Lecture 4

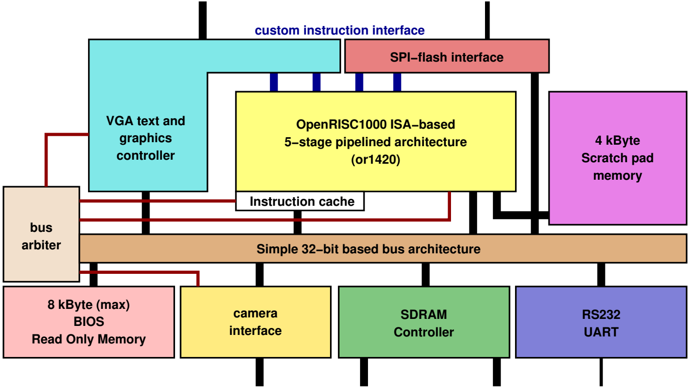
Embedded system design

Timing closure

CS476 - ESD
March 11, 2024

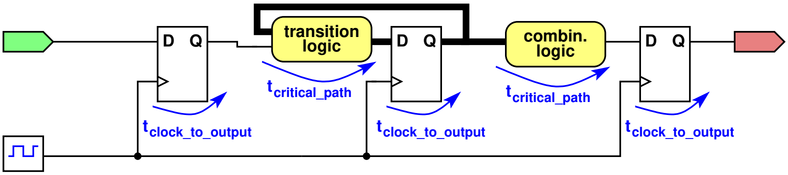
Dr. Theo Kluter
EPFL

Introduction



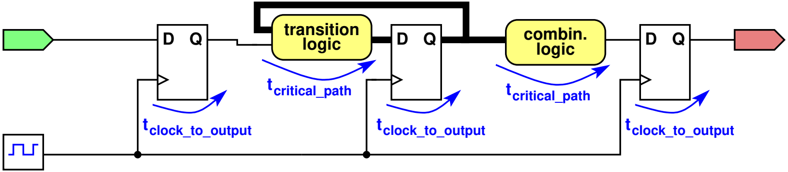
- Once we finished our architectural choices, we have to get the system running at the required frequency.
- We have to go into a phase which is called *timing closure*.
- To fully understand the timing closure we have first to go into some details of the final ASIC to be able to understand what is going on.

Remember: RTL design



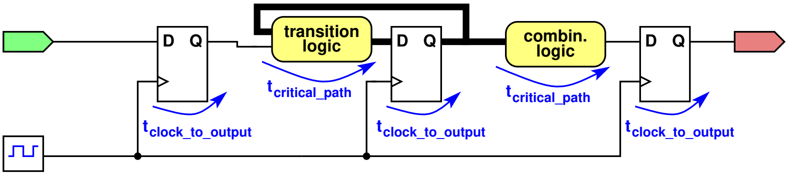
- All our designs we design synchronously using the Register Transfer Level (RTL) methodology.
- Hence all our circuits look like the simplified circuit above, where all flipflops are connected to the same clock source (throughout our chip).

Remember: RTL design



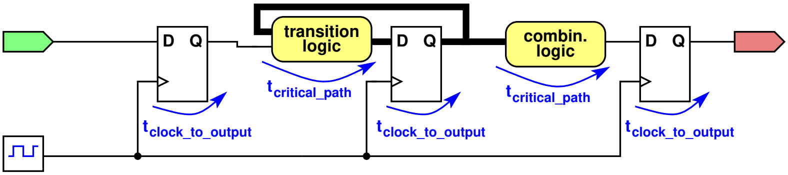
- All our designs we design synchronously using the Register Transfer Level (RTL) methodology.
- Hence all our circuits look like the simplified circuit above, where all flipflops are connected to the same clock source (throughout our chip).
- We know that due to transistor capacitance's all gates have a gate delay that causes hazards.

Remember: RTL design



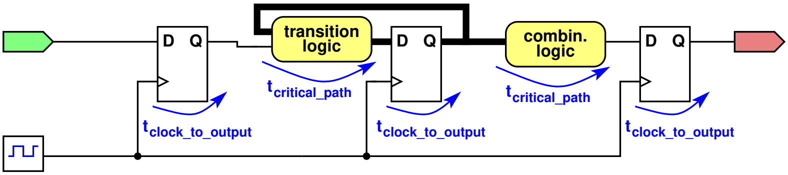
- All our designs we design synchronously using the Register Transfer Level (RTL) methodology.
- Hence all our circuits look like the simplified circuit above, where all flipflops are connected to the same clock source (throughout our chip).
- We know that due to transistor capacitance's all gates have a gate delay that causes hazards.
- The longest combinational path hence represents the critical path.

Remember: RTL design



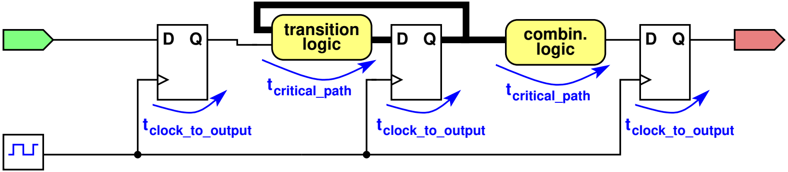
- All our designs we design synchronously using the Register Transfer Level (RTL) methodology.
- Hence all our circuits look like the simplified circuit above, where all flipflops are connected to the same clock source (throughout our chip).
- We know that due to transistor capacitance's all gates have a gate delay that causes hazards.
- The longest combinational path hence represents the critical path.
- The one thing that we did not consider yet is the question: *What happens with the clock line?*

Remember: RTL design



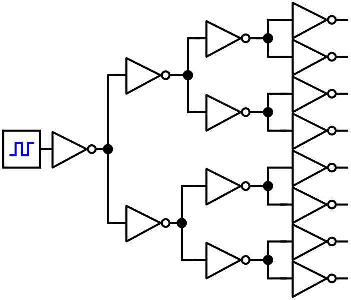
- All our designs we design synchronously using the Register Transfer Level (RTL) methodology.
- Hence all our circuits look like the simplified circuit above, where all flipflops are connected to the same clock source (throughout our chip).
- We know that due to transistor capacitance's all gates have a gate delay that causes hazards.
- The longest combinational path hence represents the critical path.
- The one thing that we did not consider yet is the question: *What happens with the clock line?*
- Just putting a wire over the whole chip probably will not work as:
    1. The clock line would have a big capacitive load.

