LA-UR-07-3335

Approved for public release; distribution is unlimited.

Title:	Project summary for Coarse Grain Reconfigurable Array Technology
Author(s):	Jan Frigo, ISR-3, Paul Graham, ISR-3
Intended for:	LANL Engineering Institute Summer Program, June 11 - Aug 3, 2007

Project Summary for Coarse Grain Reconfigurable Array Technology

Jan Frigo, Paul Graham
Los Alamos National Laboratory

Abstract— **This project involves running some benchmark code on specialized hardware processors to determine the power/operations efficiency of the hardware. Two types of Coarse Grain Reconfigurable Array (CGRA) technology will be covered; a processor-oriented, specialized RISC processor, the Stretch S5000; and a medium-grain specialized hardware-oriented technology designed for the embedded, DSP markets, the MathStar FPOA chip. These technologies use a C-based language. We would select one of these technologies based on the student's interests and background. The benchmark algorithms we have collected from the signal and image processing domain include: an FIR filter, ffts, a matched filter, digital down converter, an adaptive beam former and some image processing algorithms. The project would involve profiling one or more of these codes to determine the power/ops for the application. Also, if time allows algorithm optimizations may be implemented to improve run-time performance. The time and effort to develop the application and the experiences using the development environment would also be reported.**

Keywords: reconfigurable computing, FPGA, Coarse-grain reconfigurable array, CGRA, SIMD, MIMD, SoC

I. Project Outline

The goal of the coarse-grained reconfigurable array effort is to evaluate new medium-grained processing technologies and recommend them for use in the proliferation detection domain. CGRA technology has a basic computational unit or arithmetic logic unit (ALU) that is a higher level of abstraction than an Field Programable Gate Array (FPGA). CGRA technology is optimized for a particular application domain to maximize the advantages of their specialized processing units. Through the use of massively parallel and specialized processing units, the CGRA technologies hope to gain high performance and an improved power per operation metric compared to conventional programmable logic. Currently, embedded systems, conventional reconfigurable computing (RCC) systems, and sensor-oriented systems utilize processor-based or hybrid processor-FPGA hardware systems to address a variety of data analysis, data mining, and real-time processing applications. These systems introduce a set of known limitations such as: the communication between the processor and the configurable hardware results in programming difficulties and communication overhead that can greatly reduce overall performance; placement and routing to hardware of a very large, complex algorithm may introduce considerable latency; the development environment tool chain is expensive, time consuming, and not user-friendly. CGRAs aim to combine computation units (processor(s), hardware ALUs or combinations of both) in one architecture so as to provide a simplified programming model, improve the processor-to-hardware communication bottleneck, shorten the overall application development time, and lower the power budget for high performance computations.

FPGA's are considered fine-grain reconfigurable devices because they are programmed (and reprogrammed) at the gate level with arbitrary bit precision and functions. Besides FPGAs as the fine-grain reconfigurable platform, the general reconfigurable computing platforms takes two forms: processor-centric cluster, or medium-grain hardware processing element array.

Processor-oriented clusters such as MIT's RAW, or Ambric's AM2000 family of reconfigurable processing arrays (RPAs), use some kind of RISC core and may or may not have application-specific, reconfigurable logic blocks. As the number of processors increase, the programmer-friendliness decreases as it approaches a highly parallel processor system. In addition, because of the potential irregular size and interconnection complexity, customizing a device or building additional devices may be considered to be a complete redesign. Ultimately large scale processor-oriented cluster systems could be time consuming to program, test and design, thus, development cost may be expensive.

Most medium-grain reconfigurable platforms (e.g., MathStar's FPOA, Velogix's VX200 chip) approach implementation with an ASIC design flow. For these systems, the customization process may be just as cumbersome as their processor-oriented equivalent, due to place and routing issues when mapping to hardware and the complexities of scaling to a new device. As a result, their operating frequency is generally less than 200 MHz which may be adequate, depending on the application domain.

This project covers processor-oriented technology and a medium-grain specialized hardware-oriented technology designed for the embedded, DSP markets. These technologies are C-based as are the benchmark algorithms we have collected. The project would involve profiling these codes to determine power/ops and *hot spots* or what functions or loops cause the algorithms to run slowly. If possible, optimizations to the code could be implemented to improve the run-time performance. The time and effort to develop an application would also be reported.

