A Novel Design Flow for Fault-Tolerant Computing

Jonathan Kimmitt
Dept of Computing and Technology
Anglia Ruskin University
Cambridge, England
Email: jonathan.kimmitt@anglia.ac.uk

George Wilson
Dept of Computing and Technology
Anglia Ruskin University
Cambridge, England
Email: george.wilson@anglia.ac.uk

David Greaves
Computer Laboratory
University of Cambridge
Cambridge, England
Email: djg11@cam.ac.uk

***Abstract*—This paper presents a new hardware synthesis flow, which generates an output verifiable in a field-programmable gate array. It demonstrates the relevance of fault-tolerant synthesis as required by demanding, sustainable, safety-critical applications. Although general-purpose in capability, the technique is particularly applicable for modern processor implementations, where the consequences for undetected errors are usually catastrophic.**

I. Introduction

As commercial pressures force micro-chip manufacturers to migrate to every smaller geometries, full logic reliability can no longer be assumed due to statistical variability and single event logic upsets. Furthermore, as mass-production products consume more and more fabrication capacity, new designs are forced onto smaller geometries, whilst at the same time costs such as mask making become prohibitive except for the largest production runs. Hence the use of field-programmable gate arrays (FPGA) with very low geometries is inevitable even in safety-critical projects such as medical and aerospace.

A. Conventional Approaches

The conventional approach to the problem of statistical variability as a cause of failure through timing hazards is to simply scale the transistors and logic primitives of the previous generation, and develop more sophisticated Monte-Carlo Spice simulation [1] and noise-aware static-timing analysis [2]. This is a valid approach, but the downside is that very little of the speed benefit of the smaller transistors will feed into the final performance because of the large variability. A consequence of this is that, although prices have come down, maximum clock speeds have barely improved in the last half-decade.

B. Failure statistics

The problem of logic upsets is of great importance in FPGAs used in aerospace applications. According to Swift [3], worst case (1200km, 65° inclination satellite orbit) results for an XQR4VLX200 (the largest Xilinx [4] device in the available range of 90nm geometry process) are summarised in Table I. The smaller RAM cell is easier to upset than the flip-flop, but techniques such as forward error correction and scrubbing may be used. An upset in a critical flip-flop may require a full reset, which can result in service outage or possible loss of synchronisation with associated systems.

C. Asynchronous double-rail logic

A niche technique popular in academic circles but not particularly prevalent commercially is the use of asynchronous (i.e. without requiring a clock) logic making use of double rail indication of success (see for example [5]). Any function, after a change in its inputs, will return a signal on either of its outputs to indicate completion. The rest of the circuit always waits for a definite outcome of an earlier calculation. One disadvantage is that the correct timing of the circuit depends in general on the delays in local paths relative to the delays in global paths. Hence it is not robust against aggressive automatic timing optimisation which is an essential requirement of ultra deep sub-micron design(UDSM) where digital cell libraries may have hundreds of cells of various functions and particularly different drive strengths.

D. The chosen approach

This paper aims to improve the fault tolerance of systems by utilising the ever-increasing number of transistors on the latest CMOS processes, whilst at the same time countering the effects of UDSM statistical variability due to atomic and quantum effects. As downward price pressures mandate smaller, thinner components year-on-year, the lifetime of electronic devices is decreasing even though inherent defectivity is also decreasing. The occasional fault might be acceptable in a consumer product, assuming it can be detected and put right with little or no user intervention. It would not be allowed in high-cost, critical infrastructure, such as transport, medicine, aerospace or military applications. Although double-rail asynchronous techniques can be directly used in FPGA, for the purposes of this paper the meaning of the signals is adapted as shown in Table III. Correct operation is signalled by complementary signals, incorrect by common-mode signals. The overall scheme is then incorporated into a timing-driven flow, and then demonstrated in FPGA. The method is illustrated in conjunction with the y86 educational processor [6].

