NCL Throughput Optimization

NCL systems can be optimized for speed by partitioning the combinational circuitry and inserting additional NCL registers and corresponding completion components. However, NCL circuits cannot be partitioned arbitrarily; they can only be divided at component boundaries in order to preserve delay-insensitivity. The average cycle time for an NCL system, T_{DD}, can be estimated as the worse-case stage delay of any stage in the pipeline, where the delay of one stage is equal to twice the sum of the stage's worse-case combinational delay and completion delay, to account for both the DATA and NULL wavefronts. Algorithm 1 depicts this calculation for an N-stage pipeline, where $Dcomb_i$ and $Dcomp_i$ are stage$_i$'s combinational and completion delays, respectively.

$$
\begin{array}{l}
T_{DDmax} = 2 \times (Dcomb_1 + Dcomp_1) \\
\text{for (i = 2 to N) loop} \\
\quad T_{DDtemp} = 2 \times (Dcomb_i + Dcomp_i) \\
\quad T_{DDmax} = MAX(T_{DDtemp},\ T_{DDmax}) \\
\text{end loop}
\end{array}
$$

Algorithm 1. NCL T_{DD} estimation.

NCL pipelining can utilize either of two completion strategies: full-word or bit-wise completion. Full-word completion, as shown in Figure 1, requires that the acknowledge signals from each bit in register$_i$ be conjoined together by the completion component, whose single-bit output is connected to all request lines of register$_{i-1}$. On the other hand, bit-wise completion, as shown in Figure 2, only sends the completion signal from bit *b* in register$_i$ back to the bits in register$_{i-1}$ that took part in the calculation of bit *b*. This method may therefore require fewer logic levels than that of full-word completion, thus increasing throughput. In this example, bit-wise completion is faster (i.e., 1 gate delay vs. 2 gate delays), but it requires more area (i.e., 4 gates vs. 2 gates).

