Algorithm/Architecture Co-exploration for Designing Energy Efficient Wireless Channel Estimator

Yan Meng, Ryan Kastner, and Timothy Sherwood
University of California, Santa Barbara
Santa Barbara, CA 93106-9560, USA
{*yanmeng, kastner*}*@ece.ucsb.edu; sherwood@cs.ucsb.edu*

Abstract

Wireless networks are making the vision of ubiquitous computing a reality: users will be able to connect anytime and anywhere from anything. To achieve this vision, the next generation of wireless devices must learn about, and adapt to, the transmission environment through a process called channel estimation. In this paper, we describe a cross-cutting approach to explore the design space to solve the channel estimation problem on reconfigurable devices. In particular we focus on the matching pursuit algorithm, which is a fast and accurate iterative algorithm for multipath channel estimation. Our methodology models modern reconfigurable devices as an array of BRAM-level operation blocks ("BLOBs"), which act as flexible data paths. With the model, we describe design techniques and tradeoffs, resulting in novel optimizations at every level in building an energy efficient MP core, from the theory and algorithms to the bit level. We present results from our design space exploration over a number of different parameters, including both high level characteristics of the application, data and computation partitioning schemes, and module- and bit-level low-power techniques. The results demonstrate the effectiveness and efficiency of our approach to building a high speed and low power channel estimator. The total power saving is 25.4%. We furthur show that the local, distributed computation is, on average, 145% faster with minimum cost in power dissipation, than the global, centralized computation.

1. Introduction

Wireless communication systems are rapidly becoming the preferred method of network access, and reconfigurable devices will certainly play an important role in this new era [17, 20]. There are many computationally challenging problems to be solved in this domain, and the extreme time-to-market and rapidly shifting protocols and standards make this an area ripe for a reconfigurable solution. To meet the needs of performance with low energy consumption for supporting ever increasing bandwidths demands and increased connectivity from multiple users, performance and energy efficient implementations are required and optimized at all levels of wireless system design, from protocols to signal processing and from system level design to physical design. At the heart of the next generation wireless devices is the ability to accurately estimate characteristics of the channel. Once these characteristics are determined, the device can correct for them to enable more simultaneous users, higher bandwidth, and lower power communications. In this paper, we investigate and apply optimization techniques in the algorithmic, architectural and bit levels to build an energy efficient channel estimator in a modern reconfigurable device with high performance.

Channel estimation is a fundamental problem in communication systems with the goal of characterizing the media over which communication is propagating [18]. Wireless communication channels typically contain multiple paths due to scattering effects, and thus the received signal is composed of many delayed and attenuated versions of the transmitted signal. The received signals from multiple paths may be either destructive or constructive. When there is destructive interference, the signal may be corrupted. The signal corruption problem may be alleviated by a process called multipath channel estimation [23]. It is used to characterize all of the significant transmission paths, and is the key to building high speed wireless networks.

The overriding trend among the modern wireless communication systems is that higher data rates and bandwidth requires increasingly complicated physical and data link layer approaches [23]. As such, more computational power is required from the hardware. In or-

der to achieve high data rates using these complicated transmission techniques, we must enable efficient and flexible signal processing devices starting with channel estimation algorithms. Unfortunately, hardware implementation has been largely ignored during the development of channel estimation algorithms. While there have been many theoretically-sound approaches proposed for multipath channel estimation and multiuser detection [14, 23], these approaches have not yet been adopted by hardware designers because of the complexity of the algorithms involved and the cost associated with realizing them in an actual implementation.

In order to realize high bandwidth wireless communication schemes, we must develop tools and methodologies for efficient multiuser, multipath channel estimation. To make the leap from theory to reality an efficient and flexible high performance platform is required. Reconfigurable systems offer the necessary balance between flexibility and performance by allowing the device to be configured to the algorithm at hand [12]. Reconfigurable systems allow for the post-fabrication programmability of software with the spatial computational style most commonly employed in hardware designs and are becoming an attractive option for implementing signal processing applications [5, 6, 15] because of their high processing power and customizability. The inclusion of new features in the FPGA fabrics, such as a large number of embedded multipliers, microprocessor cores, on-chip distributed memories, adds to this attractiveness. One such example is software-defined radio (SDR) [4], which attempts to provide an efficient and inexpensive mechanism for the production of multimode, multiband, and multifunctional wireless devices. The performance and flexibility of reconfigurable devices make them viable and ideal for implementing the SDR systems.

Traditionally, the performance metrics for signal processing and indeed, most processing in general, have been latency and throughput. Yet, with the proliferation of mobile, portable devices, it has become increasingly important that systems are not only fast, but also energy efficient. Currently, commercially available FPGAs either do not have both millions of gates and low-power features, or their support for low power feature is very limited. Purely relying on technology scaling will fall short of computational capability for more advanced algorithms which are demanded by wireless system in the near future. Thus, instead of studying low-level optimization techniques, in this paper, we investigate and apply algorithmic and architectural level optimization techniques for minimizing energy consumed by FPGAs in building a multipath channel estimator. Our techniques can also be used for a next generation FPGA that has low power dissipation feature as well as high computing power.

Our main contribution is a quantitative analysis of several energy efficient techniques that has resulted in novel optimizations at every levels, from the theory and algorithms to the architecture and bitwidth. We describe our design and quantify the tradeoffs in terms of channel estimation accuracy and the energy of our implementation. We model the target reconfigurable device as an array of BLOBs and study the data and computation partitioning problem through different architectural schemes. Along with employing the clock gating technique, our final result is an energy efficient MP core that has been mapped onto a Virtex-II XC2V3000 FPGA, resulting in 25.4% of total power savings.

