Automatic Mapping of Real Time Radio Astronomy Signal Processing Pipelines onto Heterogeneous Clusters

by

Terry Esther Filiba

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John Wawrzynek, Co-chair
Daniel Werthimer, Co-chair
Professor Jan Rabaey
Assistant Professor Aaron Parsons

Fall 2013

Automatic Mapping of Real Time Radio Astronomy Signal Processing Pipelines onto Heterogeneous Clusters

Abstract

Automatic Mapping of Real Time Radio Astronomy Signal Processing Pipelines onto Heterogeneous Clusters

by

Terry Esther Filiba

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor John Wawrzynek, Co-chair

Daniel Werthimer, Co-chair

Traditional radio astronomy instrumentation relies on custom built designs, specialized for each science application. Traditional high performance computing (HPC) uses general purpose clusters and tools to parallelize the each algorithm across a cluster. In real time radio astronomy processing, a simple CPU/GPU cluster alone is insufficient to process the data. Instead, digitizing and initial processing of high bandwidth data received from a single antenna is often done in FPGA as it is infeasible to get the data into a single server.

Choosing which platform to use for different parts of an instrument is a growing challenge. With instrument specifications and platforms constantly changing as technology progresses, the design space for these instruments is unstable and often unpredictable. Furthermore, the astronomers designing these instruments may not be technology experts, and assessing tradeoffs between different computing architectures, such as FPGAs, GPUs, and ASICs and determining how to partition an instrument can prove difficult. In this work, I present a tool called Optimal Rearrangement of Cluster-based Astronomy Signal Processing, or ORCAS, that automatically determines how to optimally partition an instrument across different types of hardware for radio astronomy based on a high level description of the instrument and a set of benchmarks.

In ORCAS, each function in a high level instrument gets profiled on different architectures. The architectural mapping is then done by an optimization technique called integer linear programming (ILP). The ILP takes the function models as well as the cost model as input and uses them to determine what architecture is best for every function in the instrument. ORCAS judges optimality based on a cost function and generates an instrument design that minimizes the total monetary cost, power utilization, or another user-defined cost.

For my parents

Contents

List of Figures

List of Tables

Acknowledgments

This work would not have been possible without the help of the people around me. I am very grateful to my co-advisor Dan Werthimer for his encouragement and mentorship throughout this process. I appreciate all the time and patience he put in to help me get to this point. I am also grateful to my co-advisor Professor John Wawrzynek for his guidance which helped motivate me and helped me develop my own research ideas. I would like to thank the members of my qual and dissertation committees, Professor Aaron Parsons, Professor Jan Rabaey, and Professor Geoff Bower for their time and help shaping this work. I also would like to thank Professor Don Backer for his wisdom and support. Although he is no longer with us, he made a lasting impact on my research and I am very grateful for that.

I also need to thank the members of my research group and my lab. Thanks to the members of CASPER and the BWRC, especially Mark Wagner, Jonathon Kocz, Peter McMahon, Henry Chen, Matt Dexter, Dave MacMahon, and Andrew Siemion. I appreciate the time everyone took teaching me, helping me debug things, and discussing my research.

Finally, I would like to thank the people who looked after me and brightened my time at UC Berkeley. Thanks to my parents and my sister Michelle for always being around whenever I needed them. I thank my boyfriend Ben Schrager and his family who welcomed me into their home and treated me like family. Thanks to my friends Sam Ezell, Rhonda Adato, Leslie Nishiyama, Olivia Nolan, Nat Wharton, Orly Perlstein and Ari Rabkin who kept reminding me that I would eventually finish. And, I am very grateful for the care I received from my allergist, Dr. James Kong, which made it possible for me to complete this work.

Chapter 1

Introduction

Radio astronomers are trying to solve a very diverse set of problems. Asking questions such as "Are we alone," and "When were the first stars and galaxies formed?" and researching galactic structure and formation, the nature of gravitational wave background, the transient universe, black holes, and extrasolar planets.

Naturally, this curiosity leads to the development of larger and higher bandwidth telescopes creating a flood of data. Keeping up with the data requires constant development of new instrumentation

1.1 Motivation

Figure 1.1: The VLA and Arecibo Telescopes

The diversity of problems and telescopes create a number of parameters an engineer needs to worry about while designing an instrument. The instrument be designed based on the algorithm required to process the data, the bandwidth of the data, the number of channels, and the list goes on. Figure 1.1 shows two pictures of two very different telescopes, the Very Large Array, or VLA, in Socorro, New Mexico and the Arecibo Telescope.

Traditionally, observatories dealt with this by designing custom instruments that would run on one telescope and solve one problem. This custom approach was the only way to get the requisite processing power to analyze the radio signals, but it resulted in costly designs, because the boards, backplanes, chips, protocols, and software all needed to be designed from scratch. To make matters worse, this approach resulted in a very long design cycle, requiring 5-10 years of development before an instrument could be deployed at a telescope and by the time the instrument was released, the hardware would be out of date.

Due to their custom implementations, these instruments also lacked flexibility. Each instrument was designed specifically for a single purpose. A hardware upgrade or algorithm modification would require a complete redesign of the instrument, and another long design cycle.

While these older designs needed to trade off flexibility for performance, newer technology can offer both performance and flexibility. Programmable devices such as FPGAs, GPUs and even CPUs can provide enough processing power to keep up with the data from many new telescopes. These devices make it easy to reprogram existing hardware to support newer algorithms, and, since they are programmed using portable languages, provide a quick path to upgrade hardware without redesigning the entire instrument.

With a huge range of technology available, choosing what hardware to use to build an instrument is a challenge. New technology, optimizations, and designs are constantly being developed and with everything constantly changing it's difficult to know what is best.

1.2 Technological Challenges

When building a large instrument, there is a wide variety of technology to choose from. Unlike many applications, where the goal is to provide the fastest possible implementation, in real time radio astronomy instrumentation there is a performance target. Understanding the tradeoffs between different implementations is key to cost-effective design.

An instrument designed typically has four choices of hardware for their algorithm implementations. They can use CPUs, GPUs, FPGAs, and ASICs. Figure 1.2 shows each of these as a spectrum from general purpose to custom. The CPUs and, to a lesser extend GPUs, provide a very general purpose design experience, while FPGAs and ASICs require custom designs. This spectrum also can be used to generalize a number of other properties of the presented hardware. Platforms further right require higher design times and design complexity and provide less flexibility in the final design. And because the chips will be deployed in relatively low volume, the platforms on the right will also cost more money. The

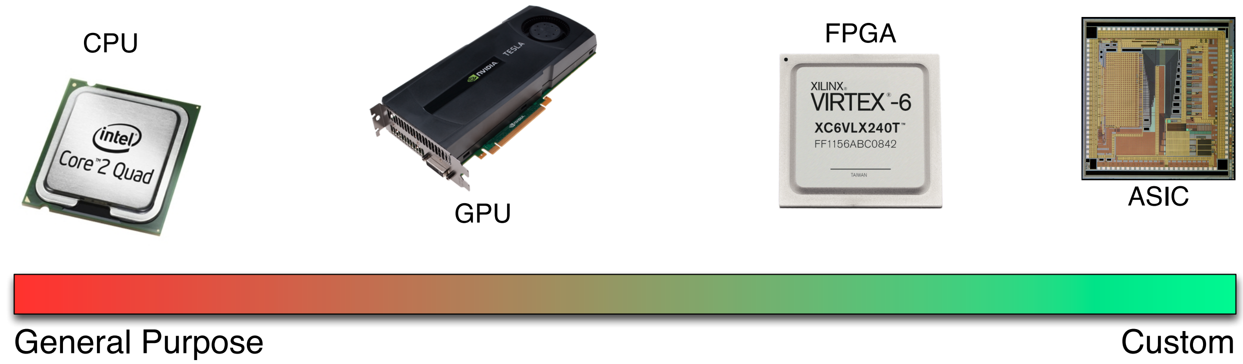


Figure 1.2: Technology Spectrum

tradeoff is in power and performance. The platforms on the right provide higher performance and lower power consumption.

To better understand these tradeoffs, consider the two platforms in the center of the spectrum, the GPU and FPGA. NVIDIA GPUs can be programmed in CUDA. Its C-like structure makes it easy for CPU programmers to pick up and provides a lot of flexibility. CUDA allows the programmer to use conditional and iterative programming the same way they would use C, easing the transition into GPU programming. FPGAs are programmed using a hardware description language, or HDL, which is less flexible and harder to learn than CUDA. In the FPGA computing model the data is streaming and everything is happening at the same time, requiring specialized programming. On the other hand, FPGAs offer an order of magnitude improvement in performance and power consumption. Each GPU requires hundreds of watts of power while an FPGA can be powered with tens of watts and a GPU can only process hundreds of megahertz of antenna data while an FPGA is capable of processing multiple gigahertz.

Although it may be easy to enumerate the differences between these platforms, understanding which is best is more difficult. In most cases, a heterogeneous approach is best. The best mix of platforms will ultimately depend on the desired instrument, available technology, as well as the price and power restrictions. This means that a brand new implementation must be designed for every new instrument.

Traditionally, designing an instrument can be a very long process. The astronomer will determine what type of instrument they want to build and what the specifications should be. The specification is handed off to a computer expert who will determine what platforms need to be used. The computer expert evaluates potential platforms. At best this may take a few hours if the computer expert is familiar with similar designs, but when working with newer technology or larger instruments this can take at least a week, possibly even months, giving the engineer time determine how to make the design fit. Any changes in the design will force the entire design process to iterate again, as was my experience developing part of a pulsar processor. To make matters worse, this process does not guarantee an optimal result. The engineer may have a bias towards the technology she or he is familiar with. Benchmarks for different technologies are very difficult to compare, as they represent different things, so the engineer might decide to simply check if it works on the preferred technology without testing alternate designs.

1.3 Optimal Rearrangement of Cluster-based Astronomy Signal Processing

This dissertation explores an automated approach to instrument design using a tool called Optimal Rearrangement of Cluster-based Astronomy Signal Processing or ORCAS. The major contributions of this work are a toolflow that allows radio astronomy experts to design cost-optimal high performance instruments without the aid of a computer expert, the ability to quickly explore different designs in the algorithmic design space, and support for disparate benchmarks to find an optimal instrument design.

This tool provides cost-optimal mappings in a short amount of time. ORCAS allows the developer to define the instrument at a high level and a cost function defining some cost the developer is aiming to minimize. The cost function can represent something as simple as the price of the instrument or can be used to represent more complex parameters of the instrument such as design time.

ORCAS assesses the performance of different types of technologies in two ways. First, by using benchmarks of existing implementations of instrument building blocks such as FFTs and FIR filters we can directly assess the performance. Second, if a benchmark is not available, the tool can use a performance model instead, giving an estimate of how the block will perform. This makes it easy to keep up with improving technology and library performance without rewriting the entire tool every time a new library version or board is released.

We evaluate ORCAS by benchmarking existing libraries, such as CUFFT and the CASPER library, and use those benchmarks to produce mappings for 3 types of instruments. Each mapping is compared to existing implementations to understand how this automated approach compares to the old approach of hand optimized mapping.

The remainder of the dissertation is organized as follows. Chapter 2 provides a more in-depth look at the algorithms commonly used in radio astronomy and their applications. Chapter 3 describes related work, done by myself and others, that preceded this work. Chapter 4 presents a high level description of the tool and explains how the tool goes from a description of the instrument to a fully mapped algorithm. The algorithm used to map the instrument is fully specified in chapter 5. Chapter 6 describes three instrument case studies and compares them to existing instruments. And finally, in Chapter 7, I present my conclusions and some opportunities for future work.

Chapter 2

Real Time Radio Astronomy Algorithms

Radio astronomy simply refers to the type of science that can be done by observing astronomical objects at radio wavelengths, rather than a specific scientific goal. There is a huge variety of different experiments, such as studying the formation and structure of galaxies stars and black holes and searching for gravity waves, traces of the first stars [19], or aliens [28]. But, despite this variety, the small number of algorithms detailed in this chapter serve as the first step in processing the data for many such projects.

Radio telescopes produce very high amounts of data. The reason for this high influx of data is twofold. First, to enable new science, new radio antennas observe increasingly higher bandwidths. Second, to sate the need for larger collecting area, rather than designing a single large dish, many new telescopes are being designed as antenna arrays, where the data from multiple antennas is combined to act as a single large dish. While it may be cheaper and easier to get a larger collecting area using small dishes rather than a single large dish, this adds additional complexity processing the data coming off the telescope. Rather than processing a single stream of data, now the instrument must process and combine multiple streams to make the array seem like a single large dish. As the size of the arrays and bandwidths for single dishes simultaneously increase, the data produced cannot be feasibly recorded in real-time. To cope with the progress in science and antenna technology, there is a constant need for new systems to process, rather than record, this data in real time. Each of these instruments begin processing the data immediately after it is digitized, and need to reduce the data without losing scientific information. Once the data is partially processed, and reduced to a manageable bandwidth, it can be stored and processed further offline.

There are a small number of real time algorithms commonly used to reduce the data. In this work, we specifically focus on spectroscopy, pulsar processing, beamforming and correlation. Spectroscopy and pulsar processing are both spectral methods of analyzing data from a single beam. They both can be used on antenna arrays but would either need to treat the array as separate dishes rather than one large dish, or the data from the dishes would need to be combined into a single beam, which can be done using the third algorithm on the list, beamforming. The last two techniques, beamforming and correlation, are both ways of combining data from multiple antennas. Beamforming combines the data into a single beam by delaying and summing the data from each antenna. Correlation doesn't aim to form a single beam, but instead is the first step towards getting an image of the sky.

2.1 Spectroscopy

A spectrometer is simply an instrument that produces an integrated, or averaged, frequency domain spectrum from a time domain signal. A real time spectrometer works by constantly computing a spectrum over short windows of data (channelization), then each channel is summed for a predetermined amount of time to compute the average power in that channel (accumulation). After digitization, there is the channelization step, where the signal is processed by a digital filter bank, and then the channels are accumulated.

High resolution spectroscopy

Increasing resolution often requires an increase in complexity in the spectrometer design. Once the number of required channels is sufficiently high, it becomes infeasible to compute the spectrum using a single filter bank. To cope with this, the channelization is done in two steps. In the first step, the signal is divided into coarse channels using a filter bank. At this point the channels are much wider than intended and can't be accumulated yet. After coarse channelization, the spectrometer treats the data from a single channel as time domain data and passes it through a filter bank again. This step breaks up the wide channel into a number of smaller channels. At this point the data can be accumulated, since it has the desired resolution.

Applications

This high resolution spectroscopy technique is used in the SERENDIP V.v (Search for Extraterrestrial Radio Emissions from Nearby Developed Intelligent Populations) project. This project is part of the SETI (Search for Extraterrestrial Intelligence) effort to detect extraterrestrial intelligence. The SERENDIP V.v project is a commensal survey at the Arecibo observatory, meaning anytime the ALFA receiver is used for any observation, the SERENDIP spectrometer will also process and record data.

The project is focused on detecting strong narrow band radio signals, requiring a high resolution spectrometer installed at the observing telescope to analyze the data. The spectrometer should resolve channels of less than 2 Hz, so that natural astronomical signals that typically have a wider bandwidth can easily be distinguished from narrower, possibly extraterrestrial, signals. The SERENDIP V.v spectrometer meets this by providing 128 million channels across 200 MHz of bandwidth, for an resolution of 1.5 Hz per channel. Since it would be infeasible to channelize a 200 MHz signal into 128 million channels using a single filter bank, the SERENDIP V.v spectrometer uses the high resolution architecture described in Chapter 3.

2.2 Pulsar Processing

A pulsar processor is an instrument designed specifically to observe transient events, such as pulsars. A pulsar is a rotating neutron star that emits an electromagnetic beam. When the beam sweeps past Earth, due to the rotation of the star, it is observed as a pulse of wideband noise. As the pulse travels through the interstellar medium (the matter filling interstellar space), the signal gets dispersed, meaning the low frequencies arrive before high frequencies, despite the fact that they were emitted at the same time.

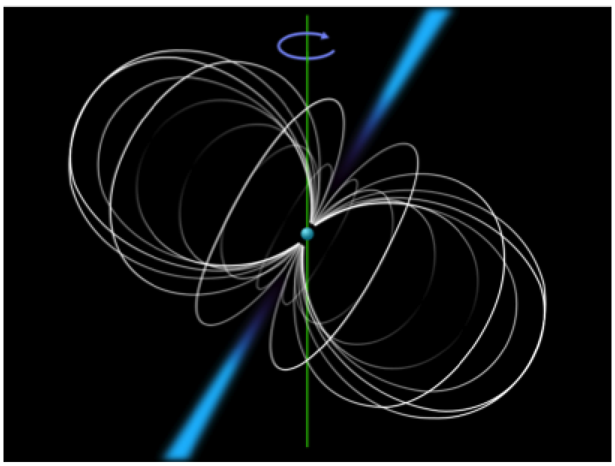


Figure 2.1: A Conceptual Pulsar Diagram

A spectrometer, as described in Section 2.1, that accumulates the spectrum would smear the pulse, so there will need to be a few adjustments to the spectroscopy algorithm to make it suitable for processing transient events. In the case of pulsars, the algorithm starts with a high-resolution spectrometer without an accumulator. Instead, the algorithm becomes specialized to detect this type of quickly occurring event.

The high resolution data is then sent to a process called dedispersion, which undoes the dispersion caused by the ISM, realigning the pulse. There are 2 techniques to do this. First, the pulse can be dedispersed by shifting the frequency channels by different amounts to compensate for the different delays, in a process called *incoherent dedispersion*. This process can't be used to reconstruct the original pulse, but due to its relatively low compute cost, is a useful algorithm to search for new pulsars. The second technique, *coherent dedispersion*, models the effect of the ISM as a convolving filter. To remove this effect, the signal is deconvolved with the model. This is more compute intensive than incoherent dedispersion, but can recover the original pulse.

After dedispersion, there is a still a lot of data and the pulse has very low SNR. The next step in processing is *folding*, or adding together many pulses, reducing the amount of data and improving the SNR. At this point, the data has been significantly reduced and can be recorded.

Applications

Some pulsars serve as very accurate astronomical time keepers. There are millisecond pulsars with extremely stable periods, allowing any perturbations in the observed pulse period to be attributed to some external effect. This makes pulsars extremely useful for conducting relativity experiments. One such example is the North American Nanohertz Observatory for Gravitational Waves, or NANOGrav, Project, which uses pulsars in an attempt to make the first detection of gravitational waves [6]. The project will try to detect gravitational waves by observing an array of pulsars, measuring the effect of the waves passing between Earth and the pulsars as changes in observed pulse arrival times.

2.3 Beamforming

Beamforming is a technique for combining data from an array of antennas. The beamformer combines the data from multiple antennas into a single beam pointed at a single point in the sky. This is achieved by delaying the signal from each antenna by a different amount and then summing the delayed signals. The signal from the intended source will not arrive at all the antennas at the same time. The delay compensates for the disparity between the arrival times, and once the signals are summed it creates constructive interference in the direction of the source, and destructive interference in other directions. These delays can be changed to point the beam at a different source or to track a single source moving across the sky.

BF Beamforming

Since the digitized signal from a radio telescopes receiver is discrete, and the amount of delay might not be an integer multiple of the sample period s, the delay d is applied to the signal, $f[n]$, in 2 steps. The delay in clock cycles is represented as d/s and broken up into it's integer and fractional parts. First, the integer part is applied as a coarse delay, $n_f = \lfloor d/s \rfloor$. This can simply be implemented by buffering the signal. Applying the fractional delay $d \bmod s$ is more complex. Since there is no observed data at that time, the signal fractional samples are calculated by convolving the signal with an interpolation filter b. Applying these 2 steps, results in a new delayed signal $f[n] = f[n - n_f] * b_{fi}$

In practice, it is common to calculate the spectrum of the beamformed signal. A typical 2 element beamformer, with discrete input signals $f[n]$ and $g[n]$ is trying to calculate the spectrum of beam i, H_i, using coarse delays n_{fi} n_{gi} and delay filters b_{fi} and b_{gi}, as follows:

$$H_i(x) = FT(f[n - n_{fi}] * b_{fi} + g[n - n_{gi}] * b_{gi})$$

We describe this as BF beamforming because the delay operation (B), happens before the FT operation (F).

FB Beamforming

In the case where many beams are required, each beam needs a different FIR filter for every antenna, and a separate Fourier transform. This is called FB beamforming because the fourier transform operations (F) occur before the delay (B). Using linearity of the Fourier transform and the convolution theorem, the computation can be rearranged as follows:

$$H_i[x] = F[x] \cdot B'_{fi} + G[x] \cdot B'_{gi}$$

Where $F[x]$ and $G[x]$ are the fourier transforms of the signals f and g. The signal delay becomes a phase shift in frequency domain that is represented by B'_{fi} and B'_{gi}. In this case, there is a FT for each antenna, rather than one FT per beam. When forming multiple beams, there is no need to recalculate the Fourier transform of the signals. So, as the number of beams increases, it can be advantageous to use this algorithm rather than the BF algorithm described in the previous section.

2.4 Correlation

Aperture synthesis is another technique to combine the data from many antennas. The goal is to form an image of the sky, using 2 steps, correlation followed by imaging. In the correlation step, the cross-correlation of each pair of antennas is calculated. Once the cross-correlation is calculated, it is possible to accumulate the data, greatly reducing the amount of data that needs to be processed in the imaging step.

The uv plane represents the Fourier transform of the 2-dimensional sky image. The cross-correlation of an antenna pair represents a point in the uv plane, called a *visibility*. Since the visibilities are not evenly or continuously sampled, the imager must interpolate points on the uv plane in an evenly spaced grid so the FFT algorithm, which relies on even spacings, can be applied to the data. The imager must also account for the fact that the response of the telescope distorts the image, undoing the effect using iterative algorithms like CLEAN or Maximum Entropy. Once this is done, a two dimensional inverse Fourier transform can be applied to recover the sky image. The book Interferometery and Synthesis Imaging by Thomson, Moran and Swenson [26] gives a detailed description of how the end to end synthesis imaging process works, but in this work we will only focus on the real time computation, the cross-correlation.

Typically, large telescope arrays do correlation in real time and finish the imaging offline, but there are a few notable exceptions. One of these is the VLBA, or Very Long Baseline Array. When designing antenna arrays, one important parameter is the distance between the antennas, or *baselines*. Arrays with longer baselines provide images with better angular resolution. The VLBA is an extremely high resolution array, achieving this resolution by using telescopes on opposite ends of the Earth. The distance between the antennas, while useful for science, creates a logistical issue for any real time processing that depends on data from different antennas. Instead of correlating in real time, the VLBA records the digitized data directly from the telescopes at each site, without any reduction. Later, the recorded data is flown to a central location, where it gets correlated.

The cross-correlation of two signals, f[n] and g[n] is defined as:

$$h[n] = f[n] \star g[n] = \sum_{i=0}^{m} f^*[i]g[n+i]$$

Like in beamforming, it is often desirable to calculate a spectrum, so the correlation step typically also calculates the spectrum of the cross-correlations. So, the final result of the correlator produces the spectrum of the visibilities, H[x].

$$H[x] = FT(f[n] \star g[n])$$

XF Correlation

An XF, or lag, correlator calculates the spectrum by calculating the cross-correlation first (X), followed by a Fourier transform (F). For a telescope with n antennas, this algorithm requires $O(n^2)$ cross-correlations, followed by $O(n^2)$ Fourier transform operations.

