Implementation and Testing of an (8, 4, 4) Viterbi Decoder

Harsh Aurora
260394216
harsh.aurora@mail.mcgill.ca

Adam Cavatassi
260409261
adam.cavatassi@mail.mcgill.ca

Xiaoqing Ma
260668927
xiaoqing.ma@mail.mcgill.ca

Dylan Watts
260406792
dylan.watts@mail.mcgill.ca

***Abstract*—This paper presents the basic knowledge and circuit design for an (8, 4, 4) Viterbi decoder. Two different designs are given to optimize either computation speed or power consumption. The decoder architecture is verified by logic simulation in Modelsim with Verilog. Timing analysis and circuit simulation in IRSIM and SPICE are done to find the computation delay and the amount of power consumed for both designs. The repository and LaTeX code for this report can be accessed by visiting: https://github.com/xiaoqing1993/ECSE548_viterbi_decoder**

I. Introduction

For this project, our goal is to design two different circuits of an (8, 4, 4) Viterbi decoder for different purpose: one is power-efficient while the other runs in a high speed. To eliminate the power consumption, carry-ripple adders are implemented in the first design. In another design, Kogge-Stone adders are introduced to speed up the computation. The schematic and layout designs for all modules and the overall decoder are finished in electric.

Simulation process of this project can be divided into two parts. The first part is logic verification, which has been conducted with Verilog using Modelsim. The second part is circuit simulation for finding delays and power consumptions. This work has been done within MATLAB, IRSIM and LTSPICE.

This report is organized as follow: Section II introduces the background knowledge of Viterbi decoder and section III presents the schematics and layouts of related modules and the overall decoder. Simulation procedures and results are presented in Section IV and V. Conclusions are drawn in Section VI.

II. Background

Error correction codes are employed to counter the effects of the noise introduced by the channel in data transmission. One such coding scheme is the Hamming (8,4,4) code, which is a systematic parity check code in which 4 message bits are used to produce an 8 bit codeword. The first four bits of the codeword are the message bits themselves (systematic), while the remaining four bits (parities) are calculated through XOR operations on the message bits three at a time. The numbers in (8,4,4) then represent the codeword length (8 bits), the number of information bits (4 bits), and the minimum distance of the code (4), which is a parameter that is used to describe the error correction capability of the code. The Hamming (8,4,4) code is constructed using the equations shown below.

$$\begin{aligned}
\text{Message: } & m_0 m_1 m_2 m_3 \\
\text{Codeword: } & c_0 = m_0 \; c_1 = m_1 \; c_2 = m_2 \; c_3 = m_3 \\
& c_4 = m_0 \oplus m_1 \oplus m_2 \\
& c_5 = m_0 \oplus m_1 \oplus m_3 \\
& c_6 = m_0 \oplus m_2 \oplus m_3 \\
& c_7 = m_1 \oplus m_2 \oplus m_3
\end{aligned} \tag{1}$$

Like all parity check codes, the Hamming (8,4,4) code can also be described using the Generator matrix G for the code.

$$\begin{aligned}
G &= \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \\
\vec{c} &= \vec{m}G
\end{aligned} \tag{2}$$

Finally, another encoding scheme more suited for hardware is the optimized Trellis of the code shown in Figure 1. This can be implemented as a state machine in which the four message bits that are passed in as an input traverse a unique path in the Trellis, through which the codeword can be extracted.

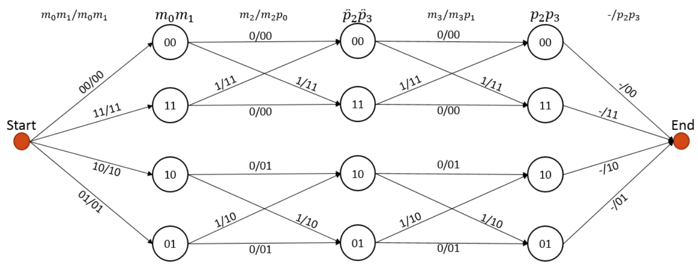


Fig. 1: Encoding Trellis for (8, 4, 4) Hamming Code

To transmit the codeword in a channel, a certain modulation scheme is necessary. Here we use BPSK modulation, which modulates binary 0 to -0.5 and 1 to 0.5. Using this modulation scheme, codeword $\vec{c}$ is modulated to the channel input $\vec{x}$. During transmission, signals are subject to corruption by noise, which can be characterized by signal-noise-ratio(SNR). The equation for computing SNR is shown below, where E_b denotes the energy per bit and N_o denotes the noise spectral density.