II. Benchmark Kernels

The benchmark algorithms selected for evaluation of the CGRA technologies are based on programmatic proliferation detection efforts, i.e., algorithms that are in use or have been used on recent projects in Los Alamos National Laboratory's (LANL) International Space and Response (ISR) Division. The signal processing kernels include: a poly-phase filter (ppf), a 4k FFT (16-bit), and an adaptive beamformer algorithm. The FIR filters and FFTs are commonly used DSP operations that have optimized FPGA implementations for comparison. The adaptive beamformer algorithm has components such as a covariance matrix inversion (LU decomposition) which are difficult to implement on FPGAs since the computation has division operations and must be converted to fixed point arithmetic from a floating point model - computing in floating point is very costly on an FPGA.

The image processing kernels selected include: kmeans clustering, pixel purity index (PPI), and a matched filter algorithm. These were chosen because they contain operations such as the dot product and convolution that need a large amount of local memory accesses and parallel computations. We wanted to choose a mix of algorithms - those that are typically optimized for FPGAs and DSP processors and some that are challenging to compute on conventional programming logic devices.

The source code for these benchmark algorithms is written in C/C++, RTL VHDL and/or Matlab/Simulink.

A. Poly-phase Filter Bank

In the field of signal detection, multi-rate filter banks have been employed to help detect RF signals in noisy environments. By decomposing a signal into various frequency subbands, filter banks enhance many algorithms because they make it easier to identify pertinent material on a band by band basis. The polyphase implementation[1] is a multi-rate filter structure combined with a Fast Fourier Transform (FFT) designed to extract subbands from an input signal[1]. The polyphase filter portion of the structure is based on a prototype baseband low-pass Finite Impulse Response (FIR) filter with symmetric coefficients, i.e., the first *n*/2 and the last *n*/2 coefficients are the same, albeit in reverse order. The remaining filters of the filter bank are frequency shifted versions of the prototype. The symmetry of this prototype filter combined with the structured frequency shifts allows for an optimal implementation of the filter bank. First, a prototype low-pass FIR filter, *h0[n]*, with the desired filter parameters is designed. The polyphase filters, *pk[n]*, are expressed in terms of the prototype filter,

pk[n] = h0[k + lM] k = 0..*M*-1, $l = 0..L-1$

n is the length of the FIR prototype, *M* is the number of polyphase filters, L is the length of the individual polyphase filters, (L = *n/M* = 4). The FFT is used following the polyphase filtering structure to provide the frequency shifts for the various channels.

B. 4k FFT

A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse.

[1]The filter structure was developed in collaboration with Prof. John Villesenor's team at UCLA.

FFTs are of great importance to a wide variety of applications such as:

- 64 pt FFT, radix = 2 or 4 - match band for ionosphere
- 512 pt FFT, radix = 2 or 4 - typical satellite time link
- 1K or 4K FFT, radix = 2 or 4 - commonly used benchmarks in the literature

C. Adaptive Beamformer

Adaptive beamforming has important applications to distributed sensor networks, in particular to energy constrained ad hoc networks. Beamforming can be applied to the sensor networks, for example, to estimate the direction of arrival of a low power signal in a noisy environment; also it can be applied to network communications for data exfiltration using collaborative coherent transmissions as a phased array thus saving power. These well-known techniques are computationally intensive and real-time. There is a multiplicity of algorithms for beamforming generally using matrix inversions, FFTs or other weighted sum type operations. In general these algorithms lend themselves to fixed precision highly parallel computations ideally performed by FPGAs but one common operation, inversion of a covariance matrix, generally works best if implemented in floating point. Prior work by [3] sponsored by NA-22 has explored the feasibility of an all-FPGA implementation of an adaptive beamformer using a Bayesian method. In this application domain, CGRA technology with floating point capability may have an advantage compared to FPGAs.

