Implementation of a Triple Modular Redundant FPGA based Safety Critical System for reliable software execution

Venkatesh Vasudevan Email:venkat@itee.uq.edu.au
Peter Waldeck Email:waldeck@itee.uq.edu.au
Hardik Mehta Email:mehta@itee.uq.edu.au
Neil Bergmann Email:bergmann@itee.uq.edu.au

School of Information Technology and Electrical Engineering
University of Queensland

Abstract

This paper describes the implementation of a TMR (Triple Modular Redundant) microprocessor system on a FPGA. The system exhibits true redundancy in that three instances of the same processor system (both software and hardware) are executed in parallel. The described system uses software to control external peripherals and a voter is used to output correct results. An error indication is asserted whenever two of the three outputs match or all three outputs disagree. The software has been implemented to conform to a particular safety critical coding guideline/standard which is popular in industry. The system was verified by injecting various faults into it.

1 Introduction

1.1 Background

Field Programmable Gate Arrays (FPGA's) are semiconductor integrated circuits (IC's)/chips that facilitate custom user logic to be programmed using a bitstream. The devices can be reprogrammed in the field whenever the logic changes thus removing the need to remove the device or design a new system. Hence the product can be upgraded in the field with new features without any replacement of parts. The whole system (processor and its peripherals) can be housed in a single FPGA thus reducing board size considerably. Time to market or development time is considerably reduced due to rapid prototyping. FPGA's allow a software-hardware co-design methodology which is a must for safety critical applications and hence it is convenient to use FPGA's for the development of the same.

1.2 Objective

Triple Modular Redundancy (TMR)(B.W.Johnson 1989) is a popular concept being used by many designers of high reliability systems. The common methodology involves development of software conforming to safe coding guidelines (viz one may use a language like Esterel (G.Berry et al 2000) to define the specifications of the system and then implement it or if coding in traditional programming languages like C, follow guidelines set out by subsets of the languages like MISRA-C(MISRA 2004)) and implementation of the same on three separate microprocessors (each on its own motherboard). Our scheme followed the same strategy. However the three microprocessor systems were implemented on one FPGA. The goal was to achieve reliability in software execution by employing hardware redundancy so that if one of the software fails due to a hardware fault (bitstream / configuration errors, stuck at faults, bit flips due to radiation etc) then the other softwares running in parallel can keep the system in operation. This is due to the fact that the hardware error occurs only in a part of the FPGA and hence only a small part is affected, not the whole FPGA. The correctness of the software was tested by debugging it using XMD (Xilinx Microprocessor Debugger) which allows the user to single step through the code. The software was included in the system after thorough testing. The software was written in C and checked for MISRA-C(MISRA 2004) rules by Abraxas Software's CodeCheck v1300 B1 tool (results in System Verification section). The code was for a seconds counter and instances of the same code were allowed to run on each of the three microprocessors. The design was targeted for a Xilinx FPGA and hence Xilinx design flows.

1.3 Literature Review

Other teams have worked on TMR systems on FPGA's. Bitstream faults were investigated by (L.Carro et al 2005) and results were based on number and placement of voters. (Rami Melhem et al 2002) have performed analysis of energy efficiency of Duplex and TMR systems. (Hyunki Kim et al 2002) have developed a TMR system based on MC68000 (microprocessor from Motorola). A number of other works were reviewed and it was found that little work has been done on implementing TMR software systems on FPGA's with a focus on improving software reliability on FPGA's. Our team was and is focussed on reliable software execution on FPGA's for implementing FPGA based Safety Critical Systems. Hence we have implemented a system which can execute its software reliably during various hardware faults.

2 System Design

2.1 The System

As shown in Figure 1 the system is composed of three Microblaze processor systems and the deciding voter. Each processor system (Figure 2) is a whole system in itself in that it contains the following essential components:

- Microblaze or PowerPC microprocessor (in this case Microblaze)

