Table of Contents

Introduction

Design project 1 required the design, simulation, and implementation of a combinational logic circuit using multiple binary inputs and resulting in multiple binary outputs. Specifications required that the combinational circuit output a single decimal on an inverted seven-segment display when given four binary inputs A, B, C, and D that represented a 5211 binary-coded decimal digit. Each segment on the display was individually controlled by separate pins that accepted an active-low signal for the segment to turn on.

A, B, C, and D represented the highest to lowest significant digits of the 5211 encoded binary input. With the limit of three two-input NAND gate chips and two chips with six inverter gates each, the implementation of all seven segments of the display was deemed improbable. These limitations and a given minimum of three implemented segments affected the overall design and implementation processes.

Design Approach

The initial design of the circuit withdrew the physical limitations put upon the project by the limited chip count. In order to get the simple input-to-output gate-level circuit model of the specified combinational circuit, several problem-solving steps were used to first derive the Boolean algebra expressions of each of the segment outputs.

The entire input-to-output relationship was first laid out on a truth table relating each combination of possible inputs to each of the outputs. Each input combination required an output bit, which was determined from specification. Karnaugh maps based on these truth tables facilitated the construction of Product of Sums and Sum of Products Boolean equation representations of the required circuits.

A	B	C	D	t
0	0	0	0	0
0	0	0	1	1
0	0	1	0	x
0	0	1	1	0
0	1	0	0	x
0	1	0	1	0
0	1	1	0	x
0	1	1	1	1
1	0	0	0	0
1	0	0	1	x
1	0	1	0	1
1	0	1	1	x
1	1	0	0	0
1	1	0	1	x
1	1	1	0	0
1	1	1	1	0

Table 1 - Sample Truth Table

CD AB	00	01	11	10
00	0	1	0	x
01	x	0	1	x
11	0	x	0	0
10	0	x	x	1

Figure 1 - Sample Karnaugh Map

Until this point, each segment driver was represented in either Sum of Product or Product of Sum form, which represent hardware AND and OR gates. After deriving the Boolean algebra expressions of the driver circuits, a conversion of the equations was necessary to satisfy the "NAND and inverter gate only" requirement. From the converted equations, a gate-level circuit model was derived.

The circuit model using only gates was used to verify proper operation of the circuit as specified by the derived equations. Once proper operation of the entire circuit, including proper output to corresponding inputs, was verified, a reduction in the implemented driver circuits was necessary before a chip-level circuit model could be constructed; the gate-level circuit model used 23 NAND gates and 16 inverter gates to implement all 7 display segments and their display circuits. The limit of three quad NAND gate chips and two six-inverter gate chips meant a total of 12 NAND gates and 12 inverter gates were available for implementation.

In order to meet the limit of 12 NAND and 12 inverter gates, two driver circuits were dropped from the final implementation, and the driver circuit equations had to be rearranged to a form where common NAND functions (and their inverses) from different output driver circuits could be reused. The rearranging of POS equations using identity 14 from Table 1-1 proved useful in reducing the hardware count. These rearranged equations were used to reconstruct the gate-level circuit model, and five output segments were deemed possible to build using the limited hardware.

A chip-level circuit model was then constructed to prototype the circuit as it would appear on the final implementation of the circuit. Five segment driver circuits were implemented. Once a second verification using the chip-level circuit model showed the proper operation of the derived circuit, the final implementation was built on the A&D circuit board.

Validation of the final implementation verified correct design of at least five of the seven initial driver circuits.

Derivation of Equations

Project specifications required each output segment of the seven-segment display to be controlled separately. Consequently, each segment was treated as a separate binary output to be controlled by the segment's driver circuit. Each segment was given a name: t, u, v, w, x, y, and z.

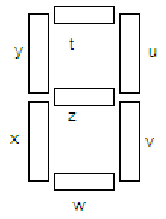

Figure 2 - Segments and Corresponding Output Values

For the 5211 binary code input, a table showing each 4-digit input to a decimal digit was given. A set of figures showing how each decimal digit should be displayed was also given.