We investigate an inverse covariance matrix computation, a floating point LU decomposition shown below in Equations 1 and 2, that is difficult to implement on an FPGA since it requires floating point arithmetic for best results.

$$\beta_{ij} = a_{ij} - \Sigma_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \tag{1}$$

$$\alpha_{ij} = 1/\beta_{jj}(a_{ij} - \Sigma_{k=1}^{j-1} \alpha_{ik}\beta_{kj}) \tag{2}$$

D. K-means Clustering Algorithm

K-means clustering is an unsupervised clustering algorithm that is a popular data mining technique. The basic principle of the image clustering process is to take an original image and to represent the same image using only a small number of pixel values[6]. The K-means clustering algorithm performs this task by attempting to minimize a cost function (the absolute value of a difference) over a set of cluster centers. First, the algorithm assigns pixels randomly to the classes, computes the centers of the classes. There is a outer loop for a number of iterations, N, which can be either fixed in advance or undetermined, and an inner loop that scans all the pixels. For each pixel we check if it still belongs to its assigned class. If not, the pixel is moved to another class and the two centers, corresponding to both the new and the old classes, are updated. The number of pixels in a class is stored as well as the sum accumulation necessary for recomputing the class centers. The class centers are periodically updated every block of B pixels.

The computation can roughly be split into three parts: the distance calculation between a pixel and a class center, the accumulator update and the center update. The most time consuming part of the algorithm is the distance computation between the pixels and the class centers, even if the class center is frequently updated. The accumulator and the class center updates represent only a small percentage of the total computation time, especially for a partition into a large number of classes. For example, for a class partition of 32, the distance computation represents more than 99.6 % of the computation time.

This iterative algorithm has the following steps:

1) Randomly assigns each data element A[i] to one of k classes
2) Compute the centers of the classes
3) Loop over the data set A
    a) Let C = class of A[i]
    b) Determine the class number K which has the minimum distance to pixel i
    c) Store class number K to NewClassIndex[i]
4) Loop over the data set A
    a) if (C = class of A[i]) is not equal to class (K = NewClassIndex[i]) then move pixel i from class C to class K and re-compute the centers of classes K and C

FPGA implementations [4] focus only on parallelizing the most time consuming part, that is the distance computation between the pixels and the class centers. A pixel stream flows through a linear array of processors. The number of processors is equal to the number of classes. A processor *k* computes a distance between the class *k* and the current flowing pixel. The result is taken at the rightmost end of the array by the filter process and corresponds to the index class for which a minimum distance has been found.

E. Pixel Purity Index

The Pixel Purity Index (PPI) is an algorithm employed in remote sensing for analyzing hyperspectral images. Particularly for low-resolution imagery, a single pixel usually covers several different materials, and its observed spectrum is (to a good approximation) a linear combination of a few *pure* spectral shapes. The PPI algorithm tries to identify these pure spectra by assigning a pixel purity index to each pixel in the image; the spectra for those pixels with a high index value are candidates for basis elements in the image decomposition.

The algorithm proceeds by generating a large number of random D-dimensional vectors, called skewers, through the hyperspectral image. For each skewer, every data point is projected onto the skewer, and the position along the skewer is noted. The data points that correspond to extrema in the direction of a skewer are identified and placed on a list. As more skewers are generated, this list grows. The number of times a given pixel is placed on this list is also tallied. The pixels with the highest tallies are considered the most pure, and the pixel's count provides its pixel purity index.

Most of the execution time of the PPI algorithm is spent in computing dot-products between the pixels and the skewers. These dot-product are highly independent and could be done simultaneously. There are many ways to parallelize the algorithm, one such approach [5] targets the limited resources available on FPGA boards. A sequential version of the Pixel Purity Index algorithm follows:

```
  PIXELS[N][D]; // an image of N hyperpixels
  SKEWER[K][D]; // a set of K random skewers
  PPI[N];       // the PPI result

// reset pixel purity index
  for (n=0; n < N; n++) PPI[n]=0;
  for (k=0; k < K; k++)  // K skewers
   {
     dpmax=MIN_INT; dpmin=MAX_INT;
     for (n=0; n < N; n++) // N pixels
      {
        // compute a Dot-Product
        dp = 0;
        for (d=0; d < D; d++)
           dp = dp + SKEWERS[k][d]*PIXELS[n][d];
        // detect extrema
        if (dp > dpmax) { imax=n; dpmax=dp; }
        if (dp < dpmin) { imin=n; dpmin=dp; }
      }

     // update PPI
     PPI[imax]++;
     PPI[imin]++;
   }
```

For each skewer, N dot-products are computed to determine the two pixels which produce the largest and the smallest dot-product. The pixel index (`PPI` vector) is modified accordingly. A pixel `n` is a candidate to be a pure pixel if `PPI[n]` has a high value.