FX Correlation

The calculation of the cross-correlation is very similar to a convolution, in fact the cross-correlation of 2 signals can be re-expressed as a convolution. The cross-correlation theorem further extends the parallel between correlation and convolution, relating the Fourier transform of the cross-correlation of two signals to the Fourier transforms of the original signals. This allows us to take our original expression for the cross correlation of two antennas and express it as:

$$H_i[x] = F^*[x] \cdot G[x]$$

This style of correlator design is advantageous for a number of reasons, including the reduction in algorithmic complexity [2]. Rather than $O(n^2)$ Fourier transform operations, an FX Correlator only needs to compute one Fourier transform for each antenna, reducing the complexity to $O(n)$. As the number of antennas gets large, this makes a significant difference in the total amount of computational power required.

Applications

Correlators are being used to detect the formation of the first stars. The LEDA, PAPER, and HERA [11] projects are all developing large correlators, correlating hundreds of antennas, to get the sensitivity necessary to make a detection.

The Very Long Baseline Interferometery or VLBI technique uses an array of antennas that are extremely far apart to achieve very high angular resolution images. The Very Long Baseline Array, or VLBA, is a group of telescopes that are used at the same time to do VLBI. Figure 2.2 shows the locations of the VLBA telescopes.

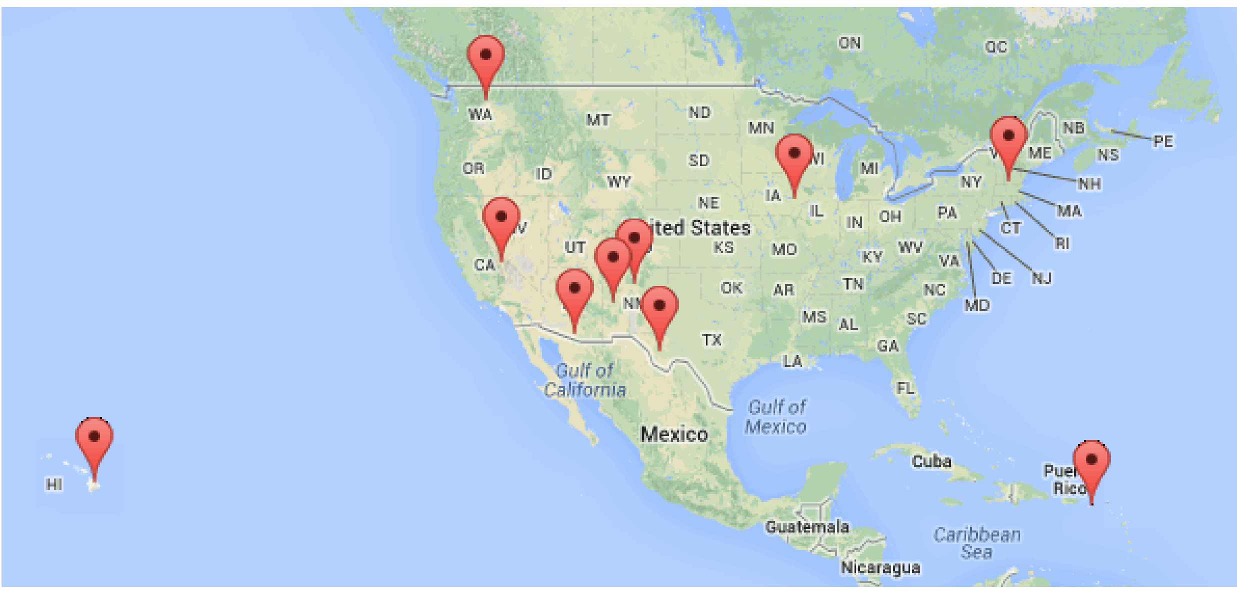


Figure 2.2: Telescope Locations in the Very Long Baseline Array

Chapter 3

Related Work

In this chapter we discuss work related to this dissertation in the fields of radio astronomy and electrical engineering. The radio astronomy work presented in this chapter share a common theme of flexibility and paramaterizability, demonstrating how common design patterns in astronomy are already being used to automate instrument implementation. The work in electrical engineering focuses on design generation, explaining what type of heterogeneous design was possible before this work, and why the past work is not sufficient to solve the radio astronomy instrumentation design problem.

3.1 Radio Astronomy

The need for high bandwidth processing manifests in many different radio astronomy applications. Keeping up with increasing computation demands has often resulted in the specialized design of instruments.

Distributed FX Correlator (DiFX)

The DiFX Correlator is a scalable software implementation of an FX Correlator [5]. DiFX was designed as a software correlator that targets CPUs in order to maintain flexibility in the design. The DiFX correlator was originally developed to do VLBI (very long baseline interferometery). The correlator is used by recording data at each VLBI site, collecting it, and playing back the recorded data through the correlator.

While this instrument does not run in real time, it is still an important instrument because it represents a parameterized software correlator design. Unfortunately, the flexibility of the software implementation comes at a high cost. Nearly 100 nodes are required to cross correlate 64 MHz of bandwidth from 10 antennas. While this design might be reasonable when a cluster is readily available, the software implementation is very costly to build from scratch.

xGPU

xGPU is a CUDA package that implements the cross-correlation operation of an FX Correlator on NVidia GPUs [3]. The CUDA kernel is expertly optimized, providing very high throughput and allowing it scale and support very large arrays. Figure 3.1 shows the performance of the kernel versus array size, scaling the array size all the way to 512 antennas. Although the kernel itself is difficult to modify without GPU expertise, the implementation of the package in CUDA ensures that users without optimization experience are able to use the tool to implement large correlators.

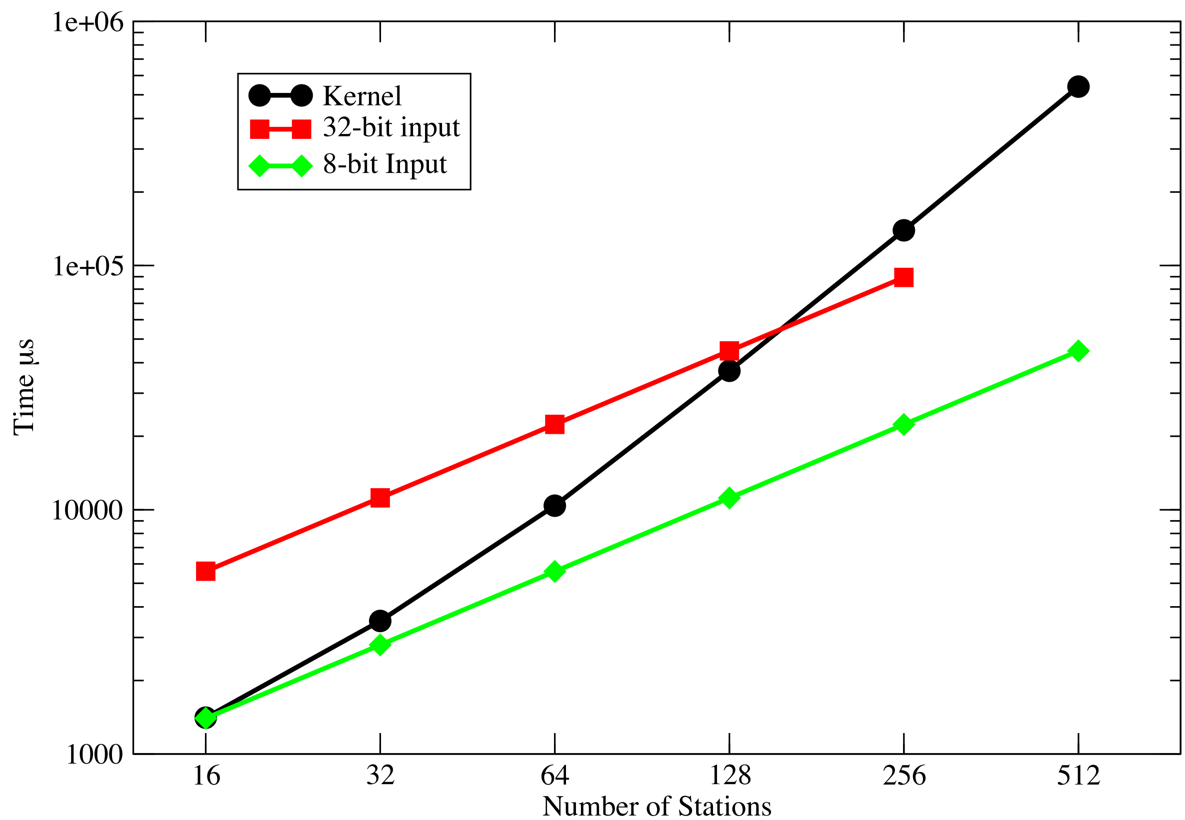


Figure 3.1: xGPU Performance as a function of array size (reprinted from Clark, La Plante, and Greenhill [3])

CASPER

At the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER), we are developing common libraries and hardware that can be used by radio astronomers developing instrumentation. The mission statement, available on the group website `casper.berkeley.edu`, concisely summarizes the goals of the group:

> The primary goal of CASPER is to streamline and simplify the design flow of radio astronomy instrumentation by promoting design reuse through the development of platform-independent, open-source hardware and software. Our aim is to couple the real-time streaming performance of application-specific hardware with the design simplicity of general-purpose software. By providing parameterized, platform-independent "gateware" libraries that run on reconfigurable, modular hardware building blocks, we abstract away low-level implementation details and allow astronomers to rapidly design and deploy new instruments.

The collaboration, started at UC Berkeley by Dan Werthimer, Don Backer, and Mel Wright, has spread to include astronomers all over the world, as shown in Figure 3.2. The group collaborates with many large digital engineering groups including groups from the Jet Propulsion Laboratory (JPL), the National Radio Astronomy Observatory (NRAO), the Arecibo Observatory, SKA South Africa, and the Giant Metrewave Radio Telescope (GMRT) in India. The diversity and far reach in this collaboration ensure that the tools continue to be general purpose and widely accessible.

Figure 3.2: Map of CASPER Collaborators

The CASPER FPGA libraries were developed to mitigate the need to redevelop common signal processing blocks for every new instrument [18]. The library provides parameterized blocks such as FFTs, digital down-converters, and FIR filters that are the necessary building

ADC Name	Max sample rate	Streams	Bitwidth
64ADCx64-12	50 Msps	64	12 bits
ADC4x250-8 or QuadADC	250 Msps	4	8 bits
KatADC	1.5 Gsps	1	8 bits
ADC1X2200-10	2.2 Gsps	1	10 bits
ADC1x10000-4	10 Gsps	1	4 bits

Table 3.1: CASPER ADC Boards

blocks for a wide range of instruments. Coupled with open source FPGA boards, such as the ROACH (Reconfigurable Open Architecture Computing Hardware), the CASPER libraries provide a useful toolbox for radio astronomy instrumentation development. The follow sections describe each of these and how they have been used to create a number of different instruments.

CASPER Hardware

The CASPER group provides set of modular FPGA boards and ADCs that are designed specifically to deal with the high bandwidth requirements of real time radio astronomy signal processing. Since each FPGA boards meets the needs of the radio astronomy community as a whole rather than a single application, the group releases a small number of boards and typically only releases a new board to take advantage of improving technology. By releasing a handful of boards, the CASPER group ensures that users get proven and tested hardware for their new instruments. The CASPER library and software, discussed below, also make it easy to upgrade the hardware since a CASPER design easily be recompiled to work with a different board and the software interface and the signal processing model are standardized. And, since the boards communicate over a common set of industry standard protocols, they can be upgraded one by one, or all at once.

Each FPGA board implements a number of high speed interfaces to send and receive data. The Z-DOK+ connectors are primarily used to interface the boards to high speed ADCs and DACs. This common Z-DOK+ interconnect implemented on nearly all CASPER ADC boards allows the astronomer to choose an ADC to match their scientific goals. The diversity of available boards is shown in Table 3.1.

Each board can also send or receive data using ethernet. The boards are designed to communicate using common protocols, like 10 Gigabit Ethernet, so a board can be upgraded without modifying how it communicates with the rest of the cluster. The use of an industry standard protocol also makes communication to non-CASPER boards simple, allowing an FPGA to create UDP packets that eventually get received by a CPU or GPU-based server, making the CASPER boards an ideal component in a heterogeneous cluster. This makes it simple to design a cluster, and allows continuous upgrades as technology improves, as the signal processing model and communication model do not change between boards.

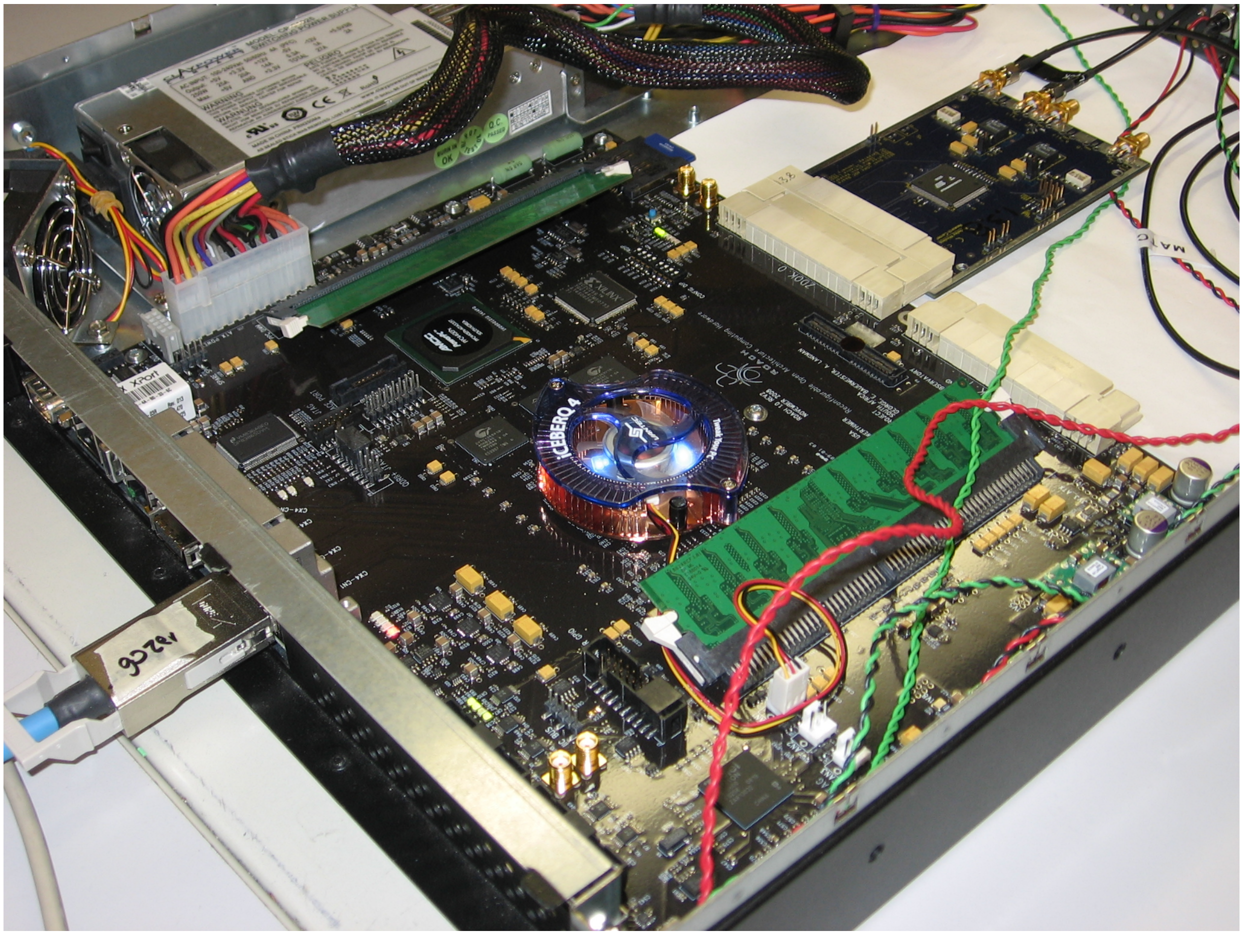


Figure 3.3: ROACH Board

Currently, CASPER has designed the IBOB, or Interconnect Break-out Board, ROACH and ROACH 2 boards. The IBOB was originally designed as an interface to digitize antenna data and send it to a server, but the powerful Virtex-II Pro make it very useful for channelizing data as well. The ROACH board is shown in Figure 3.3 with an iADC board connected via Z-DOK+ and an ethernet cable to get data off the board, This board is pin-compatible with two large FPGA chips, the Virtex 5 SX95T and LX110T, with the SX95T being preferred for its large number of DSP blocks. The ROACH has two Z-DOK+ connectors and supports up to four 10-gigabit Ethernet links. The ROACH 2 is an upgrade to the ROACH board that takes advantage of the newer Xilinx chips. The Virtex 5 chips have been upgraded to the Virtex 6 SX475T, and this board supports up to eight 10-gigabit Ethernet links.

CASPER Library

In addition to hardware, CASPER develops a DSP library that can be compiled into an FPGA bitstream. The library is implemented in Simulink, which allows for both simulation and, using Xilinx System Generator, compilation to FPGA code. The Simulink implementation is a key part of the success of this project. Using a visual programming language makes it easy to snap together blocks without getting into the low level details of FPGA programming.

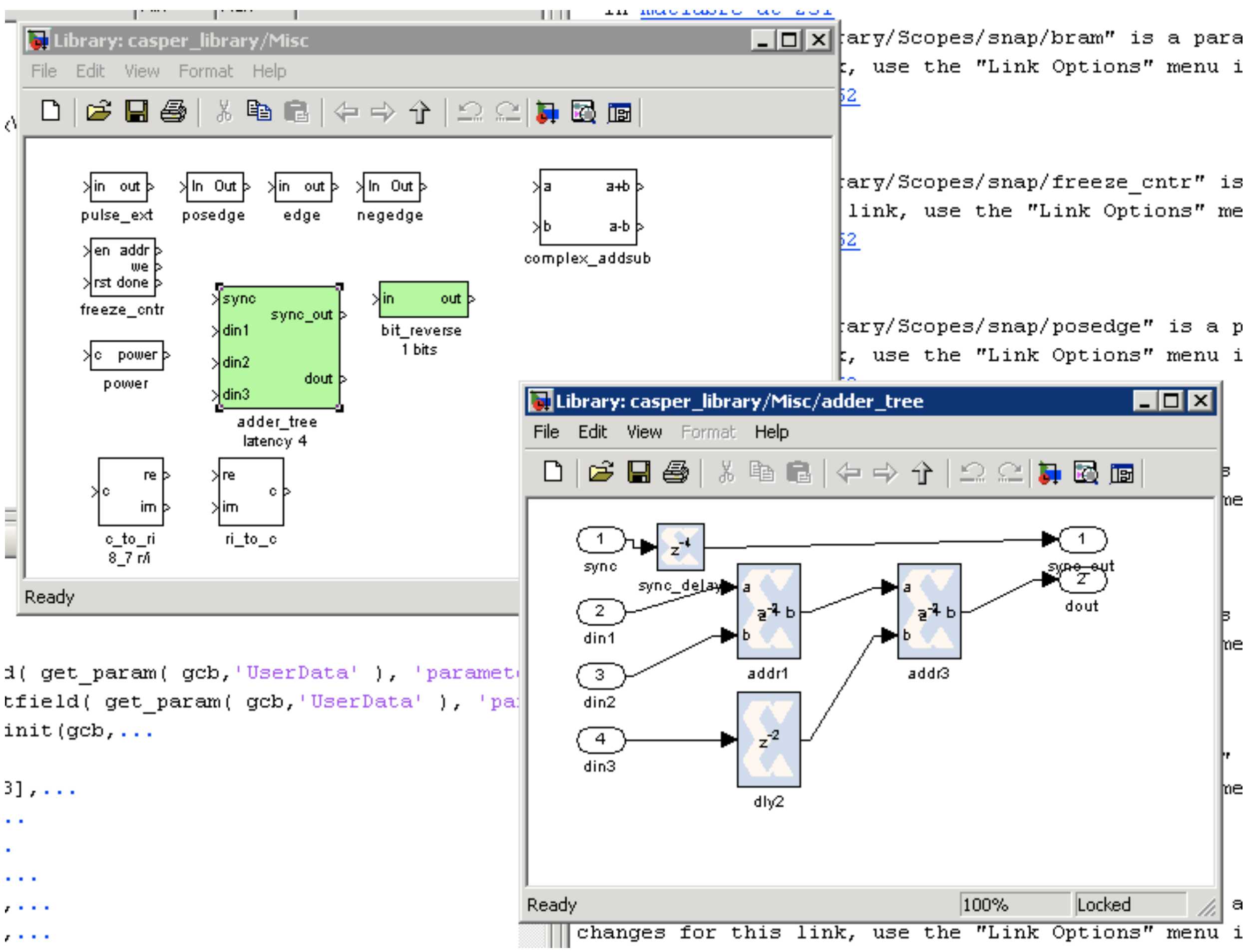


Figure 3.4: Adder Tree Simulink Simulink Diagram

The library blocks are built out of Xilinx System Generator primitive blocks. These primitives make it possible to retarget Simulink designs to different Xilinx FPGAs without changed the original implementation. Figure 3.4 shows a simple example of one of the library blocks, an adder tree. The adder tree is used to implement pipeplined, parallel addition using a binary tree. The top left is the library with the adder tree block, and the bottom right is the implementation of the adder tree block with three inputs.