Remember: RTL design



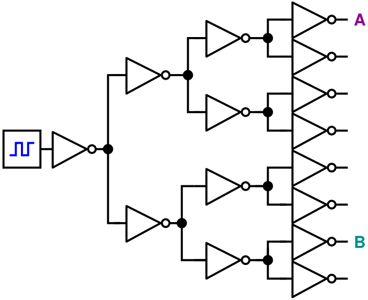
- All our designs we design synchronously using the Register Transfer Level (RTL) methodology.
- Hence all our circuits look like the simplified circuit above, where all flipflops are connected to the same clock source (throughout our chip).
- We know that due to transistor capacitance's all gates have a gate delay that causes hazards.
- The longest combinational path hence represents the critical path.
- The one thing that we did not consider yet is the question: *What happens with the clock line?*
- Just putting a wire over the whole chip probably will not work as:
  1. The clock line would have a big capacitive load.
  2. The RTL-design method assumes that the rising edge of the clock arrives at all flipflops at the same time.

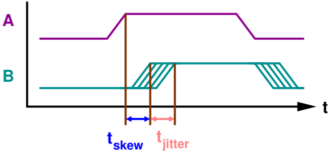
Avoiding big capacitive load



- Let's look into the first point: reducing the big capacitive load:
- Using a binary tree of inverters will reduce the load on each output, however, what is the result of this operation?

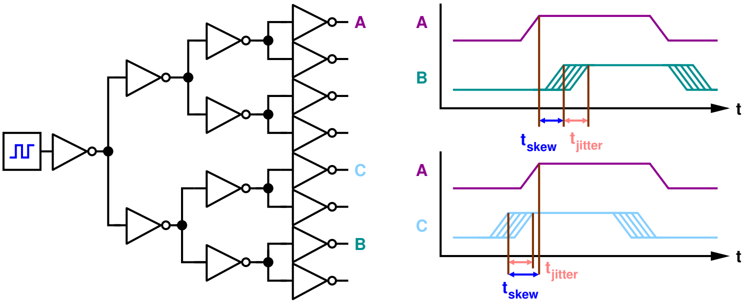
Avoiding big capacitive load





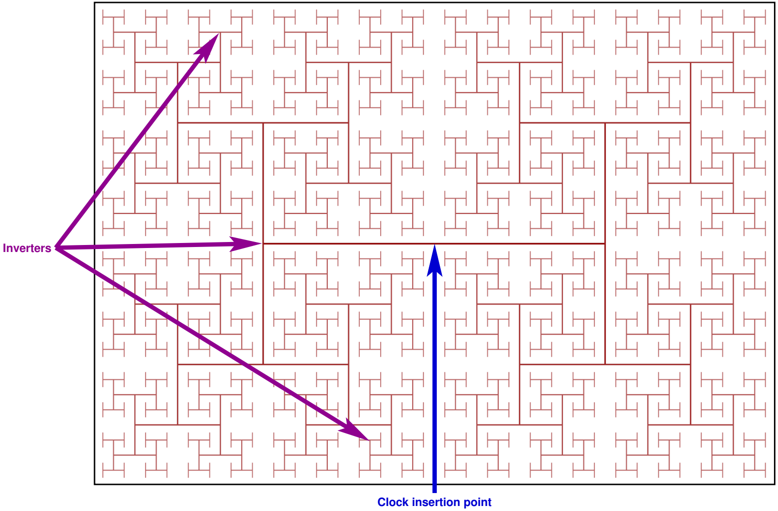
- Let's look into the first point: reducing the big capacitive load:
- Using a binary tree of inverters will reduce the load on each output, however, what is the result of this operation?
- We will introduce at the flipflop levels a clock-skew due to the fact that not all inverters have the same delay and line-length-mismatches.
- We also will have a jitter.

Avoiding big capacitive load



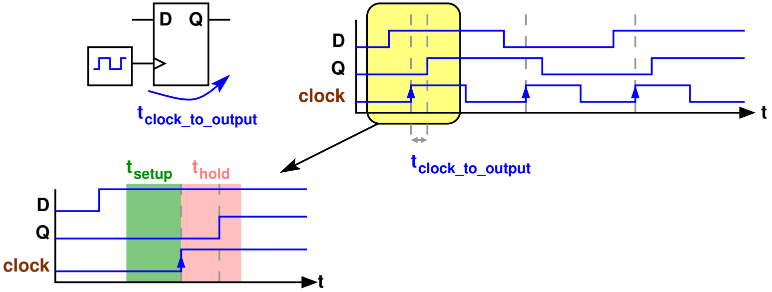
- Let's look into the first point: reducing the big capacitive load:
- Using a binary tree of inverters will reduce the load on each output, however, what is the result of this operation?
- We will introduce at the flipflop levels a clock-skew due to the fact that not all inverters have the same delay and line-length-mismatches.
- We also will have a jitter.
- Note that we can also have a negative skew that reduces the influence of the jitter.

Reducing jitter and skew



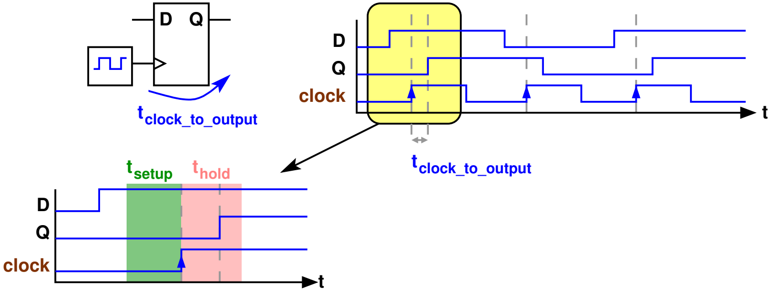
- One of the methods is to make a *clock tree* in form of a H-tree.
- However, we still have a clock-uncertainty of approx. $2 \cdot t_{skew} + t_{jitter}$.

