

High Speed Adder Design: Constructing an 8bit Naffziger Adder

Matt Aldrich, Herman Waterford

Advisor: Richard Lethin

Yale University, EE425

Abstract

A high speed 8b “Naffziger” adder making use of Ling’s equations has been designed and simulated in a 1.5u process with two layers of metal. The 8b adder is comprised of 606 transistors and occupies 1122x840 μm or .944mm². A simulated 8b add yields a result in <5ns or about 10FO4 delays. The fall time is <2ns. The maximum clocking frequency is 150Mhz. At the time of writing, the device suffers from charge sharing and voltage level problems.

1.0 Introduction

Addition time is critical to any CPU design. In most cases, adders occupy the critical path in many key areas of microprocessor operation. Fast adders are necessary in ALUs, addressing memory, and in floating point calculations. Many fast adder designs available today are known as *tree adders*, *parallel prefix* adders, or *logarithmic* adders. [1] These adders improve speed on the critical path by looking ahead across the lookahead blocks. The limiting factor in parallel prefix design is due to the varying levels of bitwise and group propagate, generate, and kill logic. It is known that incorporating higher valency design, for example, valency-4 PG logic, reduces the needed group PG logic for wide bit designs and hence reduces the number of gate delays. Through the use of Ling’s equations [2] it is shown that the bitwise PG logic can be merged into the initial valency-4 stage, thus reducing the logic by one stage. Through the use of dual rail domino logic and Ling’s equations a high speed adder is realized.

2.0 Adder Theory

Basic adders operate on the following principal:

$$S_i = A_i \oplus B_i \oplus C_{in}$$

$$C_{out} = \text{MAJ}(A, B, C_{in})$$

However, it clearly apparent that the rippling the carry one bit at a time will add an unacceptable amount of delay to the circuit. Therefore, for wide bit designs, a common technique is a fusion of both carry-look ahead and carry select techniques.

2.1 Carry Lookahead Techniques

Carry look ahead techniques involve the parallel calculation of n -valency PG logic. Generally, adder designs use valency 2 or valency 4 group logic. This means that 2 or 4

carries will be in any 1 group carry, respectively. The total amount of gate delays in any carry look ahead design is found as follows: [3]

$$\text{Gate Delay} = \log_r[n] + 1(\text{GPK})_{\text{bit}} + 1(\text{Final XOR})$$

where r is the valency and n is the bit width. The log term appears because lookahead structures delay increases with $\log(n)$ [1]. For example, in a 64 bit design a 2 carry lookahead design versus a 4 carry lookahead design results in 8 gate delays in and 5 gate delays respectively. The bit width can be made smaller to approximate gate delays in smaller adders, however the savings are not as drastic.

2.2 Carry Select Techniques

By assuming carries of both 0 and 1 into the group PG logic, two sums are generated at the same time. A multiplexer chooses the carry into each bit and a XOR gate finds the sum.

2.3 Ling's Equations

The benefit of adder design using Ling's equations is the reduction of one gate delay. Recall from the previous section that a valency-4 64 bit has a critical path of 5 gate delays. By merging bitwise PG logic into the initial valency-4 stage, 1 gate delay is reduced. This achieved by defining a pseudo-generate (sometimes called pseudo-carry). [1]

2.3.1 Definitions

Similar derivations are presented in Harris [1] but are presented here for completeness and clarity.

Define bitwise generate and bitwise propagate as:

$$G_i = A_i \bullet B_i$$

$$P_i = A_i + B_i$$

for $i=(1..4)$. Assume A_1 and B_1 are the LSB. Normally, a valency-4 carry lookahead adder's first level of PG logic is given by:

$$G_{4:1} = G_4 + P_4(G_3 + P_3(G_2 + P_2G_1))$$

This calculation can be made faster by defining a *group pseudo-generate* $H_{4:1}$ where the P_4 term is removed.

$$H_{4:1} = G_4 + (G_3 + P_3(G_2 + P_2G_1)) = G_4 + G_{3:1}$$

By plugging in A_i and B_i for G_i and P_i one arrives at the conclusion that $H_{4:1}$ is easier to compute. $H_{4:1}$ has eight terms with a fanin of 4 and $G_{4:1}$ 15 terms with a fanin of 5.

The *group pseudo propagate* signal $I_{4:1}$ is a shifted version of the group propagate signal. This accounts for the dropped propagate signal in the pseudo generate. The pseudo generate and pseudo propagate can be written compactly as follows:

$$H_{i:j} = G_i + G_{i-1:j}$$

$$I_{i:j} = P_{i-1:j-1}$$

These signals are then recursively combined to generate longer pseudo generates and pseudo carries. The actual group generate is formed from the pseudo generate. At this point, the signals are fed into a final XOR. In an 8b design, these recursive steps are not needed.

3.0 Fast Adder Design: An Eight Bit Naffziger Adder

This section describes the equations that govern how the adder operates. Originally, the adder was constructed to process 64 bit numbers. Our group follows these same equations but tailors them for smaller bit widths. Extrapolating from [3] it was estimated that optimal layout would yield a 64 adder in $1.5M\lambda$. For the purpose of understanding the adder, a bit width of 8 bits was selected. This ensured that the design would fit into the pad frame as well as allow for testing due to pin limitations.

Figure 1 gives the hierarchy of our eight bit design. This design section will be broken up according to the hierarchy. Within each subsection, the logic that governs device operation will be discussed. Equations specific to eight bit operation are discussed. For the equations that govern an arbitrary bit width design, consult [1]. Second, a schematic of the transistor logic will be given. Finally transistor sizing and logical effort values are given.

Taking inventory, we see that 16 signals, 8 for A and 8 for B are needed. In addition, because the adder is dual rail, an additional 16 “_l” signals will be needed. Table 1 summarizes the necessary hardware needed to build the adder. Some bits with simple logic can be optimized away from the carry chains. This will be addressed in the specific subsection. With the exception of the sum select gate, half the value listed for each gate is what is required to realize either _h or _l of the dual rail.

