NTNU
Norwegian University of
Science and Technology

4-7GHz Tunable Programmable Pulse Generator in 65nm CMOS

Simon L'Orange

Master of Science in Electronics
Submission date: June 2017
Supervisor: Snorre Aunet, IES

Norwegian University of Science and Technology
Department of Electronic Systems

Acknowledegment

I would like to thank Q-Free for the possibility to do something different and to all my supervisors: Anders Hagen, Brage Blekken, Snorre Aunet and Trond Ytterdal. I would also like to thank Ingrid.

Abstract

A pulse counter has been designed and simulated in a standard 65nm process technology in schematic and can be implemented in a pulse generator. The maximum values from the schematic shows a frequency of 12.5GHz and a power consumption of 2.749mW. PVT simulations at 12.5GHz work with a variation in voltage of 2.5% from a supply of 1V, a temperature variation over [-40,80] degrees and in the corners SS, SF, FS, TT and FF while meeting the specifications. This shows that it has the possibility to both be implemented in layout but also is a plausible solution to counting pulses to vary the bandwidth. It also has the possibility to work over a great range of frequencies. The synchronous counter in TSPC logic has been resimulated with addition circuitry. The addition circuitry is to make it possible to control the counter, buffers on the clock signal, control signal from a lower clock system and get the right amount of pulses out.

Abstrakt

En pulse teller har blitt designet og simulert i en standard 65nm prosess teknologi i skjematikk og kan bli implementert i en pulse generator. Maksimums verdier fra skjematikk viser en frekvens på 12.5GHz og et effektforbruk på 2.749mW. PVT simuleringer på 12.5GHz fungerer med en variasjon i spenning på 2.5% for en 1V kilde, en temperatur variasjon over [-40,80] grader og i hjørnene SS, SF, FS, TT, FF mens den opprettholder spesifikasjonene. Dette viser at den har muligheten for å både implementeres i utlegg men er også en mulig løsning for å telle pulser for å endre båndbredden. Den har også muligheten for å fungere over et stort frekvensområde. Den synkrone telleren i TSPC logikk har blitt re simulert med ytterligere kretser. De ytterligere kretsene gjør det mulig å kontrollere telleren, buffere på klokke signalet, kontroll signaler fra et lavere klokke system og å få korrekt antall pulser ut.

Contents

Acknowledegment **i**

Abstract **ii**

Content **iv**

List of Figures **vi**

List of Tables **viii**

1 Introduction **1**

2 Implementation **3**
2.1 The Pulse Generator . . . 3
2.2 The Pulse Counter . . . 4
2.2.1 Counting . . . 5
2.2.2 Programming of the counter . . . 5
2.2.3 Static to Dynamic . . . 7
2.2.4 Clock gating . . . 8
2.2.5 Enabling . . . 9
2.2.6 Loop with b7 . . . 10

3 Dynamic logic **11**
3.1 Counter principle . . . 11
3.2 Symmetry of rise and fall time of logic . . . 12
3.3 Buffer . . . 12
3.4 Dynamic gates . . . 13
3.4.1 P- and N-C^{2}MOS . . . 13
3.4.2 P- and N-precharge . . . 14
3.4.3 P- and N-blocks . . . 15
3.5 TSPC . . . 16
3.5.1 Carry-look ahead . . . 16
3.5.2 Toggle . . . 17
3.5.3 Single-clock dynamic flip-flops . . . 22
3.5.4 Counter element . . . 25

4 Speed of Transistors **26**

5 Results **27**
5.1 PVT with 64 pulses 28

6 Discussion and conclusion **30**
6.1 Discussion 30
6.2 Conclusion 33
6.2.1 Future work 34

7 Appendix **35**
7.1 Counter element for the i-bit 35
7.2 The different clocks 35
7.3 Flip-flops 35
7.4 The counter and it's internal signals 39
7.5 Enable comparision 45
7.6 PVT results for 1,2,3 and 127 pulses for 12.5 and 9.5GHz 46
7.6.1 12.5GHz 46
7.6.2 9.5GHz 48
7.7 Clock problems for 9.5GHz 50

List of Figures

2.1 The idea behind the pulse generator . . . 3
2.2 High level abstraction of the pulse counter circuit . . . 4
2.3 Inside the pulse counter . . . 5
2.4 The preset and clear transistors for programming the counter . . . 6
2.5 Programming the bits to "1" . . . 6
2.6 Programming the bits to "0" . . . 7
2.7 Timing diagram with *load_D* and *loading* and programming it to "1" 8
2.8 The clock gating circuitry . . . 9
2.9 The enable circuity . . . 9
2.10 Timing diagram of the enable with a input of *pulsecount* higher than 7. . . . 10
2.11 A close up picture of the loop *b7* make . . . 10

3.1 Counting as a clock division with the binary addition the left . . . 11
3.2 Buffers driving a large capacitive load . . . 12
3.3 The P- and N-C^{2}MOS . . . 13
3.4 The timing diagram for the P- and N-C^{2}MOS. . . . 14
3.5 The P- and N-precharge . . . 14
3.6 The timing diagram for the P- and N-precharge. . . . 15
3.7 The P- and N-blocks build up by P- and N-C^{2}MOS and P- and N-precharge . . . 15
3.8 The timing diagram for the P- and N-block . . . 16
3.9 Overlapping clocks . . . 16
3.10 Carry-look ahead logic for dynamic logic . . . 17
3.11 Toggle circuit . . . 18
3.12 Timing diagram of the toggle when the dynamic condition is satisified 19
3.13 The toggle as a divide-by-two circuit with the *ci* transistors removed 19
3.14 Toggle circuit with dual inverter . . . 20
3.15 Timing diagram of the toggle circuit when the dynamic condition is not satisfied. . . . 21
3.16 Toggle circuit with programming and dual-inverter . . . 22
3.17 N-flip-flop . . . 23
3.18 P-flip-flop . . . 24
3.19 Timing diagram for the flip-flops . . . 25

7.2 Comparission of the true clock signal and the buffered clocks at room temperature, TT corner and 1V supply with 5 pulses. 35
7.1 The TSPC toggle counter element 36
7.3 Simulation of N- and P- flip-flops with load rise at high clock 37
7.4 Simulation of N- and P- flip-flops with load fall at high clock 37
7.5 Simulation of N- and P- flip-flops with load rise at low clock 37
7.6 Simulation of N- and P- flip-flops with load fall at low clock 38
7.7 Simulation of 6 N- and P- flip-flops in series with load rise at high clock 38
7.8 Simulation of 6 N- and P- flip-flops in series with load fall at high clock 38
7.9 Simulation of 6 N- and P- flip-flops in series with load rise at low clock 39
7.10 Simulation of 6 N- and P- flip-flops in series with load fall at low clock 39
7.11 No dual-inverter at the toggles 40
7.12 Minimum preset and clear transistors 41
7.13 The outputs of the Nand PP's 42
7.14 The outputs of the carry-look ahead part 43
7.15 Counting with properly sized programming and dual-inverter transistors 44
7.16 A close up picture of counting at -40, supply voltage of 1V and the FF corner (a good condition). 45
7.17 A close up picture of counting at 80 degrees with a supply voltage of 0.975V in the SS corner (a bad condition). 45
7.18 Simulation of showing the buffered clocks at 9.5GHz with FF corner, a temperature of 80 degrees and supply of 0.975V. 50
7.19 Simulation of showing the Ais at 9.5GHz with FF corner, a temperature of 80 degrees and supply of 0.975V and a *pulsecount* as 1111101. 51

List of Tables

2.1 Truth table for loading, *preset* and *clear* 7
2.2 Truth table for *preset[7]* 8
2.3 Truth table for the nand in the clock gating 9

5.1 Results for the counter with different voltage, temperature and process corner with 64 pulses (*pulsecount* = "0111111") at 12.5GHz. . 28
5.2 Results for the counter with different voltage, temperature and process corner with 64 pulses (*pulsecount* = "0111111") at 9.5GHz . . 28
5.3 Results for the counter with different voltage, temperature and process corner with 64 pulses (*pulsecount* = "0111111") at 11.5GHz with 5% variation in the voltage. 29

7.1 Results for the counter with different voltage, temperature and process corner with 1 pulses (*pulsecount* = "1111110") at 12.5GHz with 2.5% variation in the voltage 46
7.2 Results for the counter with different voltage, temperature and process corner with 2 pulses (*pulsecount* = "1111101") at 12.5GHz with 2.5% variation in the voltage 46
7.3 Results for the counter with different voltage, temperature and process corner with 3 pulses (*pulsecount* = "1111100") at 12.5GHz with 2.5% variation in the voltage 47
7.4 Results for the counter with different voltage, temperature and process corner with 127 pulses (*pulsecount* = "0000000") at 12.5GHz with 2.5% variation in the voltage 47
7.5 Results for the counter with different voltage, temperature and process corner with 1 pulses (*pulsecount* = "1111110") at 9.5GHz with 2.5% variation in the voltage 48
7.6 Results for the counter with different voltage, temperature and process corner with 2 pulses (*pulsecount* = "1111101") at 9.5GHz with 2.5% variation in the voltage 48
7.7 Results for the counter with different voltage, temperature and process corner with 3 pulses (*pulsecount* = "1111100") at 9.5GHz with 2.5% variation in the voltage 49

7.8 Results for the counter with different voltage, temperature and process corner with 127 pulses (*pulsecount* = "0000000") at 9.5GHz with 2.5% variation in the voltage 49

Chapter 1

Introduction

The task for the programmable pulse generator is to be used in a radar. It's possible to vary the bandwidth by varying the number of pulses. This is the reason that it will be a variable bandwidth radar, and not a radar which is only ultra-wideband (UWB) or a small bandwidth radar. It can also be used in communication with the same purpose. In a radar, it is also possible to see two different object depending on how long the pulse length is. The problem was to design a pulse counter, a VCO and a switch which would make up the pulse generator in a standard 65nm process technology. The most important part to design was the pulse counter, this is the core of the pulse generator. There was not any good reference to make a pulse counter, and it developed to be the goal of the design. It was known from [4, 7] that the reduction in speed would most likely be a factor of $1.6 - 3$, depending on how complex the design is. From the first it was also known that the maximum speed of a ring oscillator with three elements would be around 16GHz in layout. This is an easy design and is one of the fastest digital components[1]. If all the delays in the ring oscillator is equal, a rough calculation show that by doubling the number of inverters. The speed will reduce by a factor of two. The speed is then around the maximum of what the pulse generator should operate at which is 7GHz. The pulse counter is therefore the most crucial part of the design, and designed first. It will count pulses at the limit of this technology, making normal arithmetic counters not possible to implement at that speed as just descirbed. A very promising solution was found to be a synchronous counter with true single phase logic (TSPC)[10]. Another solution was an arbitrary digital waveform generator (ADWG)[3] where the pulses are loaded into registers and sent out. My approach for this was to go for the synchronous counter because the ADWG was not fast enough, and the counter would have the possibility due to the speed at 0.18µm[6].

My results are a 7-bit counter that work in schematic between [-40,80] degrees, 2.5% voltage variation (0.975V and 1.025V) in the different corner at 12.5GHz with additional circuit for programming, enable the switch out and clock gating. The

[1] With digital components it is meant blocks which can make up logic functions, like complementary CMOS inverter, nand and nor

power consumption is maximum 2.749mW at 12.5GHz for 127 pulses. 127 pulses are chosen because it is the highest amount of pulses which shall be counted, and is thereby the worst case. From the [6] the resimulated version first purposed reached a speed of 4.45GHz in a 0.18nm technology with a 1.8V supply. A quick calculation show that 65nm would reach a speed around $4.45\text{GHz} \times \frac{0.18}{0.065} = 12.32$. The simulated results are better than the expected speed, but if the same calculation is done with the frequency from 3µm and 0.18µmm, the result in 0.18µm is also better than expected. My contribution to this counter is to add a static to dynamic circuit which make it possible to use the transistor introduced in [6] from a system with a slower clock, how it can be buffered and gated clocks and a enable circuitry for turning on and off a switch. These parts make up the pulse counter.

The reader should understand power consumption, complementary CMOS and why it has N- and P-network and the current equation for a transistor.

The rest of this master thesis is organized as follow: chapter 2 is my implementation of the TSPC counter with the additional circuit, chapter 3 is in-depth presentation of the logic used in the implementation, chapter 4 is some necessary theory of the transistors, chapter 5 is the result from PVT, chapter 6 is discussion and conclusion and chapter 7 is the appendix including extra simulations which is commented in the discussion.

Chapter 2

Implementation

In this text the signals are written in italics to point out that it is a signal and not a word. The signals in the figures are not in italics. Signals are both desribed as high and low and "1" and "0" when writing about their values. The word for a more constant value are referred to as a static signal, while pulses are referred to as dynamic signals.

2.1 The Pulse Generator

The idea behind the pulse generator is shown in figure 2.1. The pulse counter which has been made in schematic has been designed with these signals in mind.

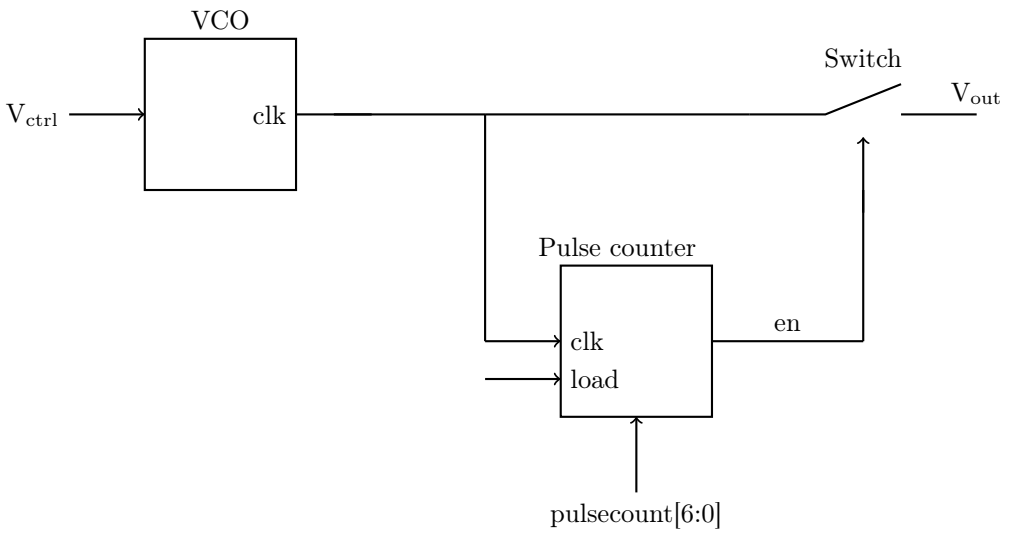


Figure 2.1: The idea behind the pulse generator

The voltage controlled oscillator (VCO) is controlled by a signal V_{ctrl}[1] and send out *clk* which have the frequency in the range $4-7$GHz. *Pulsecount* is the number of pulses that should be counted in bits and is a 7 bits signal. *Load* is the signal that loads *pulsecount* into the counter at rising edge and is equivalent to the PRF[2]. Enable, *en*, is the on and off signal for the switch which should send out, V_{out}, the number of pulses from *pulsecount*. The important thing to understand is that the system controlling the pulse generator will not operate at the same frequency. The pulse generator should work between $4-7$GHz, while a normal µ-controller may work under 100MHz, this means that both *load* and *pulsecount* will be low frequency signals compared to *clk*. A way to transform these signals to work together is also a part of the pulse counter.

2.2 The Pulse Counter

The pulse counter with the most important internal and external signals is shown in figure 2.2. Its function is to count the number of pulses from *pulsecount*, loaded in by the signal *load* at positive edges and set *en* high the number of pulses to be counted.

