R3TOS-Based Autonomous Fault-Tolerant Systems

Xabier Iturbe
Ali Ebrahim Khaled
Benkrid Chuan Hong
Tughrul Arslan
University of Edinburgh

Jon Perez
IK4-Ikerlan

Didier Keymeulen Jet Propulsion Laboratory

Marco D. Santambrogio
Politecnico di Milano

••••••In this new age of exploration, which began in the mid-20th century, we trust electronic systems to be our nose, ears, and eyes wherever we could not arrive because of human limitations (for example, travel time, lack of oxygen, corrosion, temperature, and radiation). Although the combination of human and electronic capabilities has been effective, it will not be possible in the future as exploration electronic systems reach ever more remote, unknown, and harsh environments, such as the underground ocean in Jupiter's moon Europa. Earth-driven reaction to the hazards posed by these environments (such as high-energy particles, corrosive chemical plumes, and volcanic black smokers in Europa's ocean) will be too slow, threatening the electronic explorer, which will not be accessible for people to repair. Hence, next-generation exploration electronics should be adaptive and autonomous to increase the scientific return of future missions.

Adaptivity and autonomy would contribute to increase the robustness and effectiveness of many applications operating in more mundane and better-known environments as well. Adaptivity can help daily-use consumer electronic devices fulfill each user's particular requirements and expectations, whereas self-healing would allow for making these devices more reliable, durable, and cheaper to maintain. The latter is especially important as the process technology becomes more complex and prone to failure, with the chips degrading faster and in an unpredictable way over time as a result of manufacturing-process variations (such as transistor channel length and threshold voltage).

R3TOS—the Reliable Reconfigurable Real-Time Operating System—is our solution for building efficient, self-healing, and adaptive electronics using commercial-off-the-shelf partially reconfigurable Xilinx field-programmable gate arrays (FPGAs).[1,2] R3TOS lets a system gain control over its own resources at the finest chip granularity to implement the high-level behavioral functionality specified by the system designer. R3TOS improves efficiency by using a virtually unlimited number of computation-specific circuits, which are swapped in and out of the chip as needed to complete the overall computation within the smallest time span. It increases reliability by always circumventing the use of damaged chip resources, keeping the system fault free and safe. Finally, it enhances adaptivity by autotuning system operation to the particular needs at each time, as well as to deviations in its own functioning. (For information on other approaches and issues, see the "Related Work and Technology Limitations" sidebar.)

The R3TOS approach

R3TOS is aimed at making autonomous fault-tolerant systems (AFTSs) mainstream by providing the user with a set of hardware abstraction services and fault-detection, recovery, and damage-handling strategies, as well as computation-scheduling and circuit-allocation mechanisms. All this functionality is executed by a hardware microkernel (HWμK), which is designed to be simple, reusable, and reliable in order to ease the composability and certification of R3TOS-based applications. Because the HWμK constitutes a single point of failure, it is carefully protected against faults through special hardening—for example, selective triple modular redundancy (TMR) and error-correcting code (ECC) bits in memories. The HWμK is currently implemented using on-chip FPGA standard resources, but we envision a future implementation built on the FPGA silicon.

R3TOS hardware microkernel

The R3TOS HWμK comprises three main components:[1,2] a scheduler to decide the optimum execution order of the computations, an allocator to manage FPGA resources and decide the most appropriate placement for the circuits, and a configuration manager to translate the high-level operations dictated by the scheduler and allocator into configuration commands for the FPGA. Each component is separately implemented to enable parallelism in the execution of R3TOS processes. For instance, while the scheduler decides which computation to execute next, the allocator can analyze the state of the FPGA surface, and the configuration manager can load a previously scheduled circuit in the FPGA. The parallel cooperation of simple components not only results in low runtime overhead but also in low area overhead; that is, the main computing core of all HWμK components is a tiny Xilinx PicoBlaze 8-bit processor. Overall, the HWμK consumes about 500 configurable logic blocks (CLBs) and 6 block-RAM memories (BRAMs).

Especially interesting in this article's scope is the configuration manager, which provides direct, efficient, and easy access to FPGA resources. The configuration manager supports circuit allocation, relocation, and deallocation, along with intercircuit communications and synchronization. Notably, the R3TOS configuration manager delivers three novel capabilities and addresses three important limitations imposed by current FPGA technology. First, it can modify the vertical position of relocatable circuits inside clock regions, even relocating them in between neighbor regions. In connection with this, it can compute the ECC values for the relocated configuration frames online, enabling designers to continue using Xilinx Frame-ECC logic. Finally, it can reroute local clock signals in each clock region where the relocated circuit spans on the fly.[3]

At the core, these capabilities are based on basic configuration operations, such as write, read back a set of configuration frames (RBF), blank a configuration frame, and load a partial bitstream (LPB). To increase performance, up to three different writing functions are available, each for use in a specific situation: write a single frame to a single location (WSF2S), write a single frame to multiple locations (WSF2M), and write multiple frames to a single location (WMF2S).