The paper is organized as follows. Section 2 gives a high level overview of the matching pursuit algorithm. Section 3 describes our reconfigurable computation model. In section 4, different energy design techniques are presented for building the energy efficient channel estimator. A summary and conclusions can be found in Section 5.

2. Matching Pursuit Algorithm for Channel Estimation

2.1. Multipath Channel Propagation

Wireless communication channels typically contain multiple paths due to scattering effects, and thus the received signal is composed of many delayed and attenuated versions of the transmitted signal [18]. For outdoor communications, the scatterers may be buildings, mountains, etc., while for indoor communications, the scatterers may be walls, furniture, etc. Path lengths may vary greatly. We assume delay values $\tau \in [0, T)$, where T is the symbol duration, which is reasonable in most cases.

In this paper, the multipath spread is assumed to be at most one symbol duration, which is characteristic in current DS-CDMA systems [8, 11, 19]. The multipath channel with continuous-valued delays is approximated by a sparse tapped-delay-line (TDL) filter with discrete-valued delays $(i-1)T_s$, for $i = 1, 2, \ldots, N_s$, where $1/T_s$ is the Nyquist sampling rate, and N_s is the number of samples per symbol duration T [18]. Associated with each TDL path i is a complex-valued channel coefficient f_i, with the $\{f_i\}$ given by interpolation of the true channel. A *sparse* channel is one in which $N_f << N_s$ channel coefficients are non-negligible. The TDL representation of an example 5-path channel is shown in Figure 1 (solid line).

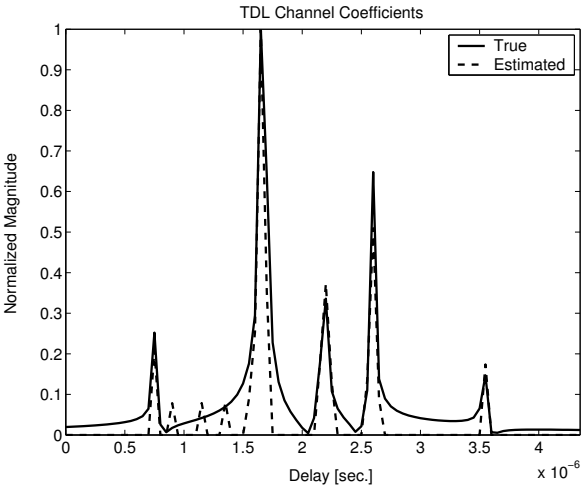

Figure 1: Multipath channel estimation with MP.

Consider the case of a single user transmitting on a multipath channel. The received signal after RF-to-baseband down-conversion and A/D sampling is denoted by

$$r = Sf + n \in \mathcal{C}^{MN_s*1}, \tag{1}$$

where M is the number of training symbols, n is the sampled additive white Gaussian noise vector, $f = [f_0, f_1, \cdots, f_{N_s-1}]^T \in \mathcal{C}^{N_s*1}$ is the channel coefficient vector, and $S \in \Re^{MN_s*N_s}$ is the characteristic signal matrix. $\Re$ and $\mathcal{C}$ represent the real and complex numbers, respectively, and $(.)^T$ denotes the transpose operation. The ith column S_i of S is the received signal due to path i if $f_i = 1$, and in general $f_i S_i$ is the received signal due to path i. S is given in [14] and is known *a priori*, since it depends only on the CDMA spreading sequence and the transmit and receive filters. Referring to the received signal model in equation (1), the multipath channel estimation problem is that of computing an estimate $\hat{f}$ of f, given S and the received signal vector r containing noise n.

2.2. The Matching Pursuit Algorithm

The Matching Pursuit (MP) channel estimation algorithm [3, 14] provides a low complexity approximation to the theoretical optimal solution, the Maximum Likelihood (ML) [14] solution, for sparse channels. Matching pursuits uses a process of successive cancellation and single-coefficient channel estimation to simplify the process.

The exact ML solution under the sparse channel constraint is given by

$$\hat{f} = \underset{f \in A_{N_f}}{\arg\min} \left\{ \|r - Sf\|^2 \right\}, \tag{2}$$

where $A_{N_f} = \{f : |f| = N_f\}$. Since the channel estimation cost function minimized in equation (2) is non-convex, an exhaustive search is required. The complexity (in terms of the number of scalar multiplications) of the optimal ML algorithm is is $\mathcal{O}(MN_s C_{N_f}^{N_s} Q^{2(N_s-N_f)})$ real scalar multiplications and $\mathcal{O}(2MN_s C_{N_f}^{N_s} Q^{2(N_s-N_f)})$ real scalar additions/subtractions, where the binomial coefficient $C_{N_f}^{N_s} = (N_s!)/(N_f!(N_s - N_f)!)$ and $1/(2Q)$ is the precision of the channel coefficient estimates. Clearly, real-time implementation of ML channel estimation is infeasible. By contrast, the MP algorithm [14] is highly efficient.