$$SNR = \frac{E_b}{N_o}(dB) \tag{3}$$

A noise vector $\vec{n} = n_0n_1n_2n_3n_4n_5n_6n_7$ can be generated from a zero-mean Gaussian distribution. By changing the variance of this Gaussian distribution, different SNR can be achieved. Since the Gaussian noise is additive, the channel observation $\vec{r}$ is computed as:

$$\vec{r} = \vec{x} + \vec{n} \tag{4}$$

The optimum decoding performance of the Hamming (8,4,4) code is achieved under Maximum Likelihood (MLE) decoding. This involves generating the entire codebook consisting of 16 codewords (one for each possible 4 bit message combination), and then calculating the distance between the received noisy channel observation and each codeword in the code book. The most likely transmitted codeword is then the one with the minimum distance to the channel output. This approach is computationally intensive, however, both in space and time. Coming back to the Trellis of the code, it can be observed that the noisy channel observation can be used in conjunction with the Viterbi algorithm to calculate the most likely path traversed along the Trellis, and hence the most likely transmitted codeword. Incorporating the modulation scheme and taking advantage of the symmetricity of the Trellis can further reduce the number of computations, and most notably eliminate the need for any multiplications. The entire decoding process can then be carried out using only 18 additions and 11 comparisons. This decoding approach is theoretically expected and verified through simulation to have the exact same decoding performance as the MLE decoder.

In the decoder, fixed point arithmetic are used to compute floating point operations. The fixed point format Qn.m represents a floating point number f as an $(n+m)$ bits binary number. The representation process can be shown as in the equation below:

$$\begin{aligned} f = & -2^{n-1}b_{n-1} + 2^{n-2}b_{n-2} + ... \\ & + 2^{0}b_{0} + 2^{-1}b_{-1} + ... + 2^{-m}b_{-m} \end{aligned} \tag{5}$$

Fixed point arithmetic is implemented in the same manner as 2‘s complement arithmetic with the exception that an overflow/underflow is handled by saturating the result at the maximum/minimum value of the Qn.m format. The performance of the Viterbi decoder for different Qn.m formats are shown in Figure 2.

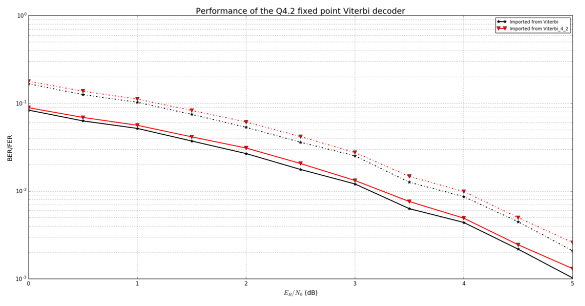


Fig. 2: Performance of various fixed point lengths

For Q3.10, the bits for integer are too few to provide an enough range. However for Q6.2, the improvement from larger integer range is limited. Therefore 4 bits for integer can be satisfactory. Q4.10 can provide a much more precise representation, but the cost is high. Despite slight quantization noise, Q4.2 is a good choice to be implemented.

III. Circuit Design

In the circuit design of this project, we will target the AMI 0.5 m process using the MOSIS scalable CMOS submicron design rules with = 0.3 m. This is the same design rule with labs from this course. In order to compare the performances of different designs, we present two kinds of schematics and layouts of the Viterbi decoder, each of which is optimized for power or speed respectively.

As we are using a data structure of 6 bits, a Viterbi decoder can be made up of several 6-bit adders, comparators and multiplexers. Schematics and layouts of these required modules and the overall decoder circuit designed in electric will be presented in the following part of this section.

A. Adder

Optimization methods for speed and power are different from each other in the choices of 6-bit adders. Carry-ripple adders are chosen for the power-efficient design, while Kogge-stone adders are selected to achieve fast computation for the other.

