

A Low Complexity Method for Detecting Configuration Upset in SRAM Based FPGAs

R. J. Andraka, P.E. and J. L. Brady
Andraka Consulting Group, Inc. 16 Arcadia Drive, North Kingstown, RI 02852

Abstract

Field Programmable Gate Arrays (FPGAs) have brought reasonably priced, high performance processing to applications that previously could not be done without expensive custom logic. This has opened the door to a variety of experiments that were previously too expensive to accomplish. The application of FPGAs to environments where upset is likely has been slower because of the high logic cost associated with mitigating upset. The concern over upset extends to the configuration program for the reconfigurable (SRAM based) FPGAs. Fortunately or unfortunately, the devices best suited to signal processing and the larger logic density devices tend to be the SRAM (Static Random Access Memory) based ones. The complexity of configuration integrity monitoring has hindered application of these devices in environments where upset is likely.

The traditional configuration monitoring approach has been to read back the configuration and check it against the intended configuration. This is time consuming, requires a separate controller, and is not compatible with logic memory elements in the Xilinx devices (shift register and memory implemented in the Configurable Logic Blocks, called CLBs). Our design, a high data rate 4096 point block floating point Fast Fourier Transform (FFT), uses those memory elements extensively in order to fit within the device size and power limitations. Rather than using the traditional read back approach, we verify the logic by periodically sending test vectors through it and checking the results. We have borrowed from the built in self test community to arrive at a novel approach using test vectors to detect changes from a baseline rather than checking for logic correctness. This leads to a surprisingly simple implementation consisting of a linear feedback shift register to generate a known pattern and a cyclic redundancy check signature to verify the output matches the baseline output.

I. Introduction

Single event upset (SEU) occurs as a consequence of radiation exposure injecting enough energy to toggle the state of a flip-flop or memory cell in a digital design. In harsh environments such as earth orbit, upsets can be quite frequent and therefore some means of detecting and recovering from such upsets are required for a reliable system^[1]. FPGAs are perhaps more vulnerable than traditional digital logic because they are effectively a

design on top of another design (the underlying design being the physical infrastructure of the FPGA which includes all the configuration storage and control logic). State within the underlying infrastructure defines the user design by setting switches that determine logic cell function and interconnect routing. Upsets within the user design can be dealt with using a variety of conventional means, including things as simple as just allowing the upset to flush out in the case of some data path designs.

The infrastructure on the other hand, is generally not very accessible to the user, making it difficult to probe for upset. This is compounded by the fact that the configuration state is typically static once configuration is complete; upsets remain until the device is reconfigured. Since the configuration state determines the logic function and connections within the user portion of the design, upsets here have a much more profound effect than simply altering the state of the user circuit. The circuit topology can actually change as a result of upset. While the ability for circuit reconfiguration is handy when it is intentional, it can wreak havoc when it happens as the result of upset. Such unintentional reconfiguration makes the effects of upset on a circuit's outputs much harder to predict than it would be for a similar design in another medium. Because the infrastructure is inaccessible, static, and has so much influence on the user design, SEU mitigation for the configuration state demands special attention.

Ideally, FPGAs would have some built in SEU mitigation on the configuration layer. Current FPGAs, however, do not have such capabilities (with the exception of an Actel device). Configuration upsets, therefore must be detected either by observing the effects on the user design, or by reading back the configuration to compare it against a reference.

II. Traditional mitigation methods

Triple mode redundancy

The gold standard in SEU detection and mitigation is Triple Mode Redundancy (TMR), which is essentially a majority vote^[2] on the results from three independent copies of the protected logic as shown in Figure 1. Triple mode redundancy will mitigate against misbehaving logic due to both user circuit faults and configuration upset, provided the failure does not affect the voting logic. While it is robust, it also requires more than three times the logic of an unprotected circuit to implement. The increased logic translates to a requirement for more or larger FPGAs to realize a given function. More logic also means increased power dissipation and larger power supplies. This added

cost and weight may be acceptable for small designs, but for larger FPGA designs such as our 4096 point FFT, it can be prohibitive. In our case both power dissipation limits and available resources prevent the use of TMR

