ECE327: Digital Systems VLSI

Semester Project Part B

Design, Characterization and Use of Standard Cells

Ioanna-Maria Panagou

2962

Fall Semester 2022-2023

Contents

1 Summary

In this part of the project, we were tasked with implementing a characterization flow and using it to characterize the standard cells designed in the first part of the project. The final result is a library file which is going to be used in the last part of the project.

2 DFF & DFFRS

Before we proceed with characterizing the standard cells, we had to correct some design failures of the sequential ones.

2.1 DFF

The most important mistake we had to mend in our D Flip-flop cell layout was the use of the polysilicon as a wire for the CLK and $\overline{CLK}$ signals. The poly-to-field area capacitance per μm^2 and fringe capacitance per μm is significantly larger than its metal1-to-field counterpart, which could result in greater internal capacitances and delay our cell. Also, we forgetfully omitted the n- and p- wells from our previous implementation along with the subtrate contacts that are necessary to prevent the latch-up phenomenon, so we added both of them. In addition, we now removed the extra inverter that inverter the Q signal and just used the output of I_4 for $\overline{Q}$. The new design of our D flip-flop is pictured in (figure 1). We also tried to make our design as compact as possible to save area. Figure 2 compares our previous and our new cell layout.

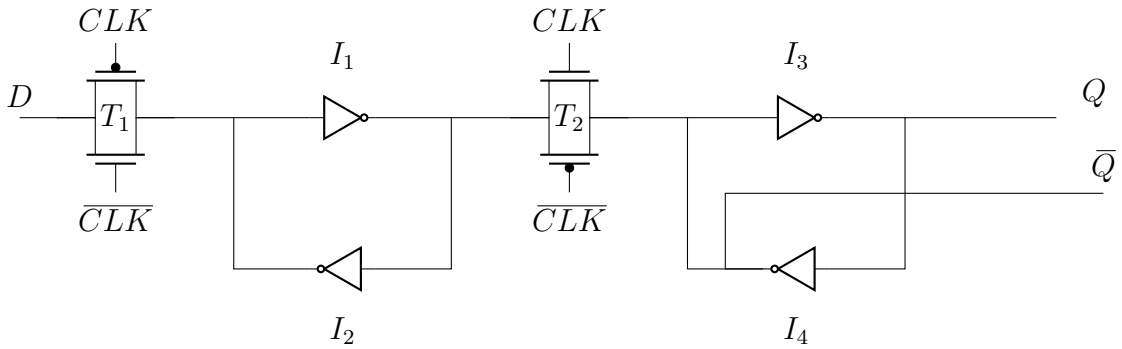

Figure 1: Reduced Load Clock Static Master-Slave Flip Flop

2.2 DFFRS

The D Flip-flop with asynchronous preset and clear also suffered from the same issues as the D Flip-flop, but the choice of architecture was also cumbersome and had redundant elements. An alternate, simpler implementation that was suggested during the examination of the first part was to replace the inverters with NOR gates. The new design is pictured in figure 3.

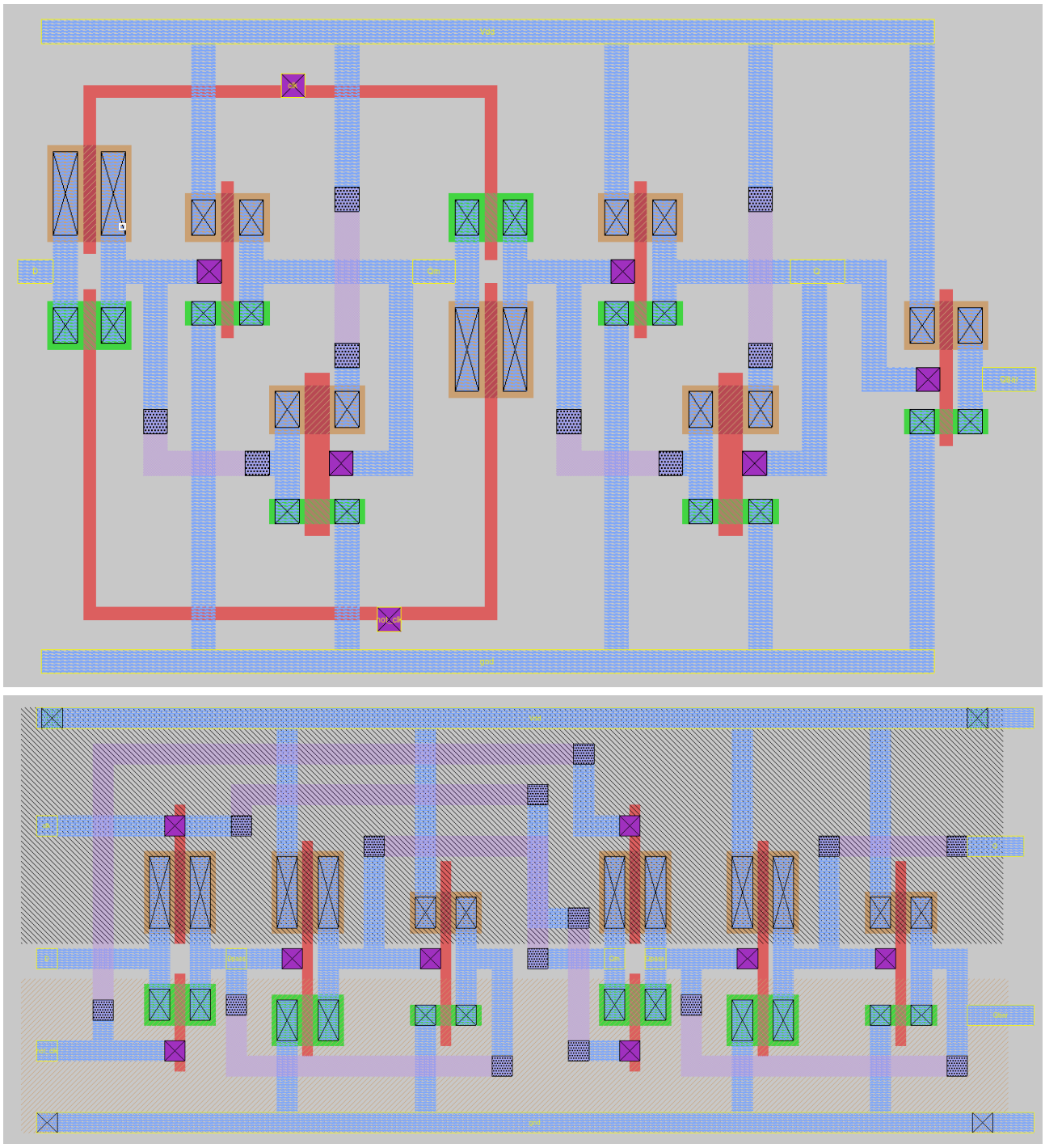

Figure 2: Old (top) implementation of the D Flip-flop and updated (bottom) implementation of the D Flip-Flop.

