Autonomous System on a Chip Adaptation through Partial Runtime Reconfiguration

Matthew French, Erik Anderson, Dong-In Kang
University of Southern California, Information Sciences Institute
3811 N. Fairfax Dr., Suite 200, Arlington, VA 22203
{mfrench, erik, dkang}@east.isi.edu

Abstract
This paper presents a prototype autonomous signal processing system on a chip. The system is architected such that high performance digital signal processing occurs in the FPGA's configurable logic, while resulting higher level data products are monitored by cognitive algorithms residing on an embedded processor. The cognitive algorithms develop situational awareness about the platform's environment, and use this information to modify and tune signal processing in real-time using active partial reconfiguration. This system was realized on a Xilinx Virtex4 FX 100 device on a pulse measurement application utilizing a Bayesian Network cognitive algorithm. Changes in the RF environment were correctly detected 96.7% of the time and mitigation filters which resulted in at least 3dB interference rejection improvement were instanced 81% of the time. This system realizes a 71.4x reduction in size compared to static implementations and a 26-43x reduction in reaction times compared to human in the loop systems.

1. Introduction

Autonomous systems are designed to automatically modify their behavior as a response to a changing environment or goals. Such systems have become increasingly important with the rise in the pervasiveness of computing. Remote sensing platforms where human interaction or communication is limited such as in space based systems, unattended ground sensors (UGS), and unmanned underwater or airborne sensors and vehicles (UUVs, UAVs) can become much more productive if they can adapt to their changing environments. A number of research projects, in diverse areas of engineering, have taken up the challenge of designing autonomous systems, some examples include [1, 2]. This paper focuses on the design of autonomous systems by leveraging the inherent features available on modern System on a Programmable Chip (SoPC) devices.

System on a Chip (SoC) solutions in general offer a number of potential benefits such as reduced power, smaller form factor, and shorter design cycles by leveraging existing IP. In an autonomous system design, a SoPC becomes a natural extension of these ideas as SoPCs can update their hardware design, responding to external stimuli, without the need of additional chips or circuitry. To reach this goal, a number of challenges must be overcome. These challenges include designing the hardware/software interface for a SoPC to perform autonomous adaptation, developing algorithms to appropriately decide when a SoPC needs to be reconfigured, and reducing the feedback to configuration time of the autonomous SoPC.

This paper presents a working prototype of an autonomous signal processing system, implemented on a Xilinx Virtex 4 FX chip. This system is demonstrated using an RF pulse detection and analysis DSP application. The application's goal is to detect pulses amongst clutter and report the critical pulse heuristics such as pulse width, amplitude, frequency, and bandwidth. The RF environment is constantly changing, where a number of dynamic factors such as location of source and sensor, multi-path reflectors, heat, and man-made interferers can clutter the response. These factors can lead to incorrect determination of the parameters of a signal of interest (SOI), or even obscure the SOI completely. The system continuously monitors the environment and uses cognitive algorithms running on the PowerPC to develop situational awareness and identify the current type of interference, if any. Based on the determination of the environment, the cognitive control algorithm then selects the appropriate mitigation technique from a library of pre-compiled partial bitstreams and loads the bitstream into a reconfigurable region within the FPGA through the Internal Configuration Access Port (ICAP).

The signal processing community has long been researching algorithms to effectively suppress various interference types. A small sampling of the complex algorithms derived is FIR filtering, non-linear filtering, wavelet-based filtering, bilinear signal synthesis, and frequency-based nulling [3, 4, 5, 6]. Each of these however has its own strengths and weaknesses to different interference types such as narrowband, wideband, continuous wave, pulsed, partial band, and impulsive interference. From a signal processing point of view, the scope of this paper is not to improve upon any existing algorithms, but rather to demonstrate a system capable of inferring what type of interference exists in the current environment, and deploy the best mitigation technique during run-time.

The paper is organized as follows. Section 2 briefly discusses autonomous systems and their related work. In section 3 we present related and previous work on active partial reconfiguration, cognitive algorithms, and the DSP core used in the prototype autonomous signal processing
system. In section 4 we discuss the system implementation details. Finally, section 5 presents the results from the prototype system.