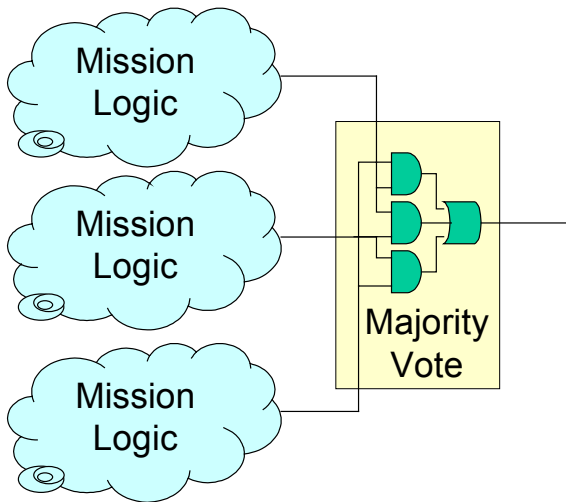


Figure 1. Triple mode redundancy requires three copies of the hardware plus majority voting logic.

Fortunately, transient errors can be tolerated in many data path intensive designs, particularly those dedicated to remote sensing. In these cases, the transient error manifests itself as noise in the data set, which can either be minimized through post processing such as averaging, or can be filtered out by discarding data sets that fall outside of some prescribed bounds. The effect of an upset to the user data path gets flushed out of the processor after a finite number of clocks. If such transient upsets can be tolerated, then only the upsets that result in a persisting error need to be dealt with. Typically, this involves only a very small amount of control logic in the user circuit and the FPGA configuration itself. That control logic can be protected either through triple mode redundancy or just that portion of the circuit or often through a simpler scheme such as illegal state recovery. The FPGA infrastructure is a bit tougher to protect, since it is intentionally hidden from the user. Triple mode redundancy on the entire user design just to protect the configuration is expensive overkill. Other methods of monitoring the configuration can provide an acceptable level of protection at a considerably reduced cost.

Configuration readback

Some FPGAs, such as the Xilinx Virtex™ families, permit read back of the configuration bitstream. A popular method of monitoring the configuration compares configuration read back from the FPGA against a reference to determine if the configuration has changed. If the comparison fails, the device is reconfigured to restore the original configuration. Configuration read back does have some limitations however, some of which place rather severe restrictions on the FPGA user design.

In particular, the Xilinx Virtex family and its derivatives have read back problems associated with the memory elements in the FPGA. Block RAM read back destroys data in the block RAM, so it cannot be included in the read back sequence while the FPGA design is also using the memory. Memory implemented in the logic fabric, either as random access memory or as 'SRL16' shift registers cannot be read back with the clock running^[3]. A quirk in the FPGA design causes the cell configuration to change if this is violated. Since read back is on a by column basis, any columns containing 'SRL16' or 'RAM16' primitives cannot be included in the read back sequence while a device is being clocked. These primitives either need to be avoided entirely in the user design, or their locations have to be known and those columns excluded during read back. The SRL16 primitives are extremely useful in signal processing designs, as they are useful for compact delay queues, reordering circuits, as well as for reprogrammable constants and tables. Our FFT design depends heavily upon them for the design speed and density.

Configuration scrubbing

A variation on configuration read back is configuration scrubbing. Configuration scrubbing takes advantage of the partial reconfiguration capabilities of the device to periodically reconfigure the device one column at a time while the device is running. It is similar in concept to memory scrubbing where memory protected by error detection and correction circuits is periodically read and the data repaired if there is a read error. Configuration scrubbing can either be done blindly where the configuration is simply re-written, or as the result of read back testing. Care must be taken to not overwrite the state of memories or flip-flops however so that the design operation is not upset. The same Virtex™ device restrictions mentioned above for memory elements also apply.

Both read back and configuration scrubbing require a fairly sophisticated external (to the FPGA) controller to handle the configuration, read back, and timing. Read back also will not detect stuck at faults in individual flip-flops that would be detected with methods that check the data coming through the data path. Methods including read back are preferable over blind scrubbing because it provides an indication of a configuration failure to downstream processing where a blind configuration does not.