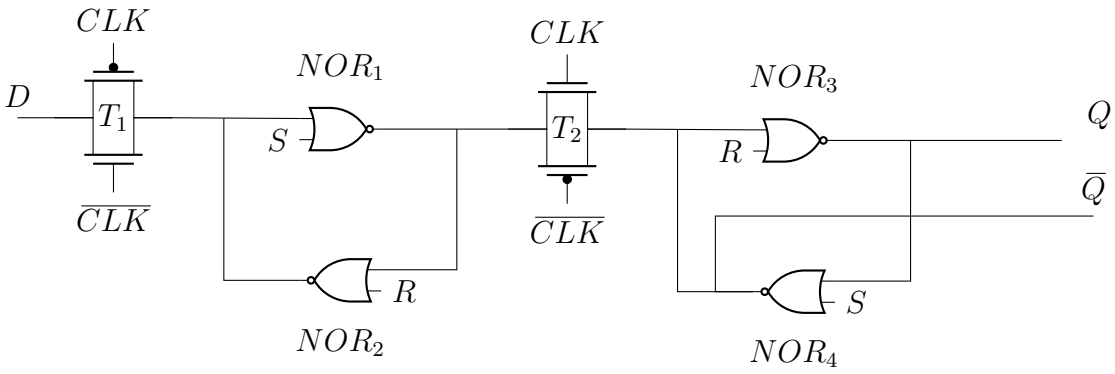

Figure 3: D Flip-flop with asynchronous preset and clear

The truth table of the NOR gate is presented in table 1. If at least one inputs is 1, the output of the NOR gate is 0 and the output is 1 only in the case where both inputs are equal to 0. With that in mind, we can analyze the operation of our DFFRS.

1. If both S and R are equal to 0, then:

   (a) If $D = 0$, the output of NOR_1 is 1 and on the rising edge of the clock, this value is passed to NOR_2 and the output Q becomes 0. Since $Q = 0$ and $S = 0$, the output of $NOR_4 = 1$ and, hence, $\overline{Q}$ has the correct value.

   (b) If $D = 1$, the output of NOR_1 is 0 and on the rising edge of the clock, this value is passed to NOR_2 and the output Q becomes 1. Because $Q = 1$ and $S = 0$, the output of NOR_4 and $\overline{Q}$ is 0.

2. If $S = 1$ and $R = 0$, then in the master latch, the output of NOR_1 gate will be 0, the input to NOR_2 will be 0 and, since $R = 0$, the output of NOR_2 will be equal to 1. The bistable element will retain this state even after S is unset and while D is equal to 1. If D becomes 0, then the state in the bistable element will change and the new value of D will appear at the output on the next rising edge of the clock. In the slave latch, the output of NOR_4 also becomes 0, so the output of NOR_3 is 1.

3. If $R = 1$ and $S = 0$, then in the master latch, the output of the NOR_2 gate becomes 0 and the output of NOR_1 becomes 1. In the slave latch, the output of NOR_3 becomes 0.

4. If both R and S are equal to 1, then the output of all gates is equal to 0, so, both Q and $\overline{Q}$ are equal to 0, which leads the D flip flop to an unstable situation. For example, if R is deasserted, then if Q first becomes 1, then the output of NOR_4 will become 0 or if $\overline{Q}$ becomes 1 first, then Q will become 0, which is not deterministic. Also, those changes will take effect immediately after S and R become low, which goes against the principle functionality of the D flip flop, whereas changes become on the active edge of the clock or the active edge of the asynchronous inputs.

A	B	F
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

Table 1: NOR gate truth table

We must ensure that NOR_2 and NOR_4 are weaker than NOR_1 and NOR_3, so that when the input to NOR_1 and NOR_3 changes, NOR_2 and NOR_4 do not fight the change, similar to what we've done with the inverters of the D flip flop above. We verified the functionality of the new D flip flop with asynchronous preset and clear using Ngspice. The old and new DFFRS layouts are compared in figure 4.

3 Configuration File

3.1 Configuration File Format

We decided to implement the configuration file in a JSON format, since it closely resembles the format of the liberty file. Our JSON file is basically a dictionary of dictionaries as shown in 5.

```
{
        cell: {
                path: <path to extracted .spice file>
                type: "combinational" or "sequential"
                pins: {
                        pin name: {
                                direction: "input" or "output"
                                function:
                                timings: {
                                        timing: {
                                                related_pin:
                                                when:
                                                timing_sense:
                                                timing_type:
                                                measurements: <list of timing or constraint arcs>
                                        }

                                }
                        }

                }
        }
}
```

Figure 5: Structure of our configuration file

We load the JSON configuration file to our python script, which results in a top-level dictionary **cells**, structured as follows:

1. It is comprised of multiple dictionaries, one for each **cell**, which have names that correspond to the actual cell they are referring to.

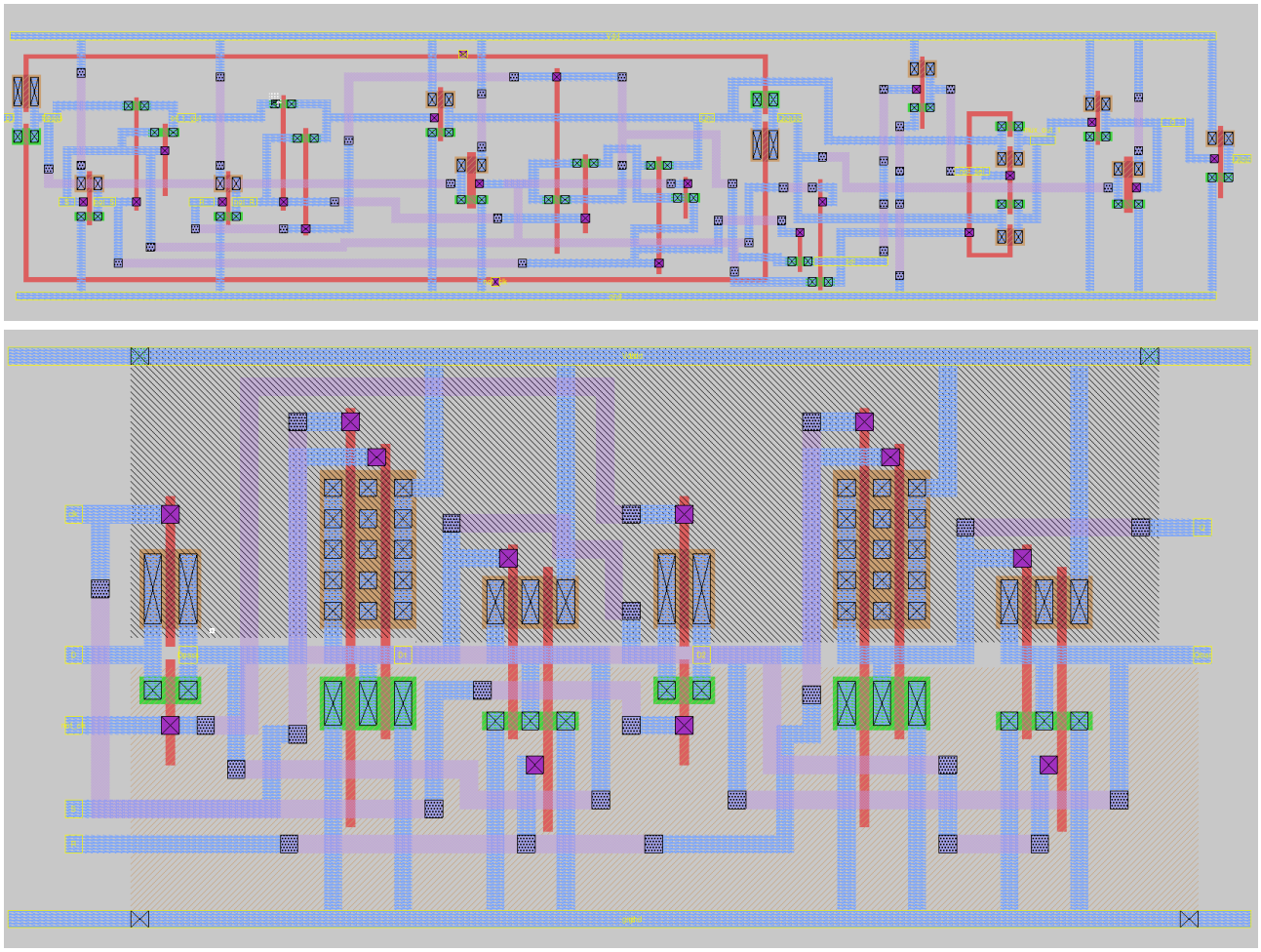

Figure 4: Old (top) implementation of the DFFRS and updated (bottom) implementation of the DFFRS.