2. Autonomous Systems

Informally, an autonomous system solves an interactive problem without human control. More formally, we define a system to be autonomous if it has a continuous loop between three abstract components: system-processing, decision-making, and adaptation manager. This relationship is depicted in Figure 1. The system-processing component is the high performance processing work engine of the system. It accepts inputs and calculates results. It is also the portion of the autonomous system that may be reconfigured depending on the environment it is working in. To determine if a reconfiguration is needed the system processing component generates feedback information about its environment. These indicators can either be internal variables used in the system processing or the system processing outputs. The feedback indicators are passed to the decision making module. It is the decision making module that must interpret the feedback values and decide if an adaptation is needed. If an adaptation is needed it specifies and passes the necessary parameters to the adaptation manager. It is the adaptation manager’s job to create and deploy an adaptation meeting the specified parameters.

The development of autonomous systems is a relatively new field of study [7]. Most current systems consider hardware as static processing elements; each abstract component of an autonomous system is software based. For example, the HELI project [8] uses a software controller to stabilize the flight of a robotic helicopter, and in [9], the authors use an automaton to improve the characteristics of wireless sensor networks. Dynamic hardware that can be updated during runtime is a new concept in autonomous systems. Leveraging partial reconfiguration to adapt a system to a changing environment is just now being looked at. Two projects recently started, the AETHER [10] project and the ANDRES [11] projects are investigating how self-adaptive systems may be architected at the hardware/software co-design level using FPGAs. Steiner [12,13] has generated a working prototype of an autonomous system using the Virtex II that not only performs active partial reconfiguration, but also generates the partial bitstreams at runtime. His research is largely in the configuration component. The research presented in this paper presents an autonomous system leveraging FPGAs that successfully implemented each of the three components in Figure 1.

3. The Adaptable Signal Processing System Technology Background

The research presented in this paper is the collaboration of four areas of research: SoC solutions, active partial reconfiguration, Bayesian networks, and a signal processing core. This section presents a background on the last three of these technologies. The techniques and research behind SoC are well known. [14] presents a comprehensive study of SoCs if a detailed discussion is needed by the reader.

3.1. Active Partial Reconfiguration

Active partial reconfiguration (PR) is an enabling technology for autonomous SoPC. Active PR is Xilinx’s terminology which means that part of the FPGA’s configuration may be updated while the remainder of the FPGA configuration remains untouched and continues processing. In effect two or more hardware cores may time-share a portion of the reconfigurable fabric much like multiple software threads time-share on a general-purpose processor. The high level architecture of a PR design is straightforward. The static region is the portion of the configuration that does not change. There are also one or more PR regions that may be reconfigured with new designs. To reconfigure a PR region, the static region loads a partial bitstream into the Internal Configuration Access Port (ICAP). A partial bitstream exclusively contains all of the FPGA configuration data for the specified PR region.

Starting with the Virtex E Xilinx fabricated FPGA chips supporting active PR. A sophisticated PR approach was developed in JBits [15], however Xilinx has stopped officially supporting it in more recent architectures. Xilinx has more recently turned to supporting modular active PR as described in [16] and demonstrated in [17 and 18]. Unfortunately early implementations of this approach limited PR module designs to simple circuits due to constraints on orientation and routing. Recent advances in Xilinx’s tool flow, described in [19 and 20], the major limitations in the initial tool flow were lifted. PR regions no longer had to span the entire column and more importantly “pass-through routing” allowed static signals to be routed through PR regions without using bus macros. Using bus macros for signals that cross regions, but not pass through regions, are still required. A bus
macro is a special netlist that helps contain interfacing signals between static and PR regions. Of particular importance was the advance in pass-through routing. With pass-through routing, during the partial bitstream generation process, the PAR is aware of any static signals that were previously routed through the region and subsequently uses different routes.

Once the original limitations were lifted the next challenge became how to build an autonomous SoPC. EDK, Xilinx’s tool for developing systems on a chip, does not natively support PR systems. Only by performing a number of manual intervention steps can an EDK system be built as an active PR system. The manual steps include locking bus macros and clock resources in the constraint file; specifying the static and reconfigurable regions; and running ngdbuild, MAP, and PAR with specific PR command line flags. In [21] Xilinx provides a reference example of a tool flow for creating a PR system by combining EDK and PlanAhead. PlanAhead is Xilinx’s tool to facilitate partitioning a design amongst large developer teams, but also contains PR hooks that simplify the process of creating a PR System [22]. PlanAhead was used extensively in the building of the autonomous signal processing system.