'Cowboy' mode

The troubles with read back of certain Xilinx features, under-appreciation of the problem, cost considerations, and just plain laziness has led to too many instances of what our sponsor affectionately calls "cowboy mode" for its carefree approach. Here, there is no checking of the configuration, instead the device gets a wholesale reconfiguration only after an obvious failure is observed in the output data. The failure is generally detected by a persistent error in received data sets. Unfortunately, a subtle error can go undetected and can considerably skew the reduced data. "Cowboy mode" mitigation is akin to mending the fence after the

horses have all escaped. We cannot recommend this technique for those planning on long careers.

III. An alternative: periodic test vectors

If we can tolerate transient errors in our data path, we can detect the persistent errors by monitoring the logic function using techniques borrowed from the Built-In Self Test (BIST) community. The techniques introduced here are not new, but they are also not typically applied to SEU detection.

Test vectors

Built in self test typically checks design function by stimulating the design with a set of canned test vectors, and checking the circuit outputs against a set of expected output vectors^[7]. This can be automated by including stimulus and check vectors in Read-Only Memory (ROM) added to the design along with addressing, data path steering and comparison logic as shown in Figure 2. In order to provide continued protection, the design must periodically run these test vectors between sets of operational data. If the process already has slack time in it, that time can be used to ‘hide’ the test vectors in the normal process cycle. Otherwise, extra time has to be inserted in the cycle to accommodate the test vectors. In the event of a test vector mismatch, the data collected since the last test vector is discarded since the integrity of the processor during that interval is not known.

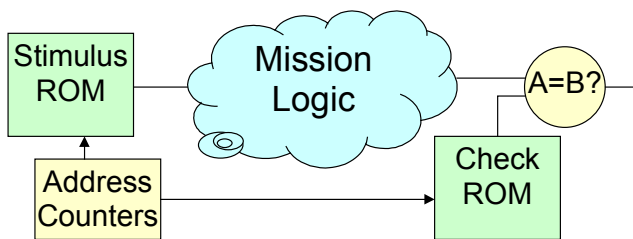


Figure 2. A test ROM can stimulate the logic with canned test vectors during idle periods. The circuit outputs can be checked against a second ROM.

Signature analysis

Unlike design debug, we are not concerned with determining logic correctness, rather we are only concerned with whether the logic has changed from a baseline or not. We can take advantage of this by reducing the output to a computed ‘signature’ and comparing just that signature rather than comparing each sample of the output. Proper selection of the signature computation will provide a signature match only when the all of the inputs to the signature computation circuit exactly match the expected inputs. A commonly used signature, based on modular polynomial products, is the cyclic redundancy check or CRC. This particular signature method is popular because the hardware is relatively simple, and the signatures are unique for each input pattern (provided the CRC width is sufficient for the block size). With this in mind, we can replace the check ROM, its address logic, and the high speed comparison in Figure 2 with a CRC circuit. The

CRC is simply a register preceded by an exclusive-OR network with inputs from the register and from the outputs of the circuit under test. CRC circuits are characterized by the polynomial, of which there are several commonly used in data integrity checking. We selected the 16 bit CRC used in the X25 modem protocol ($1+x^5+x^{12}+x^{16}$) since 16 bits provided enough width for our vector length and kept the signature computation logic reasonably lean. Other CRC polynomials could have been used just as effectively.

We can also eliminate the stimulus ROM and its address logic by substituting an algorithmic test pattern generator for the canned test vectors stored in the stimulus ROM in Figure 2. One such generator, a linear feedback shift register counter (LFSR), produces a pseudo-random vector sequence. The combination of the LFSR stimulator and the CRC signature analysis is commonly used in BIST applications^[6,7], so much so that Hewlett Packard (now Agilent) made test equipment based on this technique^[4]. The signature analysis method is illustrated in Figure 3. We seed the LFSR with a non-zero value to avoid the initial states, which tend to look like a shift register shifting 1’s into the input word on successive samples. The width of the LFSR and it’s feedback polynomial determine the length of the pseudo-random sequence it generates. We used a 16 bit LFSR, which generates a 65535 sample sequence. Our design only generates a 256 sample test vector, so we seeded the LFSR with a random pattern.