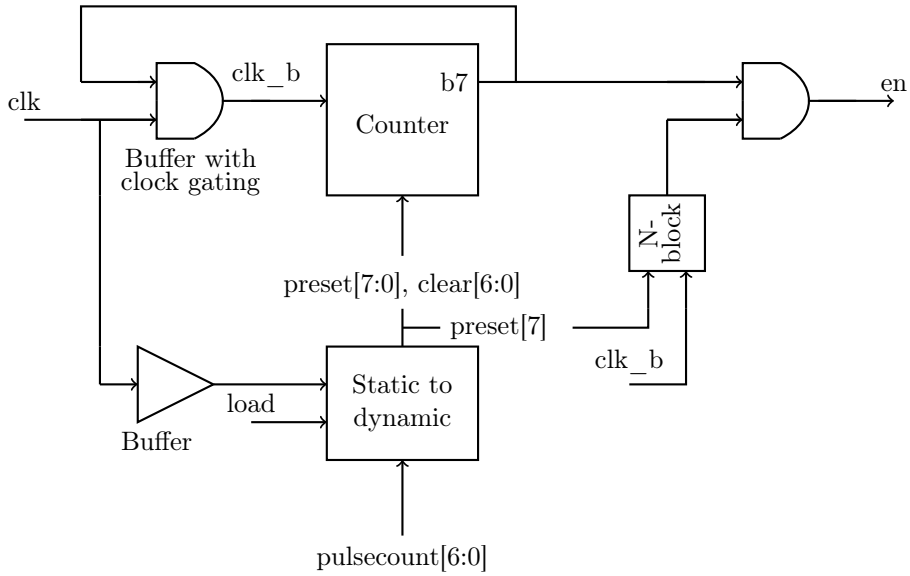


Figure 2.2: High level abstraction of the pulse counter circuit

In the figure the clock, clk, goes into two buffers. One of the buffer is also a clock gating with the input from b7 as the control signal and output the *clk_b*.

[1]Notice that the input signal to the VCO can be a part of a even larger system as a phase lock loop (PLL), but this is outside of this scope.

[2]The PRF is the pulse repetition frequency, it is how often pulses are sent out of a radar

The signal *clk_ b* stands for clock buffered and have nothing to do with the inverse of the clock. While the other buffered clock signal is clk_sta. These clocks are buffered to be able to drive enough transistors in the "Counter"-block and the "Static to dynamic"-block. The static to dynamic circuit inputs a *load* signal and the *pulsecount[6:0]* and output *preset[7:0]* and clear [6:0] as pulses to be able to program the counter. The enable out is $en = b7 + preset[7]\_en$ where *preset[7]_ en* is the output of the N-block and is a positive clock delayed *preset[7]* to get the buffered clocks in-phase.

2.2.1 Counting

The counter works by counting down to zero from specified pulses by *preset* and *clear* (depending on pulsecount) and then stops. As descirbed in the section about the pulse generator, what is possible with complementary cmos is limitied due to the speed, such as equality, addition and so forth. This is the reason that it's counting down to zero and not up from zero. A possible solution was to use *b7* as the control signal for when the counting sould end, controlling the pulse generator. This is because if it's programmed to "1" it will turn to "0" when the counting is finished. *c0* is connected to the supply to make it only dependent on when the programming end. *clk_ b* is the clock and a signal which turn of this circuit to reduce it to only static power.

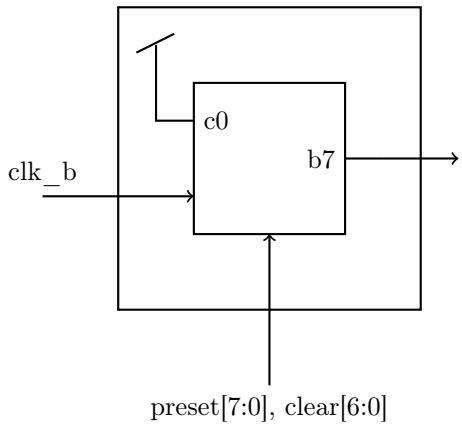


Figure 2.3: Inside the pulse counter

2.2.2 Programming of the counter

Programming of the counter is done with preset and clear transistors, shown in figure 2.4 [6]. It is two transistors, one pmos connected to the source and a nmos connected to ground. This circuit will either pull *bi* high or low depending on *preset* and *clear*. Pulling a node high or low requires time, and this time is depending on the capacitance seen in the node and the size of the transistor.

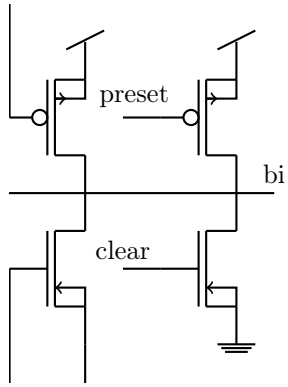


Figure 2.4: The preset and clear transistors for programming the counter

When *clear* is "1" the nmos will be turned off and pull *bi* to "0". When *preset* is "0" the pmos will pull the bit *bi* to "1". It will therefore be the inverse of *pulsecount* because a "1" from *pulsecount* will turn the bit to "0" and the invers for "0" which turn *bi* to "1". When the programming begins the clock *clk_ b* will also be turned on. A timing diagram for the programming to "1" and "0" can be seen in the figures 2.5 and 2.6. It is not possible to program the circuit and count simultaneously. This is the reason why *preset* and clear not can be active at the same time ($preset = 0$ and $clear = 1$) and have to be dynamic signals. When $preset = 1$ and $clear = 0$ the programming is off and the counting can take place.

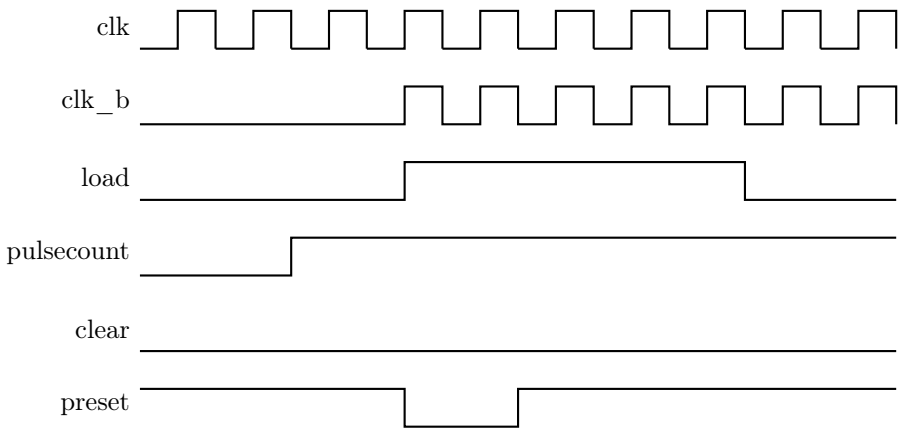


Figure 2.5: Programming the bits to "1"

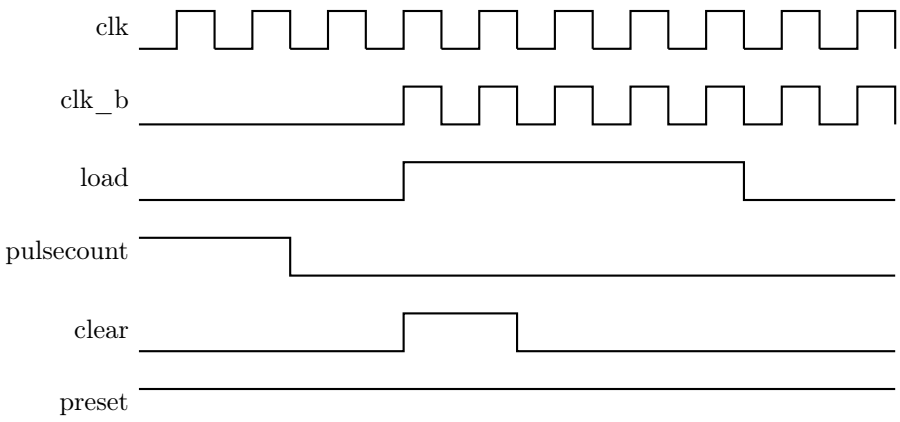


Figure 2.6: Programming the bits to "0"

2.2.3 Static to Dynamic

As described in the programming of the bits, it needs dynamic signals to be programmed. The signal *load* is controlling the update rate of pulsecount which again control *preset* and *clear*. One possible way to make a static to a dynamic signal is the use of delay elements for *load* which makes the signal $load\_D$ and a signal $loading = \overline{load\_D} + load$. The signal *loading* is an internal signal inside the "Static to Dynamic"-block and is not shown in the figures describing the signals, the same goes for the signal $load\_D$. This is because it will be equivalent to when the delayed version of *load* not yet have changed to the same value as *load*. It does not matter what kind of delay element which is used, only that it ends inside a low clock because the counting will begin at a high clock. The truth tables for *loading*, *preset* and *clear* is shown in table 2.1.

load	load_D	loading	loading	pulsecount[6:0]	preset[6:0]	clear[6:0]
0	0	0	0	0	1	0
0	1	0	0	1	1	0
1	0	1	1	0	0	0
1	1	0	1	1	1	1

Table 2.1: Truth table for loading, *preset* and *clear*

This truth table holds for all the bits except *b7*. That bit is only supposed to be programmed to "1" and never "0". The signal clear[7] and nmos is removed and a truth table for *preset[7]* is shown in table 2.2.

loading	preset[7]
0	1
1	0

Table 2.2: Truth table for *preset[7]*

A timing diagram showing the transformation of a static to a dynamic signal is shown in the figures 2.5 and 2.6. It should not be counting and programming simultaneously because it may cause glitches where the value. *load* should also only be active at rising edges, not both. A timing diagram describing the properties of the internal and external load signals is shown in figure 2.7.

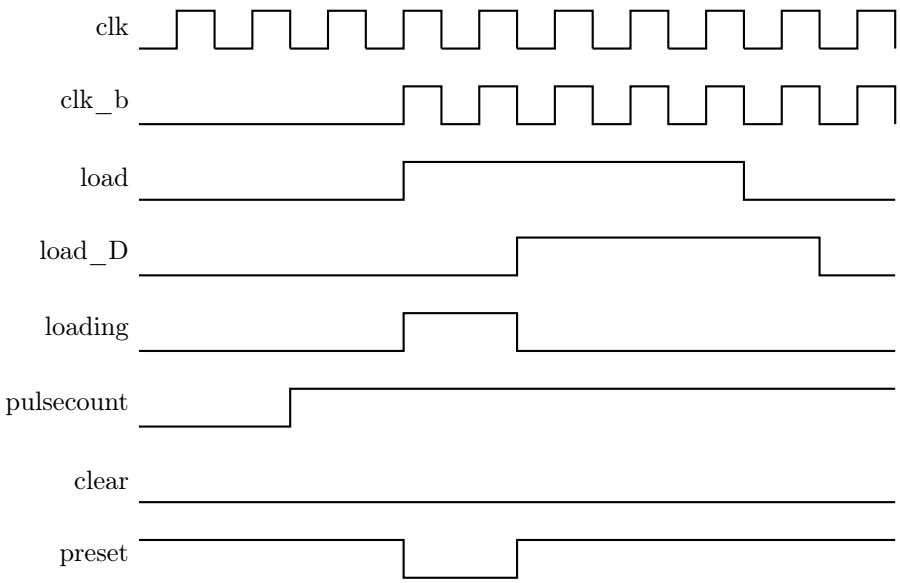


Figure 2.7: Timing diagram with *load_D* and *loading* and programming it to "1"

Looking at the figure it can be seen that *load* does not need to be perfect number of clock periods. The most important is that it end at a low clock and is long enough depending on how long it takes to program it and start the *clk_b*.

2.2.4 Clock gating

The clock gating is both a buffer and a circuit that can turn off the clock into the counter to save power and end the counting. The counter has no natural ending independent of the way it counts. When counting down, it will go to 127 again if it was not for *b7* which stops the clock (*clk_b*). The clock gating can be seen in figure 2.8.

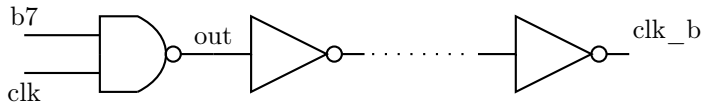


Figure 2.8: The clock gating circuitry

It is built up of a nand and an odd number of inverters. The number of inverters have to be odd because the nand works as a inverter when *b7* is "1"[3]. Else it is always turned on when the *b7* is "0". Truth table for this is shown in table 2.3 for the nand. The lower half of the table shows that it has the inverter properties when *b7* is "1".

b7	clk	out
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.3: Truth table for the nand in the clock gating

2.2.5 Enabling

The enabling circuit is the part that takes care of the correct amount of pulses are sent out and is shown in figure 2.9.

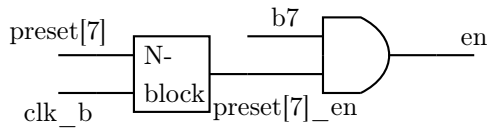


Figure 2.9: The enable circuity

The enable is a standard complementary CMOS and-gate and its inputs is *b7* and delayed version of *preset[7]*, *preset[7]_ en. preset[7]* is delayed with a N-block, making it output the value at a high clock. The reason these signals are chosen is because b7 is already a control signal. It controls the clock gating and will be active the number of pulses chosen and the time it is programmed. A positive clock delayed version of *preset[7]* is chosen as well because it is the programming of *b7*. The delayed version of *preset[7]* can only go high at high clocks is done with a N-block buffer and is when the programming end as shown in the timing diagram 2.10

[3]This can also be analyzed looking at it as a and gate with an even number of inverters

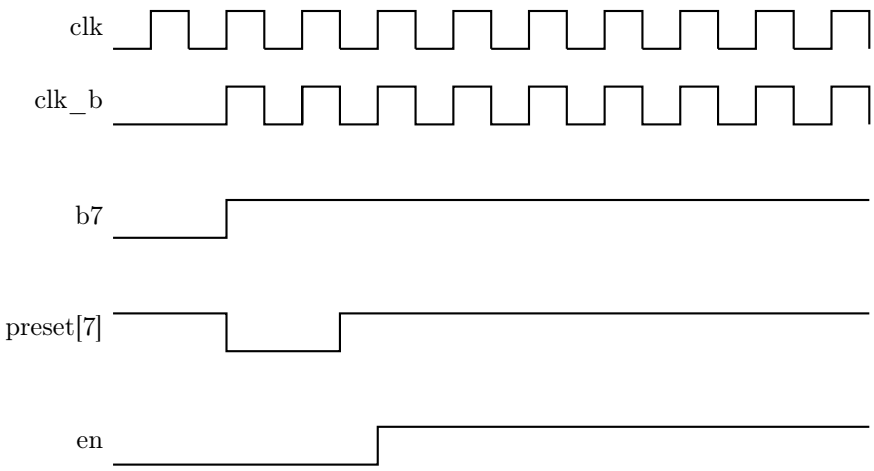


Figure 2.10: Timing diagram of the enable with a input of *pulsecount* higher than 7.

2.2.6 Loop with b7

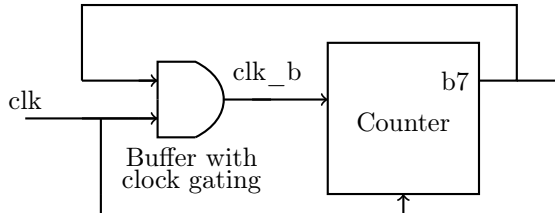


Figure 2.11: A close up picture of the loop *b7* make

As can be seen in figure 2.2 is that is contains a loop with *b7*, a close up picture is shown in figure 2.11. It is a loop in the sense that the *b7* is enabling the clock gating, but the clock gating is enabling the counter. This is possible because of how the counter is programmed, it can be programmed without having the clock on because the preset will pull *b7* high. If not enough time is given it will causes glitches and wrong values may be counted. The programming time need to be larger than the time it takes to pull *b7* high and enable the *clk_b*.

Chapter 3

Dynamic logic

There exist different logic styles, the most known is the complementary CMOS and follows strictly boolean algebra because of its P- and N-network (except latches and flip-flops). Another one which follow boolean algebra to some extend is the dynamic logic style. The reason it only follows to "some extend" is that it has monotonicy problem, it will not begin to rise or fall instantly as the complementary. This happens because the clock is also an input to the logic functions.