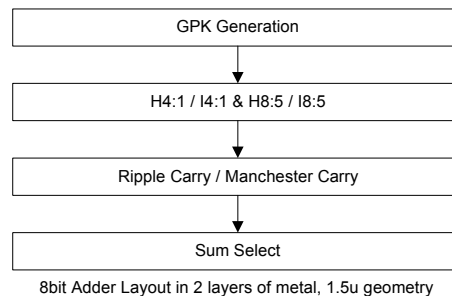


Figure 1. *Hierarchy of 8bit Naffziger style adder.*

3.1 Design Considerations: Logic and Sizing

3.1.1 Logic Family

The adder is designed to work with dual rail domino logic. A clear advantage of this is speed. This is due to lower input capacitance and lack of contention during switching.

1of3GPK	G3	H4	I4	H8	Manchester Carry / RC	Sum Select
8	4	4	4	1	10	8

Table 2. *Gate count for 8bit adder*

Static logic suffers from slow rising and falling transitions, static power dissipation, and a non zero low level noise margin. These problems can be alleviated (in theory) by using a clocked pmos transistor that acts as a resistive pull-up. However, dynamic logic is not always the best solution. Clocking, noise and charge sharing all make dynamic logic difficult to work with. Care must be taken in designing individual cells to ensure that the whole system will operate correctly together.

Because the adder requires dual rail domino throughout, the compliments of the inputs must be generated. That is $\overline{\text{BIT_h}} = \text{BIT_l}$. At a first glance, this essentially doubles the amount of hardware the adder takes to operate. However careful planning of the GPK and sum select gates reveals that transistors can be combined to save space.

As a final design consideration, because we could not guarantee that the inputs to the logic gates in the circuit to always be a “0” during pre-charge, we footed all our logic gates. This extra clocked nmos transistor helps reduce contention in the circuit. Table 2 summarizes both the precharge and evaluate modes of the clock as well dual rail domino signal encoding.

Signal h	Signal l	Clock ϕ	Meaning
0	0	0	precharge
0	1	1	evaluate ‘0’
1	0	1	evaluate ‘1’
1	1	0/1	invalid

Table 1. *At a glance logic table depicting clocking state & dual rail signal encoding*

3.1.2 Sizing

In general, the pull down transistors are chosen to give effectively unit resistance similar to techniques used in static CMOS. The first real design tradeoff encountered was sizing the pft clock. Recall that precharge occurs while the gate is “idle.” In order to reduce time spent precharging, a larger resistance is used. According to Harris, the precharge transistor is chosen for twice the unit resistance. In a pmos clock this means a minimally sized transistor (assuming unit resistance of a pmos is 2R). In an nmos clock or foot the

size can simply be made the same or greater than the pulldown nfets at the expense of greater clock loading.

While a pfet clock of $2R$ reduces clock load and parasitic capacitance [1] it is at the expense of slower rise times. Therefore, to increase transition times, we sized the pfet such that it is unit resistance (R).

In addition, it should be noted that the logical effort of footed gates is higher than unfooted however the performance is better than predicted because series nmos transistors have less resistance than predicted by simple sizing rules. The transistor sizes and logical efforts are given at the end of each subsection discussion.

Large transistors were also used to ensure that the ratio of gate capacitance to wire capacitance was kept as small as possible.

3.2 Adder Subsections

As addressed earlier, the adder uses dual rail throughout. The following equations describe only the true state designated as “_h” unless otherwise noted. The same hardware may be used to compute the compliment “_l”. Also, let $A_0 = B_0 = C_{in}$.¹

3.2.1 GPK Definitions and Implementation

There are two common ways to implement the PGK hardware. One technique requires a dual rail design to achieve both “_h” and “_l” of the signal. The other method, which we employ uses less hardware. It is known as a 1 of 3 hot PGK cell. This method guarantees that only one signal is on at any time.

PGK logic is described as follows:

$$\begin{aligned} G_i &= A_i_h \bullet B_i_h & G_0 &= C_{in_h} \\ P_i &= A_i_h \bullet B_i_l + A_i_l \bullet B_i_h = A \oplus B & P_0 &= 0 \\ K_i &= A_i_l \bullet B_i_l & K_0 &= C_{in_l} \end{aligned}$$

Figure 2 is the schematic implementation of GPK 1 of 3 hot design. It assumes that complimentary inputs are available. The pfets are chosen such that they have unit resistance, that is, twice the unit width. The nfets are all twice unit width. $g_d = 2/3$ and $p_{d_G} = 4/3$, $p_{d_P} = 6/3$, $p_{d_K} = 4/3$.

An aside on the GPK signals: The GPK are used throughout the rest of the design. Extreme care should be given on how the GPKs will be wired globally to the adder. The

¹ This is used in the calculation of $I_{4,1}$ and in the calculation of P_0 in the Manchester carry chain. In this case $P_0 = A_0 + B_0 = C_{in}$ is given by Ling’s equations and is found by A_OR_B however in the subsequent P_i calculations, the propagate term is calculated by using a 1 of 3 hot PGK technique. While the other P_i calculations could have been achieved through $A_i_OR_B_i$ it requires less hardware to use the same propagate from the bitwise GPK. The truth table for both logic types can be found in appendix A. It is shown that $P_{7,1}$ calculated with XOR techniques is the same as $P_{7,1}$ calculated with OR techniques.

bitwise GPKs will be used in the final sum select gate and the propagate terms will be used in the Manchester carry chain. Although obvious, it is pointed out that $K = G_l$. With only one gate, the complimentary G_l signals have been produced. These will be used on the “_l” portion of the circuit.