Remember: Setup and hold



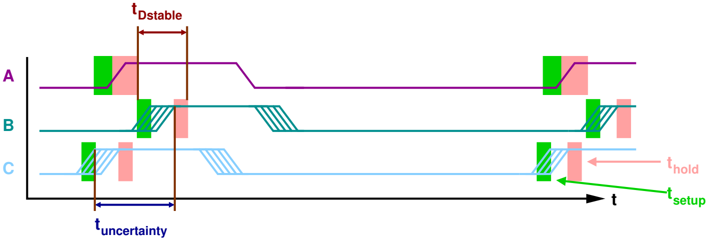
- Remember: a real flipflop has a setup and hold time in which the D-input needs to be kept stable (otherwise the flipflop goes into meta stable state).

Remember: Setup and hold



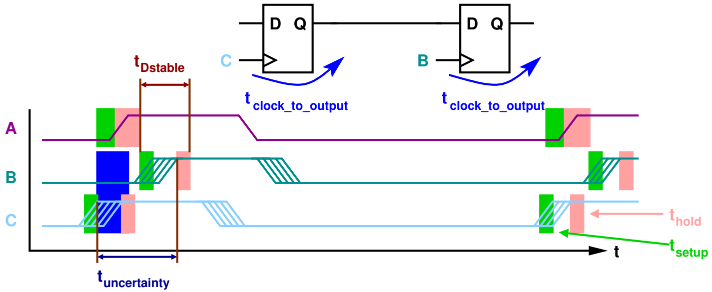
- Remember: a real flipflop has a setup and hold time in which the D-input needs to be kept stable (otherwise the flipflop goes into meta stable state).
- So which kind of situation we now can have in the real-world taking into account the *clock tree*:
  1. The path is too fast (race-condition).
  2. The path is too slow (frequency cannot be met).

Race condition



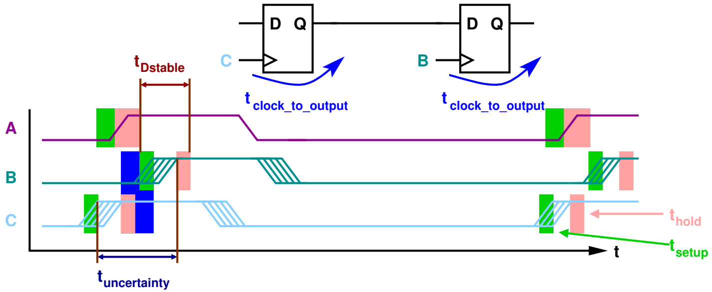
- Putting it all together gives us the above timing diagram.

Race condition



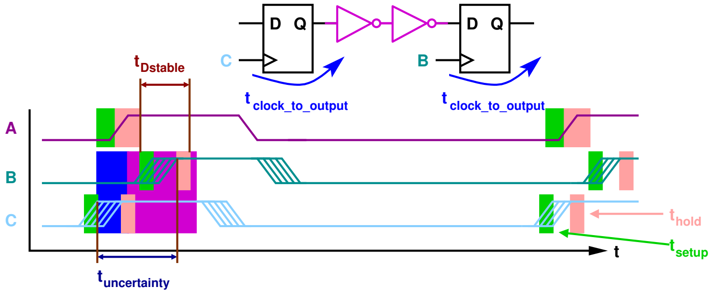
- Putting it all together gives us the above timing diagram.
- Let's take as example a shift-register, there are now two situation that can happen:
  1. The output of flipflop `C` changes before the setup-time of flipflop `B`, hence we have a functional error as the data is too early available!

Race condition



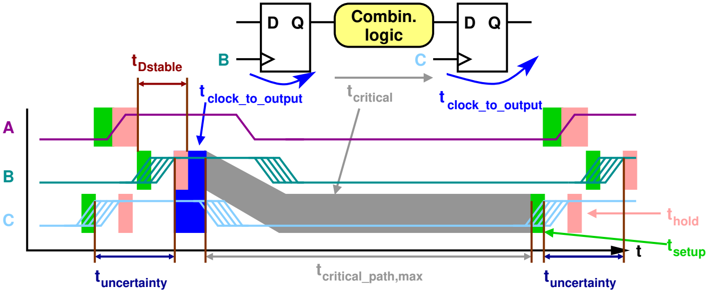
- Putting it all together gives us the above timing diagram.
- Let's take as example a shift-register, there are now two situation that can happen:
  1. The output of flipflop `C` changes before the setup-time of flipflop `B`, hence we have a functional error as the data is too early available!
  2. The output of flipflop `C` changes during t_{Dstable} of flipflop `B` which goes in meta stable state (Note that this situation will always happen independent of the clock frequency!).

Race condition



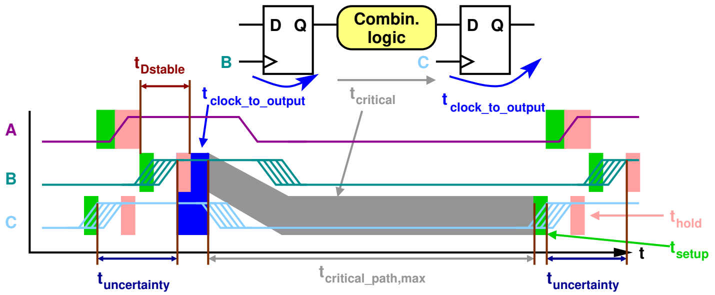
- Putting it all together gives us the above timing diagram.
- Let's take as example a shift-register, there are now two situation that can happen:
  1. The output of flipflop `C` changes before the setup-time of flipflop `B`, hence we have a functional error as the data is too early available!
  2. The output of flipflop `C` changes during t_{Dstable} of flipflop `B` which goes in meta stable state (Note that this situation will always happen independent of the clock frequency!).
- This problem can be solved by inserting a delay between the flipflops `C` and `B`. Fortunately this is done for us by the synthesis and/or P&R-tools.