Also, further modifications should be made to adjust an ordinary adder to a fixed point adder. Fixed point notation can be added and subtracted the same way that signed binary is, but overflow needs to be handled differently. In a 6-bit signed number, possible values range from -32 to 31. The modified adder needs to assert the output to -32 when it detects overflow of the low range, and it needs to assert the output to 31 when overflow in the high range is detected. Let a and b be the two inputs of an adder. y denotes the final output. The addition performed by a 6-bit fixed point adder can be expressed as a function in equation below:

$$y = \begin{cases} -32 \ (10000) & a+b < -32 \\ a+b & -32 \leq a+b < 31 \\ 31 \ (01111) & a+b \geq 31 \end{cases} \tag{6}$$

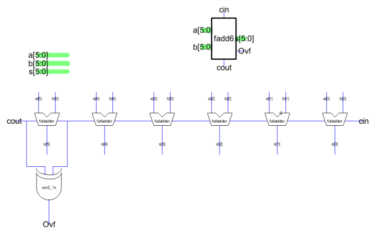
The determination of whether the sum of a and b is located in the range of $[-32, 31]$ can be made by comparing the most significant bit of one of operands and the output for a difference in sign. This subsection is going to introduce carry-ripple and kogge-stone adders as well as their modification approaches respectively.

1) Carry-ripple adder (CRA)

The basic carry-ripple adder is built from six single-bit full adders with their carry-in and carry-out ports connected. Additionally one XOR gate is needed to generate the overflow signal. The schematic and layout of the basic carry-ripple adder is shown in Figure 3. The fixed point adder consists of the basic CRA and two 6-bit multiplexers. Multiplexers take advantage of the sign bit and overflow from the sum of inputs to select the proper output. Figure 4 shows the schematic and layout of a CRA used in viterbi decoder.

2) Kogge-Stone adder (KSA)

The Kogge-Stone adder is a tree adder that uses a specific design of carry-propagate layering. The benefit of this is that it computes carry signals in parallel, in contrast to the ripple carry adder. The ripple carry adder is comprised of a string of a full adder blocks. Each full adder block must wait for the carry out from the previous full adder before it can compute its


(a) Schematic

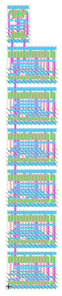
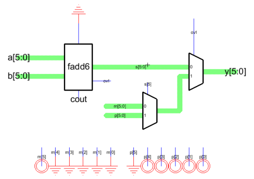

(b) Layout

Fig. 3: Circuit Design for 6-bit Full Adder


(a) Schematic

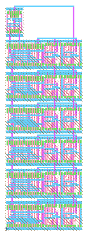
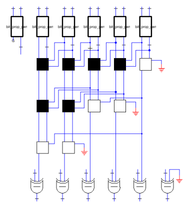

(b) Layout

Fig. 4: Circuit Design for Fixed Point 6-bit Carry-Ripple Adder

own carry out. With large number of bits, this becomes a very slow method of addition. Our Kogge-Stone adder is made up of three basic logic blocks. One that computes propagate and generate from the operands, one that computes propagate and generate based on the previous level of propagate and generate, and one that computes only generate from the previous level. Schematics and layouts of these blocks can be found in the Electric file of this project. These blocks are wired together in a way that significantly reduces the critical path of the addition. The schematic and layout of this adder is shown in Figure 5.


(a) Schematic

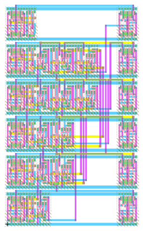
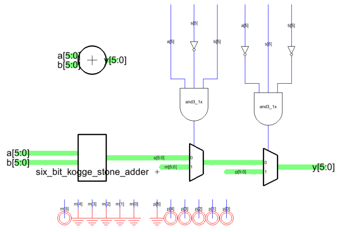

(b) Layout

Fig. 5: Circuit Design for 6-bit Kogge-Stone Adder

The Kogge-Stone adder operates only typical binary addition. The Viterbi design envisioned uses fixed point arithmetic. The Kogge-Stone adder can be modified to handle fixed point notation.The assertion of -32 or 31 can be done using six bit multiplexers that use the overflow signal as the selector. Once the multiplexer and overflow detection are appended, the Kogge-Stone adder is now a fixed point Kogge-Stone adder. Figure 6 presents the schematic and layout of the fixed point Kogge-Stone adder

