EVALUATING LOGIC RESOURCES UTILIZATION IN AN FPGA-BASED TMR CPU

Ricardo Jasinski, Volnei A. Pedroni.

Federal Center of Technological Education of Parana – CEFET/PR, Curitiba-PR, Brazil.

1. Introduction

- Discussion about the impact of traditional Triple Modular Redundancy (TMR) design techniques in the implementation of a fault-tolerant, FPGA-based CPU;
- An existing CPU (the Xilinx PicoBlaze) had its VHDL source code adapted for the creation of a code-compatible, fault-tolerant CPU;
- Traditional TMR design techniques were applied to each of the CPU's building blocks, aiming at a circuit without single points of failure;
- The redundant version was verified through simulation, and then tested in actual FPGAs;
- Register elements utilization was found to be 3.1 times that of the original CPU, while the usage of logic cells was increased by a factor of 4.6;
- Both CPUs had their operation verified at up to 70 MHz, and only a small decrease in the maximum operating frequency was experimentally observed in the TMR version.

2. Traditional TMR

- The applied techniques were proposed by Xilinx (C.Carmichael) for the protection of FPGA designs - "*Triple Module Redundancy Design Techniques for Virtex Series FPGA - Application Note 197*";
- TMR has been used successfully for the protection of FPGA designs destined to operation in radiation environments. Combined with configuration memory scrubbing, can lead to designs virtually immune to the effects of SEUs;
- Each block of logic must be protected by the adequate technique. Combinational, sequential and memory circuits have specific requirements that must be taken into account when choosing the protection strategy.

A. Combinational Logic

- Any circuit where all paths propagate directly from inputs to outputs (no logic loops);
- Also includes circuits in which the result takes more than one clock cycle to be computed;
- TMR protection strategy: simple triplication of the module (Fig. 1);
- Conclusion: purely combinational logic requires no change in original source files; simply create a new, TMR module and instantiate 3 times the original module.

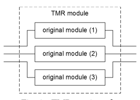
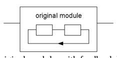


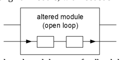
Fig. 1 - TMR version of a combinational module.

B. Sequential Logic

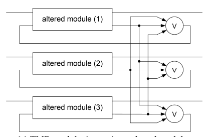
- Bit-flips can be "caught" within logic loops - requires additional care;
- TMR protection strategy:
  1. the original module must be altered, opening any feedback loop (Fig. 2a and 2b);
  2. one voter circuit is inserted in each feedback loop (Fig. 2c);
  3. a new top entity is created, containing 3 copies of the altered module, and as many voter circuits as feedback paths in the original module.
- Conclusion: sequential modules require changes in original source code. In addition, voters must be added within the new TMR entity, one for each feedback path.



(a) original module, with feedback loop



(b) altered module, open feedback loop

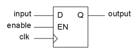


(c) TMR module, instantiates altered modules

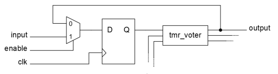
Fig. 2 – Application of TMR for sequential logic modules.

C. Registers with Enable Input

- Might have to maintain stored data for arbitrarily long periods of time, while data input is disabled (period is application-dependant);
- TMR protection strategy: substitution of such registers by self-refreshing register (Fig. 3). In the structure of Fig. 3b, while the enable input is kept low, the stored data is constantly voted between 3 redundant registers and reapplied to the flip-flop's input.



(a) original register, with *enable* input

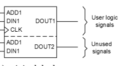


(b) altered register, contents refreshed at each clock cycle

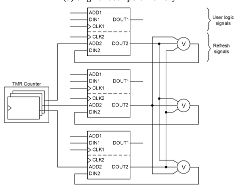
Fig. 3 – Protection of flip-flops with *enable* input.

D. Memories

- Must avoid error-build-up in stored data;
- Might have to maintain stored data for arbitrarily long periods of time (again, period is application-dependant);
- TMR protection strategy: automatic refresh circuitry (TMR counters + voters). Port A of dual-port memory block is utilized by user logic; port B is dedicated to refreshing data stored in all memory positions, without disrupting the circuit's operation.