The sequence of configuration commands that must be executed for implementing these configuration functions are stored in a BRAM. The configuration manager's PicoBlaze uses these sequences as raw configuration templates that are dynamically adjusted with the particular parameters of the operation (such as the frame address or number of words to read back or write) to be performed each time before they are sent to the internal configuration access port (ICAP). A finite-state machine (FSM) controls the

Related Work and Technology Limitations

Parris et al. give a general view of the state of the art in autonomous fault tolerance techniques using field-programmable gate arrays (FPGAs).[1] They distinguish between *passive techniques,* which rely on fixed FPGA configurations (such as triple modular redundancy [TMR]) and *active techniques,* which rely on modifying the configuration of the FPGA (that is, bitstream) to adapt to faults. In our approach, we focus on active techniques to increase system flexibility, with the objective of enabling recovery from virtually all types of faults.

Upon beginning the R3TOS project, some of this article's authors published a paper describing a roadmap for building an autonomous fault-tolerant system (AFTS) using partially reconfigurable Xilinx FPGAs.[2] This paper explained the fundamentals of an AFTS and classified previous related work into four technological levels (TLs) associated with increasing levels of flexibility:

- TL0: human-driven runtime reconfiguration.
- TL1: use of FPGA configurations created at design time independently of fault locations detected at runtime.
- TL2: limited generation of FPGA configurations that avoid fault locations detected at runtime by combining pieces of configurations that were generated at design time (that is, bitstream relocation).
- TL3: unlimited generation of FPGA configurations that circumvent fault locations detected at runtime (that is, place and route).

Because TL3 involves online synthesis capability, which is not available for commercial-off-the-shelf FPGAs, the highest achievable level using this technology is TL2. An alternative to online synthesis is to use evolutionary techniques to blindly modify the FPGA's bitstream (that is, without any knowledge of the functionality associated with the configuration bits) to create at runtime a set of functionally identical, yet physically different, configurations that compete for selection on the basis of a fitness function that favors fault-free behavior.[3] However, two major problems are associated with this quasi-TL3 technique: the long (and possibly unaffordable) amounts of time necessary to generate the new FPGA configurations, and certification issues derived from using only partially tested circuitry that is created randomly.

On the other hand, a TL2 AFTS can manage at runtime a set of already synthesized, tested, and certified circuits to meet the functionality, performance, and reliability requirements all the time. Namely, the circuits are swapped in and out of the FPGA and relocated to functional on-chip resources at all times. However, some important limitations and challenges are identified, considering the development state of Xilinx's partially reconfigurable FPGAs. These include

- limitations on the relocatability of circuitry within the FPGA due to architectural limitations (for example, chip structure and heterogeneous resources, such as BRAMs and DSPs);[4]
- limitations for distributing the clock signal to relocatable circuits;[5]
- difficulties in providing connectivity to relocatable circuits along the entire chip;[6] and, linked with this,
- limitations for generating and managing at runtime relocatable circuits because of the need to deal with the static routes across the chip.[7]

Montminy et al. present one of the few reported hands-on research efforts for building a TL2 AFTS.[8] Their approach pursued the use of three redundant relocatable modules, receiving their input data and delivering their output results through dedicated buses connected to input FPGA pins and a majority voter, respectively. However, the authors reported unsolvable synthesis errors when generating the relocatable partial bitstreams, presumably due to routing issues. Unlike this approach, our solution does not rely on any physical wires to connect the relocatable circuitry and the I/O FPGA pins. Instead, it uses virtual communication through the FPGA's configuration layer; that is, data is read back from a source lookup table and copied to a destination lookup table through the FPGA's internal configuration access port.[9]