II. Methodology

A. Conventional Methodology

A typical FPGA flow, such as that provided by Xilinx [7] consists of steps which are superficially similar to an application-specific integrated circuit(ASIC) flow, namely synthesis, mapping, placement, routing and design rule checking. In this context, the synthesis stage is open, the mapping

TABLE I: XQR4VLX200 upsets [3]

Event	Frequency
functional interruption	0.09/year
block RAM upset	13.9/day
flip-flop upset	0.8/day

TABLE II: Custom flow code statistics

Module	Line Count
Verilog Grammar	1950
Semantic Checks	2000
BDD [15] package	600
Recognising Library	525
Flattening	450
Double Rail Conversion	100
Optimisation	1050
ABC [16] wrapper	350
Edif Output	300

stage is semi-proprietary, and the remaining steps are fully proprietary. If Xilinx synthesis technology(XST) is used, the input to the whole process will be VHDL [8] or Verilog HDL [9], and the remaining steps are opaque. If third-party synthesis is used the input to the process must be in Electronic design interchange format (EDIF [10]).

B. Description of the proposed new approach

In the proposed new approach, the fault-tolerance is introduced in between synthesis and mapping. There are several reasons for this. Firstly, the format of the gate level netlist is easier to process as there are fewer alternatives to consider. Secondly it is highly desirable to be able to debug the process of introduction of fault-tolerance, and several effective Verilog simulators are available. Thirdly it is very easy to convert gate-level Verilog to EDIF as required by the mapping phase, and fourthly the uniformity of the gate-level description reduces the number of fault-tolerant cells that will have to be specially designed.

C. Detailed discussion of the conversion to fault-tolerance

The custom flow sits in the overall flow as shown in Fig 1. The entirety of the custom flow, apart from certain libraries listed below, is written in OCAML [11], in order to provide type-safety in conjunction with efficiency and economy of expression. Code size statistics are mentioned in Table II. Except where referenced, the code was created by the first author, apart from the Verilog grammar where the Backus-Naur formalism [12] written in bison [13] was converted from Verilator [14] to ocamlyacc format.

1) Optimistic and pessimistic gates: The internals of a dual-rail gate require clarification; referring to Table III again, it is apparent that the definition of logic gate is not unique. The purpose of stuck indications is not only to detect internal inconsistency, but also to forward a fault indication from a logic device's fan-in cone. If the gate forwards any fault at any input, the design will be optimum for fault detection, so can be described as a *pessimistic* gate. It can be shown by simple fault simulation that this logic is optimum for detecting faults. However there is a problem with initialisation of the circuit should it consist entirely of this kind of gate. Should it power on in the unknown state, approximately half of the flip-flops will start in the fault state, which is not useful. In the fan-out cone of the reset input to the design, we want the reset indication to dominate over the unknown

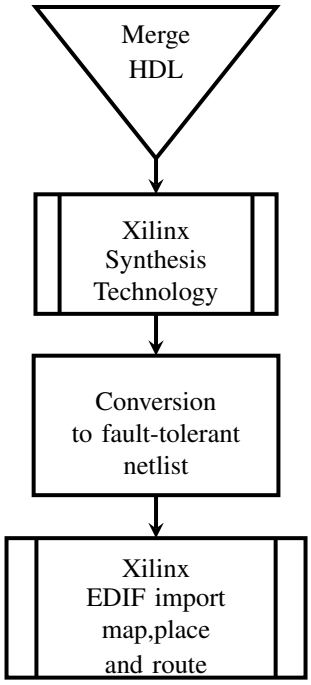

Fig. 1: Modified FPGA flow

TABLE III: Double rail logic encoding

State	Encoding
00	stuck low
01	logic low
10	logic high
11	stuck high

indication. So in order to reset the idea of an *optimistic* gate is introduced, which during reset will convert any state, into a valid state. In this version of the flow we do not have access to synthesis data structures so it is not easy to generate the reset fan-out cone. Therefore the remainder of the discussion will consist of logic networks made only with optimistic gates. It is necessary to have suitable reset structures in RTL to force initialisation of critical flip-flops. This will be true anyway for well conditioned designs.

