Yen Pin Hsiao Derek Moran,
Ian Hung Matthew Mouyang,

ECE 427 PROJECT

1.0 DESIGN STRATEGY

The purpose of this project was to explore the digital design process through VHDL and the implementation of a high performance, 16-instruction, 16-bit, microprocessor called the TOY CPU on an APEX 20K200EFC484-2X FPGA. The goal for this design was to maximize operating frequency while minimizing the total number of logic elements and clock cycles.

The design strategy adopted to meet such demand in size and performance was to first achieve a functional implementation of the processor. Any further optimization would be difficult to trace for subsequent additions on a design that was not complete.

This project can be broken down into three functional units that tested independently of each other: the ALU, control, and memory. We will generate test benches for each unit and then put everything together. This approach gives a much more holistic view of the system and stabilizes our interfaces more quickly.

In addition, on another layer of implementation, the problem may be further subdivided into identifying all the necessary states and incorporating the logic in between such that proper functionality is achieved. The advantage of this strategy is the inherent convenience during optimization such that operations performed during each clock cycle may be tightly controlled and coordinated among each other.

Implementation followed a straightforward "waterfall" model, such that the simplest and most frequent functions would finish and quickly reiterate to FETCH, while those functions that required memory access would be separated and spread over successive execution states. This technique is illustrated in Figure 1-0 below.

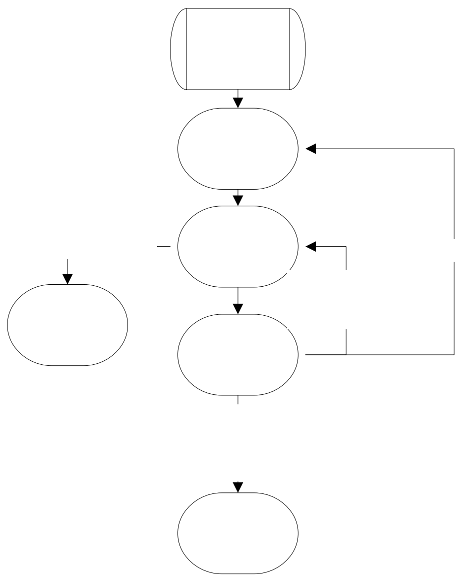

Figure 1-0: *Dataflow diagram to illustrate high-level functionality of our design before optimization*

An immediate problem with our implementation was the finite state machine that occurred beyond the main instruction process. According to one TA, an indirect clock would not infer registers; thus, synthesize the main process into a lengthy combinational path. This was easily corrected by clocking our main process. Other problems were minor (such as a carry bit setting a clock cycle too late, the Z flag not setting, etc.). These problems were introduced by mistake as we optimized, and then had to be later re-fixed.

2.0 PERFORMANCE RESULTS VERSUS PROJECTIONS

The original behavior prototype of our design contained many combinational loop back circuits. The registers such as the program counter (PC), address counter (A), and temporary register (T) turned out to be latches. These latches behaved as expected in our behavior simulation; however, they take longer time than flip-flops to get into the stable states, which limited our maximum frequency significantly. Compare to the average 50MHz maximum frequency in our class, our prototype could only run at 4.2MHz maximum. This result suggested us to re-exam our design and we found our Finite-State Machine was clocking the processor. We fixed this problem by using the clock to drive the processor directly. This became our new behavior prototype. Also, the behavior model contains the initial values assigned to the registers. These initial values were not synthesizable and they were replaced with a process that will be only executed at the startup or when the reset is pressed. Performance changes such as pre-fetching and grouping of similar blocks of VHDL are the only other differences.

Table 2-0: *Summary of CPI, Maximum Clock Speed, and Optimality Calculation*

Instruction	Clock cycle/Instruction	Instruction	Clock cycle/Instruction
JMP	5	**STT**	4
LDA	7	**OR**	7
LDI	5	**AND**	5
BEQ	3.5	**ILOG**	5
TAT	4	**LDC**	6
ADC	7	**SBC**	5
ROL	2	**OUT**	2
STT	3	**HLT**	4
# of cycles to complete the test bench:	91 cycles		
Total LE counts:	465 LEs		
Maximum clock speed:	59.5MHz		
Optimality:	1406.12076095947		

We aimed to get over 1300 score while balancing our optimization of area and performance at the outset of our design. We wanted over 40MHz, less than 525 LEs, and less than 90 cycles for the bench program. We were able to achieve all of our goals except the number of cycles. We were not able to get lower than 91 cycles.

Table 2-1: *LE Summary for Individual Components*

Component		LE count
TOY		465
-------	**SSDC**	16 (part of the 465)
RAM (synthesis)		72

Our design tries to achieve a balance between area and performance. Thus we did optimizations in each area. These optimizations are summarized as follows:

CLOCK CYCLE OPTIMIZATION

On closer inspection, EXECUTE2 could be merged with EXECUTE1 since its sole purpose was to assert write signals for OP_STT and OP_STA and read signals for OP_LDI and OTHERS for the next state. This would significantly reduce the number of clock cycles whenever memory is accessed.

Similarly, pre-fetching was employed for overlapping clock cycles. For instructions that resided in EXECUTE1, the next instruction could be fetched by asserting the RAM address signal within the current clock period. Moreover, for instructions that resided in EXECUTION2, that required prior memory to be read, the two cycles in between were utilized to fetch the next instruction. However, there was an exception for OP_STT/OP_STA since pre-fetching would overwrite the asserted ram address in their respective function.

LE OPTIMIZATION

We also noticed that the Z flag was modified within EXECUTE1 and EXECUTE2. To simplify this problem, the solution from the prior function was verified in an all-encompassing test within DECODE thus significantly reducing the number of logic elements synthesized. However, OP_STT was an exception because the value T, when 'factorized' along with the aforementioned test, reduced the maximum clock frequency since the Z-check was now convoluted. In addition, unnecessary signals and redundant coding were removed.

FREQUENCY OPTIMIZATION

We noticed that ILOG was in the critical path, and a new ILOG was designed that had a smaller fan out. This increased our fmax by several MHz. We also tried to share clock cycles wherever possible, which synopsis translated to frequency savings. We also experimented by removed any nested structures, in an attempt to reduce the length of the critical path. This was somewhat trial and error, but we were able to gain some frequency savings by this approach as well.

3.0 DISCUSSION OF VALIDATION PLAN, TEST CASES AND THEIR EFFECTIVENESS

We first of all compared test bench results to expected results for all of the sample assembly programs we were given, and the benchmark program. Then we vigorously tested the extreme cases that were most likely to cause problems. We programmed new assembly programs to generate these cases. For example, we made programs to test the wrapping behavior for ADC and SBC, the behavior of Z and C for various instructions for zero and non-zero instructions. Special programs were also written to specifically test the operations that read/write from memory. This proved to be very effective, as we found a lot of minor errors in the extreme cases. The problem was usually with the Z and C flags. The most interesting bug we found was with STT, which pre-maturely set a high Z flag low, and caused problems if the next instruction happened to be BEQ. This bug was hard to detect and unlikely to cause us immediate problems, although our testing of extreme cases found it.