It is important to note that there are other methods to implement partial bitstreams for active partial reconfiguration on Xilinx chips. This is evident by Steiner’s work [12, 13] that uses a runtime system to generate partial bitstreams. Unfortunately his work leverages intellectual property owned by Xilinx unavailable to the general public.

3.2. Cognitive Algorithms

In previous work [23] a variety of Artificial Intelligence algorithms were explored in the context of real-time embedded computing platforms. A general result of this program found that these algorithms could be implemented with success on modern embedded processors if the degrees of freedom, and thus the order of operations, were tailored with the computational platform in mind. Cognitive algorithms span a wide taxonomy of algorithms such as Probabilistic Relational Models, Logical Inference, Support Vector Machines, Genetic Algorithms, and SAT planning [24]. These algorithms are often mapped in the robotics community to human attributes such as perception, attention, memory, learning, reasoning, and problem-solving. Based on an analysis of the DSP application and the situational awareness we are trying to develop, it was determined that Logical Inference and Probabilistic Relational Models best suited our needs. More specifically, a Bayesian Network algorithm was selected as it can represent both of these algorithm classes and by adding dynamic, sequential, or relational information, can be enhanced to consider more sophisticated scenarios.

Bayes’s theorem provides a way to calculate probabilistic inference where direct conditional relationship exists. A Bayesian network enables us to represent a set of variables and their probability distribution. It is represented as a directed acyclic graph and the directed edges represent conditional probabilities. With Bayesian network we can express conditional probabilistic relations compactly and can perform probabilistic inference among the variables [25].

To aid in the development of the Bayesian Network (BN), Intel’s Probabilistic Network Library (PNL) [26] was ported to the PowerPC 405 embedded in the Xilinx chip. PNL is an open source library of routines that aid in cognitive modeling and data mining in order to facilitate BN development. It also carries routines that allow the BN to update its transition probabilities from the original training data to probabilities based on observed tendencies and to infer new states on the fly.

3.3. Signal Processing Core

The baseline pulse detection and analysis core is comprised of two principal components, a subband tuner and a parameter extraction unit, depicted in Figure 3. The subband tuner is a generic tuner as in [27] and is comprised of a demodulator and a filter bank. The filter bank is generically a set of N low pass filters, each with ½ the bandwidth of the previous filter, largely responsible for removing anti-aliasing. The filter bank implemented in our system was as large as could fit in the device and still leave room for the partial reconfiguration region, or N=6. A signal’s path through the filter bank is determined by performing a coarse grained calculation of the signal of interest’s (SOI) bandwidth and selecting the next largest filter bandwidth size in order to preserve the SOI. The backend parameter extraction unit then detects the location of a pulse in a stream of data using a rise time ratio algorithm as described in [28] to detect the leading and trailing edges of the pulse. Once a pulse is detected, the characteristics of the pulse are calculated, such as the width, magnitude, frequency, and arrival time. This signal processing core makes an excellent candidate to demonstrate the autonomous system as it was developed and implemented on an FPGA under a previous program, has been validated, and contains sufficient synthetic data to expand and use for autonomous testing.

4. Adaptable Signal Processing System Implementation

Figure 2 displays how we mapped the four technologies to create an autonomous signal processing system. The entire system is first and foremost implemented as a SoPC solution running Linux. The signal processing core described in Section 3.3 acts as the system processing component. The DSP core was modified to be an EDK core attached to the bus. The DSP core passes to the Bayesian network, implemented as a software process, parameters on each SOI. The Bayesian network uses the
SOI parameters to determine if the DSP core needs to be modified to mitigate signal interference. If interference is detected, the Bayesian network selects and pushes a partial bitstream, with an appropriate mitigation filter, to the ICAP port. Once reconfiguration is complete the DSP core can perform better given the current environment conditions.

4.1. SoPC Implementation
The adaptable signal processing system was implemented on a Xilinx ML410 development board and the implementation diagram of the resulting SoPC architecture is depicted in Figure 3. The adaptable signal processing system design was initially built using Xilinx’s Base System Builder. The system has the expected major components needed for a SoPC running Linux: a PowerPC 405 core, 256MB of memory, the tri-mode Ethernet media access control core (TEMAC), a PCI bus, and local bus provided through the instantiation of a processor local bus (PLB) and on-chip peripheral bus (PLB). The adaptable signal processing system runs at 200MHz for the PowerPC and 66MHz for the bus and remaining cores. A MicroBlaze processor could also be used in this architecture; however the PowerPC was selected to allow for future expansion in computational complexity of the BN.