From the above description it can easily be seen that all the dot-products can be computed independently: there are no dependencies between them. The parallelization takes advantage of this by computing $KS \times NS$ dot-products simultaneously, where KS and NS represent the number of skewers and pixels that can be processed in parallel.

F. Matched Filter

The matched filter is a well-known technique for detecting a signal in the presence of known forms of "clutter." By filtering with respect to pre-determined background signatures, weak signals may be recovered that might otherwise have been lost in the background. Matched filters are used in many application domains, of particular interest to us are multi-spectral image processing, RF signal detection and pulse compression. In the case of multi- or hyper-spectral image processing, the matched filter is well-suited to automatic detection of signals within image cubes containing tens to hundreds of spectral channels. The purpose of a matched filter is to match an image pixel's spectral signature against a pre-determined "background" signature. When analyzing multi- or hyper-spectral imagery with complex background clutter for small or weak targets, filtering a target image through a matched filter suppresses background spectra and thus increases response to non-background features. A bank of matched filters may be created to filter out a variety of “clutter” effects [2].

The matched filter is implemented by a convolution function. A convolution is an integral that expresses the amount of overlap of one function as it is shifted over another function. It therefore "blends" one function with another. For example, in synthesis imaging, the measured dirty map is a convolution of the "true" clean map with the dirty beam (the Fourier transform of the sampling distribution).

III. Stretch S5

Stretch Inc. was founded in March 2002. Stretch provides software-configurable processors for compute-intensive applications and standard C/C++ programming tools. The systems can address applications in the telecom, networking, video, medical markets and support evolving standards such as H.264 video encoding and 802.16-2004 wireless standards. Stretch offers development boards for the video, wireless, and biometric application domains.

A. Architecture

Stretch uses a single specialized, high performance RISC processor core, the 300-MHz Xtensa core with 16- and 24-bit instructions as shown in Figure 1. The core supports a memory managed unit (MMU) with a translation look-aside buffer(TLB) which translates virtual pages to physical pages. The external and embedded memory specifications are:

- 64-bit DDR400 SDRAM
- 256-KB SRAM
- 32-KB Data RAM
- 32-KB Data Cache
- 32-KB Instruction Cache

The I/O support on the Stretch S5 chip is as follows:

- 1 32-bit, 66MHz PCI port
- 4 programmable parallel ports
- 2 Time Division Multiplexed (TDM) ports
- 1 Generic Interface Bus (GIB)
- 2 programmable serial ports
- 2 UART ports with IrDA
- 1 General Purpose I/O (GPIO) and Interrupts
- 1 standard test port - JTAG (IEEE 1149.1)

B. Development Environment

The Integrated Development Environment (IDE) tools suite (shown in Figure 2) is a graphical interface consisting of a compiler, debugger, assembler, profiler, linker and editor. Stretch’s C/C++ compiler programs the processor and automatically configures the Instruction-Set Extension Fabric (ISEF) with application-specific instructions. A debugger and profiler in the IDE tools suite provide verification and analysis capability. 32-bit fixed point data types are supported.