Timing not met



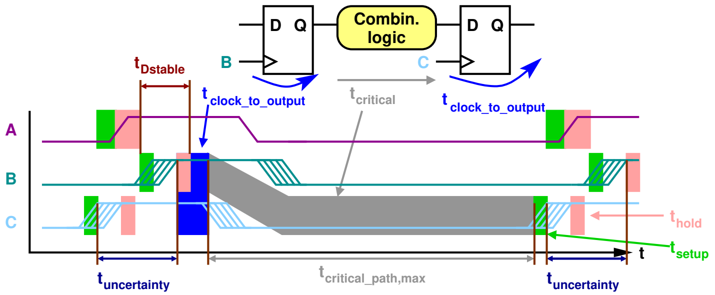
- The other situation is shown above (hence $t_{p,clock} = t_{clock\_to\_output} + t_{critical,max} + t_{setup} + t_{uncertainty}$).
- We know that during the critical path time we may have hazards on the D-input of flipflop C, and that the correct value is available after $t_{critical\_path}$.

Timing not met



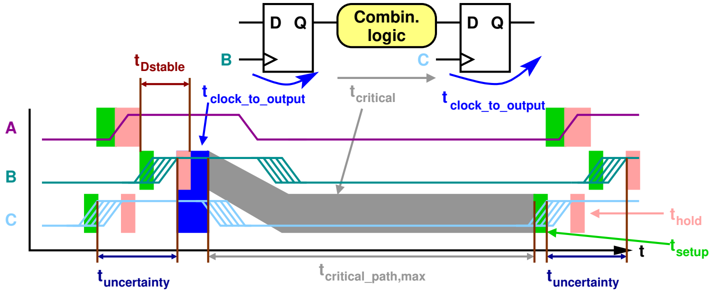
- The other situation is shown above (hence $t_{p,clock} = t_{clock\_to\_output} + t_{critical,max} + t_{setup} + t_{uncertainty}$).
- We know that during the critical path time we may have hazards on the D-input of flipflop C, and that the correct value is available after $t_{critical\_path}$.
- Note that the synthesizer and/or P&R-tool might insert in front of the combinational logic some inverters to prevent flipflop C from going into meta stable state due to $t_{Dstable}$ violation caused by hazards!

Timing not met



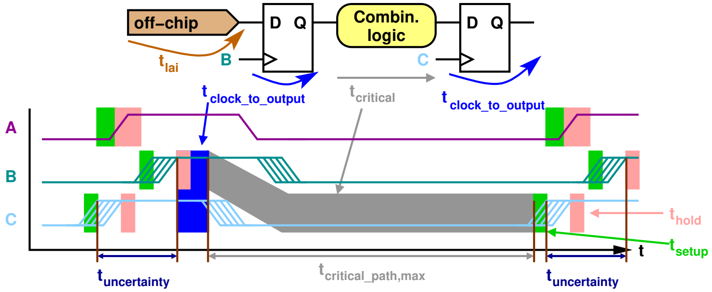
- The other situation is shown above (hence $t_{p,clock} = t_{clock\_to\_output} + t_{critical,max} + t_{setup} + t_{uncertainty}$).
- We know that during the critical path time we may have hazards on the D-input of flipflop C, and that the correct value is available after $t_{critical\_path}$.
- Note that the synthesizer and/or P&R-tool might insert in front of the combinational logic some inverters to prevent flipflop C from going into meta stable state due to $t_{Dstable}$ violation caused by hazards!
- Timing is not met when there exists at least one combinational logic path with a $t_{critical\_path} > t_{critical\_path,max}$.

Timing closure



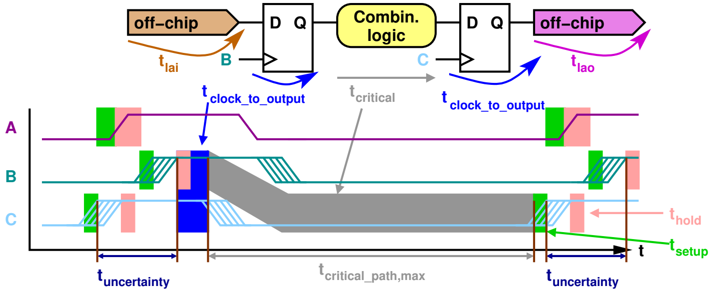
- *Timing closure* is the process of getting all $t_{critical\_paths} < t_{critical\_path,max}$.

Timing closure



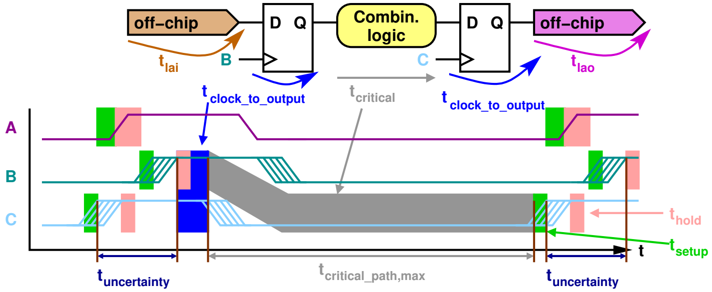
- *Timing closure* is the process of getting all $t_{critical\_paths} < t_{critical\_path,max}$.
- But that's not all, we have two more timings that need attention, namely:
  1. The latest arrival of an external input signal (t_{lai}) to the flipflop with respect to the positive clock edge.

Timing closure



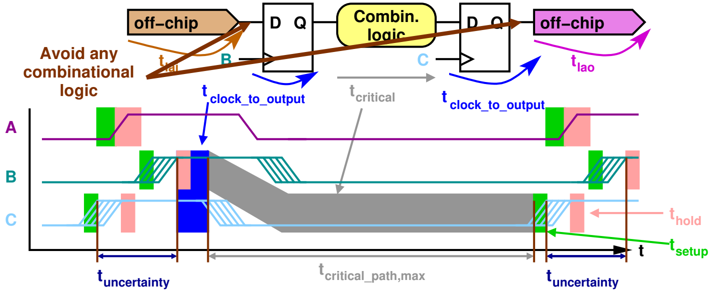
- *Timing closure* is the process of getting all $t_{critical\_paths} < t_{critical\_path,max}$.
- But that's not all, we have two more timings that need attention, namely:
  1. The latest arrival of an external input signal (t_{lai}) to the flipflop with respect to the positive clock edge.
  2. The latest arrival of the signal from a flipflop to the edge of the package (t_{lao}) with respect to the positive clock edge.

