Chapter 3. Circuits

To assess the relative merits of each technology, circuits were designed and then implemented in each of the technologies. Six circuits, three combinational and three sequential, were implemented. Full custom implementations were created with the layout tool Magic from the University of California at Berkeley. For the other technologies a Verilog file was written which could then be compiled into standard cell, sea-of-gates, and FPGA technologies. The structure of the Verilog and full custom implementations were matched as closely as possible. To further augment the study, nine larger circuits with Verilog implementations were compiled into standard cell, sea-of-gates, and FPGA technologies.

3.1. Combinational

The three combinational circuits implemented in both a full custom layout and a hardware description language are a 16-bit adder, a 16-bit barrel shifter, and an 8-bit multiplier. The adder is a ripple carry adder with two 16-bit inputs A and B, a 16-bit output Sum, and a Carry Out. The Carry In is tied to ground, though its conversion to an independent input is, by design, easily achievable without altering the circuit's size or performance. An adder consists mostly of logic, with very little routing since each input is only needed in one part of the circuit. Thus, this circuit is appropriate for modeling logic-intensive portions of a design.

The 16-bit barrel shifter shifts its 16-bit input A between 0 and 15 places to the left, as determined by the value of S. The shifted value is output as B. Zeros are shifted in from the right to fill empty bit positions. This is done in four stages, one stage for each bit of S. An

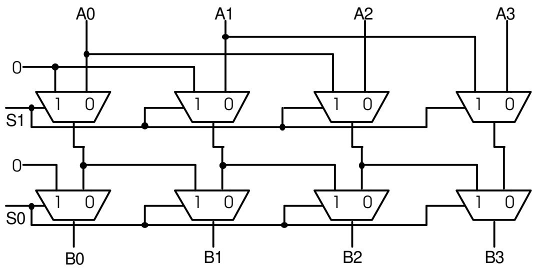

Figure 6. Two stages of a 4-bit barrel shifter implementation.

example of a 4-bit barrel shifter with two shifting stages is shown in Figure 6. This circuit is very regular, since the shift operation uses identical logic for each bit. Though the logic is simple and regular, routing connecting this logic is quite complex. This circuit is representative of routing-intensive portions of designs.

The 8-bit multiplier has 8-bit inputs A and B, and a 16-bit output Out. It is arranged as an 8x8 array of 1-bit adders, F, where position (0,0) is at the top right (Figure 7a). Each adder $F_{i,j}$ adds together the sum from $F_{i+1,j-1}$, the result of B_i AND A_j, and the carry out from its above neighbor, $F_{i,j-1}$ (see Figure 7b). Inputs from beyond the edges of the array ($F_{-1,j}$ to $F_{i,-1}$) are zeros. The result has two parts. Its lower eight bits, Out7 to Out0, are the SumOuts from adders $F_{0,0}$ to $F_{0,7}$. Its upper eight bits, Out15 to Out8, are the result of adding the 7-bit sum from adders $F_{7,7}$ to $F_{1,7}$ to the 8-bit carry out from bottom row of adders ($F_{7,7}$ to $F_{0,7}$). To increase throughput, this design can be pipelined by placing flip-flops between the rows of the array.

3.2. Sequential

The full custom versions of the four sequential circuits utilize flip-flops that are controlled by a two-phase clock. Thus all of the circuits require phi1 and phi2 to be supplied. All but an

