Design of a 32 4-bit SRAM memory module

***Abstract*—This paper discusses a detailed overview of the process involved in designing a Static Random Access Memory (SRAM) array. The SRAM array comprises a bit-cell, and peripheral circuitry which is a combination of row decoder circuitry to select the word lines for the memory array, and a column decoder circuitry which comprises a column decoder and column decoder mux. An array implementation of a 32 by 4 SRAM has been demonstrated which corresponds to a 32 4-bit array. The process node used in the implementation is a 65nm TSMC process.**

Keywords— Bit-cell, memory array, row decoder, column decoder, mux.

I. Introduction

Memories have been an integral part of digital electronics enabling the storing and retrieving of data efficiently and with speed. The memories can be classified as either volatile (RAM) or non-volatile (ROM). There exists a vast variety of types of volatile and non-volatile memories but this paper majorly focuses on the Static Random Access Memory (SRAM). The memory core can be arranged in a wide variety of different configurations. Among the diverse array configurations, the 32x4-bit memory array stands as a versatile solution, balancing storage capacity with accessibility.

A 32x4-bit memory array comprises 32-word lines of 4 bits each. The memory module comprises a memory core along with peripheral circuitry which involves pre-charge circuitry for bit line conditioning, a sense amplifier for sensing the difference in the bit lines for a successful read operation, and row and column decoder circuitry to successfully select the desired address lines and bit-cell for the data read and write operation

In terms of applications, the 32x4-bit memory array finds utility in a plethora of digital systems, including microcontrollers, field-programmable gate arrays (FPGAs), and system-on-chip (SoC) devices. It serves as a critical component for storing program instructions, data buffers, and configuration settings, enabling efficient operation and data processing in these systems.

In summary, the 32x4-bit memory array represents a versatile and efficient solution for storing and accessing digital data in a wide range of applications. Its compact architecture, high data density, and flexibility make it an indispensable component in modern digital electronics, powering diverse computing and communication systems.

The paper is structured in the following manner: section II provides an in-depth analysis of the design methodologies for the SRAM, including the design of row decoder required to select the address lines. The layout of the row decoder is done using skill code. Section III shows the design and layout of the rest of the peripheral circuitry to be used in the memory module, mainly the column decoder circuitry, followed by the layout design of the memory module. Section IV shows the complete design schematic and layout of the SRAM memory module. Conclusions are drawn in Section V. The design and layout are done using TSMC 65nm process node.

II. Row Decoder Circuitry

For the row circuitry, we have designed a 5-to-32 decoder for the 32 4-bit word of the SRAM array. The row decoder includes 5-input AND gates to access these word lines. The schematic of the AND gate is shown in Fig. 1. The Skill code is used to design the layout of the AND gate shown in Fig. 2 and the designed code is divided into sections for easier implementation as shown;

The first section shows how the database is loaded into the cadence environment,

```
    cv = dbOpenCellViewByType("IC_Design_Lab_Proj2"
"AND5_Skill_Code" "layout" "" "a")
    tf=techGetTechFile(cv) ; get info from attached tech library
    ; Dimensions are in microns
    l = .10; Length of transistors. Must be float!
    w = .20; Width of transistors
    for(i 0 5
    x=i*(2*l+1.0); place the transistors
    dbCreateParamInstByMasterName(
    cv ; the target cellview's dbId
    "tsmcN65" ; the library where the instance comes from
    "pch" ; the name of the cell to be instantiated
    "layout" ; the view of the cell to be instantiated
    symbolToString(gensym("I")) ; generates unique instance name
    x:0 ; Location of transistor
    "R0" ; orientation
    1 ; number of instances
    list(list("wff" "float" 2) list("l" "string" sprintf(nil "%fu" l))) ; the
parameters, types and values
    )
    )
```