2. Each cell contains the fields **path** that is the relative path to the .spice file extracted from MAGIC and its type, which can either be combinational or sequential. This information is used by our script. It also contains multiple dictionaries, one for each **pin**, noted as **pin name** in the figure.

3. Each pin dictionary contains the fields **direction**, which can be either input or output and the field function, in case the direction of the pin is output. It also contains a dictionary that stores all the different **timing** clauses. Each entry of the **timings** dictionary is a different timing, which are named 1, 2, 3, ... for simplicity.

4. Each timing dictionary is comprised of the fields related_pin, when, timing_sense, timing_type and measurements.

   (a) The related_pin defines the input pin to which this timing arc is referring to

   (b) The when fields defines the values of the side inputs

   (c) The timing_sense only applies when the direction of the pin is output and can be either positive_unate, negative_unate and non_unate.

   (d) The timing_type only applies to sequential cells. The different timing types for our D flip flops can be:

      - setup_rising
      - hold_rising
      - removal_rising
      - recovery_risin
      - rising_edge
      - falling_edge

   (e) The field measurements is actually a list of all the different timing/constraint arcs we would like to measure and can have the values:

      - cell_rise
      - cell_fall
      - rise_transition
      - fall_transition
      - rise_constraint
      - fall_constraint

3.2 XOR/XNOR Timing Arcs

3.2.1 XOR

This combinational gate only has timing arcs corresponding to its output pin. More specifically, we would like to measure the high-to-low and low-to-high propagation delay and the rise and

A	B	F
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Table 2: XOR gate truth table

A	B	F
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

Table 3: XNOR gate truth table

fall delays. The truth table of the XOR gate is given in table 2.

We will first try to find the unateness of the timing arcs with respect to the input A.

- When $B = 0$ and A goes from $0 \rightarrow 1$, the output also goes from $0 \rightarrow 1$ and when A goes from $1 \rightarrow 0$, the output also goes from $1 \rightarrow 0$. When $B = 1$ and A goes from $0 \rightarrow 1$, the output goes from $1 \rightarrow 0$ and when A goes from $1 \rightarrow 0$, the output goes from $0 \rightarrow 1$. We can conclude that this timing arc is actually binate, since it is different for each value of the side input B. For the liberty file, this information is coded into two different timing arcs, one when $B = 0$ and the unateness is positive and one when $B = 1$ and the unateness is negative. We are interested in measuring the cell_fall, cell_rise, rise_transition and fall_transition.

- A similar analysis for B leads in the same results, so we will also have two different timing arcs for B depending on the value of A. Those two are distinguished by the value of the when clause, which will be the value of the side input. We are interested in measuring the cell_fall, cell_rise, rise_transition and fall_transition.

3.2.2 XNOR

Again, this combinational gate only has timing arcs corresponding to its output pin. The truth table of the XNOR gate is given in table 3.

We will first try to find the unateness of the timing arcs with respect to the input A.

- When $B = 0$ and A goes from $0 \rightarrow 1$, the output goes from $1 \rightarrow 0$ and when A goes from $1 \rightarrow 0$, the output goes from $0 \rightarrow 1$. When $B = 1$ and A goes from $0 \rightarrow 1$, the output also goes from $0 \rightarrow 1$ and when A goes from $1 \rightarrow 0$, the output also goes from $1 \rightarrow 0$. We can conclude that this timing arc is actually binate, since it is different for each value of the side input B. For the liberty file, this information is coded into two different timing arcs,

one when $B = 0$ and the unateness is negative and one when $B = 1$ and the unateness is positive. We are interested in measuring the cell_fall, cell_rise, rise_transition and fall_transition.

- A similar analysis for B leads in the same results, so we will also have two different timing arcs for B depending on the value of A. Those two are distinguished by the value of the when clause, which will be the value of the side input. We are interested in measuring the cell_fall, cell_rise, rise_transition and fall_transition.

3.3 D Flip Flop Timing and Constraint Arcs

3.3.1 Timing Arcs

The output pins Q and $\overline{Q}$ of the D flip flop have timing arcs. More specifically, we are interested in measuring the the $CLK \rightarrow Q$ and $CLK \rightarrow \overline{Q}$ delays and the rise and fall delays of Q and $\overline{Q}$. The CLK has a non unate/binate relationship with the two outputs. Also, since the D flip flop is positive-edge triggered, the timing type is rising_edge.

3.3.2 Constraint Arcs

The input D has constraint arcs in relation to the CLK; setup_rising and hold_rising. For each constraint arc, we have to measure both the rise and the fall constraint.

3.4 DFFRS Timing and Constraint Arcs

3.4.1 Timing Arcs

The DFFRS has the same timing arcs as the DFF with relation to CLK. It also has some additional ones with relation to the 2 asynchronous inputs S and R. The unateness for the timing arcs between Q and R is negative_unate, between $\overline{Q}$ and R is positive_unate, between Q and S is positive_unate and between $\overline{Q}$ and S is negative_unate. For each related pin, we will measure the timing arcs for all combinations of the side inputs.

- When the pin is Q and the related pin is R, for every combination of CLK, D and S, we will measure cell_fall and fall_transition.
- When the pin is Q and the related pin is S, for every combination of CLK, D and R, we will measure cell_rise and rise_transition.
- When the pin is $\overline{Q}$ and the related pin is R, for every combination of CLK, D and S, we will measure cell_rise and rise_transition.
- When the pin is $\overline{Q}$ and the related pin is S, for every combination of CLK, D and R, we will measure cell_fall and fall_transition.

Keep in mind, that, because S and R are not allowed to be high at the same time, some timing arcs will be impossible; for example, if R is 1 and S then becomes 1, the output might not become high.

3.4.2 Constraint Arcs

Input D of the DFFRS has the same constraint arcs with relation to CLK; setup_rising and hold_rising. The asynchronous input S and R also have constraint arcs with relation to CLK. Those are the recovery and removal times, which are analogous to the setup and hold times. Recovery time is the minimum time required between the deassertion of the reset/set signal and arrival of clock edge and removal time is the minimum time required between the arrival of the clock's active edge and the deassertion of the reset/set signal. Since both S and R are active-high and the FF is positive-edge triggered, the timing types are recovery_rising and removal_rising and we the constraint is fall_constraint for both asynchronous inputs.

3.5 Methodology

Before we proceed with explaining our tool it is necessary to define the different measurements and the approach we took to measure them.

3.5.1 Cell Rise

The cell_rise is the low-to-high propagation delay. We measure the time difference between the related pin reaching the 50% of its final value on its rising or falling slope, as defined by the unateness of the timing arc and the pin reaching the 50% of its final value on its rising slope.

3.5.2 Cell Fall

In a similar fashion, the cell_fall is the high-to-low propagation delay. We measure the time difference between the related pin reaching the 50% of its final value on its rising or falling slope, as defined by the unateness of the timing arc and the pin reaching the 50% of its final value on its falling slope.

3.5.3 Rise Transition

For the rise transition, we measure the time between the pin reaching the 30% and 70% of its final value on a rising slope.