There may be doubts as to whether or not the benefits of


(a) Schematic

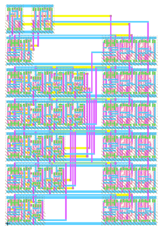

(b) Layout

Fig. 6: Circuit Design for Fixed Point 6-bit Kogge-Stone Adder

a Kogge-Stone adder can be realized with only six bits. As it stands, six bit addition requires five stages of propagation delay in the critical path of a ripple carry adder, while the critical path of a six bit Kogge-Stone adder is forced through only three stages. This 40% reduction in stages could still present an increase in speed over the ripple carry adder, architecturally speaking. However, the Kogge-Stone adder is far more complex. It uses many more logic blocks, which means it consumes more power and it has more capacitance from its abundance of wires.

B. Comparator

The comparison of two 2's complement binary numbers a and b can be done by computing $a-b$ and then determining whether the result is positive or negative by observing the result sign bit and overflow. Therefore to build a comparator, a 6-bit inverter, which is constructed with 6 single inverters from the standard cell library, and a 6-bit full adder from the previous CRA subsection are combined to compute subtraction, while an XOR gate is used to output the comparison decision. The schematic and layout of this 6-bit comparator can be found in the Electric file.

C. Multiplexer

Multiplexers are used to select the proper signal transmitting from the current stage to the next stage of a Viterbi decoder. In this project, a 6-bit multiplexer consists of 6 single-bit multiplexers from the standard cell library with their input nodes for selection connected together. The schematic and layout of this multiplexer can be found in the Electric file.

D. Overall Decoder Design

To decode an (8, 4, 4) Hamming code, the required numbers of each module are listed in table I below

TABLE I: Required Number of Individual Modules

Module	Required number
6-bit Adder	18
6-bit Comparator	11
6-bit Multiplexer	10
Single-bit Multiplexer	12
Single-bit Inverter	6

Figure 7 presents the general schematic of a Viterbi decoder assembled from modules we discussed above. Figure 8 shows the layout of the two decoders with carry-ripple adders and kogge-stone adders respectively. Both the layouts passed DRC, ERC and NCC successfully.

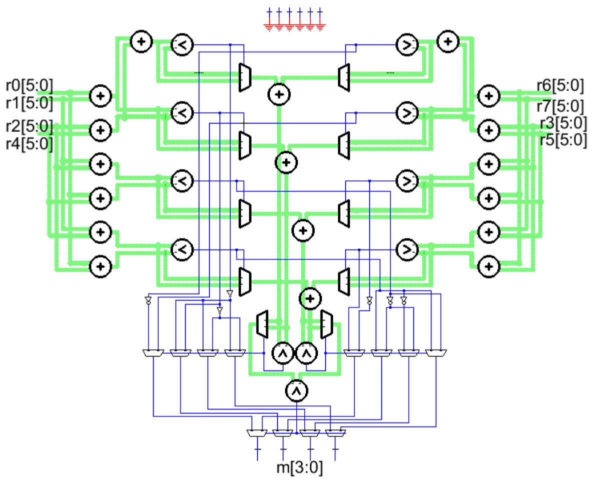
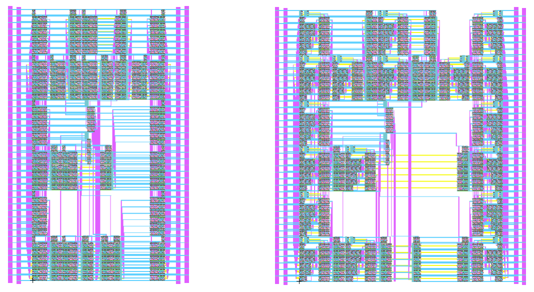


Fig. 7: Schematic of an (8, 4, 4) Viterbi Decoder



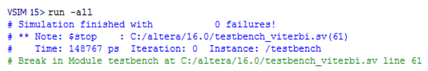
(a) Decoder with CRA (b) Decoder with KSA

Fig. 8: Layouts of 2 Viterbi Decoders

IV. Logic Verification

A. *Modelsim*

Most work of logic verification has been done in Modelsim. A fixed point Viterbi decoder has been created in verilog using ideal hardware modules. Noisy channel observations in Q4.2 format and expected decoder outputs are generated from C code. Using data extracted from C, a testbench can be built up around the Viterbi decoder. To test whether the hardware modules designed in Electric could function well, the ideal components in verilog programme can be substituted with the testing decks of every individual modules generated by Electric. One example of successful simulation in Modelsim is shown in Figure 9.

