Robust sensor systems using evolvable hardware

James Hereford, Charles Pruitt
Murray State University
Murray, KY 42071
James.Hereford@murraystate.edu

Abstract

This paper describes a system that is robust with respect to sensor failure. The system utilizes multiple sensor inputs (three in this case) connected to a programmable device (FPAA) that averages the outputs from the sensors. The programmable device is programmed using evolvable hardware (EHW) techniques. If one or more of the input sensors fails, then the controller detects the failure and initiates a reprogramming of the circuit. The system then continues to operate with a reduced number of sensors.

The failure detection is accomplished by comparing the actual system output with a Kalman filter estimate of the output. If the actual output and the filter estimate differ by amount greater than the system uncertainty, then a failure is noted. The system is robust to several different failure modes: sensor fails as open circuit, sensor fails as short circuit, multiple sensors fail, FPAA input amplifier failure.

The experimental setup is described as well as results using simulated inputs (individual voltage sources) and actual temperature sensors (PRTDs). The paper also discusses some interesting results as well as issues that must be overcome to expand the system.

1.0 Introduction

The goal of this research is to develop a system that is resistant or tolerant of processor failures; that is, fault tolerant. Fault tolerance is the ability of electrical components to continue to function despite failures (faults) in the hardware. The possible faults will be originally limited to failures in one of the sensors but faults in other parts of the system (notably, one of the processing stages) will be considered. The result is a system that can suffer damage and recover without direct human intervention. The final demonstration is to have the system running then smash a sensor with a hammer or short out an input on the FPAA and have the system recover.

The hardware setup incorporates multiple (co-located) sensors instead of a single sensor in the electronic system. A processing system is then derived based on evolvable hardware principles to take the average of all N sensors as the system output. Step 2 is to let the system run indefinitely. For testing purposes, sensor failures are artificially introduced. Once a failure in a sensor is detected, the processing hardware is "re-designed" based on evolvable hardware principles to take the average of the input sensors.

Developing a system that is fault tolerant based on reprogramming will be a significant achievement. Other researchers have built systems that "repair" themselves autonomously [Sipper 2000] but to our knowledge no one has yet reprogrammed a broken system using Genetic Algorithm techniques. Other evolvable hardware applications have reprogrammed the hardware due to changes in the mission (e.g., robot controllers) [Thompson 1996]. A second new achievement would be the development of a system that is robust to sensor failures. This will be the first system that specifically includes additional sensor inputs and reprograms the hardware due to sensor failure.

This research builds on several recently-developed techniques and concepts. The first concept is the application of evolvable hardware techniques to FPAAs. Evolvable techniques have had some success when applied to FPAAs [Flockton 1998, Flockton 1999] but they were not used to develop fault tolerant systems. As mentioned previously, other researchers have investigated the use of evolvable hardware techniques to provide robust, fault-tolerant hardware [Thompson 1996, Ortega 1999, Keymeulen 2000, Canham 2002]. These systems, however, are different from the approach presented in this research proposal. Ortega et al. [Ortega 1999] do not use evolution as an adaptive repair mechanism, even though their approach is based on bio-inspired hardware. Keymeulen et al. [Keymeulen 2000] and Canham et al. [Canham 2002] incorporate fault tolerance into the design (training) of the hardware circuit. That is, they inject the known possible faults during the evolutionary process but do not address event changes (e.g., processor or sensor failure) or reprogramming.

2.0 System overview

2.1 System diagram

The goal is to evolve a circuit that takes the average of three input sensors. (See Figure 1.) The sensors could be any type (pressure, temperature, humidity, etc) but for this research temperature sensors will be used. Temperature sensors, both thermocouples and platinum resistive temperature detectors (PRTDs), are cheap and require little supporting circuitry. All of the temperature sensors will be located near each other on a metal bar (high thermal conductivity) that can be raised or lowered in temperature. The output from each sensor circuit is a small voltage signal (on the order of a few milliVolts) that will be input directly into the FPAA chip. Voltage levels are not a problem since FPAAs have built-in amplifers, if needed.

