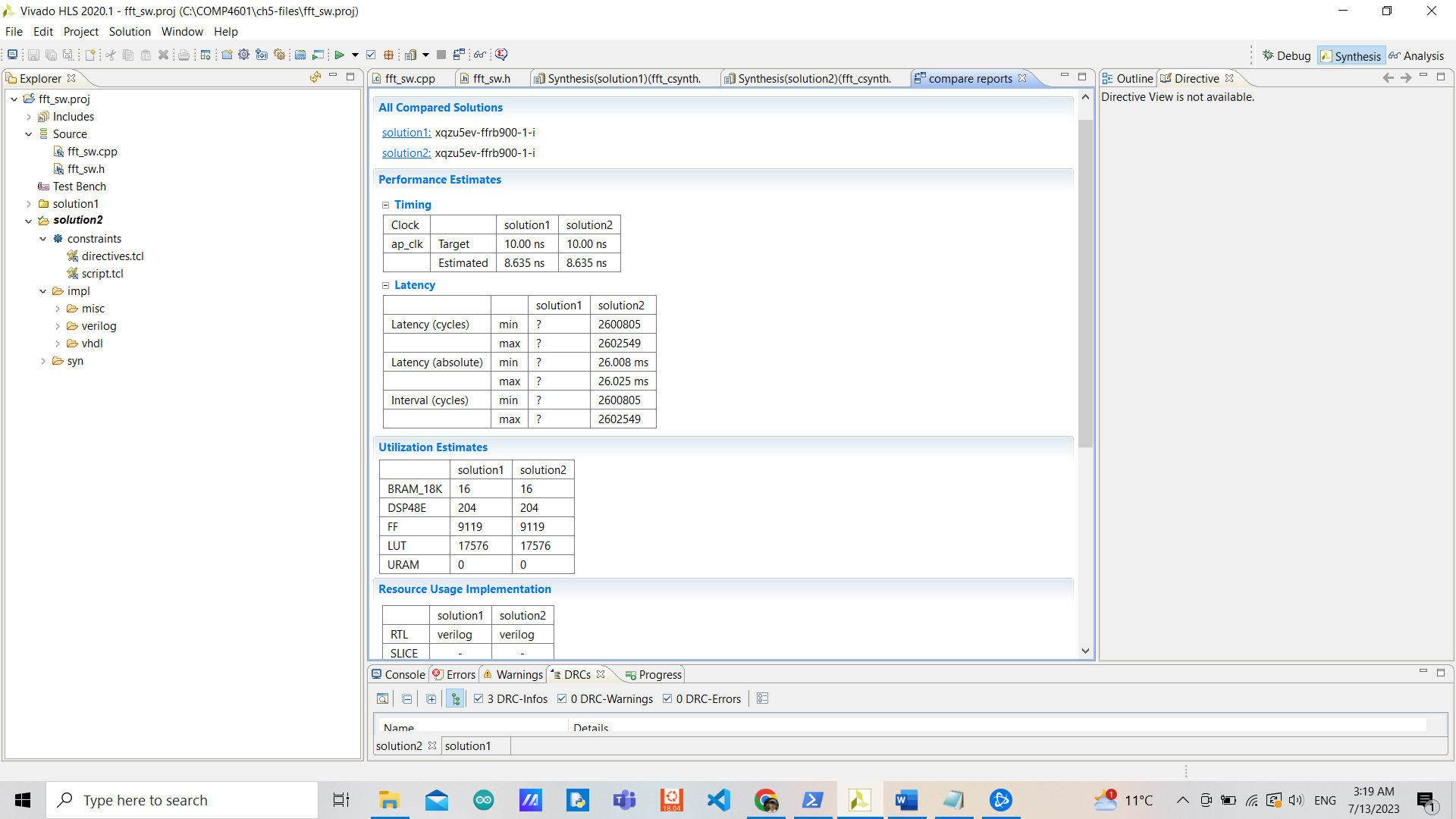
Mr. Phot Koseekrainiramon z5387411

23T2 COMP4601 Chapter 5 Lab/Week 8 Hand-in exercises

FFT implementation exercises



**Exercise 1:**

The latency/interval is not determined due to having a number of iterations depending on variables numBF and DFTpts. However, they are due to nested loops. The inner loop called dft\_loop: has a latency of 21 clock cycles. Should be having up to 512 iterations. The middle loop called butterfly\_loop: should be having up to 512 iterations depending on stages and it takes 5 clock cycles before entering dft\_loop. And the outer loop: stage\_loop: has 10 iterations (number of stages) and a latency of 28 clock cycles before entering butterfly\_loop.

In terms of utilization, this design uses 16 BRAMs (5%), 204 DSPs (16%), 9119 FFs (3%), and 17576 LUTs (15%). We have 10 stages (for a 1024-point FFT, with 512 butterfly operations per stage).

**Exercise 2:**

The estimated latency should be around ((((21\*512) + 5)\*24) + 28)\*10 = 2581960 total clock cycles from having 512 trip counts for dft\_loop, 24 trip counts for butterfly\_loop, and 10 trip counts for stage\_loop. This is because butterfly\_loop has iteration in stage. With 10 stages we achieve a median of 24, and in every stage, the body of dft\_loop is executed the same number of times in total which is 512.

What we achieved from the table above, is a minimum latency of 2600805 cycles and a maximum latency of 2602549 cycles, giving the difference between them only 0.067% difference. The resource utilization hasn’t changed since adding the directives. They are performing using the same logic, just by having an estimated number of iterations.