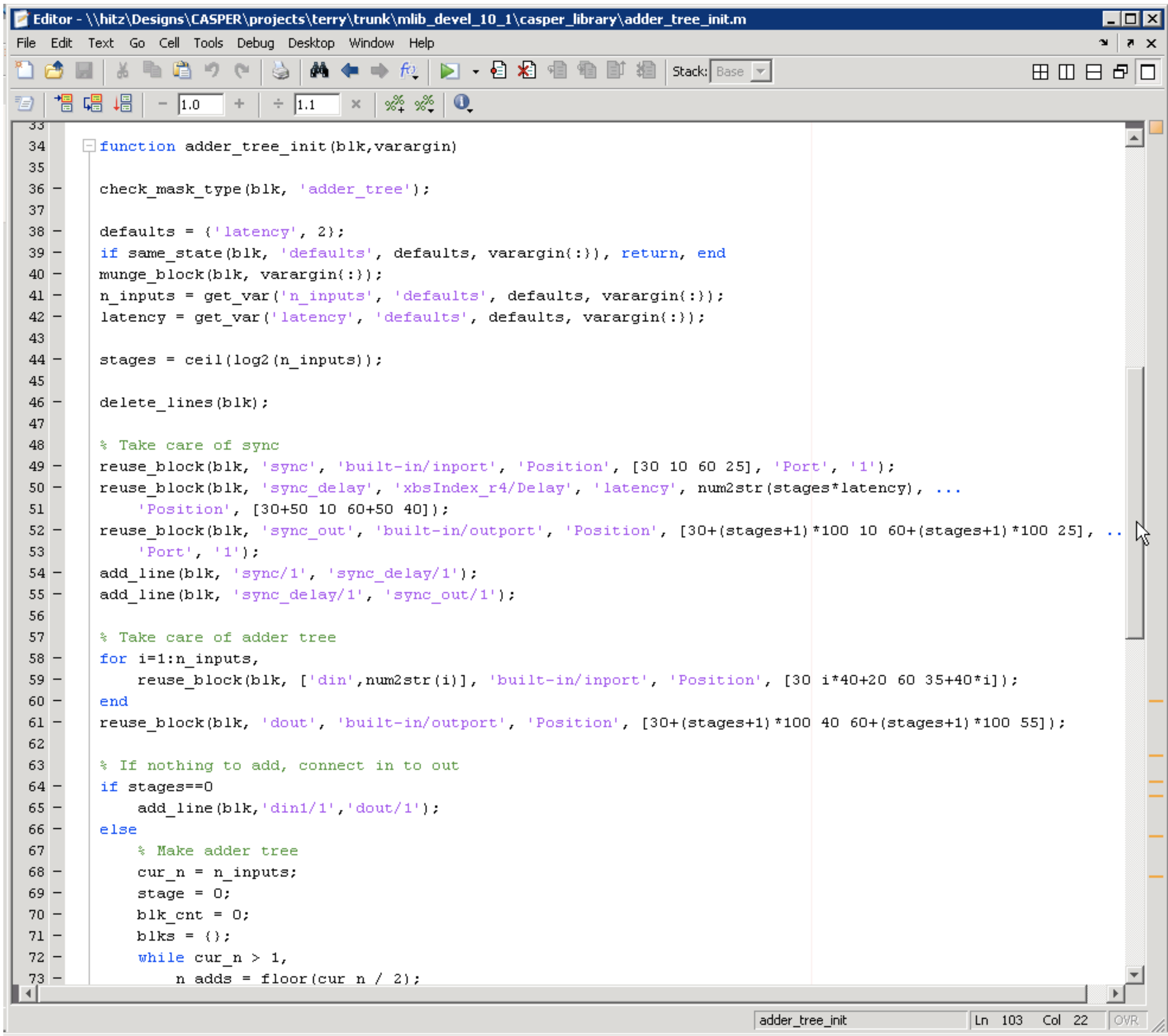


Figure 3.5: Adder Tree Simulink Mask Script

In order to configure the number of inputs, Matlab scripts are used to redraw the implementations. Figure 3.5 shows a snippet of code from the script used to redraw the adder tree block. Every parameterized CASPER block uses at least one of these scripts to redraw the underlying implementation when the parameters are modified.

The CASPER library includes DSP blocks commonly used in radio astronomy instruments. For example, the CASPER library provides FFTs, FIR filters, accumulators, digital downconverters, digital mixers which can be linked together to make an instrument. Each block is parameterized, making them useful for a variety of instruments. Figure 3.6 shows the FFT library, which contains different types of FFTs, including blocks that can process

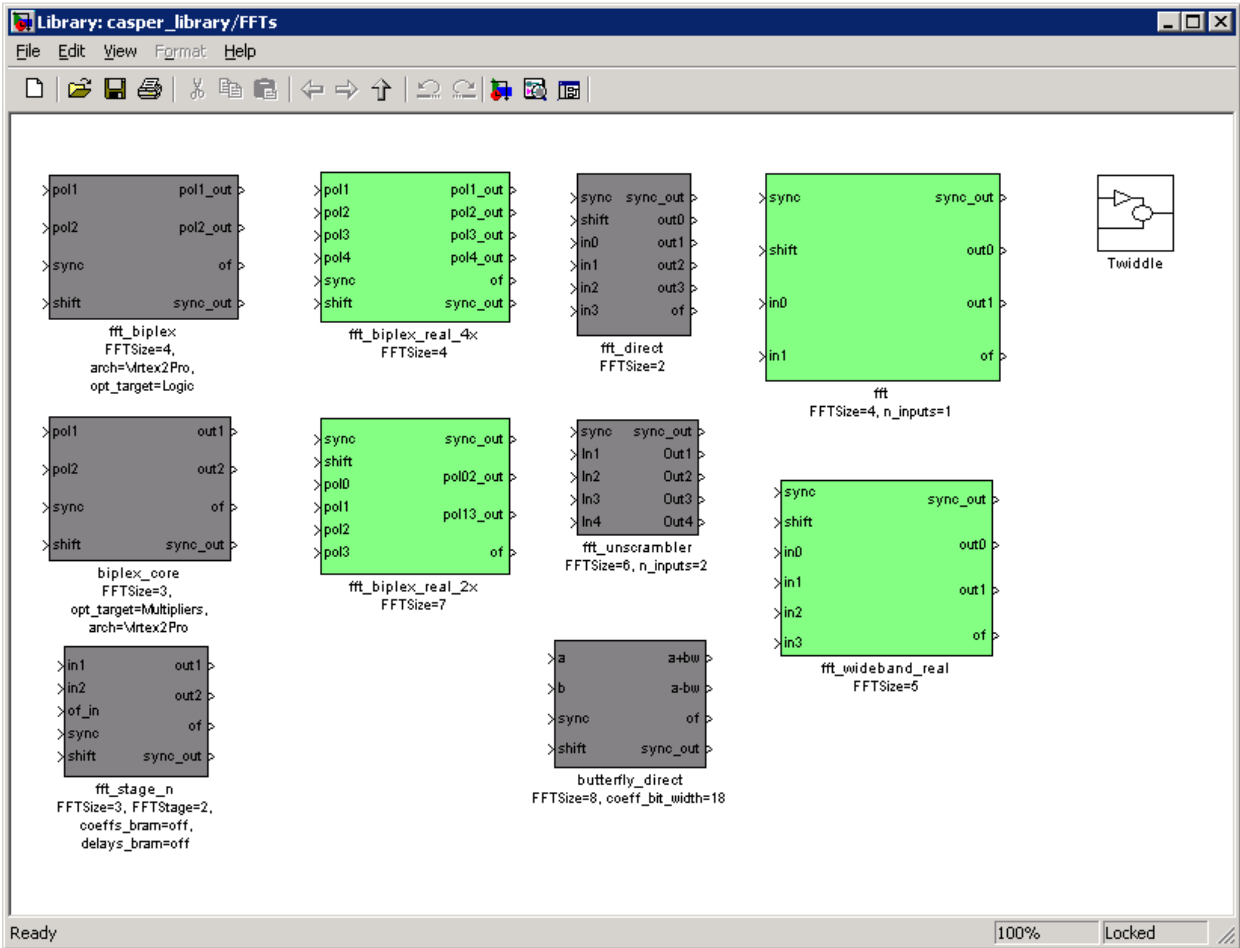


Figure 3.6: CASPER FFT Library

multiple samples in parallel and blocks that are optimized to compute the FFT of a real signal.

Figure 3.7 shows the options menu for one of the FFT blocks. In order to support a variety of instruments, this block can be reconfigured to support different FFT lengths. There are a number of other parameters provided like input bit width, which helps support a number of different ADCs or preprocessing algorithms, and FPGA-specific parameters like add latency, and multiply latency which have no effect on the result of the computation but change how the FFT gets mapped into hardware.

The CASPER library also provides an FIR filter than can be coupled with the FFT to create a polyphase filter bank or PFB. Figure 3.8 shows a comparison between the FFT and PFB response. The FFT response (on the left) has a lot of spectral leakage while the PFB (on the right) has a much sharper filter shape and a better frequency response. Despite the additional FPGA resources required by the FIR filter before the FFT, many instruments

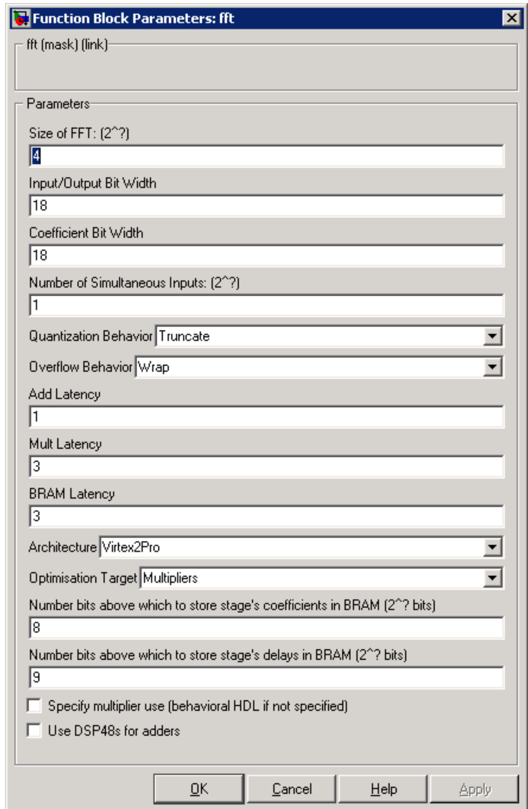


Figure 3.7: CASPER FFT Options Menu

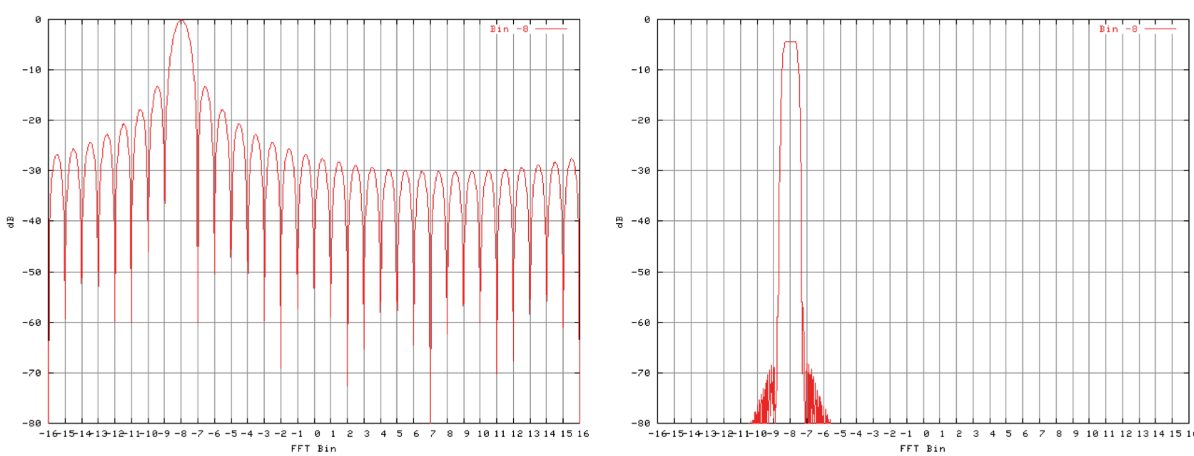


Figure 3.8: A comparison of FFT and PFB response

implement a PFB rather than an FFT because of the superior frequency response.

The CASPER library also provides a set of blocks to abstract away the implementation

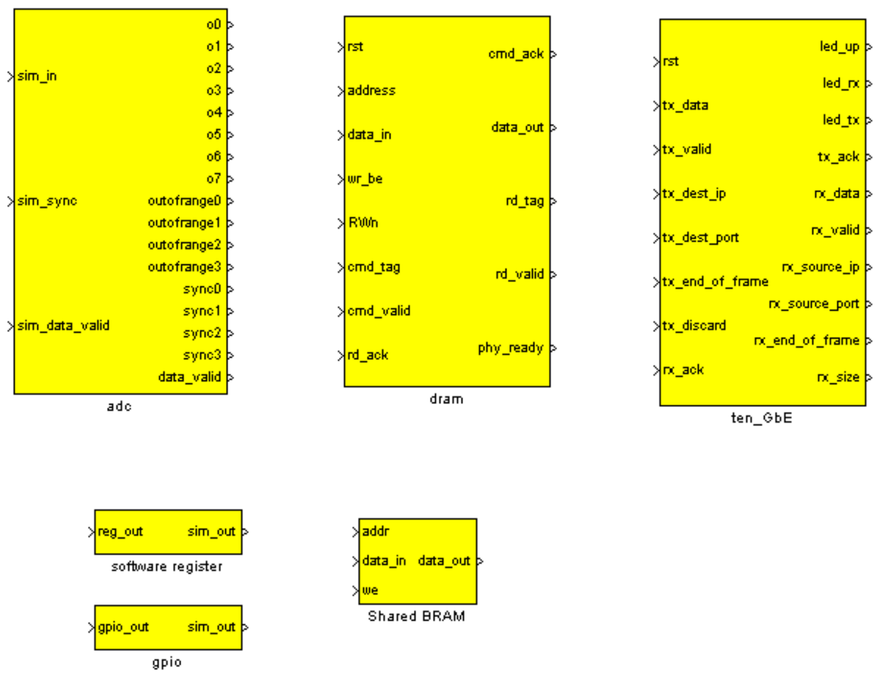


Figure 3.9: CASPER Library I/O Blocks

of I/O interfaces, which are called yellow blocks. Figure 3.9 provides some examples of what those library blocks look like. Each block provides input and output ports that correspond to the data it can send or receive. For example, the adc block has an output ports labeled *o0,o1,...,o7* that represent the data the FPGA receives from the iADC board. In addition to this, simulation ports are provided to allow the user to proved test signals mimicking the outside world. In the case of the ADC, the user could provide a sine wave into the *sim_in* port to observe how a sine wave would be processed by the system. During compilation, the CASPER XPS toolflow automatically processes the blocks and ensures the wires are connected to the correct pins.

CASPER Software

To simplify the use of the FPGA further, the CASPER boards run a modified version of Linux directly on the board. Using this Linux environment, called the Berkeley Os for ReProgrammable Hardware or BORPH, programming the FPGA is as simple as running an executable on the command line [24]. Once the board has been programmed, BORPH can communicate with the chip using an interface where some components on the FPGA like registers or memory appear as files in the operating system. These files can be accessed using normal file I/O, making it simple to send control signals or monitor the status of the chip, greatly simplifying debugging and command and control.

SERENDIP V.v

SERENDIP, or the Search for Extraterrestrial Radio Emissions from Nearby Developed Intelligent Populations, aims to find extraterrestrial intelligence by analyzing radio signals [23]. SERENDIP V.v is a high resolution SETI spectrometer that was deployed at Arecibo Observatory in 2009. The installed instrument was built using an iADC, iBOB and BEE2 board. The instrument breaks up a single 200 MHz stream into 128 million frequency channels, achieving a resolution of about 1Hz.

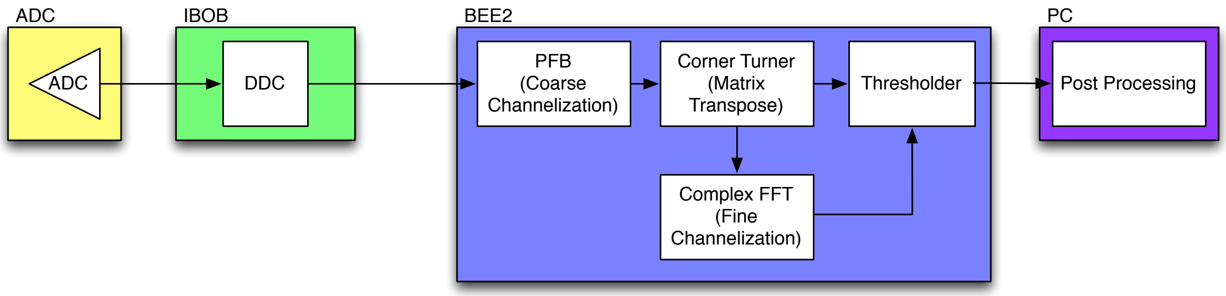


Figure 3.10: SERENDIP V.v Block Diagram

Figure 3.10 shows the dataflow for the spectrometer. The ADC data is fed into an IBOB where it is processed by a decimating downconverter. The downconverter signal is processed by a PFB and then transposed so each PFB channel is further channelized by a complex FFT. Finally, the finely channelized data is thresholded, flagging any high power narrow signals that are potentially extraterrestrial.

The SERENDIP V.v instrument processes data from the 7 beam Arecibo L-band Feed Array. The spectrometer doesn't have enough processing power to process all the beams at the same time, so they are multiplexed and fed through the spectrometer in a cycle. The instrument was successfully deployed at Arecibo Observatory in June 2009. Figure 3.11 shows the installed instrument at Arecibo Observatory. The picture on the right shows the beam multiplexer, and the left picture shows the IBOB and BEE2 boards that were used to develop the instrument.

CASPER Correlator

The CASPER Correlator is a scalable FPGA correlator design developed using CASPER hardware and libraries [17]. Figure 3.12 shows the CASPER Correlator architecture implemented using ROACH boards. The cross correlation (X-Engines) and PFBs (F-Engines) are calculated using ROACH boards, and data is transported between the boards using a general purpose 10 Gigibit Ethernet switch. This design represents a huge step forward from

Figure 3.11: SERENDIP V.v Hardware installed at Arecibo Observatory

custom ASIC correlator design, using reprogrammable hardware to update implementations and providing flexibility in the design.

The switch architecture solves the all-to-all cabling problem, making it easy to add more hardware to the correlator and validating the CASPER instrument design philosophy of attaching general purpose hardware to a switch.

Packetized Astronomy Signal Processor

Many radio astronomy instruments channelize the antenna data as soon as it is digitized. The Packetized Astronomy Signal Processor, or PASP implements a parameterized, FPGA-based channelizer, shown in Figure 3.13. The FPGA interfaces to an ADC board that simultaneously digitizes 2 signals. The samples are sent into a polyphase filter bank (PFB), consisting of an FIR filter and an FFT, which breaks up the entire bandwidth sampled by the ADC into smaller subbands. After dividing up the subbands, each band is rescaled. This step allows us to compensate for the shape of the analog filter feeding data into the ADC. After rescaling, the FPGA forms packets where each packet contains data from a single subband. The packets are sent out over CX4 ports to a 10 gigabit Ethernet switch.

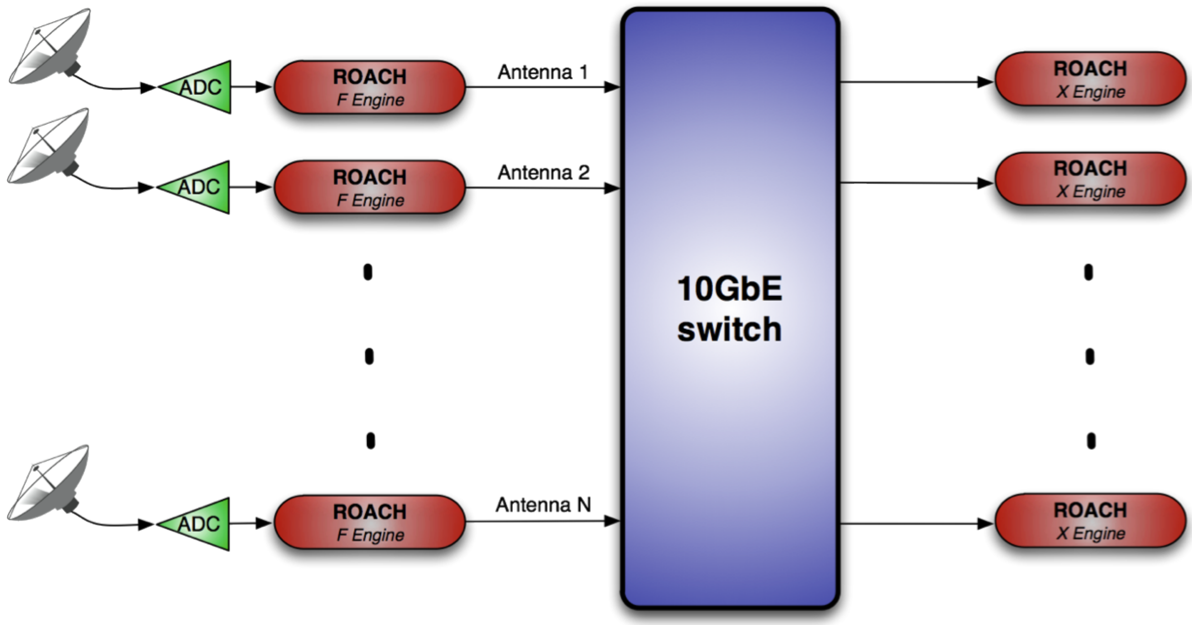


Figure 3.12: CASPER Correlator Architecture

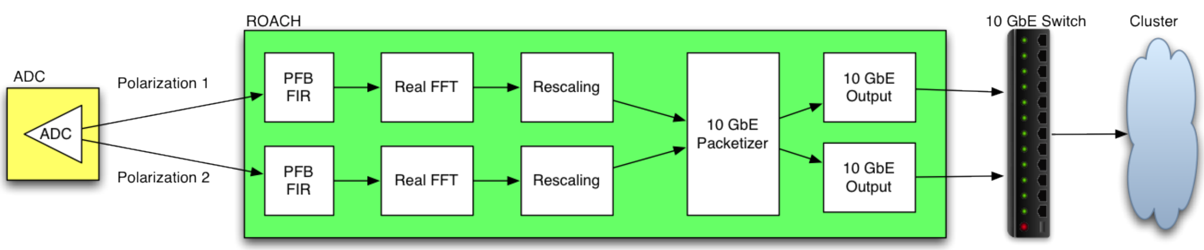


Figure 3.13: PASP Dataflow (reprinted from Filiba and Werthimer [8])

PASP is designed for flexibility. Building on the CASPER goal to automate the design of commonly used signal processing elements such as FFTs and digital downconverters, PASP automatically designs an entire FPGA instrument using only a few parameters. The user can input the desired number of subbands, CPU/GPU cluster size, and packet size and a new design is automatically generated in Simulink. Figure 3.14 shows the high level interface for the instrument. The user can update the design simply by changing a parameter, clicking ok, and recompiling the design. [8]

The interface to PASP is a simple menu that consists of four parameters. From the menu, *Number of IPs* determines the number of IP addresses the instrument will need to send data to, defined by the size of the cluster it must communicate with. *Samples per Packet* defines

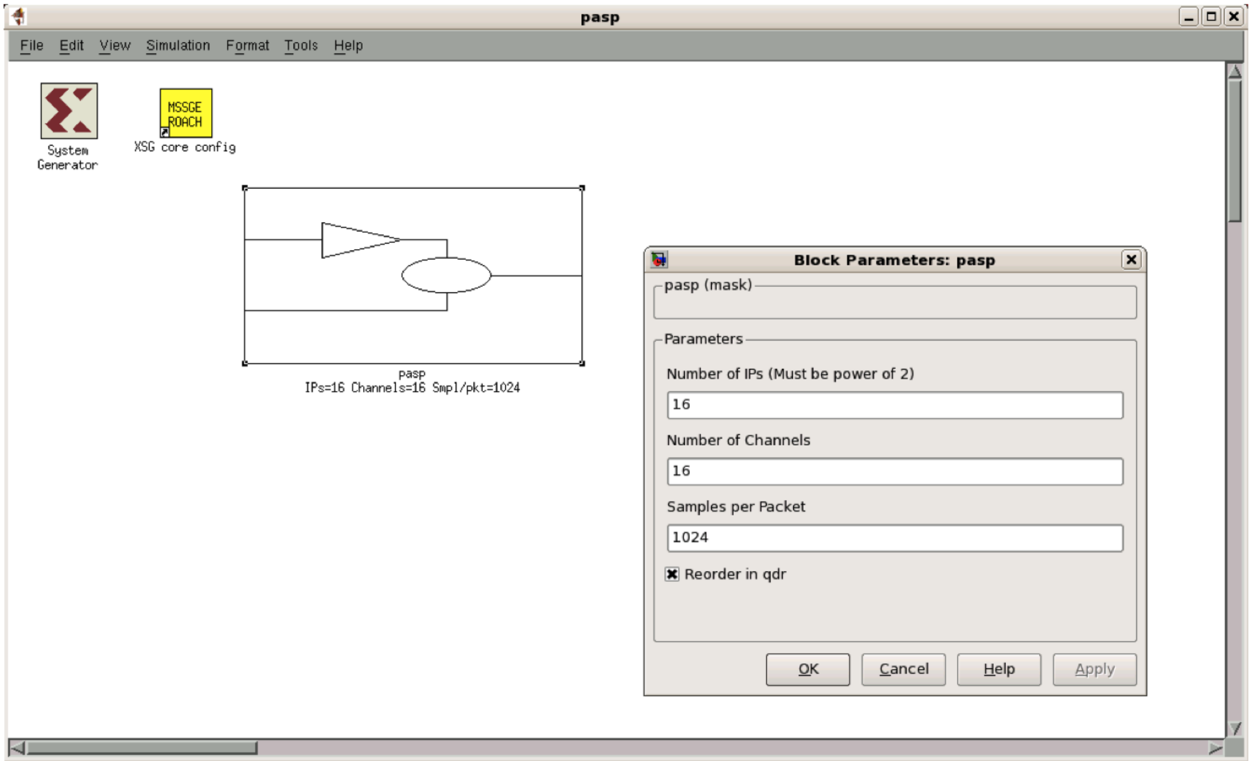


Figure 3.14: PASP Interface

the packet size, adjustable to ensure that the packet rate is low enough for the server to handle. *Number of Channels* defines the FFT length. The last parameter *Reorder in QDR*, allows the packet buffering to be handled by off chip QDR memory. This is only a parameter and not the default because it is only supported by certain boards.

