ESE 461 Final Report

Amelia Ma, Daniel Skelton, Daniel Sullivan

Dec. 11 2017

For the final project, we designed an accelerator for Multilayer Perceptrons (MLP). MLP is the deep learning model used to approximate some function f that can produce the best results. Our design is a single-layer model used to identify handwritten digits, taken from the MNIST database, a subset of a larger set available from NIST.

Design Principle and Strategies

Our design principle is to minimize area and power consumption, while still keeping an acceptable performance.

The strategies we used to achieve our design goals are:

1. Keep the size of the RAMs small, and load them repeatedly from external when needed
2. Reuse some of the Units that are idle most of the time, including the LUT and the MAC
3. Pipeline the first and the second stage of calculation to improve performance
4. Gate the Output GSRAM and REG HOLDER

Data Reuse Choice

We chose to implement the first of the three data reuse strategies provided to us, the Weight Data Reuse strategy. Figure 1 illustrates how the weight data is being reused.

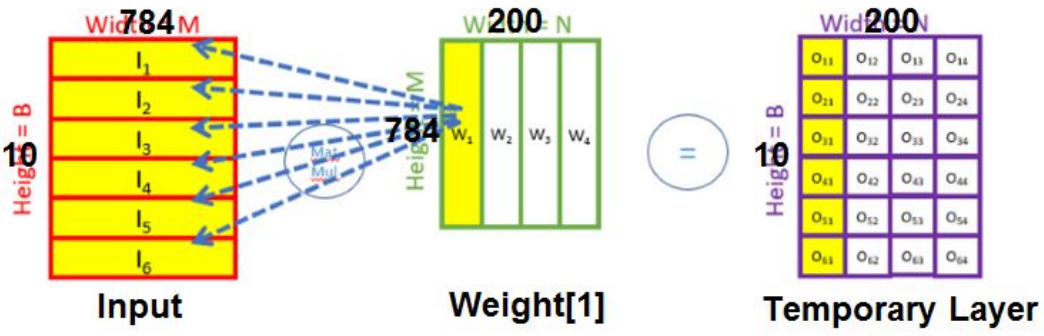

Figure 1: Weight Data Reuse

In layer 1, all 10 batches of the input are multiplied with each column of the weight. Each row*column process takes 784 cycles to complete, so in one clock cycle, a column of the input (10 elements with batch size 10) is multiplied with one weight[1] element, and stored as a partial sum.

Architecture

Based on the design strategies and the data reuse choices, we came up with our architecture. Figure 2 shows the overall architecture of the design, and Figure 3 shows the inner structure of the MAC units.