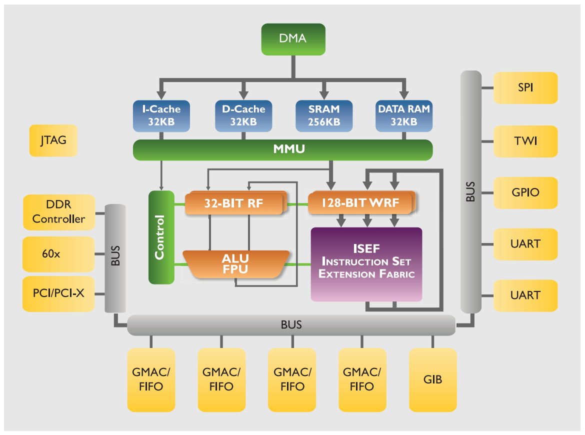

Fig. 1. Stretch S5620 Processor architecture

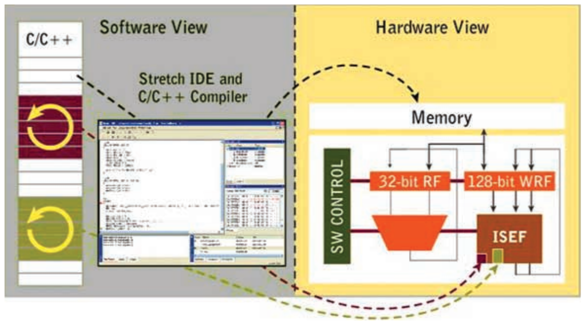

Fig. 2. Stretch Software Development Environment

C. Application Domain

The Stretch processors target the embedded market and are suited for compute-intensive algorithms. Embedded software engineers can create optimized processors using an off-the-shelf chip and their C/C++ application. C-functions can be parallelized into new instructions that execute in a single cycle. The specialized compiler technology automatically converts selected C functions into programmable logic. Most importantly, because the uniqueness of the instruction set comes directly from the application code, the processor can accelerate compute-intensive application in many markets including, but not limited to:

- High-end consumer audio-video (video conferencing, media gateways, digital TV, broadcast equipment)
- Office imaging (scanners, printers)
- Wireless communications (base stations, satellite receivers)
- Medical imaging (CAT scans, ultrasound)
- Industrial imaging (e.g., robot/machine vision)
- Video
- Wireless
- Biometrics

D. Performance

See Section **??** and Section **??** for performance results. The development boards have a pin-out for power so it can be measured per application.

The bottle-necks to algorithm performance, hardware capacity or utilization have yet to be determined. However, one limitation is the fixed bandwidth to the ISEF - there are 2 128-bit input ports and 1 128-bit output port.

IV. MathStar FPOA

MathStar designs and develops ultra-high performance semiconductors for Digital Signal Processing (DSP) and filtering applications. MathStar was founded by communications industry veteran Douglas M. Pihl and is a development stage company headquartered in the Minneapolis, MN metropolitan area. In March 2005, MathStar was selected by Honeywell to provide Field Programmable Object Array (FPOA) technology that will be incorporated into Honeywell semiconductors for satellites in military space systems.

A. Architecture

The MathStar FPOA chip shown in Figure 3 has 400 Silicon Objects - 256 ALUs, 64 Multiply Accumulates (MACs), and 80 Register Files (RFs). It has a bi-directional 800-MHz DDR 16-bit LVDS port and features differential clock inputs as well as sync inputs. The LVDS port can also be configured as a bi-directional 32-bit 320-MHz HSTL port. The device also has four sets of 44-pin 100 MHz LVCMOS General Purpose I/Os (GPIO), operating either synchronously or asynchronously. Finally, it has twelve 500-MHz 768x76-bit internal memory blocks and two 250-MHz 36-bit DDR (72-bits per cycle) RLDRAM or DDRII SRAM controllers for external memory accesses. There are more FPOA devices in development to serve a broader range of applications.

The FPOA can communicate information with external processors or hosts through a bridge to LVDS and HSTL ports. Local bus interface modules to the PowerQUICC processor and PLX-to-PCI bridges are implemented.

The FPOA Silicon Blocks can have a semi-autonomous nature. For example, each ALU Silicon Object has a program memory of eight instructions that contain both operation and communication directions. The control path is bit-wise granular and guides program execution while data is moved and operated upon via the 16-bit data path. (Multiple objects can be combined to create wider data paths.) Thus, instructions are the mechanisms that tie the independent control and data paths together within the array.

The instructions are loaded at power up and can be reconfigured by the host system. Intelligent scheduling and routing tools deterministically allocate instructions to each object before run-time. This works in conjunction with the MathStar's Party Line communication scheme which permits high speed sharing of data throughout the device (and externally).

B. Development Environment

The MathStar design flow is given in Figure 4. Algorithms must be converted to the Visual Elite/SystemC programming language for simulation and test. The environment supports clock accurate simulation and verification (with the Riveria simulator).

32- and 64-bit integer data types can be handled by cascading processor modules together. The 1-GHz speed is not