The second section enables us to place the transistors in the desired positions.

```
    for(i 0 5
    y=i*(2*l+1.0); place the transistors
    dbCreateParamInstByMasterName(
    cv ; the target cellview's dbId
    "tsmcN65" ; the library where the instance comes from
    "nch" ; the name of the cell to be instantiated
    "layout" ; the view of the cell to be instantiated
    symbolToString(gensym("I")) ; generates unique instance name
    y:-2 ; Location of transistor
    "R0" ; orientation
    1 ; number of instances
    list(list("wff" "float" 2) list("l" "string" sprintf(nil "%fu" l))) ; the
parameters, types and values
    )
    )
    dbCreateRect(cv list( "M1" "drawing" ) list( -0.4 : .8 (2*l+6.3) : 1 ) )
    dbCreateRect(cv list( "M1" "drawing" ) list( -0.4 : -2.84 (2*l+6.3) : -2.64
) )
```

The third section allows us to extend the poly and metal connections for routing.

```
for(i 0 5
x0 = i*1.2
dbCreateRect(cv list( "PO" "drawing" ) list( x0 : -2.5 x0+.1 : .515) )
)

for(i 0 5
x0 = i*1.2-0.145
dbCreateRect(cv list( "M1" "drawing" ) list( x0 : -.1 x0+.09 : 1) )
)

for(i 0 3
x0 = i*1.2+0.155
dbCreateRect(cv list( "M1" "drawing" ) list( x0 : -2.01 x0+.99 : -1.79) )
)
dbCreateRect(cv list( "M1" "drawing" ) list(-.145: -2.84 -0.055 : -1.79) )
dbCreateRect(cv list( "M1" "drawing" ) list(5.855: -2.84  5.945 : -1.79) )
for(i 0 1
x0 = i*1.2+4.955
dbCreateRect(cv list( "M1" "drawing" ) list( x0 : -2.01  x0+.09 : .21) )
)
for(i 0 3
x0 = i*1.2+0.155
dbCreateRect(cv list( "M1" "drawing" ) list( x0 : -.2  x0+.09 : -0.01) )
)
dbCreateRect(cv list( "M1" "drawing" ) list( 0.155 : -.29  5.045 : -.2) )
```

The fourth section enables us to connect the routes using vias.

```
viaDef=techFindViaDefByName(tf "M1_PO") ; get info for via from tech library
dbCreateVia(cv viaDef 5.44:-0.94 "R0" list(list("cutRows" 1) list("cutColumns" 1)))
dbCreateRect(cv list( "M1" "drawing" ) list( 4.955 : -1.025 5.525 : -.855) )
dbCreateRect(cv list( "PO" "drawing" ) list( 5.525 : -1.025 6 : -.855) )

viaDef=techFindViaDefByName(tf "M1_SUB") ; get info for via from tech library
dbCreateVia(cv viaDef 2.541:-2.74 "R0" list(list("cutRows" 1) list("cutColumns" 3)))
viaDef=techFindViaDefByName(tf "M1_NWs") ; get info for via from tech library
dbCreateVia(cv viaDef 2.541: .9 "R0" list(list("cutRows" 1) list("cutColumns" 1)))
```

The fifth section completes the layout and is the last section extending the NWELL and connecting the remaining metal connections.

```
dbCreateRect(cv list( "NW" "drawing" ) list( -0.395 : -.5  6.495 : 1.2) )

dbCreateRect(cv list( "M1" "drawing" ) list( -6.5 : -.56  .1 : -.39 ) )
dbCreateRect(cv list( "M1" "drawing" ) list( -6.5 : -.56-.27  .1+1.2 : -.39-.27 ) )
dbCreateRect(cv list( "M1" "drawing" ) list( -6.5 : -.56-2*.27  .1+2*1.2 : -.39-2*.27 ) )
dbCreateRect(cv list( "M1" "drawing" ) list( -6.5 : -.56-3*.27  .1+3*1.2 : -.39-3*.27 ) )
dbCreateRect(cv list( "M1" "drawing" ) list( -6.5 : -.56-4*.27  .1+4*1.2 : -.39-4*.27 ) )

viaDef=techFindViaDefByName(tf "M1_PO") ; get info for via from tech library
for(i 0 4
x0 = i*1.2+0.015
y0 = -i*.27-0.475
dbCreateVia(cv viaDef x0:y0 "R0" list(list("cutRows" 1) list("cutColumns" 1)))
)

dbCreateRect(cv list( "M1" "drawing" ) list( 6.245 : -1.65  7.25 : -1.48 ) )
```