Four cores were added to this baseline in order to implement the adaptable signal processing system. First, the DSP core described in section 3.3, modified to fit EDK core specifications with an OPB bus interface, was added. By connecting the DSP core to the bus, it allows the Bayesian network software process access to its data registers by using common read and write bus commands. The DSP core receives pulse data through a custom interface to a new “pulse data memory” core developed for the prototype system. The pulse data memory core may be viewed as software controlled cache. It is loaded with the raw pulse data for each pulse before the DSP core begins its analysis. The pulse data memory is 64KB of dual-ported embedded block RAM. One port of the BRAM is connected to the bus to allow the system to push pulse data to it. The second port is connected to the DSP core. With this configuration the DSP core can read data out of the pulse data memory with pipeline reads such that a new data sample is ready each clock cycles.

The third core is the reconfigurable filter core, which is a wrapper around the PR region that gets reconfigured. This is where the various interference mitigation filters are loaded during PR. It is the only custom core that does not have a bus interface. The reconfigurable filter communicates with the DSP core through a custom interface that is similar to a FIFO protocol. The signals between the cores are instantiated within EDK as a custom net. Each filter that is placed within the reconfigurable filter core must adhere to the custom interface and protocol. All signals between the DSP core and the reconfigurable filter core pass through bus macros that are instantiated within the DSP core. Five left to right asynchronous narrow bus macros and six right to left asynchronous narrow bus macros were instantiated for this design. Xilinx makes available both wide and narrow; and synchronous and asynchronous bus macros [19].
Asynchronous bus macros were used instead of synchronous bus macros to keep the clock cycle delay between the subband channel and the parameter extraction unit unchanged from the original design described in section 3.3. Narrow bus macros were used, instead of wide bus macros, for design simplification reasons. The final placement of the PR region can be seen in Figure 4, the bus macros are near the upper left corner of the PR region. The fourth core is a wrapper to the ICAP. Xilinx provides the OPB HWICAP core through their Early Access toolset [19]. With this configuration any other bus master, including the PowerPC, may write partial bitstreams into the ICAP from the bus, consequently providing a familiar abstraction in the form of bus operations for the ICAP.

Linux was our chosen operating system. This decision was made for both theoretical and practical reasons. On the theoretical side Williams points out in [29] that the Linux driver framework provides a natural abstraction for pushing partial bitstreams to the FPGA fabric. On the practical side Linux is first and foremost well known. Second Xilinx provides a reference system on a chip design for the ML410 [30] including a workflow for compiling Monta Vista Linux. This allowed our team to focus attention on the custom system design components rather than the OS. The disadvantage of Linux is that it is a general-purpose operating system with only minimal support for real time applications. Other operating system such as Hthreads [31] or ReconOS [32], although not rigorously deployed and tested, may be better suited for autonomous SoPC since they are designed specifically for real-time systems on reconfigurable hardware and have design elements meant to support active PR.

**4.2. The Reconfigurable Region for Mitigating Interference**

The autonomous signal processing system’s DSP core remains largely functionally identical to the core described in Section 3.3, with one key difference. In between the subband tuner and the parameter extraction unit, the reconfigurable filter core is instanced. Because of this additional logic was added to control data flow. In between the subband tuner and parameter extraction unit a FIFO and multiplexor are inserted to buffer and switch the data between passing directly through the subband tuner to the parameter extraction unit and going from the subband tuner to the reconfigurable region and then to the parameter extraction unit. In this manner pulses can be processed by the baseline DSP core, the results analyzed by the cognitive algorithm, and then if interference is detected, the buffered data can immediately be run after a reconfiguration on the mitigation filter as opposed to having to re-run the data over the entire DSP core.

For a proof-of-concept implementation, the interference mitigation modules will implement time-base FIR filtering, though as mentioned previously, the reconfigurable filter area can be used to implement any mitigation type that can fit in the chip area designated. For this prototype the autonomous system will handle three different interferer types: an increase in Gaussian noise, a wideband square pulse interferer, and a triangular narrowband interferer. An SOI with each of these interferers is depicted in Figure 5.