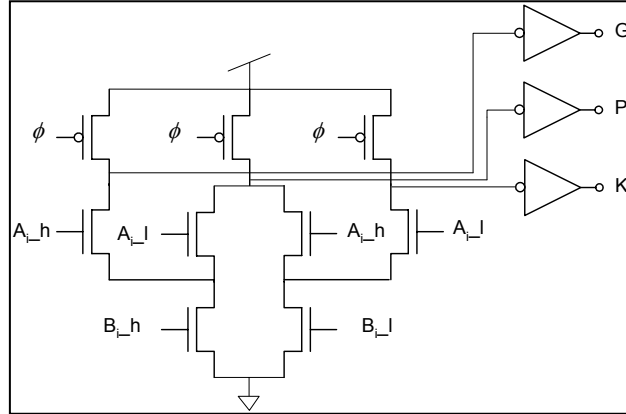


Figure 2. A 1 of 3 hot GPK schematic accepting dual rail inputs.

3.2.2 The G3H4I4 Gate

The PGK cell and the G3H4I4 cell must be thought of as always being together. This is due to the fact that both of these cells take in dual rail inputs. The GHI gate is created by simply reconstructing Ling’s equations as defined earlier. The equations below describe only the _h logic. Also note that these cells are computed by the actual bits, not with PGK.

The G3 term in this gate is merely a by product of the H4 calculation. It is to be used later in generating the actual group generate signals which are then fed into the sum select gate. This is given by the following equation (EQ NUMBER FROM LING). Make note that the GH blocks take in the least significant bit A_1 and B_1 and that A_0 and B_0 in this case are C_{in} .

Expanding these equations:

$\underline{A_1-B_4}$	$\underline{A_5-B_8}$
$G_{3:1} = G_3 + P_3(G_2 + P_2G_1)$	$G_{7:5} = G_7 + P_7(G_6 + P_6G_5)$
$= A_3B_3 + (A_3 + B_3)(A_2B_2 + (A_2 + B_2)A_1B_1)$	$= A_7B_7 + (A_7 + B_7)(A_6B_6 + (A_6 + B_6)A_5B_5)$
$H_{4:1} = G_4 + (G_3 + P_3(G_2 + P_2G_1))$	$H_{8:5} = G_8 + (G_7 + P_7(G_6 + P_6G_5))$
$= A_4B_4 + G_{3:1}$	$= A_8B_8 + G_{7:5}$
$I_{4:1} = P_3P_2P_1P_0$	$I_{8:5} = P_7P_6P_5P_4$
$= (A_3 + B_3)(A_2 + B_2)(A_1 + B_1)(A_0 + B_0)$	$= (A_3 + B_3)(A_2 + B_2)(A_1 + B_1)(A_0 + B_0)$

Figures 3 and 4 give the transistor level implementation of these logic functions. Because the GHI gates compute using the actual bits, good design will allow the bits to flow through the GPK gate to allow for easy docking with the GHI gate.

In this case, sizing for dynamic logic becomes tricky. Logical effort in this case yields worse performance results than what are found in simulation.

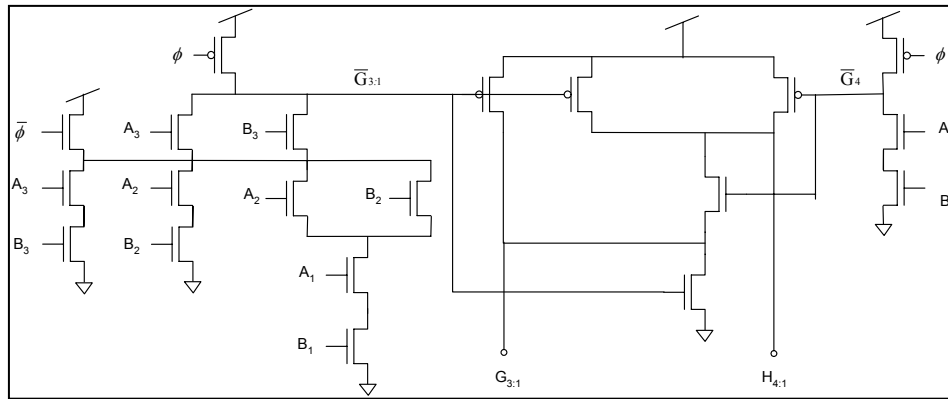


Figure 3. *H gate obtaining $G_{3:1}$ as a byproduct*

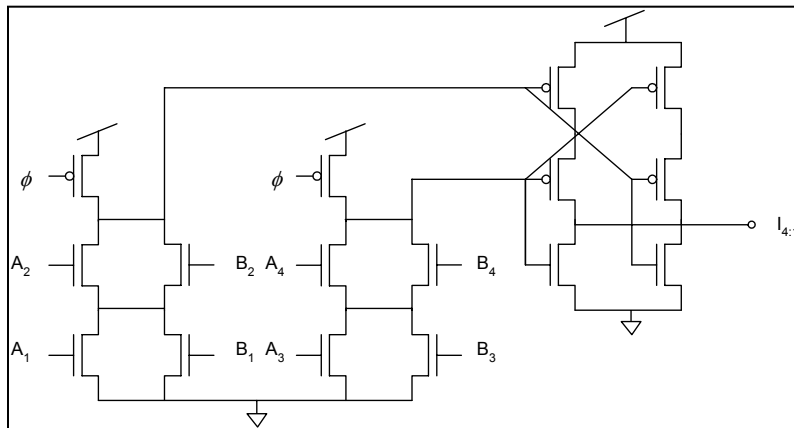


Figure 4. *I4 gate schematic*

The pmos clocks in both figures are again sized for unit resistance. The nmos are sized as twice the unit width. The nmos precharge transistor in figure 3 is used to prevent charge sharing on the loaded internal node [1]. It should be sized larger than the other nmos transistors because its voltage output is dependent on $V_{dd} - V_t$ therefore caution must be taken to avoid any threshold drop. In figure 3, $g_{d_{(G3:1)}} = 2/3$ and $p_{d_{(G3:1)}} = 6/3$. However, this is unlikely in this case because of the heavy loading. $g_{d_{G4}} = 2/3$, $p_{d_{G4}} = 4/3$. It is advisable that this gate be simulated in spice and transistor sizes chosen according to simulation results. In figure 4, $g_d = 2/3$ and $p_d = 6/3$ for both sides. The inverters in both figures' outputs should be unit inverters or skewed inverters if the logic warrants it.

