# A Low-Power Parallel Design of Discrete Wavelet Transform using Subthreshold Voltage Technology

Michael B. Henry, Syed I. Haider, and Leyla Nazhandali
Virginia Polytechnic Institute and State University
302 Whittemore, Blacksburg, VA 24061
mbh@vt.edu, syedh@vt.edu, leyla@vt.edu

## ABSTRACT

*The Discrete Wavelet Transform (DWT) is a means to analyze the frequency content of a signal and has extensive uses, including the JPEG2000 codec. Many portable and battery operated applications of DWT are expected in the near future that require a low power implementation of this transform. In this paper, a parallel VLSI implementation of a 2D lifting-based DWT processor is presented that is scalable from 2 to 256 parallel units. This design benefits from an efficient data distribution module to the parallel units, which constitutes a small overhead, and is able to significantly benefit from voltage scaling to achieve energy efficiency. In our design, the number of parallel units is increased and their speed is reduced through voltage scaling, while maintaining a constant throughput. Our results show that the optimal operating voltage of the parallel units, for a target throughput of 200MHz, is 386mV. This is below the threshold voltage, which is the voltage that turns the transistors on. Since operating a circuit in subthreshold voltage consumes 100+ times less power than running it at nominal voltage, our design is able to provide the same throughput as a reference pipelined implementation with 26 times less power consumption.*

## Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application Based Systems

## General Terms

Design, Measurement, Performance

## Keywords

Wavelet, Subthreshold, Parallel, Low Power


*CASES'08,* October 19–24, 2008, Atlanta, Georgia, USA.


## 1. INTRODUCTION AND BACKGROUND

The Wavelet Transform (WT) is a method for analyzing the frequency content of a signal, and is similar to the Fourier Transform (FT)[5]. The wavelet transform, unlike fourier transforms, can provide temporal information about when frequencies occur. It also provides better resolution for both high and low frequencies and has uses in many fields such as image processing. However, even in its more efficient forms, the wavelet transform requires a significant amount of processing power. To facilitate its use on battery powered system on chip applications or in signal processing, fast and efficient transform hardware is desirable. One portable application involving the use of wavelet transform is devices that use the JPEG2000 codec. Cameras, digital picture frames, cell phones, and mobile internet browsers are all predicted to eventually use the new JPEG2000 codec instead of JPEG[18]. Portable sensing applications can also make use of the signal processing applications that use wavelet transforms.

In this paper we propose a parallel implementation for the discretized form of the wavelet transform: the DWT. Because of its efficient parallel architecture with low overhead, this design is able to significantly benefit from voltage scaling to achieve energy efficiency. In our design, the number of parallel units is increased and the speed is reduced while maintaining a desired throughput. Our results show that the optimal operating voltage of the parallel units is well below the threshold voltage, which is the voltage that turns the transistors on. In this section, we provide the background to two of the topics related to this work: Discrete Wavelet Transform and subthreshold voltage operation.

### 1.1 DWT

The Discrete Wavelet Transform (DWT) is a discretized form of the continuous wavelet transform, performed on 1-Dimensional (1D) and 2-Dimensional (2D) arrays. The original method used to perform a DWT is convolution, a method which requires a large amount of processing. The lifting scheme [6] was proposed as a method of reducing the amount of computation. Two lifting based techniques, the (5,3) and (9,7) transforms, were chosen as the transform coders for JPEG2000 due to their advantages in compression and flexibility over the methods used in JPEG. In this paper, we focus on implementing the (5,3) transform that unlike the (9,7) version, uses integer coefficients and therefore performs lossless transformation.

