ECE M216A Project – Fall 2024 | Deadline: Dec 7th, 2024

Hardware Realization of Multi-Program Placement (Rectangle Filling)

Motivation:

Consider a compute array (128x128) and a stream of incoming programs with varying widths and heights, both ranging from 4 to 16. We use a heuristic algorithm that places the incoming program onto the compute array. The goal is to come up with a power- and area-efficient implementation of the given algorithm, subject to the processing latency constraints.

Project Statement:

Implement the assigned program allocation algorithm using minimum area and power, taking precisely 8 clock cycles of signal processing latency at the top level.

Multi-Program Placement Algorithm:

This heuristic algorithm works based on fixed-height strip splitting of the given compute array region (128 x 128). Since the incoming programs can have any height ranging from 4 to 16, we have divided the compute array region into multiple fixed-height strips. The top three strips have a fixed height of 16 units each. The remainder of the compute array region is split into 10 strips of different heights, as shown in Figure 1. Overall, there are 13 strips for program placement, enumerated from bottom (ID = 1) to top (ID = 13), as defined in Table 1. The algorithm assumes that the programs can't be rotated. The programs are placed into the respective strips from left to right.

The idea of the placement algorithm is to first perform the equifilling search (explained in the next section) and check if the program rectangle can be placed in the compute array. This is checked by evaluating the occupied strip widths against the container size. If the program can fit in a suitable strip, the occupied width of that strip will be updated with an added-on program width. If the algorithm fails to find a space for the program, it will increment the Strike counter by 1. The flow chart of the equifill algorithm is shown in Figure 2.

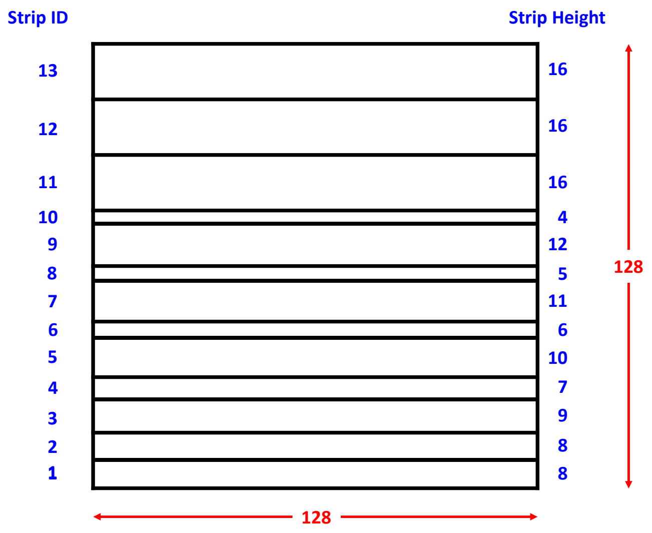

Fig. 1 Compute array segmented into 13 strips.

Equifilling search:

The algorithm tries to fill strips with closer heights alternatively and thus pack the given area from left to right while always making some space available for wider blocks (towards the right). This approach trades off reduced best-case utilization for improved mean utilization (measured in the percentage of array units occupied). The program inputs and outputs are specified in the flow chart and incorporated in the provided top-level Verilog wrapper.

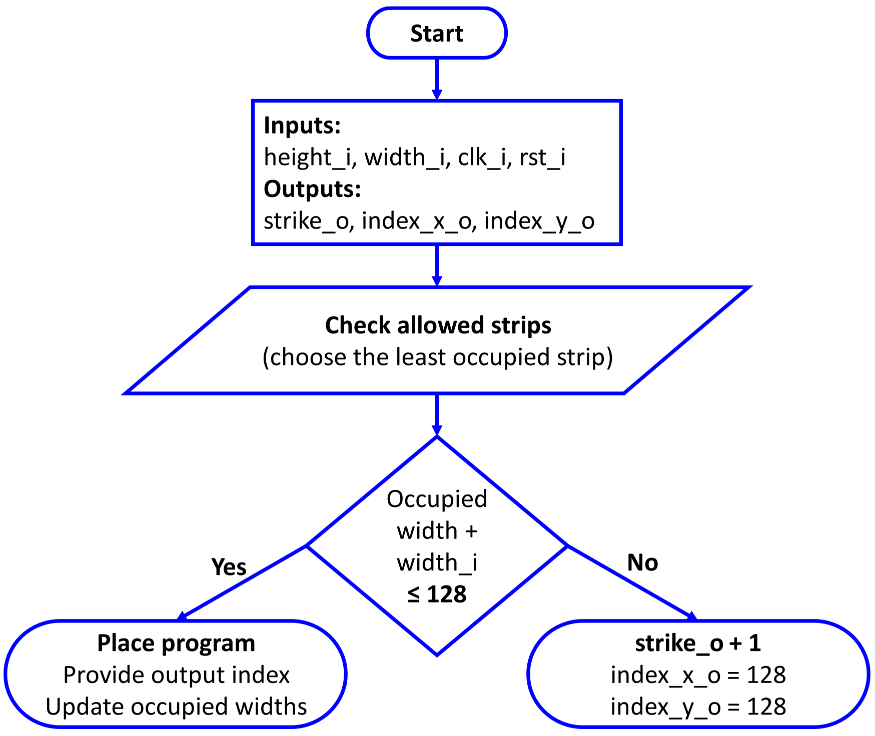

Fig. 2 Flow chart representation of the equifilling search algorithm.

Table 1 Equifilling search: a strip of height y can accommodate programs of height y or y−1*.