The matching pursuit algorithm is obtained by posing the ML estimation problem in terms of sufficient statistics, as follows.

$$-\|r - Sf\|^2 \propto 2\mathrm{Re}\left\{(V^0)^H f\right\} - f^H A f, \tag{3}$$

for signal parameter estimation and data symbol detection [23] where $V^0 = S^T r \in \mathcal{C}^{N_s*1}$ and $A = S^T S \in \Re^{N_s*N_s}$. S is known a priori, as mentioned in Section 2.1, and therefore S and A are pre-computed once for all time and stored in memory. The computation of V^0 can be parallelized as N_s vector inner products (correlations) $V_i^0 = S_i^T r$. Since the columns S_i of S are generated as filtered circular-shifted versions of the same CDMA spreading sequence, the computation of V^0 is equivalent to matched filtering the received signal r using a filter matched to the spreading sequence.

MP maximizes equation (3) iteratively, one channel coefficient $\hat{f}_{q_j}$ at a time, using a greedy approach in which q_j and $\hat{f}_{q_j}$ are selected such that the increase in equation (3) at each stage j is the largest possible. That is, the multipath signal components are estimated via successive interference cancellation. The algorithm is summarized in Figure 2. To eliminate the need for division operations, the vector a, with $a_k = 1/A(k,k)$ and $A(k,k)$ denoting the kth diagonal element in A, is pre-computed once for all time and stored in memory.

After each multipath successive interference cancellation stage, V^j is updated at the start of the next stage j as

$$V^j \leftarrow V^{j-1} - \hat{f}_{q_{j-1}} A_{q_{j-1}}. \tag{4}$$

Since the estimation of $\hat{f}$ via equation (3) depends only on V^0 and A, with A fixed, effectively the sufficient statistic is updated to reflect cancellation of the signal due to path q_j.

The algorithm terminates after stage $j = N_f$. In practice, N_f can be determined on the fly based on $|\hat{f}_{q_j}|$ and/or the SNR. For the example in Figure 1, $N_f = 15$,

Input($\mathbf{r}, \mathbf{S}, \mathbf{A}, \mathbf{a}$)
Output($\hat{\mathbf{f}}$)
// $\mathbf{r}$: received signal vector, $\mathbf{r} \in \mathbf{C}^{MN_S*1}$
// $\mathbf{S}$: $\mathbf{S}=[\mathbf{S}_1,\cdots,\mathbf{S}_i,\cdots,\mathbf{S}_{N_S}]$, $\mathbf{S} \in \mathbf{R}^{MN_S*N_S}$ and $\mathbf{S}_i \in \mathbf{R}^{MN_S*1}$
// $\mathbf{A}$: $\mathbf{A}=[\mathbf{A}_1,\cdots,\mathbf{A}_i,\cdots,\mathbf{A}_{N_S}]$, $\mathbf{A} \in \mathbf{R}^{N_S*N_S}$ and $\mathbf{A}_i \in \mathbf{R}^{N_S*1}$
// $\mathbf{a}$: $\mathbf{a}=[a_1,\cdots,a_i,\cdots,a_{N_S}]^T$, $\mathbf{a} \in \mathbf{R}^{N_S*1}$ and $a_i \in \mathbf{R}$
// $\hat{\mathbf{f}}$: estimated channel coefficients, $\hat{\mathbf{f}} \in \mathbf{C}^{N_S*1}$
MP(r, S, A, a)
1 **for** $i = 1, 2, \ldots, N_S$
// compute matched filter (MF) outputs
2 $v_i^1 = \mathbf{S}_i^H \mathbf{r}$
3 $\hat{f}_i = 0$
4 **end for**
// do successive interference cancellation
5 **for** $j = 1, 2, \ldots, N_f$
6 **for** $i = 1, 2, \ldots, N_S$
7 $\hat{f}_i^{\text{temp}} = v_i^j a_i$
8 $Q_i = (v_i^j)^* \hat{f}_i^{\text{temp}}$
9 **end for**
10 $q_j = \arg\max_{i \neq q_1,\ldots,q_{j-1}} \{Q_i\}$
11 $\hat{f}_{q_j} = \hat{f}_{q_j}^{\text{temp}}$
// update MF outputs
12 $\mathbf{v}^{j+1} = \mathbf{v}^j - \hat{f}_{q_j} \mathbf{A}_{q_j}$
13 **end for**
14 **return** ($\hat{\mathbf{f}}$)

Figure 2: The Matching Pursuit algorithm for channel estimation.

$M = 1$, and the MP channel estimate is shown (dotted line) for an SNR of 20 dB (ratio of energy per symbol to noise energy per symbol duration).

3. Reconfigurable Computation Model

The reconfigurable device paradigm is similar to that of software defined radio, in that devices can be easily re-programmed for adaptive response to operating conditions and applications. For instance, operating parameters including inter alia, frequency range, modulation type, and/or output power limitations can be set or altered. However, reconfigurable devices provide the additional benefit of programmable hardware, which allows the flexibility of software while yielding the high performance of a hardware implementation [12]. The performance and flexibility of reconfigurable computing systems make them ideal for implementing software defined radio systems.

Reconfigurable devices are a regular arrangement of programmable computational elements and communication structures, whose functionality is determined through configuration bits. There is a wide range of reconfigurable devices, which can be roughly classified by their granularity [12]. The *granularity* of a reconfigurable device is the abstraction level used to program or configure the device. FPGAs and CPLDs have logical level of abstraction. Instruction level reconfigurable devices (e.g., PRISC, Chimaera and Garp [12]) consist of computational units that perform arithmetic operations. Coarser grain reconfigurable devices, e.g., PADDI, MATRIX, RAW, synchroscalar and NAPA [12], have even larger programmable computational units.