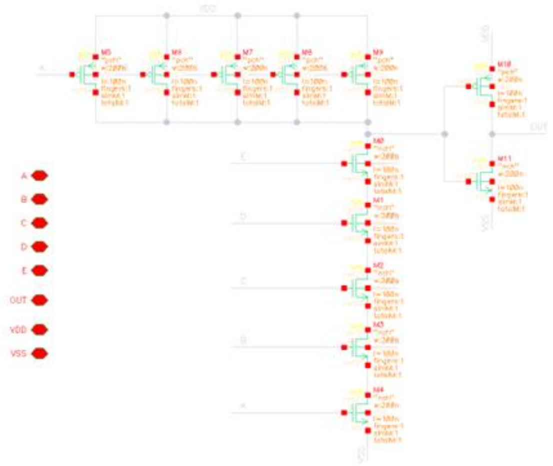

Fig. 1. Schematic of the 5-input AND gate.

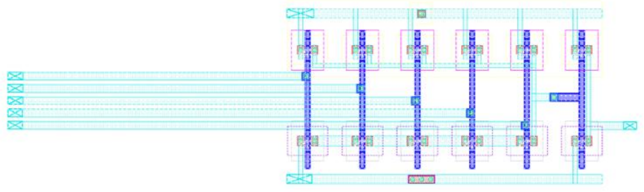
Fig. 2. Layout of the 5-input AND gate using Skill code.

Fig. 3 presents the schematic of the 5-to-32 decoder used in this project. We used the layout of the AND gate generated by the Skill code to design the layout for the decoder circuit. The decoder layout is shown in Fig. 4. The row decoder is pitch-matched with the SRAM array to achieve a rectangular floorplan.

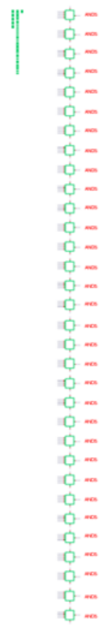
Fig. 3. Schematic of the 5-to-32 decoder.

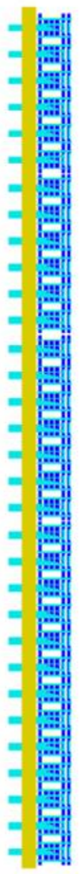

Fig. 4. Layout of the 5-to-32 decoder.

III. Column Decoder Circuitry

The column circuitry comprises a tree decoder mux that uses only NMOS transistors along with no external requirement of a decoder circuitry. This reduces the overall area of the multiplexer and makes it easier to develop. The output of the decoder goes into the sense amplifier that senses the difference in the bit lines and generates an output respectively. The column decoder mux, also known as the tree decoder mux, is designed using individual multiplexers for each column as shown in Fig. 5. The layout of the single column decoder is shown in Fig. 6. These individual multiplexers are then stitched together to connect to the bit-lines of the bit-cells in the four columns of the memory array.

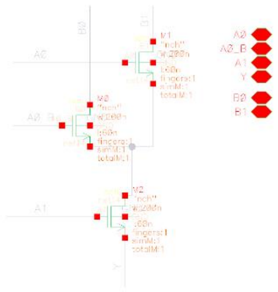

Fig. 5. Schematic diagram of the Column decoder circuitry for a single column in the memory array

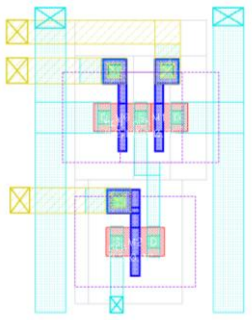

Fig. 6. Layout of the Column decoder circuitry for a single column in the memory array

The developed schematic diagram for the tree decoder mux is shown in Fig. 7. and Fig. 8. shows the designed layout.