To mitigate these interference scenarios a set of FIR filters was designed to preserve the SOI while filtering out the unwanted interferer. It should be noted that without the use of active PR it would require too many FPGA resources (slices and multipliers) of too high a precision to create a single filter which statically mitigates all three scenarios. Also, the system will implement entirely new shaped and length filters to differentiate this approach from previous adaptive hardware implementations where the filter structure is static and new coefficients are uploaded. For our testing, the noise scenario mitigation filter is implemented as a 65 tap symmetric low-pass filter, the square pulse mitigation filter is implemented as a 49-tap band-pass filter, and the triangle pulse mitigation filter is a 49-tap notch filter. The frequency responses of each of these filters are depicted in Figure 6.

The control of data through the reconfigurable mitigation filter and iteration with the cognitive algorithm can be handled either iteratively or predicatively. In iterative mode, each pulse is individually analyzed and the reconfigurable scratchpad is set up uniquely for each pulse. The benefit of this approach is that it will better select the proper filter type and increase the quality of signal processing returns. However the penalty of this approach is that it will incur the full latency of the cognitive analysis and partial reconfiguration execution, approximately 112 ms. In predictive mode, the cognitive algorithm makes decisions about the environment over a series of pulses and the incoming pulse data is always filtered with the pre-configured mitigation filter, until the cognitive algorithm has determined that there has been a change in environment. The benefit of this approach is that the cognitive analysis and reconfiguration is largely...
decoupled from the process, however the penalty is that in some cases the environment may change quicker than the cognitive algorithm execution loop and some pulses may be processed by non-ideal mitigation filtering.

4.3. Bayesian Network Development

The key development task to implement the BN was to determine what heuristics to use from the parameter extraction algorithm to determine if the sensor was seeing the effects of Gaussian noise, the square pulse interferer, or the triangle pulse interferer. To rapidly experiment with generating different interference types, observing the baseline parameter extraction unit’s response, designing mitigation filters, and measuring the improved processing results, the team developed and utilized a bit-accurate Matlab model of the DSP core. In this environment not only could the parameter extraction unit’s outputs be observed, but internal variables within the algorithm could also be exposed to the BN for quick proto-typing and experimentation.

In this environment, a training set of 10 pulses and an experimental set of another 50 pulses were created for baseline and each interference type signal. Using the training set, heuristics for the square and triangular pulses were easily developed, as they are pulses and are identified by the parameter extraction unit. In these two cases the pulse width and frequency could be used readily.

Despite Gaussian noise being a simplistic concept, training the BN to detect a random event proved quite difficult. Eventually, a heuristic that proved successful was a noise estimate calculation internal to the parameter extraction unit. This calculation sums the signal magnitude over the first 16 samples of the pulse data snapshot. This assumes that there is no signal present in the first 16 pulses, which is sometime an erroneous assumption, but in practice proved to be reasonably reliable. Another observation in our experimentation was that the noise estimate, or threshold, was unique for each branch in the filter bank, as each filter involved has different bandwidths and stopband rejection characteristics. Figure shows how the noise threshold correlated for each branch in the filter bank. Although other internal parameters could have been selected to further boost the correctness of the BN, these three parameters were most sensitive when interference was applied and keep the degrees of freedom reasonable for an embedded real-time system.

The thresholds to use in the probability determination within the BN were found by histogramming the ranges of these parameters by against the ranges of the test pulses. Figure shows an example of this histogramming for the Gaussian noise scenario for path 2 through the filter bank against the probability that the baseline filtering will produce the correct result. Here it can be seen that as the noise threshold increases, the probability of correctly identifying the incoming signals decreases. This
information can then be used to select a desired probability of success operation region and then use the monitored noise threshold values to determine when to reconfigure to a more high performance mitigation filter. More accurate probabilistic knowledge can be obtained by increasing the fidelity of the discrete parameter ranges in the histogramming, but it will lead to an exponential increase in the number of operations to evaluate during runtime. Since the target platform is an embedded system without much memory, we limited the number of discrete ranges to three for each state. The ranges are determined manually. Depending on the type of interference, the characteristics of the pulses may change, and the ranges may need to be recalibrated again.