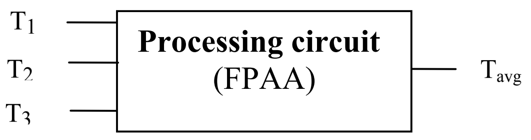

Figure 1: System level diagram. A circuit is evolved on an FPAA that takes the average of 3 input sensors.

Field Programmable Analog Arrays (FPAA) are a relatively new technology that is a spinoff of the more well-known digital Field Programmable Gate Array. The FPAA is a single chip that is divided into configurable blocks. Some FPAAs have multiple identical blocks that can be programmed to perform one of a variety of operations: addition, negation, log, antilog, amplify, differentiate, integrate, filter. Simpler FPAAs, such as the ispPAC30 from Lattice Semiconductor, have programmable input and output amplifiers with programmable interconnects among the amplifiers.

The architecture of the ispPAC30 is shown in Figure 2. The PAC30 contains four differential inputs, four input amplifiers (IA), two output amplifiers (OA), two internal adjustable voltage references, and two multiplying digital-analog converters (MDACs). Any input pin can be routed to any of the four input amplifiers or to the MDACs. In addition, either OA can be routed to any IA or MDAC. This allows the pac30 to be configured to do signal summation, feedback, or cascade gains. The IAs multiply their input by the gain setting; the gain can be set from -10 to +10 (excluding zero). Two IAs, IA1 and IA4, are equipped with an analog multiplexor that allows the input to the IA to come from one of two sources. In our experiments, one IA input was routed to an external input and the other input was left unconnected (input = 0V). By setting the multiplexor to use the unconnected input, we could effectively set the gain to zero for that IA.

The inputs into the two MDACs can be either external inputs, internal inputs, or fixed DC voltages (from the internal reference voltages). The multiplying function means that the MDAC input is attenuated by a value corresponding to an 8-bit code. The attenuation can range from –1.0 to +.99. The OAs can be configured to function as an amplifier, low-pass filter (with adjustable cutoff frequency), integrator or comparator. The output voltage is limited to 0V to +5V.

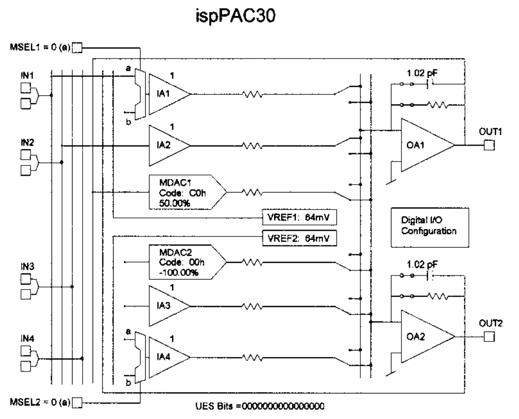

Figure 2: Architecture of the ispPAC30 FPAA from Lattice Semiconductor.

The FPAA is chosen rather than the FPGA because it eliminates the need for multiple Analog-to-Digital (A/D) converters. With 3 input sensors, the system would require 3 A/Ds. With the FPAA, the analog outputs from the temperature sensors can be sent directly into the FPAA; if amplification is required, then the FPAA can be programmed to provide amplification. In many cases, the final (averaged) temperature measurement needs to be interfaced to a digital circuit, but then only a single A/D is required. Reducing the number of A/Ds reduces the weight, cost, and power consumption of the entire system.

There are two main advantages for using evolvable hardware techniques to initially program the FPAA. First, applying evolutionary techniques to hardware design has led to many innovative designs in analog and digital circuits. Applying them here could also lead to a very innovative approach. Second, some fault tolerance arises just from the nature of the evolutionary process. The robust designs will "survive" and have a greater affect on future generations of designs than non-robust designs [Thompson 1996]. Thus, the design using EHW principles could be more robust than a design based on conventional human expert techniques.

Once a failure has been detected, the processing circuit needs to be reprogrammed to take the average of the remaining sensors. Since the goal is to have autonomous operation, it is assumed that that it is unknown which sensor has failed. Therefore, evolvable hardware techniques are applied to the now-damaged circuit.

To apply evolvable hardware techniques, a fitness function must be computed during reprogramming. To compute the fitness function, the last "good" data value will be used. That is, the previous two data points of the filtered output will be saved. If a fault is detected, then the proper output is known and is used to reprogram the averaging circuit.