3.1 Counter principle

A counter, both up and down counter follow the principle by adding or subtracting 1 each clock period. Most counters are usually made with arithmetic [5] because it is one of the simplest functions. A fast counter is also the clock divider only divides the clock by two. This can be seen in figure 3.1 with its additions analogy.

$$
\begin{aligned}
000 + 1 &= 001 \\
001 + 1 &= 010 \\
010 + 1 &= 011 \\
011 + 1 &= 100 \\
100 + 1 &= 101 \\
&\ldots\ldots \\
110 + 1 &= 111
\end{aligned}
$$

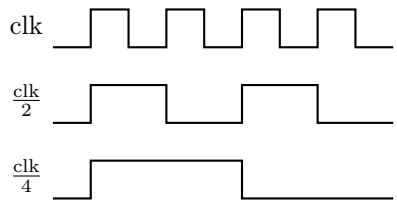


Figure 3.1: Counting as a clock division with the binary addition the left

The addition and clock division analogy also work for counting down and up for any number. It is not known to the author any counter in complementary CMOS which use clock dividers to count up or down.

3.2 Symmetry of rise and fall time of logic

To get a fast circuit, a simple but effective way is to design with symmetry in rise and fall time.

When a function is evaluated, a easy way to ensure it has enough time to evaluate it is to use the largest of the rise (t_r) and fall time (t_f), $\max(t_r, t_f) = t_{rf}$. If a system has a update rate, it will need to be slower than $t_{rf} + t$ where t are other delays. From this equation it is possible to see that it is no point in having a very fast rise time and a slow fall time, the fall time will then only slow down the system.

An easy (but powerful) approximation of the delay is $T_D = \frac{C}{I}$ where T_D is the delay in seconds, C is the capacitance and I is the current. From the transistor current equation it is known that the current is proportional to $\frac{W}{L}$ where W is the width and L is the length. This will also increase the capacitance because of a increased area. To increase the length will increase the rise or fall time, but this will only hold to the point where the capacitance increase more than the current. It's important to notice that an increase in current, so will the power. To get faster transistors, a tradeoff in power has to be made.

3.3 Buffer

A buffer could be two inverters in series where there is a increase in the inverter sizing. These are used when a function is going to drive large capacitive load[1]. It is done because if one small gate is going to drive a large capacitive load, it will need a lot of charge to charge it up or discharge it before it will reach the logic value it is supposed to. Charge is equal to current multiplied with the time, so if a large capacitor is going to be charged up to a logic "1" or charged out to "0". It will be faster to do this with larger gates and therefore adding buffers. This will add delay through the buffers, but will be less than the delay of charging or discharging of capacitive load. A figure of this can be seen in figure 3.2.

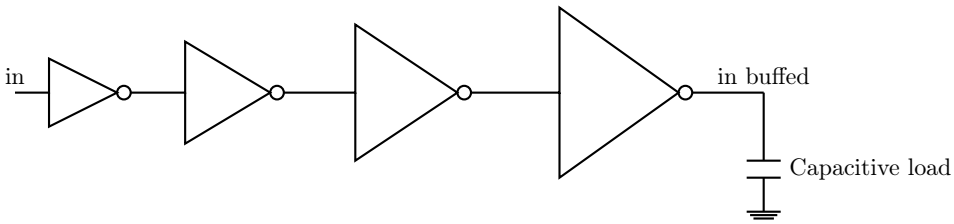


Figure 3.2: Buffers driving a large capacitive load

In the figure the increase in the inverters width is shown with a larger inverter. From [8] it can be shown that the best increase in the widths is by e. This is for

[1] Capacitive load are often used to model transistors

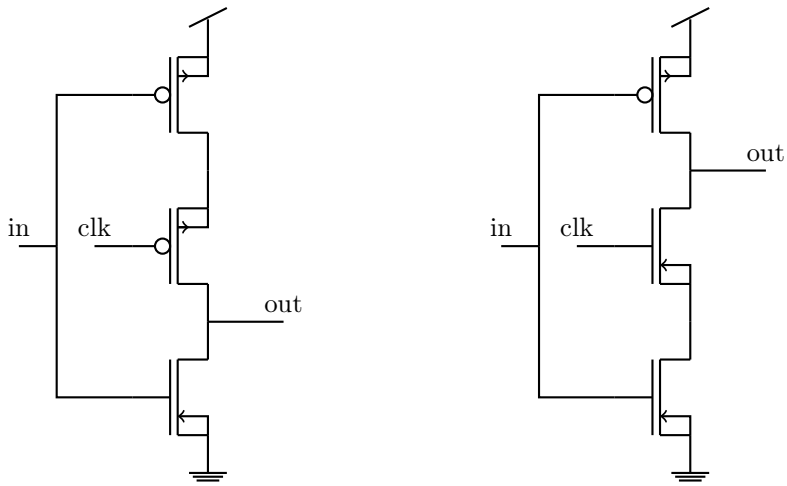
an ideal case, and with parasitic capacitance from layout the optimum would be larger.

3.4 Dynamic gates

Dynamic gates are a logic style which add the clock as a input too its gates. In this text a set of building blocks to the dynamic gates style is presented. These building blocks use both the clock as a input with either the P-network, the N-network or both as a input. It has many of the same properties as the complementary, but as already described it will suffer from monotonicy problems because of the clock. These blocks have rail to rail and can be used with the complementary.

3.4.1 P- and N-C^{2}MOS

P- and N-C^{2}MOS are building blocks in the dynamic logic and is shown in 3.3, both are inverters. These blocks are sometimes referred to as SP and SN (static pmos and nmos) [9]. The names may be misleading, but it can either has a partly static P- or N-network depending on if it is P- or N-C2MOS. The partly static is because it has a clock input to one of the network[2].



(a) P-C^{2}MOS (SP) (b) N-C^{2}MOS (SN)

Figure 3.3: The P- and N-C^{2}MOS

The SP's *out* will at high *in* begin to fall to "0" regardless of what *clk* is. When the *in* and *clk* is both low it will rise to "1". When *clk* is low it can't rise to "1", this will lock *out* to what it was last. For SN's *out* it will fall to "0" when both the

[2]Notice that the clock transistor is not a part of the network because this will mean it has two clock transistors in total, one at each side

clk and *in* is high. If *in* is high it will rise to a "1". It can't fall low when *clk* is low and locking *out*. While it can rise to "1" regardless of what the *clk* is when *in* is low. When the outputs are locked, it can fall to a "0" for SP and rise to a "1" for SN. It's locked because if *clk* fall or rise after *in*, the output will be locked to a "1" or a "0" for SP and SN if *in* don't change.

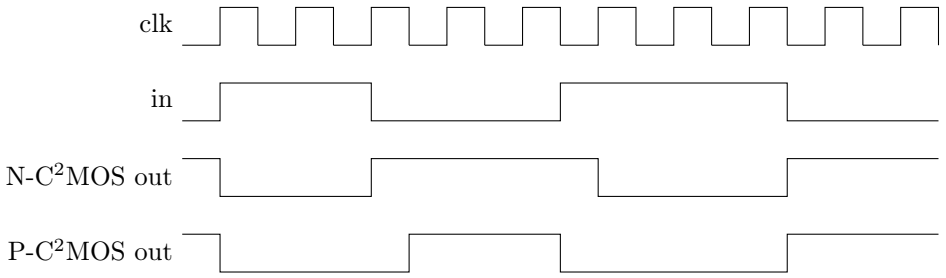
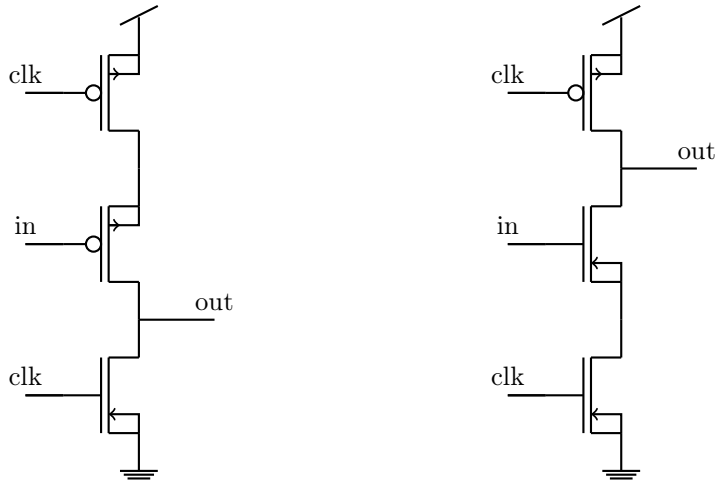


Figure 3.4: The timing diagram for the P- and N-C^{2}MOS.

3.4.2 P- and N-precharge

The P- and N-precharge blocks are shown in figure 3.5. They are referred to as PP and PN (precharge pmos and nmos) [9]. In a sense, they are the inverse of the P- and N-C^{2}MOS, the network and the clock have been changed. It has either the P- or N-network with a clock transistor at both sides.



(a) P-Precharge (PP) (b) N-precharge (PN)

Figure 3.5: The P- and N-precharge

The PP's *out* will fall to "0" every high *clk*. It can only rise to "1" if both *in* and *clk* is "0". If *in* is high *out* will be locked to "0". For the NN's *out* it is the

reversed, it will every *clk* rise to "1". When both *in* and *clk* is "1" it can fall to "0". When *in* is low *out* will be locked to "1". The timing diagram for the P- and N-network is shown in figure 3.6.

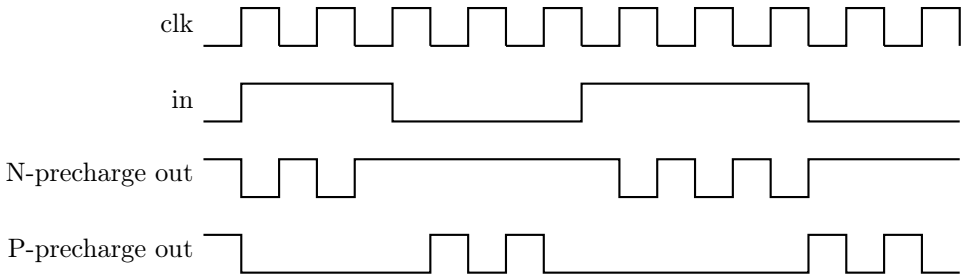
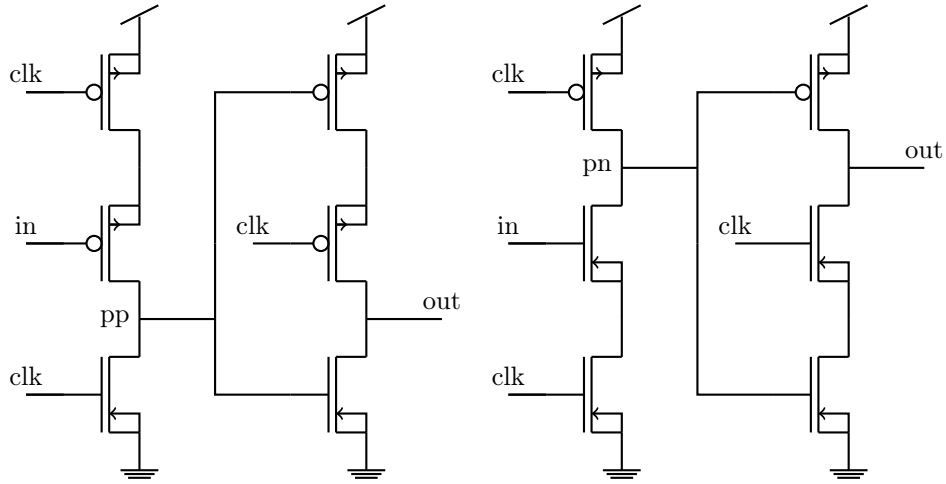


Figure 3.6: The timing diagram for the P- and N-precharge.

3.4.3 P- and N-blocks

The P- and N-blocks can be seen in figure 3.7 [9]. It is a precharge and a C^2MOS in serie.



(a) P-block (b) N-block

Figure 3.7: The P- and N-blocks build up by P- and N-C^2MOS and P- and N-precharge

The P-block works as a low clock triggered latch while the N-block works as a high clock triggered latch. For the P-block, *pp* will rise and fall with the cycles of *clk* when *in* is low. This means that *out* only can rise if *pp* is locked to "0". If *in* goes high in a high *clk*, *pp* be locked to "0". The next low *clk* *pp* will be locked to low, making *out* rise to "0". When *in* then fall low at a high *clk* nothing will

happen at *pp* before *clk* goes low. When it goes low it will make *pp* go high and *out* fall to "0". For the N-block it will be reversed, *pn* will rise and fall when *in* is high with the *clk* rise and fall with its cycles. When *in* rise to high when *clk* is low, *pn* will fall to "0" the next high *clk*. This will make *out* rise to "1". When *in* fall to "0" at a low *clk*, *pn* will be locked to "1". The next high *clk* it will make *out* fall to "0". This concludes the operation of the P- and N-block, and a timing diagram is shown in 3.8.

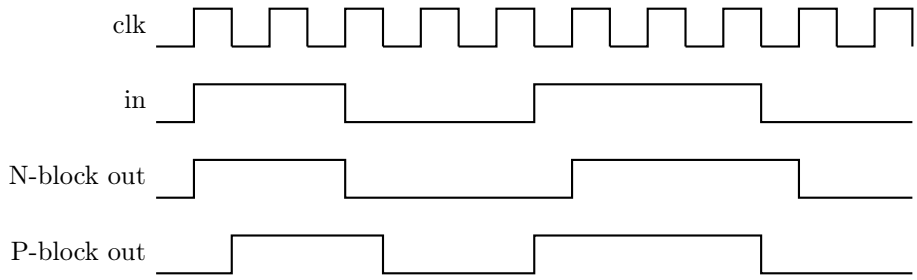


Figure 3.8: The timing diagram for the P- and N-block

3.5 True single phase clock logic

The true single phase clock logic (TSPC) is as the name suggests a single phase clock logic [9]. It is built up of P- and N-precharge, P- and N-C^{2}MOS blocks and similar structures. The TSPC do not need the inverted clock, and of this reason can be faster than complementary CMOS which need it. And action against overlapping clocks reduce the speed [2]. If no action is taken, it will cause short circuit in the circuit which will increase the power. A figure of clock which don't overlap is shown in 3.9.

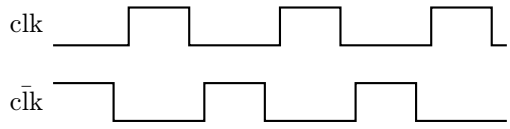


Figure 3.9: Overlapping clocks

3.5.1 Carry-look ahead

The carry-look ahead function is to output *ci* as a "1" if all the input bits are "1". *ci* is the carry of the next bit. This can be done with a P-block where the precharge part is a nand, seen in figure 3.10.

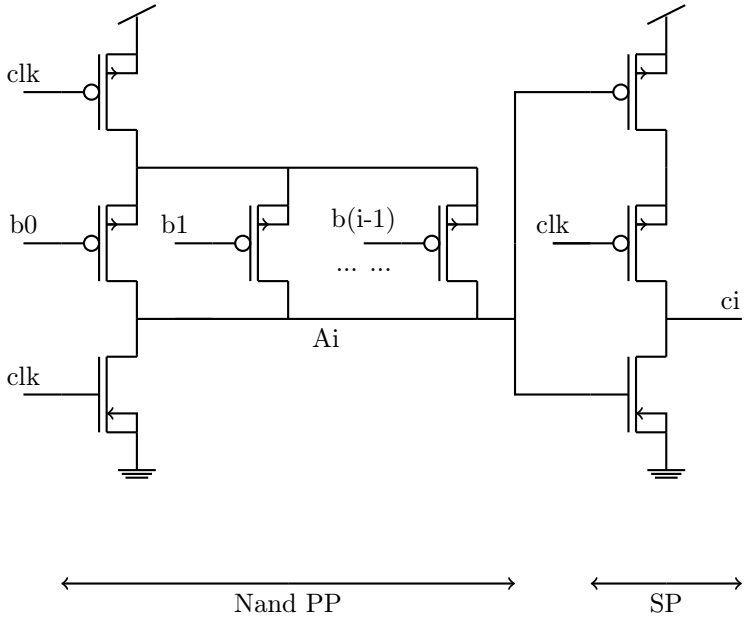


Figure 3.10: Carry-look ahead logic for dynamic logic

The Ai from the nand PP in the figure will always give out a "1" when the *clk* is low if not all the bits are "1". This will make SP be turned off and ci will be "0". Every high clocks the nand outputs An will be drawn to "0" because it is a PP. The SP can only rise at low *clk*, and it is only possible for it to rise when a previous state from the PP-part is locked. This happens when all the bits are "1" locking An to be "0" from the high *clk* and make ci rise to "1". ci is locked until An rise to "1" which wil happens the next low *clk*[3]. This is also the same as the timing diagram for the P-block in figure 3.8 except that the input *in* is multiple bits.