Figure 3. Signature analysis method replaces stimulation ROM with a test pattern generator, and the compare ROM with a sequential signature check circuit.

A fault coverage analysis (also using standard techniques from the BIST community) should be performed using the test data set applied to the design to ensure adequate fault coverage regardless of the method of generating the test vectors. Our empirical testing to date indicates that the pseudo-random sequences sufficiently exercise the 16 point FFT kernel and associated phase rotation logic in our design.

There are some special cases where a different set of test vectors might be more appropriate. For example, our design contains a 2730 sample shift register for buffering input data to align two data sets for a single FFT. In order to reduce power, we use a test vector that is constant for many clocks in order to reduce the power. We monitor the output of the shift register for the timing of the transition between two constant patterns and to make sure all of the bits out match the constant inputs. The second constant pattern is the one’s complement of the first to simplify the

check logic. This set up adequately checks the function of the shift register while keeping the power dissipation due to logic transitions to a minimum.

IV. Application

The signature analysis and related test vector techniques are applicable for designs that process data that is not safety-critical, such as the preprocessing of remote sensor data including radar, imaging and intelligence applications. In all of these applications, the on-board processing does some data reduction to limit the bandwidth required of the telemetry link to ground stations where the data is used or stored. Transient anomalies due to upsets in this type of sensor processing are quickly flushed out of the processing pipeline and affect only a few samples of data. Because of the localization of the transient anomalies, they can usually be tolerated or filtered out in subsequent processing with minimal data loss. Persistent errors (like those caused by a configuration upset) on the other hand, do not flush out of the pipeline and can therefore cause loss of all data until they are detected and corrected. While signature analysis is not suited for detecting transient faults, it excels at detecting persistent errors. The methods presented here serve well in this scenario with very little added complexity.

These techniques are not appropriate for safety critical circuits where even a momentary upset can cause loss of life or of the mission such as in attitude control systems. In these cases, a more extensive mitigation scheme, such as triple mode redundancy is needed to avert disaster^[1].

V. Advantages

The signature analysis technique has the very distinct advantage of a very low hardware overhead for implementation. The relative sizes of the analysis logic to our FFT design can be seen in Figure 6. The low complexity translates directly to lower power and more resources available for the primary design. It also provides the not so obvious benefit of enabling the use of the Xilinx SRL16 shift register and LUT RAM primitives, which for signal processing applications can represent significant area and performance advantages. These primitives are not permitted in systems that rely solely on configuration read back or scrubbing for protection against configuration upset. Signature analysis also avoids the most of the complexity of the external configuration manager logic since it is contained entirely within the FPGA.

VI. Limitations

While the signature analysis technique does have some very attractive advantages, it also has some significant limitations that may make it inappropriate for a particular application. Signature analysis will not detect transient upsets unless they occur while the test vectors are being applied. Some applications, such as remote sensing can tolerate transient upsets, while others such as flight controls cannot. In cases where transient upsets can be tolerated,

some means may need to be provided for checking the data for reasonableness in order to eliminate some of the effects of transient upset. The signature analysis and test vector methods also require part of the process cycle to be dedicated to running test vectors instead of actual data. Unless there is already slack time built into the process, this means a higher processor throughput is needed to accommodate both the mission data and the test vectors. The higher processing bandwidth translates to increased power and possibly more expensive processor hardware or reduced processor capability. These methods only provide fault isolation to the bounds of the protected circuit (the circuit surrounded by the stimulus and check logic). Finer fault resolution requires the design to be broken down into smaller protected partitions. SEU mitigation using signature analysis has some part of the logic where an upset could prevent the upset indication from reaching the reconfiguration logic (in this case, the decode of the signature). Other methods such as TMR (in the voting circuit) suffer similar soft spots. Finally, the test vectors and signature analysis used need to be considered carefully to make sure there is adequate fault coverage and that the signature does not hide certain faults. The CRC is susceptible to aliasing when the value of a faulty sample is a multiple of the CRC polynomial and the fault is a burst error or is spaced by powers of 2 samples^[7]. Different CRC polynomials offer protection to different multiple fault scenarios. These should be considered when selecting a polynomial. It may be beneficial to use two different CRC polynomials in parallel to provide better multiple fault coverage; this is an area for future study. All CRCs with 2 or more terms in the polynomial will detect single bit errors.