Changing any parameter causes the entire low-level design to be redrawn. Figure 3.15 shows the low level implementation of the instrument. Although this diagram is complex, it is generated automatically by PASP and the user never needs to look at this design to implement a working instrument.

PASP has a wide variety of potential applications due to the flexibility of the server software. In the next sections, we describe a few specific applications than can make use of this package.

Pulsar Processing

This instrument was originally designed to do pulsar science. The fast channelization on the FPGA with no data reduction makes it an ideal pulsar spectrometer, since no information is lost before sending the data to the servers. GPUs provide a good platform for pulsar processing algorithms such as coherent dedispersion [21], which can easily be used as the processing function for the server software distributed in our package. Similar to

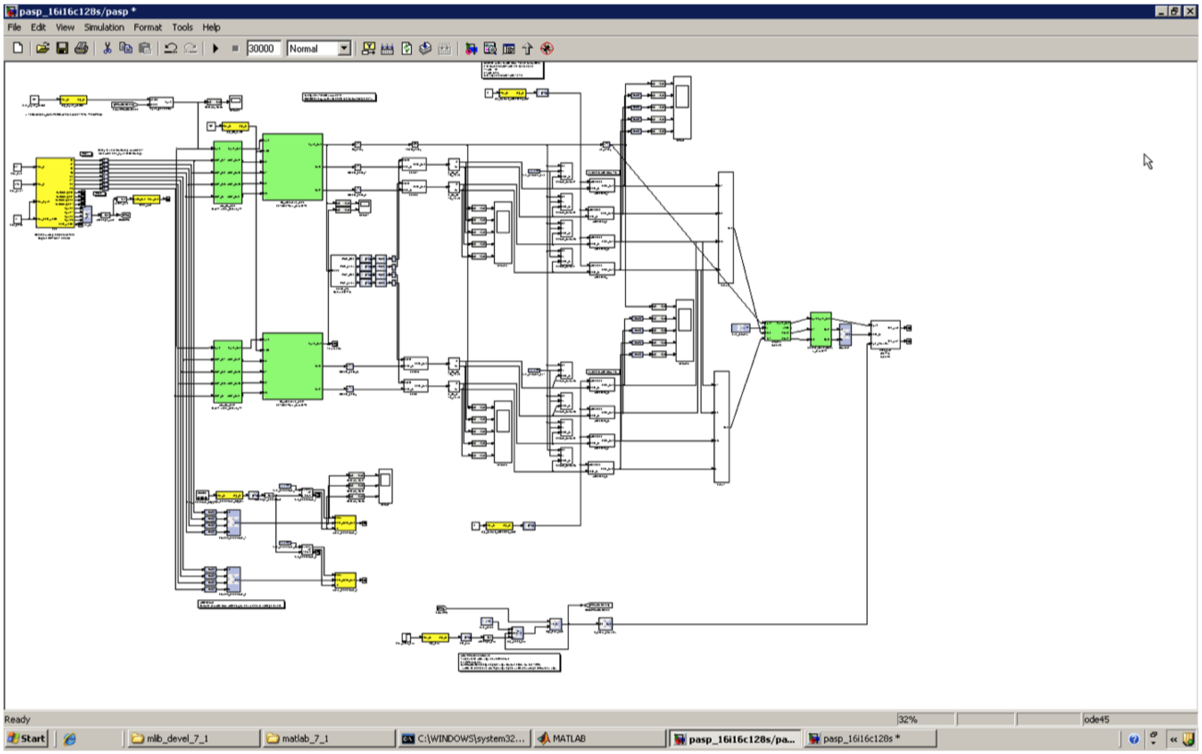


Figure 3.15: PASP Low Level Implementation

SETI instruments, pulsar instruments designed using this package can also keep up with improvements in technology with a simple recompile. [8]

This design has been used in pulsar processors that have been deployed at the Parkes Observatory, Nanay Decimetric Radio Telescope, and Effelsberg Radio Telescope and aided in the discovery of a new astronomical object, a diamond planet [1].

Heterogeneous Radio SETI Spectrometer

In the search for extraterrestrial intelligence (SETI), the ability to keep up with changes in technology allows searching instrumentation to stay on the leading edge of sensitivity. SETI aims to process the maximum bandwidth possible with very high resolution spectroscopy. This instrument allows SETI projects to easily keep up with improvements on the telescope and increasing computational power. An increase in detector bandwidth, improving the breadth of the search, can be processed simply recompiling the FPGA design and distributing the extra subbands to new servers. As computation improves, the instrument can be reconfigured to send more bandwidth to each computer, reducing the required cluster size, or improve the resolution of the instrument by doing a larger FFT on the server. [8]

The design of HRSS is based on the SERENDIP V.v design presented in Section 3.1. Figure 3.16 shows how in HRSS the IBOB and BEE2 are replaced with a single chip FPGA

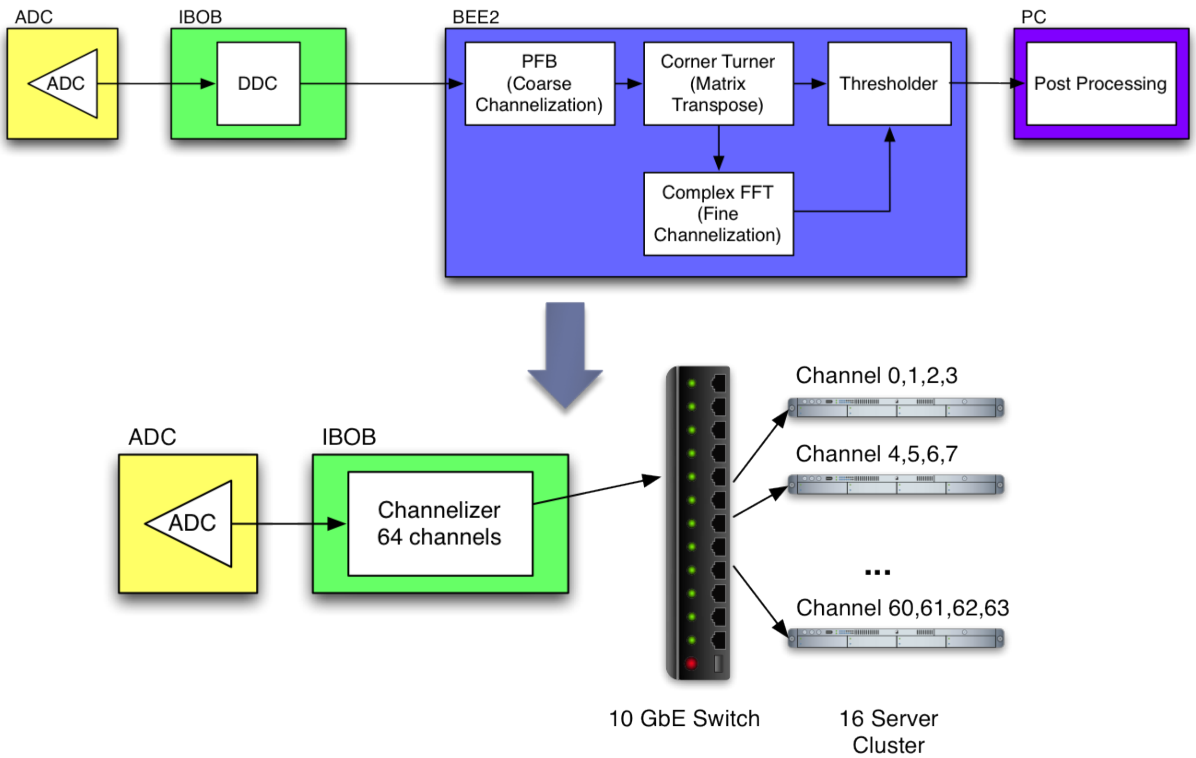


Figure 3.16: HRSS Design Based on SERENDIP V.v

board such as an IBOB or ROACH, and the fine channelization, thresholding and post processing are done on a GPU.

Unlike SERENDIP V.v, HRSS is a software package to automatically generate spectrometers with minimal user input. I automated the design of the spectrometer, creating a parameterized spectrometer that only requires a recompile to implement a change in specification. This spectrometer combines FPGAs running PASP with GPUs, doing coarse channelization on the FPGA and sending each subband to the GPUs for further processing. The server software is designed for flexibility, allowing astronomers to easily modify the processing algorithm run on the GPU and customize the instrument to fit their science goals.

The software package is comprised of two parts, the PASP design that runs on the FPGA and the server software. Like PASP, the server software is parameterized, supporting a variety of spectral resolutions and integration times. This software receives data over an Ethernet port and transfers it from the CPU to the GPU. The GPU runs an FFT and then sends the data back to the CPU to be recorded. The GPU software, like the GPU benchmark, uses the CUFFT library to run FFT. This allows for rapid deployment of a working spectrometer that is configured to take full advantage of available computing resources.

Figure 3.17 shows a high level view of a spectrometer that could be designed with this package. In this example, a ROACH board divides the input band into 64 subbands and sends them out to a 16 server cluster. An ADC is used to digitize data from the telescope and connects to the ROACH board via Z-DOK connectors. The digitized data is split into 64 subbands and sent through a 10 gigabit Ethernet switch. Each server in the cluster receives and processes 4 subbands.

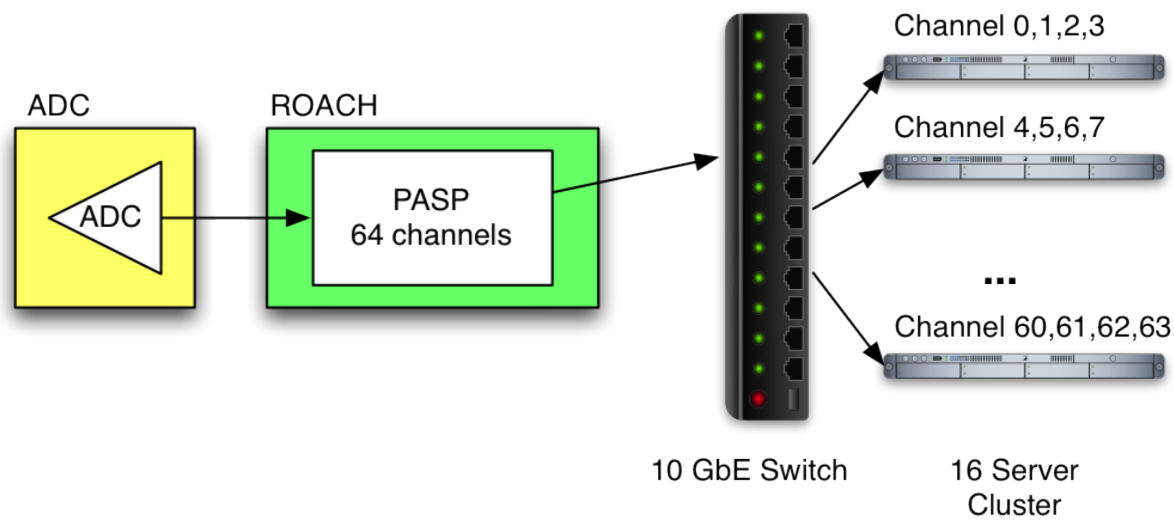


Figure 3.17: Example high level spectrometer architecture

The server software was designed so other applications could easily be implemented on the GPU without altering or rewriting the receive code that interprets the packet headers and transfers data to the GPU. Once the data is on the GPU, the software calls a process function and passes it a pointer to the GPU data. An initialization function is called before the data processing begins to do any setup needed by the processing function, and an corresponding destroy function cleans up once the processing is complete. In the spectroscopy software included in the package, the initialization function creates the FFT plan, the processing function calls CUFFT, and the destroy function deletes the FFT plan. Modifying the application run on the GPU simply requires a redefinition of these three functions. Using this interface, we successfully replaced the CUFFT processing with software developed for SETI searches designed by Kondo et al. [15]. [8]

3.2 Algorithm Tuning

Much of the work in Electrical Engineering relating to this kind of mapping uses a fixed hardware model. In this section, we describe some of that work, and explain why it is not suitable to radio astronomy instrument design.

The Metropolis project focused on mapping algorithms onto embedded systems [4] [7]. The tool provides a framework for an abstract block based description of the algorithm. This description makes it easy to stitch algorithms together without specifying the eventual hardware implementation, providing a simple path to simulation and algorithm development that is separate from the implemented design. Then, the tool automatically maps the description onto an existing heterogeneous embedded system.

The mapping generated by the tool is simply a schedule, specifying where and when each part of the computation gets executed. The tool seeks to optimize performance of the algorithm, ensuring the generated schedule runs as fast as possible on the hardware provided. This technique requires a fixed hardware model and uses the existing hardware to optimize performance. While this work is useful when a hardware model exists, it does not provide any flexibility in the hardware model while mapping the algorithm. So, when it is necessary to design the hardware to begin with, this does not solve the problem.

Additionally, this type of solution is ill-suited to mapping the algorithms required to do real-time radio astronomy signal processing. This tool assumes it must schedule a discrete task onto a fixed piece of hardware and attempts maximize the performance of the task. In the applications described in Chapter 2, the computation should always be running and needs to meet some minimum performance target. Once the performance target is met, it is better to have a tool that will minimize other costs like power or amount of hardware, rather attempting to improve the runtime of the algorithm.

There are many other scheduling projects that have issues similar to Metropolis. Axel [27] uses map-reduce to map computation onto existing cluster nodes. Theodoridis et al. present a tool in [25] that uses integer linear programming to determine if an application can feasibly be run on an an existing heterogeneous system. Again, both of these projects require a fixed hardware model, but this prerequisite that does not exist before a radio astronomy instrument is designed.

Chapter 4

High Level Toolflow

Instrument design is often done by building the instrument from scratch. This work extends the CASPER philosophy, demonstrating that entire heterogeneous instruments can be designed with minimal user input. Rather than designing a completely different instrument for every different specification, this software package is parameterized so new designs can be generated quickly.

In this chapter, I introduce a toolflow I developed called ORCAS or Optimal Rearrangement of Cluster-based Astronomy Signal processing. ORCAS automatically sifts through different designs and chooses the best platforms based on the final cost of the instrument. This automatic approach makes it easy to keep up with constant changes in technology and algorithmic implementations. ORCAS is released as an open source Python library, freely available on GitHub at `https://github.com/tfiliba/orcas`.

4.1 ORCAS Goals

ORCAS extends the CASPER philosophy discussed in Section 3.1 by providing an end-to-end toolflow that allows the user to go from a high level description of the instrument to a low level mapping automatically. While the CASPER toolflow has proven successful in the FPGA domain, it's not always clear that FPGAs are the most cost-effective implementation for a given instrument. ORCAS captures the successful elements of the CASPER tools while extending its reach to the heterogeneous domain. One of the most important elements of the CASPER tools is the library of DSP blocks. The CASPER library blocks are constantly being updated, ensuring that the library can always be compiled on the latest technology. By incorporating existing libraries implemented for CPUs, such as FFTW, or GPUs, like cuBLAS, xGPU, and CUFFT, ORCAS can also keep up with changing technology, without the need for constant updates to the tool itself. The flexibility of the libraries also allows us to buy technology at the last minute, ensuring we get the cheapest price available for an instrument that will provide the needed performance.

In addition to the elements drawn from the CASPER library, ORCAS emphasizes a twofold approach to cost reduction in instrument design. The first cost savings is in the design time. The tool is designed to map designs to hardware within a few hours. This fast mapping makes it possible to quickly assess performance on new or nonexistent technology and allows the designer to assess the impact of potential optimizations, making it easy to test a number of different design ideas without spending time testing different implementations. By incorporating tested libraries, there is much less debugging needed in the eventual implementations.

The second cost ORCAS is able to reduce is the cost of the instrument itself. While the implementation of a design has been greatly simplified by the availability of tools described in Section 3.1, none of those tools help determine which type of hardware is best. The extensive library allows the tool to support whatever hardware is cheapest, allowing the choice of hardware to be completely determined based on cost. This cost could be defined in a number of ways, depending on which parameters the instrument designer needs to minimize. Obviously the cost could refer to the monetary price of the hardware, but the tool can also be used to minimize the amount of power needed by the instrument, the physical size of the instrument, and other user defined costs.

ORCAS is designed to be be accessible to both computer experts and radio astronomers, or other domain experts. For a domain expert, ORCAS allows the user to choose a predefined instrument type and select high level parameters without worrying how it will ultimately map to hardware.

The computer expert can use the tool to define a new instrument, by specifying the algorithm, and is able to provide optimization. For example, if an engineer wrote an optimized FFT algorithm, the tool will be able to incorporate that into the final optimized result. In the end, regardless of who is using this tool, a cost-optimal mapping of the instrument gets produced.

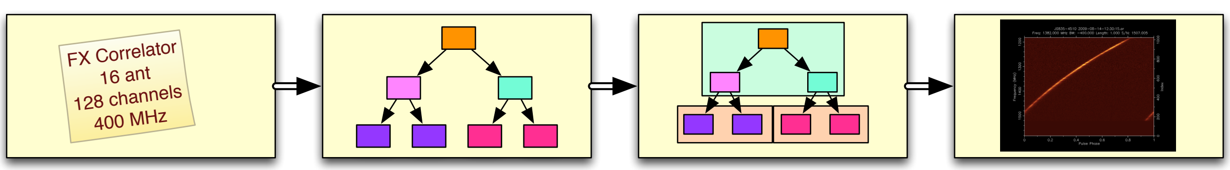


Figure 4.1: The Four Stage ORCAS Toolflow

These goals are achieved through a four stage tool that allows the user to go from a high level description of an instrument to an optimized cluster design. Each stage of the ORCAS toolflow is represented in Figure 4.1. The first stage is instrument definition, which is described in Section 4.2. This stage is designed to fit the needs of the domain expert. It allows the radio astronomer to create a predefined instrument using a handful of parameters. The instrument definition is converted into a dataflow model, as described in Section 4.3. The second stage, the dataflow model, is aimed at the computer expect, since it allows for a more detailed and flexible definition of the algorithm than the previous stage and provides the means to include optimized blocks. The dataflow model represents an abstract definition of the algorithm without taking into account the eventual hardware target. The next stage, appropriately named mapping, maps the dataflow model to specific hardware. In the mapping stage, each block defined by the dataflow model gets mapped to a specific piece of hardware. This stage takes into account hardware and network limitations to produce a cost optimal mapping of the original dataflow. Mapping is discussed in Section 4.4 and Chapter 5 provides more details about the algorithm used to produce a mapped instrument and the performance of that algorithm. Finally, once each block has been mapped to a piece of hardware, the code can be stitched together into a working instrument, as presented in Section 4.5. The rest of the chapter will describe each stage of the toolflow in more detail and explain how they are designed to meet the needs of the users they are targeting.

4.2 Instrument Definition

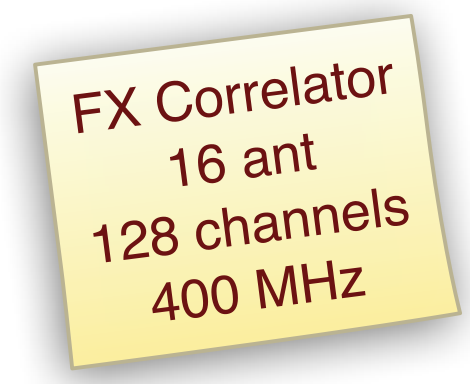


Figure 4.2: ORCAS Toolflow Instrument Definition

In the first step in the ORCAS toolflow, the user must describe the instrument using high level parameters. These parameters should all be relevant to the astronomer and abstract away any implementation details that do not pertain to the scientific goals. While it would be easy to expose many of the low level parameters at this top layer, this would force the domain expert to become a computer expert as well, exactly the scenario this tool is aiming to avoid.

The instrument description as represented in Figure 4.2 fits on a small sticky note. The idea that the parameters should be so few that they fit on a single sticky note was the driving force behind the instrument definition. An instrument designer who finds that he or she needs additional control beyond what is provided by the instrument description always has the flexibility to work with the dataflow model directly, where many low level parameters are exposed. Additional instruments or new parameters for existing instruments can be added by defining how those parameters affect the final dataflow diagram.

In software, each instrument type is represented by a Python class. The instrument parameters are used when an instrument object is instantiated. For example, creating the instrument specified on the sticky note in Figure 4.2 simply requires the following function call (variables are defined for clarity):

```
antennas = 16
channels = 128
bandwidth = 0.4 #defined in GHz

#create the instrument
myfxcorrelator = FXCorrelator(antennas, channels, bandwidth)
```

Design space exploration

In addition to defining a single instrument, an astronomer can use this tool to explore different implementations and assess the tradeoff in cost vs additional processing. Rather than specifying single values for the instrument parameters, the astronomer can choose a range to search through and have it generate a design, and an associated cost, for each value. This proves valuable when the specification of an instrument isn't fully defined. In the case of a high resolution spectrometer, the astronomer could adjust the number of coarse and fine channels, keeping the spectral resolution the same, to find the cheapest design that achieves that resolution. For polyphase filter banks, the number of taps in the FIR can be varied to assess the increase in cost associated with a better filter response. Even non-numerical parameters can be varied, such as the FIR window type, to see how it will affect the eventual design.

