A FORMAL METHODOLOGY FOR ESTIMATING THE FEASIBLE PROCESSOR SOLUTION SPACE FOR A SOFTWARE RADIO

James "Jody" Neel (Virginia Tech, Blacksburg, VA; janeel@vt.edu);
Pablo "Max" Robert (Virginia Tech, Blacksburg, VA); and
Jeffrey H. Reed (Virginia Tech, Blacksburg, VA)

ABSTRACT

Selecting the best choice of processing resources is a problem faced in the course of developing any software radio or multi-waveform platform. This problem can be viewed as one of identifying the optimal design, typically with reference to some function of cost, power, and area, from the set of feasible processing solutions. Fortunately, optimization theory provides the designer with numerous tools and algorithms that can be used to quickly search even the largest solution space. However, before applying these optimization algorithms, the feasible solution space must be identified - an onerous process in light of the large number of µPs, DSPs, and FPGAs available for use and the even larger number of ways that these resources can be combined together. This paper presents a methodology for estimating the elements of a multiple-processor multiple-waveform feasible solution space.

1. INTRODUCTION

Selecting the processing elements is arguably the most critical step in the design of a software radio. In the same manner that PC applications are limited by the choice of the PC's processor, a software radio's waveforms are limited by the choice of processing resources. Ideally, a designer would like to choose the processing elements in a way the supports all anticipated waveforms with minimal cost, power, and area. For PC applications this generally is only a matter of choosing a processor with the lowest clock rate that allows implementation of all desired applications.

For software radio applications, microprocessors (µPs) rarely provide sufficient performance to support physical layer waveforms. Instead DSPs and FPGAs must be used. Because DSPs and FPGAs exhibit significant architectural variation between processors, a simple examination of a chip's clock rate is insufficient to estimate performance. In the interest of using a single number for comparing processors, BDTI has developed a metric based on performance measures of suites of algorithms that can be implemented on DSPs [1] and FPGAs [2]. While this is an excellent tool for making general comparisons between processors, it is not generally predictive of whether a particular processor can support a particular waveform because of the variety of specialized circuitry included with DSPs and FPGAs, e.g., Viterbi co-processors [3], and two-cycle butterfly units [4].

Further, many radios make use of multiple and heterogeneous processors, e.g., a DSP with a FPGA, and split waveforms across the chips. So finding the best processor solution implies also solving for both the best waveform partition, significantly complicating this optimization problem. Repeating this process for several waveforms can be quite the daunting problem.

Formalizing this optimization problem, suppose a software radio must support a set of waveforms, Ω, while simultaneously satisfying constraints on power consumption, P, area, A, and cost, C. The problem is to identify the combination of µPs, DSPs, and FPGAs and waveform partitioning that satisfies these constraints while maximizing some design function. The combinations of µPs, DSPs, and FPGAs and the associated waveform partitions that simultaneously satisfy all of these conditions form the *feasible processor solution space*, $F(\Omega, P, A, C)$. On this set, any of a number of optimization routines can then be used to find the best choice of processors and waveform partitions. This paper focuses on the first aspect of this problem – identifying $F(\Omega, P, A, C)$ – and presents a methodology that can be applied to solve for $F(\Omega, P, A, C)$.

The remainder of this paper is organized as follows. Section 2 presents a high level view of the proposed methodology. Section 3 presents details of how waveforms are broken into processing components. Sections 4 and 5 provide details on how this methodology handles DSPs and FPGAs, respectively. Section 6 describes how the details from the preceding sections are incorporated into an identification of $F(\Omega, P, A, C)$.