(a) original dual-port memory



(b) TMR single-port memory; second port is dedicated to refresh of the stored data,

Fig. 4 – TMR Memory with automatic refresh.

3. The Original CPU

- Chosen CPU: Xilinx PicoBlaze;
- Reasons that motivated the choice: low complexity, source code in VHDL, documentation, existent SW development tools, ease of customization;
- HW details: 8-bit customizable CPU, up to 32 8-bit registers, up to 1024 program memory words, up to 31-level deep HW stack (Fig. 5 shows one specific configuration);
- Device utilization as low as 76 slices (Virtex-E/Spartan-II);
- Synthesizable description: set of 15 VHDL files; source code can be downloaded free of charge from www.xilinx.com/picoblaze.

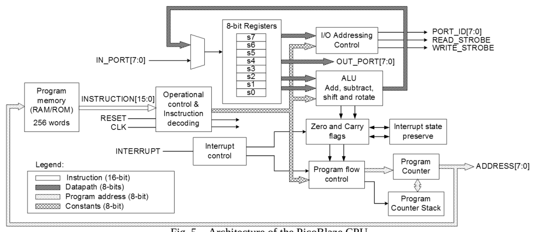


Fig. 5 – Architecture of the PicoBlaze CPU.

4. Adaptation Process

- From the original source code in VHDL, a new, functionally identical CPU was created, with each block protected by means of TMR;
- Special emphasis was given to the reuse of existing code;
- Boundaries between source files were always preserved (i.e., the basic block for the implementation of TMR is a source file);
- Adaptation steps:
  1. identification of all of the CPU building blocks;
  2. classification according to the type of logic (A, B, C, or D, as in item 2);
  3. application of the adequate protection strategy;
  4. creation of a new top entity (picoblaze_tmr), comprising all of the mitigated blocks.

5. Logic Resources Estimate

- The estimate of logic resources needed for the TMR version is relevant for two important reasons: first, it can assist the choice of an adequate device for the resulting circuit; second, it can be used as a control parameter during the adaptation process;
- Each block is synthesized, and checked against estimated values; if the redundant module uses fewer resources than estimated, this can be a sign that the compiler is performing an unintended removal of redundant logic;
- Two kinds of logic resources are considered:
  1. the number of registers, and
  2. the number of logic cells (each logic cell consists of a 4-input LUT plus a register).

- The estimated number of registers is a very meaningful parameter; an optimal synthesis tool should never use more registers than necessary, and a correct synthesis cannot implement the desired functionality with less than a certain number of storage elements. For these reasons, this estimate can be very accurate;
- The number of logic cells (LCs) is important because they are the basic building blocks of the implemented designs. Basic structures comprising one 4-input look-up table (LUT) plus a register are present in most of today's FPGAs. Most of the recent devices name this basic structure *logic cell*.
- Table 1 summarizes the formulae used for the logic resources estimates.

Table 1 – Registers and Logic Cells estimates.

Type of logic	Number of Registers	Number of Logic Cells
Purely combinational	$NR_{TMR} = NR \times 3$	$LC_{TMR} = LC \times 3$
Combinational with *enable*	$NR_{TMR} = NR \times 3$	$LC_{TMR} = (LC + N_{FF}) \times 3$ (mín.)
Sequential	$NR_{TMR} = NR \times 3$	$LC_{TMR} = (LC + N_{SE}) \times 3$
Memory	$NR_{TMR} = (NR + \log_2 M) \times 3$	---

where:	
	NR = number of registers before use of TMR
	NR_{TMR} = number of registers after use of TMR
	M = number of memory positions
	LC = number of logic cells before use of TMR
	LC_{TMR} = number of logic cells after use of TMR
	N_{FF} = number of flip-flops with *enable* input
	N_{SE} = number of state storage elements

6. Experimental Results

- The TMR version of the CPU was verified through simulation, followed by actual implementation and testing in commercial FPGAs (Xilinx Virtex-II and Altera APEX20KE). Synthesis results described below were obtained with an EP20K200EFC484-2X device, using Quartus II 4.0 compiler and synthesizer.
- No specific attempt was made to improve the timing performance or reduce the logic elements usage. All compiler settings were kept at the default values. The data presented in Tables 2 and 3 were gathered in a single compilation.