Timing closure



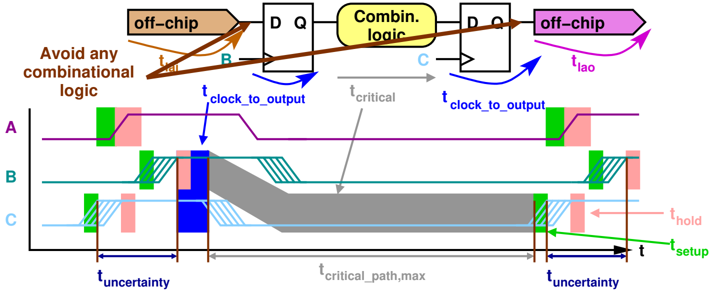
- *Timing closure* is the process of getting all $t_{critical\_paths} < t_{critical\_path,max}$.
- But that's not all, we have two more timings that need attention, namely:
  1. The latest arrival of an external input signal (t_{lai}) to the flipflop with respect to the positive clock edge.
  2. The latest arrival of the signal from a flipflop to the edge of the package (t_{lao}) with respect to the positive clock edge.
- These two numbers depend on the chips connected to this one and are in general more difficult to determine.

Timing closure off-chip



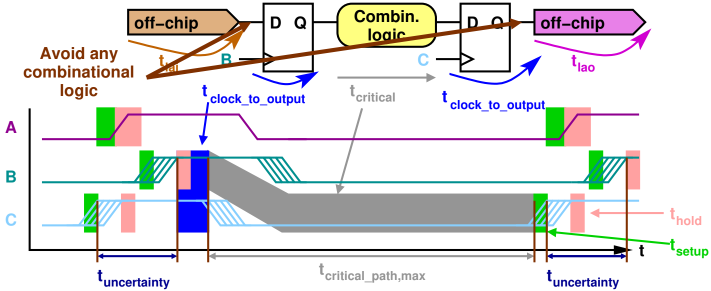
- The later aspect is “easily” solved by not using any combinational logic between the input(s) and the fist flipflop(s) and no combinational logic between the last flipflop(s) and the output(s).
- This has the advantage that you do not have any hazards outside of your chip (good thing!).

Timing closure off-chip



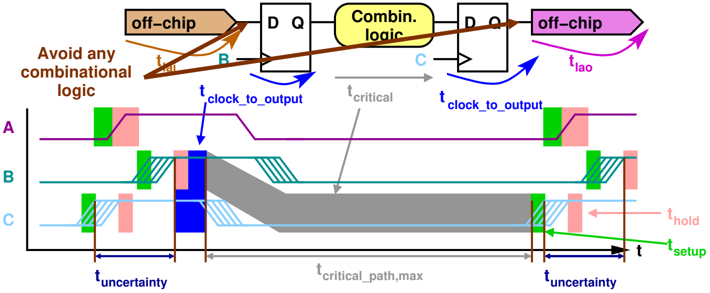
- The later aspect is "easily" solved by not using any combinational logic between the input(s) and the fist flipflop(s) and no combinational logic between the last flipflop(s) and the output(s).
- This has the advantage that you do not have any hazards outside of your chip (good thing!).
- However, this is not always possible, in this case more advanced methods are required like:
  1. Usage of a PLL/DLL to synchronize the attached chip with yours (think of DDR memory).

Timing closure off-chip



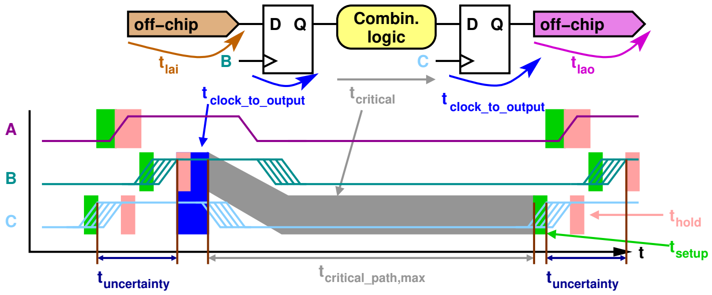
- The later aspect is "easily" solved by not using any combinational logic between the input(s) and the fist flipflop(s) and no combinational logic between the last flipflop(s) and the output(s).
- This has the advantage that you do not have any hazards outside of your chip (good thing!).
- However, this is not always possible, in this case more advanced methods are required like:
  1. Usage of a PLL/DLL to synchronize the attached chip with yours (think of DDR memory).
  2. Adding extra delays in some of the outputs to meet external timings.

Timing closure off-chip



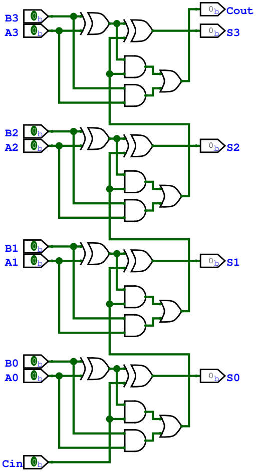
- The later aspect is “easily” solved by not using any combinational logic between the input(s) and the fist flipflop(s) and no combinational logic between the last flipflop(s) and the output(s).
- This has the advantage that you do not have any hazards outside of your chip (good thing!).
- However, this is not always possible, in this case more advanced methods are required like:
  1. Usage of a PLL/DLL to synchronize the attached chip with yours (think of DDR memory).
  2. Adding extra delays in some of the outputs to meet external timings.
- Note: even your internal delays due to the clock-tree may impose problems.....

