ECE 123 Final Lab Project Report

Alexander Wolff, Himangshu Chowdhury

Introduction

The goal of this project was to design an all digital delay locked loop (ADDLL) capable of operation for frequencies ranging from 100MHz to 1.5GHz. The fundamental function of a DLL is to adjust the phase of a digital clock relative to some reference clock very precisely through the use of feedback. To achieve this, it required a number of discrete loop elements including a phase detector, an integrator, and a programmable delay line. The following report is organized around these elements, with a section detailing the design of each, with additional sections for an architectural overview and the results of testing the design.

Architectural Overview

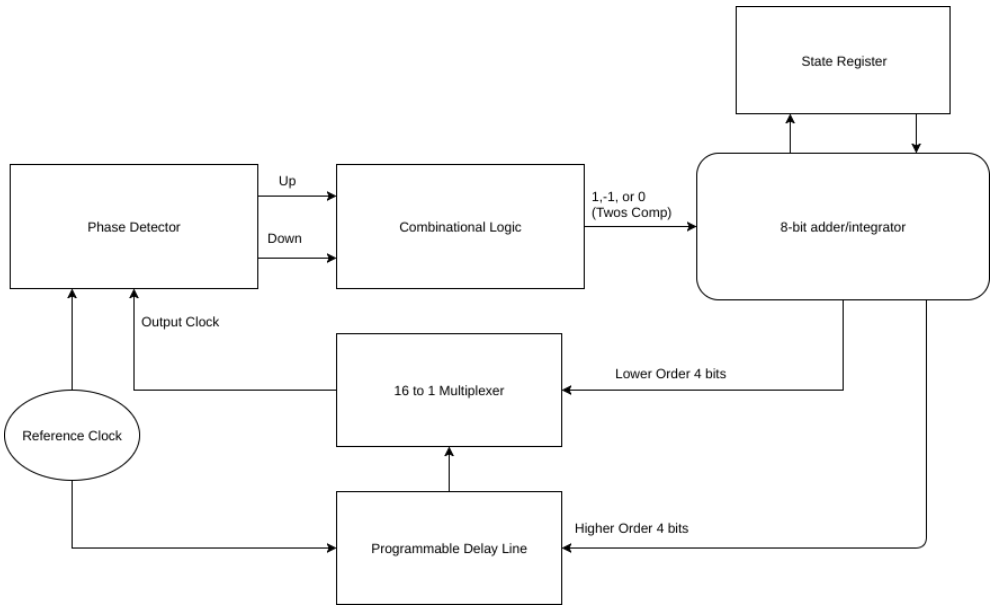
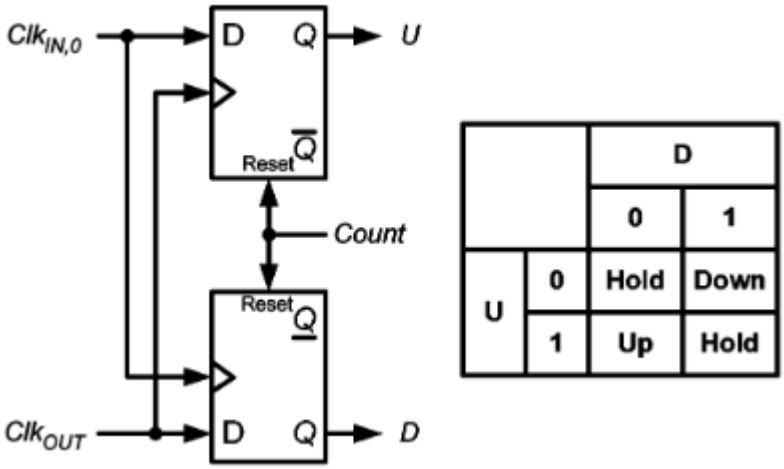