References

1. M.G. Parris, C.A. Sharma, and R.F. Demara, "Progress in Autonomous Fault Recovery of Field-Programmable Gate Arrays," *ACM Computing Surveys*, vol. 43, no. 31, 2011, article 31.
2. X. Iturbe et al., "A Roadmap for Autonomous Fault-Tolerant Systems," *Proc. Conf. Design and Architectures for Signal and Image Processing*, 2010, pp. 311-321.
3. R.F. Demara and K. Zhang, "Autonomous FPGA Fault-Handling Through Competitive Runtime Reconfiguration," *Proc. NASA/DoD Conf. Evolvable Hardware*, 2005, pp. 109-116.
4. T. Becker, W. Luk, and P.Y.K. Cheung, "Enhancing Relocatability of Partial Bitstreams for Run-Time Reconfiguration," *Proc. Ann. IEEE Symp. Field-Programmable Custom Computing Machines*, 2007, pp. 35-44.
5. A. Flynn, A. Gordon-Ross, and A.D. George, "Bitstream Relocation with Local Clock Domains for Partially Reconfigurable FPGAs," *Proc. Conf. Design, Automation and Test in Europe*, 2009, pp. 300-303.
6. P. Sedcole et al., "Runtime Integration of Reconfigurable Video Processing Systems," *IEEE Trans. Very Large Scale Integration Systems*, vol. 15, no. 9, 2007, pp. 1003-1016.
7. P. Sedcole et al., "Modular Dynamic Reconfiguration in Virtex FPGAs," *IEE Proc. Computers and Digital Techniques*, vol. 153, no. 3, 2006, pp. 157-164.
8. D. Montminy et al., "Using Relocatable Bitstreams for Fault Tolerance," *Proc. NASA/ESA Conf. Adaptive Hardware and Systems*, 2007, pp. 701-708.
9. X. Iturbe et al., "Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs," *Proc. Int'l Conf. Field-Programmable Logic and Applications*, 2011, pp. 295-300.

configuration command-flow transfers from the suitable BRAM positions to the ICAP, and in the case of read-back operations, it also commands the writing of the read-back data from the ICAP in the latter memory.

To achieve spatial isolation between the circuits running on the FPGA, a configuration guardian isolates their associated partial bitstreams in the configuration domain. The guardian, which conceptually acts as the memory protection unit (MPU) in a conventional processor, is a simple (and hence reliable) and independent entity that traps erroneous accesses to the ICAP. For instance, accesses to the HWμK configuration are considered erroneous when the HWμK is not running in privileged mode, and are thus prohibited.

Diagnosis, recovery, and damage-handling strategies

R3TOS executes multiple redundant instances of critical circuits either in parallel at distinct positions within the FPGA (spatial redundancy) or at different times (temporal redundancy).[4] The resources assigned to a circuit instance that has computed an erroneous set of results are kept in quarantine while an exhaustive diagnostic test is conducted on them. This test, which consists of loading built-in-self-test (BIST) circuits on the region in quarantine, aims to localize any damaged resources with sufficient resolution to enable reconfiguration around them.

Although this diagnostic test is performed only when a computation is erroneous, because of the HWμK's criticality, the HWμK's configuration state is periodically checked using the Xilinx built-in Frame-ECC logic to scrub any correctable fault before it leads to system failure.

R3TOS-based traction controller

To demonstrate the potential of R3TOS in dependable applications, we developed a R3TOS-based inverter controller of a real-world railway traction system that must work in a harsh thermal and electromagnetic environment provoked by the switching activity of power electronics—for example, insulated-gate bipolar transistors (IGBTs) working in the kV and kA range.[5] High temperatures accelerate silicon degradation (that is, permanent damage), whereas electromagnetic interferences (EMI) result in spontaneous soft errors.

The inverter controller is in charge of modulating a three-phase pulse-width modulated (PWM) signal according to the traction and braking orders given by the driver and the measured speed of the train each time to control a power bridge of IGBTs that convert electrical energy from the catenary into motor traction energy.[5] To ensure the train's safety, real-time performance is central in the controller: the temporal resolution must be as high as 500 ns. Our traction controller includes three main parts:

- the R3TOS HWμK;
- application-specific logic—that is, circuitry that performs the traction control and generates PWM signals (in short, the TC-PWM), and
- minimal static circuitry to handle system inputs and outputs.

All static logic (the HWμK and system I/Os) are constrained within the FPGA's upper-right quadrant, leaving the other three-quarters of the chip free of static routes to relocate the TC-PWM circuitry as needed.