2) Recognition of the library primitives: It is necessary to recognise when the flow synthesis reaches a leaf cell (also known as a library primitive) in order to prevent over-flattening of the netlist. The Verilog syntax 'celldefine is available for this purpose, however Xilinx libraries do not use it. This approach is to use the pseudo-code in Table IV. The operations are explained as follows. Firstly the location of the libraries is defined. The library directory is then scanned to identify suitable leaf cells. Any special cells used in the design are then manually added in. Finally, the library is constructed with the results shown in Table V. At this stage the special cells needed for flattening and also for EDIF output are identified. The identification is made by an analysis of the underlying function of the Verilog. This final stage marks every cell successfully identified as a library primitive.

3) The Flattening process: The input to this stage is the synthesised output of the synthesis, typically done by XST. In Table IV, the Verilog file processor_timesim.v would be the output from running XST on the module processor. Alternative front-ends HANA [17] or ODIN2 [18] could also be used

TABLE IV: Library recognition and Flattening Pseudo-code

```
library_env /home/arucad/Xilinx/13.1/ISE_DS/ISE/verilog/src/unisims/
scan_library
vparse /home/arucad/Xilinx/13.1/ISE_DS/ISE/verilog/hdlMacro/AND8.v
vparse /home/arucad/Xilinx/13.1/ISE_DS/ISE/verilog/hdlMacro/OR8.v
read_library
vparse processor_timesim.v
gen_flat_arch verilog processor
write_arch flat processor
quit
```

TABLE V: Output from the library recognition stage

```
883 library cells detected
113 non-inverting buffers detected
Using library buffer BUF B1(.O(out),.I(in));
1 inverting buffers detected
Using library inverter INV N1(.O(out),.I(in));
1 power sources detected
Using library power source VCC (.P(out));
1 ground sources detected
Using library ground source GND (.G(out));
1 tri-state buffers detected
Using library tri-state buffer BUFE B1(.O(out),.E(in),.I(en));
```

depending on the source syntax. In general the output format will be hierarchical. It is conceptually easy to convert to a flat netlist but there are a number of subtle issues. Flattening the Verilog netlist (represented internally as a hash table) takes place by means of a standard recursive descent algorithm. It is likely that the synthesis process will have introduced feed-throughs, assigns or partial busses which cause problems for an optimum approach. Instead a context-free algorithm is used which places a buffer of appropriate direction on every hierarchical boundary. A later optimisation stage is relied on to remove the redundant elements. Flattening stops when a previously identified library cell is encountered or a sub-circuit which is identified as behavioural (such as a block-RAM).

4) Conversion to double-rail logic: The input to this phase is the flat netlist internally generated by the flattening phase. Only a few primitives will be present. The fault-tolerant version of each primitive is assumed to have been hand-designed or otherwise previously generated. Suppose there exists a primitive AND2(.O(Y), .I0(A), .I1(B)) (which trivially represents the boolean function Y = A.B). This is then mechanically converted into F_AND2(.O({F_Y,Y}), .I0({F_A,A}), .I1({F_B,B})) where F_Y is the complement of Y, any other value represents a fault. It is apparent that the fault-tolerant netlist will be considerably larger than the assumed good netlist. However the requirement for inversions is virtually eliminated because AND,OR,NAND,NOR gates are freely available just by permuting the inputs and/or outputs. It is apparent that only detecting errors has been discussed, not correcting them. So the new logic would have to be used inside a higher-level protocol, such as voting logic. By contrast triple modular redundancy [19] has not been explored because the physical fault model considered in this work suggests defects will be local and rare. Therefore it makes more sense for a higher-level of protocol to discard the output of a known bad block, from a physical decision point which is well away from the putative fault origin.

5) Optimisation: After conversion and flattening there will be a degree of unwanted logic, due to the naive nature of the flattening process and due to chains of logic which have been individually converted. For efficiency of verification it is essential to optimise the netlist before entering the opaque stage of the Xilinx flow, or alternatively the putative ASIC flow.