3.5.4 Fall Transition

For the fall transition, we measure the time between the pin reaching the 30% and 70% of its final value on a falling slope.

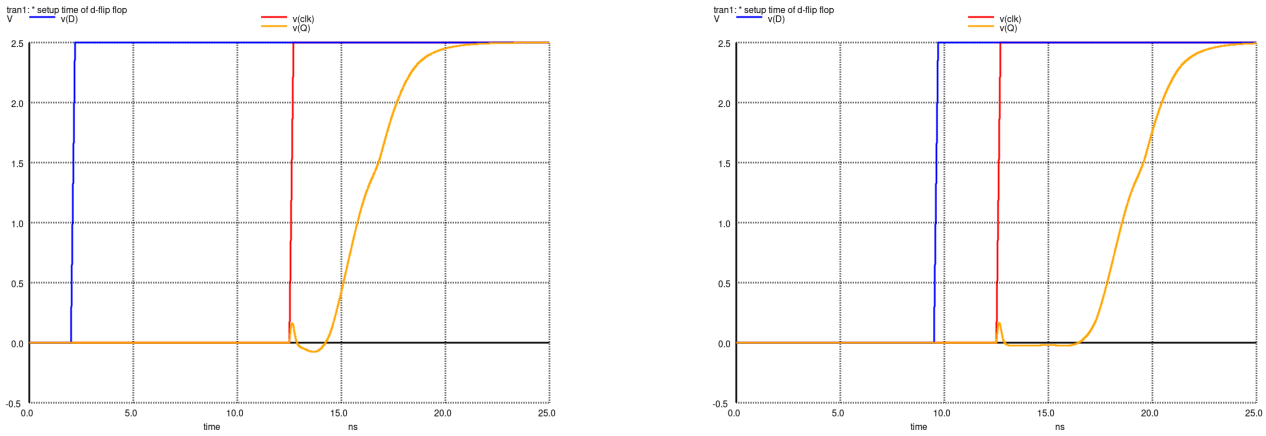

Figure 6: We can see that as we approach the last change of input D to the rising clock edge the propagation delay $CLK \rightarrow Q$ grows larger.

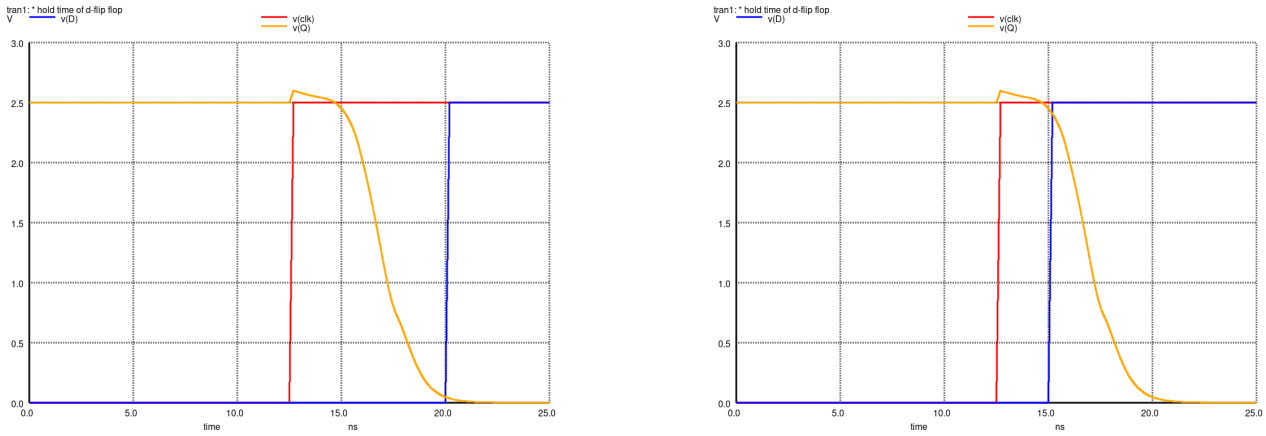

Figure 7: We can see that as we approach the first change of input D to the rising clock edge the propagation delay $CLK \rightarrow Q$ grows larger.

3.6 Rise Constraint

The rise constraint is actually how close relatively to the active clock edge the pin can rise without violating the setup, hold, removal and recovery constraints.

The setup and hold times apply to the D input of the DFF and DFFRS. We define the setup time as the time difference between the last change of input D before the active edge of the clock and the arrival of this edge, so that the $CLK \rightarrow Q$ delay is 1.05 greater than the $CLK \rightarrow Q$, if there was no setup violation. (figure 6)

Similarly, we define the hold time as the time difference between the arrival of the active clock edge and the first change of input D after this clock edge, so that the $CLK \rightarrow Q$ is 1.05 times larger than the $CLK \rightarrow Q$ without hold violation. (figure 7).

3.7 Fall Constraint

The fall constraint for the setup and hold constraint arcs are analogous to the rise constraint.

As mentioned previously, the recovery is the minimum time between the deactivation of

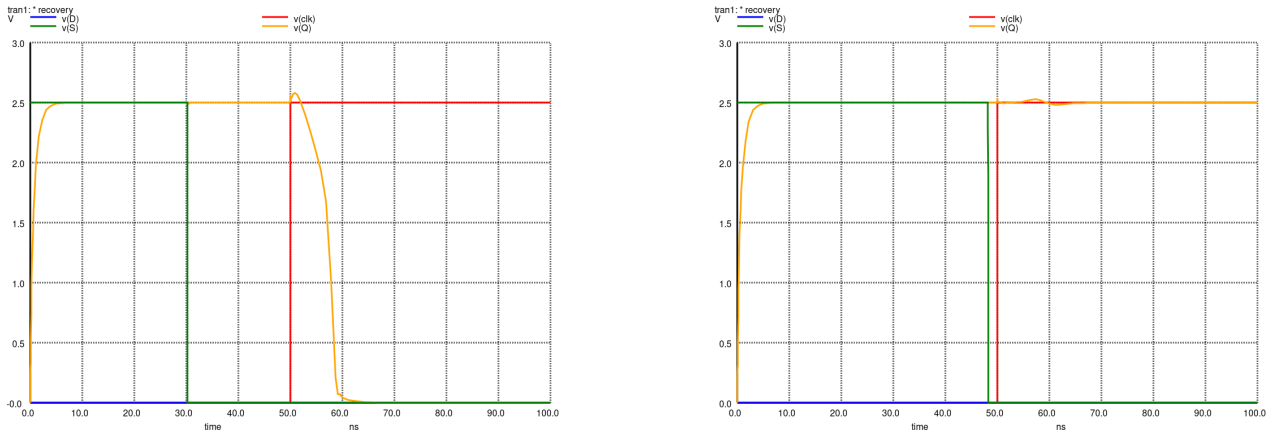

Figure 8: Deactivating S very closely to the active edge of the clock can result in wrong values or metastability in the output