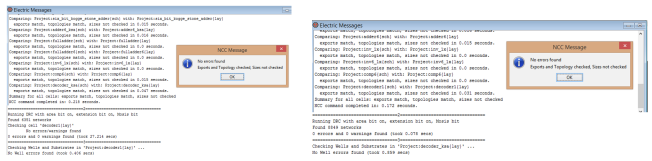


```
VSIM 15> run -all
# Simulation finished with           0 failures!
# ** Note: $stop    : C:/altera/16.0/testbench_viterbi.sv(61)
#    Time: 148767 ps  Iteration: 0  Instance: /testbench
# Break in Module testbench at C:/altera/16.0/testbench_viterbi.sv line 61
```

Fig. 9: An Example of Successful Simulation in Modelsim

B. *Electric*

Two decoder hardwares designed in Electric both passed DRC, ERC and NCC. Figure 10 shows the screenshots of two decoder layouts passing all design checks.



```
Electric Messages
Comparing: Project:six_bit_kogge_stone_adder{sch} with: Project:six_bit_kogge_stone_adder{lay}
  exports match, topologies match, sizes not checked in 0.015 seconds.
Comparing: Project:adder6_ksa{sch} with: Project:adder6_ksa{lay}
  exports match, topologies match, sizes not checked in 0.016 seconds.
Comparing: Project:fulladder{sch} with: Project:fulladder{lay}
  exports match, topologies match, sizes not checked in 0.0 seconds.
Comparing: Project:fulladder6{sch} with: Project:fulladder6{lay}
  exports match, topologies match, sizes not checked in 0.0 seconds.
Comparing: Project:inv6_1x{sch} with: Project:inv6_1x{lay}
  exports match, topologies match, sizes not checked in 0.0 seconds.
Comparing: Project:comp6{sch} with: Project:comp6{lay}
  exports match, topologies match, sizes not checked in 0.015 seconds.
Comparing: Project:decoder_ksa{sch} with: Project:decoder_ksa{lay}
  exports match, topologies match, sizes not checked in 0.047 seconds.
Summary for all cells: exports match, topologies match, sizes not checked
NCC command completed in: 0.218 seconds.
================================2================================
Running DRC with area bit on, extension bit on, Mosis bit
Found 6351 networks
Checking cell 'decoder1{lay}'
        No errors/warnings found
0 errors and 0 warnings found (took 27.214 secs)
================================3================================
Checking Wells and Substrates in 'Project:decoder1{lay}' ...
No Well errors found (took 0.406 secs)

NCC Message
No errors found
Exports and Topology checked, Sizes not checked
```

```
Electric Messages
Comparing: Project:adder6{sch} with: Project:adder6{lay}
  exports match, topologies match, sizes not checked in 0.015 seconds.
Comparing: Project:inv_1x{sch} with: Project:inv_1x{lay}
  exports match, topologies match, sizes not checked in 0.0 seconds.
Comparing: Project:inv6_1x{sch} with: Project:inv6_1x{lay}
  exports match, topologies match, sizes not checked in 0.0 seconds.
Comparing: Project:comp6{sch} with: Project:comp6{lay}
  exports match, topologies match, sizes not checked in 0.0 seconds.
Comparing: Project:decoder1{sch} with: Project:decoder1{lay}
  exports match, topologies match, sizes not checked in 0.031 seconds.
Summary for all cells: exports match, topologies match, sizes not checked
NCC command completed in: 0.172 seconds.
================================2================================
Running DRC with area bit on, extension bit on, Mosis bit
Found 8849 networks
0 errors and 0 warnings found (took 0.078 secs)
================================3================================
Checking Wells and Substrates in 'Project:decoder_ksa{lay}' ...
No Well errors found (took 0.859 secs)

NCC Message
No errors found
Exports and Topology checked, Sizes not checked
```

(a) Decoder with CRA (b) Decoder with KSA

Fig. 10: Demonstration of passing design checks

V. Circuit Simulation

A. *Testing Methodology*

While the circuit itself is not overly complicated, it requires 48 inputs leading to approximately 281 trillion input combinations. It is obvious that this is unfeasible to verify exhaustively, therefore a bounding approach must be adopted instead. To solve the issue of too many inputs, individual modules which makeup the top level will be individually tested. The maximum of which contains 12 inputs, or 4096 possible input combinations a reasonable number for exhaustive testing. Bounds can then be obtained for each of these modules, and by multiplying these bounds by the number of modules contained in the top level an approximation can be obtained for the overall circuit.