We note that a side benefit of incorporating multiple sensors is that the final system output will have a smaller uncertainty (as measured by the standard deviation) than if only a single sensor was used. Specifically, if the N sensors have identical characteristics, each with a measurement standard deviation of σ, then the standard deviation of the measurement average will be $(1/\sqrt{N})$*σ. Even for a relatively small number of sensors, the overall measurement standard deviation drops quickly; if N = 5, the standard deviation drops to .45σ or less than half the standard deviation from a single sensor.

2.2 Failure Detection

An important step is to introduce a failure (or failures) into one of the sensors and subsequently detect the failure. Introducing a failure is relatively easy but detecting the failure is not a trivial matter. A sensor failure can be simulated by disconnecting the sensor or by more extreme methods (e.g., use a hammer). A failure in an FPAA module can be simulated by either setting the block to "Off" or externally grounding the inputs for that block.

The failure detection is accomplished by comparing the actual system (FPAA) output with a Kalman filter estimate of the output [Hereford 2004]. If the actual output and the filter estimate differ by amount greater than the system uncertainty, then a failure is noted. The predicted output, denoted by $\hat{x}^-$, is determined by computing an estimate for the next data point using a Kalman filter update from the previous data point. The system uncertainty is determined by the sum of the process noise, given by a variance-like parameter Q, and the measurement noise, given by the parameter R. If the FPAA output is denoted by *z*, then a failure is detected if

$$| z - \hat{x}^- | > Q + R .$$

See [Hereford 2004] for more background on the failure detection system.

It is difficult to measure Q and R. Therefore, for the robust sensor experiments we selected Q and R based on the expected output performance. We wanted to detect a drop of at least 10 %. Since the expected output of the averaging system is 0.4 V (at room temperature), that means Q + R must be equal to or less than 0.1*0.4 or 0.04 V. We set Q fairly low, Q = 0.01 V, because this allowed more flexibility in tuning the filter. This led to R = 0.03 V. We note that several different values for Q and R were used in this experiment. The performance of the algorithm, i.e., its ability to detect failure, was not sensitive to minor changes in the ratio between Q and R.

2.3 Evolvable algorithm overview

The evolvable algorithm used to program (and reprogram) the FPAA is a standard genetic algorithm [Goldberg 1989] with crossover and mutation. The selection method is roulette wheel selection. The probability of crossover is 0.7 and the probability of mutation is 1/8. The population size, *n*, and the number of generations, *ngen*, were varied based on formulas derived in [Hereford 2004a]. In general, for the 30 bit programming string, *n* was 40 and *ngen* was 15.

Elitism was used in all of the experiments to insure that the fittest member of each generation is guaranteed a spot in the following generation. The fittest member of the each generation shows up in the following generation as the last individual. This makes tracking the results of the experiment simple.

3.0 Results

Lattice Semiconductor provides graphical, drag and drop software that runs on a host PC to program the ispPAC30. Upon request, they also offered C-code that allowed us to program the device directly using a programming string of bits. A complete configuration of the PAC30 requires 112 bits. However, for our EHW experiments we used two shorter bit strings. The first string was only 20 bits and was used to evolve the gains on three input amplifiers (4 bits each) and one MDAC (8 bits). For the 20 bit case, the interconnections among the IAs, MDACS, and OAs were fixed. The second string length was 30 bits. This programming string evolved the gains on the 4 IAs, one MDAC and the two on-board multiplexors. Again the interconnections were fixed. We note that the interconnections among the PAC30 components could also be evolved but (a) it would take longer and (b) we would then have to monitor both outputs to check for proper configurations.

3.1 Results from voltage inputs

The first step was to successfully program the ispPAC30 with the 20-bit programming string. To program the three IAs and an MDAC, the 20-bit string is broken up into four segments. Eight bits are used to program the MDAC, and three groups of four bits each are used to program each of the IAs. All internal routing of the device was done manually by setting commands within the code. The 8-bit MDAC string allowed for 255 possible combinations, which correspond to an MDAC gain of –1.0 to +0.999. The IAs were limited to integer gain values from 1 to 10.