Traction control and PWM generation circuits

Figure 1 shows the block diagram of the TC-PWM circuit, which executes a proportional closed-loop control. Both the target and actual train speed values are delivered via input lookup tables (LUTs) and subtracted from each other to produce an error value for modulating the PWM signals that control the traction motor. This involves comparing a reference sinusoidal that is dynamically generated with the required amplitude and frequency with three triangular signals, each shifted 120° in phase from the others. Each PWM phase changes its state every time its associated triangular signal crosses the reference sinusoidal. Hence, the PWM switching period, T_S, matches the triangular signal's period, whereas the PWM temporal resolution is defined by the period of the used clock, $T_{TC-PWM-CLK} = 500\text{ns}$. The TC-PWM circuit works with 208 duty-cycle increments, so the PWM switching frequency is 208 times slower than the used clock frequency (that is,

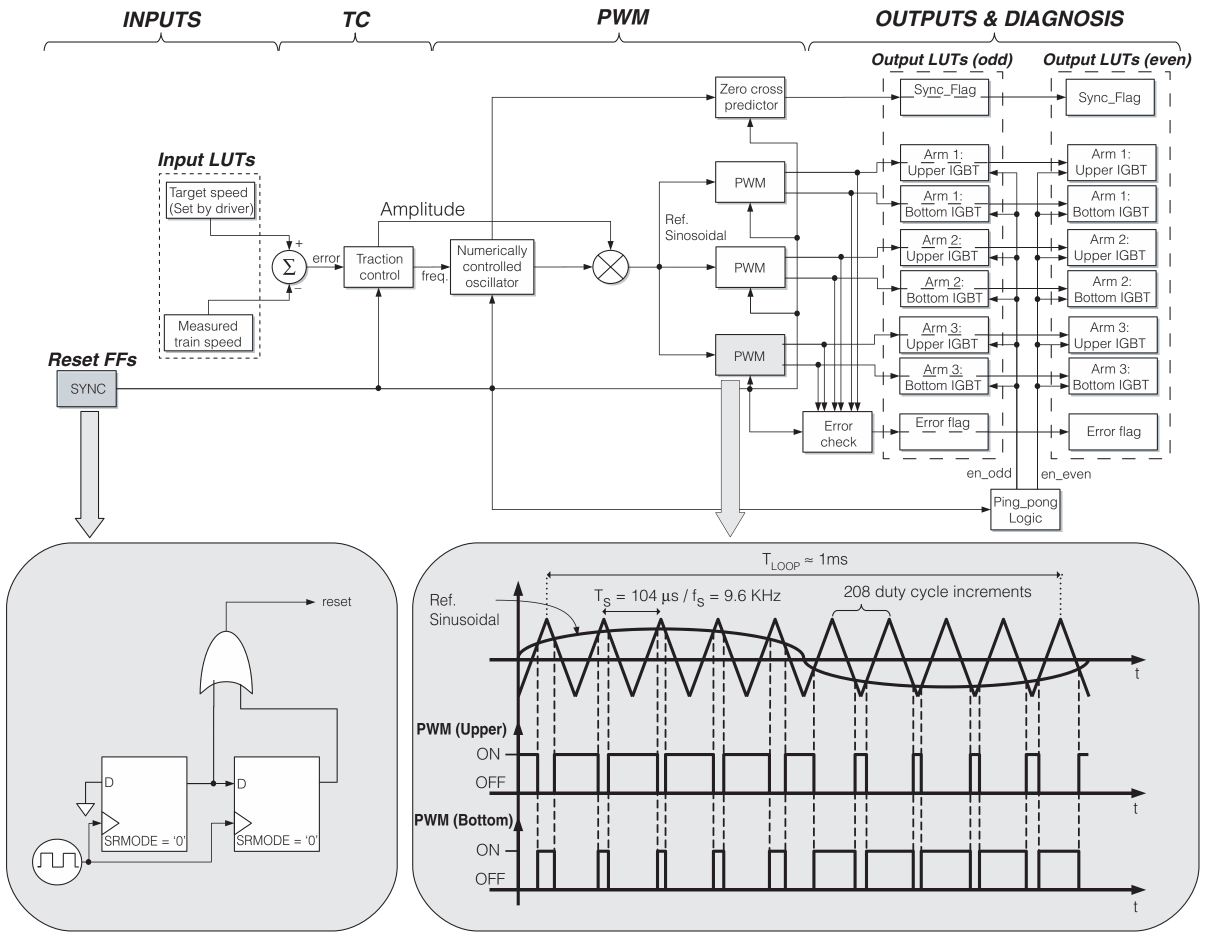


Figure 1. Block diagram and functioning of TC-PWM, the circuitry that performs traction control and generates pulse-width modulated (PWM) signals. Note that this circuit is a self-contained computation block in which the inputs and outputs are mapped to lookup tables (LUTs) and flip-flops.

$f_S = f_{TC-PWM-CLK} / 208 = 9.6\text{KHz}$). Moreover, the traction-control loop, T_{LOOP}, is closed every 10 PWM switching periods—that is, $T_{LOOP} = 10 \times T_S \approx 1\text{ms}$.