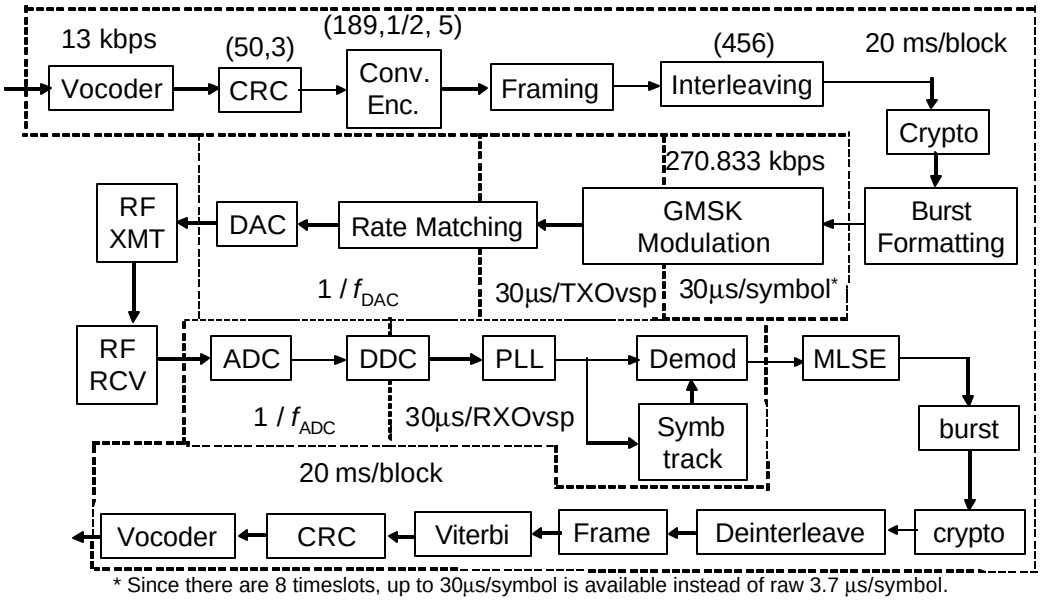

Figure 1 Componentization of a GSM Transceiver

2. METHODOLOGY OVERVIEW

Given the set of waveforms, Ω, to support, this paper proposes the following methodology to estimate the feasible processor solution space $F(\Omega, P, A, C)$.

1. Identify an initial pool of candidate processors, *F'*, containing µPs, DSPs, and FPGAs.
2. Survey the required waveforms, Ω, to identify the required waveform components and component execution times.
3. Estimate resource utilization for each component as implemented on each processor in *F*'.
4. Using the results of step 3, form the set of processor solutions from examining all possible processor and waveform partition combinations.
5. Form the feasible processor solution space, $F(\Omega, P, A, C)$, by removing from *F* all combinations of processors that violate constraints *P*, *A*, *C* and/or have insufficient resources to support Ω.

The following sections detail how each of these steps can be accomplished.

3. COMPONENT IDENTIFICATION

In this step, the waveforms in Ω are broken into their constituent processing elements (components) and the timing requirements for these processing elements are identified. While there exist many different ways to partition a waveform that yield different components (e.g., modulation and filtering may be treated as two sequential, but separate, components or combined into a single component), one logical breakdown is by clock domains and processing blocks. Considering an arbitrary waveform, it is expected that some portions of the processing will occur at the sample rate, some at the symbol rate, some at the frame rate, and perhaps some other portions at waveform/implementation specific rates. Within many of these clock domains, waveform processes operate on blocks of data, e.g., interleaving and error correction, rather than on continuous streams of data, thus introducing other logical boundaries for componentization. In addition to this componentization, multirate components that span clock domains are also encountered and further componentization beyond clock boundaries and block interfaces is also possible.

With this in mind, a waveform's processes are first separated by clock domain and then subdivided by processing blocks. Each of these subdivisions would then be deemed a waveform component. Further subdivisions may be useful, but the choice of subdivisions and utility of these subdivisions will be a function of the waveforms in Ω. This approach also permits an immediate rough identification of timing requirements for each component.

For instance, consider the componentization of a GSM physical layer transceiver waveform shown in Figure 1. In this componentization there are six distinct clock domains, each determined by a specific component in the domain. The vocoder processes blocks of data at a rate of 20ms/block. Supporting GSM modulation requires a minimal symbol processing rate of 30 µs/symbol. The modulation and demodulation/synchronization processes are frequently oversampled, specifying two additional clock domains. The sampling rates of the ADC and DAC specify two other clock domains (assuming that the ADC and DAC do not have the same sampling rate and are not equal to the modulation

rates). Within these clock domains, further componentization is primarily determined by logical block processing boundaries, the exception being the componentization indicated for the demodulation operation which has arbitrarily been divided into components for phase recovery, symbol timing recovery, and symbol demodulation. Each component must then operate at least as fast as the rate indicated for that clock domain.