1) *Capacitive Loading*

As each circuit is being tested in isolation, we must compensate for the absence of load by including a characteristic capacitance which mimics the load a circuit would drive if integrated into the overall circuit. If interconnect is neglected (as it is in the case of this report) the loading of a module consists of the output capacitance of the module and the input capacitance of the module to which it is connected. Since the output capacitance is accounted for during SPICE simulation, the remaining load can be characterised by adding a capacitor representative of the weighted average of the input capacitance of all modules in the circuit. There is also interest in the worst-case loading, which can be evaluated by substituting the average with the largest input capacitance of each module. The values of these characteristic capacitances are summarized below in Table II.

TABLE II: Characteristic capacitance for various loading conditions.

	Kogge-Stone	**Ripple Carry**
Avg Input Load (fF)	38.59	36.53
Max Input Load (fF)	75.01	68.69

2) *Power Consumption*

Our goal is to determine the average and worst case power consumption for each top-level module under various loading conditions. This can be accomplished by exhaustively simulating all possible switching conditions on LTSPICE and observing power consumption on a per-case basis. By multiplying the power statistics by the total number of modules in each decoder, we can get an idea of the average consumption and a conservative upper bound estimate, as it is assumed the probability that every module will be switching at its worst case is exceedingly rare.

3) *Propagation Delay*

There are two main simulations of propagation delay: dynamic and static. As dynamic delay represents the delay associated with each input vector, it is unfeasible to calculate for similar reasons as dynamic power consumption. Analysis will instead focus on static delay, which is characterised by converting the circuit into an edge/node weighted graph with weights representing interconnect/gate delay respectively.

In practice, static propagation delay is determined by inputting predetermined minimum and maximum standard cell delays into sophisticated circuit simulators. These would progress up through the circuit hierarchy, calculating delay at each step until an approximation is reached at the top level. A similar methodology was employed in this report, however as these tools were unavailable various simplifying assumptions had to be made to allow calculation by hand (elaborated on in the following section).

IRSIM was used to characterise min/max delay of basic gates, which were substituted into visually determined longest and shortest paths. This provided the propagation delay for each module at the lowest hierarchy level. To obtain the next level the process was iterated, substituting the values of the gates for the next level of the hierarchy, and so on until the system is fully characterised. As in power consumption interconnect was neglected, and the impact of this will be discussed in the Results section.

B. *Assumptions*

Due to the lack of sophisticated modelling software, several simplifying assumptions were required to allow for basic calculation. Much of these assumptions have the effect of producing a more conservative result, and those that do not will be discussed in detail in the Discussion section

- Assume uniform distribution of input vectors not true in general, but conservative.
- Neglect static power dissipation due to size of process, assume switching power is dominant factor.
- Delay of all outputs equal to the largest propagation delay for multi-output modules.
- False paths are neglected.
- Interconnect is neglected.
- The transistor are uniformly sized using the 1x ratio (between 1.5-2.5).

C. *Results*

1) *Power Consumption*

Average and maximum power consumption were first calculated for each module in the top-level of the circuit. Figure 11 and Figure 12 demonstrate the simulation results under average and maximum loading respectively, with the MUX6 being neglected from analysis as it is simply six parallel MUX2 modules.