**Exercise 3:**

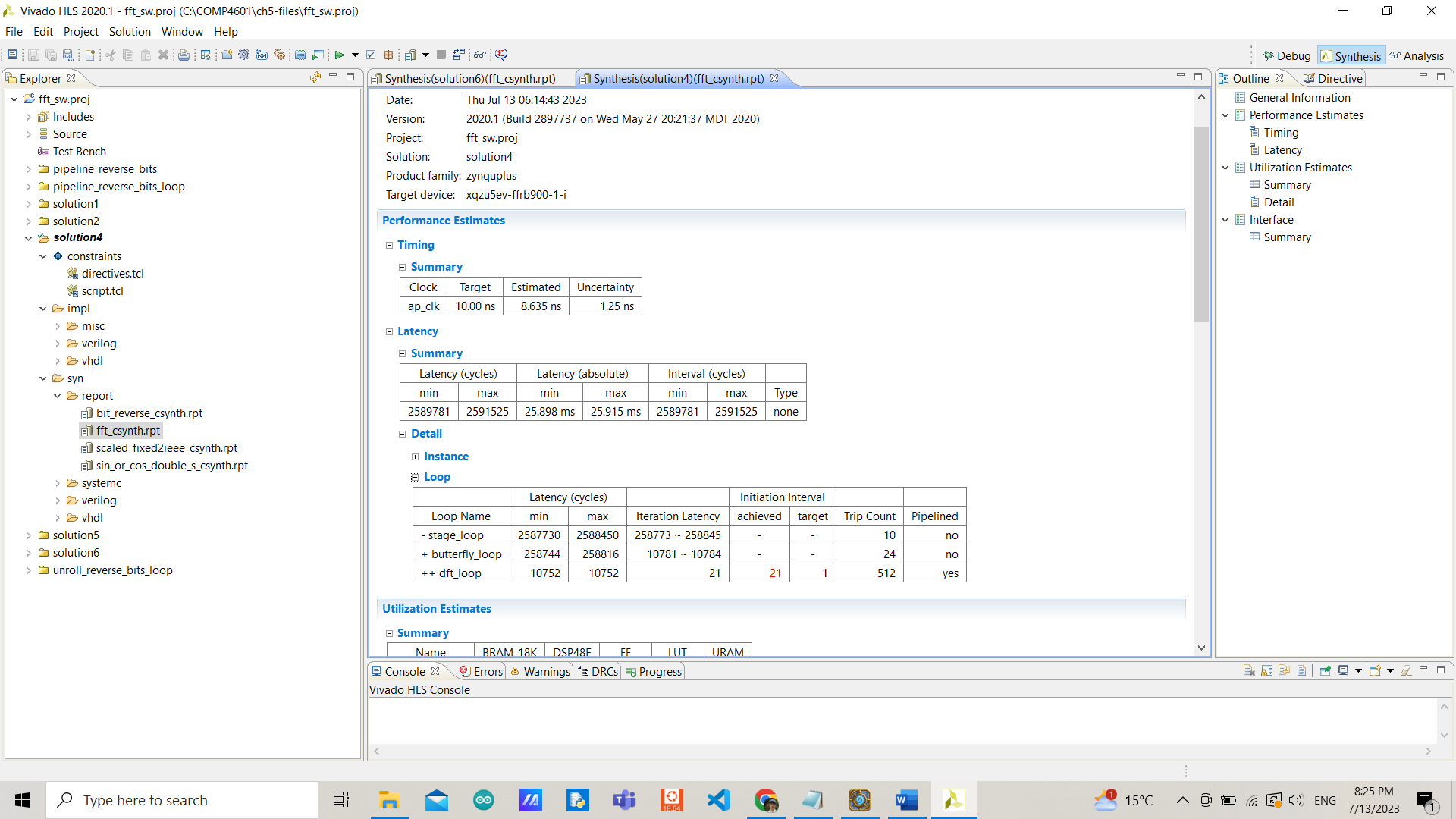
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | origin | Unrolling reverse\_bits\_loop | Pipelining reverse\_bits\_loop | Pipelining reverse\_bits |
| Clock Target (ns) | 10 | 10 | 10 | 10 |
| Clock Estimated (ns) | 2.704 | 2.704 | 2.704 | 2.704 |
| Uncertainty (ns) | 1.25 | 1.25 | 1.25 | 1.25 |
| Latency min (cycles) | 13313 | 2049 | 14337 | 2049 |
| Latency max (cycles) | 14337 | 3073 | 15361 | 3073 |
| Latency min (ms) | 0.133 | 0.020 | 0.143 | 0.020 |
| Latency max (ms) | 0.143 | 0.030 | 0.154 | 0.030 |
| Interval min (cycles) | 13313 | 2049 | 14337 | 2049 |
| Interval max (cycles) | 14337 | 3073 | 15361 | 3073 |
| BRAMs | 0 | 0 | 0 | 0 |
| DSPs | 0 | 0 | 0 | 0 |
| FFs | 105 | 47 | 106 | 47 |
| LUTs | 183 | 122 | 188 | 122 |
| URAMs | 0 | 0 | 0 | 0 |

Unrolling the reverse\_bits\_loop is the best in terms of both performance and utilization. This is due to the fact that the function itself doesn’t need logic resources and could be done with only wires, and Vidavo HLS could notify that when trying to unroll the loop (normally it would require more resources when unrolling, however, this is not the case). The same goes with pipelining the function which would automatically unroll the loop.

By pipelining the reverse\_bits\_loop, even achieving the initiation interval of 1, this makes it even worse than the original design since the bit\_reverse\_loop needs to waste 1 clock cycle entering the pipeline without parallel loading input arrays X\_R and X\_I and even requires extra logic resources.