Fig 1. Architectural Overview

The design used for this project revolves around a 8 bit binary up/down counter implemented as a 8 bit adder in conjunction with a 8 bit state register implemented as a set of D flip flops. The phase detector generates Up and Down signals depending on whether the output clock is behind(10), ahead(01), or in phase(00) with the reference. This is then passed through some combinational logic to generate a 8-bit two's complement number to indicate to the counter whether more or less delay is required. This is then added to the current output of the adder which is stored in a register composed of D Flip Flops. The lower order 4 bits from this

result then forms the control word for the 16:1 MUX which determines where in the delay line the output is tapped; this is the fine delay control. The higher order 4 bits of the result are passed into the delay line where the capacitive loading of the inverter chain gets changed; this is the coarse delay control. This gets fed back to the phase detector and the loop repeats. This approach has the benefit of being entirely digital, rather than relying on an analog loop filter element such as a capacitor for integration. There is a critical path in the design running from ClkIn of the delay line to ClkOut. The delay in this path must be less than the minimum delay needed to achieve lock at 1.5 GHz. Special care was needed in managing the capacitance and delay involved with the MUX since the large number of inputs, and high loading has the potential to introduce excessive delay to the output clock. The MUX is implemented as a tree of 4:1 Multiplexers to minimize input loading. The 4:1 MUXs were also designed using low Vt transistors since the MUX is part of the critical path running between ClkIn and ClkOut, and any delay here will lower the maximum operating frequency of the design.

Phase Detector



		D	
		0	1
U	0	Hold	Down
	1	Up	Hold

Fig 2. Design of Phase Detector

The design of the phase detector is shown above. The design is relatively simple, generating an up signal when ClkIn is 1 and ClkOut is 0 and a down signal when the opposite is true. These up and down signals serve to increment or decrement the amount of delay needed respectively. This combination is then passed through combinational logic gates which convert it to its 8-bit two's complement counterpart +1 for up, -1 for down, 0 for hold. This value is then passed to the next stage which is the binary up/down counter.

Binary Counter

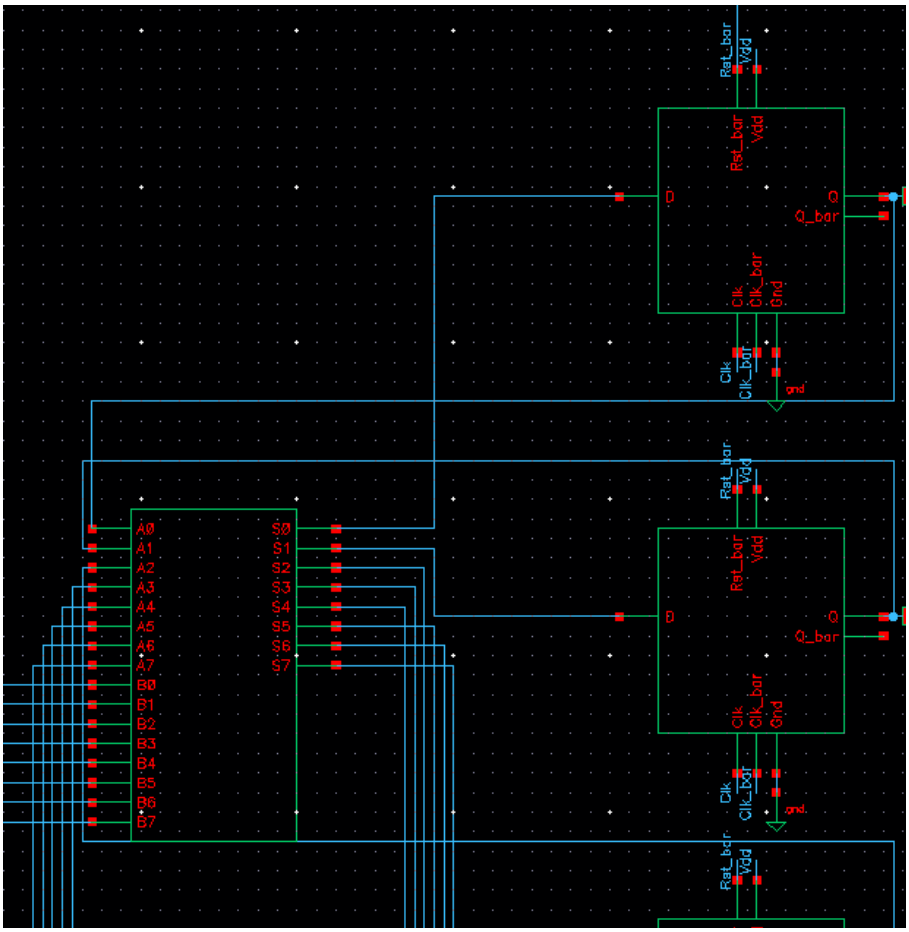


Fig 3. Binary Counter Schematic (Cadence)