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	-	2	-	3	-	4	5	-	6	-	7	-	8	9

Table 2 - Binary Input to Decimal Digit Conversion Chart

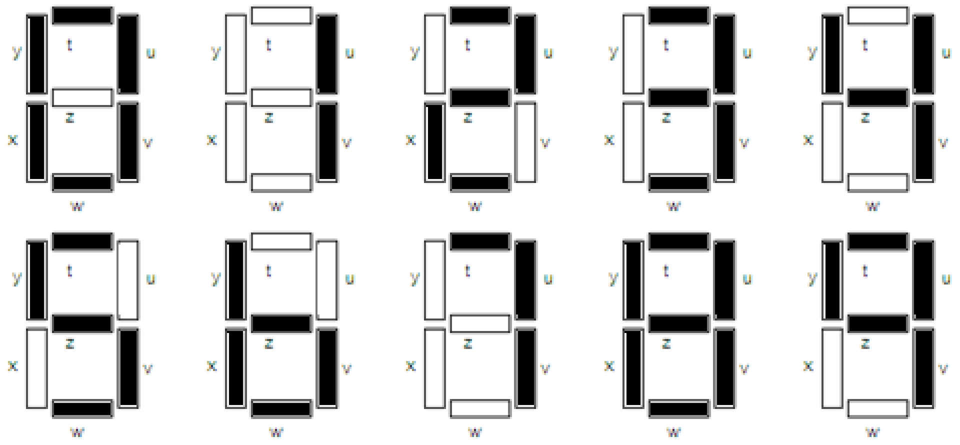

Figure 3 - Decimal Digits on 7-Segment Display

Given this information, each segment had to be evaluated and compared to the 16 different input combinations possible in a 4-bit binary input situation. Each segment was given a truth table based on which segments should be lit for a given input combination.

A	B	C	D	t	u	v	w	x	y	z
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	x	x	x	x	x	x	x
0	0	1	1	0	0	1	0	0	1	0
0	1	0	0	x	x	x	x	x	x	x
0	1	0	1	0	0	0	0	1	1	0
0	1	1	0	x	x	x	x	x	x	x
0	1	1	1	1	0	0	1	1	0	0
1	0	0	0	0	1	0	0	1	0	0
1	0	0	1	x	x	x	x	x	x	x
1	0	1	0	1	1	0	0	0	0	0
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	0	0	0	1	1	1	1
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	1	1	0	0

Table 3 - Combined Truth Table

In the above truth table, each input combination was compared to the information in Table 2 for the digit that should be displayed. The segments that were to be lit up for the given input combination were marked with a 0 in the corresponding section of the table, since the seven-segment display had an active low input, meaning a logic 0 would represent when the segment was on. Conversely, the segments that were not to be lit up were marked with 1's.

For the sections of the tables that represented the four-bit input combinations but did not represent a decimal digit in the 5211 code, "don't care" conditions were marked using x's. With the table filled out, Karnaugh maps for each of the seven outputs could be constructed.

CD AB	00	01	11	10
00	0	1	0	x
01	x	0	1	x
11	0	x	0	0
10	0	x	x	1

Karnaugh Map - t

CD AB	00	01	11	10
00	0	0	0	x
01	x	0	0	x
11	0	x	0	0
10	1	x	x	1

Karnaugh Map – u

CD AB	00	01	11	10
00	0	0	1	x
01	x	0	0	x
11	0	x	0	0
10	0	x	x	0

Karnaugh Map – v

CD AB	00	01	11	10
00	0	1	0	x
01	x	0	1	x
11	1	x	1	0
10	0	x	x	0

Karnaugh Map - w

CD AB	00	01	11	10
00	0	1	0	x
01	x	1	1	x
11	1	x	1	0
10	1	x	x	0

Karnaugh Map - x

CD AB	00	01	11	10
00	0	1	1	x
01	x	1	0	x
11	1	x	0	0
10	0	x	x	0

Karnaugh Map – y

CD AB	00	01	11	10
00	1	1	0	x
01	x	0	0	x
11	1	x	0	0
10	0	x	x	0

Karnaugh Map - z

Figure 4 - Karnaugh Maps

Considering the layout of the Karnaugh maps and NAND gate specification, Product of Sums simplification was chosen as optimal. The use of Product of Sums simplification provided many maxterms that included two input bits or less, whereas the use of Sum of Products simplification would have yielded many minterms that included three input bits. Since hardware was limited to two-input NAND gates, the use of equations derived from Product of Sums simplification were deemed more straight-forward for use in implementation.