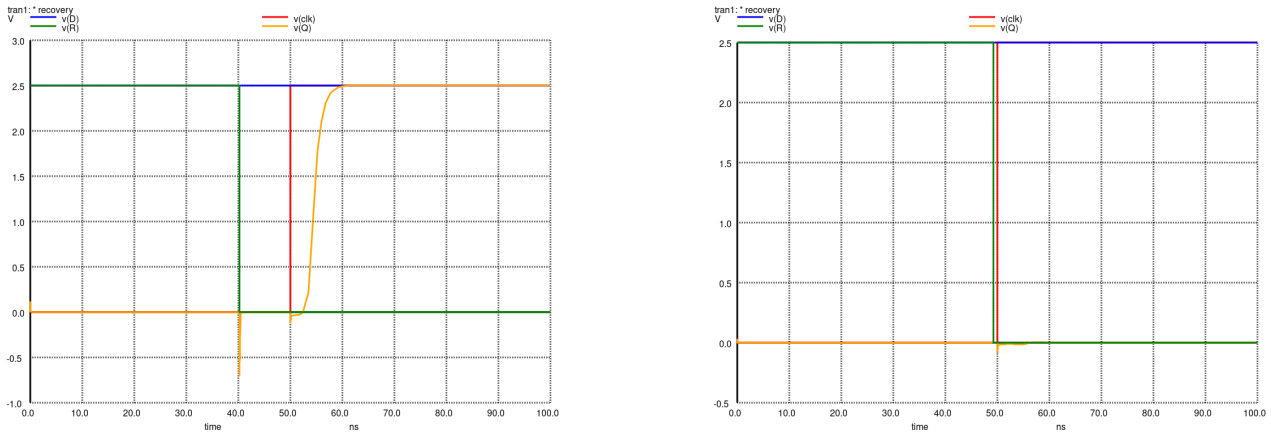

Figure 9: Deactivating R very closely to the active edge of the clock can result in wrong values or metastability in the output

the asynchronous inputs R and S and the active edge of the clock, so that the cell functions properly. Since S and R are active-low, they only have fall constraints.

For S, we can see in figure 8, that, if the deassertion of the signal takes place far enough from the rising clock edge, then the output Q takes the correct value. However, as we approach the falling edge of S towards the active clock edge, we can see that the change of S might not propagate in time to the output and Q might still have a value of 1, even if D and S are both equal to 0, when the active clock edge arrives. We are interested in finding the point where metastability might occur, so we assume that there is a recovery violation, when the output Q does not reach a value smaller than 10% of V_{dd} within the time frame between two consecutive active edges of the clock.

The recovery time for R can be found in a similar way. Figure 9 shows that if we change R very closely to the active clock edge, this change might not propagate in time to the output Q and it will stay low, even though R is deactivated and $D = 1$ when the rising clock edge arrives. We will assume that we have a recovery violation for reset, when the maximum value Q reaches is not more than 90% of V_{dd}.

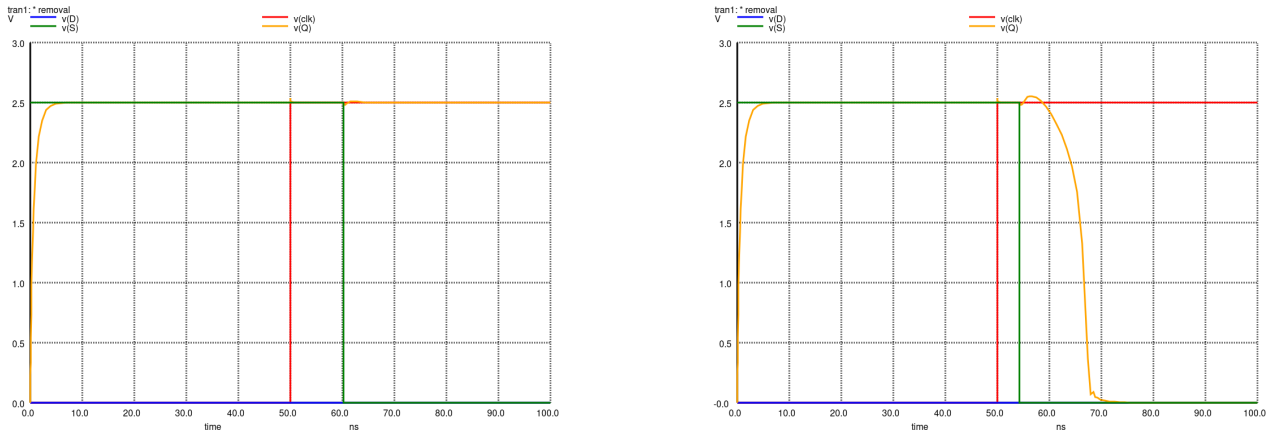

Figure 10: Deactivating S close to the active edge of the clock can result in wrong output or metastability.

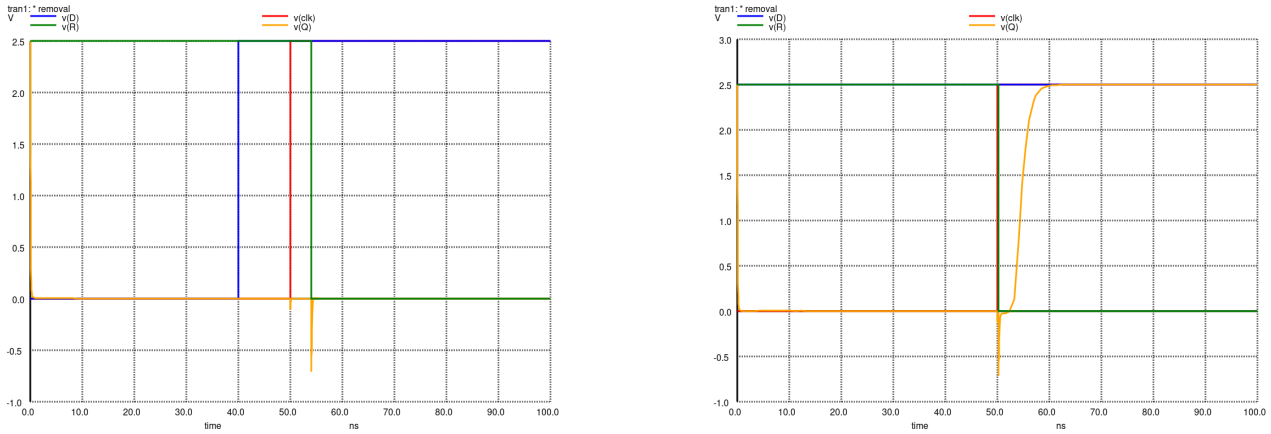

Figure 11: Deactivating R close to the active edge of the clock can result in wrong output or metastability.

Figure 10 shows what a removal violation for S could possibly look like. In the picture on the right, S is deactivated closely to the active edge of the clock and the circuit is fast enough that it propagates this change to the output, which should actually have been high. We assume that we have a removal violation for set, when the output does not reach more than 90% of V_{dd}.

Figure 11 shows what a removal violation for R could possibly look like. In the picture on the right, R is deactivated closely to the active edge of the clock and the circuit is fast enough that it propagates this change to the output, which should actually have been low. We assume that we have a removal violation for reset, when the output does not reaches more than 10% of V_{dd}.

4 Static Timing Analysis Flow

The pseudocode for our tool flow is presented in 1. Figure 12 presents the same pseudocode in a flow diagram format. For each cell, we check all its pins. For each pin, we go through each

timing arc, if one exists, and create a new spice deck based on the skeletons depending on the name and the type of the cell, named **cell.spice** for every item in the measurements list of that arc. Then, for each value of index_1 of the selected template and for each value of index_2 of the same template, we create a spice file that contains the waveforms of the inputs and a spice file that contains the appropriate measurement commands, both of which are included in cell.spice. Then, we run cell.spice appropriately and as many times as needed, collect the results and store them accordingly in the correct output file, which is named:

$$cell\_pin\_related\_pin\_timing\_type/timing\_sense\_measurement\_(when).mt$$

If the pin has a timing type, then we use that instead of the timing sense, else we use the timing type. We add *when* if it exists.

```
for each cell in cells do
    for each pin in cell.pins do
        for each timing in cell.pin.timing do
            Create new spice deck and set initial conditions;
            for each measurement in cell.pin.timing.measurements do
                for each value in index_1 do
                    for each value in index_2 do
                        Create the inputs .spice file;
                        Create the .spice file with the measurement commands;
                        Run the program appropriately;
                        Extract the results and store them accordingly;
                    end
                end
            end
        end
    end
end
```

Algorithm 1: Pseudocode for our automated analysis tool

We will start explaining each of those steps separately and in detail.