The 1D transform operates on a 1D array and produces an output that is the same size as the input. Equations 1 and 2 describe the 1D (5,3) DWT, and are derived using

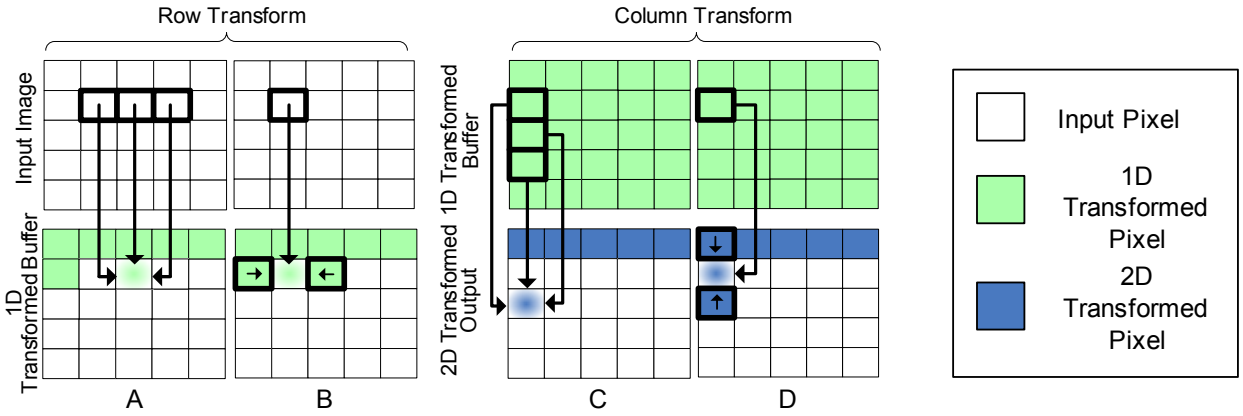


**Figure 1: Row and Column DWT Transform Figures A-B show a row transform performed on an input image. Figures C-D show a column transform performed on row-transformed data**

the lifting scheme in [6]. If a pixel's index is odd, a high pass transformation is performed on it. As seen in Equation 1, it depends on its two neighbors. The low pass calculation performed on even indexed pixels is dependent on the transformed high pass result of its two neighbors. This means that if a pixel is on an even index, it depends on its four closest *untransformed* neighbors. For data at the edges, mirroring is used to produce a reversible result [20].

$$HP(2n+1) = Input(2n+1) - \frac{1}{2}[Input(2n) + Input(2n+2)] \quad (1)$$

$$LP(2n) = Input(2n) + \frac{1}{4}[HP(2n-1) + HP(2n+1) + 2] \quad (2)$$

With a 2D transformation, these equations will be applied twice to each pixel: once in the row direction and once in the column direction. A 2D transform operates on a 2D array by first treating all of the rows of an image as 1D arrays and performing a 1D DWT on them. The columns of the resulting transformed array are then treated with another 1D DWT. Figure 1 shows the progression of a 2D DWT. In the first step shown in parts A and B of the figure, the pixels are processed row-wise. In the second step shown in parts C and D of the figure, the columns are processed. The arrows in the figure represent data dependencies. In this figure, the original untransformed pixels are shown in white, while the row-transformed and row-and-column-transformed pixels are shown in green and blue respectively.

## 1.2 Subthreshold Voltage Operation

Figure 2(a) shows a CMOS transistor identifying its source and gate. When the source voltage is above a certain ***threshold***, the transistor effectively functions like a switch responding to the changes that come from gate voltage. Lowering the voltage source or voltage scaling has been a prevalent method for improving the energy efficiency of microprocessors [3, 2]. This is due to the fact that reducing the voltage drops the energy consumption of a microprocessor quadratically, while decreasing its performance linearly. Figure 3 clearly shows this effect which presents the energy, power, and frequency trends with respect to voltage scaling [1] for the DWT parallel unit discussed in Section 3 and shown in Figure 6. It has been known for some time that CMOS gates operate seamlessly from full-voltage source to well below the threshold voltage - the voltage that turns the transistor on - at times reaching as low as 100mV [10, 14]. Recently, a number of prototype designs have shown that with careful design and replacement of these analog-like devices with standard switching counterparts, it is possible to extend the traditional voltage-scaling limit to below the threshold voltage, i.e. ***subthreshold-voltage***[2] region [22, 19, 12].