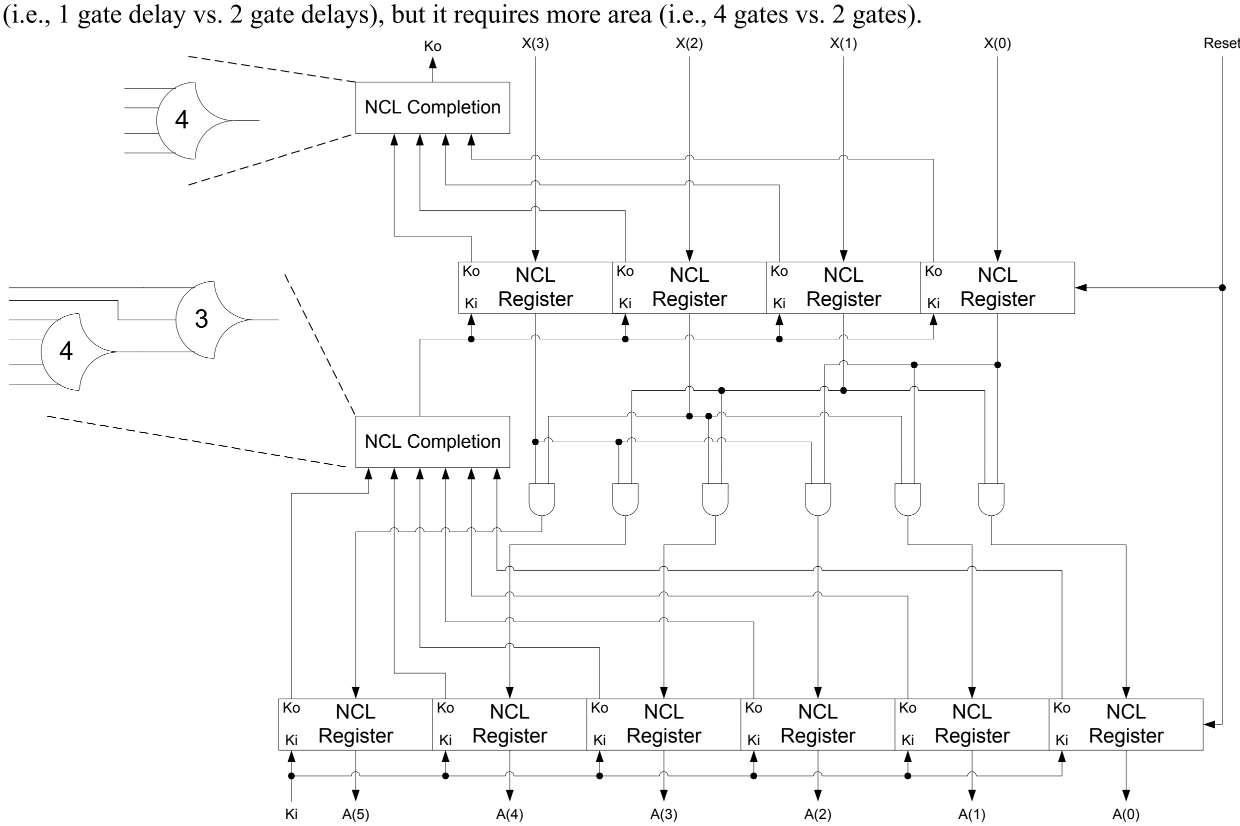


Figure 1. Full-word completion.

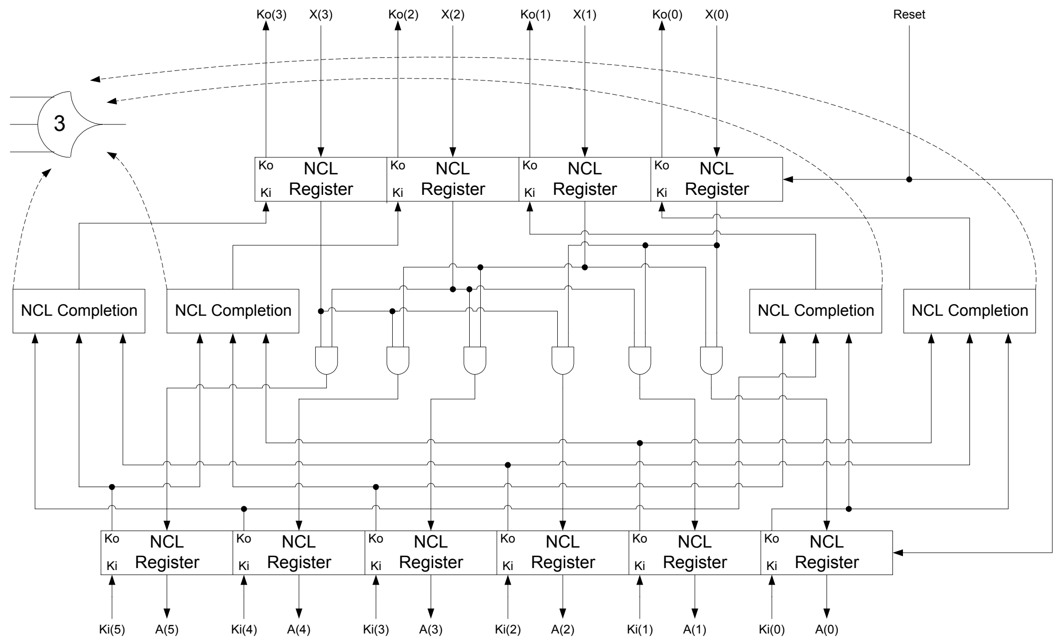


Figure 2. Bit-wise completion.