4.1 New spice deck and initialization

The first step is to create the file (cell.spice) which we will run using Ngspice. For this purpose, we have 3 skeletons; one for the combinational cells, one for the DFF and one for the DFFRS. We copy the appropriate one depending on the type and name of the cell. Those skeleton files

- Include the path to the .spice file extracted from Magic in the previous part of the assignment
- Include the input and measurement files

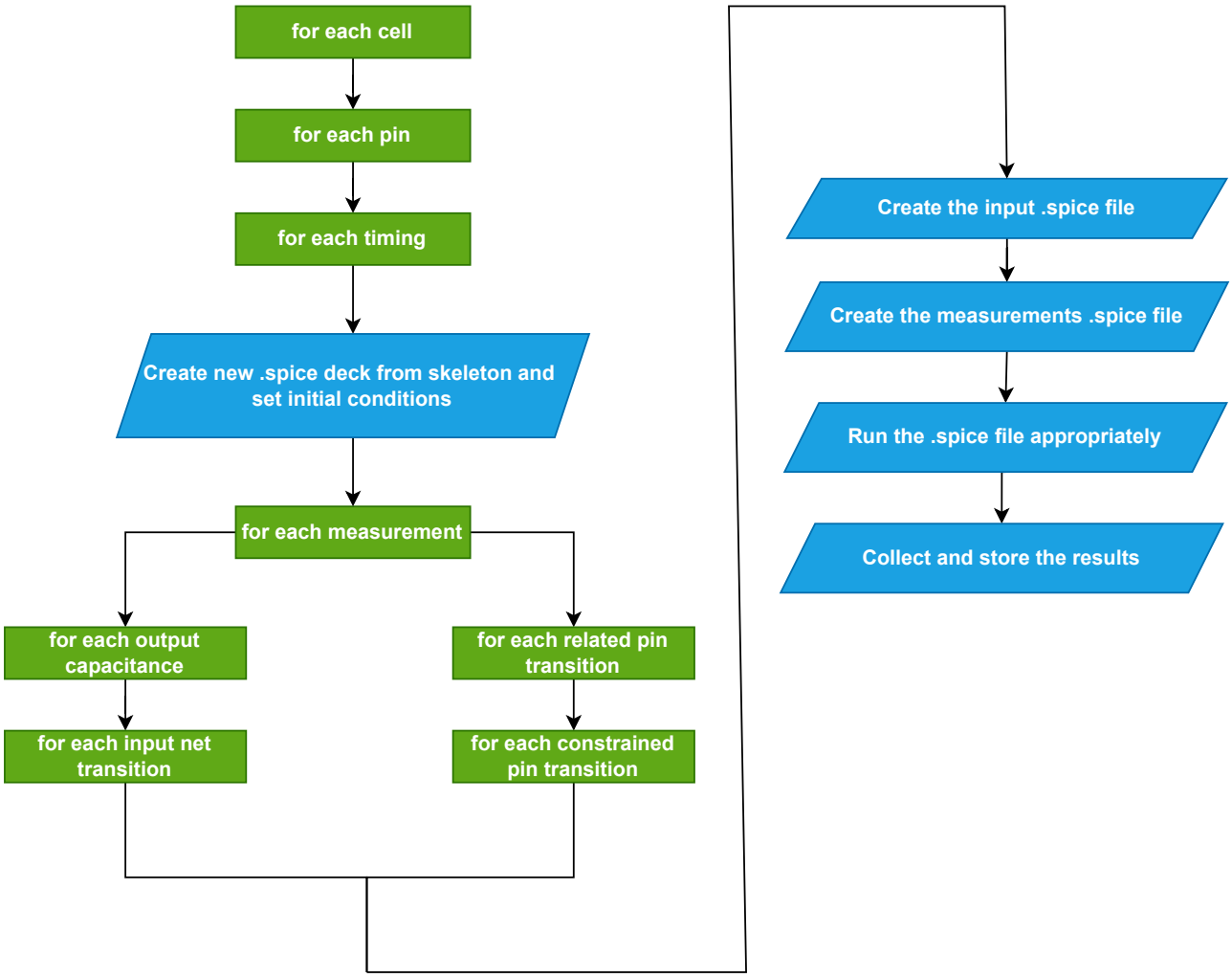

Figure 12: Flow Diagram of our automated tool

- Instatiate the cell as a subcircuit with the appropriate inputs
- Declares the global nodes and the DC inputs
- Instatiates the total output capacitances
- Declares the initial output voltages, using the .ic command
- Indicate the type, duration and tiem step of the analysis (transient for all cases)
- Contain a control segment with just the run command

Most of the above tasks are trivial, except for the initialization of the output voltages and the output capacitances. Pseudocode 2 presents the algorithm for both initializations. Notice that we do not initiaze the output capacitances for the timing arcs. That is because the output capacitance is one of the parameters that affect the timing arcs and we must loop through different values of total_output_net_capacitances and extract the results. For the setup and recovery constraint arcs, we use the maximum output capacitance that exists in the templates, so we can simulate the cell having the greatest delay and for the hold and removal constraints we use the smallest capacitance so that we simulate the case when the cell is fastest.

4.2 Creating the inputs file

The inputs file structure differs depending on if the timing type is a timing or a constraint arc (pin direction is output and input respectively).

4.2.1 Timing arcs

The input file creation is listed in 3. The "when" fields that are read from the configurations are used to set the DC values of the side inputs. Caution should be exercised when setting the value of D. If the cell is "DFFRS" and the related pin is S/R, then the when condition contains the conditions for CLK, D and R/S. If the condition for CLK is $!CLK$, then we just set D to the value that when dictates. However, if the value for CLK is CLK that means that D should have the value dictated by the when clause after the active edge of the clock. If D must be zero when $CLK = 1$ and the measurement is cell_fall or fall_transition, then D must fall after the positive edge of the clock and cannot be a DC value. We must also make sure that S and R change **after** the fall of D. The case is similar for cell_rise and rise_transition when the condition for D is D (4)

If the cell is sequential, then we also need to define the clk and not_clk waveforms. If the related_pin is CLK, then we set the D waveform depending on the measurement and the pin (Q or $\overline{Q}$), as in 5 and if the related_pin is R or S, we set the respective waveform as in 6.