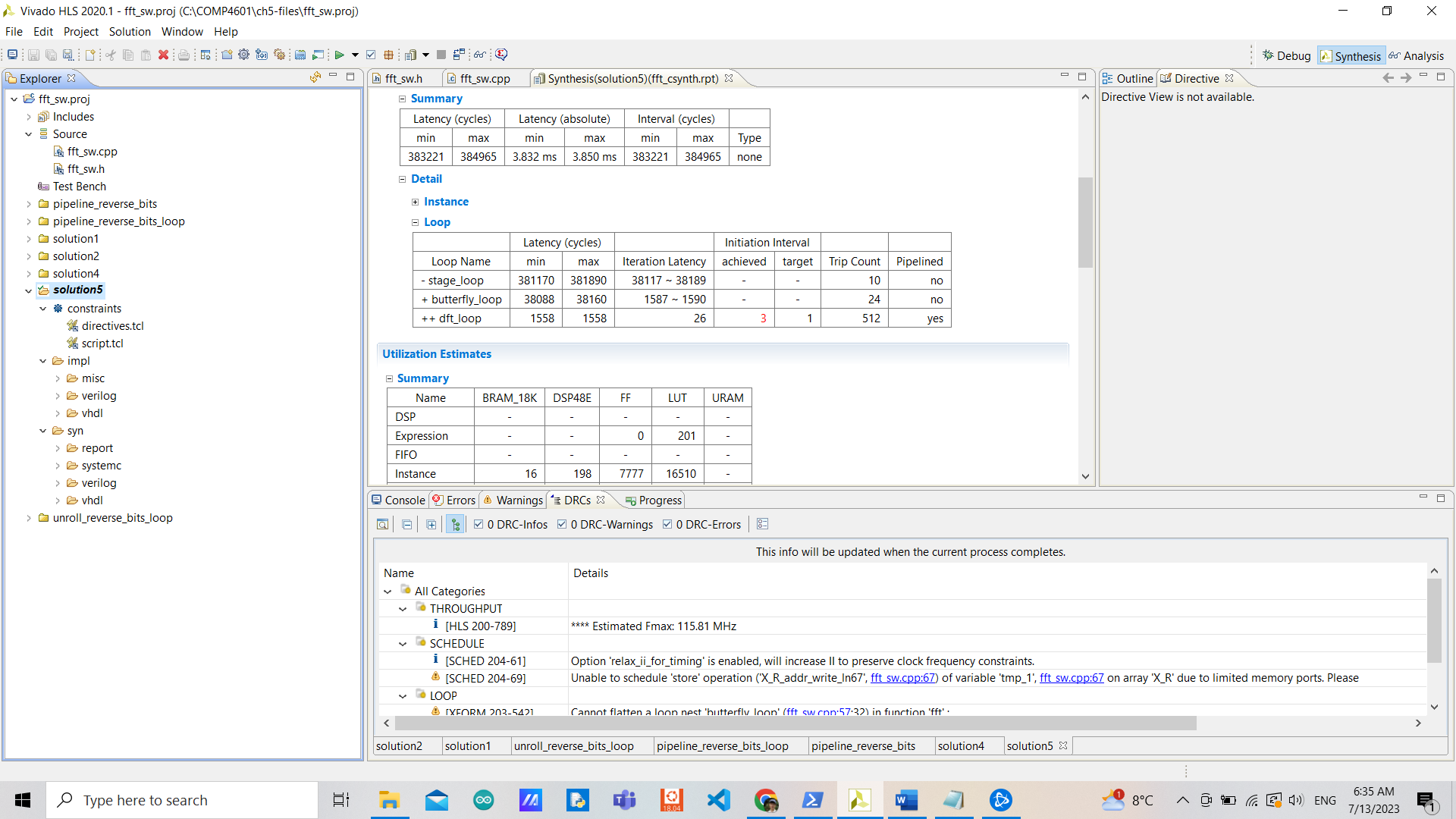
If we are focusing on adding directives to reverse\_bits\_loop then there’re no better ways than unrolling as mentioned that Vivado HLS modifies it such there’s no need for logic resources and could be done in no time. However, there might be a way to unroll or pipeline the for loop in the bit\_reverse function if we could make sure that there’re no dependencies.

**Exercise 4:**



For arrays X\_I and X\_R, they need to be written at the end of the iteration (21 clock cycles) before getting read at the start of the next iteration, this is clearly a false dependency (Read-After-Write). When indexes i and i\_lower could be intersecting. Currently achieving an initiation interval of 21 due to this issue.

**Exercise 5:**



The minimum II I achieved is 3 with an iteration latency of 26 clock cycles by directing inter (meaning the dependency is between different iterations as mentioned in Exercise 4) for arrays X\_I and X\_R with FALSE specified (allowing Vivado HLS to operate in parallel when unrolled).

Unrolling the outer loop wouldn’t do any good. Furthermore, it is impossible to pipeline stage\_loop as long as we couldn’t unroll the inner loops. In order to do that, we would need to modify many things with no guarantee of a better result. So, I would say it is not worthwhile trying to do so.

**Exercise 6:**

directives.tcl:

set\_directive\_loop\_tripcount -min 512 -max 512 -avg 512 "fft/dft\_loop"

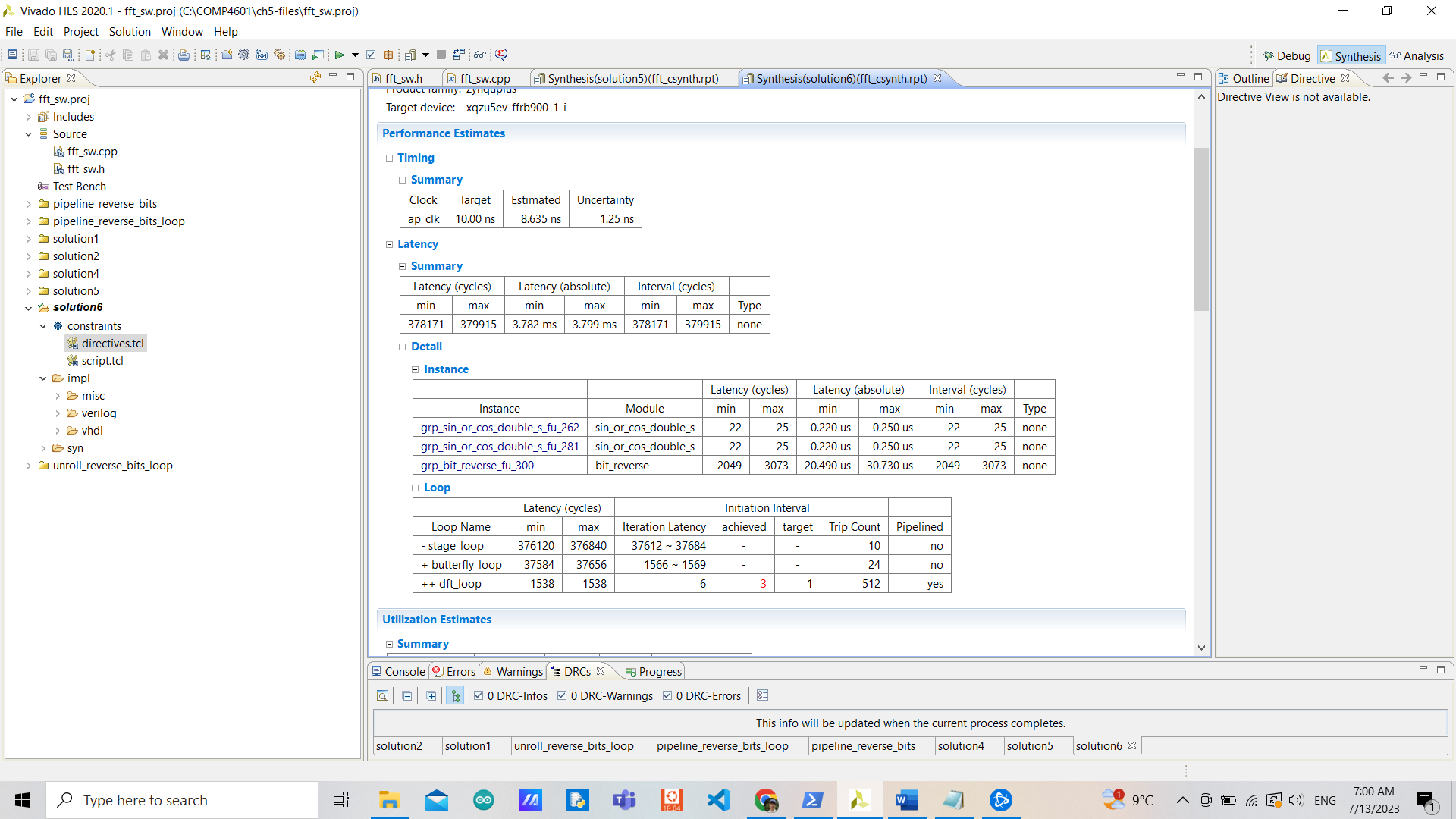
set\_directive\_loop\_tripcount -min 24 -max 24 -avg 24 "fft/butterfly\_loop"