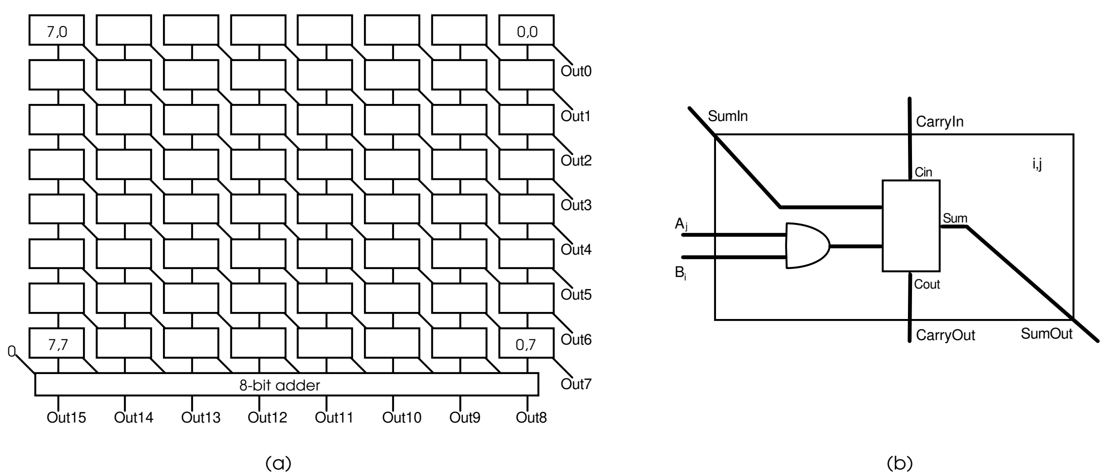

Figure 7. (a) The 8-bit multiplier's structure. (b) A single cell of the multiplier.

implementation of Conway's game of Life require the inverses to the clock as well. The standard cell, sea-of-gates, and FPGA versions use a single clock. The first sequential circuit, a state machine, is very simple. It is a stoplight controller circuit that contains a 3-bit counter made from T flip-flops. The resulting eight states control the light pattern for one direction of a stoplight. The only input is a reset signal, and the outputs are green, yellow, red, walk, blink, and don't walk signals. The perpendicular direction of an intersection could be controlled by another copy of this circuit which is offset by four cycles.

3.2.1. Game of Life

Conway's game, Life [5][6], is the first of two more complicated sequential circuits. It consists of sixteen cells which each have one state bit. The state of each cell indicates whether the cell is 'dead' or 'alive'. A dead cell becomes alive if exactly three of its neighbors are alive. A live cell dies if it doesn't have exactly two or three live neighboring cells. The cells are arranged in a grid, so each cell has eight neighbors. Cells on the border of the grid act as if their non-existent neighbors were actually dead neighbors.

One straightforward way to compute the state of a cell is to input the state of all of the neighboring cells, add these together, and set the state to logic 'one' if the result is three or if the state is currently 'one' and the result of the addition is two. However this requires redundant logic because the cells have certain sets of neighbors that are shared with other cells. For example, each cell shares its south-east, south, and south-west neighbors with the cell that is two positions to the south. In the scheme described above, both of these cells would be computing the sum of these shared neighbors' states in the process of computing the sum of all of their neighbors' states.

To take advantage of this, each cell in this implementation computes the sum of its state and

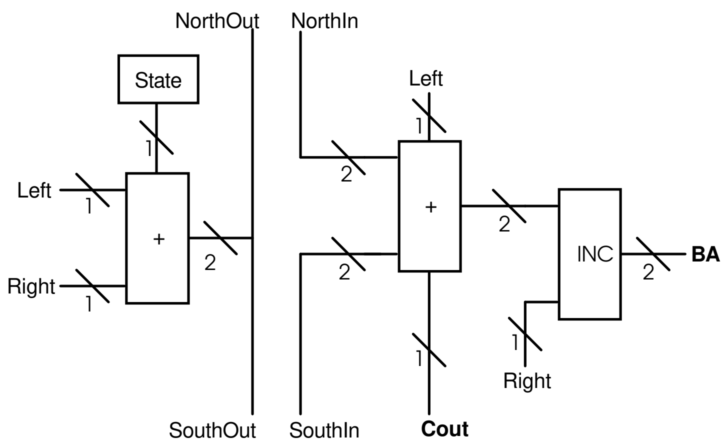

Figure 8. A single cell in the game of Life. Inputs are Left, Right, NorthIn, and SouthIn. Outputs are NorthOut, SouthOut, and State (to the Left and Right cells). B, A, and Cout are used to calculate the next cycle's value for State.