4.3 Dataflow Model

The second step of the toolflow translates the instrument definition into a dataflow model. The dataflow model represents the instrument as a set of high level blocks (such as FFTs and FIRs) and input and output connections. The second icon in Figure 4.1 shows an abstract example of this type of dataflow.

Computational Blocks

The computational blocks are a set of algorithmic building blocks. These blocks are similar to the set of blocks provided by the CASPER library or CUFFT, allowing for parameterization, but unlike the afore mentioned libraries, these blocks do not necessarily include an implementation of the function they claim to compute. Instead, each block provides some method to assess the performance of that function on each supported platform.

One way to assess performance is to simply include the code, generating a model by synthesizing or running the code and getting real performance data. But this is not always necessary or desirable. Many algorithm designers provide benchmarks that can be used directly without needing to download or run the code [10] [13]. The predictable scaling of many DSP functions can also make it easy to create a model for resource utilization [20]. And finally, the user can always estimate the performance of a block, a useful option when attempting to assess the cost savings of a potential optimization or explore mapping to hardware that does not exist.

Connection Types

In order to reduce the total number of connections that need to be defined, blocks that receive data from and send data to the same set of blocks are put into a group. Figure 4.3 shows an example FX Correlator dataflow model, where blocks grouped together have the same color. In this example, the FFTs all take data from an FIR block and must send data to the XEng blocks. The name of the group of blocks is referred to as the 'block type'.

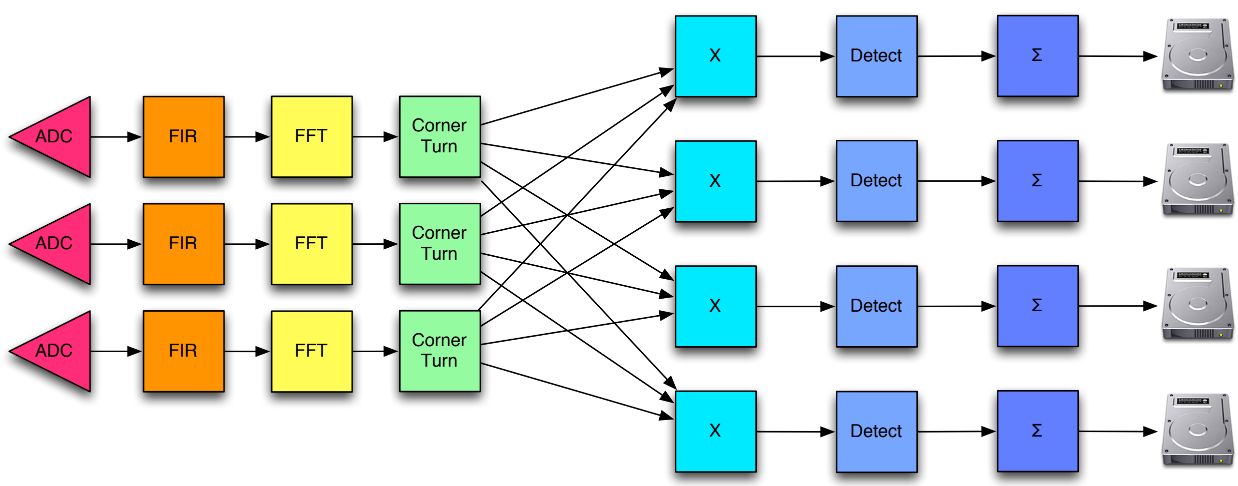


Figure 4.3: Example FX Correlator Dataflow Model Demonstrating Blocktypes

Knowing which block types must communicate with each other is not sufficient to specify the dataflow. Suppose we know we have 2 types of blocks: A, and B and blocks of type A must send their data to blocks of type B. We also need to definite the communication patterns between A blocks and B blocks. This could happen in 2 ways, 'one-to-one' and 'all-to-all'.

A 'one-to-one' connection is where every A block communicates with exactly one B block, as shown in Figure 4.4. With this type of connection, the number of A blocks must be equal to the number of B blocks. The F-Engines in an FX correlator are a good example of this type of connection. The correlator has an F-Engine for each antenna, each containing the same

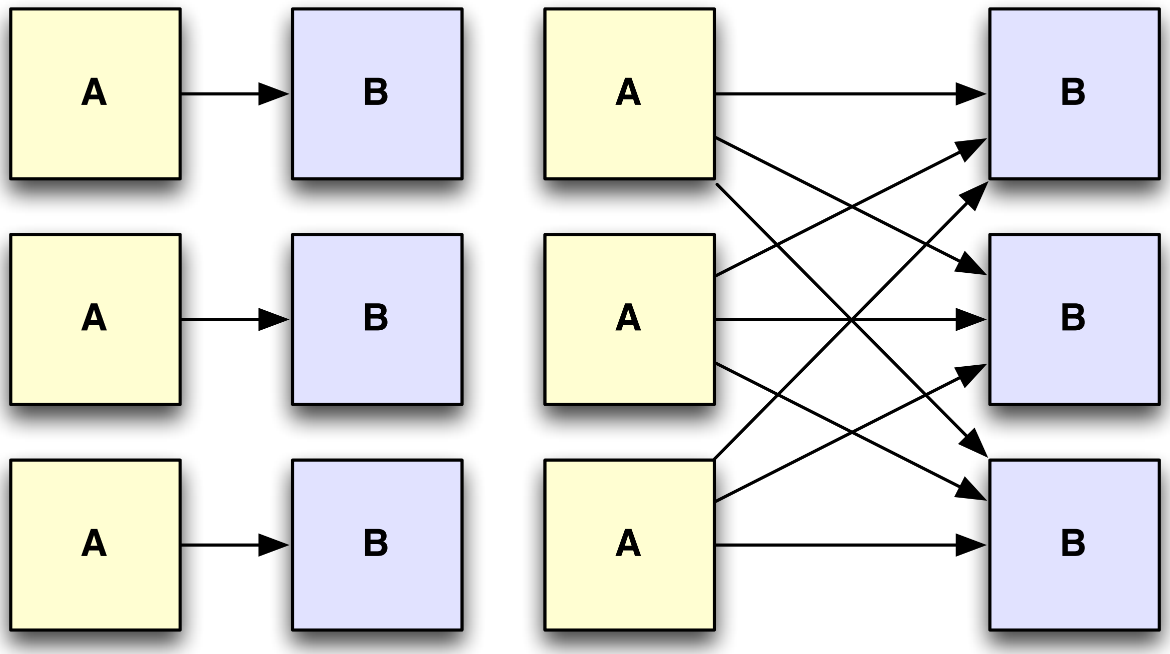


Figure 4.4: One-to-one and all-to-all connections

blocks linked in the same way. Within an F-Engine, an FIR filter must communicate with a single FFT. In general, every FIR within an F-Engine, blocktype 'A' must communicate with exactly one FFT, blocktype 'B'.

An 'all-to-all' connection occurs when every A block must send some data to ever B block. Figure 4.4 shows what an all-to-all connection between three A blocks and three B blocks will look like. In this case, every A block must send some data to every B block. For example, the type of connection between the per-antenna FFTs and the per-channel X-Engines in an FX correlator would be 'all-to-all'. Each X-Engine needs a small amount of data from every F-Engine to compute the cross-correlations from a single channel. In the 'all-to-all' case, there is no reason for the number of sending nodes needs to be the same as the number of receiving nodes.

It might seem like there are two more possible types of connection that need to be defined, 'one-to-all' and 'all-to-one'. A dataflow with a 'one-to-all' connection, shown on the left in Figure 4.5 would have exactly one A block that needs to send data to many B blocks. This is exemplified in the dataflow for a high-resolution spectrometer. The coarse channelization is done in a single FFT block, which then needs to send the data to many other FFTs to do the fine channelization. The 'all-to-one' connection is also shown on the right in Figure 4.5 is the reverse of the 'one-to-all' case. In this type of connection, there are many A blocks and they all need to send data to a single instance of a B block. An example of this arises when some processing is done in a distributed manner but the instrument needs to record

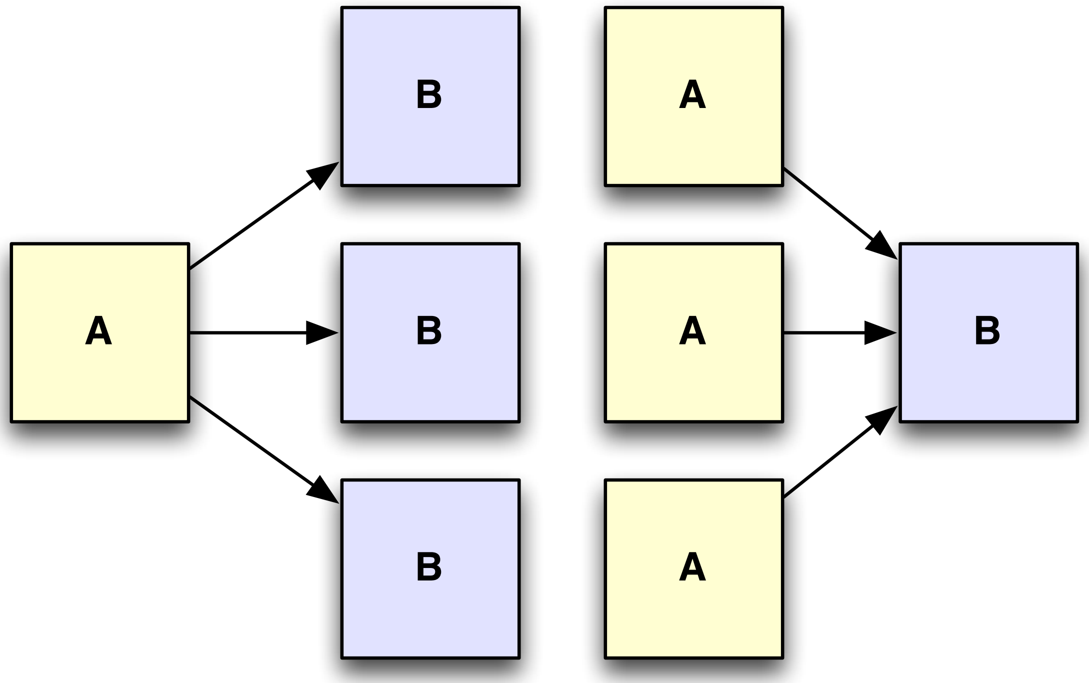


Figure 4.5: One-to-all and all-to-one connections

the final result in a central place.

It turns out, these are both special instances of the ‘all-to-all’ connection. The ‘one-to-all’ connection is simply an ‘all-to-all’ where the number of A blocks is fixed at one. Similarly, the ‘all-to-one’ connection is also an ‘all-to-all’ where the number of B blocks is fixed at one. Because of this, there is no need to include or support these cases as unique connection types.

While it may seem like additional link types exist like ‘all-to-some’ or ‘one-to-some’, this turns out to be impossible. A block of type A cannot send its data to only some blocks of type B because of the way blocktypes are defined. Any block of the same type should be interchangeable with another block of the same type. In an ‘all-to-some’ connection, blocks of type A would need to send data to B_1 but not send data to B_2. But that connection patterns implies that the blocks B_1 and B_2 are *not* interchangeable and therefore cannot have the same blocktype.

Software Representation

The dataflow model is created in the initialization function for the instrument. Each block type is added to the dataflow by creating an instance of the CBlock class and adding that object to the list of blocks. The constructor for the CBlock has the user define the number of blocks in the group, and the input and output connections, indicating the blocks it must communicate with, the connection type, and the connection bandwidth. A CBlock also needs to be instantiated with a performance model, which is discussed further in the following section. The code below gives an example of this kind of instantiation. It creates 16 FFT blocks that must take data from FIR blocks via a 1-to-1 connection, and sends data to XEng blocks via an all-to-all connection.

```
performance_model = CBlock.getFFTModel(self.platforms, bandwidth,
    input_bitwidth)
data_source = 'PFB'
source_connection = 0 #0 indicates a one-to-one connection
source_bandwidth = 0.4*8
data_sink = 'XEng'
sink_connection_type = 1 #1 indicates an all-to-all connection
sink_bandwidth = 0.4*8
antennas = 16   # indicates the number of blocks to create
self.blocks['FFT'] = CBlock(performance_model,data_source,
    source_connection,source_bandwidth, data_sink,
    sink_connection_type,fft_out_bandwidth,antennas)
```

4.4 Mapping

Once the dataflow and the computational blocks are defined, ORCAS must determine how to place each computational block into hardware. In the mapping stage, ORCAS determines what type of hardware should be used for each block, while minimizing the total cost of the hardware (in dollars, watts, or another used-defined metric). The third icon in Figure 4.1 represents an abstract mapped dataflow.

At this point, the computational blocks and dataflow can no longer be viewed as abstract algorithms. Each computational block must be paired with a performance model for each supported platform that shows the resource utilization and bandwidth requirements for that block. Using the performance model, the tool is able to test a number of hardware mappings and ensure that none of the available hardware or bandwidth resources are overmapped.

The optimal mapping is determined using an Integer Linear Program or ILP. The resources, such as memory, logic and CPU time, and bandwidth constraints are translated directly into ILP constraints. These constraints are used to determine a valid mapping, for example total bandwidth mapped to a link must be less than total link bandwidth. The variables represent the design decisions, determining where each block should be implemented. And finally, the cost function simply totals up the costs associated with each piece of hardware used.

A ILP was chosen because it has a number of positive features. Unlike a randomized algorithm such as simulated annealing, the results from a ILP are repeatable. While there might be multiple solutions with the same cost, each time the same ILP is run, it is guaranteed to find one of the solutions of optimal cost. The ILP representation also makes it easy for the user to guide the algorithm. Since the design choices are represented by variables, they can also be restricted by adding additional constraints to those variables. This representation also makes it easy to build out an existing cluster, by allowing a limited amount of zero-cost hardware.

Unfortunately, the advantages of the ILP come with a high cost, namely that an ILP is NP-Hard to solve optimally. Current ILP benchmarks are able to solve problems with a hundred thousand variables in a few hours, but beyond that size the problems become infeasible. To make matters worse, the ILP runtime is very sensitive to the solver being used and the problem structure.

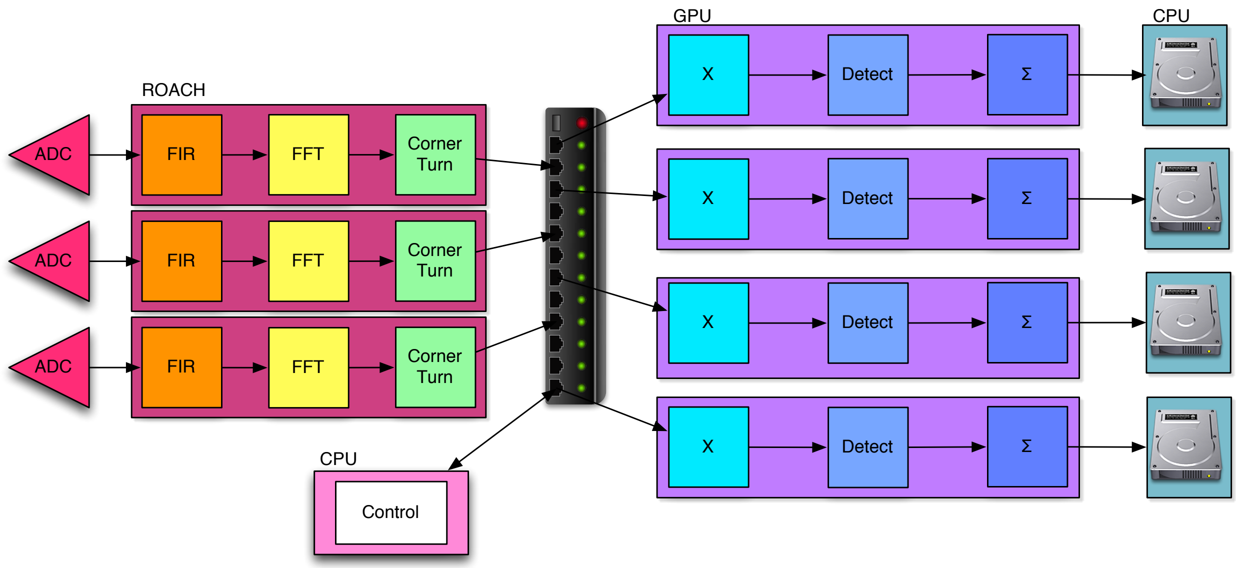
Because the runtime for the ORCAS mapper needs to be within a few hours, we use a number of techniques to reduce the ILP runtime. First, the easiest way to reduce the runtime without changing the ILP is to change the solver. There is a huge amount of variance in runtimes between different solvers and simply switching out the backend solver might cause a previously infeasible problem to become solvable. When that doesn't improve the runtime enough, it becomes necessary to modify the ILP. One way to do this is by reducing the number of variables it needs to solve for. Many of the radio astronomy instruments are very symmetric, so it is reasonable to assume that the optimal mapping would be symmetric as well. The symmetry can be preserved by forcing every block of a certain type to be implemented in the same kind of hardware, drastically reducing the number of decisions that the ILP needs to make. The ILP can also be modified to ensure that there is a single, unique optimal solution. When designs are very symmetric, the ILP will often find an optimal solution quickly but, because it is not unique, the ILP must spend a lot of time convincing itself that the solution is, in fact, optimal. Additional constraints can be added to ensure that only one of those solutions is valid, greatly reducing the amount of time it takes to verify optimality.

Chapter 5 goes into more detail on how the performance models are used as well as how the ILP is specified, implemented, and optimized to achieve a feasible runtime.

Mapped Dataflow Representation

Once the mapping is complete, the dataflow model is updated to describe what type of hardware will be used to implement each computational block. Figure 4.6 shows two or many possible mappings for the FX Correlator dataflow shown in Figure 4.3. In the top dataflow, each F-Engine and X-Engine require so many resources that they must be implemented on separate boards. Each F-Engine is implemented on an FPGA-based ROACH board and the X-Engines are implemented in GPUs. The bottom dataflow shows a less resource-hungry correlator, allowing all the F-Engines and two X-Engines to share a single board for computation.

In all designs, any links between two blocks that are not mapped to the same board must pass through a switch. In the example, all the links between corner turners and cross-



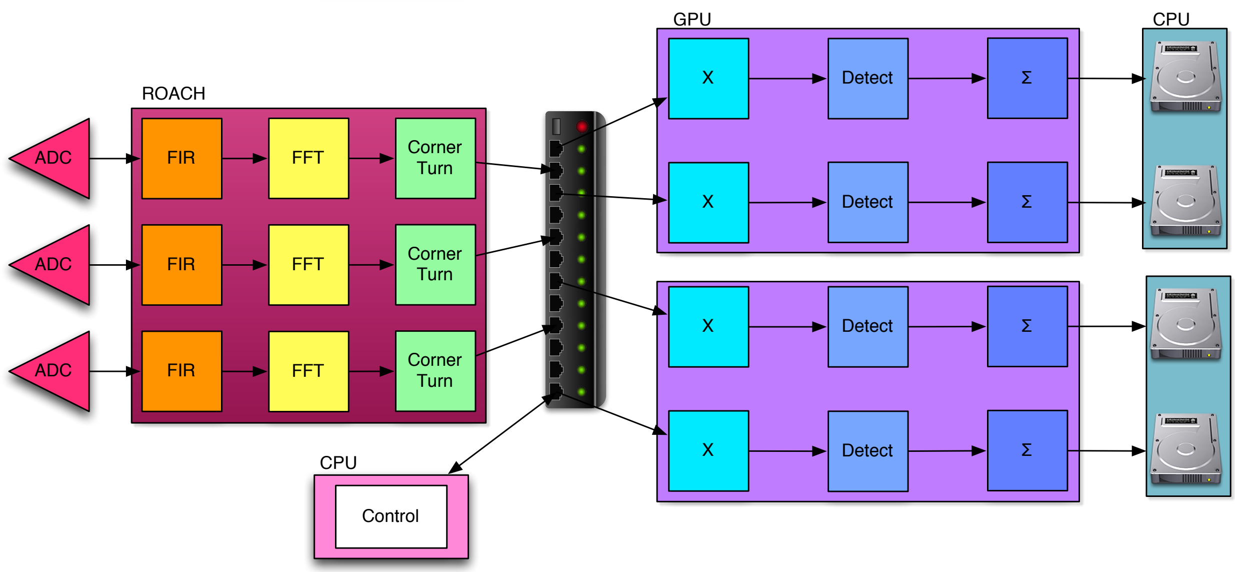


Figure 4.6: Two potential mappings for the FX Correlator

correlation blocks must run through the switch. This is represented in the diagram by a single connection from each ROACH and GPU board to the switch.

4.5 Code Generation

The last step in the process is the implementation the instrument. This step is optional, only being used when the mapped dataflow is being used to design an instrument rather than assess cost. Since the toolflow relies on established libraries, implementation only consists of stitching together existing routines.

In addition to this, common design patterns can be stitched together into parameterized implementations. This style of instrument design greatly accelerates time to science for many projects. Separating the implementation of the instrument from the hardware specification has created a design that works well for a variety of computational resources and applications. As resources improve, the instrument can improve along with them, providing the opportunity to do new science that wasn't possible on older technology. We have shown this is possible by implementing the Packetized Astronomy Signal Processor or PASP using the CASPER toolflow. [8]

Chapter 5

Algorithm Partitioning

After coming up with an instrument description, it is necessary to determine how that instrument will be implemented in hardware. I use Integer Linear Programming (ILP) to model and solve this problem. As described in Section 3.2, ILP is a powerful technique for defining and solving optimization problems. In this chapter I explain how the ILP is defined based on the dataflow model and the techniques used to make sure the program can be solved quickly.

5.1 Variables

The variables in the ILP are represent the optimal mapping for the system. Ultimately, the ILP determines which platforms should be used, and what part of the algorithm should get implemented on each platform. This is achieved by having the ILP consider some platform, and assume it can instantiate at most n_p copies of that platform. We will call a copy of the platform a board. Each board must have some variables that determine which computation blocks get mapped to it. For some board i, the number of computation blocks of type b that get mapped to it is represented by the variable $n_{i,b}$.

A solution to the ILP, with each of the $n_{i,b}$ variables filled in, gives a complete specification of the optimal mapping for that instrument.

5.2 Constraints

The constraints serve two purposes. First, they ensure that no resource is overmapped, so that the amount of hardware the ILP generates will be sufficient to do the computation required. Second, they make sure that the correct design gets implemented.

Platform Resources

Any resource on a board that gets used by mapping computation blocks to that board must be accounted for in the linear program. This is abstracted in the ILP using by adding single constraint for each resource.