Product of Sums simplification still provided a few maxterms that included three input bits. For these maxterms, further equation transformation was necessary to reduce the maxterms to a two-input AND function (NAND with inverted output) ANDed again with the third bit.

t in POS form: $(C + D)(B' + C)(A' + B')(B + C' + D')$ **t in NAND & Inverter form:** $(C'D')'(BC')'(AB)'(B'CD)$
u in POS form: AB' **u in NAND & Inverter form:** AB'
v in POS form: $B'CD$ **v in NAND & Inverter form:** $B'CD$
w in POS form: $(B + D)(B + C')(C' + D)(B' + C + D')$ **w in NAND & Inverter form:** $(B'D')'(B'C)'(CD')'(BC'D)'$
x in POS form: $(A + D)(B + C')(C' + D)$ **x in NAND & Inverter form:** $(A'D')'(B'C)'(CD)'$
y in POS form: $(B' + C')(B + D)$ **y in NAND & Inverter form:** $(BC)'(B'D')'$
z in POS form: $C'(A' + B)(B' + D')$ **z in NAND & Inverter form:** $C'(AB')'(BD)'$

Table 4 - Output Equations in POS Form and Derived NAND & Inverter Form

Driver Circuit Selection

Once the output equations were derived, a gate-level circuit model could be constructed. In the process of constructing the gate-level circuit, some of the NAND gates used in certain segment drivers were reused in others when the given result of a NAND was needed in another driver circuit. This was done for the sake of hardware count reduction. For example, almost the entirety of the *v* segment driver circuit could be reused in the *t* segment driver circuit, and a result from the *z* segment driver circuit was used in the *x* segment driver circuit.