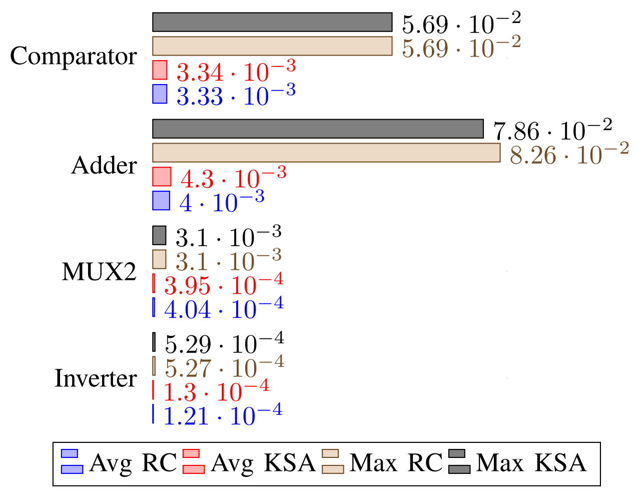


Fig. 11: Individual power consumption under average loading

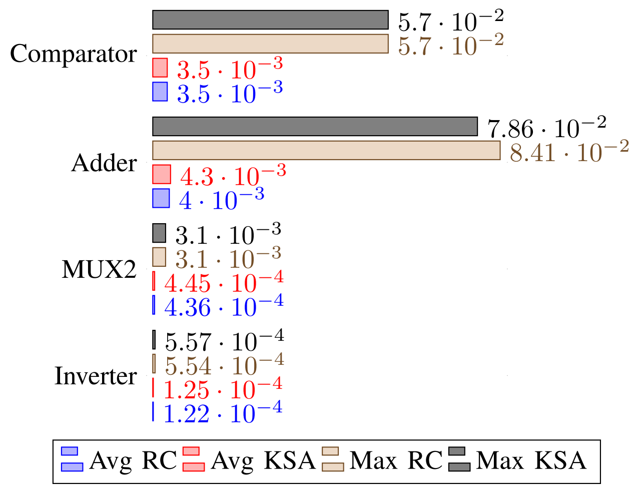


Fig. 12: Individual power consumption under maximum loading.

These values were then used to calculate the top-level power consumption for the two decoder implementations, summarized below in Table III.

TABLE III: Summary of power consumption for decoder implementations.

	Kogge-Stone		Ripple Carry	
	Avg Load	Max Load	Avg Load	Max Load
Avg Power (W)	1.43E-01	1.48E-01	1.38E-01	1.43E-01
Max Power (W)	2.27	2.27	2.34	2.36

2) *Propagation Delay*

Similar to power consumption, we can compare both the minimum and maximum propagation delay for each module located in the top level. Figure 13 and Figure 14 demonstrate the simulation results under average and maximum loading respectively, with the MUX6 module being neglected from analysis as it has the same delay as MUX2.

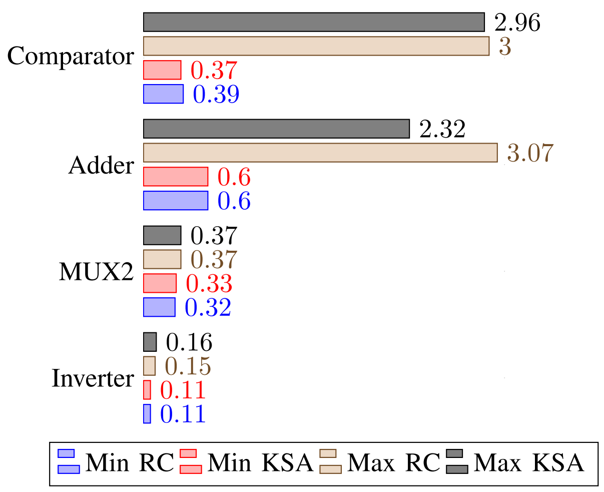


Fig. 13: Individual propagation delay under average loading.

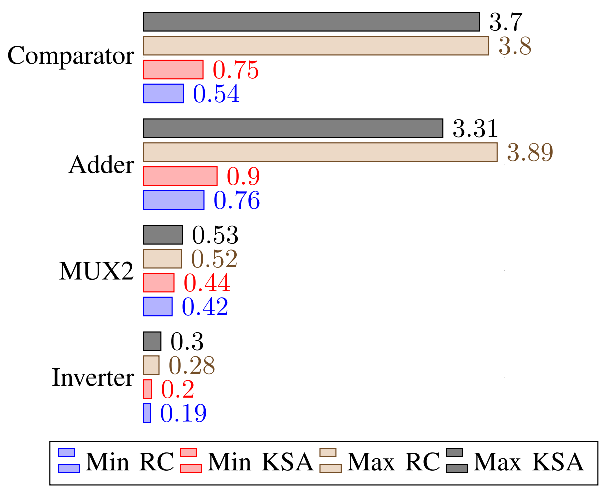


Fig. 14: Individual propagation delay under maximum loading.