Resource Limitations

For some resource, r, we use our performance model of each block to assess how much of the resource is used up by each block type. The percentage of the resource r required by some block type, or utilization, is represented by a constant (not a variable), $r_{p,b}$, where p represents the platform, and b represents the block type. Multiplying the resources required for a specific block type by the number of blocks needed on that specific board determines the total percent of resource r block type b will require on the board. Summing over all of the block types determines what percent of resource r is used in the final design, which gives us the final format of the constraint needed to ensure that some resource r is not overmapped on board i:

$$\sum_{b \in Blocks} n_{i,b} r_{p,b} \leq 1 \tag{5.1}$$

Each resource will require a separate constraint in the ILP of this form. By ensuring that the total utilization of each resource required is less than 100%, we guarantee that there are enough resources to allow all the blocks mapped to that board to complete their tasks.

Dataflow Model Constraints

After getting assurance that no resources are overused, it is important to verify that the correct design was implemented. The resource utilization constrains the value of $n_{i,b}$, but there is an additional constraint on these variables. Namely, the ILP must ensure that the correct number of blocks actually gets implemented. This adds a simple constraint to each block type:

$$\sum_{boardinboards} n_{board,blocktype} == n_{blocktype} \tag{5.2}$$

The total number of blocks of a certain type should be equal to the number of blocks of that type we actually need in the design.

Network Resources

The ILP does not design the network topology. While it would be possible to design the network using an ILP, this would add unnecessary complexity to the program (therefore

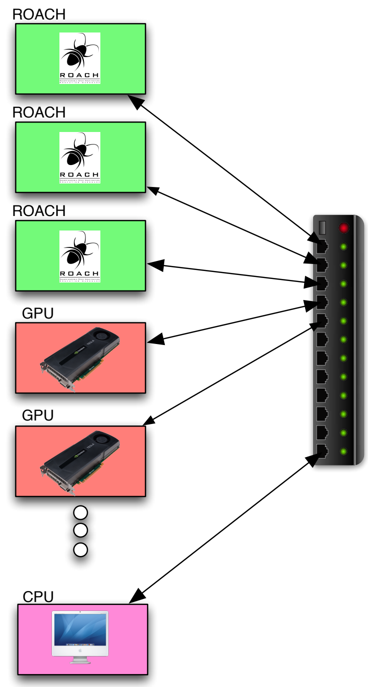


Figure 5.1: Full-Crossbar Interconnect Model

increasing the runtime) with little gain. As described in Section 3.1, most radio astronomy applications require a full-crossbar interconnect at some point, because many computational blocks have an all-to-all or one-to-all communication pattern. Rather than have the ILP redesign the same topology over and over, we simply assume this interconnect exists and every board can communicate with every other board directly over a fixed-bandwidth link.

Figure 5.1 shows an example of this topology. Each board gets connected to the same switch and can communicate with any other board on the switch, regardless of the platform type.

Bandwidth Limitations

Even a fixed network topology, there still are communication constraints that must be taken into account in the ILP. While there might be a link available between each pair of boards, the bandwidth into and out of these boards is limited. Consider Figure 5.1 again. Suppose every other board in the cluster needed to stream 10 Gbps of data to the CPU board, but it is only connected to the switch via a single 10 Gbps link. There are additional constraints to ensure that the input and output bandwidths are not exceeded.

In order to write these constrains, we must first determine how many blocks need to communicate with a block that is not located on the same board. We introduce new variables $nr_{i,b}$ to represent the number of blocks of type b on board i that need to receive data from the cluster, and, similarly, $ns_{i,b}$ to represent the number of blocks of type b on board i that need to send data to the cluster. Given the amount of data some computational block type takes as input and the number of those blocks on the board, we can multiply them together to determine the amount of input bandwidth that computational block type will require. Summing over every block type determines the total amount of input bandwidth needed by all the computational blocks on the board, creating a constraint that the total required bandwidth must be less than or equal to the total available input bandwidth. The constraint on output bandwidth is calculated the same way, generating a pair of constraints for each board, one restricting the total amount of input bandwidth, and another restricting the total amount of output bandwidth.

$$\sum_{b \in Blocks} nr_{i,b} bw\_in_b \leq bw\_in_p \tag{5.3}$$

$$\sum_{b \in Blocks} ns_{i,b} bw\_out_b \leq bw\_out_p \tag{5.4}$$

Connection Constraints

First, we observe that the these variables must be bounded by 0 and $n_{i,b}$, since there cannot be a negative number of blocks that need to communicate, and the number of blocks of type b that need to communicate can't exceed the number of blocks physically on the board. Next, we must take into account the structure of the algorithm to determine whether or not a given block needs to communicate with a separate board.

While the constraints on the total input and output bandwidth might seem simple, ensuring the values for $nr_{i,b}$ and $ns_{i,b}$ are sane requires additional constraints. Suppose we know we have 2 types of blocks: A, and B. A is a source of data, meaning it does not receive data from any computation block. Similarly, block B is a sink, with no data to send to another block. Regardless of how many A and B blocks get placed on platform i, none of the A blocks will need to receive data and none of the B blocks will need to send data. Knowing that A is a source and B is a sink tells us that $nr_{i,A} = 0$ and $ns_{i,B} = 0$.

In order to appropriately define the linear program, it is first important to look at the different ways the computational blocks in a design may need to communicate, and create appropriate constraints. We must revisit the connection types introduced in Section 4.3, and determine how the different types of links affect the linear program.

When two blocktypes are linked via a 'one-to-one' connection, communication is required when the number of A blocks is different than the number of B blocks on a single board. When there are more A blocks then B blocks, $n_{i,A} > n_{i,B}$, the number of A blocks that need to send data to the cluster is $n_{i,A} - n_{i,B}$ and none of the B blocks on that board need to receive data from the cluster. In the opposite case, $n_{i,A} < n_{i,B}$, and the none of the A blocks need to send data to the cluster, but $n_{i,B} - n_{i,A}$ blocks of type B will need to receive data from the cluster. Both of these cases are captured by the same pair of constraints:

$$ns_{i,A} \geq n_{i,A} - n_{i,B} \tag{5.5}$$
$$nr_{i,B} \geq n_{i,B} - n_{i,A} \tag{5.6}$$

When $n_{i,A} - n_{i,B}$ is non-negative, we are guaranteed that we will not underestimate $ns_{i,A}$, and when $n_{i,A} - n_{i,B}$ is negative, $ns_{i,A}$ will be forced to at least 0 because of the lower limit on the variable.

Setting the ns and nr variable is a little more complicated for the 'all-to-all' case, and requires the implementation of some conditional logic in the linear program. When any of the B blocks are not on the board i, then every A block must send data to the cluster, and $ns_{i,A} = n_{i,A}$. Otherwise, none of the A blocks need to send and $ns_{i,A} = 0$. Similarly on the receive side, if any of the A blocks are not of board i, $nr_{i,B} = n_{i,B}$, otherwise $nr_{i,B} = 0$. The conditional logic is easily implemented in an ILP.

Implementing the ILP this way results in an overestimation of the required bandwidth. In the case where $ns_{i,A} = n_{i,A}$, it's true that every block of type A will need to send some data. However, they might not need to send the full bandwidth bw_out_A to the switch, since some portion of the data sent by an A block may be required by B blocks residing on the same board. A more exact version of this calculation would also take into account the smaller bandwidth, but due to the complexity and rarity of this case the approximation is sufficient. As described in Section 3.1, many architectures have this type of connection but there very few cases where the A and B blocks connected this way reside on the same board.

5.3 ILP Implementation

In software, the ILP can be generated automatically using the dataflow model contained in an Instrument object. The generic Instrument class has a single function, called runILP() that iterates through the dataflow model, creating variables and constraints, determines the cost model depending on what the user wants to optimize for and runs an ILP solver to generate an optimal mapping. Adding on to the instrument creation example in Section 4.2, we show how the user can create and map an instrument using only two function calls in the following code.

```
antennas = 16
channels = 128
bandwidth = 0.4 #defined in GHz

#create the instrument
myfxcorrelator = FXCorrelator(antennas, channels, bandwidth)
myfxcorrelator.runILP()
```

5.4 Performance Modeling

The data that the ILP uses to measure resource utilization must come from a preexisting performance model. These models can take a number of forms. Benchmarks of compiled and running code provide the best information, but are also time consuming to obtain if they don't already exist and they require a real implementation of the block. Estimates are faster to obtain but won't be as accurate. Nevertheless, many DSP blocks have predictable performance and a performance estimate based on similar benchmarks can be very reliable.

Benchmarks of the mapped design or running code serve as a very accurate way to assess performance. Figure 5.2 shows the type of benchmarks that would be useful for an FPGA block, measuring utilization of available FPGA resources. These benchmarks were taken by compiling small designs that only had the target block. The designs and compilation results are in the ORCAS Git repository. The graphs show the utilization of registers, LUTs, BRAMs and DSPs used on the Virtex 5 SX95T. The top graph is the data for an 800 MHz FIR filter with 4 taps, and the bottom shows utilization for an 800 MHz FFT. Getting the performance model for a specific block simply requires looking up the FFT size in the graph.

Similarly, Figure 5.3 has benchmarks for the same blocks, FIR filter above and FFT below, but these are tested on a GTX 580 GPU. These benchmarks measure the runtime of each function. The benchmark data was taken by running the routine one hundred times on the target architecture to get an average runtime. These benchmarks are also available in the Git repository for this project.

Many papers also provide these kinds of benchmarks, making it easy to get accurate numbers without installing or running the code. The xGPU paper [3] has a number of graphs demonstrating kernel performance that can be used directly as an ORCAS benchmark, which will be shown in Chapter 6.

Performance data can also be represented using formulas. Figure 5.2 clearly shows some predictable trends in the FIR and FFT utilization. Primiami et al. turned this predictability into a set of equations that determine the requisite resources [20].

A more extreme example of predictability arises in the CASPER X-Engine. DSP utilization on FPGAs has a predictable linear relationship with the number of antennas correlated. Figure 5.4 displays a bar chart of the real benchmark data with a line interpolated through the points. The line passes through every point, and we see the relationship can be characterized by the following equation:

$$DSPs = 8 * antennas + 16$$

We can also use existing benchmarks to project how a block might perform on newer technology. In FPGAs, we notice the number of resources required for a block remains nearly constant between different chips. Table 5.1 shows the resource utilization of an 800MHz 32k channel FFT on three chips from three different generations, the Virtex 5 SX95T, Virtex 6 SX475T, and Virtex 7 VX980T. Aside from the number of LUTs, which slightly dropped between the Virtex 5 and Virtex 6, the number of resources required remains very stable

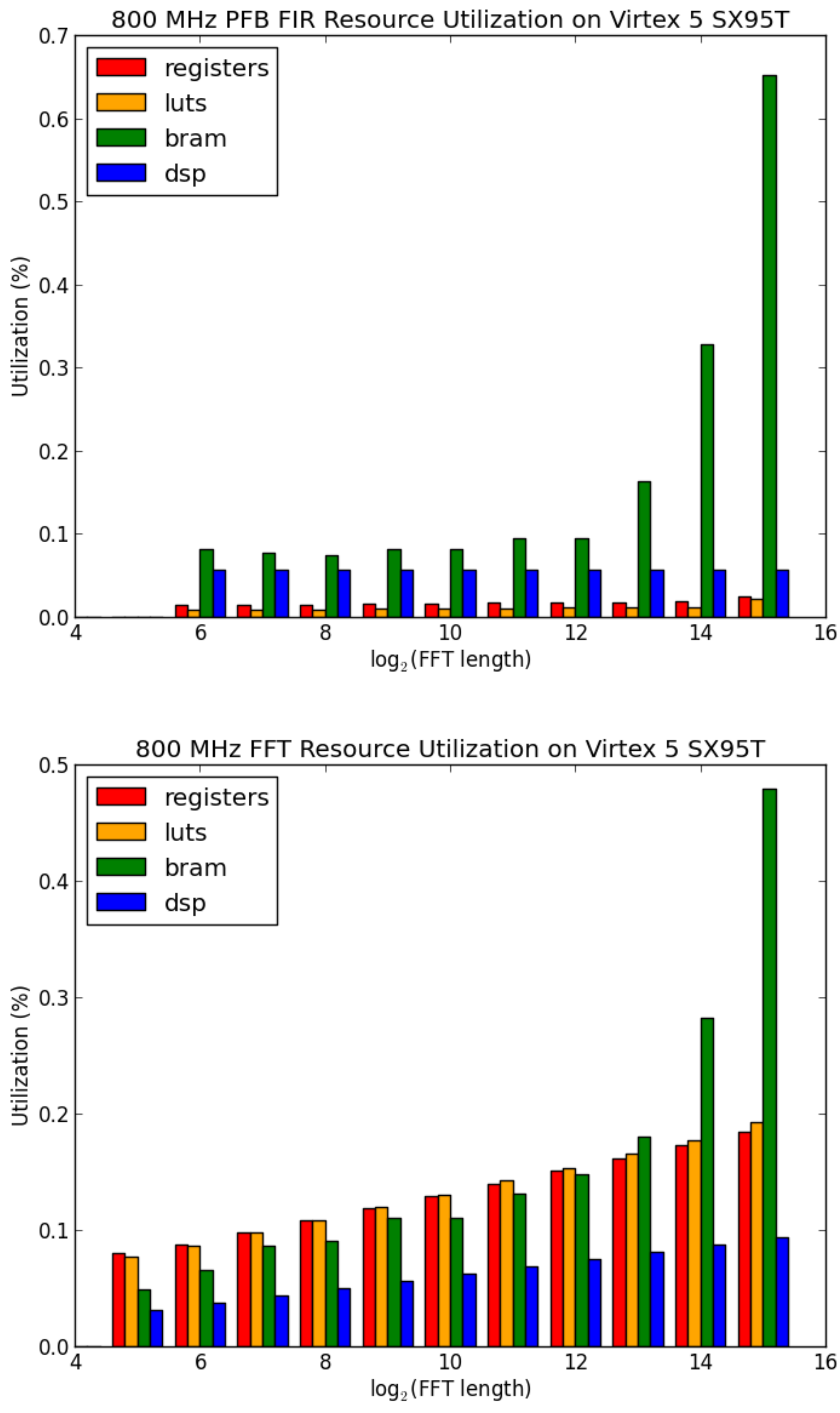


Figure 5.2: PFB FIR and FFT benchmark data on the Virtex 5 SX95T

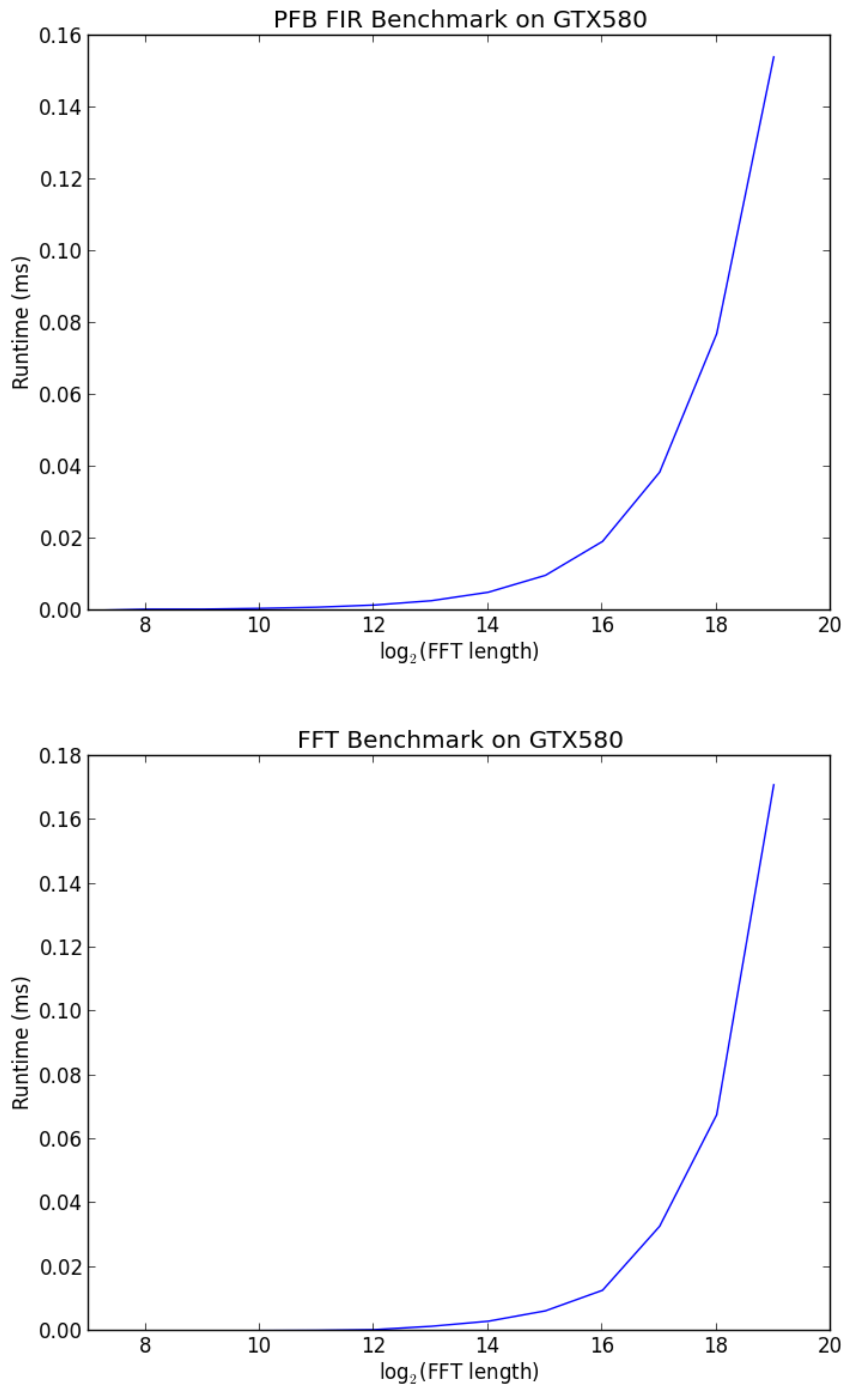


Figure 5.3: PFB FIR and FFT benchmark data on the GTX 580

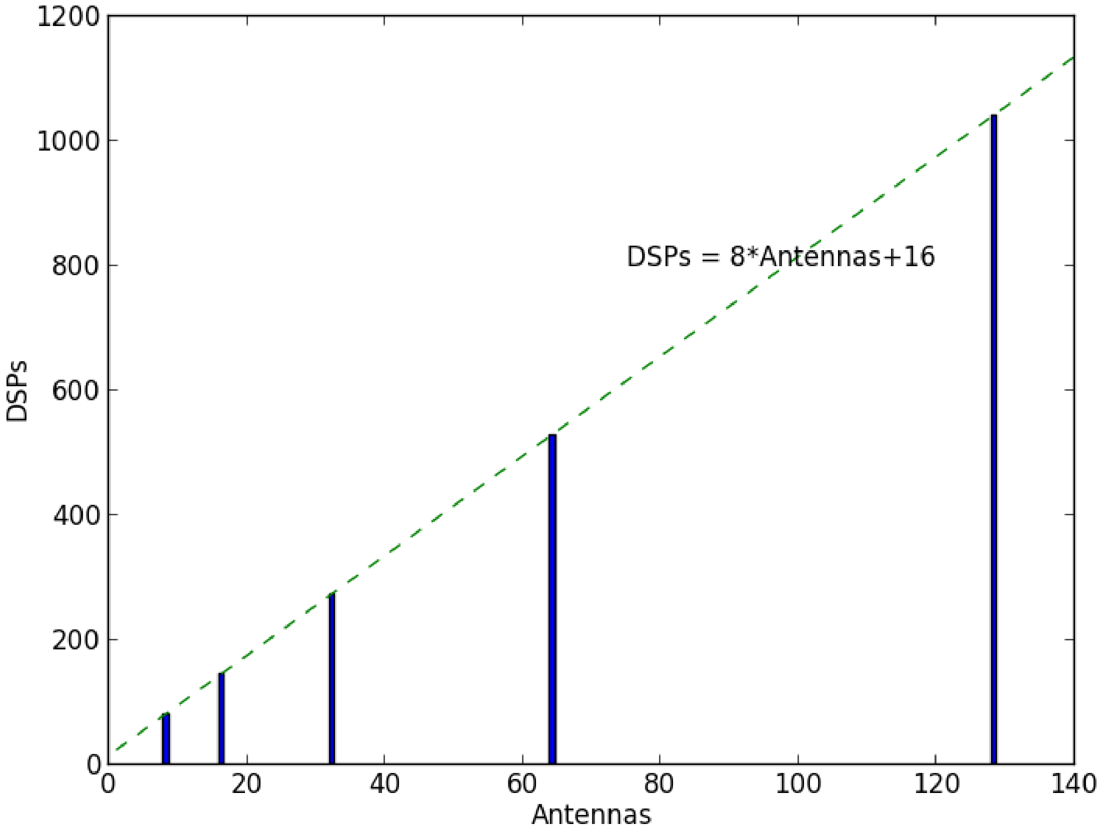


Figure 5.4: Cross-Correlator X-engine DSP Utilization

across chips. Although the utilization will change, because different chips will provide different amounts of resources, recording the number of resources required on one chip makes it possible to predict the utilization on another chip.

Similarly, projections can be made with new processor technology. Even if a new technology is not yet available to buy, a conservative and optimistic estimate of the speedup can be used to generate a conservative and optimistic estimate of the instrument cost using that new technology.

5.5 Cost Modeling

The cost function in the linear program can represent a number of properties like monetary cost, power, development time or rack space. In this work I focus on monetary costs and power. Both can be calculated simply by iterating through the boards used and adding the

Resource	Virtex 5 SX95T	Virtex 6 SX475T	Virtex 7 VX980T
Registers	10881	10789	10788
LUTs	11358	9632	9773
36k BlockRAM	100	100	100
18k BlockRAM	30	30	30
DSPs	60	60	60

Table 5.1: Comparative Resource Utilization of a 32k Channel 800 MHz FFT

Platform	Specification	Monetary Cost	Idle power	Max. power
ROACH	Virtex 5 SX95T	$6,700	55 W	75 W
ROACH 2	Virtex 6 SX475T	$10,500	60 W	80 W
ROACH 3	Virtex 7 VX980T	board in development	60 W	80 W
NRAO Server	GTX 580	$3,500	225 W	475 W

Table 5.2: Monetary and Power Costs for Common CASPER Platforms

GPU	Cost	Idle power	Maximum power
GTX 580	$500	125 W	175 W
GTX 680	$500	100 W	195 W
GTX 690	$1,000	130 W	300 W

Table 5.3: GPU Board Costs

cost of the board to the total cost. Table 5.2 shows the costs for many platforms commonly used in CASPER instruments.

The model can also take into account donated or existing hardware that will not contribute to the monetary cost of an instrument. Adding another platform, like a 'Free ROACH', with same specifications as a ROACH but a monetary cost of $0 will allow the model to use the hardware without incurring any cost. This feature is a useful way to determine if it is worthwhile to replace existing hardware for an instrument upgrade.