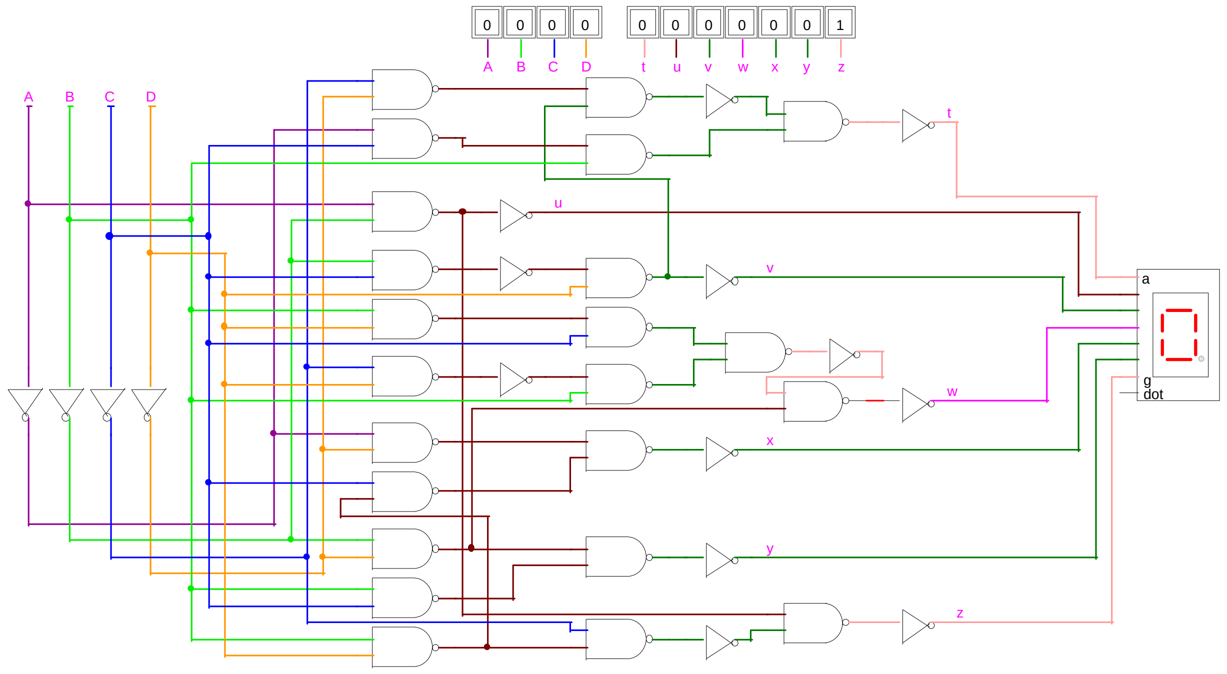

Figure 5 - Gate-level Circuit Model

Alternative designs may have shifted around some NAND gates and repositioned results available for reuse. Alternatives may have even required the use of more hardware, but this design was deemed the most conservative with hardware.

The gate-level circuit model required a total of 23 NAND gates and 16 inverter gates. With a limit of 12 NAND and inverter gates each, a couple of the segment driver circuits had to be left out from physical implementation. Careful consideration of the gate-level circuit model left us with five deployable driver circuits: u, v, x, y, and z. The w and t segment driver circuits were eliminated because they required the most gates in without the possibility of reuse. With 11 NAND gates and five inverter gates eliminated along with the t and w segment driver circuits, a physical implementation using 12 NAND and 12 inverter gates was possible.

Design and Implementation

The gate-level circuit model was used to build a chip-level circuit model in the form that it appeared on the physical implementation. Only the five driver circuits selected from the gate-level circuit model were implemented in the chip-level model.

Because of the ubiquity of inverter gates in gate-level circuit model, inverter chips needed to be somewhat spread out between the three NAND gate chips to prevent long wires from crossing back and forth over the breadboard. The implementation of driver circuits was done by logic level from left to right to minimize the length of wires reaching across the breadboard as well.

To allow for easier wiring, the colors of the wires in the chip-level model were color coded according to input bit before the first level of logic NAND gates were passed. After the first level of NAND gates, each level of wire was given its own color in order to differentiate between logic levels. Such differentiation facilitated a quick physical implementation by allowing each level of logic to be wired separately.

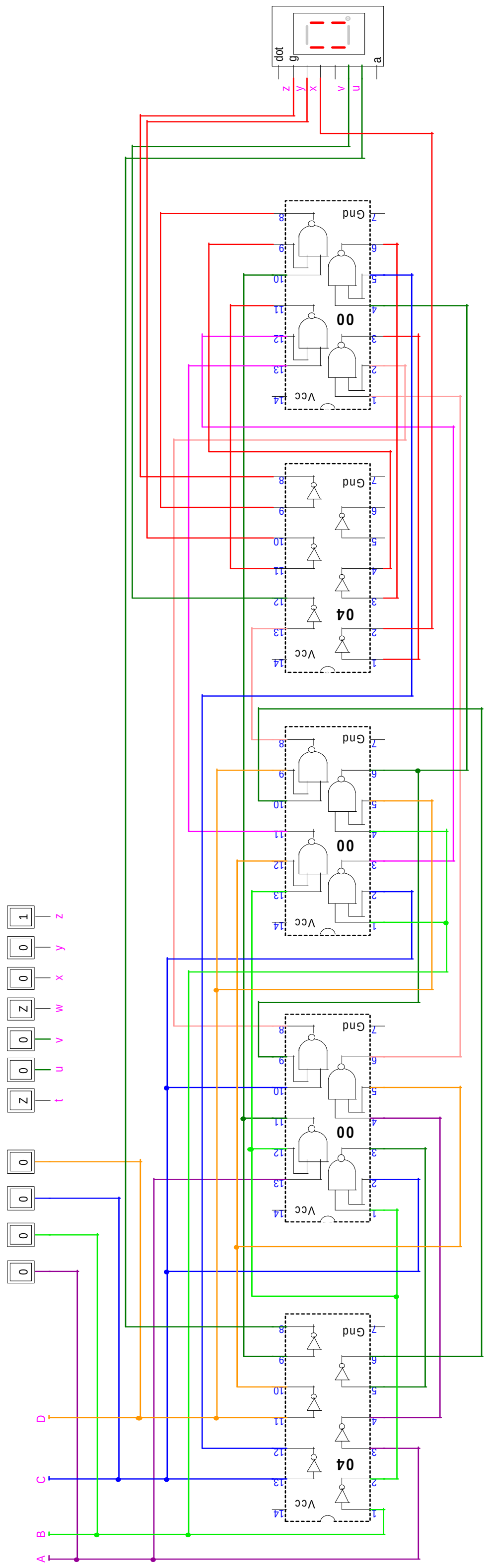

Figure 6 - Chip-level Circuit Model

Simulation Results

Both the gate-level and chip-level circuit models were tested via simulation before moving to the next stage of development. An input timing file exhausted the valid input combinations, and the resulting output timing files were deciphered to verify proper operation.