To then combine the determination between multiple situational awareness states a multi-node Bayesian network depicted in Figure 8 is utilized. Here, the type of interference is probabilistically inferred through the Bayesian network with the values of leading edge, pulse width, noise threshold, and frequency. For each interferer type, the probability of its presence in the input pulse is inferred probabilistically. For each possible state, the algorithm looks at the current heuristic value, if it is within a range determined to be characteristic of that state, a probability that state being true is assigned. The more heuristics that fall within range, the higher the probability. Once a probability has been assigned for all possible states, the highest probability is selected. This can also be expressed as a sum of conditional probabilities:

\[
P_{\text{state1}} = P(\text{state1} | le_{\text{min}} < le < le_{\text{max}}) + P(\text{state1} | \text{pwidth}_{\text{min}} < \text{pwidth} < \text{pwidth}_{\text{max}}) + P(\text{state1} | freq_{\text{min}} < freq < freq_{\text{max}}) + P(\text{state1} | nt_{\text{min}} < nt < nt_{\text{max}})
\]

Where \(le\) is the leading edge value, \(pwidth\) is the pulse width, \(freq\) is the frequency, and \(nt\) is the noise threshold. As can be seen, the current implementation of this algorithm is \(O(NM)\) where \(N\) is the number of states to evaluate and \(M\) is the number of heuristics. For this experiment these values are small, but for larger systems, a hierarchical BN may be used to reduce execution time. Since the outputs of the Bayesian network are probabilities, the correctness of the decision based on them is also probabilistic. In this implementation, once an environmental state is determined, the corresponding mitigation filter is implemented in the reconfigurable scratchpad. In more sophisticated scenarios, feedback on the success of previous iterations can be used to expand the BN to consider multiple mitigation strategies even within an assumed environmental state.

\[\text{Conditional Probability over Noise Threshold (Path 2 - Baseline Filter)}\]

\[\begin{array}{c}
\text{Path 1} \\
\text{Path 2} \\
\text{Path 3} \\
\text{Path 4} \\
\text{Path 5} \\
\text{Path 6}
\end{array}\]

\[\begin{array}{c}
P(\text{success} | \text{nontrivial}) \\
P(\text{error} | \text{nontrivial})
\end{array}\]

\[\text{Figure 8. Path 2 Histogram}\]

4.4. ICAP Drivers

The ICAP device drivers were implemented as a Linux character device with a few additional ioctl() commands. Device driver code specific for FPGA reconfiguration was adopted from Xilinx’s supplied OPB HWICAP core drivers [33]. The interface to Linux was adopted from Williams’ previous work [29]. Consequently, at the

```c
// open ICAP device
fd = open("/dev/icap", O_RDWR);

// send the size of the bitfile
ioctl(fd,XHWICAP_IOCCMD_RECONFIGURE_SIZE, lSize);

// send the bitfile to the kernel
// and reconfigure
ioctl(fd,XHWICAP_IOCCMD_RECONFIGURE,buf);
close(fd);
```

\[\text{Figure 10. Reconfiguration Code Example}\]
application programming level, partial reconfiguration is almost as easy as file I/O. Figure 10 is an example of the code used at the API level, with just 4 lines, to perform partial reconfiguration. This code is initialized with two variables the bitfile size (ISize) and bitfile buffer (buf). An unfortunate consequence of using Linux and implementing device drivers for the ICAP however is the overhead needed to copy the bitstream from user space to kernel space before pushing the file into the ICAP device. These details will be discussed in more detail in the results section.

5. Results

5.1. Implemented Active Partial Reconfiguration within EDK

Xilinx’s early access tool flow for generating EDK PR systems does work. Our adaptable signal processing system demonstrates this. Furthermore, by leveraging PlanAhead, the processing of an adaptable SoPC is much easier.

The workflow we used to generate the initial full bitstream, which includes the static and the first partial bitstream, is below. Portions of the workflow that required a nontrivial amount of processing time (greater than 1 minute) are marked with an asterisk (*).

1. Generate netlists for each EDK core. *
2. Import netlists into PlanAhead. *
3. Create static and reconfigurable areas.
4. Using PlanAhead, lock clock and bus macros.
5. Run DRC checks. *
6. Export the PlanAhead floor plan for PR flow. *
7. Manually copy clock and constraint files into correct directories of the exported floor plan.
8. Run budgeting. *
9. Static bitstream creation. *
10. Partial bitstream creation. *
11. Merge static and partial bitstreams. *
12. Copy full bitstream back into EDK.
13. Update bitstream with appropriate software application.