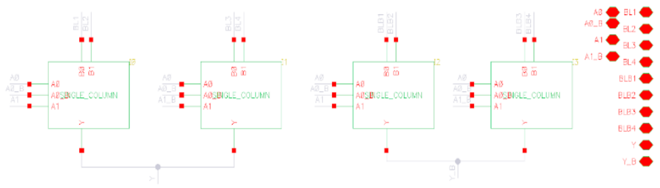

Fig. 7. Schematic diagram of a Column decoder circuit for a 32 4-bit SRAM memory array

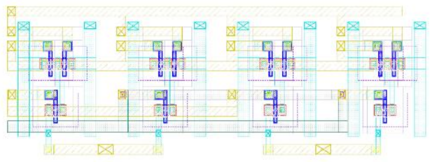

Fig. 8. Layout of the Column decoder circuitry for a 32 4-bit SRAM memory array

The output Y and Y_Bar from the tree decoder mux is fed into the sense amplifier circuit. The sense amplifier plays a crucial role in reading and restoring data from memory cells. SRAM stores data using a cross-coupled latch structure, typically consisting of six transistors per cell. The sense amplifier assists in retrieving the stored data by sensing the small voltage differentials across the memory cell when it's being accessed.

The sense amplifier works in the following ways.

A. Read Operation:

When you want to read data from an SRAM cell, the word line (row address) corresponding to that cell is activated, which in turn connects one of the two bit-lines (column addresses) to the memory cell. Due to the differential storage of data (one bit stored as a voltage level on one of the bit lines), a small voltage differential appears across the cell. This voltage difference is typically very small and needs to be amplified for accurate interpretation.

B. Amplification:

The sense amplifier detects this small voltage differential and amplifies it to a level that can be reliably interpreted as either a logic high or a logic low. It essentially compares the voltages on the bit lines and amplifies the difference.

C. Data Restoration:

After amplification, the sense amplifier drives one of the bit lines to a high voltage level and the other to a low voltage level, based on the amplified signal. This process ensures that the SRAM cell is restored to its original state after the read operation, ready for subsequent access.

The designed sense amplifier is shown in Fig. 9. The layout of the sense amplifier is shown in Fig. 10.

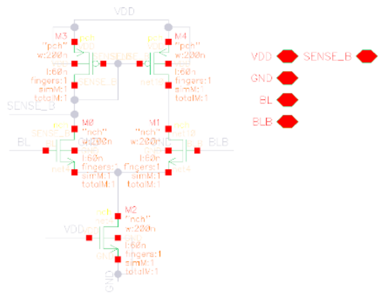

Fig. 9. Schematic diagram of the Sense amplifier

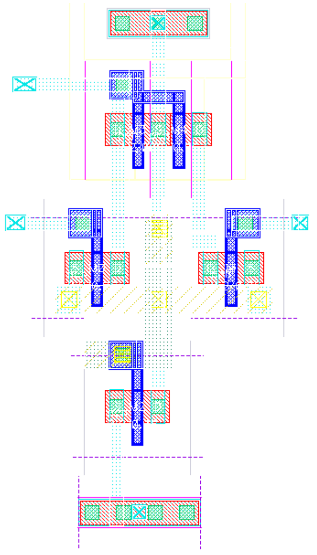

Fig. 10. Layout of the Sense amplifier

The column periphery of the SRAM also consists of a pre-charge circuitry which ensures that the bit-lines are pre-charged to VDD during the read operation through equalization of the bit-lines to minimize the voltage difference when using a sense amplifier. The schematic diagram for the pre-charge circuitry is shown in Fig. 11.

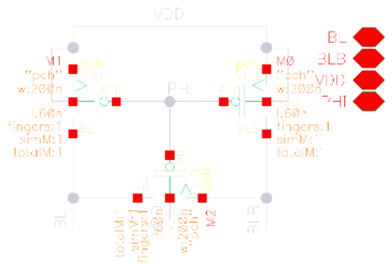

Fig. 11. Schematic diagram of the pre-charge circuit design used for the bit line conditioning.