$T	$D	$I A	$I B	$I C	$I D
999	1	0	0	0	1
1000	200	0	0	0	0
1200	200	0	0	0	1
1400	200	0	0	1	1
1600	200	0	1	0	1
1800	200	0	1	1	1
2000	200	1	0	0	0
2200	200	1	0	1	0
2400	200	1	1	0	0
2600	200	1	1	1	0
2800	200	1	1	1	1
3000	1	0	0	0	0

Table 5 - Input Timing File

The input timing file was designed with a 200 time unit delay in order to avoid complications involved in hardware propagation delays. The first and last entries were used as delimiters to the first and last input cases, so that the timing window graphs would show more clearly where the simulation started and ended. A different combination input was applied for each 200 time unit section, starting from 0 to 9 in 5211 code.

$T	$D	$O A	$O B	$O C	$O D	$O t	$O u	$O v	$O w	$O x	$O y	$O z
0	1NS	0	0	0	1	0	0	0	0	0	0	1
1NS	3NS	0	0	0	0	0	0	0	0	0	0	1
4NS	1NS	0	0	0	0	0	0	0	1	1	1	1
5NS	1NS	0	0	0	0	0	0	0	0	0	0	1
6NS	1NS	0	0	0	0	1	0	0	0	0	0	1
7NS	194NS	0	0	0	0	0	0	0	0	0	0	1
201NS	4NS	0	0	0	1	0	0	0	0	0	0	1
205NS	2NS	0	0	0	1	0	0	0	1	1	1	1
207NS	194NS	0	0	0	1	1	0	0	1	1	1	1
401NS	3NS	0	0	1	1	1	0	0	1	1	1	1
404NS	1NS	0	0	1	1	1	0	0	1	0	1	1
405NS	1NS	0	0	1	1	1	0	1	1	0	1	1
406NS	2NS	0	0	1	1	1	0	1	0	0	1	0
408NS	193NS	0	0	1	1	0	0	1	0	0	1	0
601NS	3NS	0	1	0	1	0	0	1	0	0	1	0

604NS	1NS	0	1	0	1	0	0	1	0	1	1	0
605NS	1NS	0	1	0	1	0	0	0	0	1	1	0
606NS	3NS	0	1	0	1	0	0	0	1	1	1	0
609NS	192NS	0	1	0	1	0	0	0	0	1	1	0
801NS	3NS	0	1	1	1	0	0	0	0	1	1	0
804NS	1NS	0	1	1	1	0	0	0	0	1	0	0
805NS	4NS	0	1	1	1	1	0	0	0	1	0	0
809NS	192NS	0	1	1	1	1	0	0	1	1	0	0
1001NS	3NS	1	0	0	0	1	0	0	1	1	0	0
1004NS	1NS	1	0	0	0	1	1	0	1	1	1	0
1005NS	2NS	1	0	0	0	1	1	0	0	1	0	0
1007NS	194NS	1	0	0	0	0	1	0	0	1	0	0
1201NS	3NS	1	0	1	0	0	1	0	0	1	0	0
1204NS	3NS	1	0	1	0	0	1	0	0	0	0	0
1207NS	194NS	1	0	1	0	1	1	0	0	0	0	0
1401NS	3NS	1	1	0	0	1	1	0	0	0	0	0
1404NS	1NS	1	1	0	0	0	0	0	0	1	0	0
1405NS	1NS	1	1	0	0	0	0	0	0	1	1	0
1406NS	195NS	1	1	0	0	0	0	0	1	1	1	1
1601NS	3NS	1	1	1	0	0	0	0	1	1	1	1
1604NS	2NS	1	1	1	0	0	0	0	1	0	0	1
1606NS	195NS	1	1	1	0	0	0	0	0	0	0	0
1801NS	4NS	1	1	1	1	0	0	0	0	0	0	0
1805NS	2NS	1	1	1	1	0	0	0	0	1	0	0
1807NS	194NS	1	1	1	1	0	0	0	1	1	0	0
2001NS	3NS	0	0	0	0	0	0	0	1	1	0	0
2004NS	1NS	0	0	0	0	1	0	0	1	1	1	0
2005NS	1NS	0	0	0	0	1	0	0	0	0	0	0
2006NS	1NS	0	0	0	0	1	0	0	0	0	0	1
2007NS	0	0	0	0	0	0	0	0	0	0	0	1

