Report for Synthesis of Digital Circuits Project 1

Daniel Pfister

April 17, 2024

1 Task 2: ASAP Scheduling

1. In the case of no resource constraints, the obtained latency of a schedule using As Soon As Possible (ASAP) scheduling is always optimal
2. The scheduling problem should be solved individually for each BB because the operations within the same BB are not affected by conditionals and loops and as such can be schedule together.

2 Task 3: ALAP Scheduling

1. The achieved latency using ALAP scheduling is the same as using ASAP scheduling. This is the case because the supersinks, which are the operations executed last, are scheduled at the same time as they were using ASAP scheduling.
2. The slack can be observed by comparing the start times of the operations using ASAP and ALAP scheduling.

   Scheduling Variables using ASAP scheduling:

   ```
   sv(_b) @ bb(0) := 0.0
   sv(_a) @ bb(0) := 0.0
   sv(_n) @ bb(0) := 0.0
   sv(_c) @ bb(0) := 0.0
   sv(ssrc_0) @ bb(0) := 0.0
   sv(_add) @ bb(0) := 1.0
   sv(_mul2) @ bb(0) := 1.0
   sv(_sub) @ bb(0) := 11.0
   sv(_cmp) @ bb(0) := 12.0
   sv(br_entry) @ bb(0) := 13.0
   sv(ssink_0) @ bb(0) := 13.0
   sv(_mul) @ bb(0) := 2.0
   sv(_div) @ bb(0) := 5.0
   sv(_shr) @ bb(0) := 6.0
   sv(_rem) @ bb(0) := 6.0
   sv(_mul1) @ bb(0) := 7.0
   sv(_shr3) @ bb(1) := 0.0
   sv(ssrc_1) @ bb(1) := 0.0
   sv(_mul4) @ bb(1) := 1.0
   ...
   ```

   Scheduling Variables using ALAP scheduling:

   ```
   sv(_b) @ bb(0) := 0.0
   sv(_a) @ bb(0) := 0.0
   sv(ssrc_0) @ bb(0) := 0.0
   sv(_add) @ bb(0) := 1.0
   sv(_n) @ bb(0) := 1.0
   sv(_div) @ bb(0) := 10.0
   sv(_sub) @ bb(0) := 11.0
   sv(_rem) @ bb(0) := 11.0
   sv(_cmp) @ bb(0) := 12.0
   sv(br_entry) @ bb(0) := 13.0
   sv(ssink_0) @ bb(0) := 13.0
   sv(_mul) @ bb(0) := 2.0
   sv(_c) @ bb(0) := 5.0
   sv(_shr) @ bb(0) := 6.0
   sv(_mul2) @ bb(0) := 6.0
   sv(_mul1) @ bb(0) := 7.0
   sv(_shr3) @ bb(1) := 0.0
   sv(ssrc_1) @ bb(1) := 0.0
   sv(_mul4) @ bb(1) := 1.0
   ...
   ```

   Slack can be calculated using $Slack = Cycle_{ALAP} - Cycle_{ASAP}$. A $Slack = 0$ means that the operation can only be scheduled at one given time to meet the latency constraint. A $Slack > 0$ means there is a time interval in which the operation can be scheduled.

3.
- Kernel 1: Here, ASAP and ALAP have the same hardware demand, as they both use a maximum of 2 *mul* and 1 *add* operations at the same time.
- Kernel 2: Here, it's same case as in Kernel 1 again. ASAP and ALAP have the same hardware demands. This is the case because in *BB0*, the *mul* operations have 0 slack and are therefore scheduled at same time for ASAP and ALAP scheduling.
- Kernel 3: As shown in Figure 1 and Figure 2, the *BB1* with ASAP scheduling requires more hardware to fulfill the schedule.