The three input voltages to the IAs were from a DC power supply. Three identical 50mV signals were used, and the desired (target) output was set at 500mV. The population size was set to twenty-five and ten runs were conducted. Brief results from these runs can be seen in Table 1. This table tracks the progression of each generation by showing the last member. Although thirty-five generations were used, only 14 are shown in the table. This is because every run conducted converged within 14 generations (in many cases, as few as 3 generations were needed). The average number of generations needed to converge to an acceptable solution (within 2% of the desired output) was 3.1. This means that, on average, only 75 out of over one million possible combinations were evaluated before an acceptable solution was found. Also, it took an average of 8.7 generations to converge to a solution that was within 1% of the desired output.

As a step up in complexity, the programming string was increased to 30 bits. The extra bits were used to set the gain on the fourth IA, set the polarity on all four IAs, and select which multiplexer would be used. Adding the polarity bit allowed each of the gains on the IAs to be set to integer values of –10 to 10, excluding zero.

The population size was increased to 40. Although the search space had increased, the average number of generations (based on 10 runs) needed to find a solution within 1% of the desired output was 7.2. It takes less than one second for each individual to be randomly generated, programmed to the chip, and evaluated. Therefore, the FPAA can be fully configured within four to five minutes; that is, it takes the evolutionary algorithm approximately four minutes to program the device and, if necessary, reprogram the device for a 20 bit programming string.

Several simulated failures were devised and tested. These include a single input failing as an open circuit, a single input failing as a short circuit, two inputs failing, three inputs failing, and a simulated IA failure. In all cases, the failure was detected successfully. For all failures except one, the circuit was reprogrammed to within 1.1% of the desired output. The only failure that the system was unable to recover from was when all three sensors failed.

3.2 Experimental Results

Once the system worked with fixed, stable voltages, we switched to real sensor inputs. Thermocouples were considered but ultimately rejected. The main problem with thermocouples is the fact that the voltage they produce at room temperature is very small (<1mV) and at times would even be negative. Programming the FPAA for a voltage near zero was very difficult since the global maximum (1 mV) for the genetic algorithm was at a step continuity (0 mV). Therefore, we chose to use platinum resistance temperature detectors, or PRTDs. There were several reasons for this decision.

First, the PRTD is a more linear device than the thermocouple, meaning that converting a voltage to a temperature is relatively simple. Second, the PRTD itself is just an element that changes resistance with changing temperature. It is not a self-powered sensor. Therefore, by placing it in one leg of a Wheatstone

Run Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1	0.541	0.542	0.507	0.51	0.51	0.505	0.5	0.49	0.49	0.49	0.492	0.492	0.507	0.5
2	0.466	0.463	0.503	0.507	0.506	0.503	0.502	0.505	0.507	0.505	0.505	0.505	0.506	0.505
3	0.476	0.523	0.509	0.503	0.504	0.505	0.502	0.5	0.504	0.502	0.5	0.5	0.499	0.501
4	0.636	0.508	0.507	0.505	0.503	0.5	0.499	0.5	0.502	0.502	0.5	0.501	0.502	0.501
5	0.417	0.417	0.507	0.505	0.505	0.497	0.497	0.498	0.495	0.496	0.501	0.497	0.497	0.497
6	0.503	0.505	0.505	0.51	0.508	0.506	0.507	0.506	0.505	0.509	0.505	0.505	0.507	0.505
7	0.482	0.482	0.506	0.506	0.506	0.506	0.506	0.506	0.506	0.498	0.498	0.502	0.502	0.502
8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
9	0.526	0.504	0.504	0.504	0.504	0.504	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
10	0.493	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Table 1. Best result from generations 1 through 14. Target output is 0.5 V. Results are shown for 10 separate GA runs.

bridge, we can tailor the output voltage to meet our requirements.

However, there are some drawbacks to using PRTDs. As stated above, the PRTD is not a self-powered sensor. Because of the fact that it is a resistance element and you must run current through it to receive an output voltage, there are self-heating problems that can arise from the I^2R power losses. Initially, we had planned to utilize the same 5V power supply that powers the FPAA chip to power a Wheatstone bridge circuit. However, this caused rather large (10+°C) errors in the temperature measurement. To alleviate the self-heating problem, the Wheatstone bridge circuit was powered by 1V. A diagram of this circuit is shown in Figure 3.