For the combinational cells, the process is easier (7), because the side inputs are set in the when portion and the related_pin can be set easily using the unateness and the timing arc.

```
if type is "combinational" then
    if measurement is "cell_rise" or "rise_transition" then
        init_out = 0
    else
        init_out = V_DD /* measurement is cell_fall or fall_transition  */
    end
else
    if direction is "output" then
        if measurement is "cell_rise" or "rise_transition" then
            /* Pin can be either Q or Q̄, set the other output pin to V_DD
               */
            init_pin = 0
        else
            init_pin = V_DD /* Set the other output pin to 0              */
        end
    else
        /* Constraint Arc                                                 */
        if constraint is "setup" then
            if measurement is "rise_constraint" then
                init_Q = 0
            else
                init_Q = V_DD
            end
            OUTPUT_CAPACITANCE = MAX
        else if constraint is "hold" then
            if measurement is "rise_constraint" then
                init_Q = V_DD
            else
                init_Q = 0
            end
            OUTPUT_CAPACITANCE = MIN
        else if constraint is "recovery" then
            if pin is "S" then
                init_Q = V_DD
            else
                /* Else pin is "R"                                        */
                init_Q = 0
            end
            OUTPUT_CAPACITANCE = MAX
        else
            if pin is "S" then
                init_Q = 0
            else
                /* Else pin is "R"                                        */
                init_Q = V_DD
            end
            OUTPUT_CAPACITANCE = MIN
    end
end
```

Algorithm 2: Pseudocode for output voltages and capacitance initialization

```
for total_output_net_capacitance in Timing Template do
    Replace the old output capacitance in the cell.spice file;
    for input_net_transition in Timing Template do
        Calculate the 0-100% input_net_transition;
        Set the DC Voltages from "when" (except for CLK);
        if cell is "sequential" then
            Set clk and not_clk signals;
            if related_pin is "CLK" then
                Set D to the appropriate waveforms depending on rise or fall
            else if related_pin is "R" or "S" then
                Set R and S to the appropriate waveforms
        else
            Set the related pin initial and final voltage to the appropriate value
             depending on timing sense and measurement
        end
        Create the measurements file
    end
end
```

Algorithm 3: Creation of inputs file for timing arcs

4.2.2 Constraint arcs

The input file creation for the constraint arcs is shown in 8 and 9. If a pin has constraint arcs, then the cell is definitely sequential, so we need to set the clock and its inverse signal. We also need to set the DC voltages from the *when* conditions (we don't have any special cases as we did in the timing arcs).

4.3 Creating the measurements files

4.3.1 Timings arcs

The pseudocode for creating the measurements file is listed in 10 and continues in 11. After the creation of the measurements file, we just run the cell.spice deck once, extract the results and store them in a file with a name that corresponds to this specific measurement.

4.3.2 Constraint arcs

For the constraint arcs, the answer is not straightforward; we need to run multiple simulations to estimate the setup, hold, recovery and removal times (12, 13, 14 and 15). The basic principle is that we start to change the value of the related pin (*start*) on the active edge of the clock. If we notice that there is no constraint violation, we set the constraint value to zero and exit. Else, we move *start* earlier or later, depending on the constraint, until the constraint is not violated with a relatively large step. We do this to find the approximate value of the constraint. To have more accuracy and enough resolution to be able to distinguish between successive measurements, we then begin to move *start* to the opposite direction from where

```
for condition in when conditions do
    if condition contains "D" then
        if when contained "!CLK" then
            Set the voltage of D to that of the "when"
        else
            if cell is "DFFRS" then
                if when_cond is "!D" and measurement is "cell_fall" or "fall_transition"
                 then
                    if pin is "Q" then
                        D waveform is 1 before the clock and goes to zero after the active
                         edge of the clock
                    else
                        Set the condition
                    end
                else if when_cond is "D" and measurement is "cell_rise" or
                 "rise_transition" then
                    if pin is "Q̄" then
                        D waveform is 0 before the clock and goes to high after the active
                         edge of the clock
                    else
                        Set the condition
                    end
                else
                    Set the condition
                end
                Set the condition
            end
        end
    else
        if condition does not contain "CLK" then
            Set the condition
        end
    end
end
```

Algorithm 4: Set voltages of when conditions for timing arcs

```
if measurement is "cell_rise" or "rise_constraint" then
    if pin is "Q" then
        D should go from 0 to V_DD
    else
        D should go from V_DD to 0
    end
else
    if pin is "Q" then
        D should go from V_DD to 0
    else
        D should go from 0 to V_DD
    end
end
```

Algorithm 5: D waveforms if pin is Q or $\overline{Q}$ and related_pin is CLK

if *measurement is "cell_rise" or "rise_transition"* **then**
  if *timing_sense is "positive_unate"* **then**
    Related Pin should go from 0 to V_{DD}
  else
    Related Pin should go from V_{DD} to 0
  end
else
  if *timing_sense is "negative_unate"* **then**
    Related Pin should go from 0 to V_{DD}
  else
    Related Pin should go from V_{DD} to 0
  end
end

Algorithm 6: S and R waveforms if related_pin is S or R

if *measurement is "cell_rise" and "rise_transition"* **then**
  if *timing_sense is "positive_unate"* **then**
    The pin should go from 0 to V_{DD}
  else
    The pin should go from 0 to V_{DD}
  end
else
  if *timing_sense is "negative_unate"* **then**
    The pin should go from 0 to V_{DD}
  else
    The pin should go from 0 to V_{DD}
  end
end

Algorithm 7: Inputs for combinational cells

for *related_pin_transition in Constraint Template* **do**
  Calculate the 0-100% related_pin_transition;
  for *constraint_pin_transition in Constraint Template* **do**
    Calculate the 0-100% constrained_pin_transition;
    `/* If the cell has constraint arcs it is sequential */`
    Set clk and not_clk signals;
    Set the DC Voltages from "when";
    Create the waveform for input D, S and R;
    Create the measurements file;
  end
end

Algorithm 8: Creation of inputs file for constraint arcs

if *constraint is "setup"* **then**
  if *measurement is "rise_constraint"* **then**
    Set D to go from 0 to V_{DD}
  else
    Set D to go from V_{DD} to 0
  end
else if *constraint is "hold"* **then**
  if *measurement is "rise_constraint"* **then**
    Set D to go from 0 to V_{DD}
  else
    Set D to go from V_{DD} to 0
  end
else if *constraint is "recovery"* **then**
  if *pin is "S"* **then**
    Set D to go from V_{DD} to 0
  else if *pin is "R"* **then**
    Set D to go from 0 to V_{DD}
  `/* The voltages for S and R will be set during simulations */`
else
  if *pin is "S"* **then**
    Set D to go from V_{DD} to 0
  else if *pin is "R"* **then**
    Set D to go from 0 to V_{DD}
  `/* The voltages for S and R will be set during simulations */`

Algorithm 9: Inputs for constraint arc per constraint

if *type is "combinational"* **then**
  if *measurement is "cell_rise" and timing_sense is "positive_unate"* **then**
    .measure tran cell_rise trig V(related_pin) val = 0.5 * V_{DD} rise = 1 targ V(pin) = 0.5 * V_{DD} rise = 1
  else if *measurement is "cell_rise" and timing_sense is "negative_unate"* **then**
    .measure tran cell_rise trig V(related_pin) val = 0.5 * V_{DD} fall = 1 targ V(pin) = 0.5 * V_{DD} rise = 1
  else if *measurement is "cell_fall" and timing_sense is "positive_unate"* **then**
    .measure tran cell_fall trig V(related_pin) val = 0.5 * V_{DD} fall = 1 targ V(pin) = 0.5 * V_{DD} fall = 1
  else if *measurement is "cell_fall" and timing_sense is "negative_unate"* **then**
    .measure tran cell_fall trig V(related_pin) val = 0.5 * V_{DD} rise = 1 targ V(pin) = 0.5 * V_{DD} fall = 1
  else if *measurement is "rise_transition"* **then**
    .measure tran rise_transition trig V(pin) val = 0.3 * V_{DD} rise = 1 targ V(pin) = 0.7 * V_{DD} rise = 1
  else if *measurement is "fall_transition"* **then**
    .measure tran fall_transition trig V(pin) val = 0.7 * V_{DD} fall = 1 targ V(pin) = 0.3 * V_{DD} fall = 1
else
  if *related_pin is "CLK"* **then**
    if *measurement is "cell_rise"* **then**
      if *timing_type is "rising_edge"* **then**
        .measure tran cell_rise trig V(CLK) val=0.5*V_{DD} rise = 1 targ V(pin) val=0.5*V_{DD} rise = 1
      else
        .measure tran cell_rise trig V(CLK) val=0.5*V_{DD} fall = 1 targ V(pin) val=0.5*V_{DD} rise = 1
      end
    if *measurement is "cell_fall"* **then**
      if *timing_type is "rising_edge"* **then**
        .measure tran cell_fall trig V(CLK) val=0.5*V_{DD} rise = 1 targ V(pin) val=0.5*V_{DD} fall = 1
      else
        .measure tran cell_fall trig V(CLK) val=0.5*V_{DD} fall = 1 targ V(pin) val=0.5*V_{DD} fall = 1
      end
    if *measurement is "rise_transition"* **then**
      .measure tran rise_transition trig V(pin) val=0.3*V_{DD} rise = 1 targ V(pin) val=0.7*V_{DD} rise = 1
    else if *measurement is "fall_transition"* **then**
      .measure tran fall_transition trig V(pin) val=0.3*V_{DD} fall = 1 targ V(pin) val=0.7*V_{DD} fall = 1
  else
    Algorithm 11
  end
end