the states of its left and right neighbors in one full adder (inputs A, B, and Cin), and sends the two output bits (Sum and Cout) to its north and south neighbors. Thus each cell receives 2-bit values from both its southern and its northern neighbors, which are added together in a 2-bit full adder. The carry in to this full adder is the state of the neighbor on the east, accounting for seven of the eight neighbors. If the carry out, Cout, has a value of 'one' we know already that there are too many live neighbors (at least four), so the state bit is set to 'zero'. Otherwise, the 2-bit sum is incremented in two half adders by the state of the west neighbor. Since having four neighbors and having zero neighbors both mean death for current cell, the carry out of the increment can be ignored. The 2-bit output from the increment, BA, can be used along with Cout to derive the next state of the cell as follows.

$$\mathrm{next\_state} = \overline{\mathrm{Cout}} \cdot \mathrm{B} \cdot (\mathrm{A} + \mathrm{State}). \tag{1}$$

When the input LD is asserted, this calculation is not performed. Instead, each cells takes its state from its western neighbor. Cells on the array's western edge get their next state value from the inputs Load1 through Load4. There are four columns in this design, so the entire array can be loaded in four cycles. When not loading, inputs Load1 through Load4 must be set to zero, since they are in essence the west-most column's western neighbor. The array's output is read in parallel from this circuit's sixteen outputs. Because of its cellular nature, the circuit can easily be modified to create larger Life arrays. However, the size is limited by the number of I/O pads available for the parallel outputs.

This design of the game of Life is logic intensive, because of the locality of each cell's inputs. Another important factor in a sequential circuit is the proportion of logic to state information. This circuit contains a lot of logic, but each cell only needs to keep one state value. The next sequential circuit described retains much more state information, so it is much more dependent on the available storage. This distinction is extremely important for FPGA design, where the proportion of storage to logic resources is fixed.

3.2.2. DNA Comparator

A DNA comparator circuit accepts two strings representing strands of DNA and outputs a string that determines the genetic distance between the two strands. Each strand of DNA contains a sequence of nucleotides that are represented by the letters A, C, G, and T. The genetic distance between two strands of DNA is defined as the least number of deletions and insertions needed to transform one strand into the other. For example, the string ACGCTAG and AGGCTG have a genetic distance of three, since the second can be derived from the first in three steps (-represents a deletion, and a lower-case letter is an insertion): A-GCTAG, AgGCTAG, AGGCT-G. For a detailed description of the problem see [9] and [12].

A	00
C	01
G	10
T	11

Table 1: An encoding for DNA nucleotides.

For this study, subcells were designed so that an implementation of a DNA comparator which compares two N-length strands could be arrived at by tiling 2N-1 subcells in a linear array. In this case, fifteen subcells were tiled to compare strands of DNA up to eight elements long. Each strand to be compared is input one nucleotide at a time into a linear array from each side. Two comparisons are performed simultaneously by interleaving the inputs of each problem. Each input is represented by two bits as shown in table 1, though any encoding can actually be used. Once a strand has been completely entered into the array, a Null signal is set on a separate input on that strand's side of the array. Output from the two comparisons begins to appear from the array's left side on the sixteenth clock tick as a series of 1's and 0's. The outputs for the two comparisons are interleaved. The strands' genetic distance is determined by taking the length of the DNA strand input on the left side of the array, incrementing it for each 1 output, and decrementing it for each output of 0. A more detailed description of this circuit is given in Appendix A.

During each clock tick of the computation, each cell in the array needs to store both bits of the input from both directions, two bits for the computations, and the Null indicators from each direction. However, each cell only performs one comparison and two other small computations. The ratio of state information needed to logic is much larger for this design than for Life's implementation.

3.3. Additional Circuits

Ten additional circuits were implemented in standard cell, gate array, and FPGA technologies. Each implementation of these is derived from an initial Verilog design. Included in this group are a circuit performing a bubble sort, a palindrome detector with 128 nodes, a circuit that computes an 8-point 8-bit inverse discrete cosine transform (IDCT), a DES encryption circuit, an implementation of an R4000 processor, and five circuits from the PREP benchmark[1]. These circuits were chosen because they perform a large range of functions and because a Verilog implementation of each was readily available.

