND 3:1	2.888		
TOTAL SIZE		26.752	34.352

Alu:

PART	SIZE	Total	TOTAL SIZE
Register 4 bit	26.752	3	80.256
OPCODE Register	33.44	1	33.44
Mux 2x1	11.552	3	34.656
Demux 4x16	110.048	2	220.096
Adder	73.872	1	73.872
Bitwise XOR	13.072	1	13.072
BSR	34.352	1	34.352
MUX 4x1	34.352	1	34.352
TOTAL SIZE			524.096


INPUT STAGE
DMUX
DMUX
FUNCTION STAGE
OUTPUT STAGE