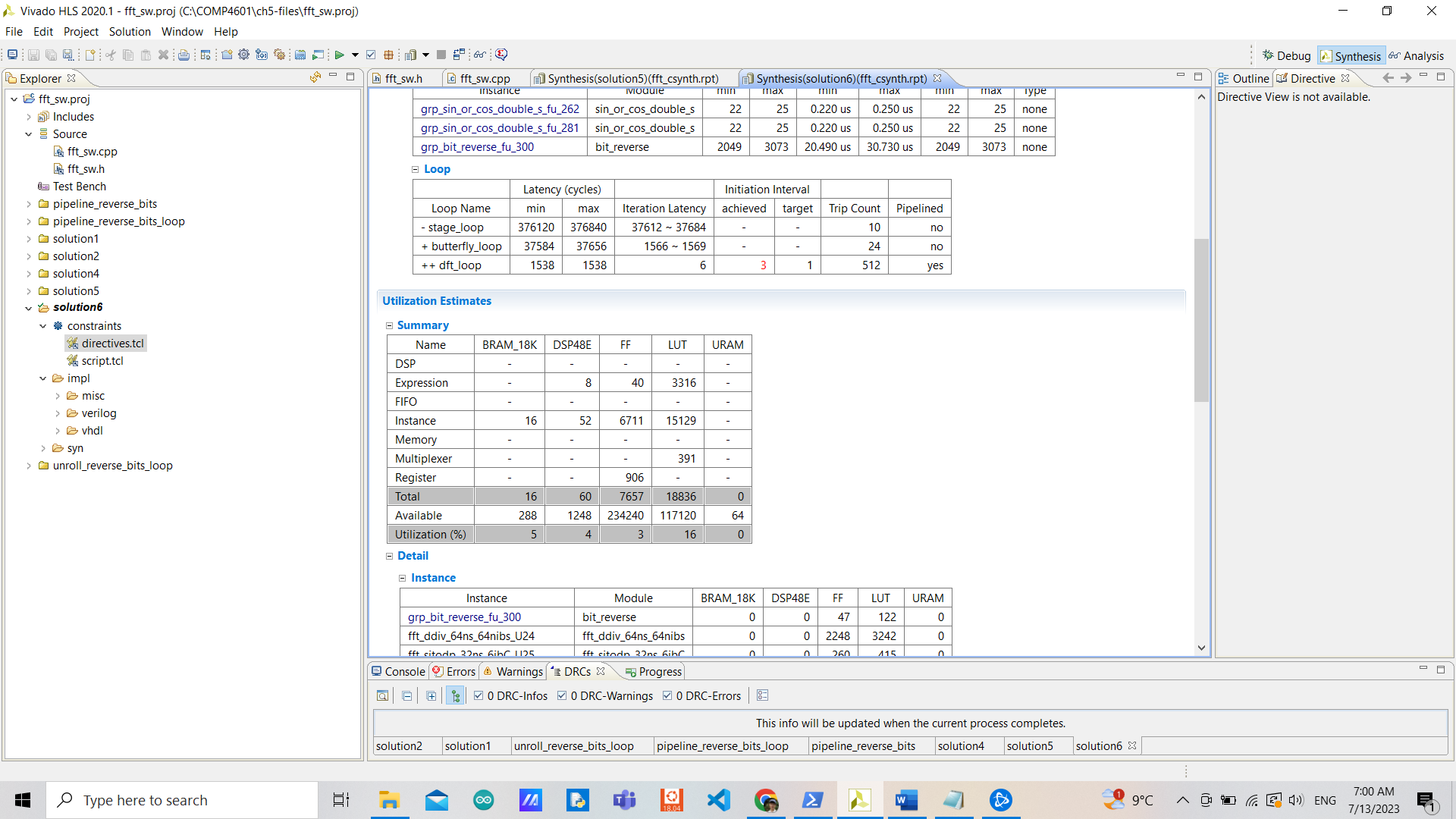
set\_directive\_unroll "reverse\_bits/reverse\_bits\_loop"