The layout of the pre-charge circuitry is shown in Fig. 12.

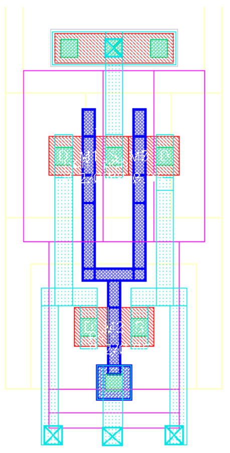

Fig. 12. Layout of the pre-charge circuit design used for the bit line conditioning.

IV. Complete Architecture Of The SRAM Memory Module

The schematic of the complete architecture of the SRAM memory module to successfully read and write the SRAM cells is shown in Fig. 13, which includes pre-charge circuits, a 5-to-32 decoder circuit, an SRAM array, a tree decoder mux, and a sense amplifier. All these sub-circuit layouts are combined to design the layout of the complete SRAM array depicted in Fig. 14.

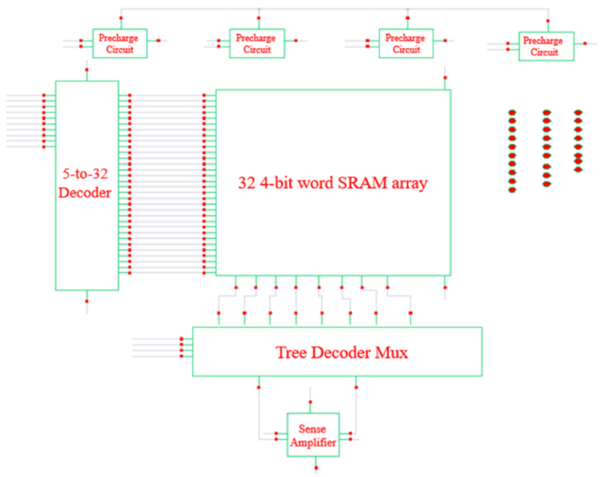

Fig. 13. Schematic of the complete architecture of the SRAM array.

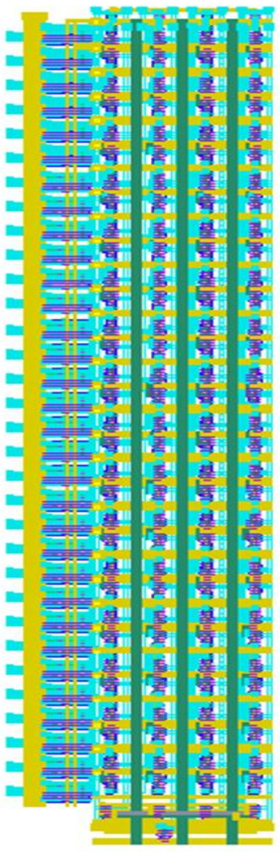

Fig. 14. Layout of the complete architecture of the SRAM array.

V. Conclusion

This project shows a successful design and layout of a 32 4-bit SRAM array along with the required peripheral circuitry for a successful read/write operation designed using TSMC 65nm process node. The memory array is designed to achieve 32 rows each 4-bit wide. The 32 rows/address lines can be selected using a 5-to-32 row decoder designed in section II. For a successful read and write operation suitable column decoder circuitry is designed and laid out which includes pre-charge circuitry for bit-line conditioning, sense amplifier and tree column decoder mux. Skill code is used to generate the layout for the 5-input AND gate, which is used for the pitch-matched row decoder design. Separate layouts are provided for all the supporting circuitry used in this design project. Finally, all the schematic cell views and layouts are combined to demonstrate the complete architecture of the SRAM array.

References

[1] N. H. E. Weste and D. M. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th edition, 2011.

[2] J. Dix, Integrated Circuit Design Lab I: Class Lecture, Spring 2024.

[3] J. Dix, Skill Code Basics – Layout: Spring 2024.