TABLE VI: C code for smoke-test

```
int array[4] = {0xd, 0xc0, 0xb00, 0xa000};

int smoke_test(int *Start, int Count)
{
  int sum = 0;
  while (Count) {
    sum += *Start;
    Start++;
    Count–;
  }
  return sum;
}

int main(void)
{
  return smoke_test(array, 4);
}
```

Since gate-level optimisation algorithms are non-trivial, the ABC [16] library of mapping routines is used. In this library the optimisation flow itself consists of a number of stages, including conversion of the input to a network, transformation into logic, conversion into and-inverter graph(AIG) form, and finally mapping. In the ABC flow hierarchical netlists are deprecated so requiring another stage of flattening. In theory the mapping library could be directly generated from the library primitives. At this stage this has not been done because of the complex internal assumptions of the ABC library. The Verilog netlist interface has a number of limitations, particularly in the area of flip-flops, so the solution is to bypass this stage and go directly into ABC as a network. If the flip-flops are anything other than simple D-types with a global clock, the network does not represent this functionality directly and so they are tracked separately in the main database. The result of this is that redundancies in clock, clock enable, and preset/clear networks will remain until the back-end flow. However this a small part of the total. The final network is restored to internal netlist format as well as restoring the flip-flop name and control signals which will be needed later.

6) Output: The resulting netlist can be written as structural Verilog for use in simulation, or as an EDIF netlist. The Verilog presents no difficulty as it is the internal format; the EDIF is another structural format which differs chiefly in that it lists each net with the cells that it connects to, instead of listing each cell with the nets that it connects to. To make a valid EDIF netlist for subsequent Xilinx processing, appropriate input or output pads must be added to every primary pin. The possibility of a bidirectional pin is not catered for in this flow. If wanted, it could be added manually using the ngdbuild feature of the Xilinx tools to merge EDIF netlists.

III. Results

The results of using the methodology on the y86 processor [6] are shown below. This is a relatively low-level register-transfer level(RTL) description. This methodology lacks a way of optimising blocks as RAMs inside ABC. A workaround is to bring the RAM up a level of hierarchy and implement it inside a top-level structural framework. This allows the processor with its dual-rail logic to be optimised down to gate-level primitives without having to worry about maintaining bus connections at the boundaries of black boxes.

TABLE VII: Compiled assembly code for smoke-test

```
/* $begin code-yso */
/* $begin code-ysa */
# Execution begins at address 0
 .pos 0
start: irmovl cstack, %esp   # Set up stack pointer
 irmovl cstack, %ebp   # Set up base pointer
 call main  # Execute main program
 halt   # Terminate program
# gcc version 3.4.6 Wed Mar  7 14:30:18 2012

# global array
# data section
 .align 2
array:
 .long 13
 .long 192
 .long 2816
 .long 40960
# text section
# global smoke_test
smoke_test:
 rrmovl %ebx, %ecx #  88 *movsi_1/1 [length = 2]
 pushl %ecx #  89 *pushsi1/1 [length = 1]
 mrmovl 8(%esp), %eax #  3 *movsi_1/3 [length = 5]
 mrmovl 12(%esp), %edx #  4 *movsi_1/3 [length = 5]
 xorl %ebx, %ebx #  87 *movsi_xor [length = 2]
L7:
 andl %edx, %edx #  55 *cmpsi_ccno_1 [length = 3]
 je L6 #  56 *jcc_1 [length = 2]
 mrmovl (%eax), %ecx #  68 *movsi_1/3 [length = 3]
 addl %ecx, %ebx #  69 *addsi_1/1 [length = 2]
 irmovl $4, %ecx #  70 *movsi_1/2 [length = 5]
 addl %ecx, %eax #  71 *addsi_1/1 [length = 2]
 irmovl $-1, %ecx #  86 *movsi_or_else [length = 3]
 addl %ecx, %edx #  73 *addsi_1/1 [length = 2]
 jmp L7 #  84 jump [length = 2]
L6:
 rrmovl %ebx, %eax #  44 *movsi_1/1 [length = 2]
 popl %ebx #  78 popsi1 [length = 1]
 ret #  79 return_internal [length = 1]
# global main
main:
 irmovl $4, %ecx #  30 *movsi_1/2 [length = 5]
 pushl %ecx #  31 *pushsi1/1 [length = 1]
 irmovl $array, %ecx #  32 *movsi_1/2 [length = 5]
 pushl %ecx #  33 *pushsi1/1 [length = 1]
 call smoke_test #  12 *call_value_0 [length = 5]
 popl %edx #  39 popsi1 [length = 1]
 popl %edx #  40 popsi1 [length = 1]
 ret #  37 return_internal [length = 1]

# The stack starts here and grows to lower addresses
 .pos 0x100
cstack:
/* $end code-ysa */
/* $end code-yso */
```