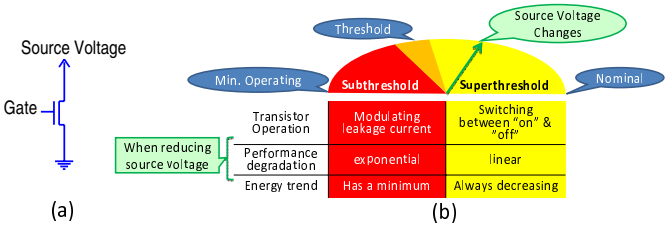


**Figure 2: Overview of sub/superthreshold operation.**

Figure 2(b) provides an overview of subthreshold and superthreshold operation differences. First, the transistors are not switching as normal in the subthreshold region; instead they simply modulate leakage current to charge or discharge their load capacitance and eventually perform computation. This, in turn, results in exponential degradation of performance in the subthreshold region as opposed to linear degradation when voltage is reduced in superthreshold. Moreover, because the system operates at much lower voltages, it becomes more susceptible to some manufacturing and operational problems such as process variation and soft errors. These issues as well as accurate modeling of subthreshold leakage are currently under investigation by several research

[1] In order to obtain these graphs, we simulated an inverter chain by applying a pulse input using SPICE and fitted models for leakage current and delay with respect to Vdd. Then, we applied the fitted models to our synthesized DWT processor. This methodology has been used before in [7, 25, 15, 16] and validated against actual silicon measurements in [7, 25].

[2] In this document, we may use super/subthreshold words in place of super/subthreshold-voltage for brevity.

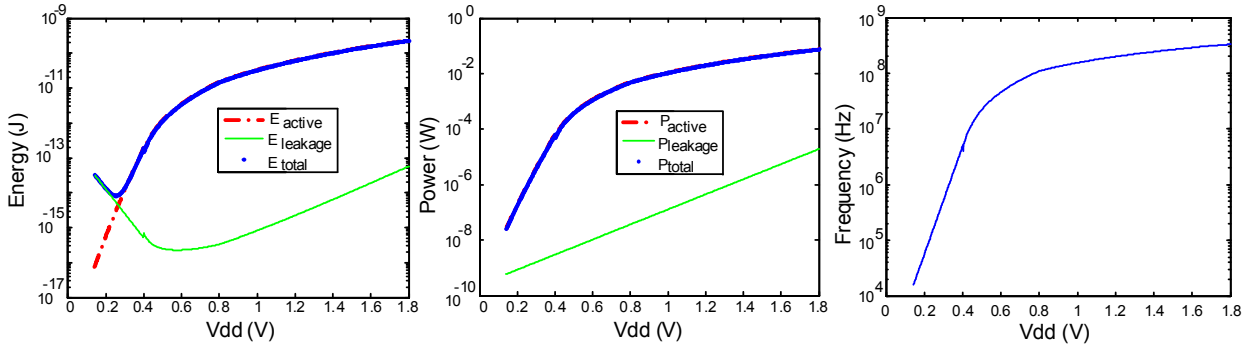


**Figure 3: Energy, power, and frequency trends w.r.t. voltage for a parallel DWT unit.**

groups in the VLSI and digital electronics area who have shown promising results [17, 11, 13, 8, 9, 22].

Figure 3 reveals another difference between the two regions: unlike in the superthreshold region, voltage scaling does not necessarily improve energy efficiency over the entire subthreshold region. While the active (dynamic) component of the energy is reduced by decreasing the voltage in both regions, the leakage (static) component eventually starts to increase exponentially in the subthreshold region. As shown in Figure 3, these two contrasting energy trends create a voltage point with minimum energy consumption [24], called ***optimum-energy voltage***, where a particular design reaches its maximum energy efficiency. In the following sections, we will argue that based on the above characteristics of the subthreshold region, it can be adapted as a great candidate for low-energy hardware design.