3.5.2 Toggle

The toggle is a circuit which toggle the output to change its value from a "1" to "0" and a "0" to "1". This can be done with the circuit in the figure 3.12 [10]. It is built up of a SP, PP and SN where the PN and SN have two inputs[4].

[3]This will happen because the circuit is counting
[4]The *clk* is not counted as an input, only ci and the output of the previous part

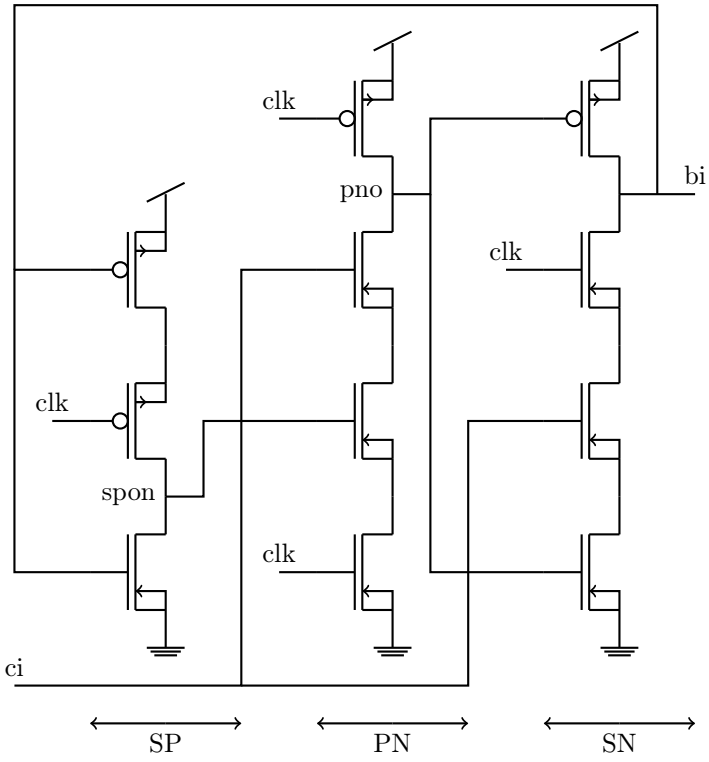


Figure 3.11: Toggle circuit

The SP part evaluate the function at a low clock while the PN and SN part does the evaluation at a high clock. The toggle circuit works in the way that the PN and SN parts can't be turned low when *ci* is "0" because it turn off the nmoses. This is because *pno* is locked to a high and turning off the pmos in SN. *bi* is therefore locked to what it was last because the nmos with *ci* is turned off. The *pno* is locked to a high because *clk* will turn it high every low clock. *bi* is feed into SP and if *bi* is "1" it will turn *spon* to "0" and if it was "0" it will turn spon to "1" the next low clock and be locked there when the *clk* goes high and low. When *ci* turns to "1" at a low clock (this will hold in the next high *clk* cycle), nothing will happens before *clk* goes high. When it does it will toggle the values because if *bi* was "1", *spon* will be "0". This means that the *pno* will be "1" and turning the SN low, which is *bi*. When *bi* falls to "0" it will turn on the pmos in the SP and make *spon* go high the next low clock. When *bi* is "0" the *spon* will be locked at a "1". Of this reason *pno* goes to "0" and turn *bi* into "1". This will make *spon* fall to "0". This conclude the operation of the toggle circuit and can be seen in the timing diagram in figure 3.12.

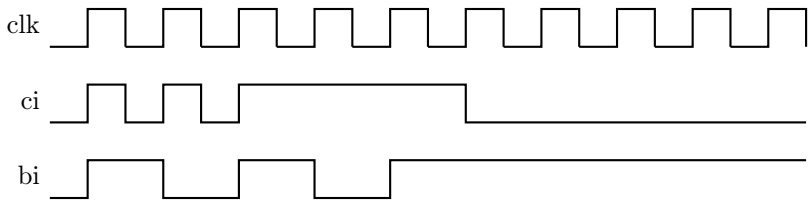


Figure 3.12: Timing diagram of the toggle when the dynamic condition is satisified

Show in figure 3.12 the toggle can be seen to work as a divide-by-two circuit. This happen when *ci* is high constant. If the *ci* transistors are removed it will always divide the *clk* by two. It is also shown that when *ci* is low *bi* will be saved from period to period. But this is not entirely true, this is only when the dynamic condition is fullyfied. It is therefore a dynamic version of the toggle.

Divide-by-two

When the input *ci* always is "1", the toggle circuit will work as a divide-by-two. The circuit will then be equivalent to the circuit shown in figure 3.13, there the ci transistors are removed.

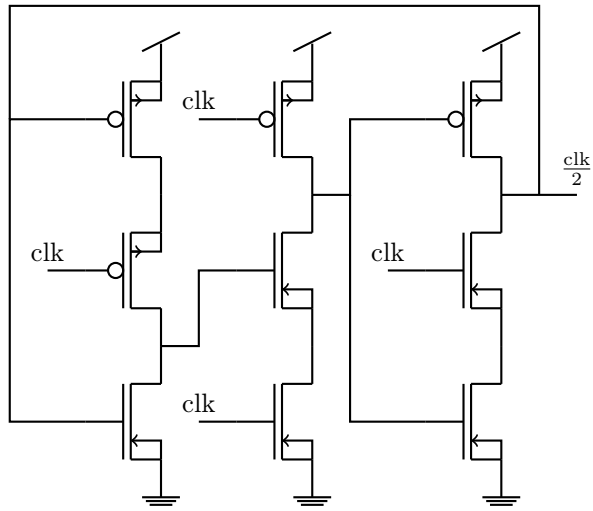


Figure 3.13: The toggle as a divide-by-two circuit with the *ci* transistors removed

Toggle with dual-inverter locking loop

When the dynamic condition is not satisfied, is when the output *bi* is not saved from one clock period to the next period. The circuit in figure 3.14 can be used. The last part added is a dual-inverter locking loop (often shorted to dual-inverter in this text). It will help pull it up and down and make it into a static version of the toggle circuit.

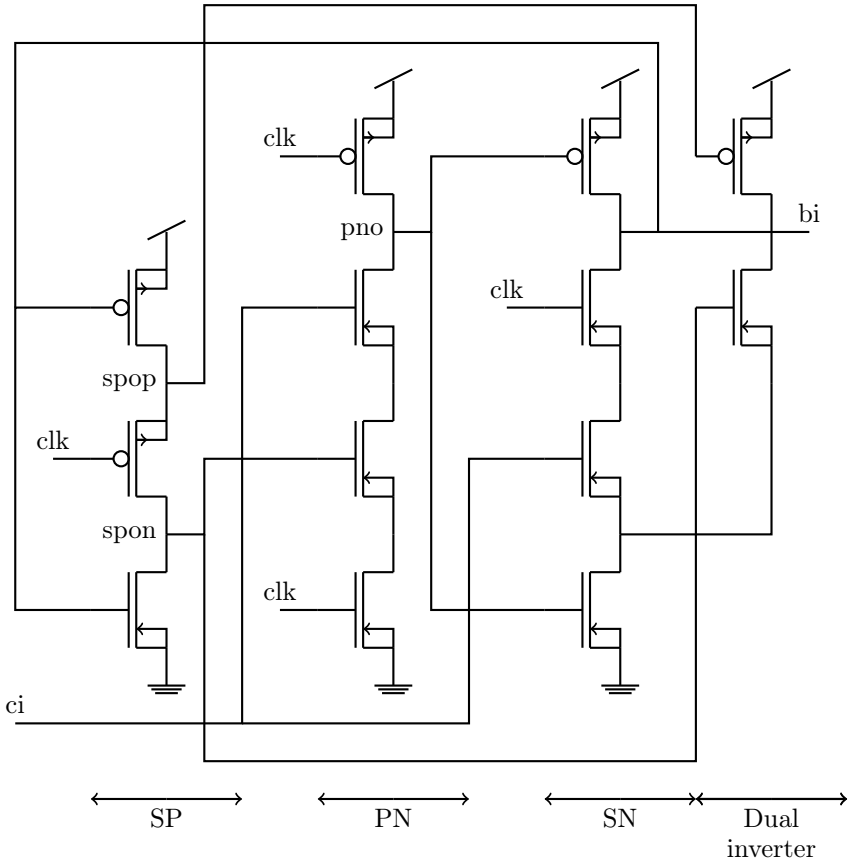


Figure 3.14: Toggle circuit with dual inverter

The dual-inverter works by taking *spon* and *spop* and feeding it into a nmos and pmos. The nmos is connected to the drain of the nmos which is connected to ground for the SN part and the pmos is connected to the supply. When *bi* is "1" and *ci* goes high, the value at *spon* will be "0" until the next low *clk* and be turned to "1". *spop* will be a "1" turning of the pmos in the dual-inverter. Because *spon* is low, *pno* will be high. This makes *bi* disharges to "0". The next low *clk* *spon* will be charged up to a "1" turning of the nmos in the dual-inverter. In the PN part *pno* is a "1" and it will turn on the nmos to ground in the SN part. This will pull the *bi* to a "0" because the nmos in the dual-inverter is on. When *bi* is "0" and *ci* goes high, *spop* will be locked at "1" until the next low *clk* turning of the pmos. *spon* will also be "1" until *bi* change, turning *pno* low. Because *pno* is low it will turn *bi* high. The next low clock *spop* will discharge to "0" turning of the pmos. From these descriptions, the dual-inverter will first begin to work in the low clock cycle after the toggling has happend. The timing diagram of how it acts when the dynamic condition is not fulified with the static and dynamic version of the toggle can be seen in figure 3.15.

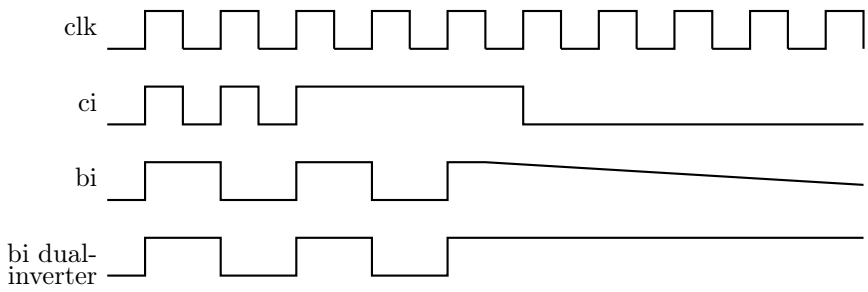


Figure 3.15: Timing diagram of the toggle circuit when the dynamic condition is not satisfied.

As seen in the timing diagram is that the output of the toggle circuit without the dual-inverter keeps falling. Leakage will occure especially when *clk* is turned on and off, while the dual-inverter will counter these effects.

Toggle with programming

To program the counter, a preset and a clear transistor have been used as in [6]. This toggle circuit can be seen in figure 3.16. The timing diagram for how it work with *preset* and *clear* can be found in section 2.2.2.

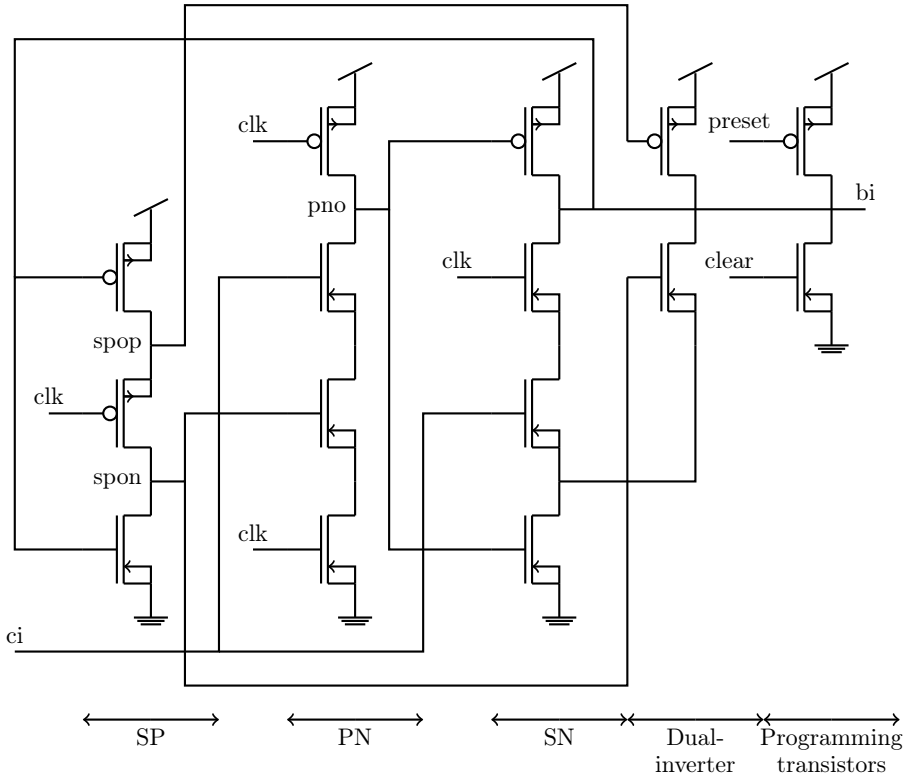


Figure 3.16: Toggle circuit with programming and dual-inverter

3.5.3 Single-clock dynamic flip-flops

Different types of latches and flip-flops are described in [9]. A SX, PX and FL(Y) flip-flop is chosen, where X stands for N or P and FL(Y) stands for a full-latch where Y is P or N. It is a single stage TSPC which can latch both high and low inputs. The N- and P-flip-flops can be seen in figures 3.17 and 3.18.

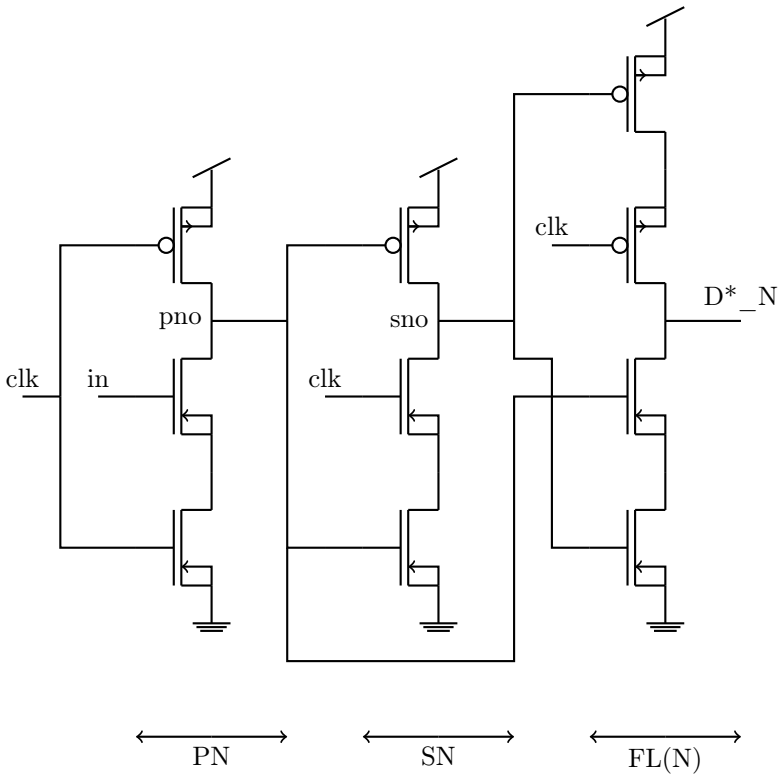


Figure 3.17: N-flip-flop

For the N-flip-flop when *in* goes high at a low clock, *pno* will be locked high until the next low clock. It will then begin to go between high and low. When *pno* goes low at a high clock, it will makes *sno* goes high. Because *pno* will be "0" at high clocks and "1" and low clocks and *sno* is locked at "1", then FL(P) output D^*_N will be locked at "0" at the next high low clock. It have then been one whole clock periode since *in* went high. When *in* goes low at a high clock, *pno* will be locked high. This will make *sno* become low and locked there. Because *pno* is "1" and *sno* is "0", this will make D^*_N rise high at the next low clock, which take half a period.