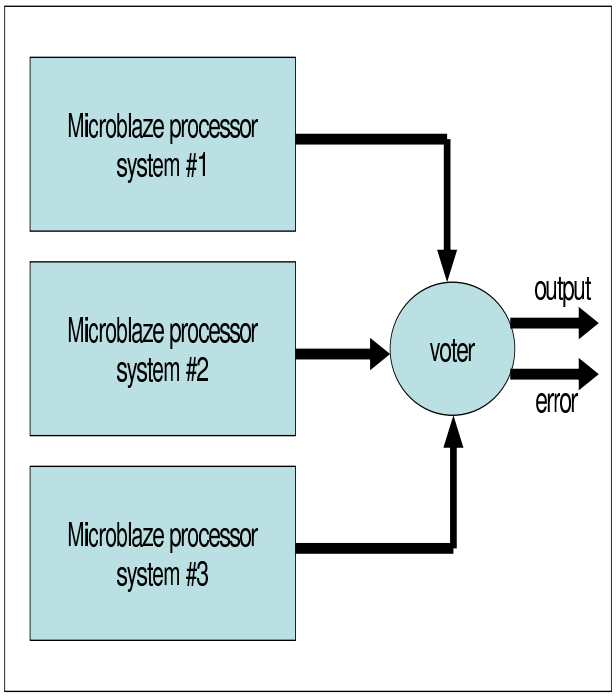

Figure 1: The System

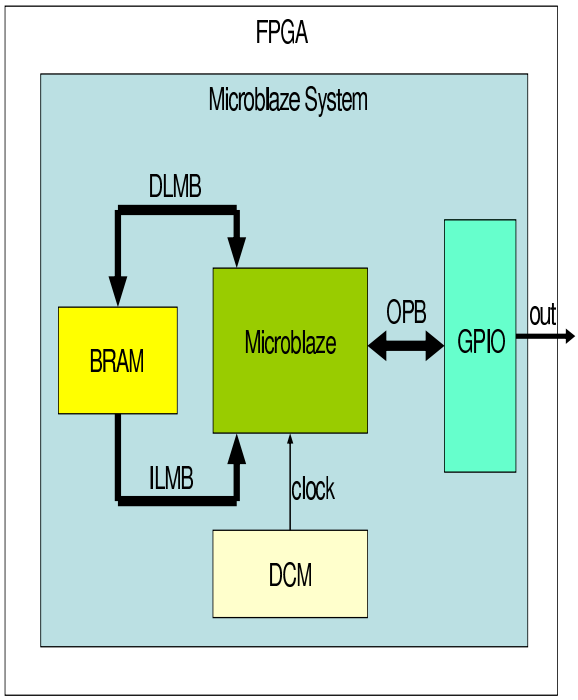

Figure 2: Microblaze System

- Program Memory (in this case Block RAM)
- Data Memory (in this case Block RAM)
- Local Memory Bus (LMB) for both instruction side and data side and associated controllers
- OnChip Peripheral Bus (OPB)
- Digital Clock Manager (DCM) and
- Peripheral (in this case GPIO (general purpose I/O)

For the above shown system, the voter peripheral and GPIO peripherals (Figure 2) were created using 'Create - Import Peripheral' a tool available in Xilinx EDK (Embedded Development Kit) for creating custom peripherals (that are not otherwise available in EDK) and attaching to the processor via either OPB(OnChip Peripheral Bus),PLB(Peripheral Local Bus) or FSL(Fast Simplex Link). In this case the peripherals were designed to interface to the OPB of Microblaze. This whole system was housed in a Virtex 4 (XC4VLX25) FPGA on the ML401 Xilinx evaluation board.
The whole system was developed using the EDK design flow by manually creating the MHS (Microprocessor hardware specification) and MSS (Microprocessor software specification) files. This means that these files were not automatically generated by EDK which is what generally happens during development. In our case we had to write out these files using the syntax for MHS and MSS files since our system could not be directly generated by EDK. These files are the deciding factors in Xilinx EDK that determine the hardware synthesized and the software libraries generated.
All the code and data resided in internal BRAM (block RAM) blocks (3 in number)
The GPIO peripherals interfaced to the microprocessor via the OPB
The voter peripheral however even though designed for interface via OPB was not interfaced to the CPU rather it was interfaced only to the GPIO outputs (inputs to the voter) and external outputs (the output or result itself and error output). The external outputs were led's driven by FPGA I/O's.