Table 6 - Gate-level Circuit Model Simulation Output Timing File

The output timing files from both the gate-level and chip-level circuit model simulations show proper operation of each segment driver once the hardware propagation delays are over and Boolean functions are properly represented by the circuit. The outputs of the segment driver circuits shown in the gate-level circuit simulation output timing table for sections with delays of below 5 NS in the output timing file showed the NAND gates' and inverter gates' propagation times, and resulting spikes of changing output values.

The chip-level circuit model simulation output timing file showed similar results. Hardware propagation delays contributed to several output entries per input combination entry, but proper operation could be verified by checking the last output entry for each given entry. The sections that were supposed to be lit up for each input was shown with a 0 output.

$T	$D	$O A	$O B	$O C	$O D	$O t	$O u	$O v	$O w	$O x	$O y	$O z
0	1NS	0	0	0	1	Z	0	0	Z	0	0	1
1NS	87NS	0	0	0	0	Z	0	0	Z	0	0	1
88NS	1NS	0	0	0	0	Z	0	0	Z	1	1	1
89NS	112NS	0	0	0	0	Z	0	0	Z	0	0	1
201NS	88NS	0	0	0	1	Z	0	0	Z	0	0	1
289NS	112NS	0	0	0	1	Z	0	0	Z	1	1	1
401NS	66NS	0	0	1	1	Z	0	0	Z	1	1	1
467NS	22NS	0	0	1	1	Z	0	0	Z	0	1	1
489NS	22NS	0	0	1	1	Z	0	1	Z	0	1	1
511NS	90NS	0	0	1	1	Z	0	1	Z	0	1	0
601NS	66NS	0	1	0	1	Z	0	1	Z	0	1	0
667NS	22NS	0	1	0	1	Z	0	1	Z	1	1	0
689NS	112NS	0	1	0	1	Z	0	0	Z	1	1	0
801NS	66NS	0	1	1	1	Z	0	0	Z	1	1	0
867NS	134NS	0	1	1	1	Z	0	0	Z	1	0	0
1001NS	66NS	1	0	0	0	Z	0	0	Z	1	0	0
1067NS	22NS	1	0	0	0	Z	1	0	Z	1	1	0
1089NS	112NS	1	0	0	0	Z	1	0	Z	1	0	0
1201NS	66NS	1	0	1	0	Z	1	0	Z	1	0	0
1267NS	134NS	1	0	1	0	Z	1	0	Z	0	0	0
1401NS	66NS	1	1	0	0	Z	1	0	Z	0	0	0
1467NS	22NS	1	1	0	0	Z	0	0	Z	1	0	0
1489NS	22NS	1	1	0	0	Z	0	0	Z	1	1	0
1511NS	90NS	1	1	0	0	Z	0	0	Z	1	1	1
1601NS	66NS	1	1	1	0	Z	0	0	Z	1	1	1
1667NS	44NS	1	1	1	0	Z	0	0	Z	0	0	1
1711NS	90NS	1	1	1	0	Z	0	0	Z	0	0	0
1801NS	88NS	1	1	1	1	Z	0	0	Z	0	0	0
1889NS	112NS	1	1	1	1	Z	0	0	Z	1	0	0
2001NS	66NS	0	0	0	0	Z	0	0	Z	1	0	0
2067NS	22NS	0	0	0	0	Z	0	0	Z	1	1	0
2089NS	22NS	0	0	0	0	Z	0	0	Z	0	0	0
2111NS	0	0	0	0	0	Z	0	0	Z	0	0	1

Table 7 - Chip-level Circuit Model Simulation Output Timing File

Verification via simulation was easier done by checking results in the timing window. Though the timing window also showed signs of hardware propagation delay, graphical representation gave much better and verifiable results.