The final entry in Table 5.2 is a high performance server, and the reported costs include a GTX 580 GPU, but that might not be the best GPU to use. The data in Table 5.3 is useful to determine how a different GPU will affect the total cost of the server. Also note that the ROACH 3 hasn't been built, so the dollar cost is unknown but we can estimate the power consumption.

5.6 Design Options

The cost optimization is guaranteed to find the cheapest instrument, but it's not necessarily going to be easy to build. Since the ILP will attempt to use every possible resource, it may end up with complex and asymmetrical designs. To simplify these designs, two options can be enabled that will add extra constraints to the ILP.

The first option is called *single design*. This option forces every board of a certain platform type to implement the same design. Instead of allowing any block to go on any board, the designs must be replicated. The second option is *single implementation*. This forces every block with the same block type to be implemented on the same platform type. More simply, if the platforms available are a GTX 580 server and a ROACH and we are placing FIR blocks, all the FIR blocks must be placed on ROACH boards or all the FIR blocks must be placed on GTX 580 servers. These options could drive the cost up, but the tradeoff may prove beneficial, as they both reduce debugging time and complexity in the final design.

5.7 Optimization

While Integer Linear Programming has a number of desirable properties, the lack of an efficient algorithm to solve it constitutes a significant obstacle in designing an ILP with reasonable performance. This section describes how the solver selection, design of this ILP, and the introduction of a few extra constraints serve to improve the performance and scalability of the program defined in this chapter.

ILP Solver Selection

The ILP is described using an open source Python package called PuLP, available at `http://www.coin-or.org/PuLP/`. PuLP does not include an integer linear program solver. Instead, it supports a number of existing solvers, allowing the user to choose which one to use.

ORCAS was originally tested using the GNU Linear Programming Kit or GLPK [9], a free open source package that is supported by PuLP. Unfortunately, as the instrument models became more complex, GLPK often failed to converge on an optimal solution after running for 6 hours on my personal laptop, a 2011 Macbook Air and an attempt to get better performance by using a high powered server was futile.

Because PuLP makes it simple to change the solver, only requiring a change to the single line of code that calls the solver, I decided to test other solvers before editing the ILP. Another solver was chosen by referring to a set of integer linear programming benchmarks publish by Koch et al.[14]. These benchmarks show the feasibility of an ILP is highly sensitive to the solver used. Recent results from those benchmarks, available online [16], found that the Gurobi solver [12] has a relatively high number of successes. Gurobi was able to solve the existing models within minutes and is the solver used to provide all the results in this work.

Guided Optimization

Another solution relies on user aid to guide the mapping. The ILP may spend time going over solutions that are obviously wrong to a human user. In this case, the user could intervene by setting some of the ILP variables manually and letting the ILP find a solution for the remaining variables. While this may be a feasible solution for a computer expert who might have some idea of what the optimal mapping should be, this not a useful technique for the domain specific experts who are not as familiar with the hardware and computational block implementations. This violates one of the basic goals of this tool described in Section 4.1. The tool needs to be accessible and usable by domain specific experts as well as computer experts, and dealing with symmetry in this way will require a computer expert in the loop to generate a mapping and get a cost estimate. To maintain usability for domain experts, guided optimization will not be the sole solution to this issue.

Combining Blocks

A good way to improve the runtime is to combine blocks that are likely to be placed on the same board. Many CASPER designs implement a polyphase filter bank using an FIR filter that is directly connected to an FFT. Observing that these should probably be close to each other, the FIR and FFT blocks can be replaced by a PFB. The resource utilization model for a combined block is created by adding the resource utilizations for each subblock.

This technique can also be used to combine blocks of the same type. ILP design optimization will become infeasible if there are many blocks that require very few resources. If the design is symmetric, as is often the case in radio astronomy, small blocks of the same type will end up in groups on the same board. Grouping them together before running the ILP will result in the same design but a dramatically reduced runtime.

Breaking Symmetry

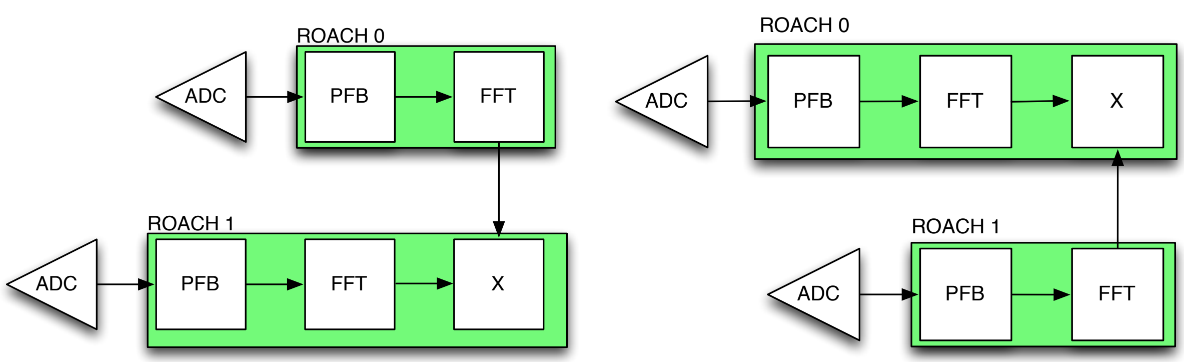


Figure 5.5: Example of Design Symmetry in the ILP

In this type of program, symmetry significantly increases the amount of time required to confirm an optimal solution. The boards that have the same platform type are interchangeable, so if board i implements some design and board j implements a different design in the optimal solution, there is another optimal solution where their designs are swapped. For example, suppose 2 ROACH boards are available to implement an FIR filter and an FFT. The ILP might observe that both blocks cannot fit on a single board and assigns the FIR to the first ROACH and the FFT to the second ROACH. This obviously seems like an optimal solution, but the ILP may also need to check the case where the FFT is placed on ROACH_0 and the FIR is on ROACH_1, only to find that it has the same cost as the previous result. Figure 5.5 shows a more complex example of a cross-correlator where the cross correlation step can be implemented on either ROACH board. Again, there are two possible solutions the ILP can find before convincing itself that either is optimal. In these simple examples there was only one other solution to search, but as the ILP and the search space grows the number of solutions that are symmetric to the optimal case will also grow.

Searching symmetric solutions can become a major time sink, because the ILP solver may find an optimal solution early on, but will require a long time to confirm that it is actually the optimal result, spending time going over many other solutions that are isomorphic to the first one.

This returns the current best result the solver knows of, but it cannot guarantee that the solution is globally optimal or, in the case where it is not a globally optimal mapping, determine if it is close to the optimal solution, since determining that is analogous to solving the ILP. Early stopping works well when it's clear that symmetry is the cause of the long runtime and the amount of time it would take to find one of the isomorphic optimal solutions is short. Even so, the lack of predictable and repeatable results makes early stopping an unappealing solution.

The solutions above outline ways to cope with the existing symmetry. Another way to reduce the runtime is to remove the symmetry altogether. In order to do this, the ILP must be modified so that only one of the isometric optimal solutions is a valid solution to the ILP. First, a variable lex_order_i is added for each board. This variable is meant to uniquely identify the design running on the board; it is simply the concatenation of all the $n_{i,b}$ variables for that board. Note that the ordering of $n_{i,b}$ variables in the concatenation is irrelevant. The only thing that matters is that the order is consistent for every board. When $lex\_order_i = lex\_order_j$ we can infer that for all blocks b, $n_{i,b} = n_{j,b}$. Otherwise, there must be some block b where $n_{i,b} \neq n_{j,b}$. Now that the designs can be identified, they can be ordered. They are simply ordered lexicographically, by adding the constraints described in Equation 5.7.

$$\forall i \geq 1 : lex\_order_{i-1} \geq lex\_order_i \tag{5.7}$$

While this makes the ILP more complex, adding both constraints and variables, it reduces the amount of time the solver takes to find a solution. This lexicographic ordering makes it impossible to swap designs between different blocks, resulting in unique and valid mappings.

Revisiting the symmetry example at the beginning of this section, the design with one FIR and no FFTs would be encoded with a $lex\_order = 10$, and the design the no FIRs and a single FFT would get the encoding $lex\_order = 01$. When the FIR is placed on ROACH_0, then $lex\_order_0 = 10 \geq lex\_order_0 = 01$, satisfying the new constraint. The solution where the blocks are swapped and the FIR is on ROACH_1 violates the new constraint $lex\_order_0 = 01 \ngeq lex\_order_0 = 10$, and will not be considered by the ILP solver.

Generalizing this, it is impossible to take a valid solution (with the lexicographic constraint) and get another valid solution by swapping distinct designs between boards. Suppose, without loss of generality, board i has a design with $lex\_order_i = x$ and board j has a distinct design with $lex\_order_j = y$ and $i < j$. Knowing that the design is valid implies $x \geq y$. Another optimal mapping exists where the designs are swapped and $lex\_order_i = y$, $lex\_order_j = x$, but we are guaranteed that this is not a valid solution to the ILP because it violates the lexicographic ordering constraint.

These constraints have been implemented in the final ILP and they drastically reduce the amount of time it takes to solve the ILP. The additional constraints do not change the cost of the optimal solution, instead they just reduce the number of valid optimal solutions. By modifying the ILP, the performance is greatly improved without sacrificing optimality or usability.

5.8 Final Mapping

The final mapping produced by the ILP is just a list of variable indicating which blocks go on which boards and the cost of the design. Figure 5.6 shows an example of the output produced by ORCAS. The design is a simple wideband spectrometer model. The mapping indicates that the design will use one ROACH board and two GTX 580 servers, placing the FIR and coarse FFT on the FPGA and the remaining fine FFTs on the GPU.

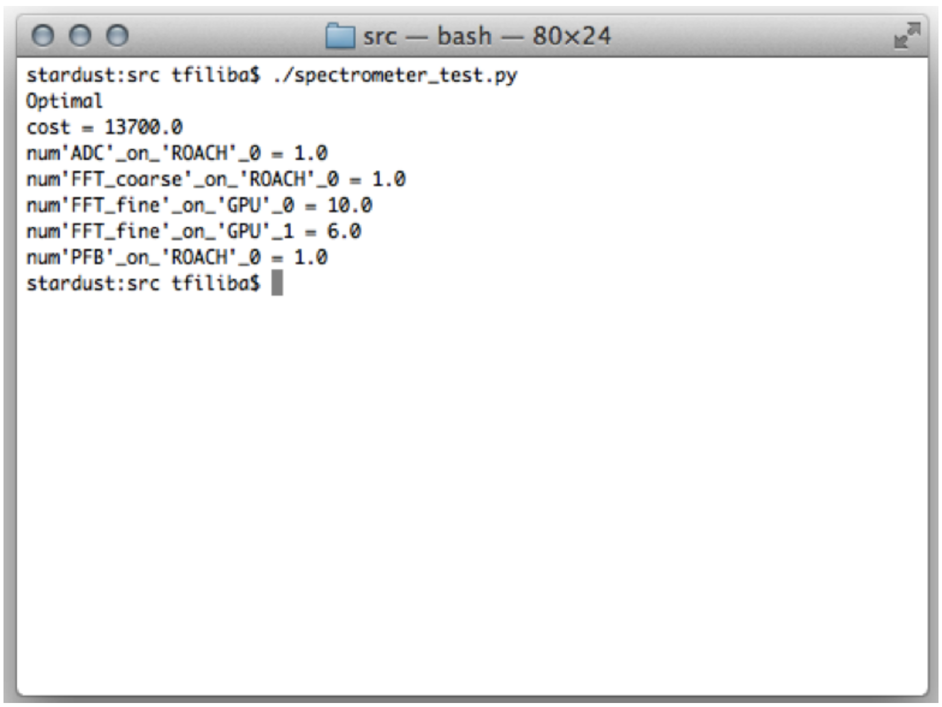


Figure 5.6: ORCAS Output

Chapter 6

Analysis

The ORCAS tool is analyzed by looking at three case studies. We created three instrument types: Spectrometer, High Resolution Spectrometer, and FX Correlator, as defined in Sections 2.1, 2.2, and 2.4. Each case study was chosen to illustrate a different aspect of the toolflow. The spectrometer gives an example of the end to end toolflow using a simple dataflow, the high resolution spectrometer allows us to explore tradeoffs in the design space and the FX correlator shows how the tool behaves when designing very large instruments.

6.1 Spectrometer Case Study

The spectrometer is a simple instrument, making it easy to follow the end to end toolflow. We design an 800 MHz spectrometer that breaks the band into 1024 channels.

Spectrometer Definition

Defining a simple spectrometer requires very few parameters. First, as with most instruments, the astronomer must specify the sky bandwidth the instrument must process, defined in MHz and the number of bits in each ADC sample. Then, the desired spectral resolution is defined in MHz per channel, or analogously, the number of channels that should be used to break up the bandwidth. Finally, the integration time needs to be defined.

One optional parameter, number of antennas, can also be defined. This describes the number of independent spectrometers that need to be created. While this parameter does not affect the end to end processing for each antenna, knowing how many spectrometers are needed allows for more efficient use of the hardware.

The 800 MHz spectrometer is created simply by defining the parameters and instantiating a Spectrometer object as follows:

```
# 800MHz spectrometer
numchannels = 1024
accumulation_length  = 10
```

```
bandwidth = 0.8 #defined in GHz
input_bitwidth = 8
fft_out_bitwidth = 4

#create the instrument
myspectrometer = Spectrometer(numchannels, accumulation_length,
    bandwidth, input_bitwidth, fft_out_bitwidth)
```

Spectrometer Dataflow

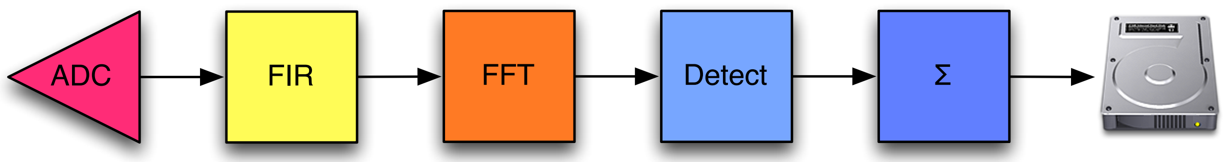


Figure 6.1: General Spectrometer Dataflow Model

The spectrometer instrument definition generates a very simple dataflow. Figure 6.1 shows the general dataflow model for a single antenna spectrometer. This model can be applied to any spectrometer, as the spectrometer parameters do not affect how many computational blocks are required or the interconnect layout. The ADC feeds data into an FIR filter. Then the filtered signal is transformed into channels in the FFT and the complex samples from the FFT are converted to power data by the detect block. Finally, the data from each channel is accumulated and saved to disk. Regardless of the parameters the astronomer chooses, the dataflow will be the same.

The parameters for each block come directly from the instrument definition. The FIR parameters come from the number of FIR taps and window shape, the FFT is simply defined by the FFT length parameter and the accumulator also is parameterized by the FFT length as well as the integration time. In this model and the follow case studies the detect stage and accumulator are combined into a single block because they both require very few resources. The code below shows how the blocks are added to the dataflow model.

```
# add the ADC
adc_bw = bandwidth*input_bitwidth
self.blocks['ADC'] = CBlock(CBlock.getADCModel(self.platforms,
    bandwidth, input_bitwidth),-1,0,0,'PFB',0,adc_bw,antennas)
self.totalblocks += antennas

# add the PFB
```

```
self.blocks['PFB'] = CBlock(CBlock.getPFBModel(self.platforms,
    input_bitwidth, numchannels),'ADC',0,adc_bw,'FFT',0,adc_bw,
    antennas)
self.totalblocks += antennas

# add the FFT
fft_out_bandwidth = bandwidth* fft_out_bitwidth
self.blocks['FFT'] = CBlock(CBlock.getFFTModel(self.platforms,
    numchannels),'PFB',0,adc_bw,'VAcc',0,fft_out_bandwidth,antennas
    )
self.totalblocks += antennas

#add the Vacc
self.blocks['VAcc'] = CBlock(CBlock.getVAccModel(self.platforms,
    fft_out_bitwidth, accumulation_length),'FFT',0,
    fft_out_bandwidth,-1,0,0,antennas)
self.totalblocks += antennas
```

Spectrometer Mapping

As a simple case study, an 1024 channel 800 MHz spectrometer is mapped using the ROACH and GTX 580-based NRAO server as potential platforms. ORCAS produces a solution that maps the entire design to a single ROACH board. This solution is obviously correct since the bandwidth cannot be processed by a single GPU and a single ROACH is cheaper than a combination of boards. The mapping produced by ORCAS is shown below.

```
Optimal
cost = 6700.0
is_used_'ROACH'_0 = 1.0
num'ADC'_on_'ROACH'_0 = 1.0
num'FFT'_on_'ROACH'_0 = 1.0
num'PFB'_on_'ROACH'_0 = 1.0
num'VAcc'_on_'ROACH'_0 = 1.0
```

6.2 High Resolution Spectrometer Case Study

The high resolution spectrometer is a useful instrument for SETI. The two stage channelization creates a large number of channels without using an FFT that is too large to fit on a single board.

High Resolution Spectrometer Definition

The main difference between a spectrometer and a high resolution spectrometer is the need for two stages of channelization rather than just one. The sky bandwidth, integration time, and number of antennas are defined in the same way as the previous spectrometer type.

The spectral resolution is defined differently, because both the coarse and fine resolutions need to be defined. The coarse resolution defines how many channels the whole sky bandwidth should be broken up into initially. The fine resolution defines how many channels each coarse channel is broken into. Both can be described in MHz per channel.

High Resolution Spectrometer Dataflow

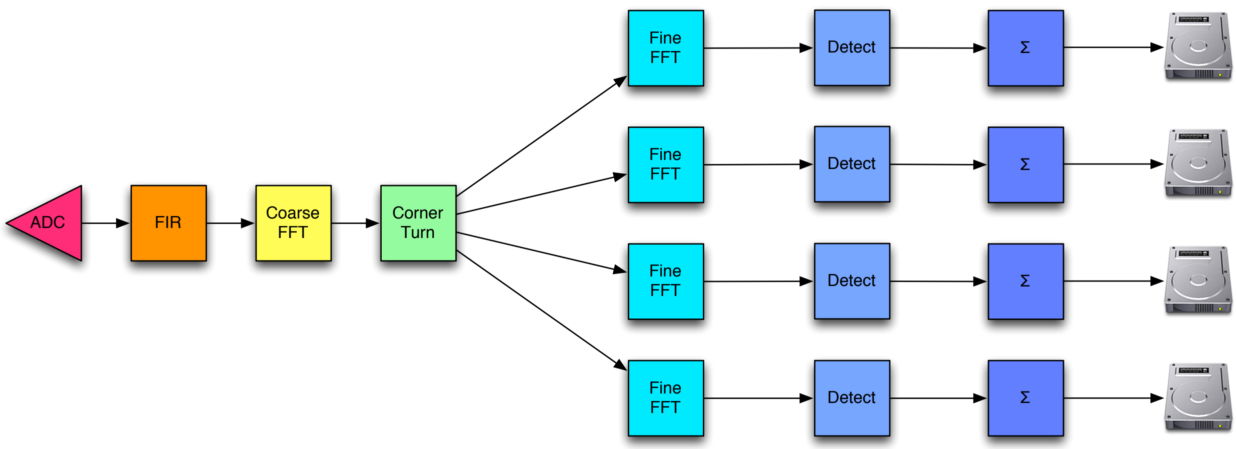


Figure 6.2: Example High Resolution Spectrometer Dataflow Model

The high resolution spectrometer dataflow does depend on the parameters specified in the instrument description. An example dataflow is shown in Figure 6.2. The first three blocks in the dataflow are exactly the same as the spectrometer dataflow described in the previous section. An ADC feeds data into an FIR filter followed by an FFT and reorders the data in the corner turn block, grouping data from the same channel together. After the corner turn, the algorithm is modified to accommodate the higher resolution required. The first FFT divides the band into a number of coarse channels and then each coarse channel must be further divided into a number of fine channels. The coarse FFT must feed its data to a separate fine FFT for each coarse channel, so the number of fine FFTs in the dataflow diagram will vary based on the number of coarse channels. At this point, each coarse channel is processed in an independent pipeline which the finely channelizes the data, calculates the power of the finely channelized data in the detect block, and accumulates and records the data to disk. The example in Figure 6.2 shows a spectrometer that divides the data into 4 coarse channels.

Algorithmic Exploration

The Arecibo L-band feed array, pictured in Figure 6.3 has 7 dual-polarization beams. The SERENDIP V.v instrument was only able to process one beam at a time, but its planned successor, SERENDIP 6, will process 300MHz from each beam-pol.

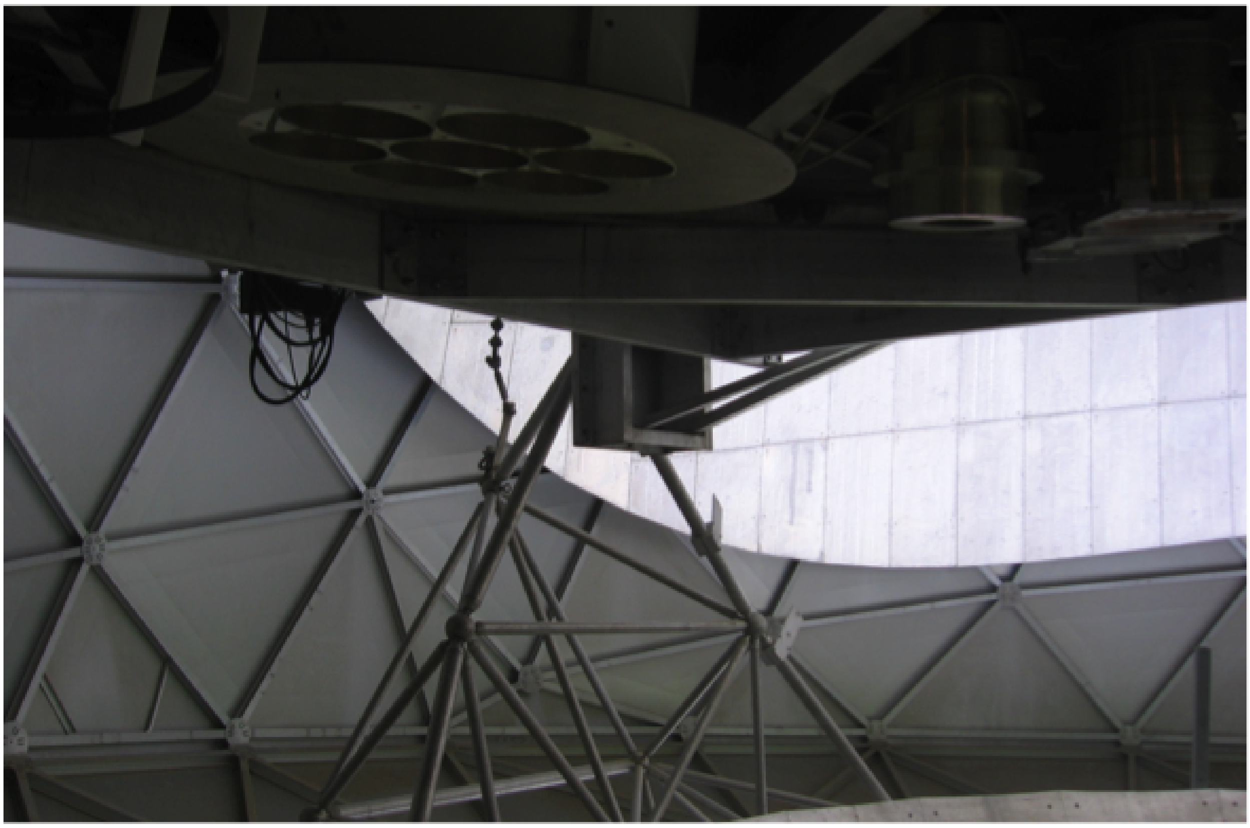


Figure 6.3: Arecibo ALFA Feed

In this case study, we analyze the design space for a 300 MHz 256 million channel spectrometer, similar to the SERENDIP 6 instrument. This instrument provides an interesting case study because the number of channels is so large. The number of channels in the coarse and fine FFTs can be varied, as long as the product remains 256 million. We explore this design space by varying the dimensions and number of antennas to see how the channel balance affects the final cost of the instrument.