4. ESTIMATING COMPONENT RESOURCE UTILIZATION ON PROCESSORS

The optimization problem necessitates estimates of the processing resources, power, and area needed to implement each waveform component. Of the three parameters, estimating processing resources is the least straight-forward and demands the most attention.

4.1 Estimating DSP Processing Resources

Motivated by the fact that even though different processors include different circuits that impact cycle counts in different ways, all processors must perform the same operations to implement the same component, the proposed methodology uses the following steps to estimate DSP processing resources:

1. Predict the number and type of operations required to support the component.
2. Form a cycle estimate by subtracting from the operations prediction the cycle modifiers identified from a review of the chip's architecture.

This approach has the advantage of being able to quickly generate cycle estimates for a component for a large number of different processors armed only with the knowledge of the architecture of each processor. Further this approach naturally lends itself to computer implementations, valuable in light of the size of $F(\Omega, P, A, C)$.

4.1.1 Operations Prediction

To predict the operations required the waveform component is first described in pseudo-code with an eye on capturing all relevant operations, including memory accesses, loop control operations, and pipeline filling and flushing. The total number of operations indicated by the pseudo-code are then tabulated and parameterized (e.g., in terms of block size, radix, or constraint length) to give a raw estimate of the total operations required to support the waveform component. Operations related to loop control and other specialized subprocesses are noted and separated for reasons that will be made clear in Section 4.1.2.

As an example of this process, Table 1 shows the tabularized operations for a FFT waveform component parameterized in terms of block size, N, and radix r and assuming precalculated twiddle factors. Note that in addition to the traditional computational operations associated with the FFT, all memory accesses required to support the computation are also included in the operations estimate. Also for reasons that will be made clear in Section 4.1.2, the operation estimate is subtotaled for butterfly operations, bit reversal shuffling, and loop control.

Table 1 Tabularized FFT Operations

Arithmetic (Butterfly) Operations	
Additions = (N/2) $\log_r N$ complex additions	4 $N \log_r N$
Multiplications = (N/2) $\log_r N$ complex multiplications	2 $N \log_r N$
Linear Memory Access (1 complex twiddle read, 2 complex data read, 2 complex data write)	10 $N \log_r N$
Bit Reversal Shuffling	
Linear Access (indices, read, write)	3 N
Control (Add, compare)	2 N+1
FFT Control	
Stage Loop	
Loop Control (Add, compare)	2 $\log_r N$
Other ALU operations	5 $\log_r N$
Butterfly Control Loop	
Loop Control (Add, compare)	2 (N-1)
Other ALU operations	3 (N-1)
Inner Butterfly Loop (executes $N \log_r N$ times)	
Loop Control (Add, compare)	2 (N-1) $\log_r N$
Other ALU operations	3 (N-1) $\log_r N$
Total Estimated Operations	(19 N +2) $\log_r N$ + 10N -4

Total estimated operations based on this pseudo-code process for several common waveform components are listed in Table 2.

4.1.2 Cycle Estimation

For a processor that can implement only a single operation at a time, the cycle estimate for a waveform component would simply be the operations estimate. However, most processors include extra circuitry for supporting simultaneous complex operations that would result in a cycle estimate different from the operation estimate. To estimate the cycles a processor uses to implement a particular waveform component, we subtract the operations performed in specialized circuitry indicated by the processor's architecture from the operation estimate. The following briefly describes the effect a number of several commonly encountered circuits have on the raw operation estimate.

Bit Reversal Addressing - The (I)FFT operation is a common and cycle intensive waveform component (particularly for OFDM waveforms). Bit reversal addressing eliminates operations associated with bit reversal shuffling.