VII. Application Example: a 4K block floating point FFT

As an application example of this technique, we present our design of a 4096 point block floating point FFT designed under contract to Los Alamos National Laboratories as part of a space based reconfigurable radio^[8]. This design occupies two identical FPGAs, each of which houses a 4096 point complex FFT set up to perform two 4K point real-only transforms using a complex 4096 point algorithm. A third FPGA is dedicated to converting the output to logarithmic format, performing the final reordering, and selecting only the frequency bins of interest for output from the module. The pair of FFT chips must keep up with a 100 MS/sec input data stream with a one-third (33%) overlap between successive input sets. The input stream is real-only data, so we perform a complex FFT with two input sets, then separate the results at the output.

The application is a low earth orbit application, so radiation tolerance is required. We used the Xilinx QPRO family, which is a radiation tolerant version of the original Xilinx Virtextm family. The largest member of the family is the XCV1000, a million gate device. The device architecture and timing characteristics are identical to the original

Virtex™ family in the slowest (-4) speed grade. The XCV1000 has 32 4096 bit dual port block memories and 12288 logic ‘slices’. A logic slice basically consists of a pair of 4 input programmable look up tables (LUTs), a pair of flip-flops, and some added gates for carry propagation for fast arithmetic logic. Due to thermal cycling considerations, the power dissipation for each FPGA may not exceed 6 Watts.

The input to the two FPGAs is sequenced so that two successive 4096 point real-only data sets are captured by one FPGA. While that FPGA is busy processing that set pair, the next two real-only data sets are captured by the other FPGA, then the cycle repeats with a period of 10922 100 MHz clocks. The FPGA delays the first set by 2730 clocks to align it with the second, then transforms the pair using a complex transform. The FFT results are separated into the two complex frequency domain data sets and converted to polar form before returning the result to memory. The FFT is performed in three passes through the data. Each pass performs a radix 16 FFT on the data, and rotates (twiddles) the phase of the result in preparation for the next pass as is required by the mixed radix^[9] algorithm. The processing pipeline consists of two nearly identical parallel paths, each handling half the data, in order to have sufficient time complete the three passes within the allowed 10922 clock cycle. A fourth pass simultaneously loads new data and extracts processed data from the memory, as shown in Figure 4. . The added pass for loading and unloading data leaves the FFT and rotator pipeline idle, so it provides an ideal time to run test vectors through the processing pipe.

The limited amount of memory available on the FPGA forces an in-place implementation of the FFT algorithm so that the memory is written in the same address order it is read from on each pass. The address order is permuted for each pass to achieve the necessary data reordering.

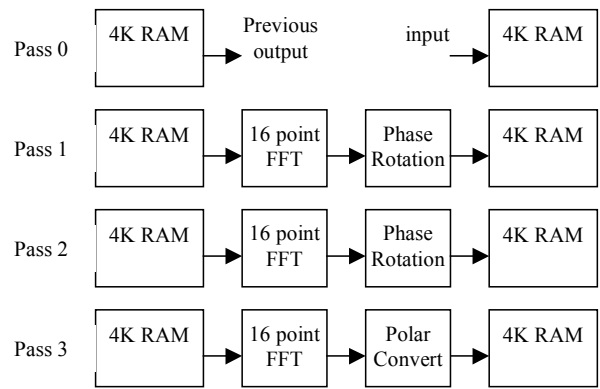


Figure 4. 4K point FFT is accomplished using 3 passes through a 16 point kernel and phase rotator. A fourth pass is used to load and unload data from the memory.