The H and I signals will then be combined in the next step to create a long carry. In addition, the I signals will also be used in the ripple carry chain.

3.2.3 Ripple Carry & Manchester Carry Chain

Now that the long pseudo carry has been built, the last step is creating group generates. Together, along with the PGK logic, these signals will be fed in the sum gates. The generate signals are computed in two steps.

The first step calculates the 3rd and 7th group generates assuming both a pseudo carry of both 0 and 1. In return, both possibilities of the group generate will be created and selected later. We define the notation as follows:

$$\begin{aligned} G^0_{-1:0} = 0 & \quad G^0_{3:0} = G_{3:1} & \quad G^0_{7:0} = G_{7:5} + I_{8:5}(G^0_{3:0} + G_4) \\ G^1_{-1:0} = 1 & \quad G^1_{3:0} = G_{3:1} + I_{4:1} & \quad G^1_{7:0} = G_{7:5} + I_{8:5}(G^1_{3:0} + G_4) \end{aligned}$$

The $G^{0/1}_{-1:0}$ signals along with the $G^{0/1}_{3:0}$ and $G^{0/1}_{7:0}$ are then fed into a Manchester carry chain where similar logic is then generated. One carry chain computes $G^{0/1}_{\{0,1,2\}:0}$ and the other computes $G^{0/1}_{\{4,5,6\}:0}$ thereby creating 8 group propagate signals. Given one possibility of the pseudo carry ($G^{0/1}_{-1:0} = 0/1$), two Manchester carry chains of length 3 are needed to generate all six signals. Therefore for both pseudo-carries at total of 4 chains are needed. Thus to generate both the _h and _l logic a total of 8 chains are needed.

The equations describing these operations can be summarized:

$G^{0/1}_{\{0,1,2\}:0}$	$G^{0/1}_{\{4,5,6\}:0}$
$G^0_{0:0} = G_0 + P_0 G^0_{-1:0}$	$G^0_{0:0} = G_0 + P_0 G^0_{-1:0}$
$G^1_{0:0} = G_0 + P_0 G^1_{-1:0}$	$G^1_{0:0} = G_0 + P_0 G^1_{-1:0}$
$G^0_{1:0} = G_1 + P_1(G_0 + P_0 G^0_{-1:0})$	$G^0_{1:0} = G_1 + P_1(G_0 + P_0 G^0_{-1:0})$
$G^1_{1:0} = G_1 + P_1(G_0 + P_0 G^1_{-1:0})$	$G^1_{1:0} = G_1 + P_1(G_0 + P_0 G^1_{-1:0})$
$G^0_{1:0} = G_2 + P_2(G_1 + P_1(G_0 + P_0 G^0_{-1:0}))$	$G^0_{1:0} = G_2 + P_2(G_1 + P_1(G_0 + P_0 G^0_{-1:0}))$
$G^1_{1:0} = G_2 + P_2(G_1 + P_1(G_0 + P_0 G^1_{-1:0}))$	$G^1_{1:0} = G_2 + P_2(G_1 + P_1(G_0 + P_0 G^1_{-1:0}))$
<p>Note: $G_0 = C_{in}$; $P_0 = A_0 + B_0 = C_{in}$ $G^{0/1}_{-1:0}$ are constants and may be optimized away</p>	

Because the logic of $G^{0/1}_{3:0}$ is straightforward it is not given. Figure 5 represents the schematics for $G^{0/1}_{7:0}$. Both $_h$ and $_l$ versions can easily be combined into one process. Sizing is consistent with previous gates. Figure 6 gives the transistor schematics for the above table of equations.