To maximize throughput while minimizing latency and area, the following algorithm should be used to optimally partition an NCL circuit. Steps 1 and 2 initially partition an NCL circuit into stages of *primary components*, where a primary component is defined as a component whose inputs only consist of the circuit's inputs or outputs of components that have already been added to a previous stage. Steps 3 and 4 then calculate the combinational delay (i.e., *Dcomb*) and completion delay (i.e., *Dcomp*) for each stage and the maximum delay for the entire pipeline (i.e., *max_delay*), utilizing both full-word and bit-wise completion strategies. Finally, Step 5 merges stages to reduce latency and area, as long as doing so does not decrease throughput. Note that when merging stages the new merged combinational delay (i.e., *merged_comb*) is not necessarily $Dcomb_i + Dcomb_{i+1}$. Take for example two full adders in a ripple-carry adder: $Dcomb_i = 2$ and $Dcomb_{i+1} = 2$, but merged_comb = 3, since the *carry* output of a full adder has only 1 gate delay.

```
1)  i = 1
2)  loop until all components are part of a stage          -- initially partition into stages
        add all primary components to stage_i
        i = i + 1
    end loop
3)  N = i −1
    max_delay_FW = 0
    max_delay_BW = 0
4)  for j in 1 to N loop                                   -- calculate worse-case cycle times
        Dcomb = max delay of stage_i's components          -- for both full-word and bit-wise
        B = # of outputs from stage_j                      -- completion
        Dcomp_j = ⌈Log_4 B⌉
        if ((Dcomb + Dcomp_j) > max_delay_FW) then
            max_delay_FW = Dcomb + Dcomp_j
        end if
        B = # of inputs to stage_j
        max_outputs = 1
        for i in 1 to B loop
            num_outputs = number of outputs of stage_j generated by input_i
            if (num_outputs > max_outputs) then
                max_outputs = num_outputs
            end if
        end loop
```

```
        Dcomp = ⌈Log4 max_outputs⌉
        if ((Dcomb + Dcomp) > max_delayBW) then
            max_delayBW = Dcomb + Dcomp
        end if
    end loop
5)  if (max_delayFW > max_delayBW) then                                  -- bit-wise design is faster
        num_stages = call mergeBW function
        output bit-wise pipelined design
    elsif (max_delayBW > max_delayFW) then                               -- full-word design is faster
        num_stages = call mergeFW function
        output full-word pipelined design
    else
        num_stagesBW = call mergeBW function
        num_stagesFW = call mergeFW function
        if (num_stagesBW > num_stagesFW) then                            -- full-word design has less latency
            output full-word pipelined design
        elsif (num_stagesFW > num_stagesBW) then                         -- bit-wise design has less latency
            output bit-wise pipelined design
        elsif (area of full-word design > area of bit-wise design) then  -- bit-wise design is smaller
            output bit-wise pipelined design
        else
            output full-word pipelined design
        end if
    end if
                                 mergeFW function
    num_stages = N
    for k in 1 to N-1 loop                                               -- merge stages to decrease latency
        merged_comb = max combinational delay of stage_k and stage_k+1 merged into a single stage
        if (merged_comb + comp_k+1 ≤ max_delayFW) then
            merge stage_k into stage_k+1
            delete stage_k
            num_stages = num_stages – 1
        end if
    end loop
    return num_stages
                                 mergeBW function
    num_stages = N
    for k in 1 to N-1 loop                                               -- merge stages to decrease latency
        merged_comb = max combinational delay of stage_k and stage_k+1 merged into a single stage
        B = # of inputs to stage_k
        max_outputs = 1
        for i in 1 to B loop
            num_outputs = number of outputs of stage_k+1 generated by input_i
            if (num_outputs > max_outputs) then
                max_outputs = num_outputs
            end if
        end loop
        merged_comp = ⌈Log4 max_outputs⌉
        if (merged_comb + merged_comp ≤ max_delayBW) then
            merge stage_k into stage_k+1
            delete stage_k
            num_stages = num_stages - 1
        end if
    end loop
    return num_stages
```

Algorithm 2. NCL pipelining algorithm.