The 16 point FFT kernel is a very fast and compact fixed point hardware implementation of the Winograd 16 point FFT algorithm^[9,10,11] with 16 bit inputs and 21 bit outputs. We selected this algorithm over a more traditional Cooley-Tukey approach in order to achieve maximum performance with minimum area. This IP core depends heavily upon the Xilinx SRL16 shift register primitives for intermediate storage, data reordering, and matching delays. If that primitive were not available, the core would occupy more than double the area, and would likely not achieve the same level of performance. A Cooley-Tukey approach would have required either more passes with a smaller radix kernel, or additional intermediate storage in the form of either LUT RAM or SRL16 primitives. The increased size would make the full design too large to fit in the FPGA. Because of the limited resources, neither triple mode redundancy nor configuration readback are viable options for configuration monitoring. Figure 5 contains a block diagram of one FFT FPGA.

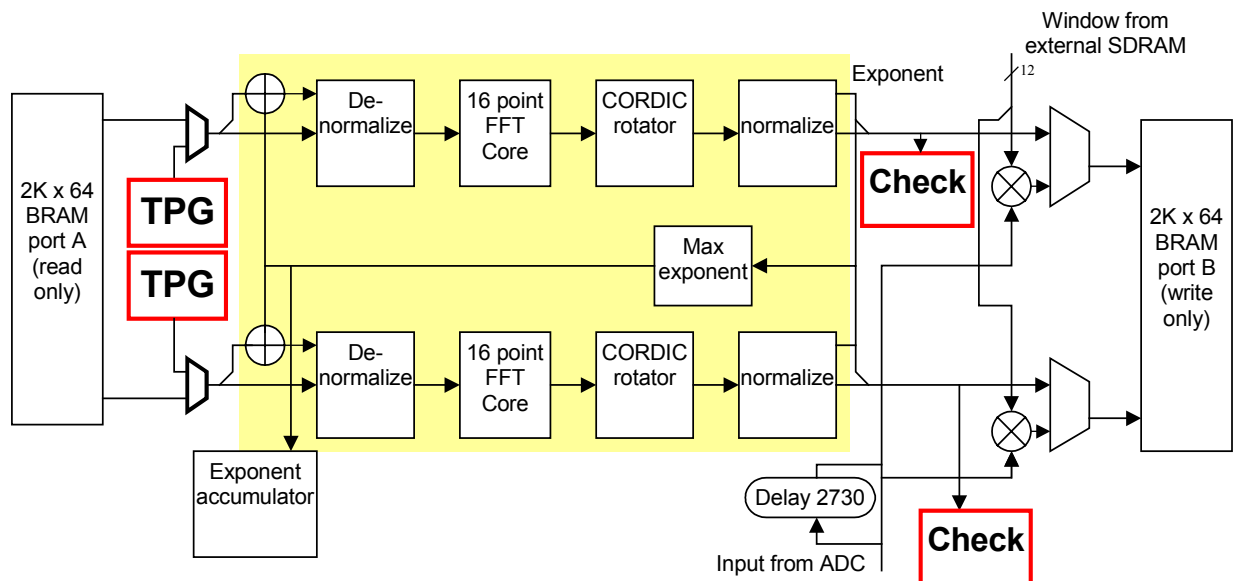


Figure 5. 4096 point block floating point FFT block diagram

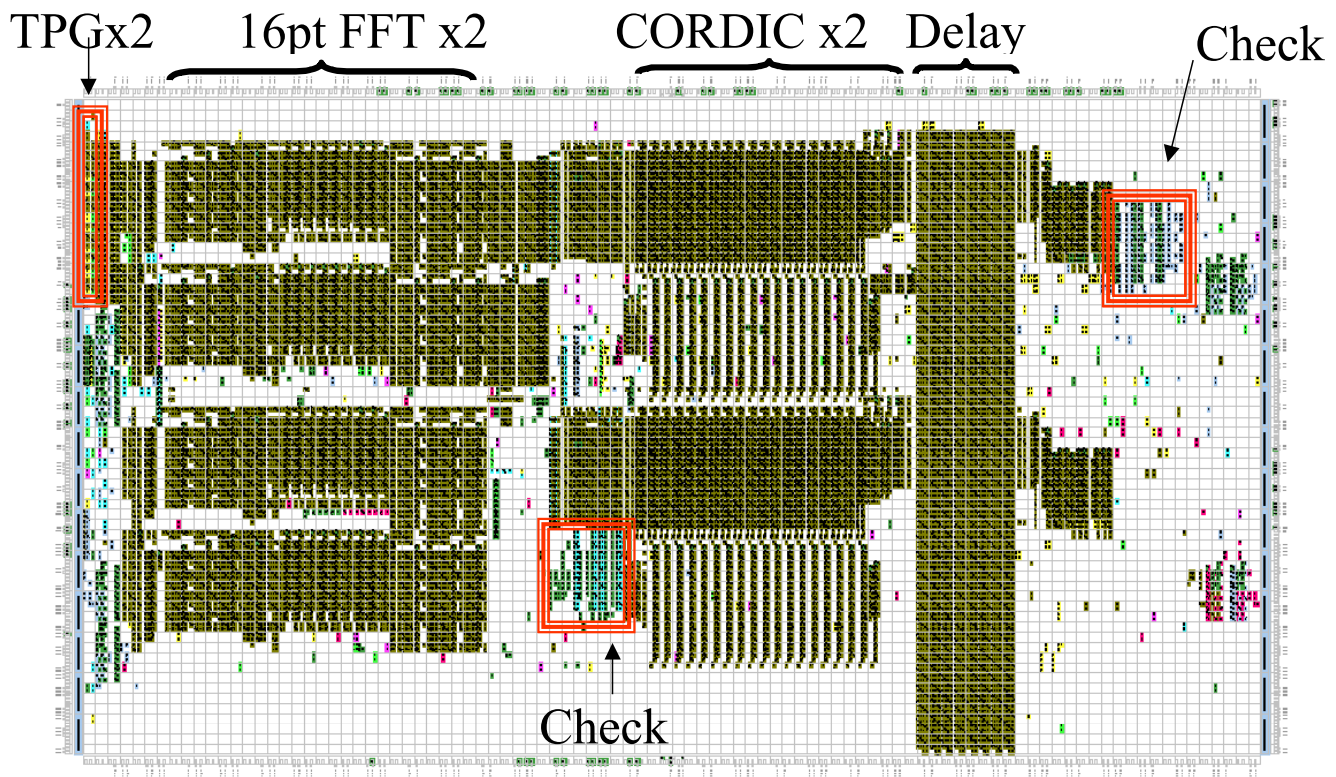


Figure 6. 4K FFT floorplan occupies approximately 45% of a Xilinx Virtex1000. The test pattern generators (TPG in the figure) and CRC check circuits are boxed to illustrate the relatively small size of the added logic. In this design, SRL 16 elements are restricted to specific columns to minimize unreadable columns so that the signature analysis could be augmented by read back analysis.

Configuration integrity is monitored using the signature analysis method proposed in this paper. The design is partitioned into three separate test regions, one for each of the two parallel process paths, and a third for the 2730 clock input delay. As of the time of this paper, we had not selected a test method for the unshaded logic in Figure 5. The test pattern generators for the data path are implemented as 16 bit linear feedback shift registers (LFSR) extended to provide a 32 bit output. We picked a random seed for the LFSR to avoid the all zeros state. These are shown as “TPG” in Figure 5. The check logic is a CRC-16 with the X25 Modem protocol polynomial, which is designed for detecting burst errors. We chose the X25 polynomial because we would expect a configuration error to appear as a burst error on the FFT output. The pipeline logic protected by the signature analysis is shown in the shaded area in Figure 5.

The delay queue is monitored by using a bit pattern and its inverse when the queue is not holding a set of input data. The bit pattern is held for a number of clock cycles, then inverted once and held one time in each 10922 clock cycle. A state machine and simple decoder check the integrity of the queue output and issues a fail signal if the bit pattern is not correct or arrives at the wrong time at the queue output.