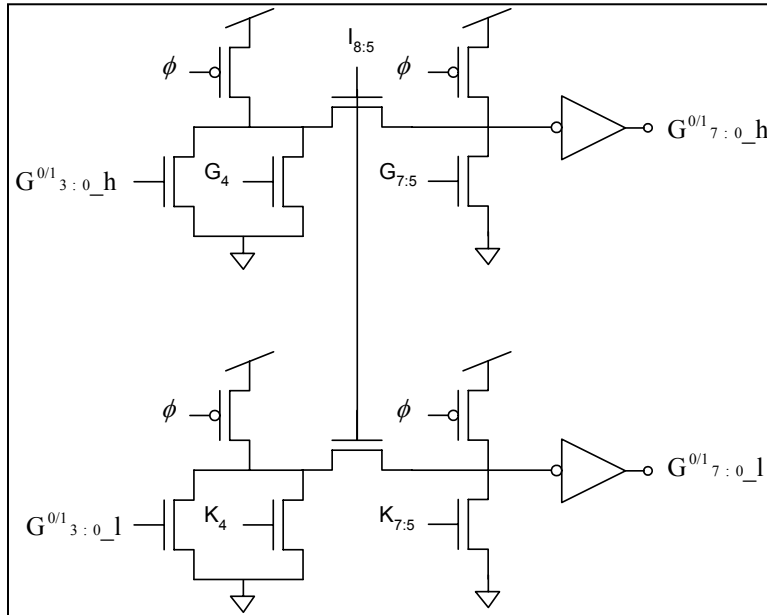


Figure 5. $G_{7:0}$ gate schematic

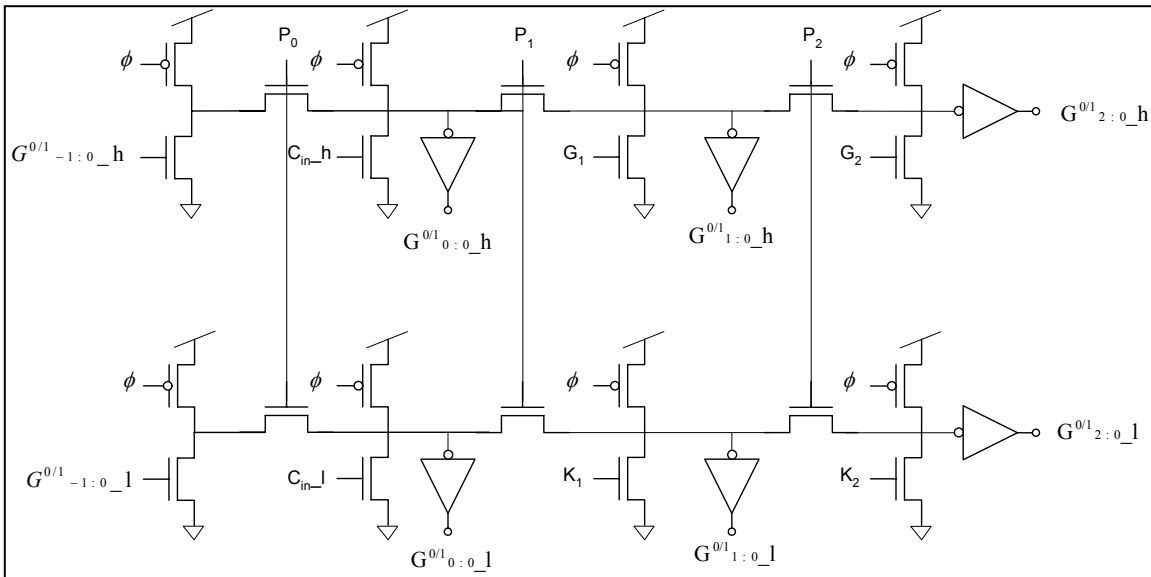
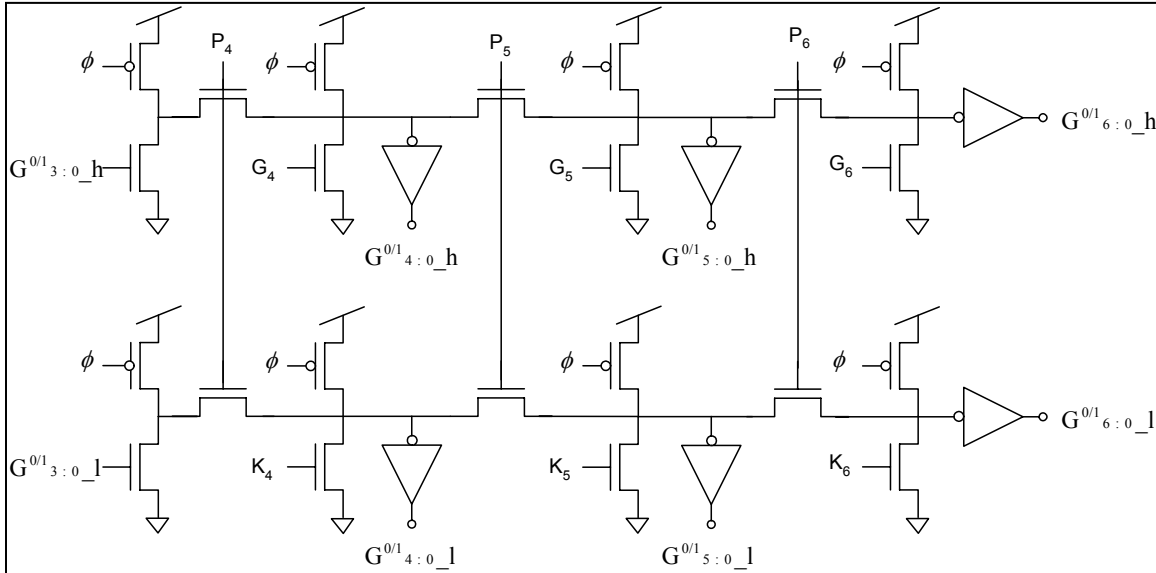


Figure 6. Schematics to generate $G^{0/1}_{\{0,1,2\}, \{4,5,6\}:0(_h/_l)}$ (on next page)



3.2.5 Sum Select Gate

At this point, all the necessary signals are now waiting at the nodes of the sum gate. A multiplexer chooses the appropriate carry into each bit and the XOR finds the sum. These two functions are combined into one large gate. The gate accepts dual rail logic and computes $\bar{P}_i = G_i + K_i$ at the gate. The H signals drive a large fanout and should be buffered accordingly. [1]

The equation that describes this function is given as:

$$S_i = P_i \oplus (C_{in_l} \bullet G^0_{i-1:0} + C_{in_h} \bullet G^1_{i-1:0})$$

where $i=1 \dots 8$

Therefore, the bitwise propagate terms are XOR'd with the carry in signal AND'd with the bitwise group propagates. It is shown in figure 8.

4.0 Testing

In order to guarantee that all nodes are precharged the complimentary signals must both be low at some points. Therefore, at the inputs to the circuit at two input multiplexer is needed. It switched by the clock such that when the clock is low both signals are low and when the clock is high the input runs through an inverter and the compliment is generated. At this point, the clock would be switched high and the output would be taken.