Table 2 Estimated Operations for Common Waveform Components

Module	Parameters	Operations(Arithmetic, Logic, Multiplications, Memory)
(Real) Filtering	N = filter length	$6N + 3$
FFT	N = block length r = radix	$21N \log_r(N) + 10N - 4$
Correlation	N = block length	$6N + 3$
CRC	N = block length r = polynomial order	$5(N+r) + 1$
Convolutional Encoder	K = constraint length $1 / r$ = code rate N = block length	$[3(1+Kr) +2K+5](N+K)+K+1$
Viterbi Decoder	K = constraint length $1 / r$ = code rate N = block length	$3+3(2K-1) + (N-4K)(4+2K+1) + [10+2K(17+3r)](N+K)$
Interleaver	N = block length	$5N+3$
Interpolation/Decimation (CIC)	N = CIC stages R=I/D factor	$3(N+Nr)+1+r$
Transcendental (LUT)	N = iterations	1
Transcendental (CORDIC)	N =stages	$12N + 1$
Transcendental (Series)	N =block length	$5N + 1$
Equalizer (LMS complex)	N =block length	$30N + 16$

Butterfly circuits - A dedicated butterfly circuit typically permits the dedicated arithmetic operations for each butterfly operation to be replaced with the number of cycles used in a call to the butterfly circuit. Frequently this will eliminate the operations associated with computing the butterfly, but not the associated memory accesses.

Circular addressing - Circular addressing effects a modulo operation for addressing and in the limit of high order filters eliminates a comparison operation for each iteration of a filtering loop.

Error correction - Error correction, particularly Viterbi and Turbo decoding, are frequently the most significant bottleneck to implementing many waveforms. As such, several processors provide dedicated hardware in the form of a Viterbi decoder coprocessor or dedicated single-cycle Add-Compare-Select (ACS) circuitry. A single cycle ACS circuit replaces the operations associated with ACS operations with one cycle per ACS call. Co-processors completely eliminate the original error correction operations, but add cycles for interfacing with the co-processor.

MAC units - MAC units permit multiplication and accumulation with a single instruction and thus can be treated as eliminating the accumulation operations.

SIMD (Single Instruction Multiple Data) - A processor with SIMD capability can treat longer data words as a number of smaller data words. For example, one 32-bit word could be processed as two independent 16-bit words. Generally, SIMD has the effect of dividing block lengths (typically variable N in Table 2) by the SIMD factor (nominal word size/effective word size).

VLIW architecture - Some DSPs can execute multiple instructions at the same time on differing functional units. Somewhat loosely, we are referring to all architectures capable of supporting different instructions for different functional units as VLIW. A first cut approximation for a cycle estimate is to divide all operations by the number of simultaneous operations that can be supported. More accurate estimates can be made by noting the exact functional elements available and subtracting the operations that can be performed in parallel.

Zero-overhead loop control - Zero-overhead looping offloads loop control (conditional branching and counter decrementing) to a dedicated circuit. Frequently, this eliminates most of the cycles consumed in loop control operations. One cycle remains for each loop to load a special register.

It should be noted that many of these specialized circuits are simultaneously present and that the order that these modifiers are applied can impact the accuracy of the estimate.

4.2 Estimating Power Consumption

For the methodology, we need to associate a dynamic power consumption estimate with each component; static power consumption is addressed in a later step. For our purposes we assign the chip's high activity power consumption estimate from a vendor spreadsheet [5] to each component as while the component is processing, most of the resources will be in use.

5. ESTIMATING FPGA RESOURCE UTILIZATION

As was the case for estimating DSPs resource utilization, we need to associate utilized processing resources and dynamic power consumption with each waveform component.

5.1 Estimating FPGA Processing Resources

For FPGAs, we studied communications oriented products from Xilinx (Virtex) and Altera (Stratix) – Virtex II, Virtex II Pro, Virtex IV, and Stratix II. Both vendors provide extensive and well documented libraries of code (see [6] and [7]) that can be used to estimate resource utilization for many of the most common waveform components in terms of utilized processing fabric components, memory units, and embedded functional units (multipliers for Virtex products and DSP blocks, which support complex multiplication, for Stratix products).