A. Preparing the pre-synthesis simulation

The pre-synthesis simulation, also known as the RTL simulation, requires the processor RTL as mentioned above, a test bench, a dual-port RAM (either behavioural or Xilinx specific) and the software to be tested. For the smoke test (that is, the trivial program that attempts to prove some function, but if it fails provides ease of debugging) the C code shown in Table VI is used. On the y86, instructions and addressing modes are drastically cut down from the x86 processor series, requiring manual modification of x86 compiler output for instructions or modes that are not valid. The alternative adopted here is to remove the invalid instructions from the compiler register-transfer expressions. However only a subset of C will be supported (For example the only addressing mode is 32-bits). The output of the modified compiler is shown in Table VII. It is apparent that the C code needs to be topped and tailed with a machine-specific initialisation. If the code download feature of the y86 inside the testbench is utilised the execution of this program will result in the 4-word sum ABCD(hex) being left in a register. The exercise can be repeated with the same result on the output of the flow.

B. Applying the tool-chain

Applying the tool-chain described above produces a report of the form Table VIII. The corresponding result without the double-rail logic is in Table IX. As expected the number of flip-flops is doubled, the number of combinational gates will be more than doubled (provisional results). A naïve transistor count is available by multiplying the instances of a given cell in column 5 by the transistor count of a basic CMOS implementation in column 6, with a resulting ratio of 2.9:1. *These results do not include the overhead of deciding what to do when an error is detected. This overhead will vary by application but it could be substantial. However as gate densities reach beyond millions of gates per square millimetre, this is unlikely to be a deciding factor.*

1) FPGA results: To obtain the FPGA overhead, the Xilinx EDIF flow is run, resulting in the report files as shown in Table X. and Table XI. In this architecture, the slice usage ratio rises by about 5:1. This is not surprising since this is a generic, not an FPGA-specific flow, so assumptions about the mapping of the logic do not necessarily carry over to LUT based architectures. In particular the LUT architecture performs poorly when many signals fan in to a flip-flop. This will inevitably be the case when trying to detect logic faults coming in from a wide fan-in cone and pass them on to the next stage via a flip-flop pair. A configurable logic block(CLB) normally has two flip-flops. In this technique the CLB will only hold one bit of (4-state) information instead of the usual two. However in most designs the size is dominated by on-chip memory. The user can choose whether to replicate the double rail feature in RAM or not. To reduce overhead using the parity feature would be preferable. By contrast the overhead reported by Miller [19] using triple modular redundancy is 9:2 for LUTs and 3:1 for flip-flops(excluding the processor, which was a hard macro and therefore not very realistic).