Timing closure on-chip



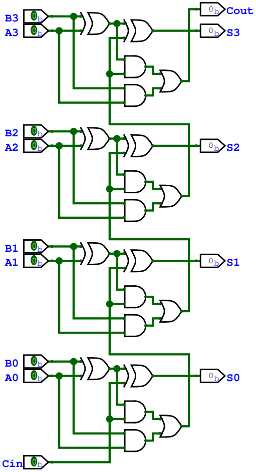
- The on-chip aspect has some methods that you can use, but be aware, the synthesis tool might be more “intelligent” than you are (compare the compiler for a programming language).
- These methods are more for things that the synthesizer does not know about (for example what does your program do):
  - Fined-grain paralyzing.
  - Multi-cycling.
  - Pipelining.

Speeding-up your circuit



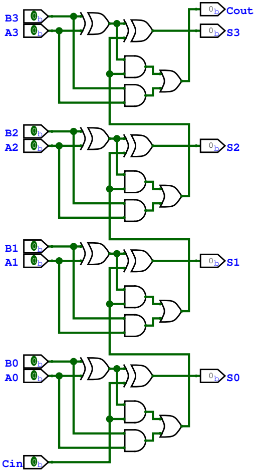
- As example we take a 4-bit *carry-ripple adder (CRA)*.
- Assume that this adder is in the critical path.

Speeding-up your circuit



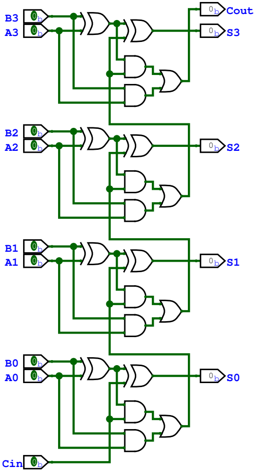
- As example we take a 4-bit *carry-ripple adder (CRA)*.
- Assume that this adder is in the critical path.
- The critical path from this adder goes from `Cin` through the `and`- and `or`-gates up to `Cout`/`S3`.

Speeding-up your circuit



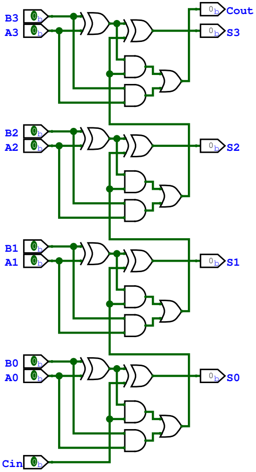
- As example we take a 4-bit *carry-ripple adder (CRA)*.
- Assume that this adder is in the critical path.
- The critical path from this adder goes from `Cin` through the `and`- and `or`-gates up to `Cout/S3`.
- So what can we do to speed-up this circuit, there are basically three methods:
  - Trading-off bigger area/energy consumption against speed.

Speeding-up your circuit



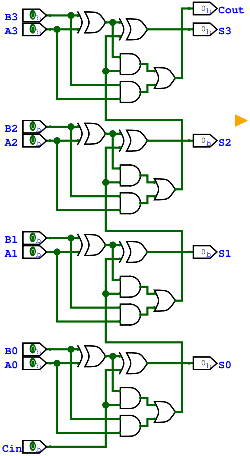
- As example we take a 4-bit *carry-ripple adder (CRA)*.
- Assume that this adder is in the critical path.
- The critical path from this adder goes from `Cin` through the `and`- and `or`-gates up to `Cout`/`S3`.
- So what can we do to speed-up this circuit, there are basically three methods:
  - Trading-off bigger area/energy consumption against speed.
  - Trading-off speed against area/energy consumption.

Speeding-up your circuit

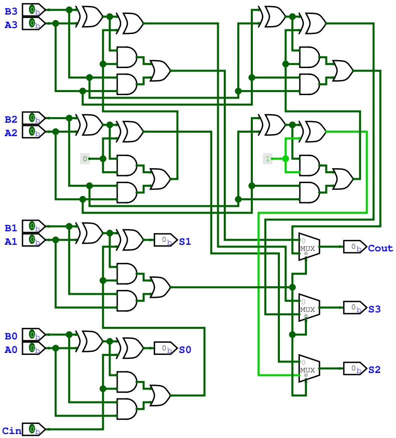


- As example we take a 4-bit *carry-ripple adder (CRA)*.
- Assume that this adder is in the critical path.
- The critical path from this adder goes from `Cin` through the `and`- and `or`-gates up to `Cout`/`S3`.
- So what can we do to speed-up this circuit, there are basically three methods:
  - Trading-off bigger area/energy consumption against speed.
  - Trading-off speed against area/energy consumption.
  - Trading-off latency against speed.

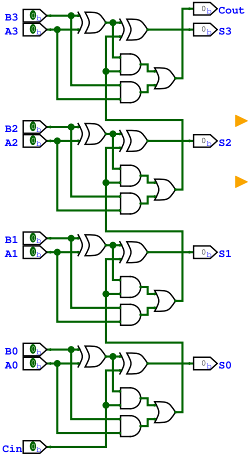
Fined-grain paralyzing



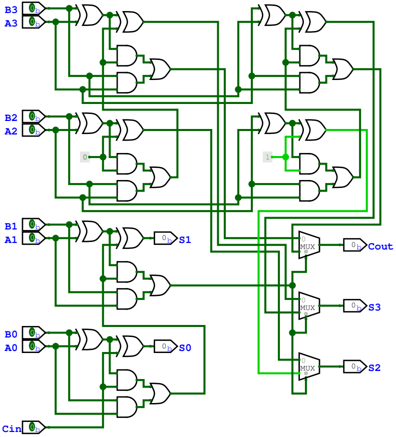
- In this method we cut the circuit (critical path) in 2 (or more) parts.



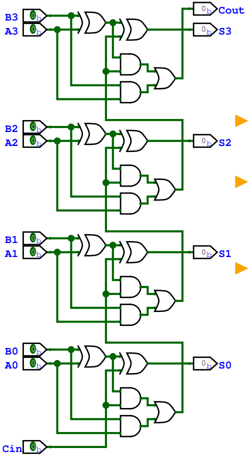
Fined-grain paralyzing



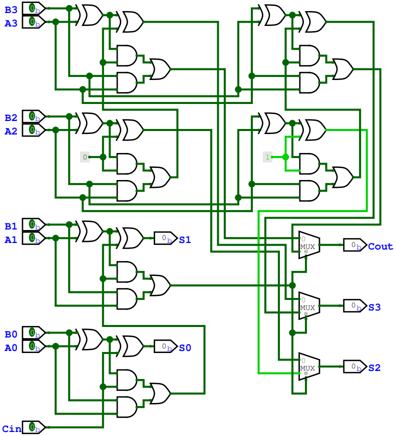
- In this method we cut the circuit (critical path) in 2 (or more) parts.
- The above part is duplicated and calculates the two answers depending the result of the carry.