The value of the PWM signals at every $T_{TC-PWM-CLK}$ cycle is written to a set of output LUTs in the TC-PWM circuit. Each PWM phase is assigned two pairs of LUTs to store 16 consecutive PWM samples for the upper and bottom IGBT arms in the rectifier bridge, respectively. Each pair of LUTs assigned to the same IGBT arm operates in a ping-pong fashion, odd and even. Although the 16 PWM samples stored in the odd LUTs are accessed by the HWμK through the ICAP to be delivered to the FPGA output pins, the even LUTs are filled with the next 16 PWM samples, and vice versa. Hence, the LUT pairs alternate in operation every $16 \times 500\text{ns} = 8\,\mu\text{s}$. This functioning is mandatory for achieving a sufficiently high throughput via the ICAP to work with a temporal resolution of 500 ns.

The TC-PWM circuit also includes an error_flag LUT to indicate any deviation from its expected functioning, such as no pulse generation within a PWM switching period, or a pulse duration shorter than the minimum pulse width. To ease access through the ICAP, these LUTs are mapped to the same columns as output PWM LUTs.

With the objective of increasing the system's availability and diagnosing functioning errors, three redundant TC-PWM circuit

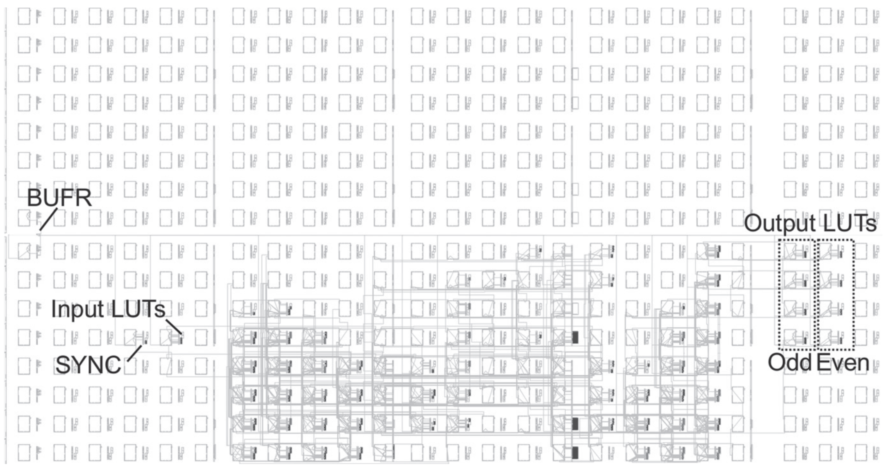


Figure 2. Field-programmable gate array (FPGA) implementation of the TC-PWM circuit. Note that the only signal that crosses the boundaries of the circuit is the clock. The LUTs and flip-flops used to implement the interface of the circuit are mapped to different FPGA resource columns, allowing independent access to them through the internal configuration access port (ICAP).

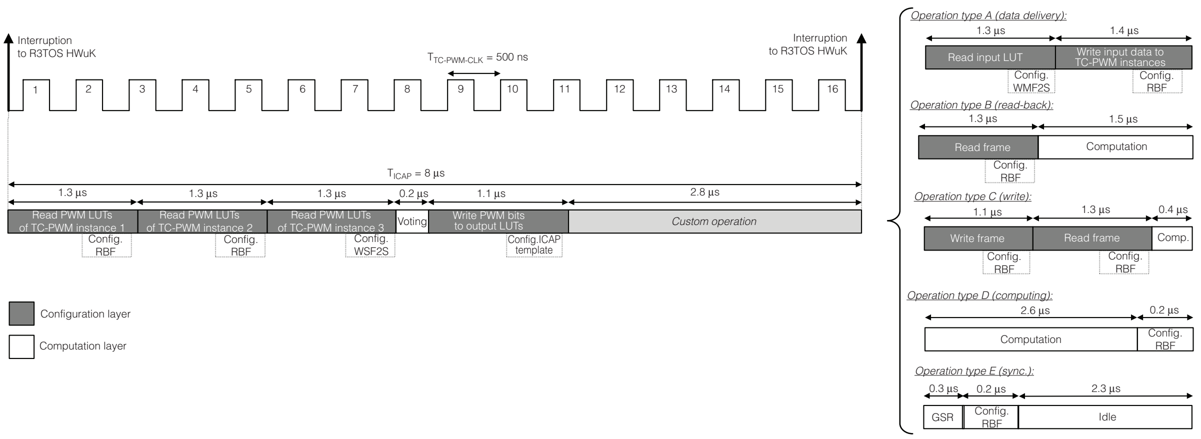
instances are kept running on the FPGA. Relocation is done only when a TC-PWM circuit instance does not work correctly in the original location (for example, error_flag=‘1’) and scrubbing does not fix the problem. The target location is decided by the R3TOS allocator, and the configuration data used in the relocation process is directly read back from any of the other TC-PWM circuit instances with error_flag=‘0’.