Algorithm 10: Algorithm for creating measurements file

if *measurement is "cell_rise"* **then**
  if *timing_type is "positive_unate"* **then**
    .measure tran cell_rise trig V(related_pin) val=0.5*V_{DD} rise = 1 targ V(pin) val=0.5*V_{DD} rise = 1
  else
    .measure tran cell_rise trig V(related_pin) val=0.5*V_{DD} fall = 1 targ V(pin) val=0.5*V_{DD} rise = 1
  end
if *measurement is "cell_fall"* **then**
  if *timing_type is "negative_unate"* **then**
    .measure tran cell_fall trig V(related_pin) val=0.5*V_{DD} rise = 1 targ V(pin) val=0.5*V_{DD} fall = 1
  else
    .measure tran cell_fall trig V(related_pin) val=0.5*V_{DD} fall = 1 targ V(pin) val=0.5*V_{DD} fall = 1
  end
if *measurement is "rise_transition"* **then**
  .measure tran rise_transition trig V(pin) val=0.3*V_{DD} rise = 1 targ V(pin) val=0.7*V_{DD} rise = 1
else if *measurement is "fall_transition"* **then**
  .measure tran fall_transition trig V(pin) val=0.7*V_{DD} fall = 1 targ V(pin) val=0.3*V_{DD} fall = 1

Algorithm 11: Continuation from algorithm 10

we left with a much smaller step, until we find when the first violation occurs. Finally, we calculate the constraint value as the difference between the clock edge and the value of *start* (or opposite for hold and removal) and add an additional 30% to make sure that there will be no constraint violation.

4.4 Library File Composition

After the end of all simulations, we loop through all measurements in a similar fashion as before, open the corresponding .mt file and write them properly in the lib file. This part of the program also computes the area for each cell, using the path of the extracted spice file.

4.5 Verilog modules

The creation of the Verilog modules required defining the primitives, instatiating the primitives inside the actual module and using specify blocks to model the delays between input and output.

For the combinational circuits, defining the primitives was straightforward, since the entire truth table can be extracted by evaluating the function associated with the output pin using Python. For the sequential circuits, however, using the configuration file, we had to identify the active clock edge and whether the S and R asynchronous inputs were active-low or active-high.

For the specify blocks, our task was to initialize them to a unit delay for this part of the project. We created associations for each input with every output pin.

```
/* D has been set in the inputs previously for the below measurement   */
```
Measure the $CLK - to - Q$ delay without setup violation (c2q);
```
/* start is the time where D starts changing                          */
```
Set variable *start* equal to the active edge of the CLK;
Measure the $CLK - to - Q$ delay (using .measure) (cur_c2q);
if *cur_c2q < 1.05 * c2q* **then**
    setup = 0;
    break;
end
dt = 1ns;
while *cur_c2q ≥ 1.05 * c2q* **do**
    start = start - dt;
    Measure cur_c2q;
end
dt = 1ps;
while *cur_c2q < 1.05 * c21* **do**
    start = start + dt;
    Measure cur_c2q;
end
setup = 1.3*(active_clk_edge - start);

Algorithm 12: Setup simulations

```
/* D has been set in the inputs previously for the below measurement   */
```
Measure the $CLK - to - Q$ delay without hold violation (c2q);
```
/* start is the time where D starts changing                          */
```
Set variable *start* equal to the active edge of the CLK;
Measure the $CLK - to - Q$ delay (using .measure) (cur_c2q);
if *cur_c2q < 1.05 * c2q* **then**
    hold = 0;
    break;
end
dt = 1ns;
while *cur_c2q ≥ 1.05 * c2q* **do**
    start = start + dt;
    Measure cur_c2q;
end
dt = 1ps;
while *cur_c2q < 1.05 * c21* **do**
    start = start - dt;
    Measure cur_c2q;
end
hold = 1.3*(active_clk_edge - start);

Algorithm 13: Hold simulations

```
/* deassert is the time where S or R starts changing                    */
Set variable deassert equal to the active edge of the CLK;
if pin is "S" then
    min = 0;
    Measure the minimum value Q takes in the interval between 2 active edges of the
     clock. if min < 0.1 * V_DD then
        recovery = 0;
        break;
    end
    dt = 10ns;
    while min < 0.1 * V_DD do
        deassert = deassert - dt;
        Measure min;
    end
    dt = 10ps;
    while min ≥ 0.1 * V_DD do
        deassert = deassert + dt;
        Measure min;
    end
else
    max = 0;
    Measure the maximum value Q takes in the interval between 2 active edges of the
     clock. if max > 0.9 * V_DD then
        recovery = 0;
        break;
    end
    dt = 10ns;
    while max < 0.9 * V_DD do
        deassert = deassert - dt;
        Measure max;
    end
    dt = 10ps;
    while max ≥ 0.9 * V_DD do
        deassert = deassert + dt;
        Measure max;
    end
end
recovery = 1.3 * (active_clk_edge - deassert);
```

Algorithm 14: Recovery simulations

```
/* deassert is the time where S or R starts changing                    */
Set variable deassert equal to the active edge of the CLK;
if pin is "S" then
    min = V_DD;
    Measure the minimum value Q takes in the interval between 2 active edges of the
     clock. if min > 0.9 * V_DD then
        removal = 0;
        break;
    end
    dt = 10ns;
    while min < 0.9 * V_DD do
        deassert = deassert + dt;
        Measure min;
    end
    dt = 10ps;
    while min ≥ 0.9 * V_DD do
        deassert = deassert - dt;
        Measure min;
    end
else
    max = V_DD;
    Measure the maximum value Q takes in the interval between 2 active edges of the
     clock. if max < 0.1 * V_DD then
        removal = 0;
        break;
    end
    dt = 10ns;
    while max > 0.1 * V_DD do
        deassert = deassert + dt;
        Measure max;
    end
    dt = 10ps;
    while max ≤ 0.1 * V_DD do
        deassert = deassert - dt;
        Measure max;
    end
end
removal = 1.3 * (active_clk_edge - deassert);
```

Algorithm 15: Removal simulations

5 Conclusion

The results indicate that the propagation delay and rise and fall delays are greater as the total output net capacitance and input net transition increase. For both the DFF and the DFFRS, the hold time is 0. The recovery time is mostly 0 for the DFFRS, except for the cases where the constrained pin transition is the maximum and the removal time is mostly non-zero except for the case, where, again, the constrained pin transition is the maximum.