A. Logic Resources Utilization

- Table 2 shows a comparison of logic resources utilization between the original CPU and the TMR version, as well as the estimated number of registers and logic cells required by each block;
- As can be seen, all but one of the estimates were 100% accurate. As regards the number of registers, all estimated values were verified;
- Regarding the number of logic cells, the only module that presents a value different than the estimated is the interrupt_logic block, which is a combinational circuit with enable inputs. As previously mentioned, in such cases the estimative is given as a minimum value, and therefore all values found are in agreement with the proposed resources estimation methodology.

Table 2 – Synthesis results × estimated values.

	Registers			Logic Cells		
Module name	Original	Estimated	With TMR	Original	Estimated	With TMR
arithmetic	9	27	27	27	81	81
carry_flag_logic	1	3	3	4	15	15
interrupt_capture	2	6	6	3	9	9
interrupt_logic	3	9	9	5	24	30
IO_strobe_logic	2	6	6	2	6	6
logical_bus_processing	8	24	24	8	24	24
program_counter	8	24	24	52	180	180
register_and_flag_enable	3	9	9	5	15	15
register_bank	64	201	201	146	--	813
T_state_and_Reset	3	9	9	3	12	12
shift_rotate	9	27	27	16	48	48
zero_flag_logic	1	3	3	5	18	18
stack_counter	2	6	6	5	21	21
stack_ram	40	126	126	52	--	351
picoblaze (at entity level)	0	0	0	57	171	171
Total	155	480	480	390	--	1794

- Table 3 summarizes the increase in resources usage, according to the type of logic;
- As can be seen, the intuitive notion that the TMR design uses approximately three times as many resources as the original design is verified only for the number of registers (with a 3.1 TMR/non-TMR ratio);
- For the number of logic cells, which are the really meaningful resources when choosing a device for implementation, the results indicate a 4.6 TMR/non-TMR ratio.

Table 3– Increase in resources usage by type of logic.

	Registers			Logic Cells		
Type of Logic	Without TMR	With TMR	Ratio	Without TMR	With TMR	Ratio
Pure Combinational	33	99	3.0	118	354	3.0
Combinational w/ Enable	5	15	3.0	14	63	4.5
Sequential	13	39	3.0	60	213	3.5
Memory	104	327	3.1	198	1164	5.9
Total	155	480	**3.1**	390	1794	**4.6**

B. Maximum Operating Frequency

- A simple test was conducted with the aid of a frequency multiplier circuit (PLL) available within the FPGA;
- Both CPUs were exercised with a specific test program, in order to verify their functioning at various frequencies. Starting at 50 MHz and increasing in steps of 5 MHz until the frequency of 70 MHz, both CPUs executed all operations correctly;
- At the frequency of 75 MHz, the TMR CPU ceases to work, while the original CPU still executes all tests, but produces erroneous results for some logical and arithmetic operations. At 80 MHz, neither of the CPUs can run the test program;
- These results show only a small decrease in the maximum operating frequency of the TMR version, when compared to the original CPU;
- The frequency at which this difference is detected is compatible with the delay values introduced by the voting circuits (in the order of a few nanoseconds). In addition to the delay introduced by the voters, when a module is triplicated, it often cannot be fit in the same group (or groups) of LCs, and thus can no longer benefit from the fast local routing connections. In this case, the penalty is a decrease in the system's maximum operating frequency.

7. Conclusions

- A methodology for the adaptation of an existing CPU into a TMR, fault-tolerant design has been proposed and evaluated. The objective of creating a fault-tolerant CPU which is still code compatible with the original version was achieved;
- The adaptation process was presented, which is based on classical TMR protection schemes. One interesting benefit is that no information about the logic modules is needed, besides the kind of logic implemented. It means that it is possible to implement TMR versions of existing designs, without a full understanding of its architectural details.
- The developed method for the estimation of logic resources can be used to assist device assignment in future TMR projects, instead of empirical or statistical estimates.