2.2 Software

Three instances of the same code were created as shown below (incorporating fault injection) conforming to MISRA-C(MISRA 2004) rules. Due to the presence of three code instances and identical hardware architectures, true concurrency of execution of operation was obtained.
The code in this case is for a simple counter whose two least significant bits are output to the voter. The voter in turn outputs the two bit value to LED's on the board. The header file 'xparameters.h' contains definition of the base address of the GPIO (general purpose input output) whose output is connected to the voter input. The header file 'vgpio.h' contains function prototypes of GPIO read/write functions. It also includes the 'xbasic_types.h' header file which contains typedef'd data type definitions e.g Xuint8. The counter runs at a speed of 1 Hz due to the fact that the external clock frequency is 100 MHz and the number of clock ticks that are counted are also the same number (ledX_delay where X=0,1,2).
The code is shown to demonstrate how safe code was written and how we plan to write it in future (e.g no use of pointers, use of macros, adequate function prototypes etc). The three codes although the same were different when testing for faults namely stuck at faults. For example (see fault injection in code) the GPIO's were stuck at either 0x00 or 0xff for one of the codes with the remaining codes being intact.

```
Microblaze System #1
//counter0.c
#include "xparameters.h"
#include "vgpio.h"

#define led0_base XPAR_VGPIO_0_BASEADDR
#define led0_delay 100000000
#define led0_offset 0
void delay(void)
int main()
{
Xuint8 led0_data;
```

```
led0_data = 0x00;
while(1)//GPIO write
{
//Stuck at fault injection
//VGPIO_mWriteReg(led0_base,led0_offset,0xff);
//VGPIO_mWriteReg(led0_base,led0_offset,0x00);
VGPIO_mWriteReg(led0_base,led0_offset,led0_data);
delay();
led0_data++;
}

return 0;
}

void delay(void)
{
        Xuint32 led0_delay_value;
        led0_delay_value = 0x00;
        while(led0_delay_value < led0_delay)
        {
                led0_delay_value++;
        }
}
```

Microblaze System #2

```
//counter1.c
#include "xparameters.h"
#include "vgpio.h"

#define led1_base XPAR_VGPIO_1_BASEADDR
#define led1_delay 100000000
#define led1_offset 0
void delay(void)
int main()
{
Xuint8 led1_data;
led1_data = 0x00;
while(1)//GPIO write
{
//Stuck at fault injection
//VGPIO_mWriteReg(led1_base,led1_offset,0xff);
//VGPIO_mWriteReg(led1_base,led1_offset,0x00);
VGPIO_mWriteReg(led1_base,led1_offset,led1_data);
delay();
led1_data++;
}

return 0;
}

void delay(void)
{
        Xuint32 led1_delay_value;
        led1_delay_value = 0x00;
        while(led1_delay_value < led1_delay)
        {
                led1_delay_value++;
        }
}
```

Microblaze System #3

```
//counter2.c
#include "xparameters.h"
#include "vgpio.h"

#define led2_base XPAR_VGPIO_2_BASEADDR
#define led2_delay 100000000
#define led2_offset 0
void delay(void)
int main()
{
Xuint8 led2_data;
led2_data = 0x00;
while(1)//GPIO write
{
//Stuck at fault injection
//VGPIO_mWriteReg(led2_base,led2_offset,0xff);
//VGPIO_mWriteReg(led2_base,led2_offset,0x00);
VGPIO_mWriteReg(led2_base,led2_offset,led2_data);
delay();
led2_data++;
}

return 0;
}

void delay(void)
{
        Xuint32 led2_delay_value;
        led2_delay_value = 0x00;
        while(led2_delay_value < led2_delay)
        {
                led2_delay_value++;
        }
}
```

For the above code instances, the libraries were generated by EDK after parsing MHS and MSS files. New drivers were created by the tool for the user created peripherals viz GPIO and voter. However only driver for the GPIO peripheral is being used.

2.3 Peripherals

This section explains design of peripherals and the design methodology employed as it impacts the ease of development. As has already been pointed out the GPIO and voter peripherals were created by the team using 'Create - Import Peripheral' a utility that is shipped alongwith Xilinx EDK. Of course one can use Xilinx provided GPIO cores as well. Development of our own GPIO peripherals happened as a result of issues related to the device driver. The voter peripheral was created due to our adopted design methodology. The design phase can actually have two different design flows. One is the Xilinx ISE (Integrated System Environment) flow (Figure 3) and the other is the EDK flow (Figure 4). Looking at Figure 1, one might say that the ISE flow looks like the logical one since the voter can be entirely developed in ISE (VHDL RTL) and the microprocessor file developed in EDK can be included in the ISE project and both the entities can be instantiated within a top level entity thus completing the entire system. This design flow works allright if only the microprocessor file is included and synthesized in ISE. It can be made to work with the VHDL and uP files instantiated together but this involves tinkering which goes very deep into the Xilinx files.
Hence we decided to take the EDK design flow methodology. Obviously since this is not ISE a direct implementation of the voter was not possible. Hence we had to create our own voter peripheral the discussion of which is carried forward in the subsections. The case of our own GPIO peripherals is also discussed in the following subsection.