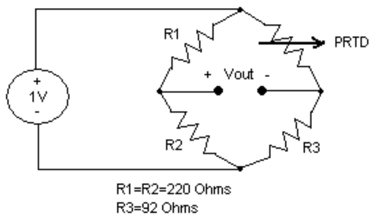

Figure 3: Wheatstone bridge circuit used with each PRTD sensor.

With the errors due to the PRTD at a minimum, three bridge circuits like the one in Figure 3 were constructed to provide the input voltages to the FPAA. The output of these bridges is on the order of 40 mV at room temperature. The next step was to run another experiment with simulated errors to test the ability of the program to recover when the inputs to the FPAA came from real temperature sensors.

In all of the experiments, the output voltage of each of the three bridges was measured before the initial programming step. Ten times the average of these three voltages was used as the target value during initial programming. (The target value is shown in the first column of Table 2.) The three bridge outputs were routed to input amplifiers IA1, IA2, and IA4 on the FPAA. Failures were introduced to input 1 (for one failure) and inputs 1 and 4 (for two failures), since the gains of IA1 and IA4 could be set to zero. Also, as in experiments with the DC voltages, the last acceptable output before the failure occurred was used as the new target value in the reprogramming step.

The failures in this set of experiments were the same as the ones used in the previous set of experiments. One or two PRTDs were replaced with either an open or short circuit to simulate different types of failure. When the PRTD fails as an open circuit, the output of the bridge is 0.5V; as a short circuit, the output is –0.5V. An FPAA failure was also simulated by disconnecting one of the lead wires on an IA. In addition to monitoring the ability of this system to recover from each of the introduced failures, we also noted the physical configuration of the FPAA after the original programming and after the reprogramming step had been completed. This allowed us to track what changes had been made to the configuration. Table 2 gives a brief description of each failure and the results obtained from the trial run. In addition to voltage outputs from the PAC30, the table also indicates how close to the target value the configuration is and the physical configuration of the chip after both programming and reprogramming.

Several conclusions are immediately apparent from Table 2. First, the reprogramming did not take longer than the initial programming. For the reprogramming step, we started with a random initial population and did not use the previous best result (or previous best generation) as a starting point. For an operational system, we would like quick reprogramming to (a) minimize down time and (b) reduce errors due to environment change. Second, the gain on the faulty sensor was reduced to zero or set to a small value for each failure. One notable example is the case where two PRTD sensors were set to open circuits. The Genetic Algorithm set the gains for the two damaged sensors to 0, even though it was unknown which sensors were damaged. To then reach the target output, the gain of the third IA and the MDAC had to be set to the maximum value. Third, the system is able to recover from any off-chip faults and even some on-chip faults. The last row in Table 2 shows the results from a simulated IA failure (the input leads were shorted out leading to the IA output always being zero). The system was able to detect a fault and then reprogram.

The only failure from which our system was not able to recover was total sensor failure. As long one sensor was operational, the reprogramming step was successful. For example, Figure 4 shows the programming and reprogramming steps of the experiment when two of the three inputs fail as open circuits. Each data point represents the last individual in each generation. This last individual is the fittest member from the previous generation. The dotted lines represent ±5% of the target value. As one can see, within three to four generations, a good solution has been reached. Further generations only serve to refine the solution. Ten generations were used in the initial programming step. The solution was within 0.05% of the target value. Then the failure was introduced and detected. The reprogramming step was initiated and arrived at a solution that was within 4% of the target value.

Target value	Output after initial programming, (% of target)	Setup after programming (IA1, IA2, IA4, MDAC)	# of generations to converge	Failure Type	Output after reprogramming, (% of target)	Setup after reprogramming (IA1, IA2, IA4, MDAC)	# of generations to converge
0.4147V	0.4128 V (0.46%)	-1, 8, 10, 0.566	3	1 PRTD open circuit	0.406923 V (1.417%)	0, 1, 10, 0.857	2
0.4147V	0.415554 V (0.206%)	3, -1, 10, 0.810	3	1 PRTD short circuit	0.416785 V (0.296%)	-3, -7, -7, 0.448	4
0.4196V	0.419595 V (0.0012%)	10, 10, 7, 0.37	4	2 PRTDs open circuit	0.41889 V (0.168%)	0, 10, 0, 0.999	8
0.4445V	0.448942 V (0.999%)	10, 4, 0, 0.739	3	2 PRTDs short cicuit	0.455566 V (1.475%)	-3, 10, 0, 0.228	4
0.4453V	0.447538 V (0.496%)	-4, 6, 10, 0.779	5	Simulated IA failure	0.448125 V (0.131%)	-2, 10, 6, 0.582	2