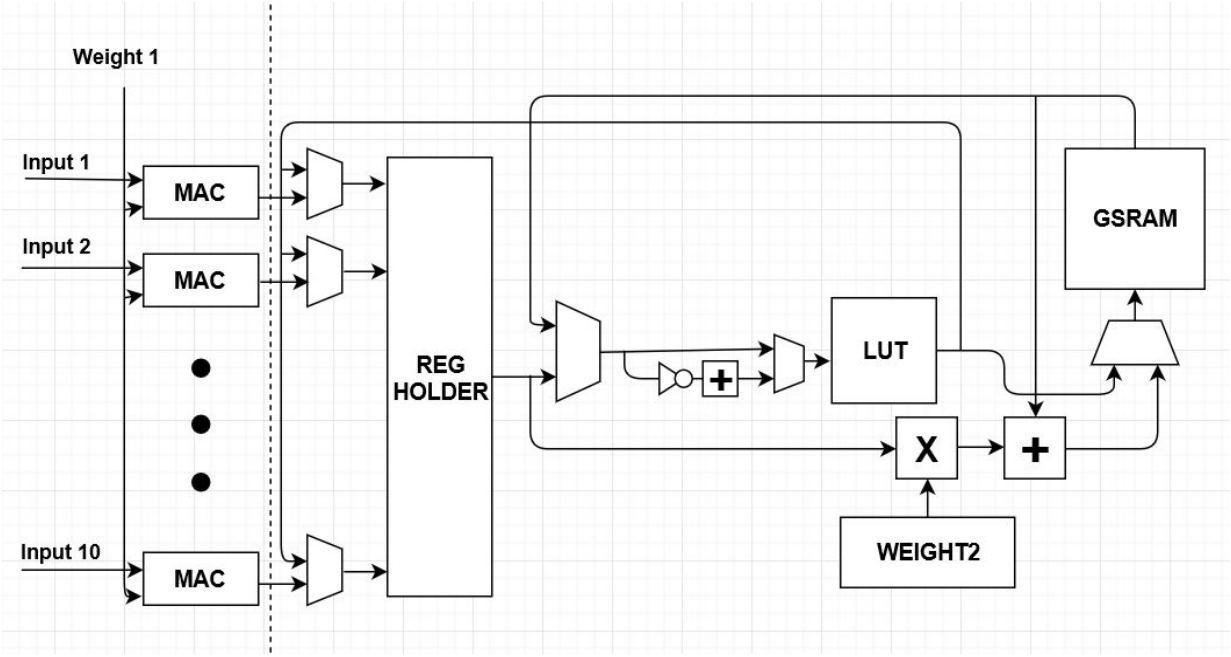

Figure 2: Design Architecture

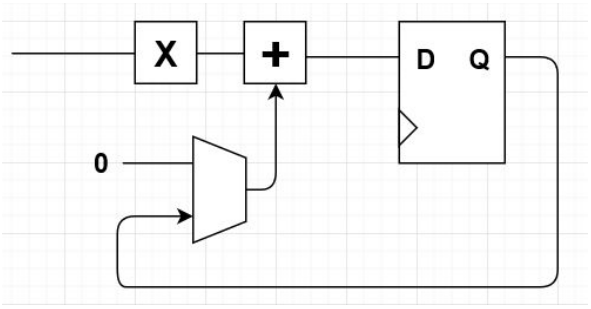

Figure 3: Inner Structure of MAC

Below are explanations for all of the parts specified in the architecture.

Input: The input register file has 10 16-bit values (a column of the input Matrix), each going into a MAC unit for the computation. The input column is updated every clock cycle, moving from left to right, and goes back to the first column when it reaches the last one. The inputs are read from external memory.

Weight 1: The weights for the first layer. It is also read from external memory at a rate of one data (16 bits) per clock cycle. It first goes through the column, and when one column is finished it goes to the next.

MAC: See Figure 3 for the inner structure of the MAC. The MACs in our design are combinational, and they will finish an operation (multiplication + accumulation) in one clock cycle. Besides the normal logic inside a MAC, we also added a MUX to reset the MAC without interrupting the input streaming. With the MUX, the MAC doesn't have to wait for one clock cycle to refresh its register.

Reg Holder: The register file used to hold the results from layer 1 after 784 accumulations. When each round of accumulation is over, the Reg Holder takes in all 10 values stored in the registers in the MACs. The Reg Holder first outputs to the LUT, one data at a time, and when activation is done the data goes back to it. When all 10 values are activated, the Reg Holder then outputs them to the multiplier for the second layer sequentially.

LUT: The Look-Up Table is implemented in a way that saves the area. Since the upper and lower halves of the sigmoid function are symmetric, we only implemented one half of it, and used the most significant bit to map to the correct value when the other half is needed. The LUT is first used for Layer 1 activation, and then reused for Layer 2 activation.

Weight 2: A length 10 register file that holds a row of the Weight 2 Matrix. Because of the way we reuse the Layer 1 weight, the results from Layer 1 come in columns. Thus, we load a single row of Weight 2 each time a row of data is available at Reg Holder. At every clock edge, a row

data from Layer 1 is multiplied with a column data from Weight 2, and accumulated with the partial sum stored in GSRAM.

GSRAM: A 10*10 register file that holds the partial sum and also the final output. For each round of operation (one Weight 1 column multiplication, there are 200 rounds in total), all the data in the GSRAM will be refreshed, which takes 100 clock cycles. When the 200 rounds are all finished, the accumulation values stored in the GSRAM will go through the LUT to activate, and then go back to the GSRAM to produce the final output.

Controller

The control logic is independent from the data path. Below are the output signals from the controller.