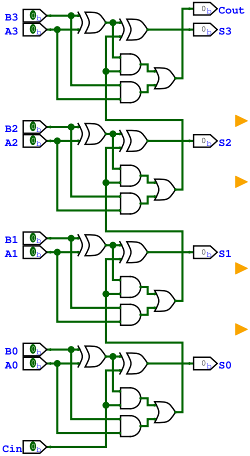
Fined-grain paralyzing



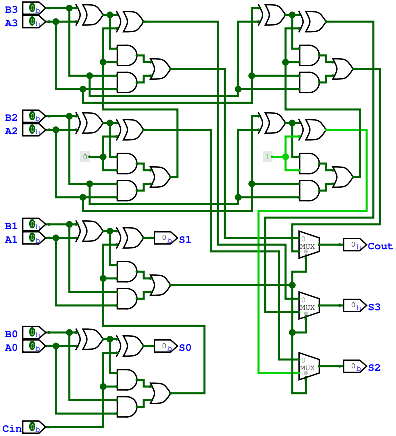
- In this method we cut the circuit (critical path) in 2 (or more) parts.
- The above part is duplicated and calculates the two answers depending the result of the carry.
- Finally the real carry selects the correct result.



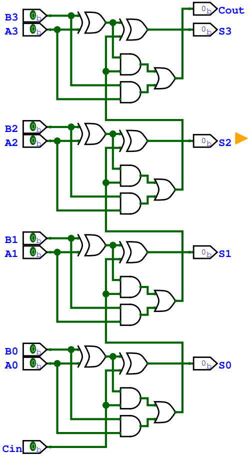
Fined-grain paralyzing



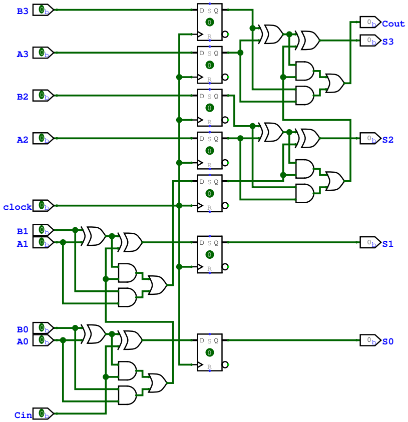
- In this method we cut the circuit (critical path) in 2 (or more) parts.
- The above part is duplicated and calculates the two answers depending the result of the carry.
- Finally the real carry selects the correct result.
- We now have a *carry select adder (CSA)* that is almost twice as fast.



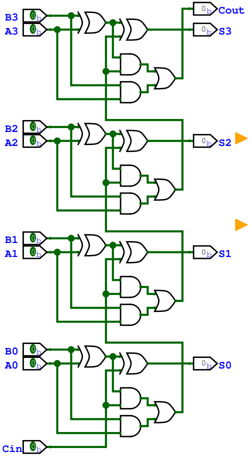
Pipelining



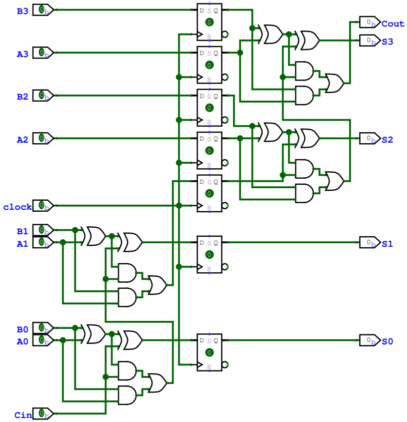
In this method we divide the critical path in 2 (or more) parts and place a row of flipflops between the parts.



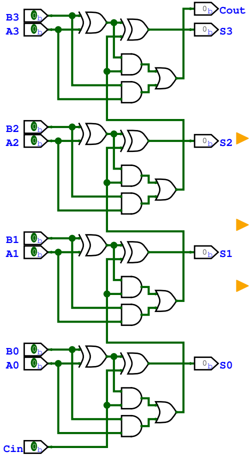
Pipelining



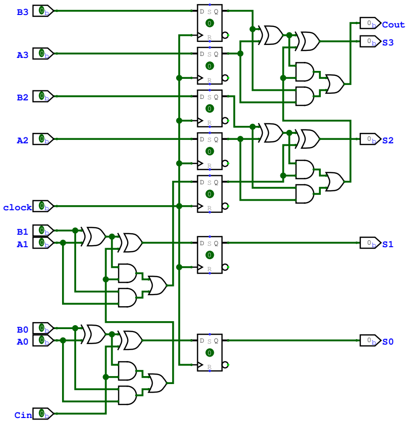
- In this method we divide the critical path in 2 (or more) parts and place a row of flipflops between the parts.
- The advantage is that we can do a calculation each cycle.



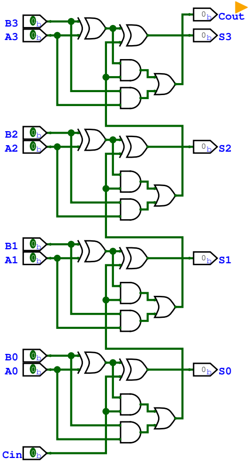
Pipelining



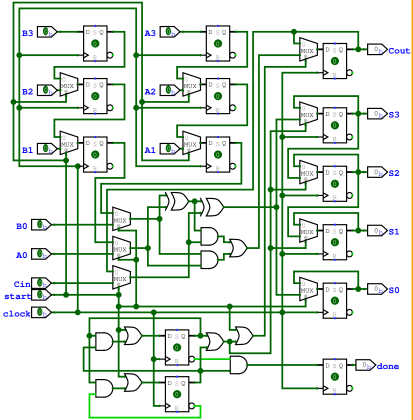
- In this method we divide the critical path in 2 (or more) parts and place a row of flipflops between the parts.
- The advantage is that we can do a calculation each cycle.
- However, we introduce a latency. This could cause problems in case of a feed-back loop.