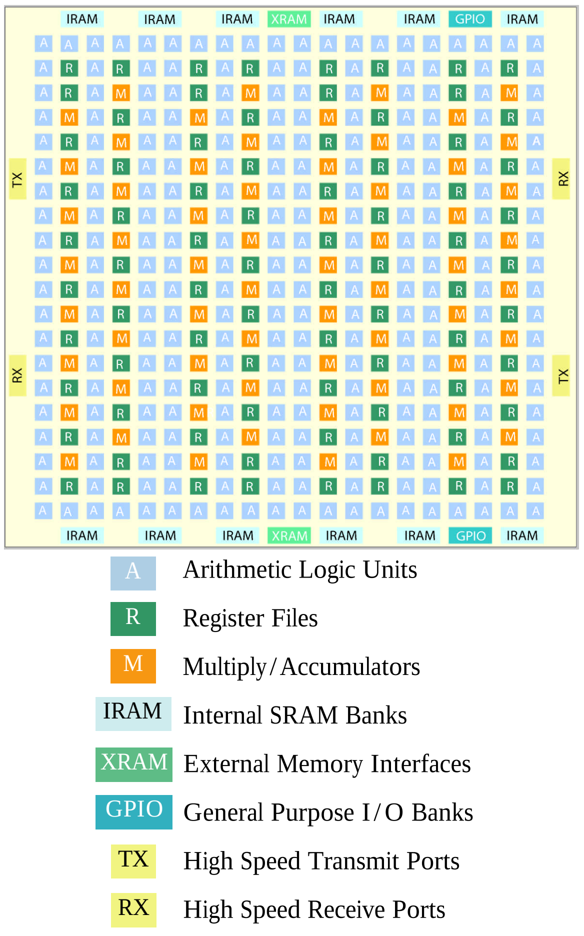

Fig. 3. FPOA architecture

maintained in this configuration. Floating point and double precision types can be emulated.

C. Application Domain

The FPOA is suited for higher performance DSP applications like high-sample-rate complex Fast-Fourier Transforms (FFTs) and Finite Impulse Response (FIR) filters. MathStar is targeting the commercial and military/aerospace DSP market. Their prototype boards can cascade into larger systems through the LVDS and HSTL high speed interfaces. No large-scale systems have been built to-date.

D. Performance

The FPOA devices have 400 1-GHz silicon objects, resulting in a device with 400 gigaoperations/second of performance. The MathStar FPOA claims to deliver 2-4 times the performance of an FPGA while retaining the flexibility of a programmable device.

Power consumption is highly application dependent. The SOA13D40-01 FPOA with 400 silicon objects running at 1-GHz with 100% utilization can consume approximately 10 to 30 Watts of power. However, power consumption depends on many factors such as the run-time activity, operating frequency, voltage supply, temperature conditions, process variations, etc. These numbers should be considered estimates.

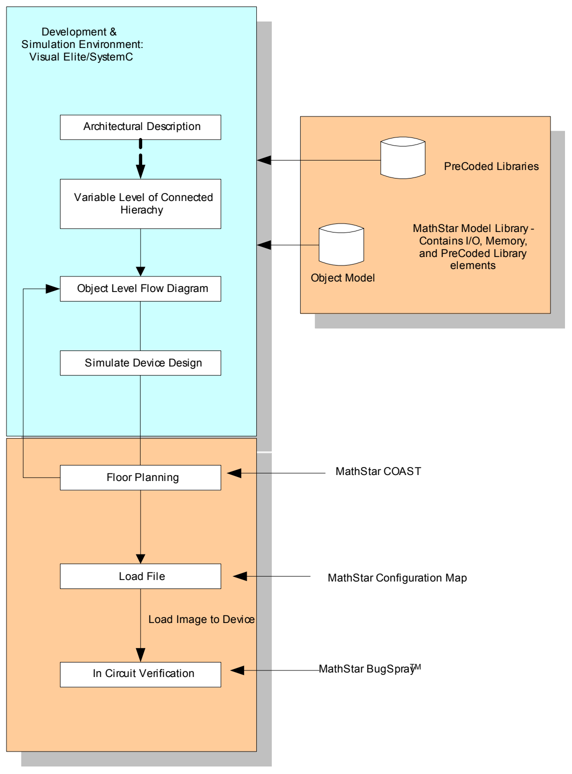
Development & Simulation Environment: Visual Elite/SystemC
Architectural Description
Variable Level of Connected Hierachy
Object Level Flow Diagram
Simulate Device Design
PreCoded Libraries
Object Model
MathStar Model Library - Contains I/O, Memory, and PreCoded Library elements
Floor Planning
MathStar COAST
Load File
MathStar Configuration Map
Load Image to Device
In Circuit Verification
MathStar BugSpray™

Fig. 4. MathStar FPOA Design Flow

One bottle-neck to algorithm performance is the 16-bit processor array. Larger data sizes require cascaded processors and, thus, the 1-GHz speed cannot be maintained.