The three TC-PWM instances must be synchronized every time any of them is relocated to a different position on the FPGA. The implemented synchronization mechanism (SYNC) is based on a chain of flip-flops. These are the only flip-flops that are not masked to respond to the FPGA’s global set reset (GSR) internal signal, and therefore they generate a simultaneous reset pulse in the three TC-PWM instances when the GSR signal is toggled by the HWμK. The proper instant to synchronize the triplicated TC-PWM instances is when the reference sinusoidal crosses through 0. This instant is computed using zero cross predictor logic, accounting for the (fixed and known) delay introduced by R3TOS when the GSR signal is toggled, and written to a set of sync_flag LUTs.

Figure 2 shows the FPGA implementation of the TC-PWM circuit, spanning the bottom half of a clock region and being clocked from a local clock buffer (that is, BUFR). The I/O LUTs, as well as the SYNC logic, are allocated to dedicated columns in the rightmost and leftmost edges of the circuit. Overall, the TC-PWM circuit uses 182 CLBs, one BRAM, and three DSP blocks. The capability delivered by R3TOS for managing FPGA resources at the finest chip granularity increases the vertical relocatability of TC-PWM circuit instances (that is, enhances system flexibility), thus increasing the chances of finding an allocation for the three TC-PWM instances in which all damaged on-chip resources are circumvented. Namely, R3TOS brings a 4× relocatability improvement factor over the state of the art when using Virtex-4 FPGAs, as there are four BRAMs in a Virtex-4 clock region, but this factor can be as high as 20× when using latest 7-Series FPGAs.

I/O static circuitry

The input circuit handles the FPGA input pins through which the measured and target train speed values are received. Likewise, the output circuit handles the FPGA output pins



Interruption to R3TOS HWuK
$T_{TC\text{-}PWM\text{-}CLK}$ = 500 ns
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
T_{ICAP} = 8 µs
1.3 µs
1.3 µs
1.3 µs
0.2 µs
1.1 µs
2.8 µs
Read PWM LUTs of TC-PWM instance 1
Read PWM LUTs of TC-PWM instance 2
Read PWM LUTs of TC-PWM instance 3
Voting
Write PWM bits to output LUTs
Custom operation
Config. RBF
Config. WSF2S
Config.ICAP template
Configuration layer
Computation layer
Operation type A (data delivery):
1.4 µs
Read input LUT
Write input data to TC-PWM instances
Config. WMF2S
Operation type B (read-back):
1.5 µs
Read frame
Computation
Operation type C (write):
0.4 µs
Write frame
Comp.
Operation type D (computing):
2.6 µs
Operation type E (sync.):
0.3 µs
2.3 µs
GSR
Idle

Figure 3. Reliable Reconfigurable Real-Time Operating System (R3TOS)-driven functioning: internal configuration access port phases.

through which the PWM control signals are issued. The latter circuit is based on LUTs that mimic the scheme used in the TC-PWM circuit—that is, a pair of LUTs working in a ping-pong fashion, even and odd, alternating in operation every 8 μs. These LUTs use the same clock signal as the TC-PWM circuit, thus delivering a PWM sample to the IGBTs every 500 ns.

R3TOS-driven functioning

In this application, R3TOS is responsible for carrying out ICAP-based communications between the TC-PWM circuit instances and the static I/O LUTs, as well as ensuring that the system is functioning correctly. As Figure 3 shows, this functionality is implemented in cyclic ICAP phases of duration T_{ICAP}. It is vitally important to ensure the ICAP phases are correctly synchronized with the functioning of TC-PWM circuitry, namely with the output LUT ping-pong switch. Hence, the ICAP phases must span exactly 16 TC-PWM clock cycles (that is, 8 μs).

As Figure 3 shows, every ICAP phase allocates time for both reading back the PWM bits from the output LUTs in the TC-PWM circuitry and writing the majority-voted values to static LUTs connected to FPGA output pins. Additionally, each ICAP phase allocates a time slot for carrying out other custom operations that can be executed at a slower rate. These operations include delivering input data to the TC-PWM circuitry at every control loop, checking the configuration correctness of the R3TOS HWμK, relocating TC-PWM circuit instances, and injecting faults into the system. To reduce the time overheads, the configuration templates of the next operations to be executed are adjusted while the previous operations are executed by the FPGA's configuration logic (see the dotted-line rectangles in Figure 3).