## 2. STATE-OF-THE-ART PIPELINED IMPLEMENTATION OF DWT

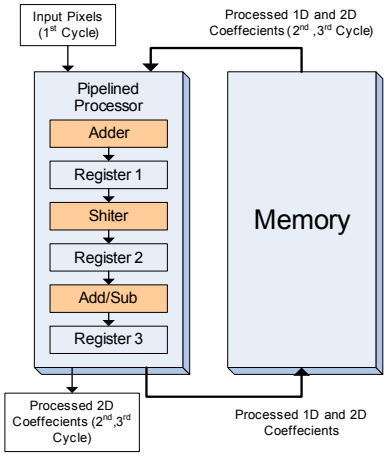


**Figure 4: Reference Pipeline Implementation The reference implementation uses a single pipelined processor to perform row and column transforms. It must store intermediate values in an SRAM bank.**

As can be seen by Equations 1 and 2, a 1D (5,3) transform requires 2 add/subtracts and a shift that is used for division by 2 or 4. The '+2' for the low pass step is a rounding operation and is present for JPEG2000 compatibility. It can be implemented using additional logic within an adder and therefore would not require another adder. The reference implementation, described in [21], breaks the DWT operations into three pipeline stages, where each stage represents a mathematical operation from equations 1 and 2. The single processor scans and processes an entire row, then moves onto the next row and repeats.

The data is fed into the reference implementation serially from memory at a certain throughput. A single input pixel of a 2D-DWT is processed twice; once for the row transform and once for the column transform. When the processor is operating on an even row, it cannot perform a column transform because the operation requires processed data from the next row, which is not available. The result of this dependency is that while the pipelined design is on odd rows, it must perform three transforms for every pixel: A row transform, a column transform for an odd row and an additional column transform for an even row. To facilitate the ability to perform a 1D DWT three times in one data clock cycle, the DWT processor runs at three times the speed of the data clock. Hardware utilization is not optimal because when the processor is working on even rows, 67% of the time no operations are being carried out.

### 2.1 Memory Organization of Pipelined Implementation

All column transformations require 1D row transformed data, and column transformations for even indices require column transformed data from the odd neighboring indices. This dependant data needs to be temporarily saved in an external memory bank. Three memory banks are required: one bank stores column transformed data for even column transformation, and two banks store row transformed data. These banks are implemented using SRAM. The pipelined design also requires 6 internal registers that save previous values and input values.

### 2.2 Potential Weaknesses

Using a single DWT processor to perform all of the row and column transforms has its drawbacks. There are three inputs into the DWT processor, but there are a number of different sources for these inputs. For row transforms, the inputs come from the input registers and internal registers. For column transforms, the inputs come from the memory banks. The edges and corners further complicate things because they are special cases that require mirrored data. The inputs are also different depending on if the processor is operating on an even or odd row or column. The end result is that each input to the DWT processor has up to 9 different possible sources. This requires a multiplexer for each input.

# 3. PROPOSED PARALLEL IMPLEMENTATION

The parallel implementation is designed so that multiple columns are processed at the same time. This means the parallel system will require more hardware, but will be faster and through voltage scaling techniques, the power consumption of the system as a whole can potentially be reduced.

## 3.1 Design Overview

Figure 5 shows a 1D parallel transformation system. Each column has its own dedicated (5,3) processing unit. This unit, which we call a DWT processor, performs the complete DWT operations mentioned in Equations 1 and 2 in one single clock cycle. The odd column (high pass) results depend on the untransformed data of their neighbors so the odd DWT processors are wired as such. The even columns (low pass) depend on the *transformed* data of their neighbors so they are wired to a register that is connected to the output of the high pass DWT processors next to them. This is relatively simple, but it only performs a 1D transformation. To perform a 2D transformation, the row transformed data needs to be saved and processed a second time. Our proposed parallel design uses another set of DWT processors for this purpose. We call the DWT processors that transform the input pixels *row processors*, and the ones that work on the result of these row processors *column processors*.