MAC_reset: Resets the partial sum stored in the MAC register. Selects the 0 in the MUX shown in Figure 3.
Reg_Holder_in: Write-enable signal for the Reg Holder.
Reg_Holder_mux: 0 for inputs from the MACs (10 inputs at once), 1 for input from the LUT (one at a time, needs address)
Reg_Holder_addr: Address of the Reg Holder.
LUT_mux: 0 for input from the Reg Holder, 1 for input from the GSRAM.
Weight2_addr: Column address for the Weight 2 register file. Ranges from 0 to 9.
GSRAM_in: Write-enable signal for the GSRAM.
GSRAM_mux: 0 for the input from the adder, 1 for the input from the LUT.
GSRAM_addr_row: Row address of the GSRAM.
GSRAM_addr_col: Column address of the GSRAM.

Detailed Operation

The dash line in Figure 2 represents our two pipeline stages. To the left is the first stage, also the first layer, which takes 784 clock cycles to complete. Since 784 corresponds to the number of pixels in one complete image, this means one weight and one pixel is being multiplied per

clock cycle for each image. In our design, we have ten images being computed in parallel because it was the batch size for the assignment.

On the right side of Figure 2 is the second stage. The output data from stage 1 enters stage 2 by loading into the REG HOLDER. From here, the ten data values are serially activated by the LUT block. Each individual activation takes four cycles, and the resultant data is written back into the same address in the REG HOLDER that it was stored at after exiting stage one. Next, the ten activated values are multiplexed through stage 2's lone MAC unit where they are multiplied by a weight value stored in the WEIGHT2 SRAM. This multiplexing process happens ten times for each activated value stored in the REG HOLDER. This equates to 100 multiply and accumulate operations that occur during this stage. Each individual multiplex across the ten activated values use the same weight value in the MAC. The next multiplex pass over the 10 values uses a weight value in an adjacent column of the second weight matrix.

After the 100 MAC operations complete, all weight values in a row of the weight 2 matrix have been multiplied by the activated stage 1 values for the current interation. The activated stage 1 values themselves map to a column of the hidden layers resultant matrix, where each row in the matrix is a different input image. The 100 outputs from the MAC are stored in the GSRAM. Since the GSRAM is a 10x10 memory with each memory cell containing 16-bits. Each partial product output from the stage 2 MAC gets their own location in the GSRAM. Each row in the GSRAM corresponds to a separate image. These stored values also serve as accumulate values that are used in the stage 2's MAC when their corresponding activated stage 1 output enters the unit.

One whole pass through the stage two unit takes 241 clock cycles. One cycle is used to load the stage one outputs into the REG HOLDER. The next 40 cycles are used to activate the stage one outputs. The final 200 cycles are required for the 100 MAC operations. Each MAC operation takes two cycles because data has to be read from the GSRAM and then written back to it to complete one multiply and accumulate.

Stage 2's activation only occurs after the 10x10 output matrix has been calculated. Each of the 100 values store in the GSRAM are sent to the LUT and then written back into their position in

the GSRAM. This whole process takes 300 clock cycles to complete. One cycle is for reading from the GSRAM, one is for LUT activation, and one is for writing back to the GSRAM.

Behavioral Simulation

We ran the Simulation for the entire design and we were confident about the final answers we got. Figure 4 shows the Simulation graph of all the 200 rounds of calculation.