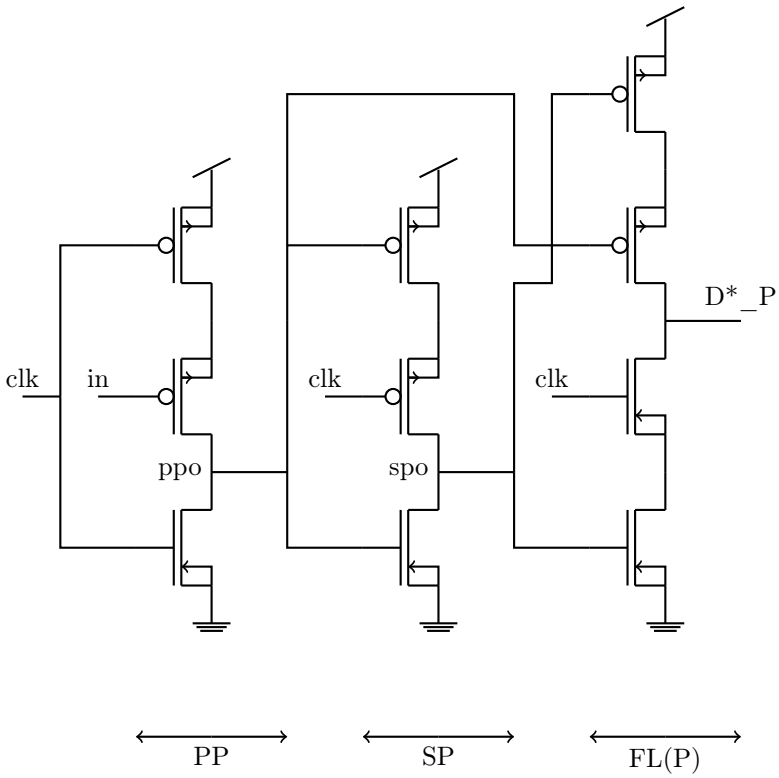


Figure 3.18: P-flip-flop

If *in* goes high at a low clock for the P-flip-flop, *ppo* will be locked at low. This will make *spo* be locked at a high. Then *ppo* is "0" and *spo* is "1" and $D*\_P$ will the next high clock be locked at "1". This is half a period after *in* changed. If *in* goes low at a high clock, *ppo* will first rise high the next low clock, and then it will switch between low at high clocks and high at low clocks. *spo* will then be locked low. This make *spo* "1" and *ppo* will fall to "0" the next high clock locking $D*\_P$ at "1", this takes one clock period.

The examples for the N- and P-flip-flops are described with a *in* as "1" at a low clock and *in* as a "0" for high clock. This also holds the reversed, that they respectivly rise high or fall low at a high and a low clock. If *in* for the N-flip-flop change at a low clock, it will change the output after 1 period. If it changes at a high clock it will go half the periode before it change. This is the reversed for a P-flip-flop, if *in* change at a low clock, it will change the output after half a period. If the change happens at a high clock it will change after 1 period. The timing diagram for the flip-flops are shown in figure 3.19.

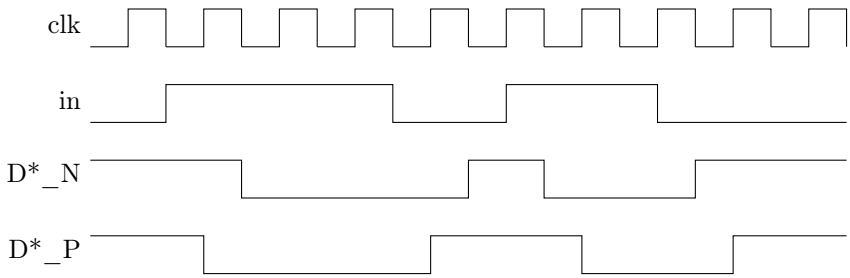


Figure 3.19: Timing diagram for the flip-flops

3.5.4 Counter element

The complete counter element for a TSPC toggle counter is the CLA and the toggle in serie for the i-bit. A figure of this can be found in appendix 7.1 in figure 7.1. The first bit (b0) can be made by connecting the supply to the *ci* (making a divide-by-two circuit). How the toggle and the CLA work is descirbed in section 3.5.1 and 3.5.2. The counter element works in the way that the CLA evaluate at low *clk* while the toggle will evaluate at high *clk*. The evaluations are separated in different *clk* cycles making it possible to increase the speed of the *clk*. The logic depth is also lower because it is evaluated in different cycles[2]. The CLA will output a "1" *ci* if all the bits are "1" in the low *clk* and lock it into the high *clk*. This will trigger the toggle to change *bi* from "0" to "1" or "1" to "0". The TSPC toggle counter works in a similar way as a divide-by-two counter. The CLAs synchronizes all the toggle bits, making it act as a divide-by-two counter as can be seen in figure 3.1. If *bi* is the input of the CLA, it will count up. If $\bar{B}i$ (the inverted) is chosen instead it will count down[6].

Chapter 4

Speed of Transistors

The intrinsic speed is the maximum speed a transistor can have when it is biased at a voltage and the small-signal current out exceeds the current in from the gate. A first-order equation can be found 4.1[1].

$$f_t \approx \frac{g_m}{2\pi(C_{gs} + C_{gd})} \approx \frac{3\mu_i V_{eff}}{4\pi L^2} \tag{4.1}$$

In equation 4.1 the $V_{eff} = V_{GS} - V_{ti}(V_{SB})$. The intrinsic speed of the transistor is given by both the length of the gate, L, and the $V_{SB} = V_S - V_B$. The voltage at base, V_B of the transistor is possible to change. One important effect of changing the base voltage is that it will increase or decrease the current through the base of the transistor. This base current must be inside a valid region of what the transistor can handle.

Chapter 5

Results

The result in table 5.1 and 5.2 is for 64 pulses with a sinusoidal clock with 1 volt peak to peak (zero to one) and a frequency of 12.5 and 9.5GHz respectively over process-, voltage-, and temperature variations (PVT). More PVT results can be found in appendix 7.6. The length of the simulation have been kept at a minimum, but is 13ns for all the simulations of PVT except the table 7.8 which is 17ns. This is because 127 pulses at 9.5GHz is longer than 13ns. The variations of the temperature are over [-40,80], the voltage variations is 2.5% of the supply which is 1 (0.975 and 1.025) and over the corners SS, SF, FS, FF and TT. "S" stands for slow, "F" stands for fast and "T" is typical where the first letter is the nmos and the second letter is the pmos. The numbers in the table is *en* from figure 2.1. If there space in the table is left blank, it is error in the counting. The result in table 5.3 is PVT simulation for a clock with the frequency 11.5GHz, but a wider voltage range of 5%. Only the maximum and minimum power are added for each column. The length of *en* is calculated from equation 5.1.

$$\text{en} = \frac{\text{t}_{\text{en-fall}} - \text{t}_{\text{en-rise}}}{\text{T}} \tag{5.1}$$

In the equation T is the period which is equal to $\text{T} = \frac{1}{\text{f}}$ where f is the frequency of the pulses, and $\text{t}_{\text{en-fall}}$ and $\text{t}_{\text{en-rise}}$ are the times where the signal *en* fall and rise to 90% of its value.

5.1 PVT with 64 pulses

T	V	SS	SF	FS	TT	FF
-40	0.975	63.91	63.96	63.92	63.95	63.95
0	0.975	63.86	63.94	63.9	63.94	63.94
80	0.975	63.72	63.89	63.75	63.92	63.93
-40	1	63.94	63.98	63.94	63.96	63.96
0	1	63.91	63.97	63.93	63.96	63.95
80	1	63.8	63.93	63.87	63.94	63.95
-40	1.025	63.96	63.99	63.96	63.97	63.97
0	1.025	63.94	63.98	63.95	63.97	63.97
80	1.025	63.86	63.95	63.91	63.96	63.9
	max power[W]	1.496m	1.577m	1.545m	1.625m	1.885m
	min power[W]	1.256m	1.318m	1.275m	1.319m	1.414m

Table 5.1: Results for the counter with different voltage, temperature and process corner with 64 pulses (*pulsecount* = "0111111") at 12.5GHz.

T	V	SS	SF	FS	TT	FF
-40	0.975	63.95	63.97	63.94	63.96	62.99
0	0.975	63.95	63.96	63.94	63.95	62.97
80	0.975	63.94	63.96	63.93	63.94	
-40	1	63.97	63.98	63.96	63.97	63.01
0	1	63.96	63.98	63.95	63.97	62.99
80	1	63.96	63.97	63.95	63.96	
-40	1.05	63.97	63.99	63.97	63.98	63.03
0	1.05	63.97	63.98	63.97	63.97	63
80	1.05	63.97	63.98	63.97	63.97	7.988
	max power[W]	1.447m	1.503m	1.499m	1.543m	2.263m
	min power[W]	1.202m	1.238m	1.23m	1.239m	718u

Table 5.2: Results for the counter with different voltage, temperature and process corner with 64 pulses (*pulsecount* = "0111111") at 9.5GHz

T	V	SS	SF	FS	TT	FF
-40	0.95	63.92	63.96	63.92	63.94	63.94
0	0.95	63.89	63.95	63.91	63.94	63.94
80	0.95	63.8	63.92	63.85	63.92	63.92
-40	1	63.97	63.99	63.96	63.97	63.97
0	1	63.96	63.99	63.95	63.97	63.97
80	1	63.91	63.97	63.93	63.96	63.96
-40	1.05	63.99	64.01	63.98	63.99	63.99
0	1.05	63.99	64.01	63.98	63.99	63.99
80	1.05	63.97	64	63.97	63.99	63.99
	max power	1.574m	1.656m	1.637m	1.701m	1.933m
	min power	1.16m	1.212m	1.177m	1.213m	1.295m

Table 5.3: Results for the counter with different voltage, temperature and process corner with 64 pulses (*pulsecount* = "0111111") at 11.5GHz with 5% variation in the voltage.

Chapter 6

Discussion and conclusion

6.1 Discussion

The discussion begins with the choices that have been made throughout the master period and ends with the results and the conclusion afterwards. The results are not compared to other results in [6], because the additional circuitry makes a significant difference even though it's written it's a resimulated counter. The counter has also only been made in schematic, and the power will therefore be seen as an upper limit due to the reduction in speed if layout is made. This also means that the dynamic power consumption will fall, but the static will remain the same.

The pulse counter was the only part that was designed and simulated of the pulse generator in schematic. The sizing of transistors was chosen at minimum threshold voltage for two fingers for all the transistors if nothing else is specified. This was because the results had shown the increase in the speed was not that much greater for a ring inverter [4]. The sizes for three compared to two fingers was almost the double, doubling the power. It was therefore a trade-off in power and speed and 2 fingers was chosen. It was found that only the counter would work at 12.5GHz for minimum threshold voltage for 1,2 and 3 fingers transistors. None of the transistor were increased in lengths, only in widths. This were done because of equation 4.1 showing that an increase in the length would reduce the speed with a factor of the length squared. Another possibility was to add a base voltage; this has not been done. If a base voltage was to be added, it would increase the speed at the cost of complexity. This is because a voltage reference must be designed to get the wanted voltage at the base. The base voltage is therefore connected to the source of both the nmos and pmos, this is also shown in all the figures with transistors.

The assumption that has been made in the optimization of the sizes of the toggle and CLA were too optimize rise and fall time, rather than power. The CLA was divided into the Nand PP and the SP inverter, these were optimized for the

A1 and c1 (notice that this is the second bit). The values found were used for all the bits. For the Nand PPs for the higher bits only the pmos was copied because the worst case was kept in mind. The pmos that have the input *b0* for the first Nand PP (b1) would be copied for the higher bits. The reason for this was that if all of pmoses was off (the best case). The rise time for the Ai would rise faster for larger i because it would be more pmoses to drive it to "1", this was only limited by the *clk* pmos in series. But when all the bits except one was on. Say that for the CLA for bit eight (*b7*), the seventh bit (*b6*) was "0" being the only pmos pulling the output high. If the size of that transistor was less than the other, the output may not rise as fast and high as it should. This would make the SP see *A7* as a "0" instead of a "1". This again would make *c7* be "1" when it was supposed to be "0". To speed up simulations all the toggle circuits was equal, making all the *ci's* drive the same load and therefore only the SP values for bit 1 was used for every bit. For the toggles, there was added an inverter at the Bi to subtract instead of addition. This were the optimized for the inverter and the SN structure in the toggle because this was the major contribution of the rise and fall time. All the toggles were optimized to be equal, but the inverters were optimized differently.

The reason for counting down instead of up was because there were two possible ways to count. It was possible to count down from *pulsecount* to zero or up from $127 - pulsecount$. It was not considered to count from zero and up because it would then need to check all the bits. To check all the bits will require a 14-bits equal function which most likely would generate large delays. Another possible solution was to add the eight bit, *b7*, and use that as a control signal. This is because if it is initialized to "1" it will turn to "0" when the counting is finished. This solution was possible with both the down and up counting ($127 - pulsecount$). But the up counting would probably require a buffer, especially for the lower bit which would drive all the higher bits CLAs. The down counter will only need one inverter, making it has less delays[1]. The down counter would therefore most likely be able to operate faster than the up counter and chosen.

In the figure 2.2 showing the inside of the pulse counter it is seen that it has two buffers for the clock. These buffers have a total of 6 elements for the *clk_b* and 4 elements for the *clk_sta*. These numbers were found just by checking how many buffers were needed to be able to drive the system. Because the nand and the first inverter in the *clk_b* was minimum, making the buffer have 1 dummy inverter because it needs the increase in the sizes in four stages. The same for the *clk_sta*, it has four inverters where 1 is a dummy because it only needs the increase in the sizes in three stages. The increase in the length was a factor of 3 (close to "e") for each inverter compared to the one before it. Some tweaking were needed to get a smoother clock as shown in section 7.2. By smoother it means that the low and high part was more symmetrical. But these buffers will make delay of the clock signal, as is shown there. What can be seen is that the *clk_sta* is almost 180 degrees out of phase compared to *clk_b*. This affects the choice of flip-flop

[1] 1 inverter will have less delay than 2 inverters

because preset and clear signal should ends inside the low clock of *clk_b*. And if these two buffered clocks were in-phase it should have been a N-flip-flop. But because of the phase a P-flipflop is chosen. The timing diagram for the flip-flops can be seen in figure 3.19 and simulation of the flip-flops with 1 and 6 elements in series can be seen in the appendix 7.3. Because the output of one flip-flop is the inverted input, there are a 6 in series (even number) for the delay element in the "Static to Dynamic"-block. This is also done to get enough time to program the counter and start *clk_b*. Even though 6 flip-flops in series is a little too much, it is also a safeguard against *load* signal which comes at a low clock and make it lose half a period compared to if it comes at a high clock. This effect is shown in figure 7.9 with 6 P-flip-flop in series. From figure 7.12 it can be seen that if all the preset and clear transistors are minimum, *b7* can't go high and start the counter. In fact, none of the bits can be pulled high. Of this reasons all the sizes of the preset and clear transistor have been chosen to be 3 times larger than minimum width.

All the figures described next are found in the appendix 7.4. The simulation with and without the dual-inverters are shown in figure 7.11 and 7.15. From the figures, it can be seen that the dynamic condition is broken already at *b2*. To get the toggles to work they will need to be the static versions. By also comparing figures, it's seen that the programming phase is more stable with the the dual-inverter (the bits do not have large variation while it is programmed). It is also seen that the programming is most unstable for the lower bits, especially *b0*. The dual-inverter is of this reason added to all the bits both because the dynamic condition is broken for the higher bits (*b2* and above) and because the lower bits are more stable while it programs, even when the dynamic condition is satisfied for these bits. The sizes of the dual-inverter transistors is 2 times the minimum width.