Designing a suitable digital integrator for the DLL was particularly challenging. Most published papers when dealing with DLL design recommend or exhibit the use of a capacitor as the loop filter. This was not an option for our design since it was to be all digital. This led to the design of an 8-bit up/down counter as the integrator. The design is composed of an 8 bit adder, which was designed using the HDL full adder block available in ahdlLib library in Cadence, in conjunction with a synchronous 8-bit register composed of 8 D flip-flops. The result from each addition was stored in this register, and then fed back to the adder. The other input to the adder was the 8 bit 2's complement value generated by the phase detector. This allowed us to count up or down and to use this 8 bit value as the control word for our variable delay elements. The drawback to this approach is that it is only capable of counting linearly, and cannot jump between coarse adjustment and fine adjustment. It can only ever move the delay one step up or down, meaning that in some cases, lock acquisition could be relatively slow. However, given the time and design constraints, it was determined that this was the simplest effective solution. The 8 bits of the control word are used to adjust two characteristics of the delay line. The lowest order four bits are used to change where in the inverter chain clock out is tapped. The higher

order 4 bits are used to change the capacitive loading in the delay line. Changing the tapping point can be thought of as a “fine” adjustment, and changing the loading can be thought of as a “coarse” adjustment. This is discussed further in the Delay Line section of this report.

Delay Line

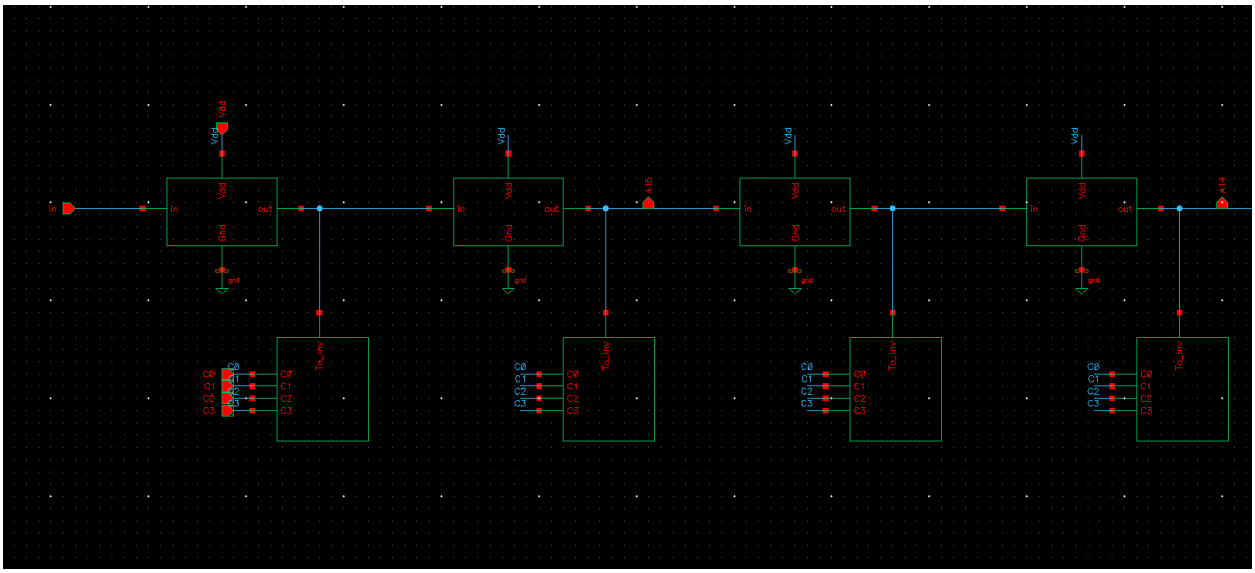


Figure 4. Delay Line Schematic (Cadence)