There are five types of operations:

- *Type A operations (data delivery).* Type A operations invoke RBF and WMF2S configuration functions and are periodically executed at every T_{LOOP} to deliver the same input speed values to the three TC-PWM circuit instances.
- *Type B operations (read back).* Type B operations invoke an RBF configuration function and allow some computing time. They are used to start a read-modify-write cycle when followed by a Type C operation. The computing time can be used to check the correctness of the ECC codes associated with the read-back frame, invert a bit in the read-back frame when injecting a fault, or modify some specific configuration data when relocating a TC-PWM instance (for example, ECC values in a frame).

- *Type C operations (write).* Type C operations invoke WSF2S and RBF configuration functions and allow some computing time. Like Type B operations, these are used when scrubbing a soft error, injecting faults, or relocating a TC-PWM circuit.
- *Type D operations (computing).* Type D operations extend the computing time of Type B and C operations. This can be used for decoding a Frame_ECC syndrome to find the upset bit in a frame, or for computing a relocated frame's ECC values.
- *Type E operations (synchronization).* Type E operations are used to toggle the GSR signal to synchronize the three TC-PWM instances when one of them is relocated. Because of the strict timing requirements, Type E operations are executed only when a synchronization instant is detected, and they have the highest priority.

The functioning of the R3TOS-based traction controller can be thus compared with a (highly predictable) time-triggered system, where the micro-tick period is equal to $T_{\mathrm{TC-PWM-CLK}}$ and the macro-tick period is equal to T_{ICAP}.

Experimental results

To characterize the fault-tolerance capabilities of the R3TOS-based traction controller prototype, which was implemented on a Xilinx VC4LX160 FPGA, we conducted a set of fault-injection campaigns. We pseudorandomly injected the faults at equal time intervals, and only in the used FPGA area (that is, TC-PWM circuit instances and the HWμK). Namely, a list with the relative position within the HWμK and TC-PWM circuit of the configuration bits to be corrupted was randomly generated offline following a uniform fault distribution and used online to determine the position of the injected faults based on the location of the TC-PWM circuit instances at each time. For simplicity's sake, the used fault model neglected the fact that different FPGA resources have distinct fault rates, but it did consider the fact that most used FPGA resources are more prone to failure. The injected faults to the R3TOS HWμK were assumed to be correctable (that is, soft errors), whereas five per 1,000 of the faults injected to the TC-PWM circuitry were stuck-at faults aimed at modeling noncorrectable hardware damage. To simplify the implementation, no diagnostic test was used to detect the simulated damage; the corresponding resources were directly identified as damaged to the R3TOS allocator.

The fault-injection campaigns were conducted both on the developed R3TOS-based traction controller and a standard controller protected with static TMR that does not implement any recovery mechanism. The comparison between these two antagonistic approaches (flexible yet complex versus fixed and simple) is useful for evaluating the influence of R3TOS HWμK on reliability. Note that, unlike in the R3TOS-based controller, the critical part of the standard controller (that is, the voter) is a nearly negligible portion of the system. We let both systems run for 1 minute, repeating the experiments up to 25 times for each fault rate. We considered that a failure occurred when all of the error_flags of the TC-PWM circuit instances were activated, all the TC-PWM circuit instances generated different PWM bits, or the R3TOS HWμK was stuck.

Figure 4 depicts the average failure rate observed in both systems. Although this grows in the standard controller with the injected fault rate, it is only significant (that is, greater than 40 percent) in the R3TOS-based controller for fault-injection rates higher than 32 faults per second. For higher injection rates, the time needed to completely check (and scrub) the configuration of the R3TOS HWμK, which is on the order of tens of milliseconds, approaches the time interval at which faults are injected, and therefore, faults start to accumulate and eventually lead to an R3TOS HWμK failure. Nonetheless, the failure rate of the R3TOS-based controller is still 40 percent lower than that of the standard controller. On the other hand, for injection rates lower than 32 faults per second, the R3TOS-based controller fails only when faults randomly affect critical elements in the HWμK (that is, less than 10 percent of the time), showing an improvement in the failure rate greater than 80