From the PVT results in sections 5 and the appendix 7.6, it's showed that pulse counter was tested at both 12.5, 11.5GHz and 9.5GHz with 64 pulses. The appendix 7.6 has the values 1,2,3 and 127 at 12.5 and 9.5GHZ. The reason for 12.5GHz were to get the highest possible speed because of the uncertainty of the speed in layout by this relatively large system (compared to a ring oscillator with three elements). The higher the speed in schematic, the more likely it is to work in layout. The number of pulses was chosen to check the low values, 1,2 and 3, values where all the bits change in the beginning, 64, and where all are high, 127. Odd and even number is also checked with these numbers. The main problem in this circuit is the speed and the different delays from the clocked. Two different clock signal and the enable output is shown in 7.16 and 7.17. Here it's shown that by changing the condition from bad to good changes the signal a lot. The design have to work over these changes. It is also shown that the different conditions have different delays of the *clk_b* and *clk_stat*. It was found that the P-flip-flop worked over most of these changes and was of this reason used. It can also be seen from the tables for the PVT that it counts very linearly for the different regions for 12.5GHz. The error in the counting is not very large, and for the same conditions it almost ends at the same decimal for the different numbers, thereby counting

linearly. The results for 11.5GHz was added to show how much a decrease in the speed would make a possible increase in the voltage variation because a voltage regulator will have a larger variation than 2.5% [2]. For the 9.5GHz it's seen that it has a lot of empty spaces, meaning that it's an error in the enable. The problem with this can be seen in figure 7.18. As is shown in the figure is that the clocks not are very much out of phase, the high clk_stat end halfway inside clk_b. This makes it program and count at the same time, and none of the Ai have the time to rise enough, ending the counting by toggling the $b7$. This is shown in figure 7.19. The calculator which calculated the values from the simulation had 3 significant bits. This is the reason why the counting of 127 pulses not always have decimals.

The maximum power for the counting at 12.5GHz in a 13ns simulation is 2.749mW. This is both the static and the dynamic power consumption. The clock buffer helps shut off the counter when the counting is finished reducing the power. For the counting of 1 pulse the maximum power consumption has been 682.7uW. This difference is a factor of 4 which is high and much power is most likely wasted from the buffered clock which not is clock gated, clk_stat.

From the results, it is also possible to see that the length of *en* is not the exact right amount of pulses, it has an error. The bandwidth function BW = $\frac{1}{T}$ is a good approximation for the bandwidth. From this equation it can be understood that the bandwidth, BW, is more sensitive to variation when the pulse length, T, is small. The difference between 0.9 and 1 pulses is larger than 63.9 and 64 pulses. And for 127 pulses which do not have decimal (only three signifcant numbers) is not a problem because it is a good indication that it works, and the excessive bandwidth by this error is small. It is also important to understand that *en* will drive a transistor of some sort which will work as a switch. The output V_{out} can't of this reason not fall and rise instantly, smoothing out the edges of the signal. This is because the transistor can be view as a RC-circuit.

6.2 Conclusion

A possible solution to a pulse counter which can be a part of a pulse generator have been designed and simulated. The pulse counter can be controlled by a lower frequency system. The PVT analyses shows that it works at 12.5GHz and for some values of 9.5GHz. Delays between buffered clocks and their variations is the main problem with getting 9.5GHz to work. The excessive bandwidth because of the error in the length of *en* is tolerable. Statistical simulation have not been done because of the problems in the speed, voltage variation and the result for these values don't pass for 9.5GHz. If a reduction in speed is with a factor of 2, it will not work for the highest frequencies, only the lower ones. It would at least need to be able to operate at 14GHz if a factor of 2 is used. The voltage variation is also too small, but by reducing the frequency it is possible to increase the voltage variation.

[2] A comment from Trond Ytterdal

This can be fixed by either increase or decrease (nmos or pmos) the voltage at the base of the transistors, or a reduction in the length of them can be made. If a reduced length is chosen, the technology must be changed. The difference between the power for counting 1 pulse and 127 pulses is by a factor of 4. This means that it uses much static power compared to dynamic which should be reduced.

6.2.1 Future work

The future work described what should be noticed with this design and what should be changed to get it more likely to work. The VCO and the switch should be made to complete the pulse generator. And the circuit should be made with minimum threshold voltage transistor with 3 fingers to get it more likely to work at a larger frequency, and maybe even more fingers. This is because the speed with more fingers will be closer to the intrinsic speed. A clock distribution should also be designed to make the buffers more realistic. The large inverters in the buffers should be chopped into many small depending on the distribution. The buffers should also have an increase in size of 4 and not 3 [8] and getting the buffered clocks more in-phase. If the clock is in-phase the flip-flop must be changed to a N-type because it ends at a low clock. The clock for the “Static to dynamic”-block should as well be clock gated to further reduce the power.

Chapter 7

Appendix

7.1 Counter element for the i-bit

7.2 The different clocks

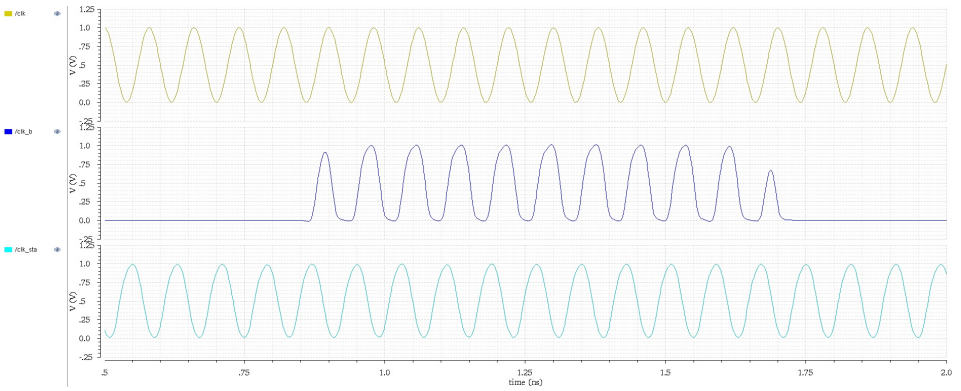


Figure 7.2: Comparission of the true clock signal and the buffered clocks at room temperature, TT corner and 1V supply with 5 pulses.

7.3 Flip-flops

This simulation have been done at 80 degrees, with a voltage supply of 1V in the SS corner.

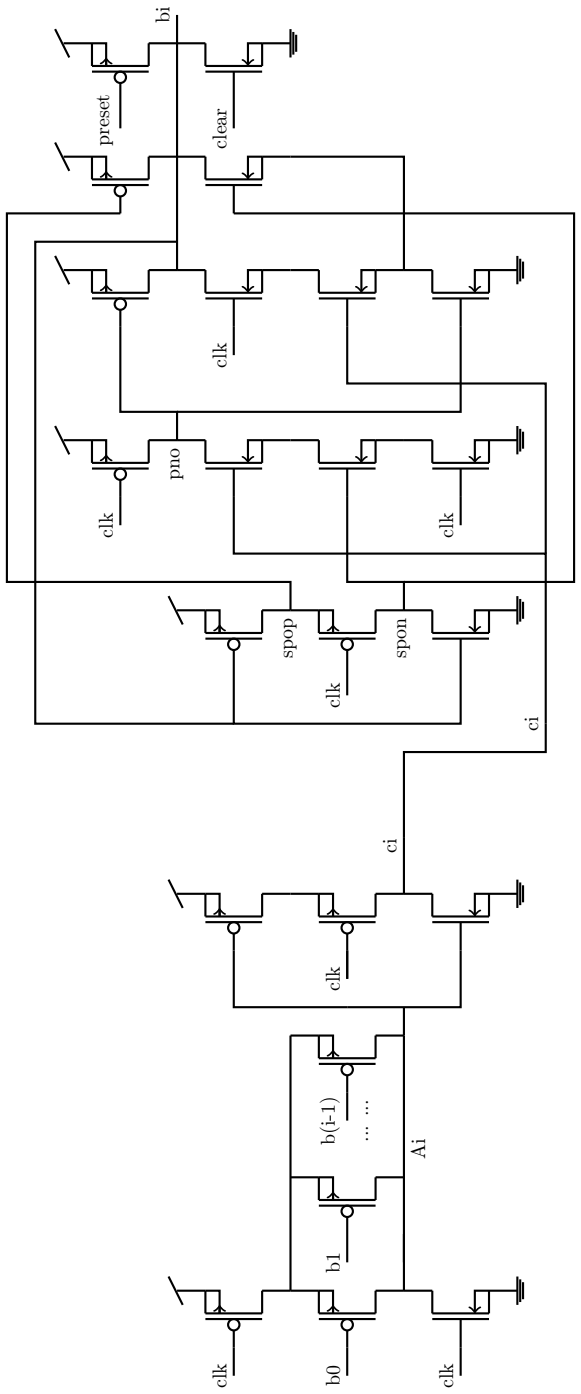


Figure 7.1: The TSPC toggle counter element

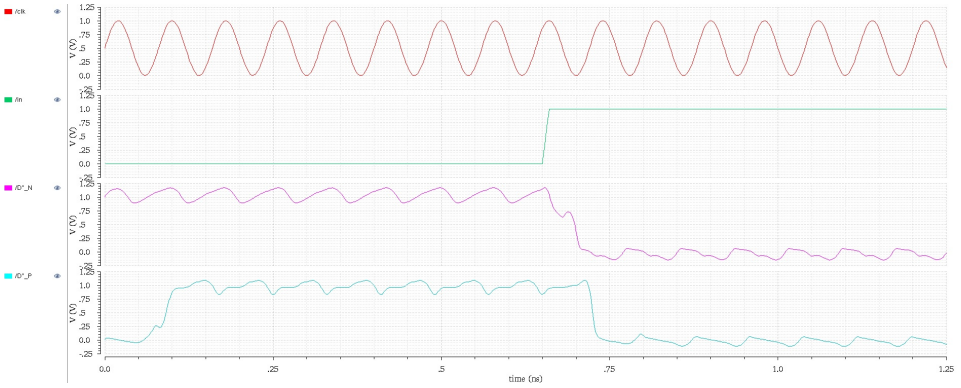


Figure 7.3: Simulation of N- and P- flip-flops with load rise at high clock

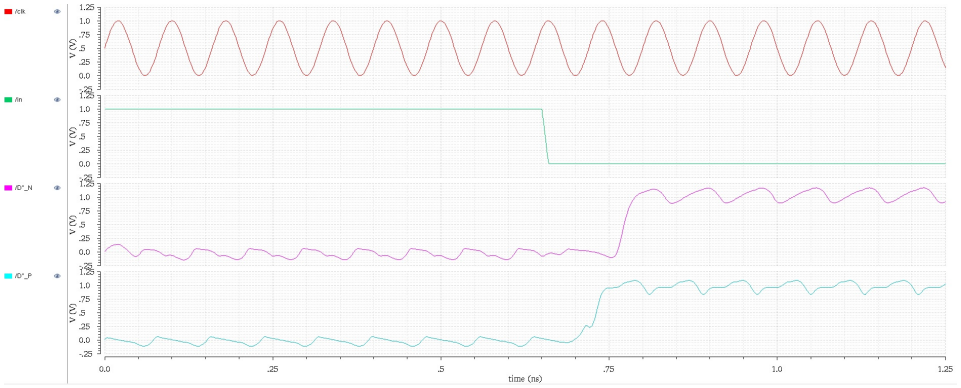


Figure 7.4: Simulation of N- and P- flip-flops with load fall at high clock

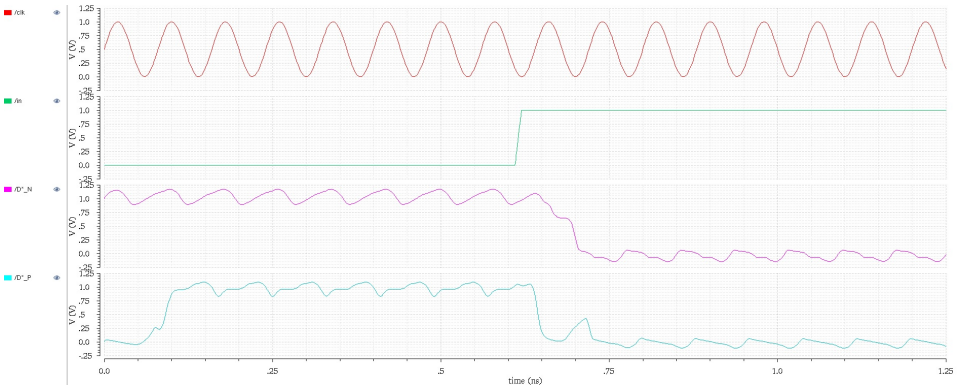


Figure 7.5: Simulation of N- and P- flip-flops with load rise at low clock

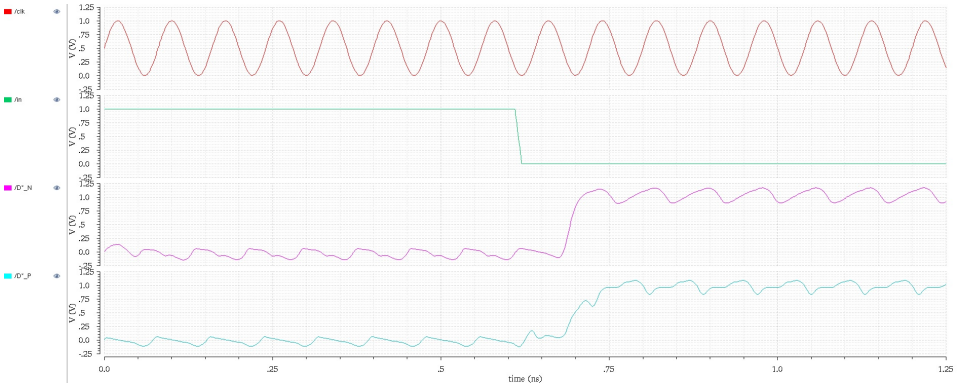


Figure 7.6: Simulation of N- and P- flip-flops with load fall at low clock

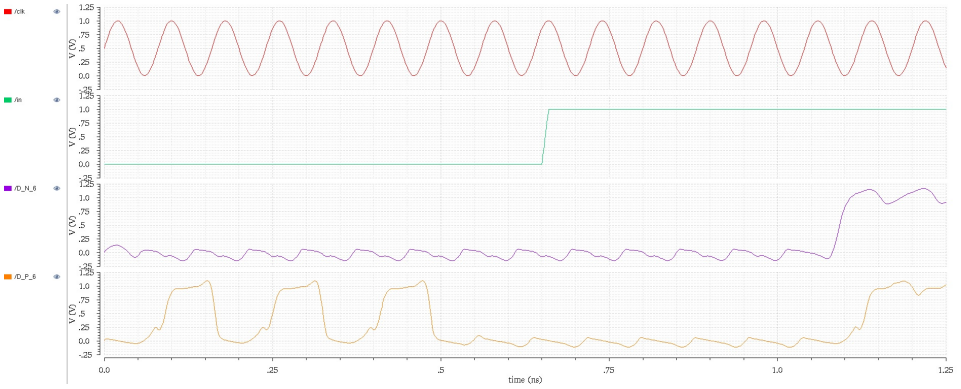


Figure 7.7: Simulation of 6 N- and P- flip-flops in series with load rise at high clock

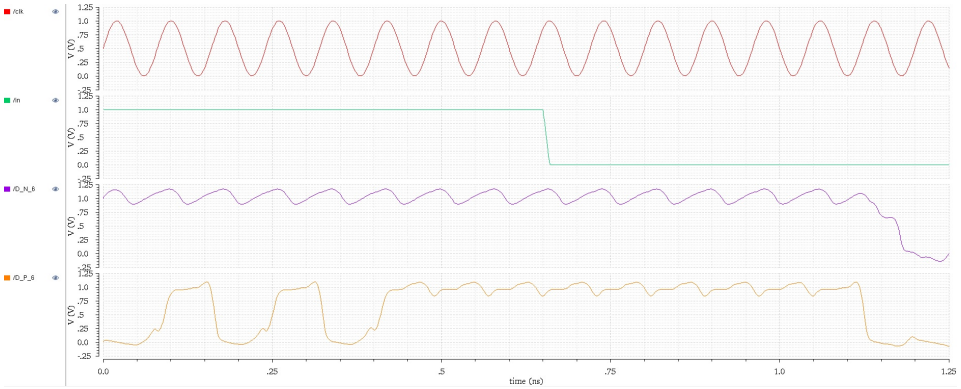


Figure 7.8: Simulation of 6 N- and P- flip-flops in series with load fall at high clock