We also heavily floorplanned the design for several reasons. First, floorplanning offers repeatable performance after recompilation of the design. By explicitly placing the elements, only the route is determined by the automatic tools,

so we can guarantee the design will meet performance even if it is later recompiled. Floorplanning also helps to minimize the routing by placing the logic so that connections are to nearby neighbors as much as possible. By minimizing the routing, we also minimize power dissipation. The floorplan also restricts the SRL16 primitives to specific columns in the FPGA so that we minimize the number of columns that cannot tolerate a configuration read back (21 of the 96 columns contain SRL16 primitives). The logic that is not protected by the signature analysis circuits (unshaded logic in Figure 5), as well as the single point of failure soft spot in the signature checking logic, is placed in columns that do not have SRL16s so that read back can check those circuits. The design floorplan is shown in Figure 6. The added signature analysis logic is highlighted to show its relatively small size.

VIII. Future work

While we have empirically tested the signature analysis for our application, a more rigorous fault coverage analysis still needs to be done to determine the level of protection actually provided. Some study into the effectiveness of various CRC polynomial choices and test pattern generators is also needed for various classes of pipelines to be protected. A look at using multiple signature circuits operating on the same data should also be investigated as a means of making multiple fault detection more robust.

IX. Conclusions

We have proposed a method borrowed from the Built-In Self Test community for detecting configuration upset in FPGA designs. This method offers considerably reduced complexity in applications where it is appropriate. The savings in complexity leaves more of the FPGA resources available for the primary purpose of the hardware, allowing for more processing capability per device. This technique also permits use of powerful device features that had been off limits using conventional configuration read back techniques. We've identified limitations of this method, particularly the inability to detect transient upset events and the possible sensitivity of the fault coverage to the test pattern and signature algorithms used. By using the techniques presented here, we were able to put a processor into an FPGA that would not have been possible using conventional SEU mitigation techniques. We feel that these ideas have a fairly broad applicability for space borne sensor programs.

X. Acknowledgements

The development of the 4K FFT design was done under contract for Los Alamos National Laboratories. The authors wish to thank Michael Caffrey, Mark Dunham, Paul Graham, Scott Robinson and the rest of the team at Los Alamos for the technical and financial support for the project leading to this paper.

XI. References

- [1] Brinkley, P. and Carmichael, C., "SEU Mitigation Design Techniques for the XQR4000XL," *XAPP181*, Vol 1.0, March 2000.
- [2] Fuller, E., Caffrey, M., Salazar, A., Carmichael, C., and Fabula, J., "Radiation Characterization, and SEU Mitigation, of the Virtex FPGA for Space Based Reconfigurable Computing," *MAPLD 2000 Proceedings*, p30, September 2000.
- [3] Carmichael, C., "Correcting Single-Event Upsets through Virtex Partial Reconfiguration," Xilinx Application Note XAPP216, June, 2000.
- [4] Chan, A.Y., "Easy-to-Use Signature Analyzer Accurately Troubleshoots Complex Logic Circuits," *Hewlett-Packard Journal*, pp.9-14, May 1977.
- [5] Smith, J.E., "Measure of the Effectiveness of Fault Signature Analysis," *IEEE Transactions on Computers*, C-29, No.6, pp.510-514, 1980.
- [6] Stallings, W., "*Data and Computer Communications*", Prentice Hall, pp 168, 1997.
- [7] McCluskey, E. J., "*Logic Design Principles with emphasis on Testable Semicustom Circuits*", Prentice Hall, pp466-469, 1986.
- [8] Caffrey, M. "A Space Based Reconfigurable Radio", *MAPLD 2002 Proceedings*, Sept 2002.
- [9] Smith, W. W. and Smith, J. M., "*Handbook of Real-Time Fast Fourier Transforms*", IEEE press, New York, 1995.
- [10] Winograd, S., "On Computing the Discrete Fourier Transform," *Mathematics of Computation*, Vol. 32, No. 141, pp. 175-199 (1978).
- [11] Blahut, R. E., "*Fast Algorithms for Digital Signal Processing*", Addison Wesley Longman, Inc, 1985