Strip ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Strip Height	8	8	9	7	10	6	11	5	12	4	16*	16*	16*
Program Height	7,8	7,8	8,9	6,7	9,10	5, 6	10,11	4,5	11,12	4	13, 14, 15, 16		

* except for the y=16 strips: these strips can accommodate program heights of 13, 14, 15, or 16.

In this way, the algorithm keeps the occupied width of any two strips to be somewhat close. For example, when a block with height 4 comes in, it can either go into the strip of height 4 or height 5 depending on which strip has a lower occupied width. Thus, when a rectangle with a height y comes in, we check the occupied width of the strips with height y and y+1, as in Table 1.1, and place it on the lower-occupancy strip.

Note:

- The programs with a height of 12 can only fit in one strip.
- If the top three strips are of equal occupied width, priority is for the higher strip.
- If the bottom two strips are of equal occupied width, priority is for the lower strip.
- For a program height y, if both y and y+1 strips have equal occupied width, priority is for the strip of height y.
- When placing a program of height y in a strip with height greater than y, the vertical alignment of the program is towards the bottom of the strip.

The equifilling search algorithm ensures that for a randomly distributed input, we have evenly filled strips, while also reserving space for bigger blocks towards the right. This equifilling search method increases the array mean utilization (percentage) significantly.

Top-Level Verilog Module:

The top-level wrapper **M216A_TopModule.v** containing the required I/O (defined below and in Figure 2), input registers (this counts as latency of 1), and output registers (this also counts as latency of 1) is provided. You need to incorporate your design into this top-level module.

The inputs 'height_i' and 'width_i' arrive every **4** clock cycles, so you need to make sure there is no overlapping placement, even if there are two consecutive blocks with the same heights. Thus, after the first 8 clock cycles, output will start coming out, which should change every 4 clock cycles (corresponding to different inputs).

Inputs:

height_i	5-bit incoming program height (ranges from 4 to 16)
width_i	5-bit incoming program width (ranges from 4 to 16)
clk_i	1-bit clock, this is the system master clock
rst_i	1-bit reset, if rst_i is "high," clear all the occupied widths to "0"

Outputs:

index_x_o	8-bit left-bottom index, horizontal, of the allocated program
index_y_o	8-bit left-bottom index, vertical, of the allocated program
strike_o	4-bit strike counter If Strike happens, (index_x_o, index_y_o) = (128, 128)

Design Constraints:

- Latency from any of the inputs to any of the outputs is precisely 8 clock cycles (including 2 cycles of latency from input and output registers)
- Minimize area and energy (energy = power / clock frequency)
- Maximize clock frequency.

Project Testbench:

One testbench, **M216A_TB.v**, has been provided on bruinlearn. The output generated by your design must match all the transitions in that testbench. The provided testbench will tell you if your design failed any of the test cases. If you implemented the algorithm correctly, this test bench will suffice. To guard against hard-coded test-bench single-point solutions, your final submitted code will be additionally tested with some other test cases to ensure that the correct algorithm is implemented.

Two files namely input_test.txt and output_test.txt have also been provided on bruinlearn. The testbench reads these two files to simulate and verify the top-level wrapper M216A_TopModule.v Copy all these files to the same directory where you have placed the top-level module and the testbench. Invoking ModelSim from this same working directory ensures that the testbench can read these files.

Project Deliverables:

- Design and performance summary document (file name: **Group_N.pdf**)
    - N indicates your group number
    - Include the names of all group members at the beginning of this document
    - List the following specs: Maximum clock rate, Area, Power, Hold Time slack
    - Provide a top-level block diagram of your design and its key elements
    - List key features of your design in no more than 5 lines of text
- All files below should be packed into one ZIP file (file name: **Group_N.zip**)
    - N indicates your group number
    - Top-Level Verilog Module (**M216A_TopModule.v**)
    - Your synthesis script (**Group_N.tcl**)
    - Synthesized Area and Power Reports (Reference: Lab2)
        - Files: **Group_N.Area, Group_N.Power**
    - Synthesized Timing Report (Slack) for your maximum clock frequency
        - Make sure you do not have any hold time violations
        - Files: **Group_N.TimingSetup, Group_N.TimingHold**

Project Submission:

Upload the following files to bruinlearn by **4 pm on Dec 7th, 2024**. Both the files should be uploaded before the deadline to be considered as a valid submission.

Upload to Gradescope: **Group_N.pdf**
Upload to Assignments: **Group_N.zip**

Project Timeline:

Week 6 -Understand the algorithm and clarify all the queries
-Instructors to freeze the project by Nov 8th, 10 pm
-Students to declare project teams by Nov 8th, 10 pm (Project sign-up sheet)

Week 7-8 -Implement the given algorithm in Verilog
-Verify the functionality against the provided testbench

Weeks 8-9 -Minimize area and power
-Reduce the number of adders and comparators used against the provided latency budget
-Your design should take exactly 6 cycles of latency (total latency from input to output = 1 + 6 + 1 = 8 cycles)

Week 10 -Inspect for timing violations
-Fix any setup and hold time violations and generate the timing reports

Submission Template:

- **Group_N.pdf (1-page document, 12-pt font, 1-inch margins)**

ECE M216A Project, Fall 2024
Group-N Team Members:
<Name1>, email1
<Name2>, email2
<Name3>, email2

Group-N Performance Summary

Max f_{Clk} [MHz]	Area [μm^2]	Energy [pJ]	Hold Time Slack [ps]

[Provide top-level architecture, and indicate its key building blocks]

Figure. Architecture block diagram and its key building blocks.

Design Highlights: List key features of your design in no more than 5 lines of text.

- **Group_N.zip**

M216A_TopModule.v
Group_N.tcl
Group_N.Area
Group_N.Power
Group_N.TimingSetup
Group_N.TimingHold