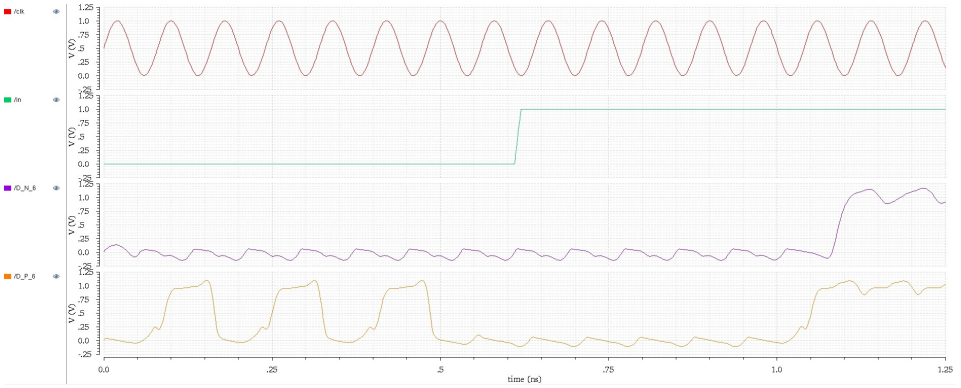


Figure 7.9: Simulation of 6 N- and P- flip-flops in series with load rise at low clock

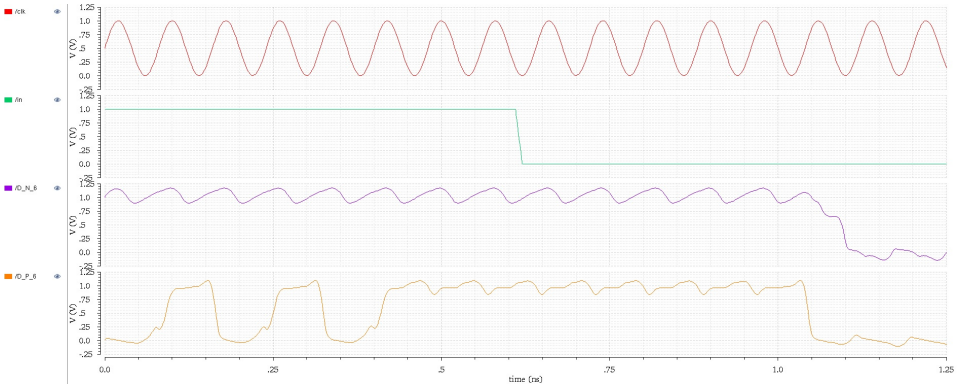


Figure 7.10: Simulation of 6 N- and P- flip-flops in series with load fall at low clock

7.4 The counter and it's internal signals

The counting is done with 127 pulses and at a temperature of 80 degrees with a voltage supply of 0.975V.

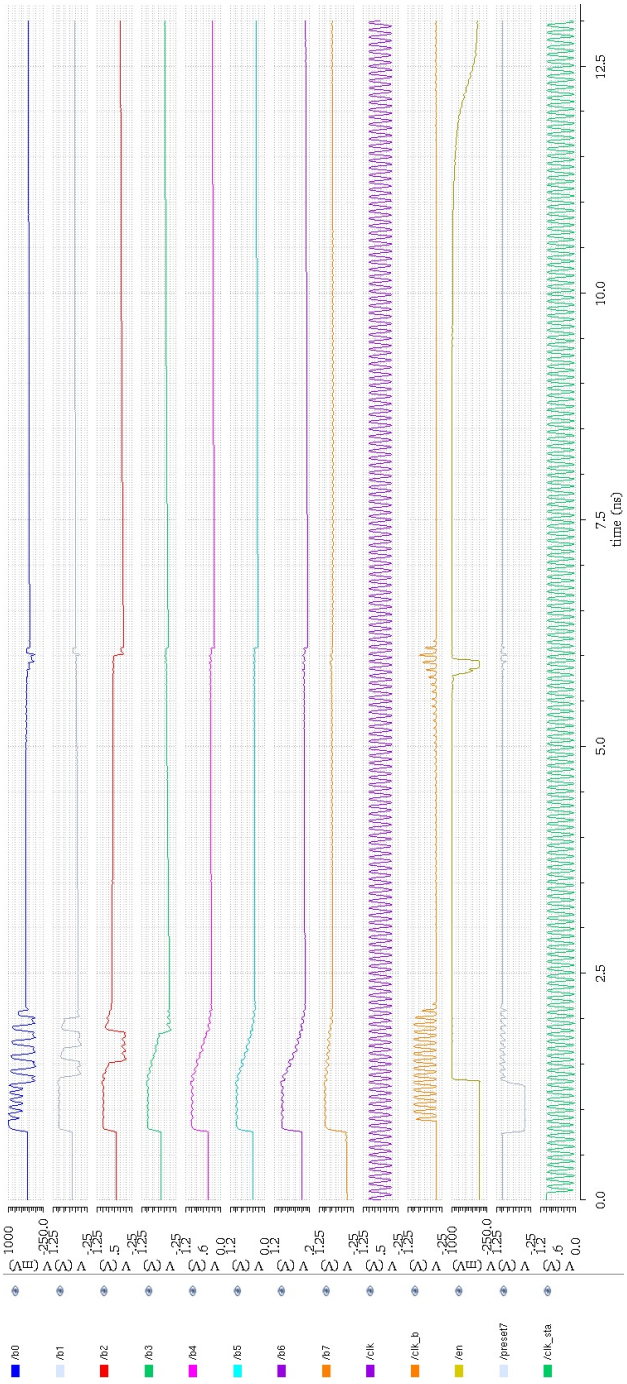


Figure 7.11: No dual-inverter at the toggles

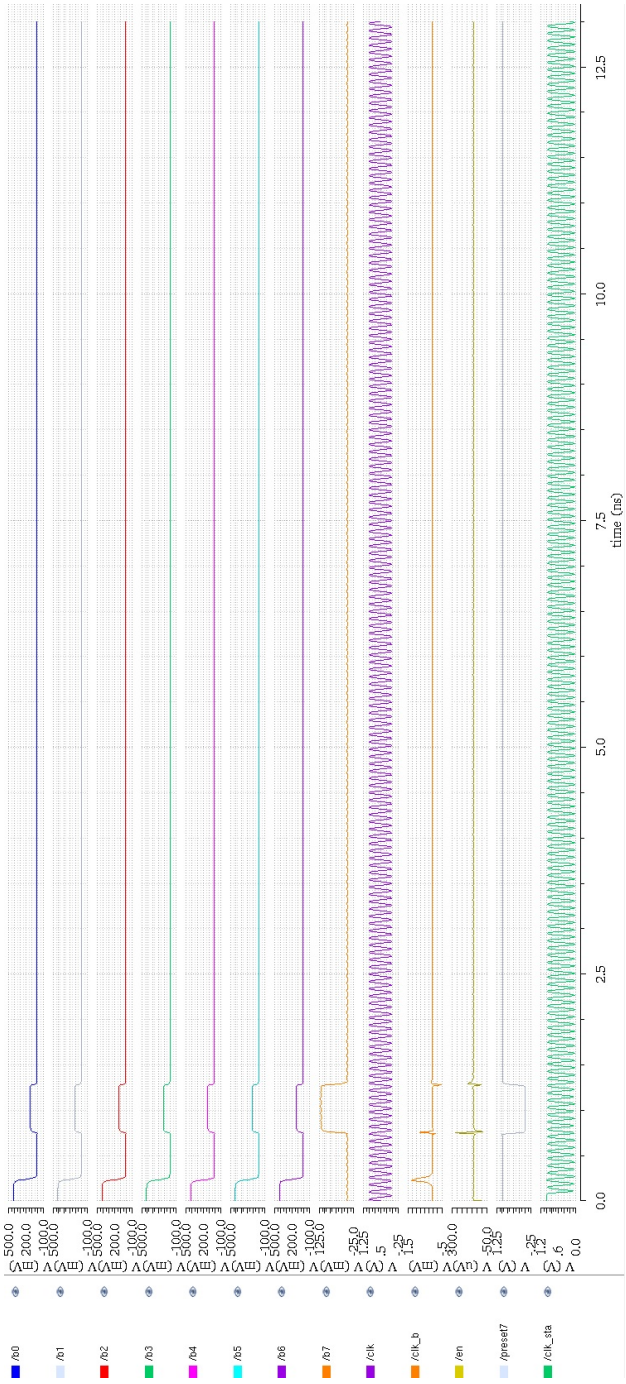


Figure 7.12: Minimum preset and clear transistors

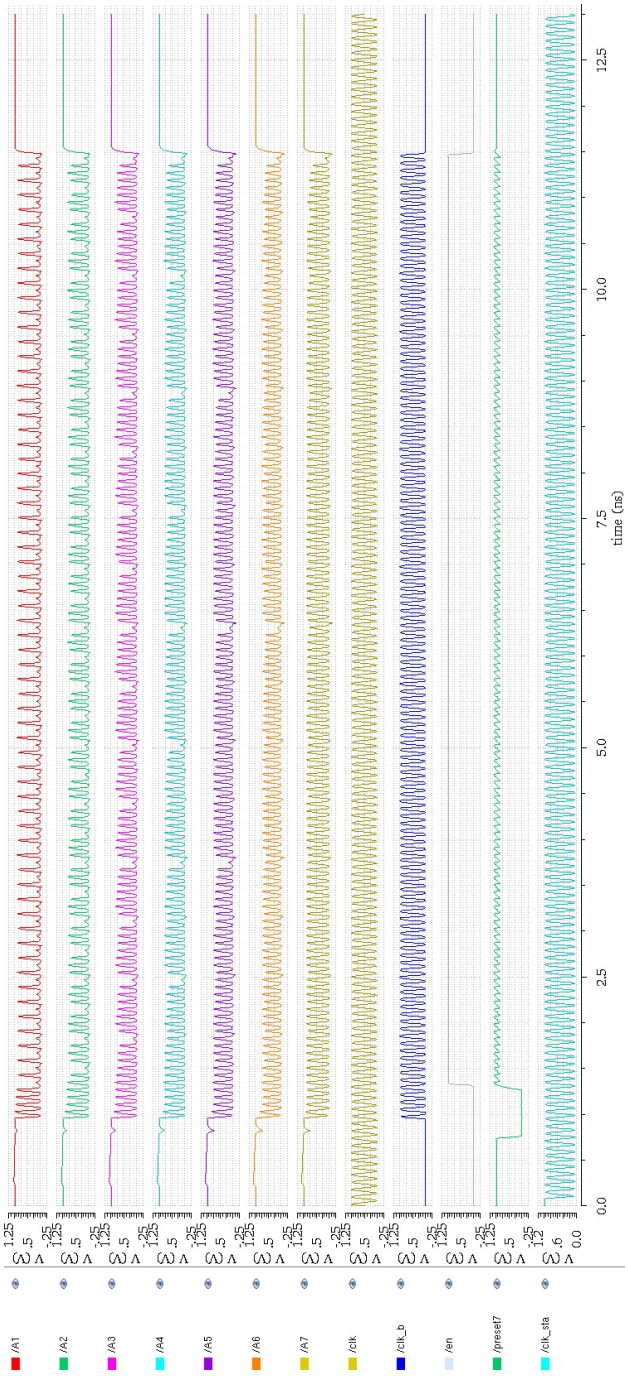


Figure 7.13: The outputs of the Nand PP's

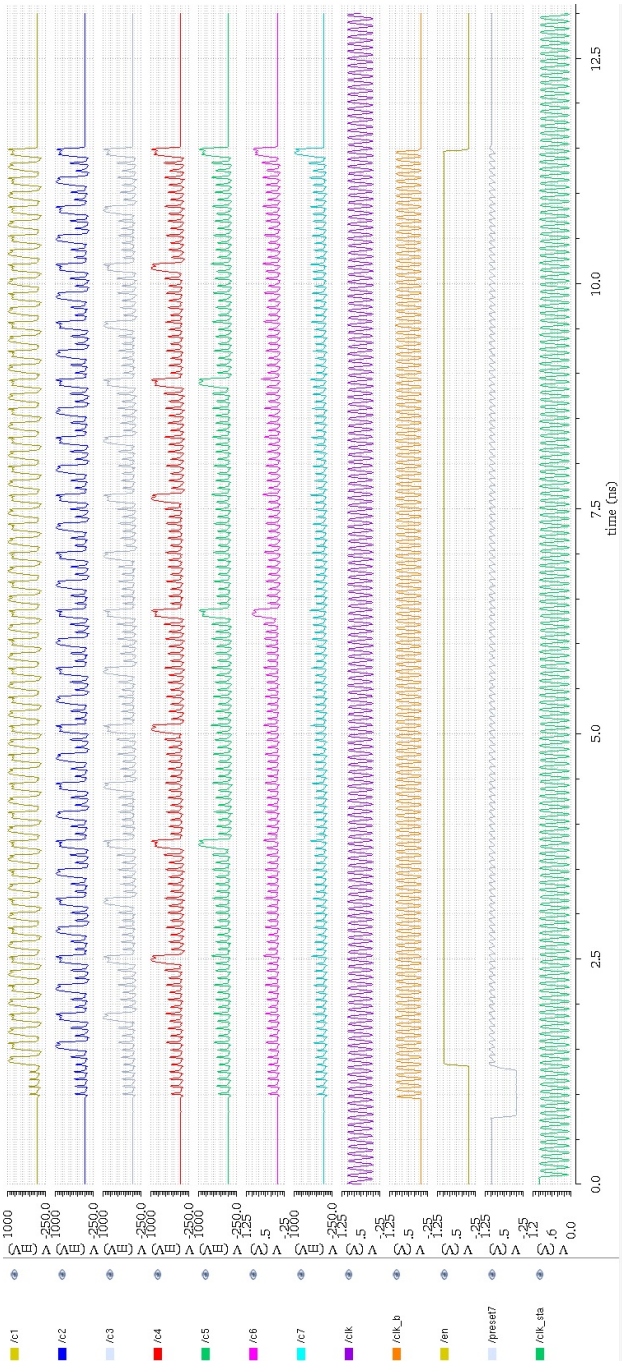


Figure 7.14: The outputs of the carry-look ahead part

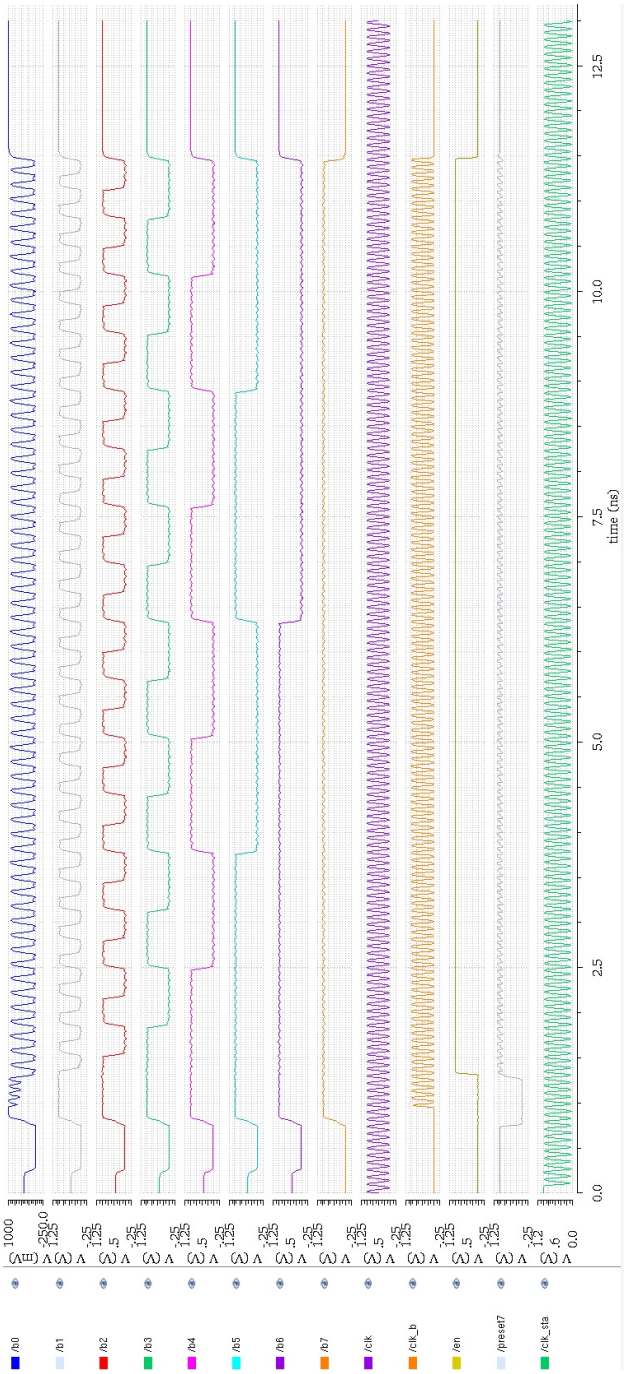


Figure 7.15: Counting with properly sized programming and dual-inverter transistors

7.5 Enable comparision

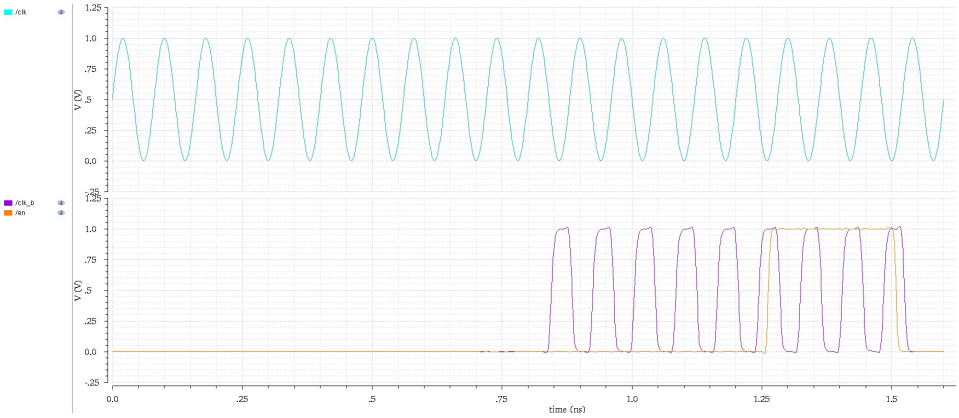


Figure 7.16: A close up picture of counting at -40, supply voltage of 1V and the FF corner (a good condition).