Completing the above workflow for the adaptable signal processing system took roughly 8 hours on a high end workstation (quad core 2.66GHz Xeon with 5GB RAM). Generating each additional partial bitstream took roughly 2 hours to complete (workflow not shown). The merged adaptable signal processing system (the static and the largest PR regions combined) required nearly 90% of the slice resources on the chip, with the PR region covering nearly 30%. The difficulty with this workflow is that if a change had to be made in the static portion of the design, the entire workflow had to be restarted.

Before implementing partial reconfiguration, the signal processing system was able to build, meet timing, and run at 300MHz and 100MHz for the processor and bus respectively. However, when implemented as a PR system the timing constraints had to be scaled back to 200MHz and 66MHz to meet timing in place and route. The cause of the timing errors was due to the size of the PR region. The first adaptable signal processing systems implemented had a relatively small PR region, roughly 10% of the chip, which met timing at 300MH and 100MHz. The final design though had a relatively large PR region consuming 30% of the chip, causing more routing resources to be shared between pass through static routes and the PR region, requiring initial timing goals to be scaled back.

Table 1 lists the number of FPGA resources needed to implement the adaptable signal processing system. “Static” represents the static region of the chip, and “Interferer 1,” “Interferer 2,” and “Noise” represents the specific filters implemented for the PR region. The V4FX60 has a total of 26,624 CLB slices. The PR region covers 7680 of these slices. Although this is roughly 50% more than what appears to be needed for the PR region, MAP was allocating most of the filter slices to SLICEM components, of which there are only 3840, due to the shift registers used in the filter implementation.

5.2. Signal Processing Results

To analyze the signal processing results we carried out a small scale experiment with synthetic data. A training set of 40 pulses (10 each of baseline, noise, square interferer, and triangular interferer) was used to develop the probabilities for the Bayesian network. The testing experiment then utilized 200 pulses not used in the training set (50 of each type). The system performs the baseline DSP processing and determines if the input pulse contains one of the known interference patterns. If so, it decides on the type of interference pattern based on the highest probability from the Bayesian network.

From the discussion in section 4, some of the interference mitigation filters are very similar to one another. As such, there is still some signal processing benefit to identifying that interference is present, but not selecting the optimal mitigation filter. This is common for the interferer 1 and noise cases, though depending on where in the pulse repetition cycle the interferer 2 snap shot is captured, there can also be a benefit in selecting the noise mitigation filter in that scenario as well. In order to determine the partial benefit realized, statistics were kept identifying the percentage of time a pulse was categorized as a different type. The partial mitigation category then refers to cases where the interference type was misclassified, but there was still at least a 3dB reduction
in the signal strength of the interferer. The >3dB category is then the sum of all cases for which the system realized at least a 3dB rejection of the interferer (correct classification + partial mitigation) and the >10dB category is the percentage of time there is a 10dB or greater reduction in the interferer. Since the interference mitigation filters preserve the SOI, there is no penalty for false alarms.

Overall, the algorithm correctly detects that there has been a change in the environment 96.7% of the time, yielding at least 3dB interference rejection 80.6% of the time, and yielding at least a 10dB interference rejection 64.7% of the time. Additionally, as noted in section 4.3, using more heuristics or subdividing the ranges of the heuristics to finer granularities can increase the accuracy of the Bayesian network.

### Table 2. Signal Processing Results

<table>
<thead>
<tr>
<th>Noise type</th>
<th>No noise</th>
<th>Interferer 1</th>
<th>Interferer 2</th>
<th>Gaussian Noise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct detection and mitigation</td>
<td>52%</td>
<td>86%</td>
<td>52%</td>
<td>56%</td>
</tr>
<tr>
<td>Incorrect detect as no interference</td>
<td>NA</td>
<td>0%</td>
<td>8%</td>
<td>0%</td>
</tr>
<tr>
<td>Incorrect detect as Interferer 1</td>
<td>8%</td>
<td>NA</td>
<td>6%</td>
<td>26%</td>
</tr>
<tr>
<td>Incorrect detect as Interferer 2</td>
<td>0%</td>
<td>4%</td>
<td>NA</td>
<td>18%</td>
</tr>
<tr>
<td>Incorrect detect as Noise</td>
<td>40%</td>
<td>10%</td>
<td>36%</td>
<td>NA</td>
</tr>
<tr>
<td>Partial Mitigation</td>
<td>NA</td>
<td>10%</td>
<td>12%</td>
<td>26%</td>
</tr>
<tr>
<td>&gt;3dB Mitigation</td>
<td>NA</td>
<td>96%</td>
<td>64%</td>
<td>82%</td>
</tr>
<tr>
<td>&gt;10dB Mitigation</td>
<td>NA</td>
<td>86%</td>
<td>52%</td>
<td>56%</td>
</tr>
</tbody>
</table>