The granularity gives a notion of the underlying freedom of the device. A coarse grain device limits your flexibility. For example, you may be forced to store data in a specific register and choose from a prespecified set of operations. A lower granularity level allows you to specify arbitrary memory organizations and complex customized functional units. These fine grain devices can be configured to efficiently implement irregular functions. Furthermore, we can implement functions using any data width, e.g. an 18 bit multiplier or 24 bit adder, which can be customized to the application at hand. However, if we are executing only common operations, a coarser grain device will be the better option, since these operations are implemented using fixed hardwired "ASIC" components. The prespecified operations are built precisely for that operation and do not incur the overhead associated with building it using programmable logic elements. For example, a DSP application that requires a lot of word-size addition and multiplications would be best suited to a device with instruction level granularity. If an application requires a bunch of Boolean operations, then device with logic level reconfigurability would perform the task most efficiently. In general, the more closely the application data is matched to the granularity, the more efficient the device will execute the application.

We choose a computational model that logically and physically divides the fine grain logic fabric into coarser grain *BRAM-level operation blocks (BLOBs)*. Each BLOB consists of a BRAM, fixed multiplier, and neighboring CLBs. The CLBs are equally divided across the BLOBs. For example, the target chip in our experiments, Virtex-II XC2V3000 FPGA, has 3584 CLBs. Dividing by 96 (the number of BRAMs and fixed multipliers) yields approximately 37 CLBs in each BLOB. A BLOB is capable of performing any number of simple instructions, e.g., multiplication (on the fixed multiplier), addition (on the CLBs), and any other type of custom instruction that can be implemented on the CLBs. Additionally, it has an 18-Kbit BRAM that can act as a register file, mini cache, etc. Each BLOB is es-

sentially a fully customizable data path, which causes most of the system energy consumption and delay.

Dividing the reconfigurable device into BLOBs has many advantages. First, we allow application developers to design using a higher level of abstraction. They can view the reconfigurable device as a sea of processors, which is an increasingly common method for developing computational fabrics [12]. Second, since the BLOBs are configurable at the logic level, we can program the fabric using a variety of different data flow and control methods, including SIMD and MIMD. Finally, the BLOB organization maintains the spatial model of computation that allows the reconfigurable device to perform a large number of parallel operations, and therefore achieve high performance.

4. Energy Efficient Design Techniques

Energy and power are often used interchangeably, however, they are not the same. Energy is the product of average power dissipation and latency. Therefore, it is necessary to understand power dissipation and its effect on latency and vice versa in order to better understand energy dissipation.

In this section, we will first briefly describe sources of power dissipation in FPGA based reconfigurable devices. We will then discuss techniques for achieving low energy dissipation by one of the three methods: lowering power dissipation, lowering latency, or lowering the product of the power and latency, at different abstraction levels, from the arithmetic and architectural to the algorithmic level. The result of our design is an energy efficient IP core that can be readily tuned to the requirements of its applications and initiated in any number of wireless devices.

4.1. Energy Dissipation in FPGA based Reconfigurable Devices

Several studies on power dissipation of reconfigurable devices have appeared in recent works [16, 22]. These works review that power dissipation in reconfigurable devices is primarily due to programmable interconnects. For instance, in the Virtex-II, the dynamic power dissipated in the interconnects is about 50% to 70%, while the remaining is being dissipated in logic, clock and I/O blocks. These results are different from ASIC technology, where clock distribtion typically dominates power dissipation [2]. The programmable interconnects consist of multiple pre-fabricated row and column interconnect wire segments of various lengths, with used and unused routing switches attached to each wire segment.

The second important factor that affects the power dissipation in reconfigurable devices is resource utilization. In typical reconfigurable system designs, a large number of of the resources are not utilized and do not have dynamic power dissipation.

Switching activity is another important factor that is used to determine the amount of dynamic power dissipation of each hardware resource. It depends not only on the type of the design but on the input stimuli.

To obtain the power consumption information of the target Xilinx chip XC2V3000, we did low-level simulation of the VHDL codings of our design with Mentor Graphics Modelsim and generate simulation results (.vcd file). The input vectors for the simulation was obtained from the high level simulation of the MP algorithm. The design was synthesized using Xilinx Synthesis Technology and the place and route file (.ncd file) was obtained. These two files were fed into Xilinx XPower tool to evaluate the average power dissipation. Energy dissipation was obtained by multiplying the average power by latency.

With good understanding of the sources of the power dissipation and the way to obtain the power dissipation, we can now discuss design and optimization techniques to achieve an energy efficient channel estimator, from the bit level to the algorithmic level design.

4.2. Binding Energy Efficient IP Cores

As modern chips are increasingly providing high computational power with the fixed components [24], like microprocessor cores, embedded multipliers and on-chip distributed memory, a very important factor to consider in designing systems is to choose energy efficient bindings and map operations onto available hardware resources. Different bindings affect energy consumption of the reconfigurable device greatly. For example, embedded multipliers, such as those in the Virtex-II and Altera Stratix families, can be more energy efficient than the multipliers implemented with CLBs. Our analysis shows that the energy consumed by a CLB-based multiplier is approximately twice of the energy consumption by an embedded multiplier core.

4.3. Bit Level Optimization — Bitwidth Analysis

Floating point functional units take much longer execution time and consume copious amount of power than their fixed-point counterparts. Therefore, we employ the fixed point representation in the rest of our study, which we show can provide reasonably accurate results.