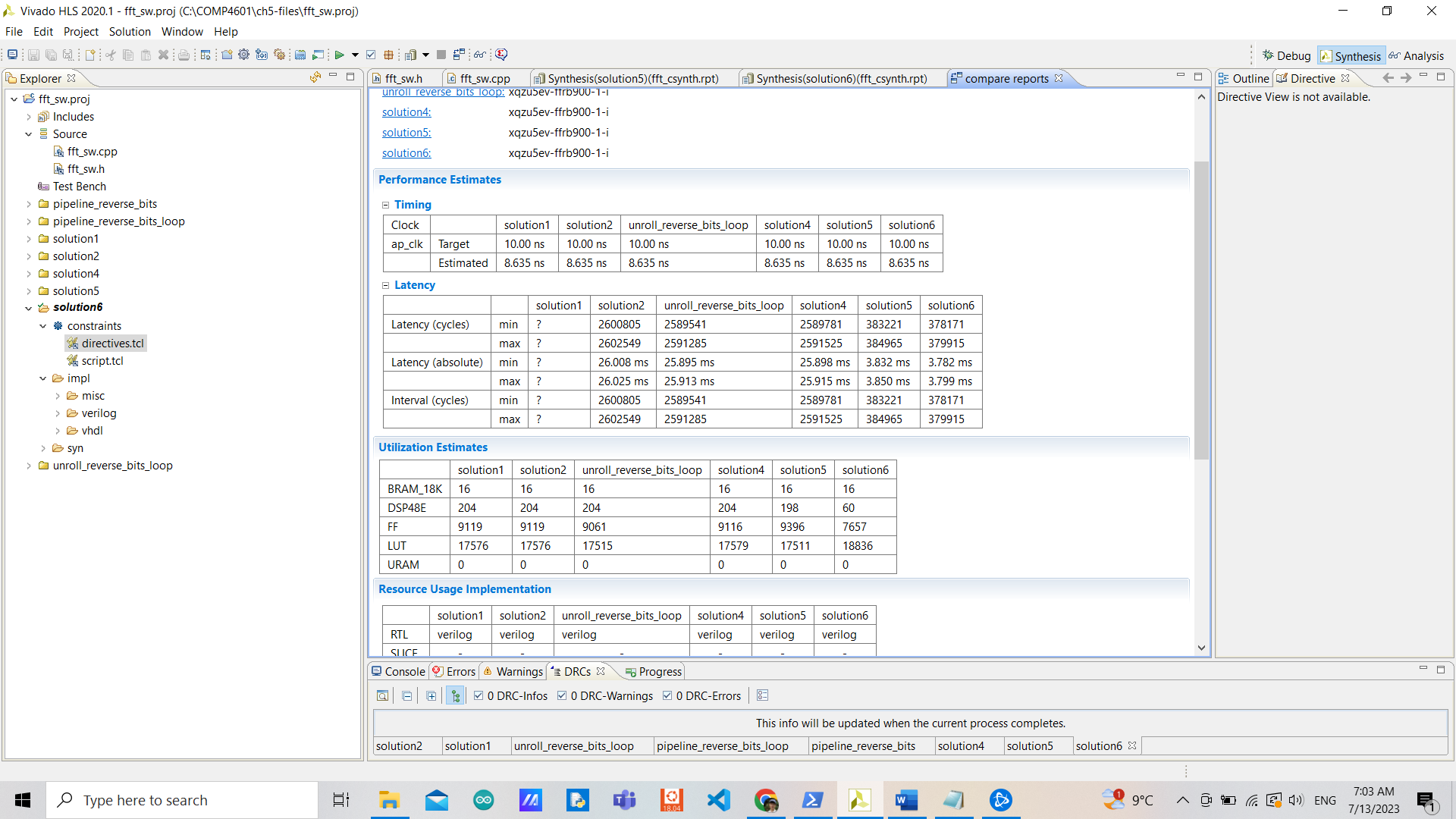
set\_directive\_pipeline "fft/dft\_loop"

set\_directive\_dependence -variable X\_R -type inter -dependent false "fft/dft\_loop"

set\_directive\_dependence -variable X\_I -type inter -dependent false "fft/dft\_loop"