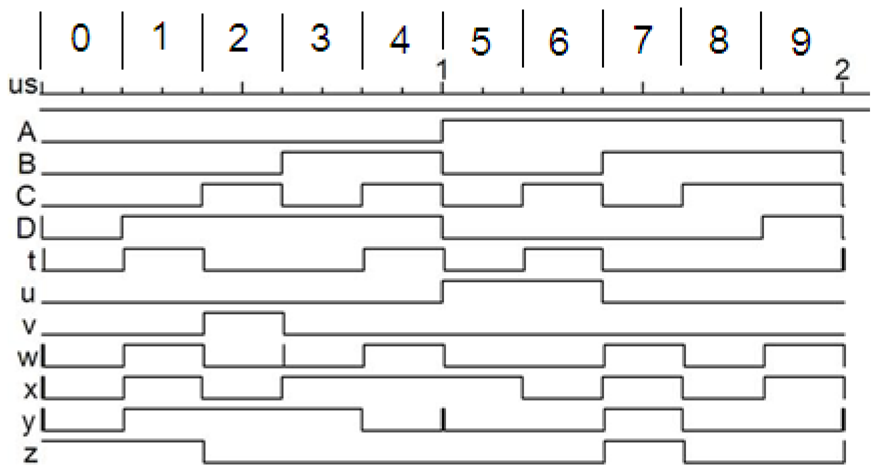

Figure 7 - Gate-level Circuit Model Simulation Timing Window

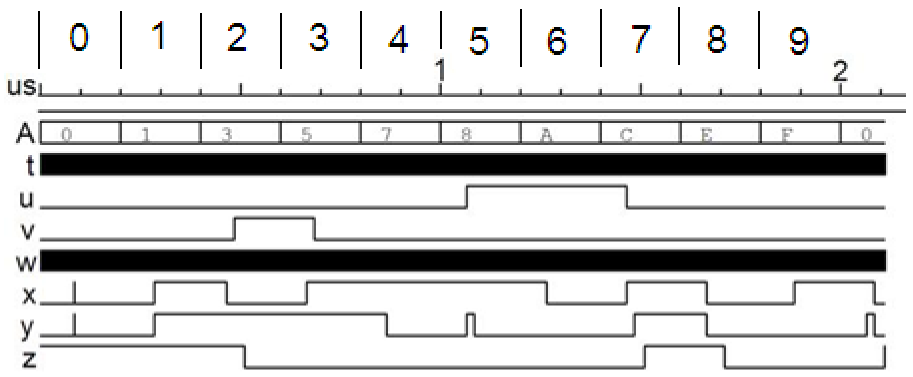

Figure 8 - Chip-level Circuit Model Simulation Timing Window

Both the simulation timing output files and timing windows show longer periods of instability for the chip-level circuit model in the output when input values change, the most notable example being in the output value of y when the input value changes from 4 (0111 in 5211 code) to 5 (1000). This difference in output propagation can be traced down to the difference in propagation delays of the gates used to simulate the two different models. Since the simulated chips were imbued with a realistic propagation delay, the chip-level circuit model showed a much more realistic propagation time.

The chip-level circuit model simulation timing window showed indeterminate values for the t and w segment outputs as they should have, since these two driver circuits were not implemented. Despite these differences, proper operation of the circuits could be verified after enough time was given for hardware propagation delays.

Conclusion

I felt the limitations on chip count and consequent segment implementation count provided for useful and applicable practice in reducing hardware usage. By reducing hardware and increasing efficiency, I gained valuable experience in reducing parts and production cost.

Though we were limited by availability to three quad NAND chips and two six-count inverter chips in our implementation, I felt that the use of three-input gates or other Boolean function gates may have reduced the hardware count. Circuits using these different gates may or may not have reduced final production cost, but it would certainly have simplified the design process. With the available hardware, however, five of seven total segments were successfully designed and implemented.