However, some waveform components for our study were only available for one or the other architecture. This necessitated the development of a technique for converting resource estimates from one architecture to another. To convert Virtex multipliers to Stratix DSP blocks, we evaluated the ceiling of the number of Virtex multipliers divided by four.[1] We then treated memory one Virtex block SelectRAM (1K x 18 bits) blocks as 32 Stratix M512 RAM blocks, 8 M4K RAM blocks, and 1/32 of a M-RAM block (4Kx144 bits).

Conversions between different processing fabric elements can also be readily approximated. Virtex library implementations are specified in terms of Slices and Configurable Logic Blocks (CLBs) which contain four slices. Stratix implementations are specified in terms of Adaptive Logic Modules (ALMs), Logic Array Blocks (LABs) of 8 ALMs, and effective Logic Elements (LEs) which represent 2.5 ALMs. A conversion between the two architectures can be approximated by noting the rough equivalence of the basic computational elements – Slices and ALMs – which are shown in Figure 2 and Figure 3, respectively. A tabulation of these conversions is provided in Table 2.

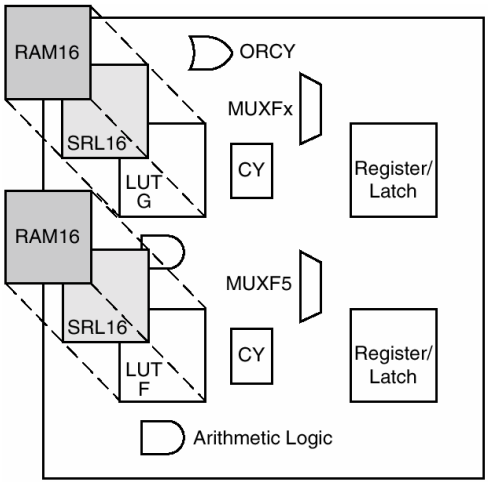

Figure 2 Slice from Figure 15 in [8]

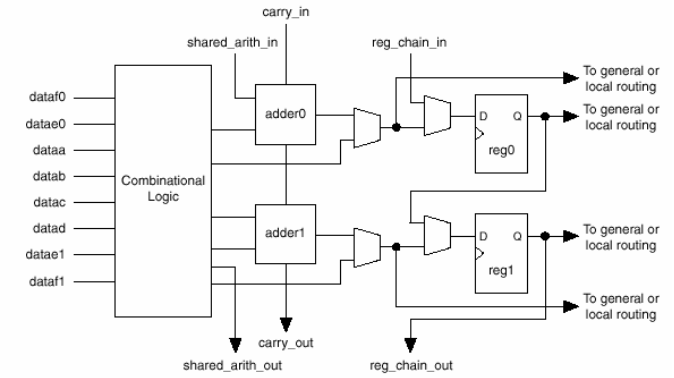

Figure 3 Adaptive Logic Module (ALM) from Figure 2.5 in [9].

Table 3 FPGA Resource Conversion Matrix

	CLB	Slice	LAB	ALM	LE
CLB	1	4	0.5	4	10
Slice	0.25	1	0.125	1	2
LAB	2	8	1	8	20
ALM	0.25	1	0.125	1	2.5
LE	0.1	0.4	0.05	0.4	1

4.2 Estimating FPGA Power Consumption

Both Xilinx and Altera provide tools for estimating power consumption based on resource utilization (see [10] and [11]) for each of their products. By using these tools dynamic power consumption estimates for each of the waveform components can be readily calculated. Examples of dynamic power consumption and processing resource estimates for selected 802.11a waveform components are listed in Table 4.

[1] This approximation is a little facile as Stratix DSP blocks operate in a number of different modes, none of which directly support the use of three multipliers.