2) Timing: The timing tables come from the place and route(PAR) results. The reports are shown in Tables XII. and XIII. Degradation is approximately 2:1 which is acceptable. *Again, the latency is expected to increase once a higher level protocol to deal with the consequences of an error has been added.*

IV. Benefits

It can be seen that the introduction of the current methodology carries a substantial area penalty and some speed penalty. But there are benefits:

A. Security

1) Confidence in the computation: In the early days of computing it was assumed a computer would either produce the correct result(as defined by its programming), or crash in a visible manner, such as the well-known blue screen of death. The intermediate scenario, where nothing visible goes wrong but the output is not correct either, will become increasingly common as delay faults [20] become more difficult to detect. Using dual-rail logic, the computer positively asserts that all is well on every clock cycle, and any faults will become apparent after a latency only determined by the sequential depth between input and output. In most designs this value is rather low in order to achieve high throughput. The recovery mechanism would be application dependant. In an engineering

TABLE VIII: Usage report for y86 (double-rail technology)

name	ios	primary	sequence	instances	transistors	total	truth table
XOR2	3	xor		152	4	608	O = I0 and not I1 or not I0 and I1
GND	1	binnum		784	2	1568	G = 0
RAMB16_S9_S9	18	empty		8			
NOR2	3	nor		12936	4	51744	O = not I0 or I1
NOR3	4	nor		5586	6	33516	O = not I0 or I1 or I2
NOR4	5	nor		859	8	6872	O = not I0 or I1 or I2 or I3
NAND2	3	nand		1064	4	4256	O = not I0 and I1
NAND3	4	nand		255	6	1530	O = not I0 and I1 and I2
NAND4	5	nand		87	8	696	O = not I0 and I1 and I2 and I3
FDPE	5	memory	posedge if	775	12	9300	Q = q_out
MUXF5	4	empty		6	4	24	O = 0
FDCE	5	memory	posedge if	775	12	9300	Q = q_out
AND2	3	and		9496	6	56976	O = I0 and I1
AND3	4	and		1730	8	13840	O = I0 and I1 and I2
AND4	5	and		337	10	3370	O = I0 and I1 and I2 and I3
AND5	6	and		90	12	1080	O = I0 and I1 and I2 and I3 and I4
AND8	9	and		8	18	144	O = I0 and I1 and I2 and I3 and I4 and I5 and I6 and I7
VCC	1	binnum		259	2	518	P = 1
FD	3	memory	posedge	2	6	12	Q = q_out
BUF	2	buf		518	4	2072	O = I
OR2	3	or		4137	6	24822	O = I0 or I1
OR3	4	or		2183	8	17464	O = I0 or I1 or I2
OR4	5	or		364	10	3640	O = I0 or I1 or I2 or I3
OR8	9	double		84	18	1512	O = I0 or I1 or I2 or I3 or I4 or I5 or I6 or I7
INV	2	not		3790	2	7580	O = not I
					Grand Total	252444	

TABLE IX: Usage report for y86 (normal/non double-rail technology)

name	ios	primary	sequence	instances	transistors	total	truth table
XOR2	3	xor		397	4	1588	O = I0 and not I1 or not I0 and I1
GND	1	binnum		50	2	100	G = 0
RAMB16_S9_S9	18	empty		8			
FDCE	5	memory	posedge if	775	12	9300	Q = q_out
AND2	3	and		6434	6	38604	O = I0 and I1
AND3	4	and		141	8	1128	O = I0 and I1 and I2
AND4	5	and		28	10	280	O = I0 and I1 and I2 and I3
AND5	6	and		4	12	48	O = I0 and I1 and I2 and I3 and I4
AND8	9	and		15	18	270	O = I0 and I1 and I2 and I3 and I4 and I5 and I6 and I7
VCC	1	binnum		17	2	34	P = 1
FD	3	memory	posedge	1	6	6	Q = q_out
BUF	2	buf		3754	4	15016	O = I
OR2	3	or		1743	6	10458	O = I0 or I1
OR3	4	or		11	8	88	O = I0 or I1 or I2
OR4	5	or		49	10	490	O = I0 or I1 or I2 or I3
OR8	9	double		150	18	2700	O = I0 or I1 or I2 or I3 or I4 or I5 or I6 or I7
INV	2	not		3665	2	7330	O = not I
					Grand Total	87440	