[1]The benchmarks and the R4000 implementation come from the PREP organization's test suite. They can be found on the internet at URL http://www.prep.org/.

The Programmable Electronics Performance Corporation, or PREP, developed their benchmark to compare FPGAs and their design tools. They are seeking to establish metrics by which the different programmable devices can be analyzed. Also they wish to compare the effectiveness of the CAD tools used with these devices.

3.3.1. **Bubble Sort**

The bubble sort circuit accepts eight 8-bit values as input, and outputs those values in sorted order. This circuit repeatedly uses a subcell that accepts two 8-bit inputs, compares them, and outputs them in sorted order. First this cell is called to compare the first two of the eight inputs. The larger value is then compared with the next input value. This is repeated until only the largest of the input values is left, which is stored as the first output. The entire process is repeated with the remaining seven inputs, and continues to be repeated until every input has been sorted and placed into an output.

3.3.2. **Palindrome**

The palindrome detector has an 8-bit input, a clock, a reset, and a 1-bit output. On each clock tick a new input is accepted. The output is set if there is a string of two or more consecutive inputs that ends with the most recent input and forms a palindrome (reading the values from first to last and from last to first results in the same sequence of values). This is recursively determined. Assume a string of length n is a palindrome. On the next cycle a new input is added to the end of the palindrome. If this input matches the value in front of the original n-length string, the new string of n+2 elements is another palindrome (see Figure 9). So each cycle, every string between 2 and 128 elements is analyzed. If the first and last elements of the string are equal, and the string that was two elements shorter last cycle was a palindrome, then the current string is also a palindrome, so a palindrome flag is set. Strings of length two and three are palindromes if their first and last elements match. So the input to each n length problems is the new input to the circuit, the last element from the n-1 length problem on the previous cycle (which becomes the last element for the n-length problem during this cycle), and the palindrome flag from the n-2 length problem from the previous cycle. Thus each sub-problem must output its last element and a flag that is set if its n elements form a palindrome. Since each sub-problem only needs to manipulate two elements and a flag, the logic is not

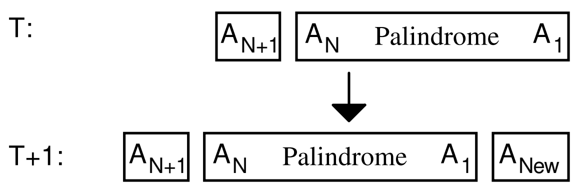

Figure 9. Recursive determination of palindromes. If the sequence from A_1 to A_N is a palindrome at time T, and A_{New} is the same as A_{N+1}, then there exists a palindrome at time T+1 of length N+2.

complex. The output of the palindrome circuit is true if any of the palindrome flags are set.

3.3.3. Inverse Discrete Cosine Transform

The discrete cosine transform (DCT) [1] and its inverse (IDCT) are both used extensively in image processing, such as in the compression standards of JPEG and MPEG[19]. The DCT converts a sequence of values to another representation that essentially concentrates the noticeable features of the original set of numbers into a smaller set of values. The less noticeable features can then be removed, saving storage space. When restored with the IDCT, the result is different than the original, but for the case of image data, its results are close enough for the human eye. Of course, if no data were removed, the IDCT would restore the image perfectly.

This implementation of the IDCT converts eight values of the transform back to eight original values. By definition, each of the original values is computed by adding all of the eight transform values together after scaling each transform value by a constant. This is a very routing intensive procedure. The particular algorithm used by this circuit was developed by Weiping Li [1].

3.3.4. R4000

The implementation of the R4000 processor used here was written by Joel Naumann, Amarpreet Chawla, and Micheal Povlin from IDT, Stratacom, and QuickLogic[2]. The R4000 is a 64-bit machine that uses the MIPS-III instruction set. This implementation is a 32-bit version containing eleven instructions from the MIPS-III instruction set architecture (ISA) consisting of two load/store, seven arithmetic, and two branch instructions. It has no pipelining or first-level cache. For more information on the MIPS instruction set, see [14].