2.3.1 VGPIO

The GPIO core is not the standard Xilinx GPIO core. Its called VGPIO which was developed by us for purposes described in the following.
The Xilinx GPIO is a complex core with certain functionalities that we did not require and the device driver too has complex usage in that it has to be initialized and configured the correct way so that it becomes suitable for our system. Our GPIO, the VGPIO has a simple device driver and one can

immediately write to or read from it without any special initialization and configuration.

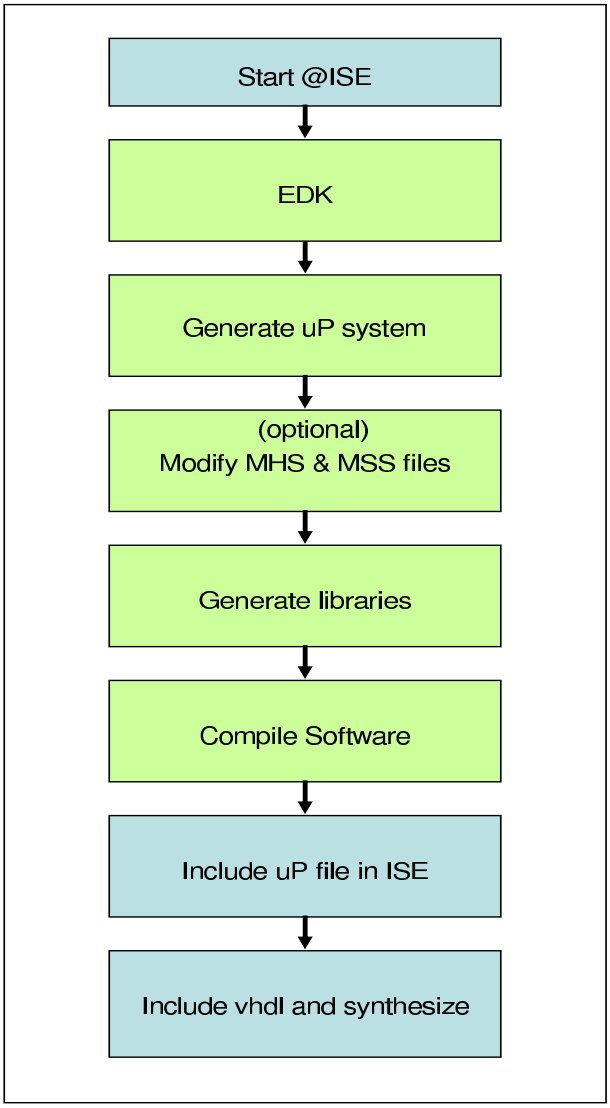

Figure 3: ISE Design Flow

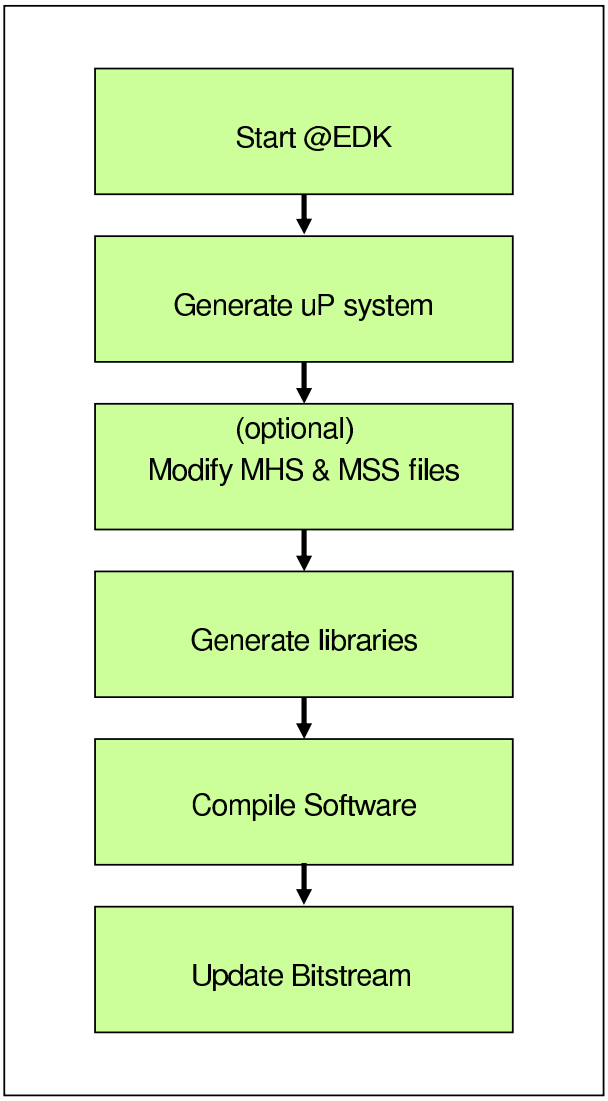

Figure 4: EDK Design Flow

Figure 5 shows the architecture of our GPIO. As shown it's a simple 32 bit register (for simplicity since Microblaze is 32 bit) featuring big endian (as Microblaze follows big endian) format.

- VGPIO device driver
  EDK builds libraries containing drivers for each peripheral in the design after parsing the MHS and MSS files. In case of the VGPIO peripheral we used the basic write function which was provided by default for writing into the VGPIO register.

Before implementing the final system (Figure 1) we implemented the system as shown in Figure 6. This scheme had timing issues in that the outputs of VGPIO's (connected to inputs of voter) would arrive at the same time however the voter would have its third input (coming from Microblaze #2) much earlier thus only two inputs to the voter would match resulting in error LED turned on.