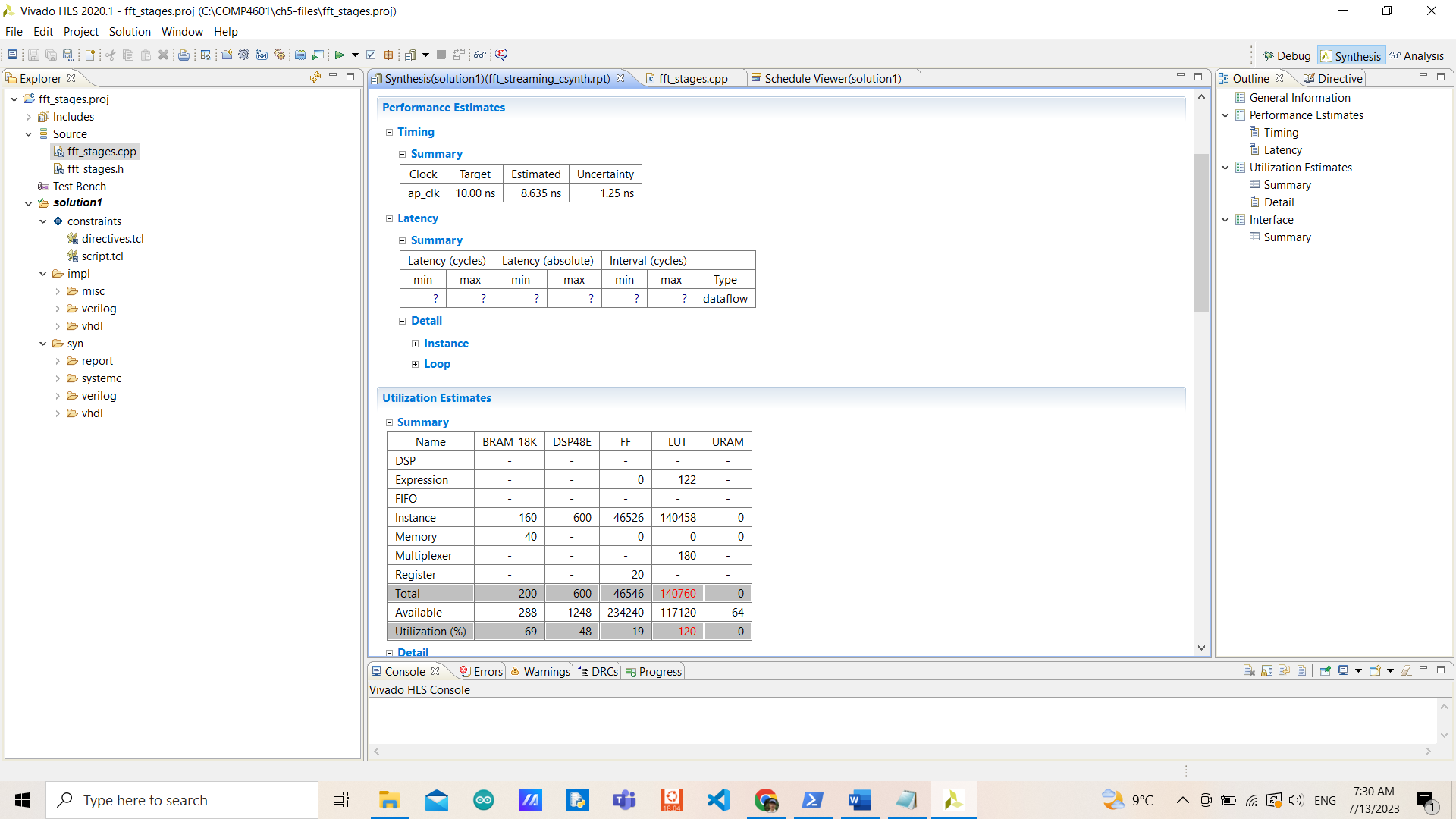


Solution3 is better in terms of both performance and utilization compared to solution1 as explained in Exercise 3. Even though solution4 requires fewer resources, it is no faster than solution3 in not achieving a fewer II due to false dependency but overall, still better than the original design. Solution5 is the best among all designs using floating type after fixing the dependency, with slightly more FFs than solution3 but fewer DSP from achieving II of 3.

It is clear that solution6 performs the best with fewer II (3) and iteration latency of inner loops (only 6 for dft\_loop) compared to solution5 (using float). This greatly reduces the usage of DSPs by more than twice compared to other designs and fewer FFs, however, with an increased number of LUTs as a trade-off.

Implementing the dataflow code in hardware

**Exercise 1:**



The behavior of iterations is similar to the first exercise of FFT but without stage\_loop. The inner loop called dft\_loop: has a latency of 2 clock cycles. The outer loop called butterfly\_loop: takes 4 clock cycles before entering dft\_loop. Those are called as a function, inside fft\_streaming after bit\_reverse is called, by a number of stages instead of looping with an outer loop with the number of stages as the number of iterations. Now that the number of stages isn’t fixed, each time this function is called would have a different number of iterations according to the input stage (when being called in data flow).

In terms of utilization, this design uses 200 BRAMs (69%), 600 DSPs (48%), 46546 FFs (19%), and 140760 LUTs (120%). Which is exceeding the number of LUTs available (impossible to be implemented).

**Exercise 2:**

derective.tcl:

set\_directive\_loop\_tripcount -min 512 -max 512 -avg 512 "fft\_stage/dft\_loop"

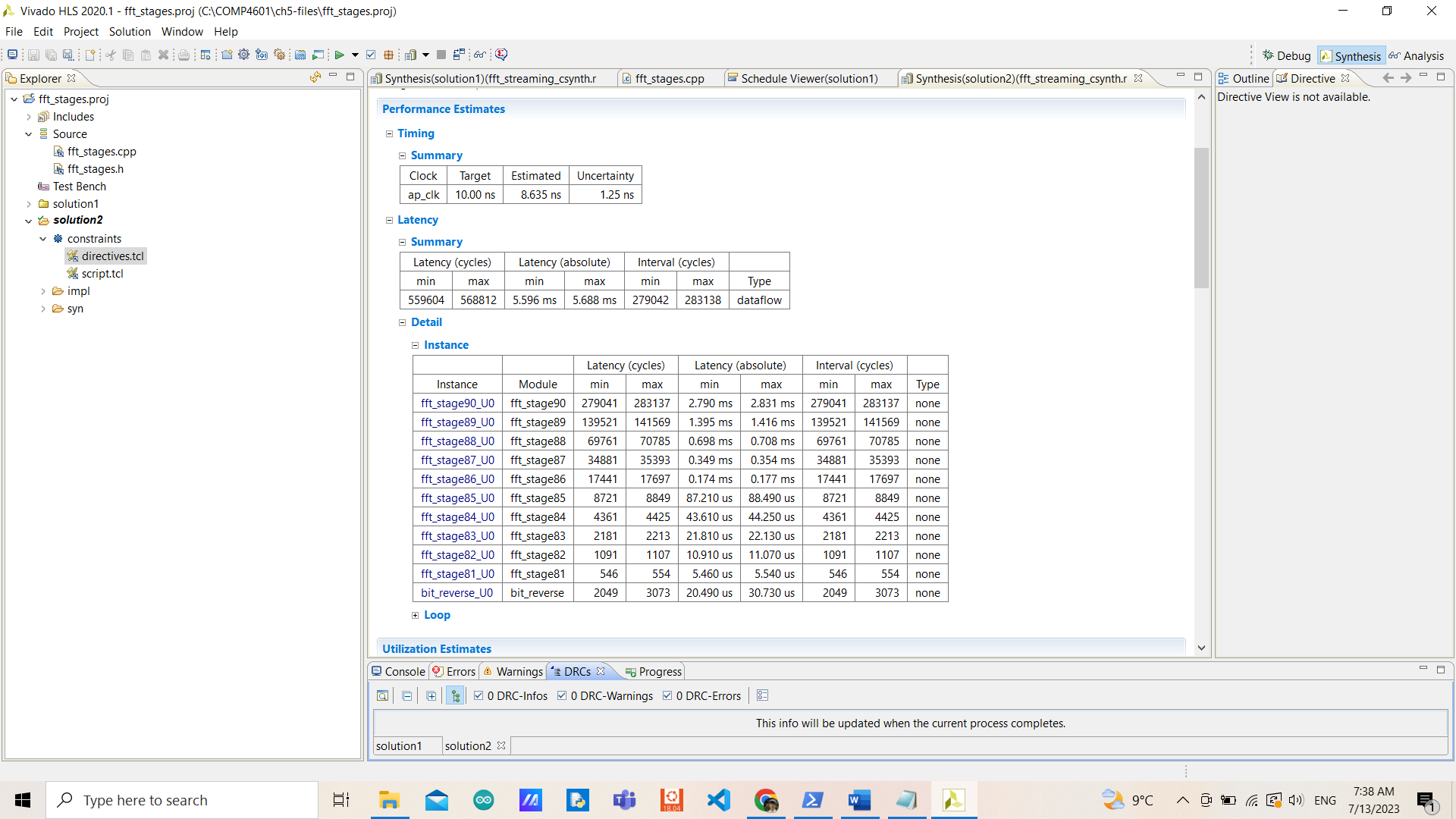
set\_directive\_pipeline "fft\_stage/dft\_loop"