Using the hierarchical procedure outline in the testing methodology, the results for propagation delay of the top-level circuit implementations are summarized in Table IV.

TABLE IV: Summary of power consumption for decoder implementations.

	Kogge-Stone		Ripple Carry	
	Avg Load	Max Load	Avg Load	Max Load
Shortest (ns)	1.64	3.63	2.34	3.09
Longest (ns)	16.95	22.62	19.32	24.63

D. Discussion

Upon examination of the results, The Kogge-Stone decoder does indeed operate faster than the Ripple-Carry decoder in nearly all cases as expected.This increase in speed can be attributed to the reduction of the longest path through the adder as demonstrated in Figures 13 and 14.

These results are verified by extracting a netlist from both Schematic and Layout, and inputting the values into IRSIM for analysis. It can then be observed from Figure 15 that the values fall within the calculated bounds when interconnect is both included and neglected.

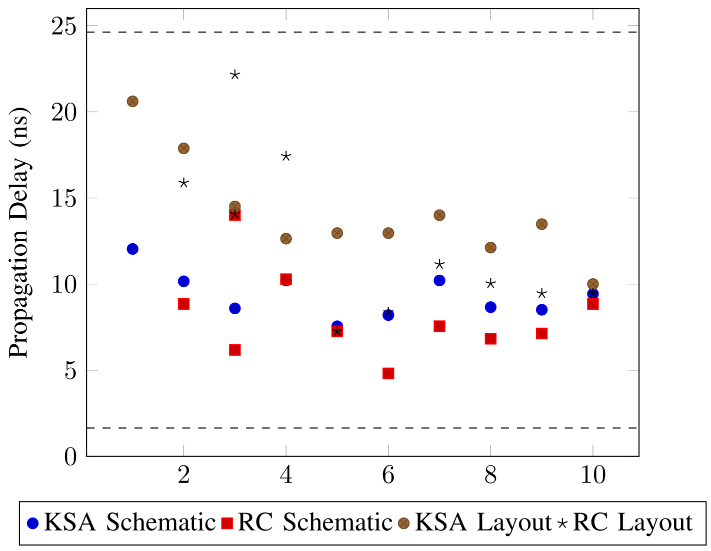


Fig. 15: Full circuit propagation delay analysis

Power consumption however provides an unexpected result. Although on average the Kogge-Stone consumes more power as predicted, the Ripple-Carry dominates in the worst-case, with an explanation found in Figures 11 and 12. Although the Kogge-Stone adder contains more modules and interconnect, each is a small compound gate and interconnect has been neglected. The Ripple-Carry adder is composed of fewer modules, but with unoptimized gate widths contributing large capacitance. It is expected that if interconnect is included and gate widths optimized, the Kogge-Stone will indeed consume more power.

Another interesting result is that the larger modules (comparator and adder) appear to be invariant to load. This is due to internal capacitance dominating over the relatively small load capacitance. This hypothesis has been verified by an additional SPICE simulation with an increased load, resulting in the expected results.

VI. Conclusion

Architecturally, a Kogge-Stone adder is fast and power hungry due to large capacitance from complex wiring, however the overhead of calculating propagate and generate signals in parallel provide an increase in performance. The Ripple-Carry adder remains the simplest form of binary addition, saving on power but operating much slower due to a large critical path.

Although the two decoder implementations differ only by these adders, our comparison yielded interesting results. Addition is highly parallelized in the Kogge-Stone decoder, minimizing the longest path and increasing overall speed. While the Ripple-Carry decoder operated slightly slower, it also consumed more power due to design assumptions. It is expected that if interconnect is included and gate widths optimized, it will consume less power as predicted.

References

[1] C.N.West and D.M. Harris, *CMOS VLSI Design A Circuits and Systems Perspective*, 4th ed., Addison-Wesley, 2011.

[2] VLSI Concepts, and online center for all who have interest in the Semiconductor industry; http://www.vlsi-expert.com/2011/03/static-timing-analysis-sta-basic-timing.html, Wed.March 9 2011.