V. 8 Week Plan

- Week 1-2 Become familiar with the tools and development environment for the Stretch S5000 processor and/or the MathStar FPOA chip.
- Week 3 Try a simple FIR filter example, profile code and write an optimization. Record the run-time improvement.
- Week 4 Measure the power for the FIR filter example and record time and effort it took to use the development environment.
- Week 5-7 Try an image processing algorithm and repeat steps from Week 3-4.
- Week 8 Write up report with results and make a presentation.

VI. Structural Dynamics Application

The structural dynamics motivation for this project comes from the area of machining of metal parts. For certain applications, typically involving high-precision components, it is desirable to detect the variability in material properties of the workpiece material. Such properties of interest may include grain size or hardness. However, standard testing procedures for these properties are destructive in nature and can therefore not be applied to a finished component. An alternative approach relies on the observation that machining can be viewed as a high strain rate, high temperature material test. If various in-process (i.e., during machining) measurements are made (e.g., acoustic, vibration, video), it seems reasonable that a correlation could be developed between these measured quantities and the material properties of interest. This would then allow one to use the machining process itself as a diagnostic for material property variability. Recent work at LANL has attempted to do just that. In particular, high speed video (4000 frames/sec) was recorded for hundreds of cuts (yielding about 250 GB of video) involving workpieces with several different hardnesses (as measured on the Rockwell C scale). Two successive frames of a clip of this video are shown in Figure 5. After preliminary analysis of the video streams by humans, it seemed that the velocity of the cut chip was closely correlated with workpiece hardness. However, there are many challenges in extracting chip velocity from the video. For instance, due to reflection from the chip, lighting can vary significantly. Additionally, "clutter" (i.e., metal dust) can build up on the tool. Finally, because the chip formed is typically continuous, it can "flop" around and occasionally block or at least blur the field of view. To address these challenges a sequence of image processing algorithms were applied. These algorithms include high pass filtering, interframe differencing, and registration. While these algorithms are relatively easy to implement, they can take quite some time to run on 250 GB of data and on a standard PC. Thus, in order to have a real-time chip velocity measurement tool, it is desirable to implement these algorithms on special purpose, high speed processors, which is the goal of this project.

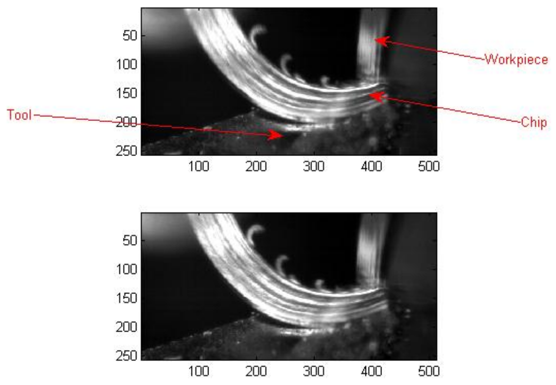

Fig. 5. Video clips of a cut

VII. Equipment Requirement

Workstation running Windows XP and Stretch and/or MathStar development environment tools. We currently have the hardware and software tools for this project. The tools include power planners or pin-outs on the hardware to determine the power/ops calculation.

References

[1] Joseph Arrowood. Comparison of filter banks for signal detection. In *LAUR Number 99-4551*, Los Alamos, NM, March 2000.
[2] Jeff Bloch. rcc.lanl.gov/imaging/index.php. 1999.

[3] Scott D. Briles, Joseph Arrowood, Thierry Cases, Dakx Turcotte, and Etienne Fiset. Real-time implementation of an adaptive bayesian beam-former. *13th IEEE Workshop on Statistical Signal Processing*, July 2005.

[4] Dominique Lavenier. FPGA Implementation of the K-Means Clustering Algorithm for Hyperspectral Ima ges. *Los Alamos National Laboratory LAUR 00-3079*, 2000.

[5] Dominique Lavenier, James Theiler, John Szymanski, Maya Gokhale, and Janette Frigo. Fpga implementation of the pixel purity index algorithm. In *SPIE, FPGAs and Reconfigurable Processors for Computing and Applications, vol 4212*, Boston, MA, November 2000.

[6] Miriam Leeser. Applying reconfigurable hardware to segmentation for multispect ral imagery. In *HPEC 2000*, Boston, MA, September 2000.