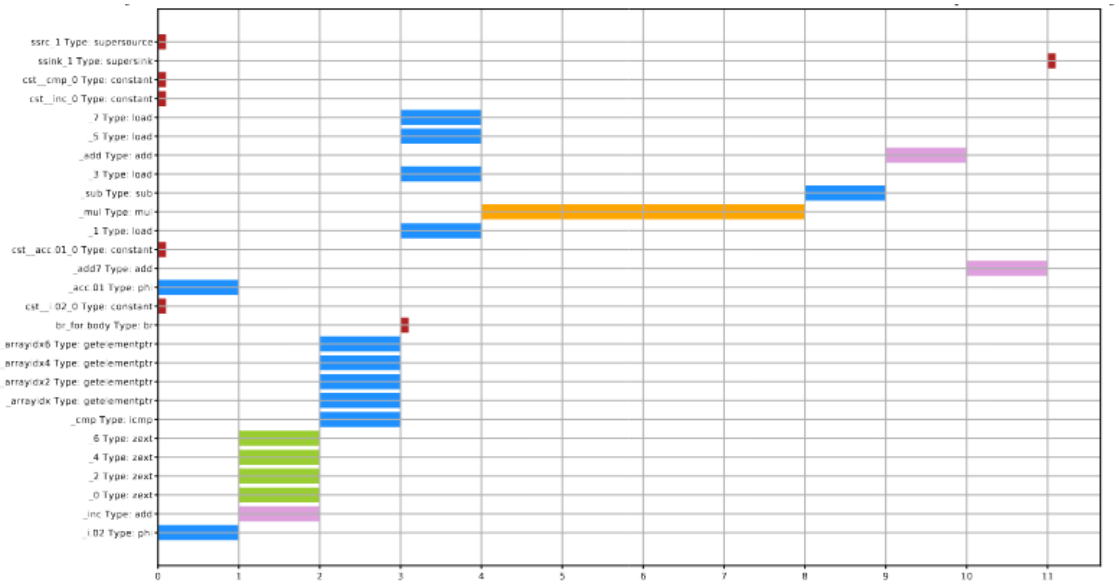

Figure 1: Kernel 3 with ASAP scheduling

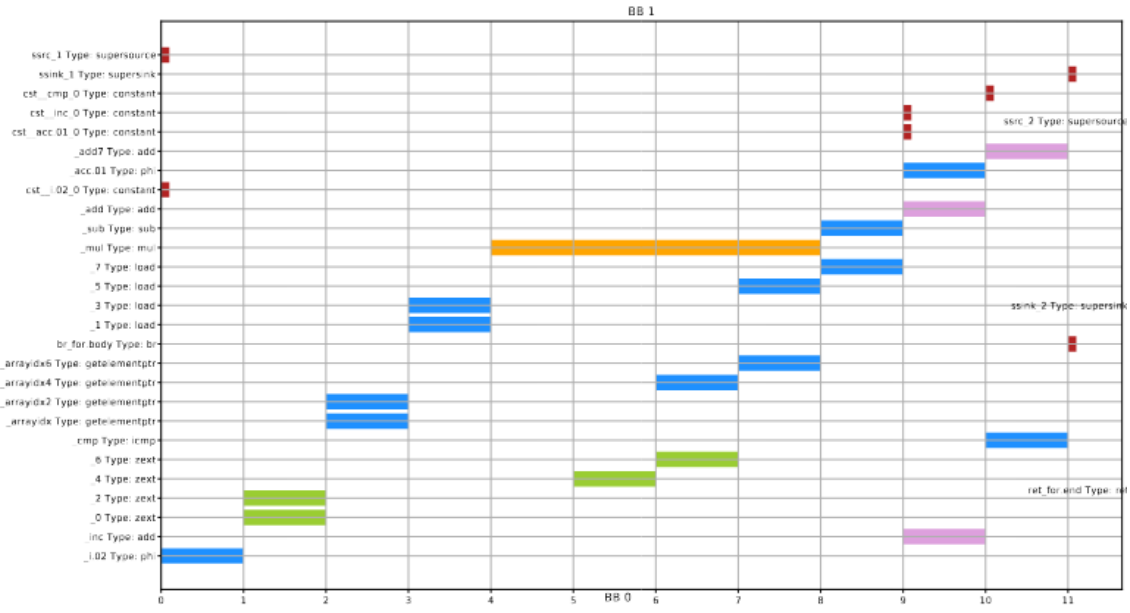

Figure 2: Kernel 3 with ALAP scheduling

Scheduling more of the same operation in the same cycle means that more hardware is necessary to execute the schedule. In the case of kernel 3, ASAP scheduling causes cycle 1 to need 4 *zext*

hardware units to execute properly, as opposed to the 2 *zext* units needed for ALAP. However, ASAP scheduling only needs 1 *add* unit, compared to the 2 units needed with ALAP, so there is a small tradeoff to be made.

- Kernel 4: For this kernel, ASAP scheduling requires 5 *add* units, while ALAP scheduling only needs 2.

3 Task 4: ASAP with Resource Constraints

1.
   - Kernel 1:
     - Setting the resource constraints to the lowest possible, i.e. {*'add': 1, 'mul': 1, 'zext': 1*} causes the highest achievable latency of 16.
     - Since there is only 1 *add* operation and 3 *mult* operations (of which maximum 2 can be scheduled at the same time). All resource constraints higher than {*'add': 1, 'mul': 2, 'zext': 1*} lead to the same schedule. They all have a latency of 13.
   - Kernel 2:
     - Because kernel 2 only has 2 *mult* operations, with 0 slack each, meaning they can never be scheduled at the same time, as well as 0 *add* and *zext* operations, resource constraints have no impact on this kernel's ASAP schedule. The schedule therefore always has a latency of 10.
   - Kernel 3:
     - Kernel 3 is the only kernel with *zext* operations
     - Setting the resource constraints to the lowest possible, i.e. {*'add': 1, 'mul': 1, 'zext': 1*} causes the highest achievable latency of 16.
     - Setting the resource constraints to the highest necessary, i.e. {*'add': 1, 'mul': 1, 'zext': 4*} for this kernel, leads to the lowest achievable latency of 11.
   - Kernel 4:
     - Setting the resource constraints to the lowest possible, i.e. {*'add': 1, 'mul': 1, 'zext': 1*} causes the highest achievable latency of 27.
     - Setting the resource constraints to the highest necessary, i.e. {*'add': 5, 'mul': 2, 'zext': 1*} for this kernel, leads to the lowest achievable latency of 22.
2.
   - Kernel 1: As shown in Figure 3, setting the resource constraints to a value larger or equal to the maximum number of needed resources, produces an identical schedule to ASAP without resource constraints