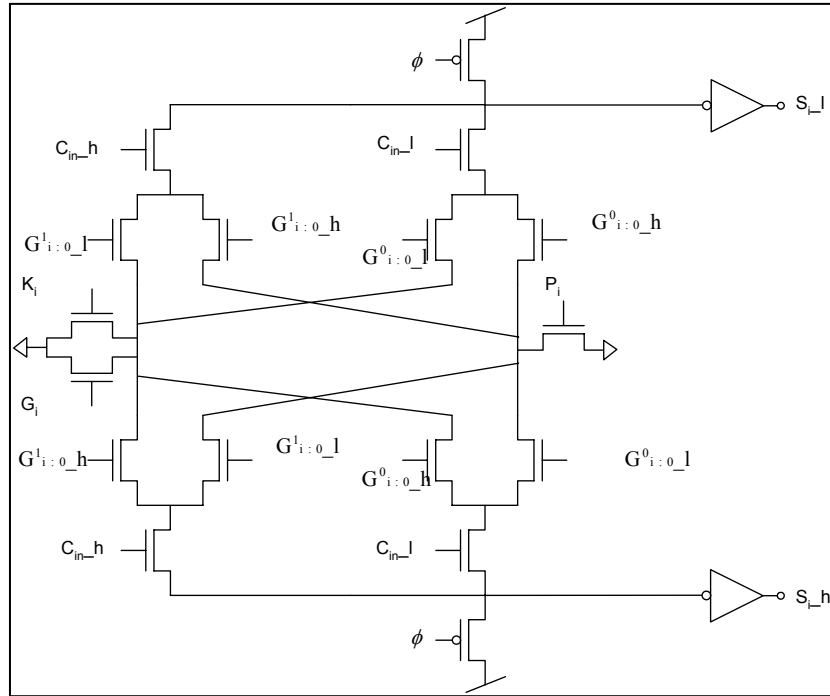


Figure 7. Sum select gate. If this sum select gate were to compute bits $i > 16$ then the carry in would depend on recursively combined H signals.

5.0 Conclusion and Final Remarks

The adder has been simulated in IRSIM using a 1.5u process. The output is obtained in roughly 10 FO4 delays under nominal conditions. The clocking speed is 150Mhz. This falls short of initial performance predictions.

In its current state, the design suffers from charge sharing and noise problems. The individual cells simulate properly, but when wired, some outputs remain indeterminable under certain test cases. We have inserted buffers on large fan outs and on long signal paths however there is still noise at some outputs for certain test cases. Many hours were spent debugging and cleaning up the circuit however in order to pinpoint the problem, a high level simulation based upon transistor sizes should be constructed and analyzed. That way, the effects of fan out and loading can be better understood. Modifications to the IRSIM parameters yield correct results, but sizing and wiring should be adjusted to improve signal voltages at the sum inputs.

Because each adder cell is complicated and many signals are reused throughout the circuit, layout is very complex. The greatest problem our layout suffers from is excessive wiring. While this is generally *the* drawback in tree adder design, the individual cells, now fully developed and operational, should be reworked and fitted such that wiring is reduced. Care should be given to reduce coupling noise on the signals as described in Harris. If long global wiring cannot be avoided, buffers should be added and taken out

later if not necessary. The cells, while functional, can be further optimized so that individual stages fit better together. For example, the pitch of H4 and I4 should be made longer so that 4 GPK blocks are aligned nicely. This would in turn, increase the chip however, at 8 bits, space is not the limiting factor.

If dynamic logic proves too troublesome, future work could be spent developing a static CMOS version of the circuit. Static CMOS may be more appealing, especially for an introductory VLSI class. Consider this: all gates in this design were footed and multiple buffers were needed to preserve signal integrity. Clock load from transistor sizing, charge sharing, and signal noise were all problems encountered in this design. To prevent this charge sharing effect, weak keeper transistors must be added on the outputs. At that point, the extra hardware alone makes static implementation that more appealing.

If anything, this paper and project serve as the first step toward realizing the adder and as an introduction into high speed adder design. In particular, we learned about modern digital design practices and spent much time learning exactly how *robust* it really is. Therefore, it is the group's hope that this paper serve as an easy to understand supplement to both Naffziger's and Harris' work so that future groups can concentrate on further developing and configuring the gate cells and optimizing the layout.

References:

- [1] Harris, D & Weste., N, "CMOS VLSI Design," Addison Wesley, *draft* pp 350-353
- [2] Ling, H. "High Speed Binary Adder," IBM J. Research, Vol 25, No. 3 p 156, May, 1981.
- [3] Naffziger, S., A "A Sub-nanosecond .5um 64b Adder Design," ISSCC96, pp 362-363, 1996

Appendix A

Optimization of Naffziger Adder Propagate Cell (an original contribution)

The adder makes use of the logic that $P = A + B$. However, because we used 1 of 3 hot PGK logic, the $P = A + B$ actually is $P = A \oplus B$. This saves on cells needed to generate all the bit PGKs because the same propagate serves for both the _h and _l sections of our design. At first, this may appear to pose a problem, but recall that GHI generation is done bitwise and not from PGK signals. Therefore, by using 1 of 3 hot encoding, the amount of P cells needed are reduced by a half. The following proves that this is a valid design process. [GIVE EQUATION NUMBER]

$$\begin{aligned} G_{out} &= G + (orP)(Gin) \\ G_{out} &= G + (xorP)(Gin) \end{aligned}$$

Columns 1, 2 are inputs.
 Column 3 is $G = AB$
 Column 4 is Gin , left as a variable
 Column 5 is $orP = A + B$
 Column 8 is $xor P = A xor B = AB' + BA'$

The remaining columns have their operations explicitly stated.
 The truth table for the Manchester series should be generalized as:

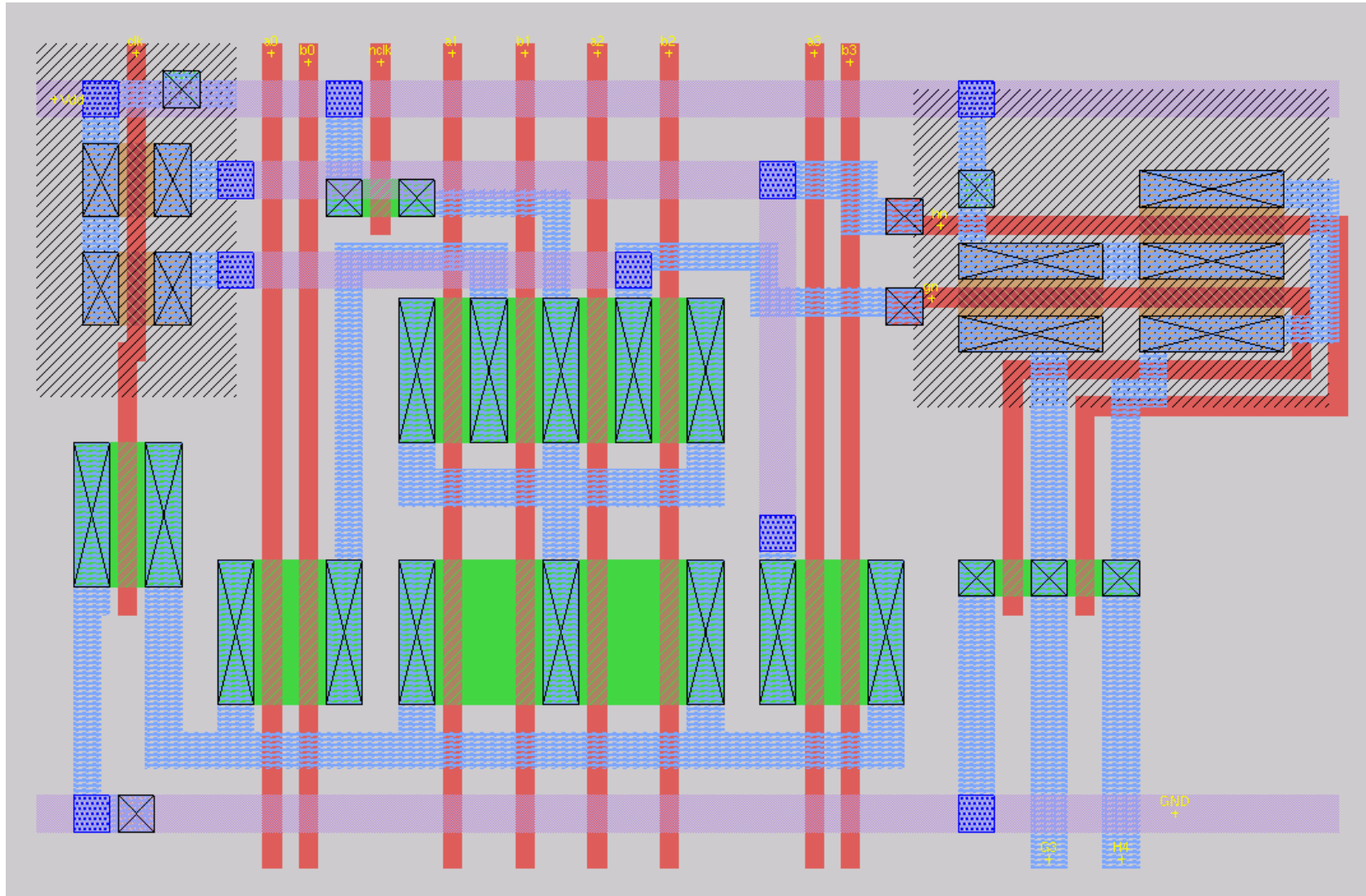
1	2	3	4	5	6	7
A	B	G	Gin	orP	(orP)(Gin)	$G_{out} = G + (orP)(Gin)$
0	0	0	Gin	0	0	0
0	1	0	Gin	1	Gin	Gin
1	0	0	Gin	1	Gin	Gin
1	1	1	Gin	1	Gin	1

Our Manchester series is generalized as:

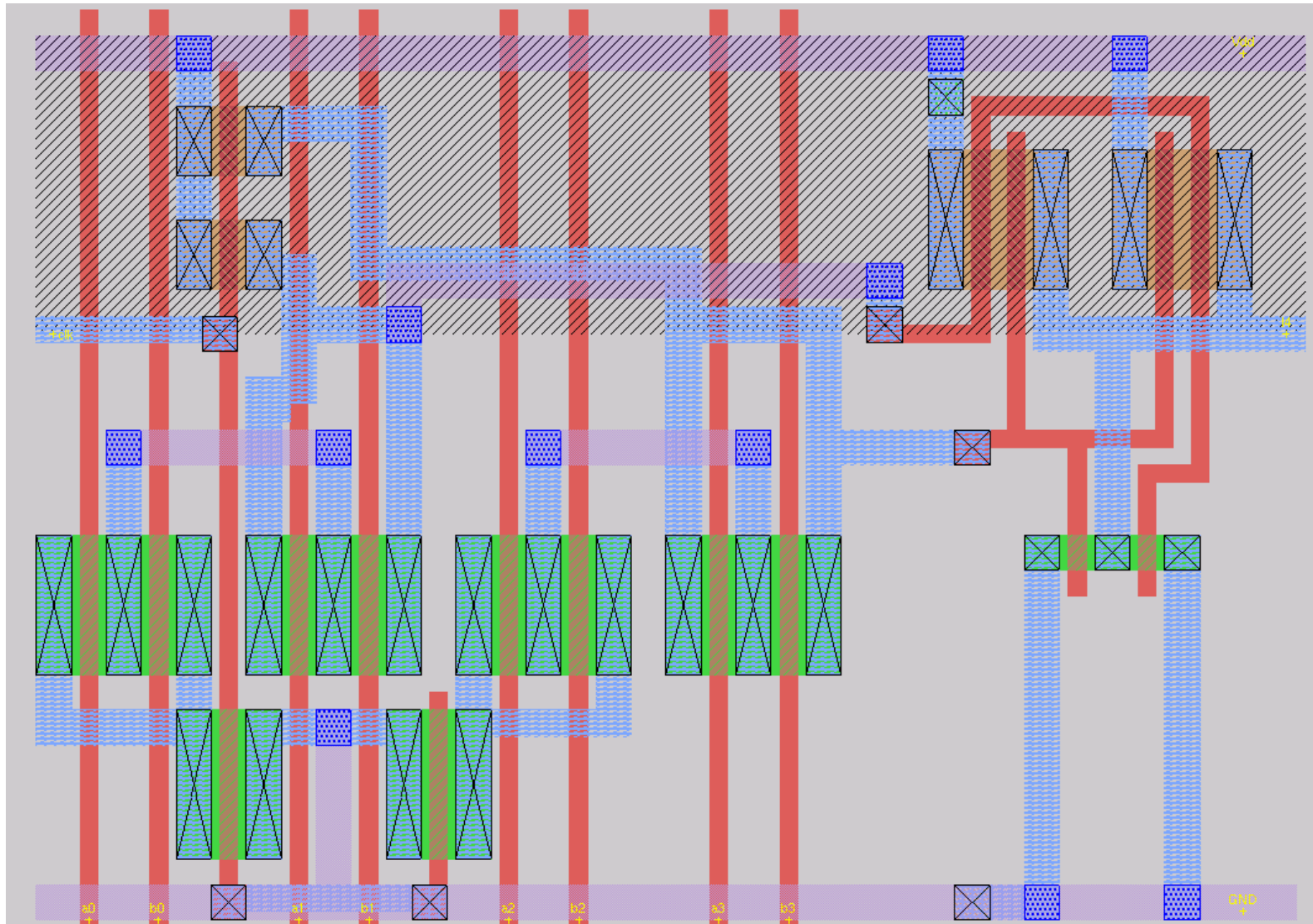
1	2	3	4	8	9	10
A	B	G	Gin	xorP	(xorP)(Gin)	$G_{out} = G + (xorP)(Gin)$
0	0	0	Gin	0	0	0
0	1	0	Gin	1	Gin	Gin
1	0	0	Gin	1	Gin	Gin
1	1	1	Gin	0	0	1