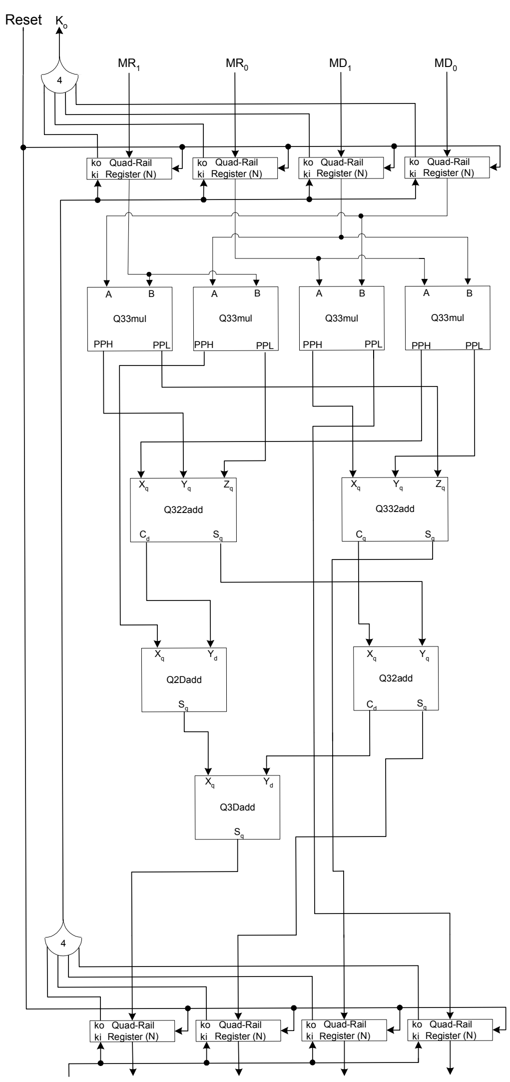
As an example, the non-pipelined quad-rail multiplier in Figure 3 has a worse-case combinational delay of 8 and a completion delay of 1, such that T_{DD} = 18. Applying Steps 1-4 of the pipelining algorithm to the quad-rail multiplier yields the results shown in Tables 1 and 2 for full-word and bit-wise completion, respectively. These tables show that the full-word pipelined design has a T_{DD} (i.e. 2 × *max_delay*, to account for both the DATA and NULL wavefronts) of 10 gate delays, while the bit-wise pipelined design has a T_{DD} of 8 gate delays; hence the bit-wise pipelined design is preferred, since it maximizes throughput. Applying Step 5 of the algorithm to merge stages results in both the full-word and bit-wise pipelined designs merging Stages 3 and 4, such that both designs only require 3 stages. The new D_{comb} is 3 and the new stage delay for both designs is 4. Note that *max_outputs* for the bit-wise design changes to 2 for the merged stage, such that D_{comp} becomes 1.

Table 1. Full-word completion pipelining.

Stage	D_{comb}	# Outputs	D_{comp}	delay
1	2	8	2	4
2	3	6	2	5
3	2	5	2	4
4	1	4	1	2
			max_delay	5

Table 2. Bit-wise completion pipelining.

Stage	D_{comb}	max_outputs	D_{comp}	delay
1	2	4	1	3
2	3	2	1	4
3	2	2	1	3
4	1	1	0	1
			max_delay	4



Component Type	Output Gate Delays	
	Carry / PPH	Sum / PPL
Q33mul	1	2
Q332add	3	3
Q322add	2	3
Q32add	2	2
Q2Dadd	N/A	1
Q3Dadd	N/A	1

Figure 3. 4-bit × 4-bit unsigned quad-rail multiplier.

NCL system throughput can also be increased by applying the NULL Cycle Reduction (NCR) technique, depicted in Figure 4, which increases the throughput of an NCL system by decreasing the circuit's NULL cycle time, without affecting its DATA cycle time. Successive input wavefronts are partitioned so that one circuit processes a DATA wavefront, while its duplicate processes a NULL wavefront. The first DATA/NULL cycle flows through the original circuit, while the next DATA/NULL cycle flows through the duplicate circuit. The outputs of the two circuits are then multiplexed to form a single output stream. NCR can be used to speedup slow stages in a NCL pipeline that cannot be further divided (e.g., Stage 2 in the quad-rail multiplier shown in Figure 3). The application of NCR to only the slow stages in a pipeline increases the throughput for the entire pipeline. NCR can also be used to increase the throughput of a feedback loop, which cannot be increased by any other means, again increasing throughput for the entire pipeline. Figure 4 depicts the NCR architecture for a dual-rail logic circuit utilizing full-word completion; however, NCR is also applicable to quad-rail circuits and bit-wise completion. Quad-rail logic only requires a redesign of the Demultiplexer and Multiplexer circuits to handle quad-rail signals, whereas bit-wise completion requires removal of the Completion Detection component and replication of the Sequencer components, such that each input/output bit has its own Sequencer#1/Sequencer#2 component, respectively.