The delay line consists of a chain of 32 back to back inverters (groups of two between taps) with a variable capacitor bank after every inverter. The delay line “coarsely” adjusts the delay by changing the control word being fed into the capacitor bank. The capacitors are chosen in a way so that when the control word is 0000, there is no capacitive loading and causes the least delay and when the control word is 1111 there is maximum loading causing the maximum delay. Capacitors are switched on and off via nfet control transistors, as shown below. The ratio of the capacitors were decided by simulating at the lowest frequency (100 MHz) and at the highest frequency and testing different capacitances for which the output locks. This led us to get the ratios 1 : 6.2 : 38.5 : 238.3, so that we could have 16 possible combinations for coarse adjustment.

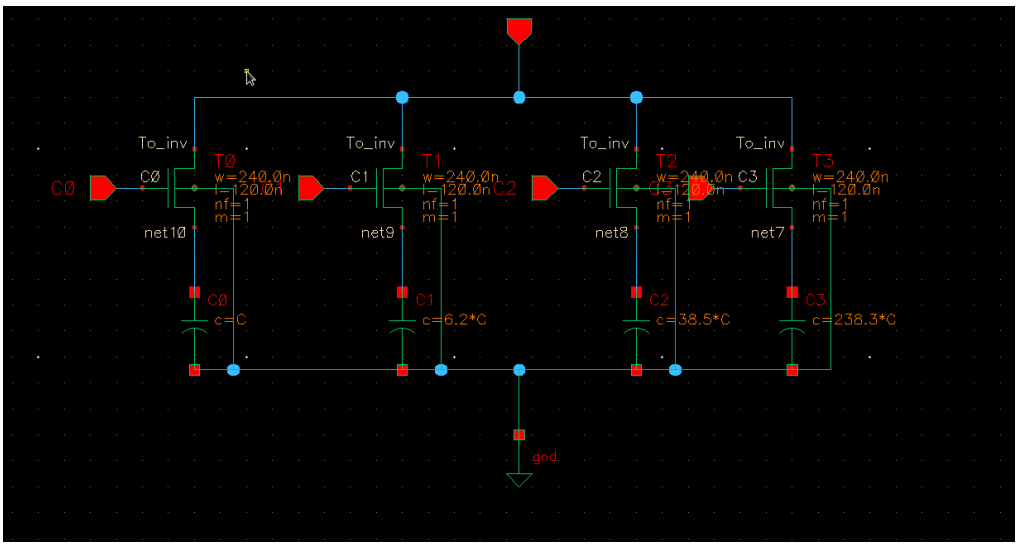


Figure 5. Capacitor Bank Schematic (Cadence)

The output of each pair of inverters is fed as an input to the 16:1 MUX, which gives multiple delays in the chain. Due to the nature of digital systems, there will always be some error/jitter in the phase of the output clock since the required exact delay, for a perfect lock, will almost never be one of the outputs generated in the delay line.

Testing & Simulation

In testing we were able to achieve phase lock at 1.5GHz, and several intermediate frequencies. The results of these simulations are shown below. We determined through a separate simulation that we have sufficient delay to lock at 100MHz. However, running the full DLL at this frequency would take prohibitively long to simulate, for reasons discussed in the following section.

All the following images show the startup state (after Reset on the flip flops are off) on the top, and the locked state on the bottom.