We readily see that column 7 yields the same results as column 10 thus proving additional P logic for the _l of the circuit can be removed and optimized by using 1 of 3 hot logic.

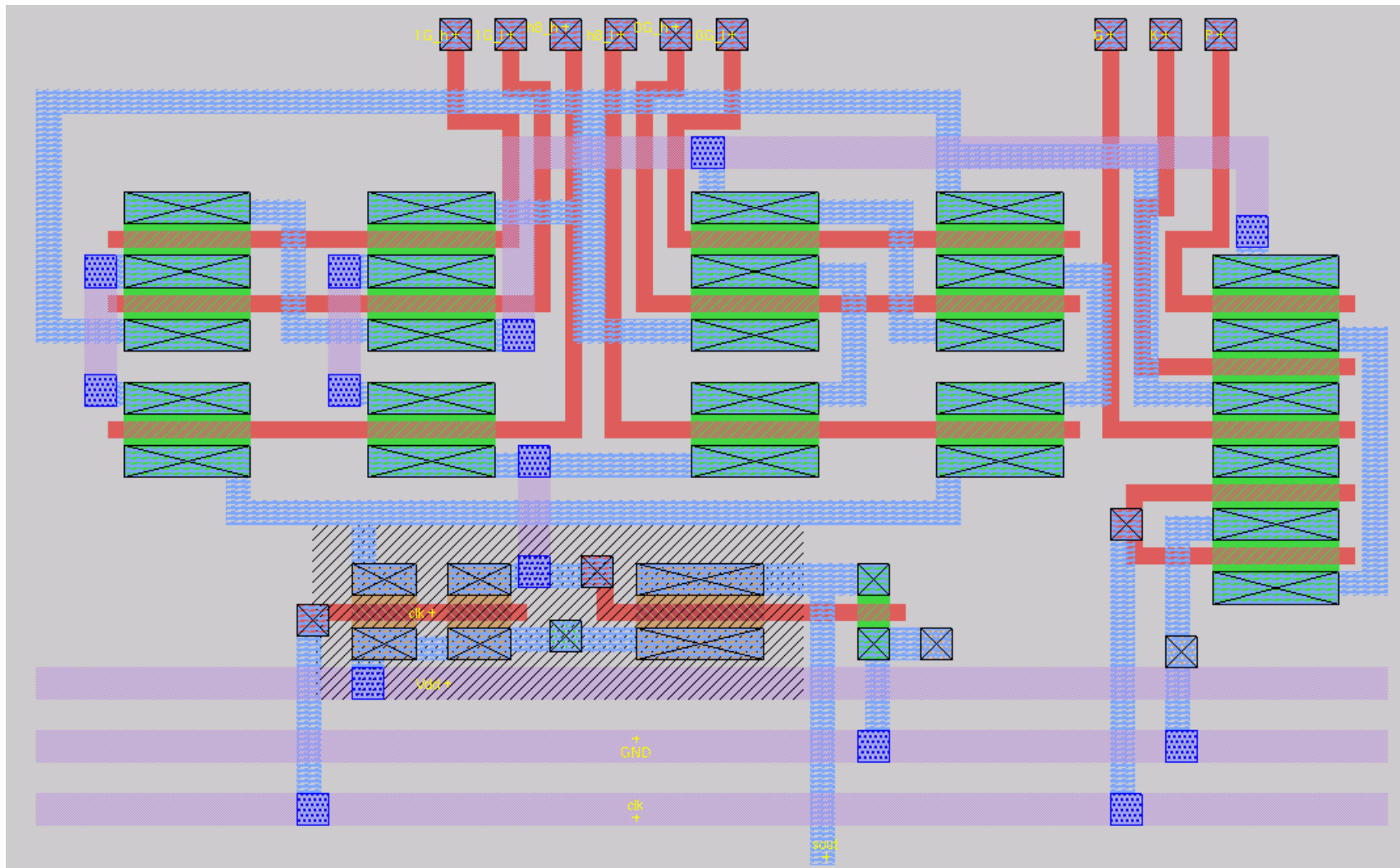
G3H4



I4



Sum Select Gate



Final Adder Layout