5.3. Architecture Performance Results

A key aspect of the adaptable signal processing system is how fast it can execute the autonomous control loop. Namely, how quickly can it pull heuristics from the DSP core, execute the BN, pull a partial bitstream from off chip memory, copy the bitstream from user space to kernel space in Linux, send the data to the ICAP interface, and reconfigure the PR region. Table 3 below captures the performance of our system.

While for our application, any reaction time less than human analyst reaction times (3-5 sec) is sufficient, reducing this feedback loop time will make FPGAs competitive in a wide range of autonomous systems. As can be seen the worst offender is transferring the bitstream data to the ICAP port. This is due to our architecture implementation. In the baseline EDK system implemented, the PowerPC must pass data first across the PLB bus, then to a bridge connector, and then to the OPB bus running at 66 MHz to the ICAP core. This version of the PowerPC cannot do burst transactions so only one data word can be sent at a time. The experiments indicate that approximately 23 cycles per word are being achieved in this configuration, not including data cache misses. The second largest time contribution comes from passing data from user space to kernel space in Linux, incurring a data copy.

### Table 3. Autonomous Loop Execution Times

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristics transfer to PowerPC</td>
<td>7 ms</td>
</tr>
<tr>
<td>BN execution</td>
<td>~300 us</td>
</tr>
<tr>
<td>Off chip bitstream acquisition</td>
<td>10 ms</td>
</tr>
<tr>
<td>Linux memory space copy</td>
<td>16 ms</td>
</tr>
<tr>
<td>Data transfer to ICAP</td>
<td>70.7 ms</td>
</tr>
<tr>
<td>Reconfiguration time</td>
<td>8 ms</td>
</tr>
<tr>
<td>Total</td>
<td>112 ms</td>
</tr>
</tbody>
</table>

Although [20] reports that a partial bitstream for 25% of a V4LX60 running at 100MHz can be loaded in less than 1ms, in our final configuration it took significantly longer as we are looking at end-to-end system times. The extra overhead is both due to the user space to kernel space copy penalty, as well as the time needed to transfer the bitstream over the bus to the ICAP device. An important future work item is to minimize the overhead to load a partial bitstream. One idea is to modify the ICAP core to be a bus master allowing it to pull the bitstream file directly from memory and therefore limiting the interaction required by the PowerPC.

6. Conclusion

This paper presented our research on an adaptable signal processing system. The system leverages four technologies, SoPC, active PR, Bayesian networks, and signal processing, to create an autonomous system that effectively mitigates changing interference to extract pulse characteristics from radio frequency data. The system was built for, and tested on, a ML410 development board with a Xilinx V4FX60 chip running Linux. The prototype system identifies that interference has been introduced into the environment correctly 96% of the time, selects the correct mitigation filter 65% of the time and selects a filter which partially mitigates the interference 16% of the time. The FPGA can be reconfigured in 112ms to mitigate the interference from a library of partial bitstream files. When compared to full static implementations, the number of mitigation filters, implemented as partial bitstreams, is limited only by the amount of system memory, potentially storing hundreds of mitigation filters, whereas a static implementation on the V4FX60 could only instance 7 filters, limited by the number of FPGA resources. The PR system has virtually created a significantly larger chip. In terms of reaction times compared to human operators or human in the loop.
systems, it is estimated that a human analyst would require 3-5 seconds to properly analyze each signal after it is initially processed, determine if there is interference, and select the appropriate mitigation response. The similar analysis loop in this system requires only 112ms, for a 26-43x reaction time increase. This system also has the advantage over human operated systems in that in never gets tired, goes on vacation, or demands pay increases. This capability directly translates into an increase in previously undetectable information products, a turn-around time orders of magnitude quicker than full human-in-the-loop systems, and a reduction in mission size, weight, and power requirements.

7. References