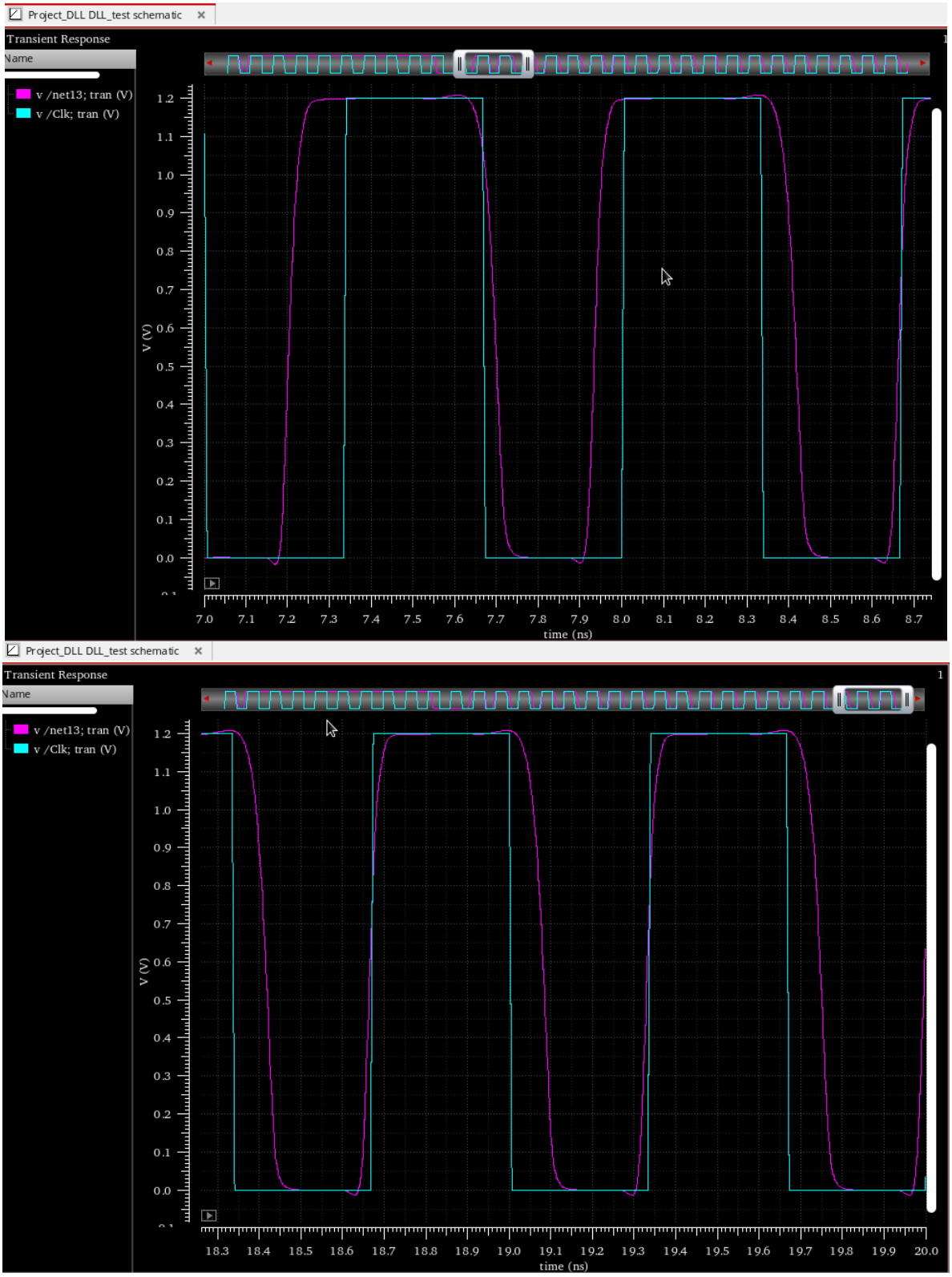


Fig 6. 1.5 GHz Operation

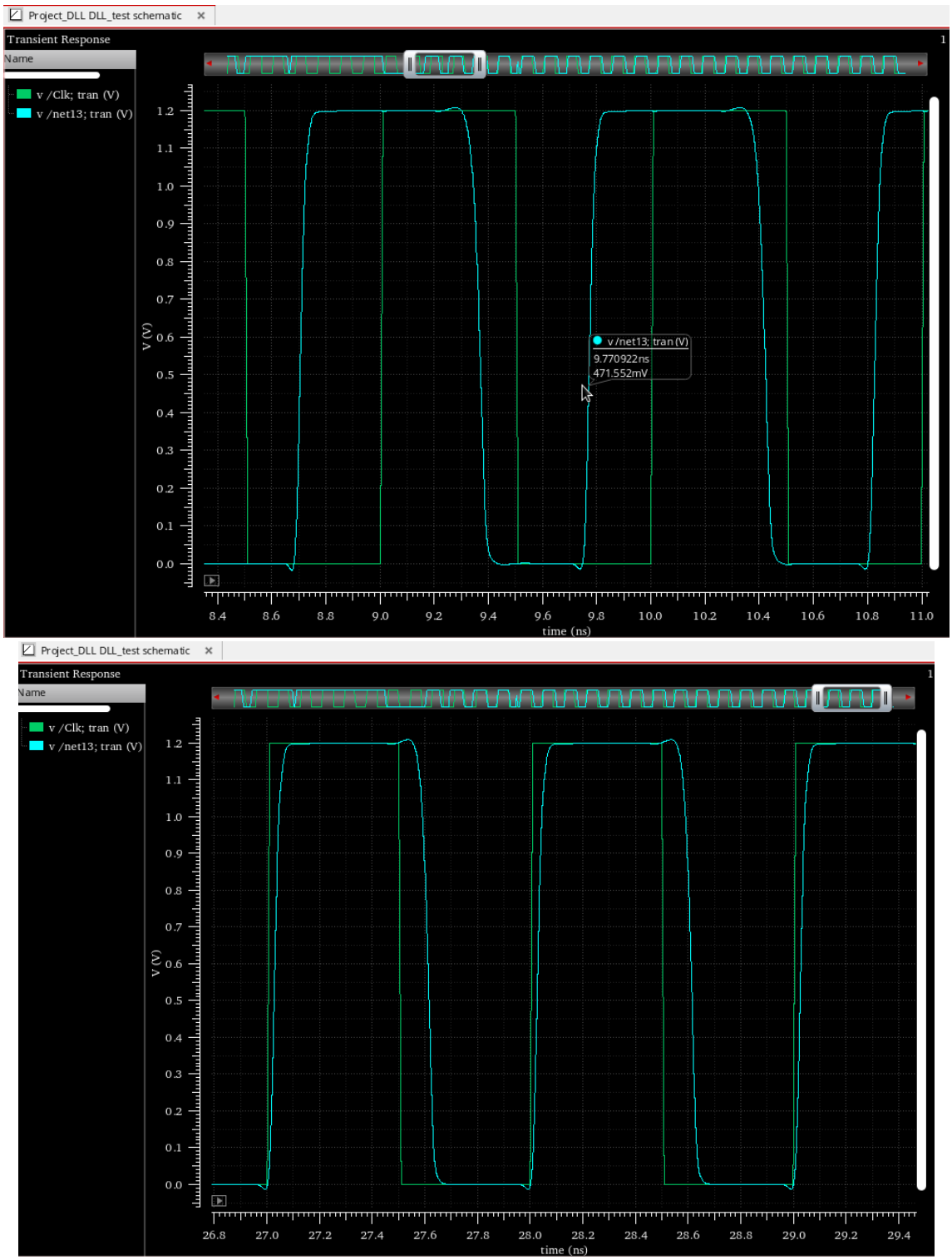


Fig 7. 1GHz Operation

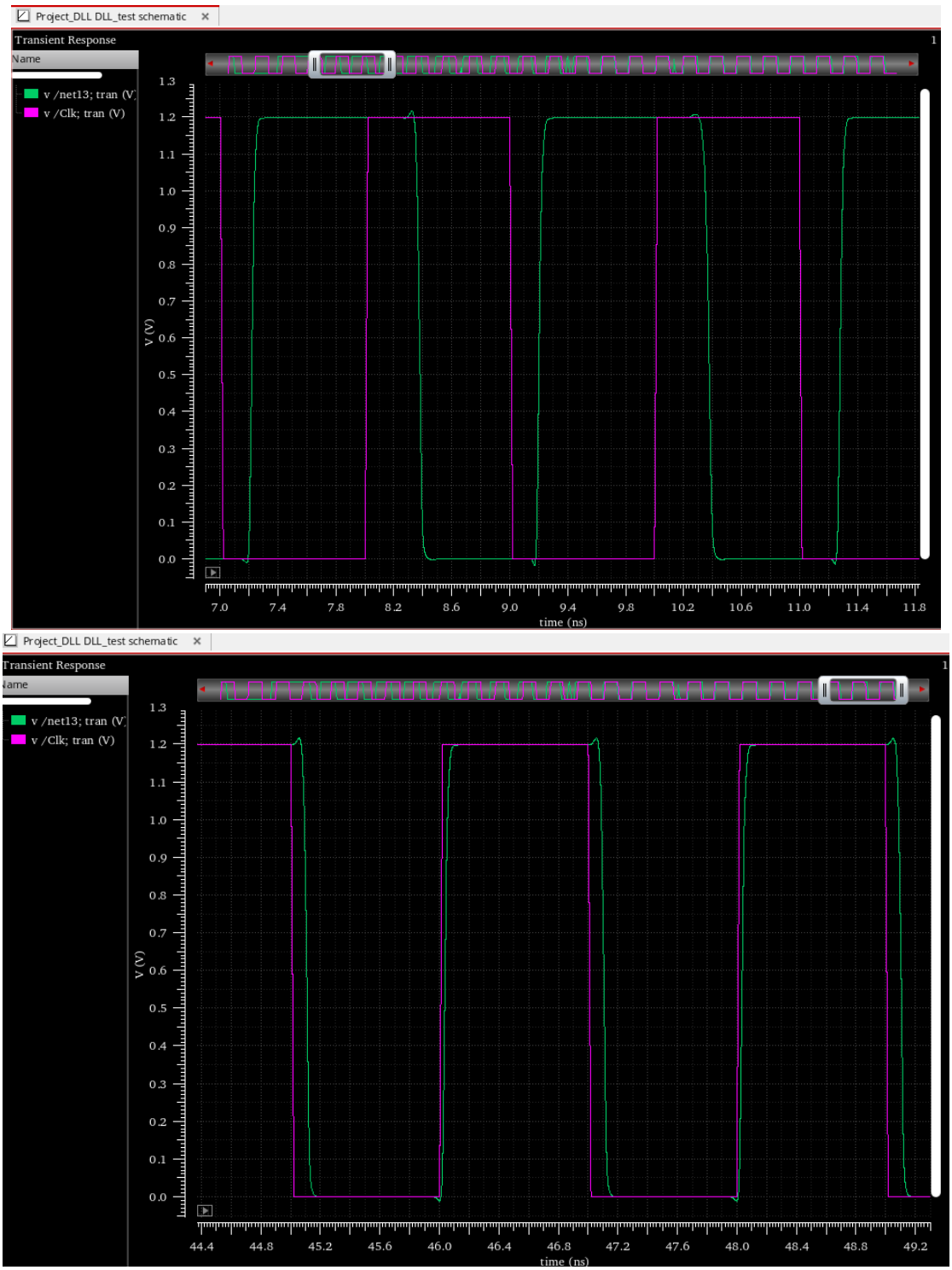


Fig 8. 500MHz operation

Fig 9. 100MHz operation (not completely locked)

At 100 MHz, the delay line finely adjusts to try and lock as shown in figure but after a few periods, it detects that the clock is not locked and coarsely adjusts by switching to a different capacitive loading in the delay line. It would lock ultimately after going through all combinations of capacitors but our simulation time exceeded the time we had on hand.

Results & Conclusion

We were able to meet the requirements of a digital DLL capable of operation from 100MHz to 1.5GHz. However, we did not have sufficient time to run simulations for process corners, which could affect our ability to meet specs. Additionally, there are several improvements to the design which could have been implemented with more time. The lock time for the lower frequencies was very long, sometimes on the order of microseconds. This is due to the fact that our counter starts at minimum delay and then sequentially counts up to the optimum. For 100 MHz this is very close to the maximum delay, which means it must count through approximately 256 combinations before the correct one is reached. This could be remedied by designing a system by which the “coarse” adjustment is counted through first and then the “fine” adjustment is added. This would make the maximum acquisition time 32 cycles. Unfortunately, the simulation time exceeded the time we had on hand before having a chance to implement such functionality.