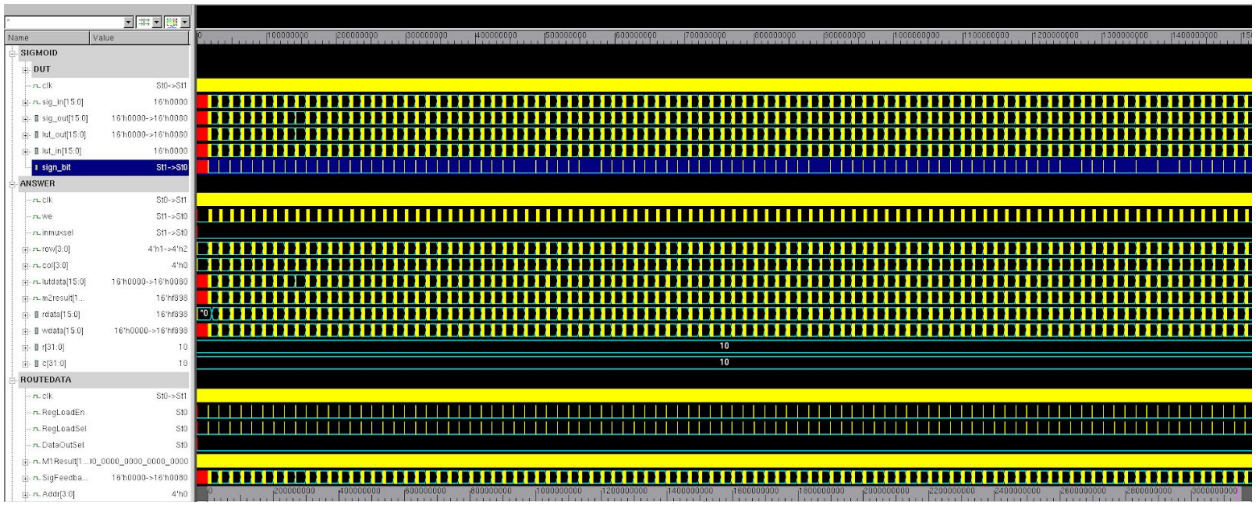

Figure 4: Overall Simulation Result

Figure 5 shows a more detailed Simulation screenshot, with the first 10 outputs:
The outputs can be observed being written to the output GSRAM in the ANWSER section of the simulation. The row and col indicate the row and column of the 10x10 GSRAM. When the “we” signal goes high, the value on the wdata line is written to memory on the following rising clock edge.

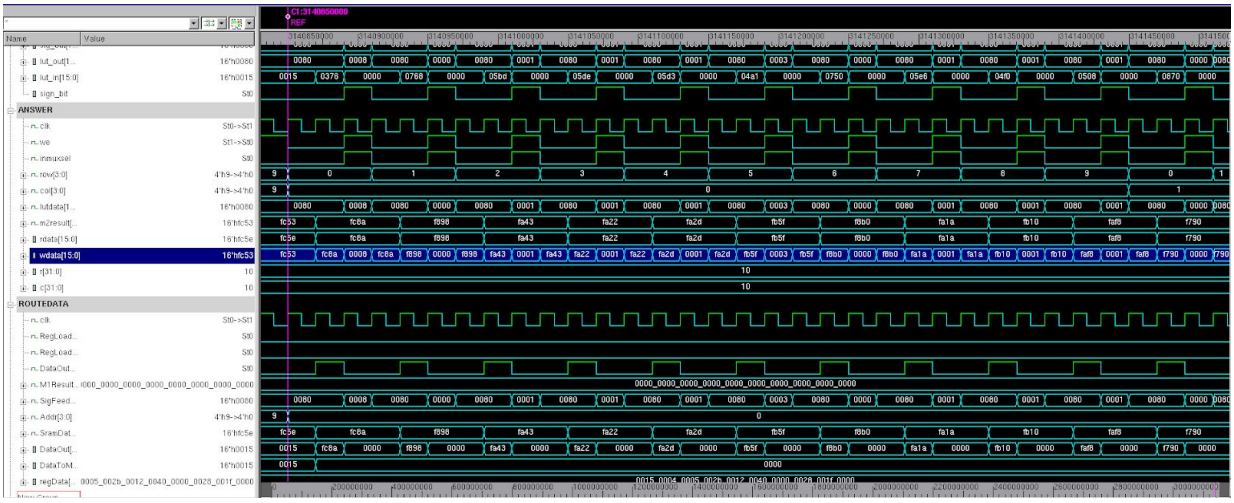

Figure 5: Simulation Result for the first 10 outputs

Figure 6 demonstrates how we used clock gating to optimize our power consumption. See the Optimization section for details. The gate control signals, named “Gate” in the ROUTEDATA section and “gate” in the ANSWER section, are highlighted in blue in the figure. When the signals are asserted the signals “clkGate” and “gate” mirror the system clock, which is named “clk” in the simulation below. The “clkGate” and “gate” signals drive the sequential logic in their modules. Close study of the figure below will reveal the gated signals are asserted when the modules are being utilized.