TABLE X: Y86 device summary (double-rail logic)

```
Device Utilization Summary:

Number of External IOBs                539 out of 640     84%
   Number of LOCed IOBs                539 out of 539    100%

Number of RAMB18X2s                      6 out of 148      4%
Number of Slices                      8055 out of 17280   46%
Number of Slice Registers             1552 out of 69120    2%
   Number used as Flip Flops          1552
   Number used as Latches                0
   Number used as LatchThrus             0

Number of Slice LUTS                 18506 out of 69120   26%
Number of Slice LUT-Flip Flop pairs  18506 out of 69120   26%
```

TABLE XI: Y86 device summary (normal logic)

```
Device Utilization Summary:

Number of External IOBs                539 out of 640     84%
   Number of LOCed IOBs                539 out of 539    100%

Number of RAMB18X2s                      6 out of 148      4%
Number of Slices                      1527 out of 17280    8%
Number of Slice Registers              776 out of 69120    1%
   Number used as Flip Flops          776
   Number used as Latches                0
   Number used as LatchThrus             0

Number of Slice LUTS                  3895 out of 69120    5%
Number of Slice LUT-Flip Flop pairs   3895 out of 69120    5%
```

situation, it might be enough to just hit the reset button and start again. In an engine management system, it would make sense to switch to the backup system automatically based on the good/fault output status.

2) Inscrutability: In systems such as smart cards, the security of the secret keys depends on the lack of a suitable cryptographic attack. Where access to the power line is available, on some smart-card designs the power drain will depend on the internal logic state. If the device can be made to execute the same algorithm repeatedly, it is possible to infer the secret keys or dramatically reduce the search space by measuring the power consumption profile. This situation naturally arises as a result of the different rise and fall times of NOR and NAND gates. However with dual rail logic, an equal number of flip-flops are high and low, and we can make NOR and NAND structures from the same transistor topology. Other software-base obfuscation measures are still required.

3) Harsh environments: In ionising radiation intensive environments, a gradual shift in the transistor threshold will occur which will affect timing and symmetry and eventually lead to failure. To alleviate this problem a dose of radiation can be pre-applied under controlled conditions to try to ensure all transistors will shift by the same amount. However the effects

TABLE XII: timing (double-rail logic)

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
Autotimespec constraint for clock net qq_clock	SETUP	N/A	72.181ns	N/A	0
	HOLD	0.056ns		0	0

TABLE XIII: timing (normal logic)

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
Autotimespec constraint for clock net qq_clock	SETUP	N/A	33.598ns	N/A	0
	HOLD	0.042ns		0	0

of radiation dose are unpredictable in operation and having a positive confirmation of correct operation is valuable.

B. Adaptive clocking

Since there is a definite go/no-go status at all times, the clock speed/voltage scenario can be adjusted to match the temperature and process. A more expensive system could also control temperature. All that is needed is a worst case program to execute which utilises the deepest combinational paths, and a model of how the delay varies with environment. Because of difficulties with global skew, clock-gating is usually replaced by flip-flop enables in FPGAs, unless there are overriding power consumption reasons. In this case extra control modes might be needed to bring the worst case logic into operation. Calibration may be done by varying the clock rate until a failure occurs. With a good environmental model and a suitable safety factor, a high level of confidence combined with optimum relative power consumption may be obtained.

V. Limitations

The flow thus described only handles flat netlists. These are unwieldy and memory intensive when large designs are employed. The support for a full hierarchical flow is a topic for further research. More investigation is needed to determine the optimum design to minimise area overhead and maximise fault coverage. A new tool option is needed to mix double rail gates of different truth tables to ensure proper initialisation without unnecessary masking of faults.

VI. Conclusion

It has been shown that the introduction of double-rail logic into a digital synthesis flow is suitable for computer automation, and can be integrated with current FPGA tools. The use of 4-states rather than 2 is robust for detecting single-event upsets and hard faults caused by defects and out-of-tolerance PVT conditions. It is applicable to defects caused by hard radiation and electro-migration as well. The solution is also highly symmetrical and thus robust against differential power-line crypto-analysis. The main disadvantages relate to increased power consumption, higher silicon cost, and reduced speed. These disadvantages would weigh heavily in consumer applications but typically would not be significant in critical systems where the electronics is a small proportion of the total cost and safety is paramount.