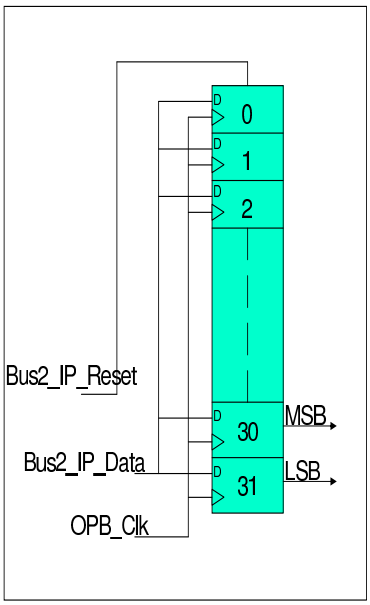

Figure 5: VGPIO Architecture

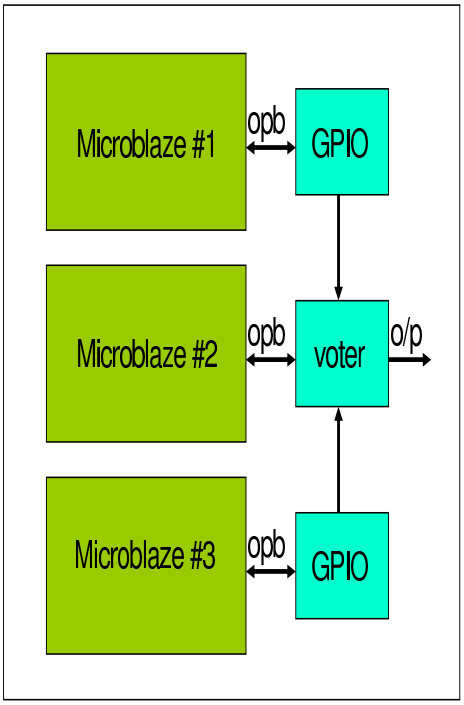

Figure 6: Initial System

2.3.2 Voter

The voter architecture is basically that of a comparator which looks at three inputs and checks for various combinations of inequalities. The voter too was built using 'Create - Import Peripheral' though it wasn't interfaced to any Microblaze. It contained both the OPB interface hardware as well as the comparator hardware both of them working in parallel without any connection whatsoever. We had to opt for this approach as the solution was the EDK design flow and hence we needed the voter to be available in EDK so that we could include it in our MHS description. The following VHDL snippet describes the voter architecture. Note that the OPB part has been omitted as it is not required to be shown.
The VHDL code for the voter (a 2 bit module) basically compares the three inputs with each other and passes that value which appears on at least two inputs to the output. An error is generated if either only two inputs match or all inputs mismatch. In the event where all inputs mismatch the output is made "00". The inputs can assume values "00","01","10" or "11" hence the output can assume these same values. Even if only two inputs match, the voter will still output the value keeping the system in operation but at the same time it indicates to the personnel that an internal error has occured.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