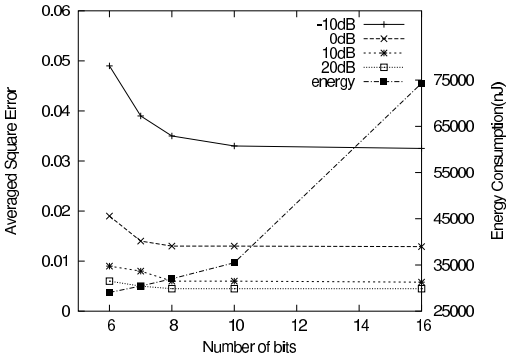

Figure 3: The tradeoff of channel estimation accuracy vs. the number of fixed-point bits.

An important consideration for implementing the matching pursuit algorithm is decision on the number of fixed-point bits. The larger the bitwidth, the more accurate the estimation results. Conversely, bigger bitwidths lead to larger and slower functional units, which has obvious negative effects on the latency and power. Therefore, it is imperative to find a good tradeoff between accuracy, latency and power.

To explore the design space in this dimension, we conducted bit-width analysis [9, 21]. Figure 3 shows the results for the average squared error (ASE) of the channel estimation vs. number of fixed-point bits for SNRs of -10, 0, 10, and 20 dB, where SNR is defined as the ratio of the desired signal energy to the noise signal energy; both are measured over one symbol duration. The results are averaged over three different multipath channels, with 30 ensemble runs (different noise realizations) per channel. Referring to Figure 3, it is clear that as SNR increases, the accuracy gets better, and 8 bits is sufficient over all SNRs to achieve accurate multipath channel estimation. Fewer number of bits (e.g., 6 bits) can lead to minor improvements in performance and power. This, however, comes at the cost of the large system error. Conversely, large bitwidth (16 bits) should not be used because it takes longer execution and larger power consumption, and only can improve the accuracy by a small degree. Therefore, a bitwidth of 8 is used in our study, which hits a sweet spot between accuracy, performance and power.

4.4. Architectural Level Optimization — Data and Computation Partitioning

Since reconfigurable devices provide the freedom to map various architectures, choosing the appropriate architecture affects the energy dissipation. Based on previous studies, interconnect dissipates a large amount of power. Therefore, minimizing the number of long wires or global communications between building logic blocks is beneficial. In the following section, we discuss the importance of the data and computation problem for on-chip communications and evaluate different architectures based on different schemes of partitioning the MP data and computations to meet the needs of low energy consumption.

4.4.1. Data and Computation Partitioning Problem Clock frequency has risen exponentially over the years and the fraction of the chip that is reachable by a signal in a single clock cycle has decreased exponentially [10]. Architectures that rely on global signals are quickly becoming infeasible [7]. Therefore, care must be taken to distribute the data and operations onto reconfigurable systems in a manner that limits the amount of global communication.

We formalize the data and computation partitioning problem using the following architectural assumptions.

1. The programmable logic contains C configurable logic blocks (CLBs).
2. There are B BRAMs. Local BRAM can be utilized by architectural synthesis tools for local intermediate data; however, it is only used for this purpose.
3. The CLBs and BRAMs are equally distributed across the chip. Furthermore, we assume that the CLBs and BRAMs are equally divided into B BLOBs where each group consists of 1 BRAM and C/B CLBs.
4. CLBs can read/write data in the BRAM of the same group, which is called local access with a total latency of l clock cycles. If CLBs access data stored in BRAMs from another group (called a remote access), it takes a total r clock cycles ($r = l + d$) since we assume an average of d clock cycles will be taken because of the longer r outing distance. Note that d is dependent on the distance of the BRAM that is being accessed.

4.4.2. Data and Computation Partitioning Before going into details of the specific data and computation distributions schemes, we provide stripped down overview of the matching pursuit algorithm presented in its full, formal, mathematical glory in Section 2. MP compares the received signal vector $\mathbf{r}$ with time delayed versions of a known training sequence that it expects to receive from a transmitting user. The $\mathbf{S}$ matrix represents these time delayed training sequences. The i-th column in the $\mathbf{S}$ matrix corresponds to the training sequence delayed by i samples. The training

sequence can be viewed as the transmitting device signature. The transmitting device sends its signature before sending data, so that the receiver can characterize the wireless channel between the transmitting device and itself. It then uses this channel estimate to demodulate future unknown data that it receives from that transmitting user. The data in the **S** is calculated using the signature (training sequence), CDMA spreading sequence and the transmit and receive filters, all of which are known a prior [1].

Matched filtering boils down to multiplying each sample of the received vector **r** with the corresponding sample of a column in the **S** matrix. Then, we accumulate all of these multiplied values to get a single value. This value represents the correlation of the received signal vector (**r**) with a time delayed version of the training sequence (S_i). The correlation value between **r** and column S_i corresponds to the likelihood that the received signal has been delayed by i samples. However, it is important to note that multipath may cause destructive or constructive interference. Therefore, a high correlation value does not necessarily mean that the received signal has a path delayed by that number of samples.

Matching pursuits works by performing matching filtering of the received signal with all of the delayed training sequences (again, corresponding to the columns of the **S**). It takes the delay with the highest correlation and subtracts that signal from the received signal vector. The algorithm iterates, continually canceling more signals, until it finds a sufficient characterization of the channel. Matched filtering takes a large majority of the computation time in MP, hence we focus on data and computation distribution for the filter. We evaluate different data and computation partitioning schemes for trading off energy and delay, and apply the optimal partitioning scheme into building the energy efficient MP core.