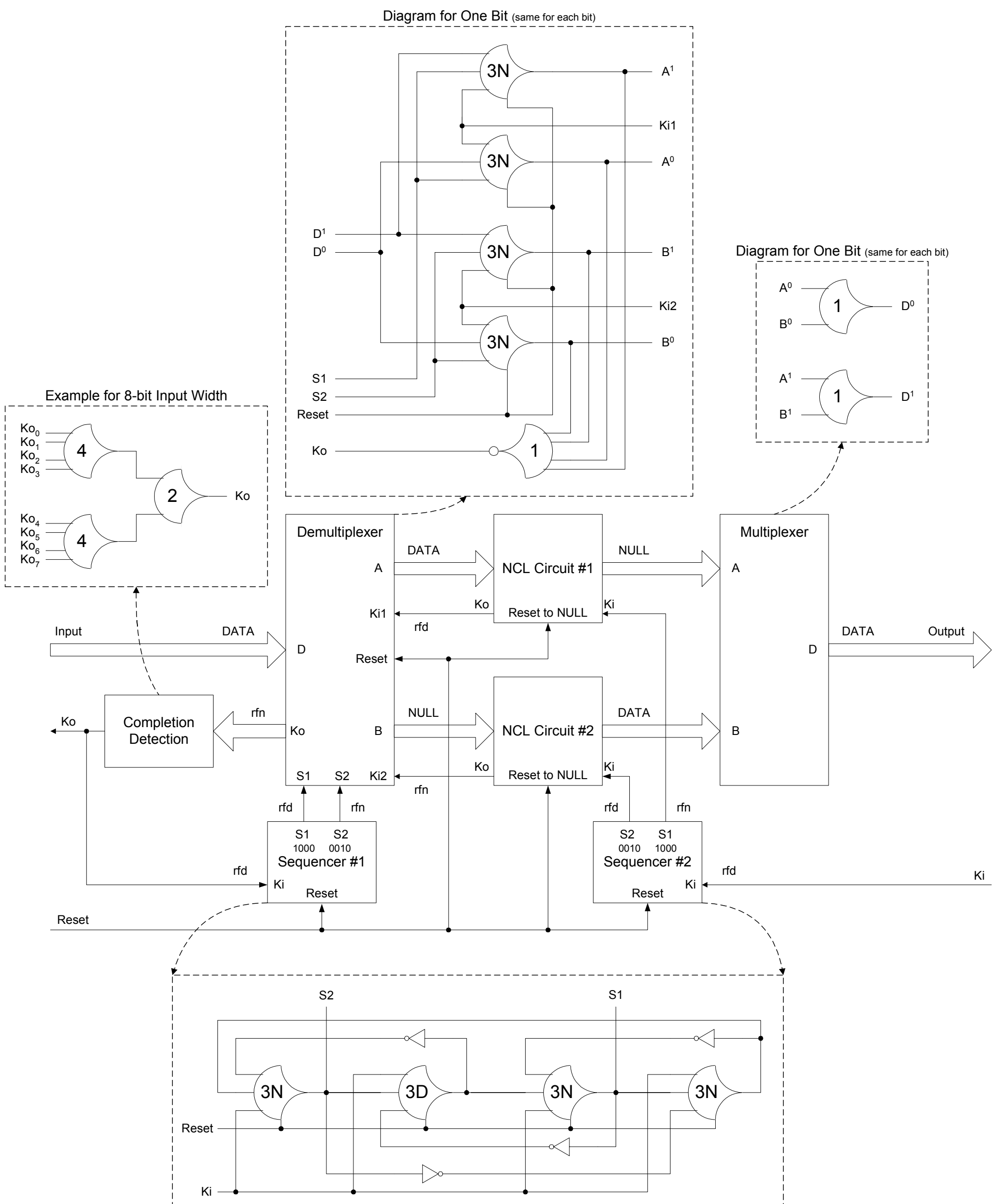


Figure 4. NCR architecture for a dual-rail circuit utilizing full-word completion.