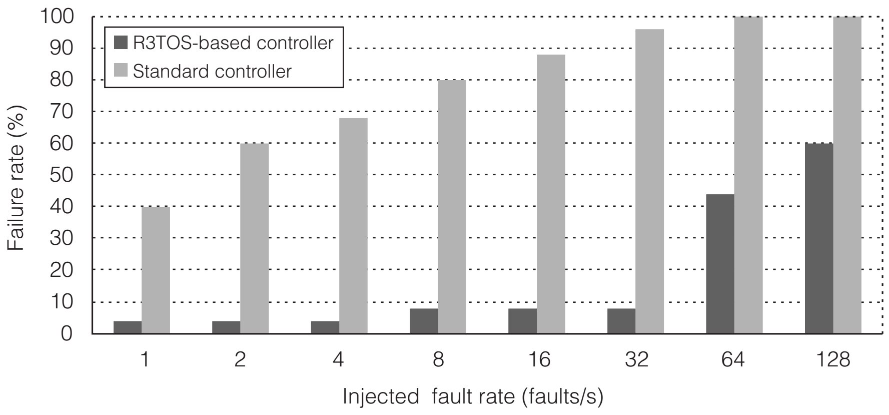


Figure 4. Observed failure rate in fault-injection campaigns. The results show that R3TOS reduces the failure rate, especially when exposing the controller to units of upsets per second.

percent with regard to the standard controller. A fault rate as high as tens of upsets per second is aggressive. For instance, the highest amount of upsets measured by NASA's spacecraft operating at low Earth orbit (700 km) in one day is about 1,200 (less than one upset per minute).[6]

These results are promising because they show that the benefit derived from the increased TL2 flexibility provided by R3TOS is far more important than its vulnerability, owing to the existence of a single point of failure, which indeed seems to be robust enough. We posit that currently available quasi-TL3 approaches (such as competitive runtime reconfiguration[7]) that also have a single point of failure (that is, a reconfiguration controller) would show worse behavior, owing to the long time required to converge into a valid solution, if reached. Even worse, in safety-critical applications—such as the railway traction controller we present in this article—using randomly generated FPGA configurations that are not certified is completely forbidden by certification entities. Nonetheless, in noncritical applications, these approaches could be a last chance to recover from situations where accumulated faults have completely ruined the system configuration, or when the FPGA is notably damaged and it is impossible to find enough adjacent functional resources for the relocatable circuitry.

Future work includes extending the TL2 AFTS prototype toward adaptivity and efficiency, as well as testing it in a real harsh environment, such as space. In this vein, we are providing support for using R3TOS in the SUPSAT CubeSat (http://supsat.eu), where the traction-controlling circuit we present in this article might well be replaced with a larger set of circuits, each specialized in efficiently processing each type of data collected from the sensorized environment. Onboard data processing will enable the CubeSat to autonomously decide what to do next (that is, which circuit to configure next), while also helping it to improve energy efficiency, because only the significant pieces of information will need to be sent down to Earth. In this context, R3TOS could help to extend the lifetime of the CubeSat as the device ages and degrades by reassigning the FPGA resources assigned to peripheral func-tionality to vital computations.

References

1. X. Iturbe et al., "R3TOS: A Novel Reliable Reconfigurable Real-Time Operating System for Highly Adaptive, Efficient and

Dependable Computing on FPGAs," *IEEE Trans. Computers*, vol. 62, no. 8, 2013, pp. 1542-1556.

2. X. Iturbe et al., "Runtime Scheduling, Allocation and Execution of Real-Time Hardware Tasks onto Xilinx FPGAs Subject to Fault Occurrence," *Int'l J. Reconfigurable Computing*, vol. 2013, 2013, article 905057.
3. X. Iturbe et al., "Online Clock Routing in Xilinx FPGAs for High-Performance and Reliability," *Proc. NASA/ESA Conf. Adaptive Hardware and Systems*, 2012, pp. 85-91.
4. A. Ebrahim, T. Arslan, and X. Iturbe, "A Fast and Scalable FPGA Damage Diagnostic Service for R3TOS Using BIST Cloning Technique," *Proc. Int'l Conf. Field-Programmable Logic and Applications*, 2014, doi:10.1109/FPL.2014.6927386.
5. U. Viscarret et al., "Design of Power Electronic Building Blocks (PEBB) for MultiMW Modular Traction Converters," *Proc. IEEE Energy Conversion Congress and Exposition*, 2010, pp. 4217-4222.
6. C. Poivey et al., "In-Flight Observations of Long-Term Single Event Effect Performance on Orbview-2 and Xray Timing Explorer Solid State Recorders," *Proc. IEEE Nuclear and Space Radiation Effects Conf.*, 2003, article 20030025777.
7. R.F. Demara and K. Zhang, "Autonomous FPGA Fault-Handling Through Competitive Runtime Reconfiguration," *Proc. NASA/DoD Conf. Evolvable Hardware*, 2005, pp. 109-116.