Table 2: Results before and after each of the failures. The left 4 columns show FPAA configuration and output after initial programming and the right 3 columns show the configuration after the failure and after reprogramming.

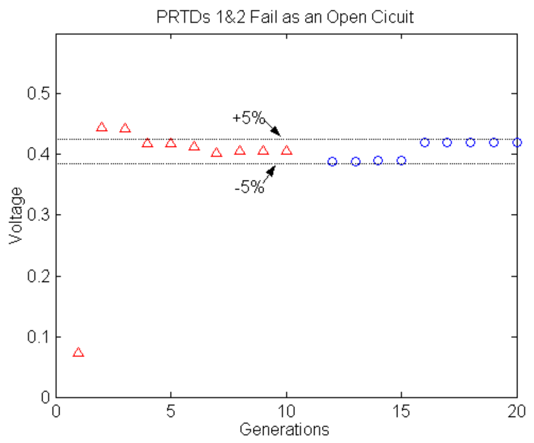

Figure 4. Error detection and reprogramming. Triangles show circuit output during initial programming. Failure (2 inputs fail as open circuits) occurs (output is off chart), then circles show circuit output during reprogramming.

4.0 Discussion

4.1 Interesting results

One of the interesting results of these experiments is the fact that many different FPAA configurations give the same output voltage. Thus, there are many redundant configurations that the genetic algorithm may find. For example, one particular solution may rely entirely on sensors 2 and 3. This solution is insensitive to a failure of sensor 1. If sensor 1 fails, no error is detected. This occurred once during one of the experiments conducted with the voltage inputs. After initial programming was complete, input 1 was disconnected. This resulted in a change of less than 1% in the output from the chip. Therefore, no error was detected. There are many other possibilities that could have similar results.

Another interesting observation is the ability of this system to recover from almost anything but complete sensor failure. If any failure causes the output to vary significantly, it will be detected and reprogramming will configure the device around this failure. And because the IA gains can be negative, failures that result in both positive and negative voltage inputs can be accounted for. Therefore, we can speculate that if a failure occurs within the chip itself that causes a fluctuation of the output voltage, and this failure is localized within one part of the chip (i.e. it only effects one or two of the inputs), not only will the error be detected, but the reprogramming step will be able to reconfigure the device around the error. Thus, only the inputs that remain unaffected will be used. If this is the case, then our temperature sensor system is not only tolerant to faults in the sensors, but also partial failures of the chip itself.

All of these results show that the system is adept at recovering from failures which take place after the initial programming. However, one issue not discussed above is the situation in which an error occurs during programming. The success of the programming step when this type of failure occurs depends upon when the error takes place. If it occurs in the first few generations, the program is able to recover and program to an acceptable solution. However, if it occurs in a later generation, the initial programming is generally not successful. At this point, the chip is configured to an output that is much different than the target output. The error detection algorithm then compares the FPAA output to the target output. Because they are different, an error is detected and reprogramming is initiated. Although the system is not able to detect this type of error the instant it occurs, it is able to recover well from it. This holds true for the situation in which the reprogramming step may not arrive at a configuration that is satisfactory. The output from this configuration will be seen as an error, and programming will initiate again.

Because the failure detection method described above compares the output to the last acceptable value, temperature fluctuations can be introduced after the chip has been initially configured. Therefore, the output of the chip may change dramatically, as long as it does not do so rapidly. Then, if there is a sensor failure at the new elevated or lowered temperature, the target value for the reprogramming will be the last acceptable value of the output. Thus, the system will be reconfigured to operate correctly within the new environment – not the one in which it was originally configured.