entity voter is
port(voter_in0:std_logic_vector(0 to 1);
voter_in1:std_logic_vector(0 to 1);
voter_in2:std_logic_vector(0 to 1);
voter_out:std_logic_vector(0 to 1);
sys_error:std_logic);
end entity voter;

architecture voter_arch of voter is
signal voter_in0_store:std_logic_vector(0 to 1);
signal voter_in1_store:std_logic_vector(0 to 1);
signal voter_in2_store:std_logic_vector(0 to 1);
begin
voter_action : process( voter_in0_store,
                        voter_in1_store,
                        voter_in2_store ) is
begin
if(voter_in0_store = voter_in1_store) and
(voter_in0_store /= voter_in2_store) then
voter_out_store <= voter_in0_store;
sys_error_store <= '1';
elsif(voter_in0_store = voter_in2_store )
and (voter_in0_store /= voter_in1_store) then
voter_out_store <= voter_in0_store;
sys_error_store <= '1';
elsif(voter_in1_store = voter_in2_store ) and
(voter_in1_store /= voter_in0_store) then
voter_out_store <= voter_in1_store;
sys_error_store <= '1';
elsif(voter_in0_store = voter_in1_store) and
(voter_in1_store = voter_in2_store ) then
voter_out_store <= voter_in0_store;
sys_error_store <= '0';
else
voter_out_store <= "00";
sys_error_store <= '1';
end if;
end process voter_action;
voter_out <= voter_out_store;
sys_error <= sys_error_store;
end architecture voter_arch;
```

As shown in the above code the voter looks for different match possibilities and asserts an error if only two inputs match or all three inputs don't match. Figure 7 shows the voter architecture.

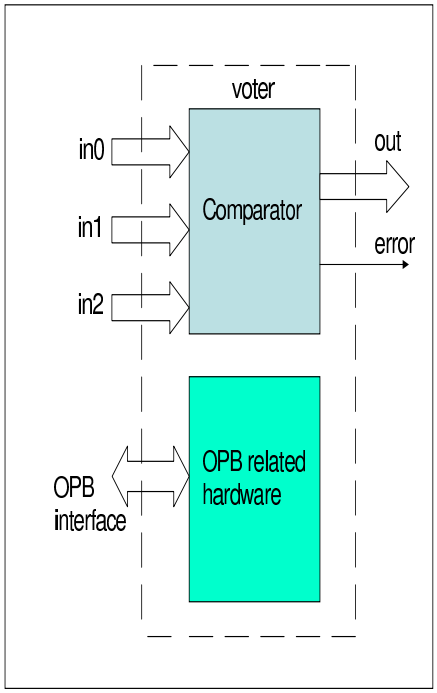

Figure 7: Voter Architecture

As shown the voter has two distinct parts, the comparator which performs the voting action and the dummy OPB part used only to get the voter peripheral included in the MHS file via the 'Add Edit cores' option in EDK. The comparator is purely combinational in nature for now and may be made sequential in future. Following is the MHS code snippet to illustrate how the voter was interfaced.

```
BEGIN voter
 PARAMETER INSTANCE = voter_0
 PARAMETER HW_VER = 1.00.a
 PORT voter_in0 = voter_sig0
 PORT voter_in1 = voter_sig1
 PORT voter_in2 = voter_sig2
 PORT voter_out = sys_out_io
 PORT sys_error = sys_err_io
END
```

As shown the voter is interfaced only to the inputs and outputs. Voter_sig0, voter_sig1 and voter_sig2 are internal connections connecting the VGPIO outputs to the voter inputs. Similarly sys_out_io and sys_err_io are connections to the outputs.

3 System Verification

3.1 MISRA-C compliance

MISRA-C(MISRA 2004) sets out guidelines for the use of C language in safety critical systems. Our software was checked for MISRA-C(MISRA 2004) rules by Abraxas Software's CodeCheck MISRA-C(MISRA 2004) rules checker. The tool basically parses the user's C code and lists warnings corresponding to each MISRA-C rule that has not been satisfied in the code. Following is the results obtained by the rules checker.