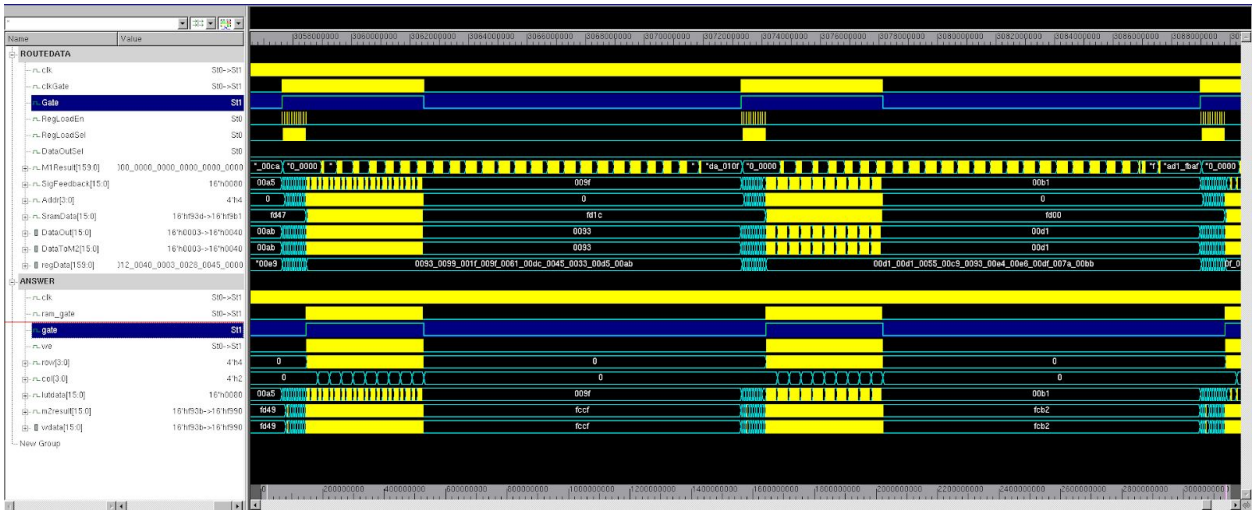

Figure 6: Clock Gating Figure

Table 1 below shows the output matrix in hexadecimal for the 10 images given in piazza:

Image	1	2	3	4	5	6	7	8	9	0
1	0008	0000	0001	0006	0010	00DF	0001	004A	0000	0073
2	0000	0000	0000	0001	0002	00AB	0000	0099	0001	00EF
3	0001	0001	0000	0009	0003	0093	0000	004A	0001	0080
4	0000	0000	0000	000F	0008	000D	0001	0021	0001	001C
5	0001	0000	0000	0003	0003	009F	0000	00A5	0002	0012
6	0003	0000	0001	0005	0003	0099	0000	0080	0000	0080
7	0000	0000	0000	0002	0001	00C0	0000	0080	0009	0067
8	0001	0005	0000	0007	0005	0099	0000	005B	0001	0093
9	0002	0000	0001	000A	0000	0099	0000	0067	0000	007A
10	0002	0000	0000	0003	0005	003B	0000	001C	0003	0007

Table 1: Output Matrix for Test Images 1-10

All of our output images were interpreted to be either six, eight, or zero (see red highlighted values). Image one's values were close to the posted answers but had some associated error. Perhaps, this was because we always truncated bits instead of rounding. The highest probability for image one was for the number six. The number six also had the highest probability on the posted solution.

Optimization

Below are the strategies we used to further optimize our design:

1. Reducing the size of the input SRAM. The input SRAM was 10*16-bit, but since the first 7 bits of each input data are always 0, we reduced each data to 9 bits, creating a final 10*9-bit input SRAM. This resulted in a reduction in the number of adders for our Layer 1 MACs, which took a large percentage of the area of our entire design. The same was

done with weight1. These reductions also reduced the number of I/O pins and the size of the buses which brought the weight1s and pixels on-chip.

2. Clock gating the output GSRAM and REG HOLDER (see schematic above). Both memories were idle most of the time and therefore, wasting a considerable amount of power. With a 20 MHz clock before clock gating, the total power consumption after DC synthesis was approximately 5.90 mW. After clock gating, it dropped to 1.57 mW. Although some of the drop in power can be attributed to shrinking our memory units, much of it was related to gating. Unfortunately, I did not save the synthesis results with only the memory shrink, but if memory serves me correct, it was about 4.7 mW. Regardless, the amount of power saved was non-trivial.

Test Bench

Since the memory on our chip is limited, our test bench essentially functioned as an off-chip memory controller. As a result, all pixels and weights are passed into our design via the test bench. Our approach made it was necessary to carefully time the delivery of the image inputs and the weight 1 and weight 2 values with the operation of our design. It also made our design more easy to synthesize as our memories were not optimized out.