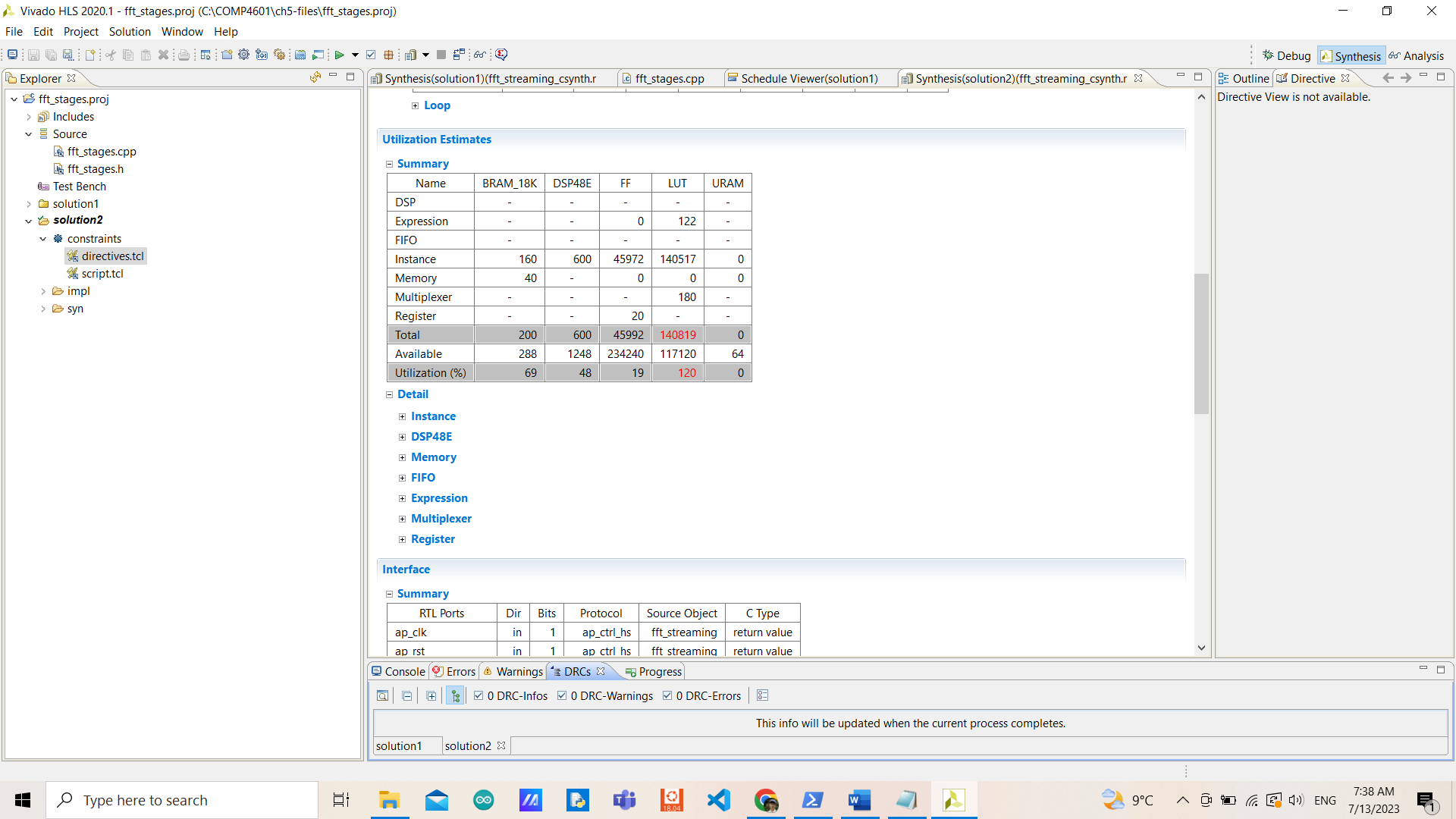
set\_directive\_dependence -variable X\_I -type inter -dependent false "fft\_stage/dft\_loop"

set\_directive\_dependence -variable X\_R -type inter -dependent false "fft\_stage/dft\_loop"

set\_directive\_loop\_tripcount -min 24 -max 24 -avg 24 "fft\_stage/butterfly\_loop"

set\_directive\_unroll "reverse\_bits/reverse\_bits\_loop"





|  |  |  |
| --- | --- | --- |
|  | fft\_sw/solution6 | fft\_stages/solution2 |
| Clock Target (ns) | 10 | 10 |
| Clock Estimated (ns) | 8.635 | 8.635 |
| Uncertainty (ns) | 1.25 | 1.25 |
| Latency min (cycles) | 378171 | 559604 |
| Latency max (cycles) | 379915 | 568812 |
| Latency min (ms) | 3.782 | 5.596 |
| Latency max (ms) | 3.799 | 5.688 |
| Interval min (cycles) | 378171 | 279042 |
| Interval max (cycles) | 379915 | 283138 |
| BRAMs | 16 | 200 |
| DSPs | 60 | 600 |
| FFs | 7656 | 45992 |
| LUTs | 18836 | 140819 |
| URAMs | 0 | 0 |

It is clear that the new design is no better than the old one in any terms, moreover, it is so demanding in resources that the number of LUTs used is still exceeding those available. With a greatly increased total latency and around 10 times more resources compared to the old design (due to the fact that each function stage needs to be implemented independently). But now dft\_loop of the new design achieved an initiation interval of 1 which is fewer (than 3) but the gap between minimum and maximum latency is also increased.