Table 4 Parameterized Estimations of Resource Requirements for Selected Algorithms

802.11a Component	Virtex Slices	Virtex Multipliers	RAM Blocks	Virtex2 Power (mW)	Virtex4 Power (mW)	Stratix II Power (mW)
FFT	1331	9	8	1051	417	245
Coarse Timing Recovery	462	12	0	494	169	85
Fine Timing Recovery	1100	2	0	739	313	202
Channel Estimation	776	10	2	673	250	145
Channel Equalization	40	4	1	93	25	8
Viterbi Decoder	1084	0	2	701	304	202
DDC	805	0	2	528	228	150

6. FORMING THE FEASIBLE SOLUTION SPACE

For our problem, a solution is a choice of processors and a partitioning for each waveform in Ω. The feasible solution space is the set of all possible solutions that satisfy the design constraints. After completing the earlier steps indicated in this process, the feasible processor space can be formed by taking the following steps:

1. Identify the set of all combinations of processors that satisfy maximum area and cost constraints, $F(A, C)$.
2. Using the componentizations of the waveforms in Ω, identify all possible partitions of these waveforms across all elements of C, $F(\Omega, A, C)$.
3. Based on the identified timing constraints and the estimated processing resources, eliminate the elements of $F(\Omega, A, C)$ that cannot support the implied resource allocation (insufficient cycles for DSPs or insufficient memory, multipliers, or fabric elements for FPGAs) to form $F'(\Omega, A, C)$.
4. Calculate power estimations for each element in $F'(\Omega, A, C)$ by combining the dynamic power consumption estimates for each waveform component with the static power consumption of the processors. For FPGAs, this is accomplished by summing these numbers. For DSPs, this is accomplished by multiplying the high activity power dynamic power estimate by the fraction of cycles used in the solution and adding this number to the static power consumption of the DSP. Finally, eliminate those solutions that violate power constraints to yield the feasible solution space, $F(\Omega, P, A, C)$.

7. CONCLUSIONS

This paper has presented a systematic approach for identifying the set of processors and partitions that can be expected to support a specified set of waveforms. This approach does not require the designer to be able to write assembly or VHDL code for each processor under consideration. Rather, the designer only needs to know the architecture and typical data sheet parameters for each chip. This simplification permits the designer to consider a wider range of processors and presumably find a better solution. Because of the sheer number of operations needed to support this methodology, particularly those steps listed in Section 6, computer aided tools are a necessity to form $F(\Omega, P, A, C)$. However, as a subsequent optimization search over $F(\Omega, P, A, C)$ is expected to occur, the electronic formulation of this $F(\Omega, P, A, C)$ greatly simplifies the search process by eliminating a painful data entry step.

It should be noted that there are a number of limitations to these estimations. For example, resources consumed in inter-component (routing, buffering) and inter-processor data transfer have been ignored. For DSPs, we have implicitly treated the power consumption of memory, computation, and control operations equivalently. Further, this paper does not address some design tradeoffs involving placement and utilization of memory (cache versus external) and different structures for the algorithms (e.g., transcendental versus CORDIC). However, these issues can be readily integrated into the proposed framework.

8. REFERENCES

[1] K. Williston, and J. Bier, "Evaluating the Latest DSPs for Communications Infrastructure Applications"

[2] M. Tsai, J. Bier, J. Eyre, "Evaluating FPGAs for Communication Infrastructure Applications", SDR Forum Technical Conference 2002. San Diego, CA.

[3] SPRS226D, "TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors Data sheet" (Rev. D) . Nov. 2003, Revised Oct 2004.

[4] CEVA-X 1620 Datasheet, Jan 2005.

[5] VC5510 Power Estimation Spreadsheet. http://focus.ti.com/docs/apps/catalog/resources/appnoteabstract.jhtml?abstractName=spra972

[6] Xilinx Reference Implementations. Availabe online: http://www.xilinx.com/ipcenter/

[7] Altera Reference Implementations. Availabe online: http://www.altera.com/products/ip/ipm-index.html

[8] DS031 (v3.3), "Virtex-II Platform FPGAs: Complete Data Sheet," June 24, 2004.

[9] Stratix II Device Handbook, Volume 1, Jan 2005.

[10] Virtex2 Pro Power Estimation Spreadsheet, http://www.xilinx.com/cgiin/power_tool/power_Virtex2p

[11] Stratix2 Power Estimation Spreadsheet, http://www.altera.com/support/devices/estimator/pow-powerplay.html