Future Optimizations

There are several optimizations that were not made due to time restrictions. We were unable to gate the WEIGHT2 SRAM in stage 2 of our design. This would have likely resulted in a non-trivial amount of power saving. Also, there are some wasted clock cycles in the process where stage 1's outputs are activated by the LUT. Doing the activation more quickly would have resulted in some modest power gains when matched with clock gating. Another possible optimization would be writing custom, pipelined multipliers for stage 1 and/or stage 2. Since we use the IP core combinational multipliers in our design, they were our worst case path. Perhaps a two stage pipelined multiplier would result in decreased latency and faster execution times with minimal gain in area. Also, it would have been a fun experiment to add sequential multipliers in the second stage of our design. Since they are smaller than combinational multipliers, we may have even been able to add more than one and still used less area than we

did in our design. Although, the multiple clock cycles and increased number of registers needed for sequential multipliers would have likely increased our power consumption.

Synthesis

We ran the Design Compiler synthesis tool at approximate clock frequencies of 20 MHz, 157.36 Mhz, and 240 MHz. The power, area, worst path latency, and total execution can be seen in Table 2. The 20 MHz synthesis was completed first to establish a baseline to work from. The other two clock frequencies represent upper and lower bounds for frequencies that still meet the minimum requirements given for the project while also operating within the physical limitations of our design. The 15.73 MHz clock is the slowest the chip can operate and still meet the execution time requirement of 10 ms while 240.00 MHz is the fastest it can run without any timing violations.

Approx. Clock Frequency (MHz)	Power (mW)	Area ($\mu m^{2)}$	Slack (ns)	Execution Time (ms)
20.000	1.5663	1,035,805.037	35094.39	7.87
15.73	1.2334	1,035,805.037	48650.39	10.00
240.00	20.1108	1,092,562.643	0.54	0.66

Table 2: Synthesis Results of Optimized Design

Place & Route

Figure 7 shows the layout of our design. There was a clock gate added to our design and a change to our sigmoid function after this schematic was created. Due to time constraints, we were unable to rerun Cadence. While these changes do affect the layout, the design is nearly identical.

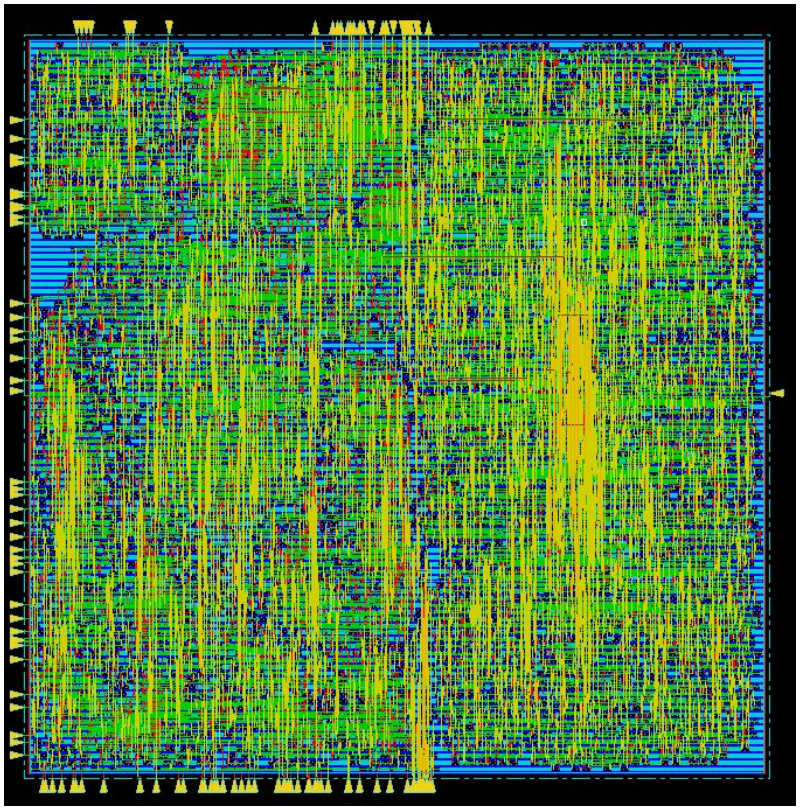

Figure 7: Design Layout