```
Abraxas Software (R) CodeCheck
Windows Version 1300 B1 DEMO
Copyright (c) 1988−2006, by
Abraxas Software Inc.
All rights reserved
Checking extended ANSI C file
counter0.c with rules from misra04.cc

counter0.c(9): Warning W0076: counter0.c(9):
Rule 76 (REQUIRED) 16.5 Functions with no
parameters shall be declared with parameter
type void.
counter0.c(9): Warning W0071: counter0.c(9) :
Rule 71 (REQUIRED) 8.1 Functions shall always
have prototype declarations. {DEFN}
counter0.c(9): Warning W0074: counter0.c(9) :
Rule 74 (REQUIRED) 16.4 identifiers given for
any of the parameters decl and/or defn must
be same
counter0.c(14): Warning W0071: counter0.c(14) :
Rule 71 (REQUIRED) 8.1 Functions shall always
have prototype declarations. {CALL}
File counter0.c check complete.
```

As shown above the applicable rules have been pointed out by the checker wherever required and we have verified our code for potential hazards at these points. For example line 9 warning W0076→This means that MISRA-C rule number 76 has not been satisfied in the code and so on. Similar results were obtained for the files counter1.c and counter2.c
The warnings with the level 'REQUIRED' have been taken care of but they appear because some part of the code is in header files which were not included (they were initially included which gave rise to warnings due to the presence of other header files which was not an indication of non safety critical code) during CodeCheck run. For example rule 71 at line 14 says that functions shall always have prototype declarations which has been implemented in the header files. Hence the warnings are not serious and the code can be considered to comply with MISRA-C guidelines.

3.2 Microblaze compiler output

The Microblaze gcc compiler output for each of the software files (counter0.c,counter1.c,counter2.c) has been shown. The fact that the compilation took place without errors is justified due to the presence of the size of code (hex 6ac,6ac,6ec) and executable.elf file for each of the three files. The result also shows the code and data memory map used viz for the first code instance (counter0.c) the program memory and data memory start at 0x0000,
for the second code instance (counter1.c) the program memory and data memory start at 0x4000 and
for the third code instance (counter2.c) the program memory and data memory start at 0x8000.
Following is the Microblaze compiler (mb−gcc) output:

```
At Local date and time: Mon Apr 24 15:55:37 2006
Command xbash −q −c "cd /cygdrive/d/aSCSa1/;
/usr/bin/make −f system.make program; exit;"
Started...

mb−gcc −O2 microblaze_0/code/counter0.c −o
counter0/executable.elf \
−Wl,−defsym −Wl,_TEXT_START_ADDR=0x0000
−mno−xl−soft−mul −g −I./microblaze_0/include/
-L./microblaze_0/lib/ \
−xl−mode−executable \
mb−size counter0/executable.elf
text data bss dec hex filename
664 12 1032 1708 6ac counter0/executable.elf
mb−gcc −O2 microblaze_1/code/counter1.c −o
counter1/executable.elf \
−Wl,−defsym −Wl,_TEXT_START_ADDR=0x4000
−mno−xl−soft−mul −g −I./microblaze_1/include/
−L./microblaze_1/lib/ \
−xl−mode−executable \
microblaze_1/code/counter1.c:30:2: warning: no
newline at end of file
mb−size counter1/executable.elf
text data bss dec hex filename
664 12 1032 1708 6ac counter1/executable.elf
mb−gcc −O2 microblaze_2/code/counter2.c −o
counter2/executable.elf \
−Wl,−defsym −Wl,_TEXT_START_ADDR=0x8000
−mno−xl−soft−mul −g −I./microblaze_2/include/
−L./microblaze_2/lib/ \
−xl−mode−executable \
microblaze_2/code/counter2.c:30:2: warning: no
newline at end of file
mb−size counter2/executable.elf
text data bss dec hex filename
728 12 1032 1772 6ec counter2/executable.elf
Done.
```

3.3 Fault Injection

Various hardware faults were emulated both in software and hardware. Stuck at faults were emulated by inserting fault injection code in software (see software section). Stuck at fault means that the nodes are stuck either at logic high or low due to a configuration / bitstream error or fabrication defects or hardware design fault. Bit flip faults were emulated by inserting 'bit flip fault' code in VGPIO VHDL file (code hasn't been shown as file is too large to be included).

3.4 Board level Verification

The system was run on ML401 Xilinx Evaluation board by tying the outputs to LED's. Following is a listing of the user configuration file (UCF) created in

EDK.