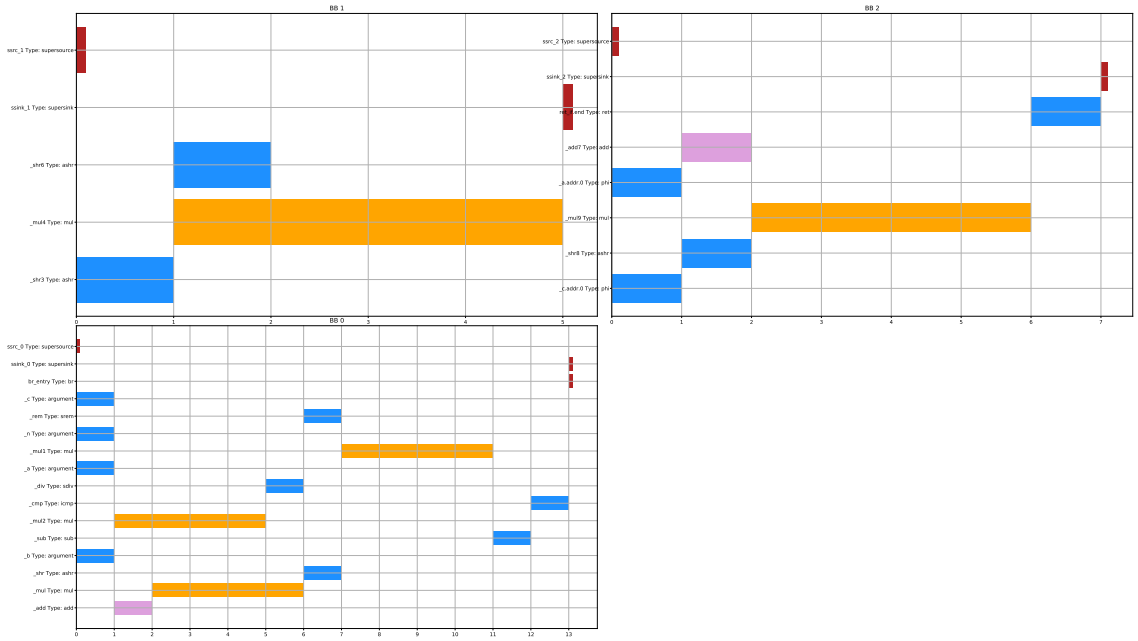

Figure 3: Kernel 1 with ASAP scheduling and resource constraints

- Kernel 2: Resource constraints have no impact on the schedule of this kernel, for reasons stated above.
- Kernel 3: For resource constraints {*'add': 1, 'mul': 1, 'zext': 4*} (or higher), the schedule is the same as ASAP without constraints.
- Kernel 4: For resource constraints {*'add': 5, 'mul': 2, 'zext': 1*} (or higher), the schedule is the same as ASAP without constraints.

4 Task 5: Pipelined Schedule

1. For N loop iterations, the total latency can be calculated as

$$total\ latency = latency + (N-1) * II$$

where *latency* refers to the latency of 1 iteration. The total latencies of the kernels for N loop iterations as well as the required area are listed below.

- Kernel 1: $II = 1$ and $latency = 13$, $total\ latency = 13 + (N-1)$, required area: 1 *add* and 2 *mult* units
- Kernel 2: $II = 1$ and $latency = 10$, $total\ latency = 10 + (N-1)$, required area: 1 *add* and 1 *mult* units
- Kernel 3: $II = 2$ and $latency = 11$, $total\ latency = 11 + (N-1) * 2$, required area: 1 *add*, 1 *mult* and 4 *zext* units
- Kernel 4: $II = 19$ and $latency = 22$, $total\ latency = 22 + (N-1) * 19$, required area: 5 *add* and 2 *mult* units

2. The achievable II's are listed above.

5 Task 6: Resource Constraints with Pipelining

1. No, the MRT heuristic does not compute the optimal scheduling. This is the case because it uses a iterative approach which simply increases the II until a valid MRT schedule is found. However, this method misses schedules where a slightly different arrangement of operations might be beneficial.

2. The MRT heuristic suffers from the limitation that all operations of one iteration must be executed consecutively (without any NOPs in between). The upside to this limitation is that it is very easily computable and thus very efficient.

3. The heuristic could be improved by also adding the possibility of adding NOPs in between operations or reordering the operations. However, this would lead to a much higher computational load and the MRT's efficiency would suffer greatly.

6 Final Words

All the tests pass for all kernels and methods, however, for some reason I have not been able to find, the tester sometimes fails on *kernel_4* using the methods *asap_rconst* and *pipelined_rconst*. This behaviour is very irregular, as consecutive runs of the tester (without touching anything in the code) can lead to different results of the tester. Other students have also reported this behaviour, as shown in the Moodle Forum.