3.3.5. DES

The Data Encryption Standard (DES) was endorsed by the National Bureau of Standards in 1977. It uses a 56-bit key to encode 64-bit blocks of data. There are three primary steps in DES: an initial permutation (IP), sixteen iterations of a function f, and an inverse to the initial permutation (IP^{-1}). At each iteration of f, a new key is input. For a detailed description see [4].

This circuit is used to aid a software version of a DES encoder/decoder. The circuit accepts a 64-bit input, a 56-bit key, and a mode flag. The mode flag determines whether the circuit computes IP, f, or IP^{-1}. The function f is largely made up of permutation tables, so this circuit is highly routing intensive with relatively little logic.

---

[2] A brief description of this implementation of the R4000 processor is given in the Verilog source available at the PREP web site at http://www.prep.org/testbnch/synth.htm#R4000

3.3.6. PREP Benchmarks

Prep1 runs four bytes of input through a 4-to-1 multiplexer to an 8-bit register. The output from the register is then sent to a shift register. The registers latch the data on a positive edge of a clock signal. A positive edge of a reset signal clears both registers to zero. Two selection bits are input for the multiplexer, and a shift/load input determines whether the shift register loads from the other register or left-shifts its current value. When shifting, the high bit is wrapped back to the low bit.

Prep2 contains two input registers, A and B, and one output register. Each register latches its input on a rising edge of the circuit's clock signal, and clears to zero on the rising edge of a reset signal. Two 1-bit inputs determine if each of the input registers load 8-bit input DATA2, or if they keep their current value. If the value of A does not equal the value of the output register, the output register is incremented. Otherwise, it is loaded from a multiplexer. Another 1-bit input signal determines if the multiplexer passes 8-bit input DATA1 to the output register, or if it passes the value of B.

Prep3 and Prep4 are state machines. Prep3 implements a small Mealy state machine with an 8-bit input, and 8-bit output, and eight states. The state encoding used is one-hot. Initially the circuit is in the START state, and it returns if a reset signal is asserted. If reset is not asserted, a rising clock edge sends the circuit to its next state which is determined by the input as shown in the state diagram in Figure 10. Prep4 is a Moore state machine with an 8-bit input, an 8-bit output, and sixteen states. Its underlying structure and functionality are similar to that of Prep3.

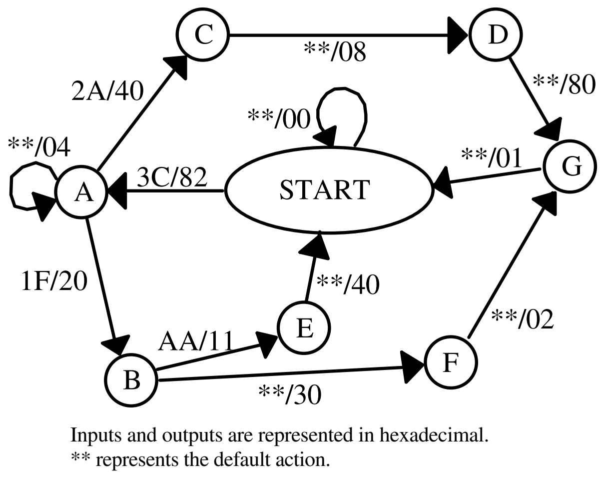

Figure 10. State transition diagram for the Prep3 benchmark circuit.

Prep9 is a memory mapping circuit. It takes as input two 8-bit address bytes which are combined to form a 16-bit address, and an address strobe (AS) signal. It has single bit outputs A through H representing eight output devices, and a bus error (BE) flag. The upper memory addresses of some computer architectures are used for I/O communication instead of traditional memory functions. This "memory" is divided into sections reserved for different I/O devices. When data is written to these memory addresses, it is sent instead to the external device. Similarly, reading these memory addresses accepts data from an input device. For this implementation, the upper memory is divided into eight sections, A through H. If the AS signal is true, this circuit determines which section the address falls into, and sets one of eight signals which corresponds to the appropriate section on the next rising clock edge. If the address does not fall into any of the eight sections, the BE signal is set.