An important consideration for implementing the matching pursuit algorithm is the distribution of the data and computation. The matching pursuit algorithm and, in particular, the matched filtering, exhibits enormous opportunity for parallelization. Each matched filter operation can be performed in parallel. This corresponds to computing the received signal and path delays correlations in parallel. Furthermore, each multiplication of the matched filter can be performed in parallel. This corresponds to computing the samples of the received vector and path delay in parallel. However, we must then accumulate all of these sample correlations. The fastest method of doing this would be through the use of an adder tree. This would allow us to compute the matched filter in $O(log_2|r|)$ time at the expense of $O(|r|^2)$ multipliers. While the number of samples varies depending on the spreading sequence, typically, you need around 100 samples for the training sequences (we use 88 samples in our experiments). This would require approximately 10000 multipliers, which is far more resources that is available on even the largest reconfigurable devices. Even if such a device did exist, and we can easily extrapolate Moore's Law a few years to where such a device exists, we can rarely afford to devote the entire system to matched filtering. Therefore, it is imperative that we study the relationship of performance and design parameters between different data and computation partitioning schemes.

Seeing how the fully parallel scheme is infeasible, we must look to alternative schemes to serialize parts, if not all, of the operations to different tradeoff design metrics, e.g. delay, throughput, area, power, etc. Matched filters involve a quite large amount of data. A poor data distribution will result in large data transfer times, which can eliminate all of the benefits gained through the parallelization. Therefore, it is necessary to carefully distribute the data onto the target device to achieve good performance. The two-dimensional nature of the **S** matrix guides us to the following two schemes for data distribution.

The local scheme distributes the **S** matrix into BLOBs by column (see Figure 4). Additionally, the received vector **r** is replicated and distributed to each BRAM. Therefore, each BLOB computes a matched filtering of **r** and a delayed training sequence. This sequentializes the computation of the individual matched filters, but computes all of the filters in parallel.

The BLOB is configured as a multiply-accumulate (MAC) datapath, using the fixed multiplier and an adder implemented on the CLBs. This scheme uses distributed local control logic, i.e. each BLOB is controlled locally. This requires a BLOB for each filter, which is equal to the number of samples (88 in our case).

One could also imagine distributing multiple columns into the same BLOB. This provides a tradeoff between execution time and area. Since the BRAM has limited number of ports, we would have to sequentialize the matched filtering for each column in the BLOB in all but the smallest column sharing schemes. For example, if two columns $\mathbf{S_i}$ and $\mathbf{S_{i+1}}$ are distributed in the same BLOB, then the computation of the matched filter $\mathbf{S_{i+1}^T r}$ follows the computation of matched filter $\mathbf{S_i^T r}$.

1 Please see [14] for exact method to calculate the values of the **S** matrix.

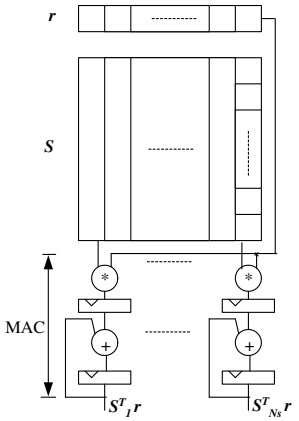

Figure 4: The local scheme for computing matched filter outputs. Matrix **S** *are partitioned into columns, which are then distributed into block RAMs. Each matched filter shares an embedded multiplier and an adder, and its output is from a multiplication and accumulation (MAC) unit.*

The global scheme distributes the **S** matrix into the BLOB by row (see Figure 5). Here, matched filters are computed in parallel, while the path correlations are computed sequentially. More precisely, the matched filter sample multiplications are computed in parallel, and the accumulate stage requires an adder tree. The adder tree is fully pipelined to allow overlapping execution of multiple matched filter accumulate stages.

Each sample of the received vector **r** is divided into the BRAM of the BLOB. The BLOB is simply configured to perform multiplication. The accumulate stage is computed separately and requires global control logic and data transfers of the multiplied samples from each BLOB.

Once again, we can tradeoff execution time for area by assigning multiple rows to each BLOB. This corresponds to sequentializing the matched filter operations. Each sample present in the BLOB would be executed in parallel, since we are limited by the number of ports on the BRAM and the number of multipliers that we implement in the BLOB. This would still require a separate adder tree, however, we can start performing MACs within the BLOB. Once again, consider the case where we partition two rows $\mathbf{S_j}$ and $\mathbf{S_{j+1}}$ onto the same BLOB. In this case, we could perform two multiplications, corresponding to the two samples from every path delay from **S**. However, we could also accumulate these two samples locally and send the two sample accumulated result to the adder tree for accumulation of the full delayed path. The BLOB data path here would resemble the data path from the local scheme. As both schemes move towards more serial implementations, we would indeed reach a point where the global

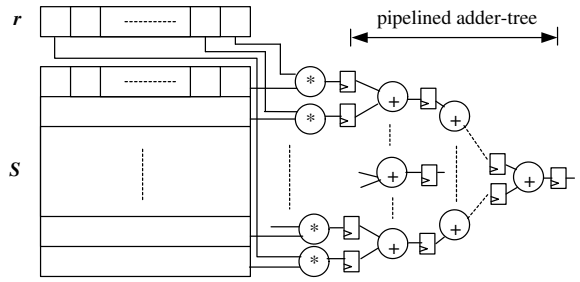

Figure 5: The global scheme for computing matched filter outputs. Matrix **S** *are partitioned into rows, which are then distributed into block RAMs. Each match filter uses N_S number of multipliers and its output is from the pipelined adder-tree, which combines the multiplication results.*

and local schemes are equivalent. This would happen when both schemes become fully sequentialized.