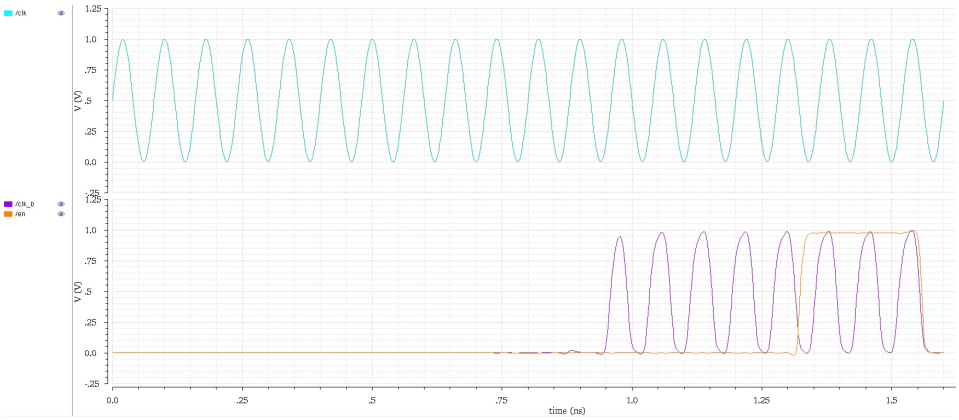


Figure 7.17: A close up picture of counting at 80 degrees with a supply voltage of 0.975V in the SS corner (a bad condition).

7.6 PVT results for 1,2,3 and 127 pulses for 12.5 and 9.5GHz

7.6.1 12.5GHz

T	V	SS	SF	FS	TT	FF
-40	0.975	895.4m	956.1m	913.7m	944.7m	949.9m
0	0.975	849.5m	943.8m	890.1m	940.1m	943.6m
80	0.975	720.8m	889.6m	751m	922.2m	932.9m
-40	1	928.8m	972.2m	934.5m	957.7m	961.7m
0	1	896.8m	965.9m	922m	956.6m	958.5m
80	1	802.7m	927.6m	868.2m	944.3m	950.9m
-40	1.025	952.8m	984.3m	951.2m	969.7m	972.5m
0	1.025	932.3m	981.7m	943.4m	968.9m	970.7m
80	1.025	859.9m	955.4m	912.7m	962.1m	964.7m
	max power	682.7u	721.7u	708.9u	757.3u	917.8u
	min power	572.1u	599.2u	582u	599.6u	650.1u

Table 7.1: Results for the counter with different voltage, temperature and process corner with 1 pulses (*pulsecount* = "1111110") at 12.5GHz with 2.5% variation in the voltage

T	V	SS	SF	FS	TT	FF
-40	0.975	1.897	1.957	1.913	1.944	1.949
0	0.975	1.852	1.945	1.891	1.939	1.943
80	0.975	1.723	1.892	1.752	1.922	1.931
-40	1	1.93	1.972	1.934	1.957	1.961
0	1	1.899	1.967	1.923	1.956	1.957
80	1	1.805	1.929	1.87	1.944	1.949
-40	1.025	1.954	1.984	1.95	1.968	1.971
0	1.025	1.933	1.982	1.943	1.968	1.969
80	1.025	1.862	1.956	1.914	1.961	1.963
	max power	697.7u	739.9u	728.8u	768.1u	938u
	min power	583.1u	615u	595.4u	613.9u	665.1u

Table 7.2: Results for the counter with different voltage, temperature and process corner with 2 pulses (*pulsecount* = "1111101") at 12.5GHz with 2.5% variation in the voltage

T	V	SS	SF	FS	TT	FF
-40	0.975	2.903	2.97	2.923	2.954	2.958
0	0.975	2.854	2.958	2.9	2.951	2.953
80	0.975	2.723	2.9	2.756	2.935	2.943
-40	1	2.938	2.987	2.945	2.969	2.971
0	1	2.904	2.981	2.931	2.967	2.968
80	1	2.807	2.938	2.876	2.958	2.96
-40	1.025	2.962	2.998	2.961	2.98	2.981
0	1.025	2.94	2.997	2.954	2.98	2.978
80	1.025	2.865	2.967	2.922	2.974	2.974
	max power	707.2u	749.9u	739u	784.9u	947.3u
	min power	591.5u	622.1u	604.2u	622.4u	674u

Table 7.3: Results for the counter with different voltage, temperature and process corner with 3 pulses (*pulsecount* = "1111100") at 12.5GHz with 2.5% variation in the voltage

T	V	SS	SF	FS	TT	FF
-40	0.975	126.9	127	126.9	127	127
0	0.975	126.9	127	126.9	126.9	126.9
80	0.975	126.7	126.9	126.9	126.9	126.9
-40	1	126.9	127	127	127	127
0	1	126.9	127	126.9	127	127
80	1	126.8	127	126.9	126.9	127
-40	1.025	127	127	127	127	127
0	1.025	126.9	127	127	127	127
80	1.025	126.9	127	127	127	127
	max power	2.286m	2.462m	2.448m	2.462	2.749m
	min power	1.905m	2.019m	1.981m	2.006	2.139m

Table 7.4: Results for the counter with different voltage, temperature and process corner with 127 pulses (*pulsecount* = "0000000") at 12.5GHz with 2.5% variation in the voltage

7.6.2 9.5GHz

T	V	SS	SF	FS	TT	FF
-40	0.975	950.5m	968.5m	940.3m	957.5m	
0	0.975	948.2m	966m	939m	953.7m	
80	0.975	943m	960.1m	933.1m	947.6m	
-40	1	962.5m	979.2m	953.3m	968.1m	
0	1	962.1m	978.3m	952m	966m	
80	1	960.1m	974.6m	951m	961.6m	
-40	1.025	972.6m	988m	963.7m	976.1m	
0	1.025	973.9m	987.2m	964.8m	976.4m	
80	1.025	972.5m	985.7	963.6m	972.7m	
	max power	585.9u	608.7u	608.5u	640.3u	785.1u
	min power	480.4u	494.6u	492.8u	495.9u	535u

Table 7.5: Results for the counter with different voltage, temperature and process corner with 1 pulses (*pulsecount* = "1111110") at 9.5GHz with 2.5% variation in the voltage

T	V	SS	SF	FS	TT	FF
-40	0.975	1.949	1.968	1.939	1.957	985m
0	0.975	1.948	1.965	1.937	1.953	970.1m
80	0.975	1.942	1.958	1.932	1.946	
-40	1	1.961	1.978	1.952	1.968	1.004
0	1	1.961	1.977	1.951	1.966	985.5m
80	1	1.959	1.972	1.949	1.96	
-40	1.025	1.971	1.987	1.963	1.975	1.025
0	1.025	1.972	1.986	1.963	1.976	998.5m
80	1.025	1.971	1.984	1.962	1.972	
	max power	602.7u	628.9u	629.8u	652.5u	795.2u
	min power	492.7u	511.8u	507.4u	511.4u	552.3u

Table 7.6: Results for the counter with different voltage, temperature and process corner with 2 pulses (*pulsecount* = "1111101") at 9.5GHz with 2.5% variation in the voltage

T	V	SS	SF	FS	TT	FF
-40	0.975	2.951	2.696	2.941	2.958	1.984
0	0.975	2.95	2.967	2.94	2.954	1.97
80	0.975	2.944	2.96	2.933	2.947	3.959
-40	1	2.963	2.98	2.954	2.969	2.003
0	1	2.963	2.978	2.954	2.967	1.985
80	1	2.961	2.974	2.951	2.961	1.975
-40	1.025	2.973	2.989	2.964	2.977	2.023
0	1.025	2.974	2.988	2.966	2.977	1.998
80	1.025	2.973	2.985	2.964	2.973	1.989
	max power	613.1u	637.9u	638.3u	668.5u	813.8u
	min power	500.9u	517.8u	516.3u	519u	559.1u

Table 7.7: Results for the counter with different voltage, temperature and process corner with 3 pulses (*pulsecount* = "1111100") at 9.5GHz with 2.5% variation in the voltage

T	V	SS	SF	FS	TT	FF
-40	0.975	127	127	125	127	126
0	0.975	127	127	126.9	127	126
80	0.975	126.9	127	126.9	126.9	127
-40	1	127	127	125	127	126
0	1	127	127	127	127	126
80	1	127	127	127	127	128
-40	1.025	127	127	125	127	126
0	1.025	127	127	127	127	126
80	1.025	127	127	127	127	128
	max power	1.644m	1.703m	1.757m	1.744m	1.973m
	min power	1.357m	1.299m	1.387m	1.399m	1.478m

Table 7.8: Results for the counter with different voltage, temperature and process corner with 127 pulses (*pulsecount* = "0000000") at 9.5GHz with 2.5% variation in the voltage

7.7 Clock problems for 9.5GHz

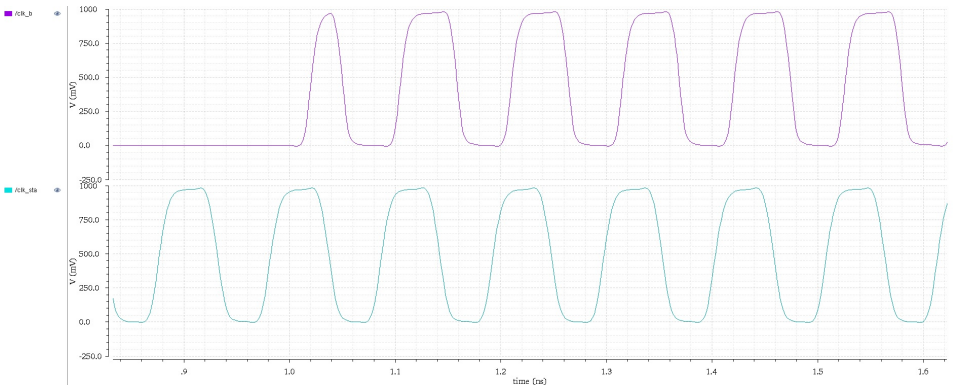


Figure 7.18: Simulation of showing the buffered clocks at 9.5GHz with FF corner, a temperature of 80 degrees and supply of 0.975V.

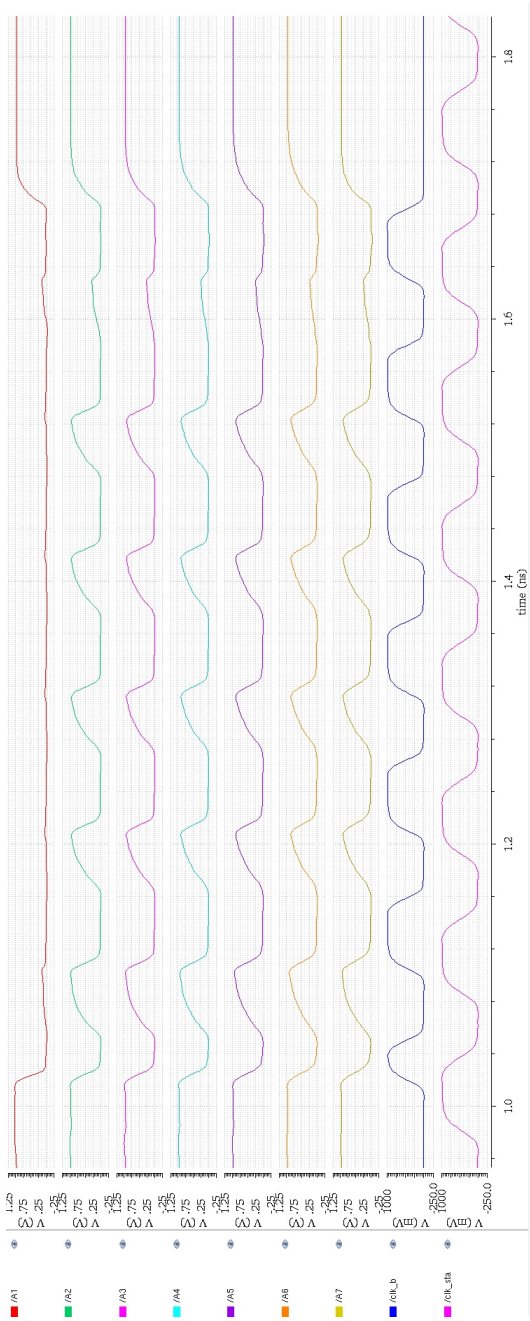


Figure 7.19: Simulation of showing the Ais at 9.5GHz with FF corner, a temperature of 80 degrees and supply of 0.975V and a *pulsecount* as 1111101.

Bibliography

[1] Tony Chan Carusone, David Johns, and Kenneth Martin. *Analog Integrated Circuit Design*. John Wiley & Sons, 2013.

[2] Christer Svensson Jiren Yuan. "High-Speed CMOS Circuit Technique". In: *IEEE Journal of Solid-State Circuits* (1989).

[3] Frank Leong et al. "A 6.5 GHz Arbitrary Digital Waveform Generator". In: *IEEE* (2012).

[4] Simon L'orange. *Micro Power and Multi GHz Pulse Generator*. Tech. rep. Norwegian University of Science and Technology, 2016.

[5] Milos D. Ercegovac Mircea R. Stan. "Long and Fast UpDown Counters". In: *IEEE* (1998).

[6] Prudhvi Raj Thota and Ashis Kumar Mal. "A High Speed Counter for Analog-to-Digital Converters". In: *IEEE* (2016).

[7] Magne Værnes, Trond Ytterdal, and Snorre Aunet. "Performance comparison of 5 Subthreshold CMOS flip-flops under process-, voltage-, and tmeperature variations, based on netlists from layout". In: *IEEE* (2014).

[8] Neil H. E. Weste and David Money Harris. *Integrated Circuit Design*. Pearson, 2011.

[9] Jiren Yuan and Christer Svensson. "New Single-Clock CMOS Latches and Flipflops with Improved Speed and Power Savings". In: *IEEE Journal of Solid-State Circuits* (1997).

[10] J.-R. Yuan. "Efficient Cmos Counter Circuits". In: *Eletronics Letters* (1988).