This instrument is designed using ROACH boards and the GTX 580-based NRAO server as supported platforms, and uses the FIR and FFT benchmarks presented in Chapter 5. To aid the linear program, we assume reordering the coarse FFT data is infeasible on the GPU and force the design to reorder the data on the FPGA.

Table 6.1 shows the results of this design space exploration. The optimal configurations for each row are highlighted in purple. We observe that extreme values for the number

Dimensions / **Inputs**	256 by 524,288	512 by 262,144	1024 by 131,072	2048 by 65,536	4096 by 32,768
1	2 GPUs 1 ROACH $13.7k	2 GPUs 1 ROACH $13.7k	2 GPUs 1 ROACH $13.7k	1 GPU 1 ROACH $10.2k	1 GPU 1 ROACH $10.2k
3	5 GPUs 1 ROACH $24.2k	4 GPUs 1 ROACH $20.7k	4 GPUs 1 ROACH $20.7k	3 GPUs 1 ROACH $17.2k	3 GPUs 2 ROACH $23.9k
5	8 GPUs 2 ROACH $41.4k	7 GPUs 2 ROACH $37.9k	7 GPUs 2 ROACH $37.9k	5 GPUs 2 ROACH $30.9k	5 GPUs 2 ROACH $30.9k
7	12 GPUs 2 ROACH $55.4k	Not solved	9 GPUs 3 ROACH $51.6k	8 GPUs 3 ROACH $48.1k	7 GPUs 3 ROACH $44.6k

Table 6.1: 134 Million Channel High Resolution Spectrometer Design Space

of channels tend to increase costs, likely because it's difficult to put larger blocks together on the same board and get high utilization of the hardware. Each test was run with a 30 minute time limit on my personal laptop, a 2011 Macbook Air, to ensure that the designs could converge quickly. One test, the 7 antenna 512 by 262,144 channel spectrometer did not converge within the specified time. While it might be able to converge given more time, the resulting table makes it clear that the optimal configuration is unlikely to lie in that square, and there is no need to spend extra time trying to get a solution.

6.3 FX Correlator Case Study

FX Correlator Definition

An FX Correlator is also defined by the amount of bandwidth it processes, number of channels, and integration time, but now the number of antennas is a necessary parameter.

FX Correlator Dataflow

The FX dataflow model is based on the algorithm used by the CASPER correlator described in Section 3.1. The processing model is described by replicating two basic pipelines, called an F-Engine and an X-Engine. The number of times each pipeline needs to be replicated depends on the number of antennas and number of channels this correlator requires.

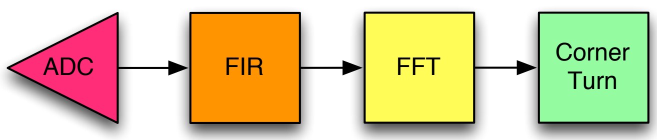


Figure 6.4: FX Correlator F-Engine Model

An F-Engine, pictured in Figure 6.4 is responsible for channelizing the data from a single antenna. It takes in data from an ADC, and channelizes the data using an FIR and FFT to create a polyphase filter bank, or PFB. Then the data from the PFB is rearranged by the corner turn block, by grouping together data from the same channels. The number of F-Engines in the correlator dataflow will vary with the number of antennas.

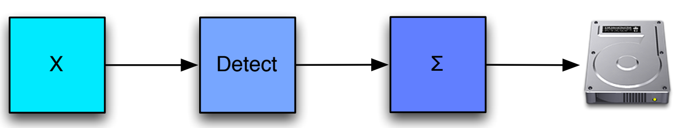


Figure 6.5: FX Correlator X-Engine Model

The second pipeline, the X-Engine, processes the channelized data. Each X-Engine takes a single channel of data from every antenna in the array, cross-correlates the data, calculates the power of the baselines in the detect stage, accumulates each baseline and stores the accumulated data to disk. Figure 6.5 shows the pipeline for a single X-Engine. Since each X-Engine only operates on a single channel, the total number of X-Engines in the correlator must be the same as the number of channels in the FFT.

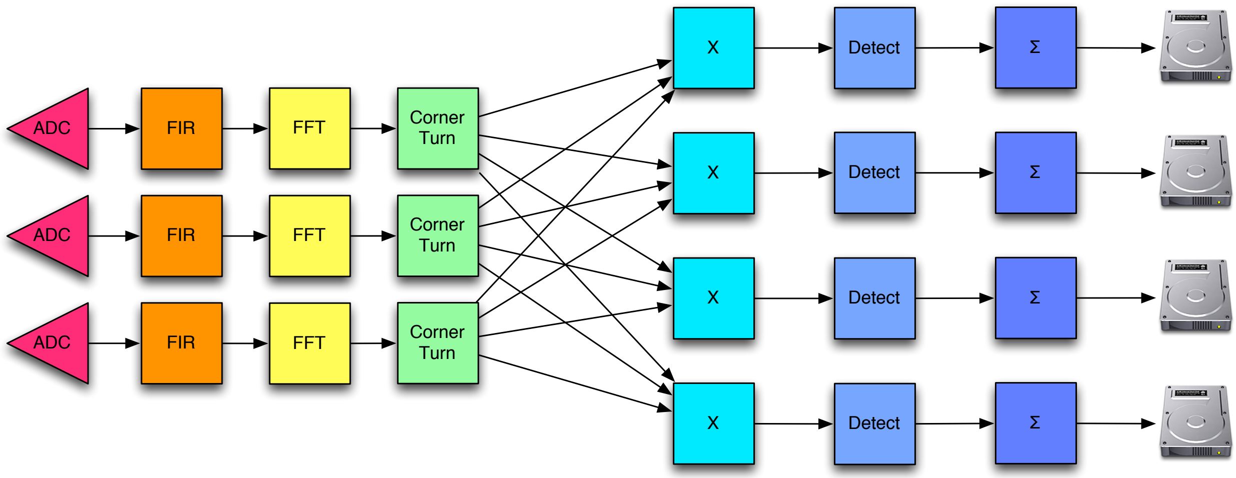


Figure 6.6: Example FX Correlator Dataflow Model

The dataflow for an FX correlator will vary quite a bit based on the input parameters. Figure 6.6 shows an example three antenna four channel FX correlator. The left half of the figure has three F-engines, one for each of the three antennas. The right half has four X-Engines, one for each channel. In the center, since each X-Engine requires data from every F-engine, the cross-correlation blocks, represented by an X and the Corner Turn blocks are connected in an all-to-all configuration.

FX Correlator Mapping

The FX Correlator is evaluated by investigating how correlator design scales from 16 to 512 dual polarization antennas. Note that a 512 dual polarization antenna has 1024 inputs because each antenna produces 2 streams of data. The design is based on the planned HERA Correlator that processes 100 MHz of data and divides it into 1024 channels.

I use the technique of block combining described in Section 5.7 in two ways to ensure the problem can be solved quickly even when the number of antennas is large. We expect the FIR and FFT will be placed together so these blocks are combined into a single block called a PFB. Then, since we expect to see multiple PFBs on the same board, every group of four PFBs is combined into a single block that requires four times the resources of a single PFB.

The X-Engines are combined in the same way, since processing a single channel takes very few resources.

The symmetry of the correlator dataflow makes this a good instrument to observe the effects of the ILP options, single design and single implementation. Just by looking at the results for the 16 antenna mapping with both options disabled, it is already apparent that the ILP is putting small blocks anywhere it can fit them, without much regard for the existing symmetry. In the results below we see that the XEng blocks have been allocated asymmetrically.

```
Optimal
cost = 30600.0
num'ADC'_on_'ROACH'_0 = 3.0
num'ADC'_on_'ROACH'_1 = 1.0
num'PFB'_on_'ROACH'_0 = 2.0
num'PFB'_on_'ROACH'_1 = 1.0
num'PFB'_on_'ROACH'_2 = 1.0
num'Transpose'_on_'ROACH'_1 = 2.0
num'Transpose'_on_'ROACH'_2 = 2.0
num'XEng'_on_'GTX580'_0 = 50.0
num'XEng'_on_'GTX580'_1 = 50.0
num'XEng'_on_'GTX580'_2 = 26.0
num'XEng'_on_'ROACH'_2 = 2.0
total_GTX580 = 3.0
total_ROACH = 3.0
```

In this case, enabling both options doesn't alter the cost but for larger correlators making the design symmetric will require additional ROACH boards. One observed side effect of this approach is the fact that the design might end up creating more blocks than the design originally required. Consider the case where seven 'A' blocks need to be distributed among four boards. If the single design option is off, the ILP can allocate those blocks however it wants. But enabling the single design option makes it impossible to allocate them across those four boards and would come up with a solution that requires seven boards. This is clearly inefficient and is solved by relaxing the constraint on the total number of blocks. Rather than requiring the total number of implemented 'A' blocks to sum to the total number of required blocks, the constraint requires that the number of implemented 'A' blocks is greater than or equal to the number of required blocks. In the example, it would allow the instantiated design to add another 'A' block and just put two blocks on each board. Relaxing this constraint can also inadvertently create extra blocks that aren't needed. The mapping fo the 16 antenna placement with both options enabled, listed below, overspecifes the number of XEng blocks to make all the GPU designs the same.

```
'ADC'_is_on_ROACH = 1.0
'PFB'_is_on_ROACH = 1.0
'Transpose'_is_on_ROACH = 1.0
```

```
'XEng'_is_on_GTX580 = 1.0
cost = 37300.0
num'ADC'_on_'ROACH'_0 = 1.0
num'ADC'_on_'ROACH'_1 = 1.0
num'ADC'_on_'ROACH'_2 = 1.0
num'ADC'_on_'ROACH'_3 = 1.0
num'PFB'_on_'ROACH'_0 = 1.0
num'PFB'_on_'ROACH'_1 = 1.0
num'PFB'_on_'ROACH'_2 = 1.0
num'PFB'_on_'ROACH'_3 = 1.0
num'Transpose'_on_'ROACH'_0 = 1.0
num'Transpose'_on_'ROACH'_1 = 1.0
num'Transpose'_on_'ROACH'_2 = 1.0
num'Transpose'_on_'ROACH'_3 = 1.0
num'XEng'_on_'GTX580'_0 = 50.0
num'XEng'_on_'GTX580'_1 = 50.0
num'XEng'_on_'GTX580'_2 = 50.0
total_GTX580 = 3.0
total_ROACH = 4.0
```

Table 6.2 and Table 6.3 show the entire design space optimized for dollars. The tests with both options enabled consistently require at least as many boards as the placement that did not preserve symmetry. In both cases, we see a quadratic scaling in the number of GPU servers required and a linear scaling in the number of ROACH boards indicating that the cross correlation step should be implemented on GPUs and the channelization should be on FPGAs.

We also note that enabling these options increases the execution time of the ILP. The execution time benchmarks were run on a server with two Quad-Core AMD Opteron 2376 Processors and 16GB of RAM. The increase in runtime when enabling either or both options is expected since these options directly affect the ILP structure by adding a number of constraints.

Since HERA is a planned instrument it is also interesting to see how this design will map onto newer technology and compare that to the planned design. To do this, I analyze the same design but I use the ROACH 2 board and a server that contains two GTX 690s costing $5,500 as target platforms. Rather than get new benchmarks for the GTX 690 and a ROACH 2, I use existing benchmarks to estimate the performance of these blocks on the new architecture. The FPGA benchmarks are obtained by assuming the number of resources required will be the same on the ROACH 1 and the ROACH 2. These resource benchmarks are divided by the amount of available resources on the ROACH 2 to get the percent utilization for each resource. The GTX 690 board has two Keplar GPUs while the GTX 680 only has one, so, for the server, we assume the performance of a server with two GTX 690 boards is four times the performance of a server with a single GTX 680. CASPER

Dual Pol Antennas	GTX 580 Servers	ROACH Boards	Price	ILP Execution time
16	3	2	\$23.9k	1.87 seconds
32	6	4	\$47.8k	4.44 seconds
64	11	8	\$92.1k	10.21 seconds
128	32	16	\$219.2k	72.85 seconds
256	121	31	\$631.2k	314.01 seconds
512	456	61	\$2004.7k	1231.26 seconds

Table 6.2: FX Correlator Design Space using ROACH boards and GTX 680 servers with Single Implementation and Single Design options disabled optimized for dollars

Dual Pol Antennas	GTX 580 Servers	ROACH Boards	Price	ILP Execution time
16	3	2	\$23.9k	2.14 seconds
32	6	4	\$47.8k	4.81 seconds
64	11	8	\$92.1k	11.35 seconds
128	32	16	\$219.2k	43.24 seconds
256	121	32	\$637.9k	281.18 seconds
512	456	64	\$2024.8k	1880.38 seconds

Table 6.3: FX Correlator Design Space using ROACH boards and GTX 680 servers with Single Implementation and Single Design options enabled optimized for dollars

Memo 48 [22] has performance data for the cross correlation block on a GTX 680, so the utilization is reduced by a factor of four to estimate the cross correlation performance on the target server. The FIR and FFT performance is estimated by using the resource utilization for the GTX 580 and reducing it by a factor of four. This provides a reasonable estimate, since the GTX 680 may provide a performance increase over the GTX 580, but we don't expect a significant increase without reoptimizing the code.

Figure 6.4 shows the performance data for the same design on the newer platforms with single design and single implementation enabled The smaller correlators don't see a significant price benefit from the added resources, likely because of bandwidth limitations, but the larger correlators are able to take advantage of the additional resources and the 512 antenna correlator gets a cost reduction of over 50% simply by switching to the next generation technology.

Dual Pol Antennas	Dual GTX 690 Servers	ROACH 2 Boards	Price	ILP Execution time
16	2	1	$21.5k	1.88 seconds
32	3	2	$37.5k	3.90 seconds
64	6	4	$75.0k	13.51 seconds
128	11	8	$144.5k	33.37 seconds
256	26	16	$311.0k	141.38 seconds
512	94	32	$853.0k	550.69 seconds

Table 6.4: FX Correlator Design Space using ROACH 2 boards and Dual GTX 690 servers with Single Implementation and Single Design options enabled optimized for dollars

Chapter 7

Conclusions

This dissertation presents an end to end solution that allows domain experts to take a high level idea and and translate it into a design indicating what types of hardware should be used to implement the instrument. The ORCAS tool is able to generate these designs quickly, making it a useful tool to explore the design space of different astronomy algorithms. I demonstrate three types of instrument design, and explore how changing the parameters affects the implementation and costs of these instruments.

In the results, it is clear that this approach is much quicker than designing instruments by hand. The ORCAS flow is able to map small designs in a few seconds and larger designs still take less than an hour to map. Coupled with optimization techniques that reduce the number of blocks that need to be placed, ORCAS provides a scalable solution for mapping large instruments. By allowing the user to leverage existing benchmarks, ORCAS reduces a traditional design cycle of a least a week down to an hour. Furthermore, ORCAS provides a better design experience than the traditional approach. The quick feedback is a key strength of this work, allowing the astronomer to vary the parameters of the algorithm and see how it affects the total cost.

Even when a benchmark isn't available, ORCAS makes it easy to add an estimated benchmark based on an existing one. To get more accurate results, getting new data for a single benchmark is very fast. The GPU benchmarks can be run in a few minutes and an FPGA benchmark for a small block can get results in under an hour. So even if new benchmark data is needed, the entire design cycle will still take less than a day.

7.1 Future Work

The success of this work opens up a lot of related projects to improve upon the existing mapping capabilities. While the instruments developed with ORCAS are designed for radio astronomy, ORCAS was developed as a general purpose tool and the expansion of the instrument set into other fields would provide interesting new challenges. The existence of a DSP library and benchmarks would make it easy to transition to other DSP applications.

Since other applications might not need a full-crossbar network, it would also be useful to integrate network design into the ILP. In radio astronomy, it is reasonable to assume this network exists and ignore the cost when designing instruments. The toolflow currently allows the user to minimize the number of ports required but this still assumes the presence of a full-crossbar interconnect. In applications with different network architectures, the costs can significantly change with the network design as well as the platforms used; future versions of this tool that are used for these applications will need to take those costs into account.

There are also improvements that can be made to the model, but they need to be balanced with the ILP to ensure fast runtimes. As telescope arrays get larger, a single computational block may need to span multiple boards. As we saw in the FX Correlator case study, this is going to be the case for the X-Engine in the near future. Simply supporting that capability would be useful, but it would be better if the model was able to determine how to split these blocks between boards.

Finally, the ILP aims to reduce cost in any way possible and often does so by cramming unrelated blocks onto the same platform. ORCAS does support an option that forces the tool to create exactly one design on each platform, but this does not prevent the tool from putting unrelated blocks in the same design. To make the resulting designs more straightforward, it would be useful if the ILP could preserve the problem structure when translating the design to hardware.

Bibliography

[1] M Bailes et al. "Transformation of a star into a planet in a millisecond pulsar binary." In: *Science* 333.6050 (Sept. 2011), pp. 1717–1720.

[2] John Bunton. *ALMA Memo 342 - An Improved FX Correlator*. Tech. rep. 342. CSIRO, Telecommunications and Industrial Physics, Australia, Dec. 2000.

[3] M A Clark, P C La Plante, and L J Greenhill. "Accelerating radio astronomy cross-correlation with graphics processing units". In: *International Journal of High Performance Computing Applications* 27.2 (May 2013).

[4] Abhijit Davare. "Automated Mapping for Heterogeneous Multiprocessor Embedded Systems ". PhD thesis. University of California, Berkeley, 2007.

[5] A T Deller et al. "DiFX: A Software Correlator for Very Long Baseline Interferometry Using Multiprocessor Computing Environments". In: *Publications of the Astronomical Society of the Pacific* 119.853 (Feb. 2007), pp. 318–336.

[6] P Demorest et al. "Gravitational Wave Astronomy Using Pulsars: Massive Black Hole Mergers & the Early Universe". In: *Astro2010: The Astronomy and Astrophysics Decadal Survey* 2010 (2009), p. 64.

[7] Douglas Michael Densmore et al. *Metro II Execution Semantics for Mapping*. Tech. rep. UCB/EECS-2008-16. Electrical Engineering and Computer Sciences University of California at Berkeley, Feb. 2008.

[8] Terry Filiba and Dan Werthimer. "Automatic Generation of Heterogeneous Spectrometers for Radio Astronomy". In: *General Assembly and Scientific Symposium, 2011 XXXth URSI* (2011), pp. 1–4.

[9] *GNU Linear Programming Kit*.

[10] Naga K Govindaraju et al. "High Performance Discrete Fourier Transforms on Graphics Processors". In: *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. Microsoft Corporation. Ieee, 2008, pp. 1–12.

[11] Lincoln J Greenhill and D Backer. "Hydrogen Epoch of Reionization Arrays". In: *American Astronomical Society Meeting Abstracts #217*. Jan. 2011, p. 107.10.

[12] Gurobi Optimization Inc. *Gurobi Optimizer Reference Manual*.

[13] Srinidhi Kestur, John D Davis, and Oliver Williams. "BLAS Comparison on FPGA, CPU and GPU". In: *Proceedings of the 2010 IEEE Annual Symposium on VLSI*. 2010, pp. 288–293.

[14] Thorsten Koch et al. "MIPLIB 2010". In: *Mathematical Programming Computation* 3.2 (June 2011), pp. 103–163.

[15] Hirofumi Kondo et al. "A Multi-GPU Spectrometer System for Real-Time Wide Bandwidth Radio Signal Analysis". In: *ISPA '10: Proceedings of the International Symposium on Parallel and Distributed Processing with Applications*. Sept. 2010, pp. 594–604.

[16] H Mittelmann. *Mixed Integer Linear Programming Benchmark*. URL: `http://plato.asu.edu/ftp/milpc.html`.

[17] Aaron Parsons et al. "A Scalable Correlator Architecture Based on Modular FPGA Hardware, Reuseable Gateware, and Data Packetization". In: *Publications of the Astronomical Society of the Pacific* 120.873 (Sept. 2008), pp. 1207–1221.

[18] Aaron Parsons et al. "PetaOp/Second FPGA Signal Processing for SETI and Radio Astronomy". In: *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on* (2006), pp. 2031–2035.

[19] Aaron R Parsons et al. "The Precision Array for Probing the Epoch of Reionization: 8 Station Results". In: *The Astronomical Journal* 139.4 (Mar. 2010).

[20] Rurik A Primiani, Jonathan Weintroub, and Jonathan deWerd. *SMA Wideband Correlator: Memo 1 Polyphase Filter-bank Utilization*. Tech. rep. 1.

[21] Scott M Ransom et al. "GUPPI: Green Bank Ultimate Pulsar Processing Instrument". In: *American Astronomical Society Meeting Abstracts #214* 214 (Dec. 2009).

[22] Christopher Schollar. *Profiling xGPU code on the GTX580 and GTX680*. Tech. rep. 48. Sept. 2012.

[23] A Siemion et al. "Current and Nascent SETI Instruments in the Radio and Optical: SERENDIP V.v, OSPOSH and HRSS". In: *Astrobiology Science Conference 2010: Evolution and Life: Surviving Catastrophes and Extremes on Earth and Beyond* 1538 (Apr. 2010), p. 5378.

[24] Hayden Kwok-Hay So. "BORPH: An Operating System for FPGA-based Reconfigurable Computers". PhD thesis. ProQuest, 2007.

[25] G Theodoridis, N Vassiliadis, and S Nikolaidis. "An integer linear programming model for mapping applications on hybrid systems". In: *IET Computers & Digital Techniques* 3.1 (2009), p. 33.

[26] A Richard Thompson, James M Moran, and George W Swenson. *Interferometry and Synthesis in Radio Astronomy*. 2nd ed. Wiley-VCH, 2001.

[27] Kuen Hung Tsoi and Wayne Luk. "Axel: A Heterogeneous Cluster with FPGAs and GPUs". In: *Proceedings of the 18th Annual ACM/SIGDA International symposium on Field programmable Gate Arrays*. Department of Computing Imperial College London, UK. 2010, pp. 115–124.

[28] Dan Werthimer et al. "The Berkeley SETI Program: SERENDIP III and IV Instrumentation". In: *Astronomical Society of the Pacific Conference Series* 74 (1995), p. 293.