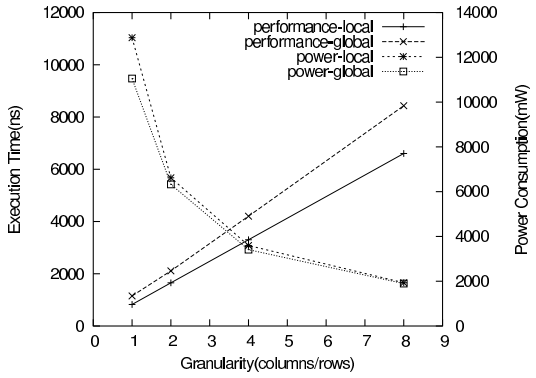

Figure 6: The tradeoff of performance vs. power by using both the local and the global scheme.

Figure 6 shows the results of performance and power consumption vs. granularity (number of columns or number of rows partitioned into each block RAM for the local scheme and the global scheme, respectively). The figure illustrates that for both schemes as the number of columns/rows of the **S** matrix packed into the same block RAM increases, the execution time increases linearly and the power consumption of hardware resources decreases exponentially. This observation can be explained by understanding that each matched filter is sequentially executed if multiple columns/rows are within one block RAM, which takes less power consumption but longer execution time. Depending on different system requirements, designers can follow the curve to implement the matched filters optimally. In the situation where high data rate is re-

quired, the best implementation is to employ the local scheme and distribute every column into each block RAM to achieve the best performance.

When we compared the best results of the each individual scheme in terms of performance and energy consumption (see Figure 6 and Figure 7), we can see that the local scheme can achieve faster execution time (376ns faster) and less energy consumption (4143nJ less) than the global scheme. This is primarily due to the fact that the pipelined adder-tree enlarges the system delay caused by a large amount of global communication across the chip. Therefore, the local scheme with one column in each BLOB is employed in our design, which can achieve the least amount of global communications and the lowest energy consumption.

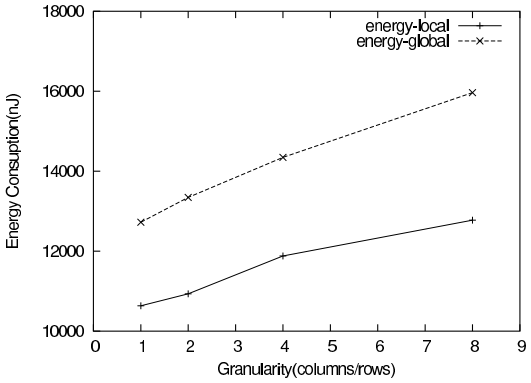

Figure 7: The energy consumptions by using both the local and the global schemes.

4.5. Algorithmic Level Optimization — Module Disabling

At the algorithmic level, a lot of optimizations can be conducted [12], with a strong impact on the system's power consumption. The MP algorithm is designed for easily parallelization, ie. the BLOB-level computation of each path can run independently of others. This enables the clock gating technique to disable BLOBs that are not in use or have been detected during the computation. By disabling the unnecessary computation modules, the power dissipation can be reduced. In MP, for instance, after canceling a significant path, the computation of this path is useless but still consumes power. With the BLOB model of the execution of each path, the implementation can exploit clocking gating to disable the computation modules of those detected paths.

As discussed in [13], for radiolocation applications the MP stopping criterion can not only ensure that

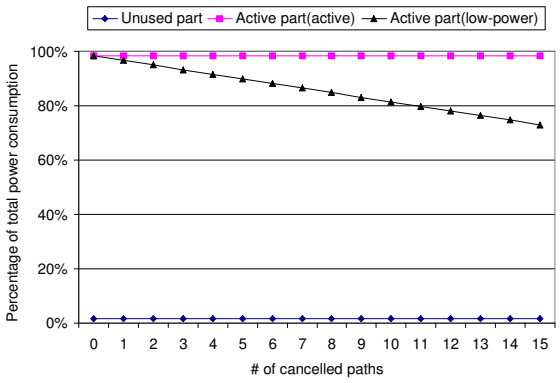

Figure 8: Percentage of power savings with the algorithmic level optimization.

the most significant paths are accurately detected, but guarantee detection of the direct path, which can be consequently used to measure the line of sight distance from the transmitter to the receiver using the TOA-based method. When the stopping condition is met based on the stopping criterion, the MP core can be simply switched off to further save power.

In FPGAs, clock gating can be realized either by using primitives such as BUFGMUX to switch from a high frequency clock to a low frequency clock [20] or by introducing a sleep transistor to switch the unnecessary modules off [1].

To study the power consumption of the total on-chip hardware resources, we divide the resources into three types: unused, active, and disabled parts. The unused part is the amount of hardware resources that are not contribute to building the MP core. The active part includes the resources that are always actively executing, which contribute to both dynamic and static power dissipation. The disabled part has no dynamic switching and thus only contributes to static power consumption.

Figure 8 shows the percentage of the total power consumption as the paths are detected and canceled by disabling the unnecessary modules during the computation. From the figure, we can see that the unused part consumes 1.67% of total power consumption. When we apply the algorithmic level optimization for achieving low power, the power consumption decreases linearly as the detected paths are successively canceled, and for each cancellation the power saving is about 1.69% compared with leaving all the canceled paths active. After detecting 15 paths, the MP terminates, resulting in a total power saving of 25.4%, and running 216 times faster than executing the MP algorithm on an 2.17GHz AMD Athlon XP microprocessor with 1GB RAMs.