References

[1] M. Maxim A Gheorghe, "A novel physical based model of deep-submicron CMOS transistors mismatch for Monte Carlo SPICE simulation," *Circuits and Systems 2001 ISCAS 2001 The 2001 IEEE International Symposium on*, vol. 5, pp. 511—-514 vol. 5, 2001.

[2] I. Nitta, T. Shibuya, K. Homma, F. Laboratories, F. Limited, and F. V. Limited, "Statistical Static Timing Analysis Technology," *Fujitsu*, vol. 523, no. October, pp. 516–523, 2007.

[3] G. Swift, C. Carmichael, and G. Allen, "Virtex-4QV static SEU characterization summary," *JPL Publication*, 2008.

[4] H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey, "A review of Xilinx FPGA architectural reliability concerns from Virtex to Virtex-5," *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, pp. 1–8, 2007.

[5] C. LaFrieda, B. Hill, and R. Manohar, "An Asynchronous FPGA with Two-Phase Enable-Scaled Routing," *2010 IEEE Symposium on Asynchronous Circuits and Systems*, pp. 141–150, 2010.

[6] R. E. Bryant and D. R. O. Hallaron, "Verilog Implementation of a Pipelined Y86 Processor," 2011. [Online]. Available: http://csapp.cs.cmu.edu/public/waside/waside-verilog.pdf

[7] I. Xilinx, "Virtex-5 FPGA User Guide," p. 385, 2008. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf

[8] C. Delgado Kloos and P. T. Breuer, *Formal Semantics for VHDL*, ser. The Kluwer international series in engineering and computer science, P. T. Breuer and C. D. Kloos, Eds. Kluwer Academic Publishers, 1995, vol. 76.

[9] M. Gordon, "The Semantic Challenge of Verilog HDL," *The Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*, pp. 136–145, 1995.

[10] P. Stanford and P. Mancuso, *EDIF Electronic Design Interchange Format, Reference Manual for Version 2 0 0*. Electronic Industries Association, Washington D.C., 1989.

[11] D. Remy and J. Vouillon, "Objective ML: An effective object-oriented extension to ML," *Theory and Practice of Object Systems*, vol. 4, no. 1, pp. 27–50, 1998.

[12] R. S. Scowen, "Extended BNF-a generic base standard," in *Software Engineering Standards Symposium*, vol. 3, no. 1, ISO/IEC 14977. Citeseer, 1993, pp. 6–2.

[13] C. Donnelly and R. Stallman, "Bison The Yacc-compatible Parser Generator," 2006. [Online]. Available: http://www.gnu.org/software/bison/manual/bison.pdf

[14] W. Snyder, "Verilator-3.805," 2010. [Online]. Available: http://www.veripool.org/ftp/verilator_doc.pdf

[15] J.-C. Filliatre, "Binary decision diagrams (BDDs)," 2010. [Online]. Available: http://www.lri.fr/~filliatr/ftp/ocaml/bdd/

[16] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*. Springer, 2010, pp. 24–40.

[17] P. Ahmad, "HDL Analyzer and Netlist Architect," 2011. [Online]. Available: http://sourceforge.net/projects/sim-sim/

[18] P. Jamieson, K. Kent, and F. Gharibian, "Odin II - An Open-source Verilog HDL Synthesis Tool for CAD Research," *(FCCM), 2010 18th*, 2010.

[19] G. Miller and C. Carmichael, "Single-Event Upset Mitigation Design Flow for Xilinx FPGA PowerPC Systems," *System*, vol. 1004, pp. 1–28, 2008.

[20] E. Chmelaf, "Fpga interconnect delay fault testing," *International Test Conference 2003 Proceedings ITC 2003*, pp. 1239–1247, 2003.