Implementing the streaming code in hardware

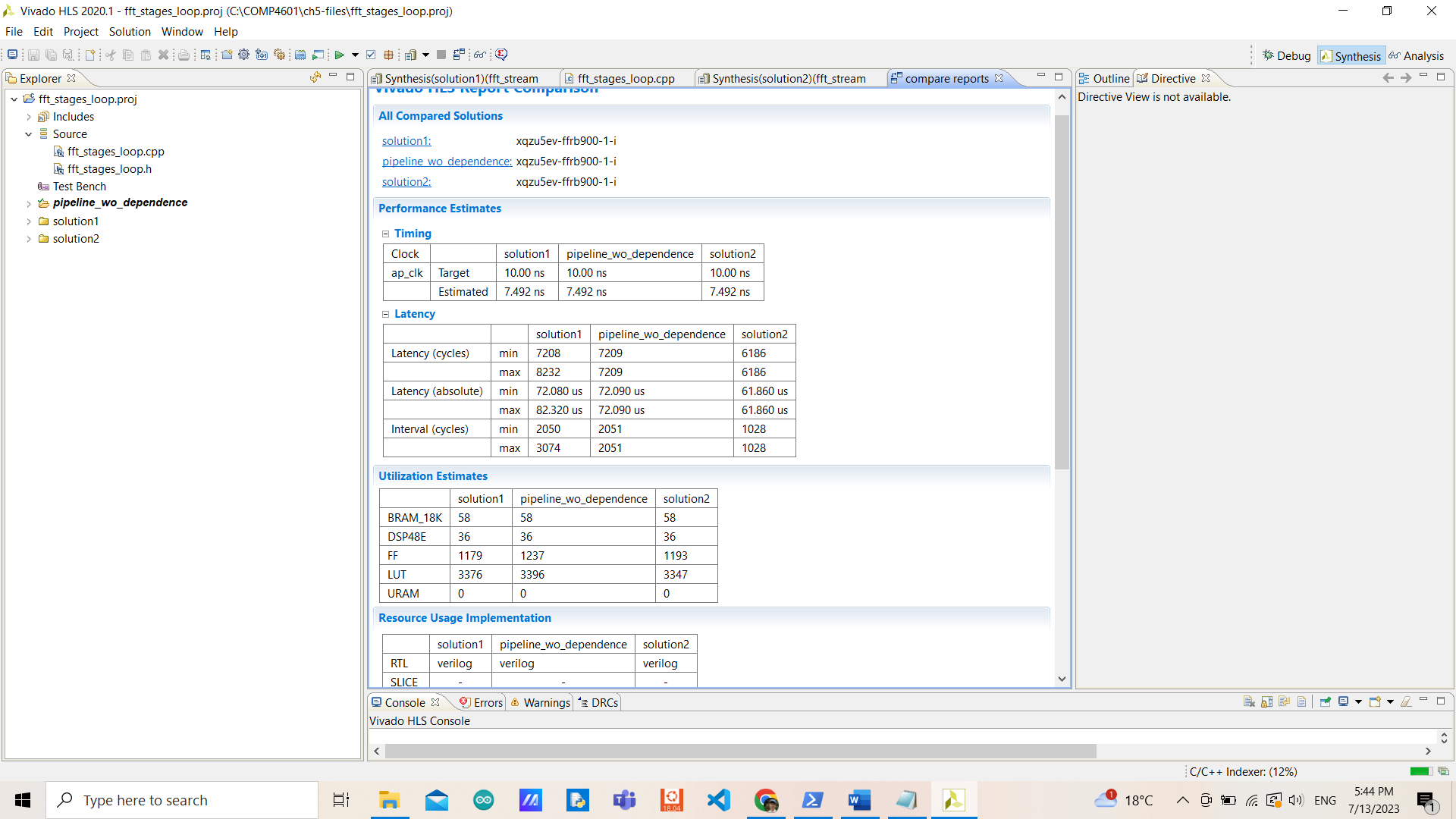
**Exercise 1:**

|  |  |  |
| --- | --- | --- |
|  | fft\_stages/solution2 | fft\_stages\_loop/solution1 |
| Clock Target (ns) | 10 | 10 |
| Clock Estimated (ns) | 8.635 | 7.492 |
| Uncertainty (ns) | 1.25 | 1.25 |
| Latency min (cycles) | 559604 | 7208 |
| Latency max (cycles) | 568812 | 9232 |
| Latency min (ms) | 5.596 | 0.072 |
| Latency max (ms) | 5.688 | 0.082 |
| Interval min (cycles) | 279042 | 2050 |
| Interval max (cycles) | 283138 | 3074 |
| BRAMs | 200 | 58 |
| DSPs | 600 | 36 |
| FFs | 45992 | 1179 |
| LUTs | 140819 | 3376 |
| URAMs | 0 | 0 |

The latency for the new design becomes 7208-9232 clock cycles with 2050-3075 clock cycle intervals. The new design is better in both terms of performance and utilization with even fewer estimated clock periods. And the utilization is not exceeding that available on the board anymore.

The new design has butterfly\_loop pipelined with an initiation interval of only 1.

**Exercise 2:**



If only pipelining without determining a false dependency, the initiation interval of the bit\_reverse\_loop wouldn’t be able to achieve 1 (only achieving 2) due to that issue (similar to Exercise 4). This causes the task latency and interval to be no better than the old design (solution1) since the new interval is equal to the minimum old interval.

After directing dependence to arrays OUT\_I and OUT\_R with false inters, it was able to achieve an initiation interval of 1 and improve overall performance with a slightly different number of FFs and LUTs usages.

|  |  |  |
| --- | --- | --- |
|  | fft\_sw/solution2 | fft\_stages\_loop/solution2 |
| Clock Target (ns) | 10 | 10 |
| Clock Estimated (ns) | 8.635 | 7.492 |
| Uncertainty (ns) | 1.25 | 1.25 |
| Latency min (cycles) | 2600805 | 6186 |
| Latency max (cycles) | 2602549 | 6186 |
| Latency min (ms) | 26.008 | 0.061 |
| Latency max (ms) | 26.025 | 0.061 |
| Interval min (cycles) | 2600805 | 1028 |
| Interval max (cycles) | 2602549 | 1028 |
| BRAMs | 16 | 58 |
| DSPs | 204 | 36 |
| FFs | 9119 | 1193 |
| LUTs | 17576 | 3347 |
| URAMs | 0 | 0 |

It is clear that dataflow outperforms in terms of both performance and utilization, with no differences between min-max latency and fewer estimated clock periods. Even though it requires fewer DSPs, FFs, and LUTs, it requires a significantly increased number of BRAMs usages that could be concerned with the limited number of resources available on board (not exceeding but high demands).

With enough resources from a processor, this design should be able to be implemented. Note, it is required to know what could be done in isolation and what couldn’t.

However, we are not taking accuracy into account even though we are using 2 different data types here, this would be one thing that we need to check for further improvement.

directives.tcl:

set\_directive\_pipeline "bit\_reverse/bit\_reverse\_loop"

set\_directive\_dependence -variable OUT\_I -type inter -dependent false "bit\_reverse/bit\_reverse\_loop"

set\_directive\_dependence -variable OUT\_R -type inter -dependent false "bit\_reverse/bit\_reverse\_loop"