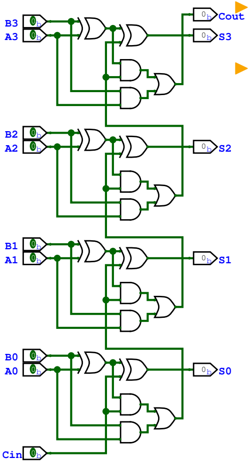
Multi-cycling



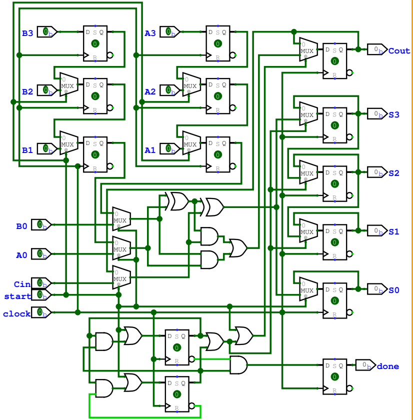
In this method we calculate at each cycle one bit.



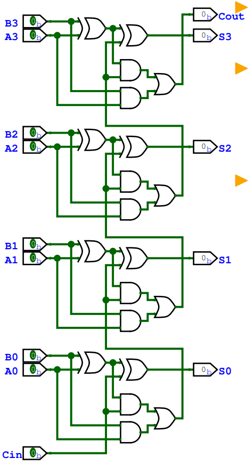
Multi-cycling



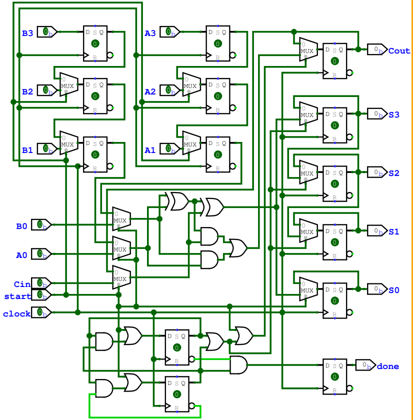
- In this method we calculate at each cycle one bit.
- Of course this has an impact on the performance, as now the addition takes 4 cycles instead of a single cycle.



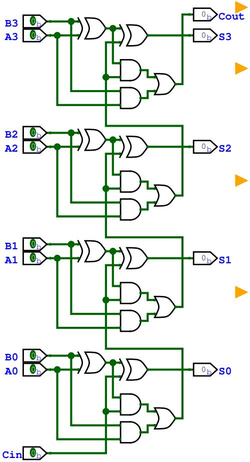
Multi-cycling



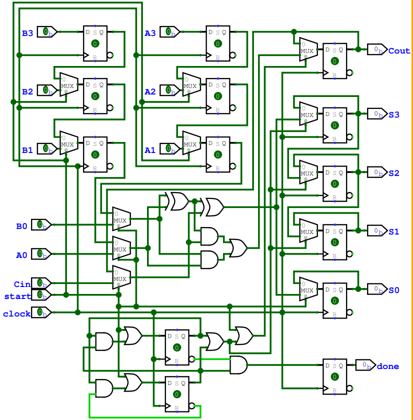
- In this method we calculate at each cycle one bit.
- Of course this has an impact on the performance, as now the addition takes 4 cycles instead of a single cycle.
- But think of the alternative, slowing down all the other functions as we need to reduce the maximum frequency of the CPU.



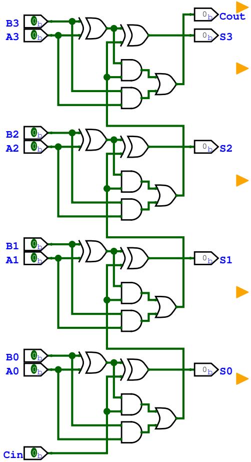
Multi-cycling



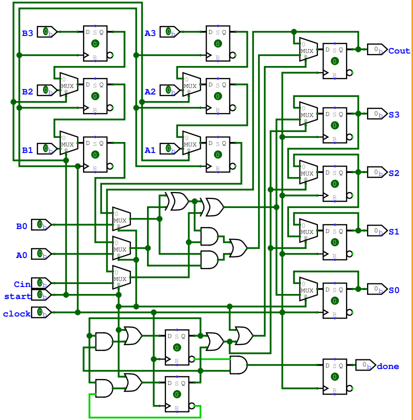
- In this method we calculate at each cycle one bit.
- Of course this has an impact on the performance, as now the addition takes 4 cycles instead of a single cycle.
- But think of the alternative, slowing down all the other functions as we need to reduce the maximum frequency of the CPU.
- Very often we perform a *radix-N* multi-cycle operation where at each cycle N-bits are determined.



Multi-cycling



- In this method we calculate at each cycle one bit.
- Of course this has an impact on the performance, as now the addition takes 4 cycles instead of a single cycle.
- But think of the alternative, slowing down all the other functions as we need to reduce the maximum frequency of the CPU.
- Very often we perform a *radix-N* multi-cycle operation where at each cycle N-bits are determined.
- Of course, when `A` and `B` are guaranteed to be constant between `start` and `done`, we can replace the input shift-registers by a multiplexer.



Conclusion

- We have seen the details that determine the maximum speed with which we can safely operate a circuit.

Conclusion

- We have seen the details that determine the maximum speed with which we can safely operate a circuit.
- We also have visited three methods how to speed-up a critical path.

Conclusion

- We have seen the details that determine the maximum speed with which we can safely operate a circuit.
- We also have visited three methods how to speed-up a critical path.
- Each of these methods makes a trade-off between area, energy consumption, complexity and speed.

Conclusion

- We have seen the details that determine the maximum speed with which we can safely operate a circuit.
- We also have visited three methods how to speed-up a critical path.
- Each of these methods makes a trade-off between area, energy consumption, complexity and speed.
- It depends on the requirements which of these methods can be applied to a given hot-spot.