5. Conclusion

Wireless connectivity is playing an increasingly important role in communication systems. To meet the demands of higher data rate and higher multi-user capacity, channel estimation has been employed as the key to modern communication algorithms. Given the frequency with which new wireless protocols are developed and deployed, an ASIC based approach is a poor option due to it's lack of programmability. On the other side, with the computational demands that the current generation of signal processing algorithms place on a device, a microprocessor simply could not provide enough throughput. A reconfigurable device provides an excellent balance between these two extremes, and also presents an unique opportunity to designing and optimizing the signal processing algorithms in concert with the actual hardware implementation. In this paper, we described a cross-cutting approach to explore the design space to solve the channel estimation problem on reconfigurable devices.

As high performance and energy efficient implementations remain as a design challenge, we focused on building a high speed and energy efficient matching pursuit IP core, which has been optimized at all levels of the system design: bit, architecture, and algorithm levels. At the bit level, we have studied the tradeoff of energy consumption vs. accuracy. At the architectural level, we have investigated many different data and computation partitioning schemes, and found that an effective way of partitioning an application is to treat the reconfigurable device as a collection computational blocks, where each block has a single block of memory and an associated set of computational abilities. We call each of these logical units a BRAM-level operational block, or "BLOB". We have demonstrated that by keeping both the control and the data signals local to each BLOB to the greatest extent possible, the implementation can achieve the highest performance and the least energy consumption. At the algorithmic level where the decision has the strong impact on the system power consumption, we have employed the clock gating technique to disable the unnecessary computation modules, achieving 25.4% of total power savings, which can execute 216 times faster than running the algorithm on a state of the art microprocessor.

References

[1] A.Gayasen, Y.Tsai, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, and T. Tuan. Reducing leakage energy in fpgas using region-constrained placement. In *FPGA*, 2004.

[2] Altera, Co. *Stratix II low power design techniques*, Mar. 2005.

[3] S. F. Cotter and B. D. Rao. Sparse channel estimation via matching pursuit with application to equalization. *IEEE Trans. Communications*, 50:374–377, Mar. 2002.

[4] M. Cummings and S. Heath. Mode switching and software download for software defined radio: the SDR Forum approach. *IEEE Comms. Magazine*, Aug. 1999.

[5] A. Alsolaim et al. Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems. In *FCCM*, 2000.

[6] L. Jian et al. A dynamically-reconfigurable power-efficient turbo decoder. *FCCM*, 2004.

[7] V. Agarwal et al. Clock rate versus ipc: the end of the road for conventional microarchitectures. *ISCA*, 2000.

[8] V. Erceg et al. A model for the multipath delay profile of fixed wireless channels. *IEEE J. Select. Areas Commun.*, 17(3):399–410, March 1999.

[9] A. A. Gaffarl, O. Mencerl, W. Luk, and P. Y. K. Cheung. Unifying bit-width optimisation for fixed-point and floating-point designs. In *IEEE Trans. on Computer-Aided Design*, Nov. 2001.

[10] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proc. of the IEEE*, 89(4):490–504, 2001.

[11] M. R. Karim and M. Sarraf. *W-CDMA and cdma2000 for 3G Mobile Networks.* McGraw-Hill, New York, 2002.

[12] R. Kastner, A. Kaplan, and M. Sarrafzadeh. *Synthesis Techniques and Optimizations for Reconfigurable Systems.* Kluwer Academic, Boston, 2004.

[13] S. Kim. *DS-CDMA Multiuser Channel Estimation Algorithms with Applications to Radiolocation.* PhD thesis, Univ. of California, Santa Barbara, March 2005.

[14] S. Kim and R. A. Iltis. A matching pursuit/GSIC-based algorithm for DS-CDMA sparse channel estimation. *IEEE Signal Processing Letters*, Jan. 2004.

[15] D. U. Lee, W. Luk, C. Wang, and C. Jones. A flexible hardware encoder for low-density parity-check codes. *FCCM*, 2004.

[16] F. Li and L. He. Power modeling and characteristics of field programmable gate arrays. *IEEE Trans. on Computer-aided design*, Oct. 2005.

[17] Y. Meng, A. Brown, R. Iltis, T. Sherwood, H. Lee, and R. Kastner. In *Design Automation Conference*, 2005.

[18] J. Proakis. *Digital Communications.* McGraw-Hill, New York, NY, 1995.

[19] T. Rautiainen, G. Wolfle, and R. Hoppe. Verifying path loss and delay spread predictions of a 3D ray tracing propagation model in urban environment. In *IEEE Vehicular Technology Conference (VTC'02)*, Sept. 2002.

[20] V.K. Prasanna S. Choi, R. Scrofano and J. Jang. Energy-efficient signal processing using fpgas. In *FPGA*, 2003.

[21] M. Stephenson, J.Babb, and S. Amarasinghe. Bitwidth analysis with application to silicon compilation. In *IEEE Trans. on Computer-Aided Design*, Nov. 2001.

[22] T. Tuan and B. Lai. Leakage power analysis of a 90nm FPGA. In *CICC*, 2003.

[23] S. Verdú. *Multiuser detection.* Cambridge University Press, New York, Oct. 1998.

[24] Xilinx, Inc. *Virtex-II 1.5V Field Programmable Gated Arrays Datasheet*, Oct. 2001.