```
Net sys_clk_pin LOC=AE14;
Net sys_clk_pin IOSTANDARD = LVCMOS33;
Net sys_rst_pin LOC=D6;
Net sys_rst_pin PULLUP;
## System level constraints
Net sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin
10000 ps;
Net sys_rst_pin TIG;

## FPGA pin constraints
Net sys_out_pin<0> LOC=E2;
Net sys_out_pin<0> IOSTANDARD = LVCMOS25;
Net sys_out_pin<1> LOC=E10;
Net sys_out_pin<1> IOSTANDARD = LVCMOS25;
Net sys_err_pin LOC=A5;
Net sys_err_pin IOSTANDARD = LVCMOS25;
```

The above listing shows connections of various signals (clk,rst,sys_out_pin(0),sys_out_pin(1),sys_err_pin) to FPGA pins (AE14,D6,E2,E10,A5) respectively. The constraints applied also show the I/O standards in effect, for example the sys_clk_pin has an I/O standard LVCMOS33 which means Low Voltage CMOS 3.3V. Similarly the sys_out_pins and sys_err_pin have I/O standard LVCMOS25 (Low Voltage CMOS 2.5V). The sys_rst_pin has been pulled up to Vcc (power viz logic high) since on the board the reset (sys_rst_pin) is active low (meaning that the system is reset when sys_rst_pin is logic low). This basically means that when reset button is not pressed the sys_rst_pin will be logic high and when reset button is pressed it will be logic low. The system level constraints have three commands namely TNM_NET, TIMESPEC and TIG whose explanation is as follows :
TNM_NET means that sys_clk_pin is to be used in a timing specification.
TIMESPEC defines the clock period viz 100MHz.
TIG means that sys_rst_pin is to be ignored for a timing specification.

4 Conclusion

A triple modular redundant technique for reliable software execution in the event of hardware faults adhering to MISRA-C(MISRA 2004) rules was implemented and verified on a Virtex 4 FPGA (XC4VLX25) on the ML401 Xilinx Evaluation Board. The technique exhibited true concurrency in behaviour and operated correctly for long periods of time. This TMR work is one step in our investigation of reliable FPGA based programmable controller for safety critical applications.

5 Future Work

Presently the three copies of code are identical to each other. To guard against design faults in software it is possible that each of the three codes is different in that the functionality of the codes remain same however the implementation of this functionality differs from code to code. For example the simple counter implemented in this case can be implemented in three different ways viz
1. For loop
2. While loop
3. Do While loop
The issue with such methodology is the timing difference that may occur at outputs of each Microblaze System due to which outputs arrive at different instants of time thus giving an unstable operation in that the voter would signal error every now and then which could cause system malfunction.
We propose to investigate in this direction of code diversity in TMR systems and challenges involved in design of voter for such circumstances.

References

MISRA(2004), MISRA-C : 2004, Guidelines for the use of the C language in critical systems.

IEC(1998), International Standard IEC 61508 - Functional safety of electrical/electronic/programmable electronic safety related systems.

B.W.Johnson(1989), Design and Analysis of fault tolerant digital systems.

G.Berry and the Esterel team(2000), The Esterel v5_91 System Manual.

F.Lima Kastensmidt, L.Sterpone, L.Carro, M.Sonza Reorda(2005), On the Optimal Design of Triple Modular Redundancy Logic for SRAM based FPGA's, *in* 'IEEE Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05)', Vol. 2, pp. 1290–1295.

Elmootazbellah Elnozahy, Rami Melhem, Daniel Mosse(2002), Energy Efficient Duplex and TMR Real Time Systems, *in* 'Proceedings of the 23rd IEEE Real Time Systems Symposium (RTSS'02)', pp. 256–266.

Hyunki Kim, Hyung Joon Jeon, Keyseo Lee, Hyuntae Lee(2002), The Design and Evaluation of All Voting Triple Modular Redundancy System, *in* '2002 IEEE Proceedings Annual of Reliability and Maintainability Symposium', pp. 439–444.