4.2 Issues

There are several issues that must be addressed if the robust system is to be expanded to include more sensors. A major issue that needs to be addressed is the type of programmable device. The ispPAC30 from Lattice Semiconductor has been good for the experiments presented in section 3. However, we would like to expand the number of input sensors. Having more input sensors would lead to a further reduction in the standard deviation of the output and thus better data quality. More sensors would also allow for more lead time before the system fails, which is good for graceful degradation applications. With small sensors, such as MEMS, available, the physical size of having many input sensors is not a problem.

Unfortunately, the ispPAC30 is limited to only four inputs. FPAAs from other manufacturers were investigated but they were either no longer being manufactured (e.g., Motorola MPAA020) or difficult to obtain (Anadigm). Lattice Semiconductor makes other FPAAs but none of them has 15 or 20 possible inputs. FPGAs, on the other hand, can have many possible inputs but then there is the added complexity of converting an analog output to a digital output for each sensor.

Another issue with the ispPAC30 is that the gains on two of the input amplifiers can not be set to 0. Some sensors, such as the PRTDs used here, are part of an input circuit. When the sensor is faulty, the input circuit will provide a voltage to the FPAA. The desire would be to "zero out" the input amplifier with the bad sensor. However, that is not possible with the ispPAC30.

A second issue is keeping the input constant during the reprogramming. It is assumed that the input can be kept constant during the initial programming since that could be done in a controlled environment. However, if the robust system is in an operational environment, then the input can not necessarily be kept fixed. For example, the input temperature may change during the reprogramming. Since we use the last good value as our target during reprogramming, we will program the device to the wrong value. This will lead to bias errors in the output.

5.0 Conclusions

We have developed a system that is robust or fault-tolerant. It programs a circuit (embodied on an FPAA) to average N input sensors, where N is three for our experimental results. It detects a failure using a Kalman filter approach, then reprograms the FPAA to take the average of the input sensors after failure.

The system is robust to several different failure modes: sensor fails as open circuit, sensor fails as short circuit, multiple sensors fail, FPAA input amplifier failure. The only input failure that the system can not recover from is when all of the sensors fail. An overall change in the input environment (e.g., the temperature increases) is expected and does not trigger a failure and a subsequent reprogramming.

Acknowlegements

The authors would like to acknowledge the support of the Kentucky NSF EPSCoR Research Enhancement Grant program and the Murray State University Committee on Institutional Studies and Research. We also would like to thank Lattice Semiconductor, especially Chris Dix and Ed Ramsden, for technical support related to the ispPAC30 and Pete Miller and Nick Galyen for help with the laboratory experiments.

References:

[Canham 2002] R. O. Canham, A. Tyrrell, "Evolved fault tolerance in evolvable hardware", IEEE Congress on Evolutionary Computation 2002, Honolulu, HI, 2002.

[Flockton 1998] S. J. Flockton, K. Sheehan, "Evolvable hardware systems using programmable analogue devices", *IEE Colloquium Digest*, pp. 511 – 516, 1998.

[Goldberg 1989] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[Hereford 2004] J. Hereford, N. Galyen, "Failure detection for multiple input system", *Proceedings of 2004 IEEE SoutheastCon*, Greensboro, NC, March 2004.

[Hereford 2004a] J. Hereford, D. Gwaltney, "Design space issues for intrinsic evolvable hardware", 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, June 2004.

[Keymeulen 2000] D. Keymeulen, A. Stoica, R. Zebulum, Y. Jin, V. Duong, "Fault-tolerant approaches based on evolvable hardware and using reconfigurable electronic devices", *Proceedings of the IEEE International Integrated Reliability Workshop*, pp. 32 – 39, 2000.

[Ortega 1999] C. Ortega, A. Tyrrell, "Biologically inspired fault tolerant architectures for real-time control applications," Control Engineering Practice, pp. 673-678, 1999.

[Sipper 2000] M. Sipper, E. Ronald, "A New Species of Hardware", IEEE Spectrum, pp. 59-64, March 2000.
S. Liu, L. E. Holloway, "Active sensing policies for stochastic systems", IEEE Transactions on Automatic Control, February 2002.

[Thompson 1996] A. Thompson, "Evolutionary techniques for fault tolerance", *Proc. UKACC Int. Conf. on Control*, pp. 693 – 698, 1996.