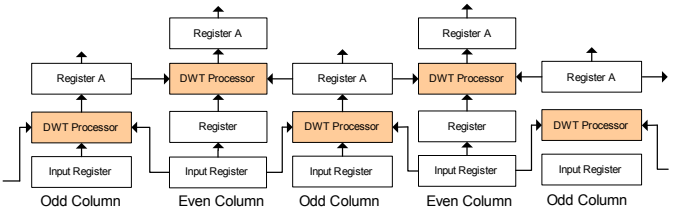


**Figure 5: A parallel version of a 1D (5,3) processor. High pass data from the odd columns are saved in a register. In the next clock cycle, the even columns use that data to process the low pass data.**

Figure 6 shows how row and column processors are organized in a simple 2x2 design. A chain of 5 registers is connected to the output of a row DWT processor. The row transformed data shifts up through these registers. A column DWT processor connects to these and performs a second DWT on the row transformed data. Like the row transform, a column high pass and column low pass result is produced. During one clock cycle, the column processor is connected to the bottom three registers of the register chain. These registers all contain row transformed data and the result is a column high pass result. Because the column low pass transformation requires the already processed column high pass result, when a column high pass result is produced, it is inserted into the register chain. During the next clock cycle, the column processor is connected to the top three registers. These three registers contain row transformed data in the middle, and column high pass data in the top and bottom. The result of this column transform is a low pass result. Thus, in two clock cycles a 2D high pass and low pass result is produced; or in other words, one 2D transformed pixel is produced in each clock cycle. Note that the even and odd columns share a register and odd and even columns are wired slightly differently.

The parallel design does not require an external memory module or SRAM. The only memory used are flip flop based registers within the parallel units. The control of these registers are very simple and usually involves shifting a value from a previous register, and shifting out a value to the next register. There is no need for address decoding and complicated wiring that SRAM based modules require.

## 3.2 Working with Large Images (Striping)

We refer to the maximum image width a design can transform without breaking up the image as the *width of the design*. For the pipelined design, the width is determined by the width of the memory banks used to store intermediate values. For the parallel design, the width is the number of parallel units. When dealing with an input image that is $n$ pixels wide, it is possible to transform the image with an implementation that has a width less than $n$. This can be done by striping the image. As shown in Figure 7, striping breaks the image into stripes and processes all of the columns of a stripe in the image before moving onto the next stripe.

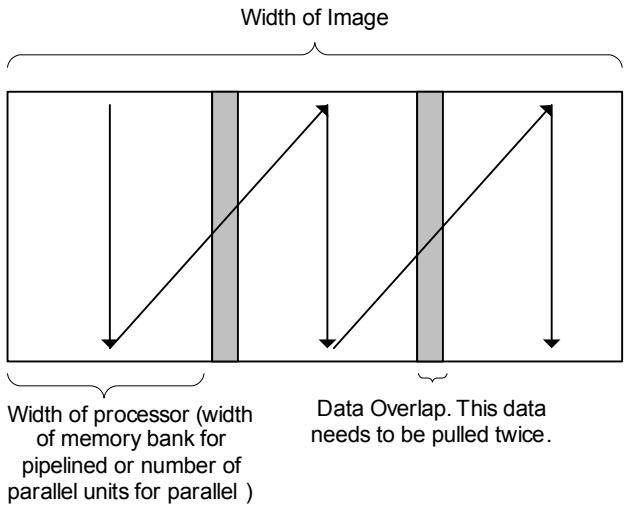


**Figure 7: Striping an input image If a processor does not have the memory or number of parallel units necessary to handle an image of a certain width, the image can be broken into thinner stripes. The shaded area needs to be retrieved twice from memory and is thus overhead.**

There is a penalty associated with striping: a row transform operation on the edge of a stripe requires data from the stripe next to it. If the edge of the stripe is an odd column, it requires the pixel next to it on the neighboring stripe. If the edge is an even column, it requires two pixels from the neighboring stripe. Assuming that the width of the processor is a power of two, there is always one edge that is on an even column, and one edge that is on an odd column. These additional dependencies to the left and right of the stripe must be loaded from memory to the internal storage of the design.

The penalty associated with striping is an increase in bus power consumption. Additional parallel units are not needed when striping because the extra data is only needed to satisfy a dependency of the calculations of edge pixels. For even numbered columns on the edge of a stripe, *row transformed* data is required as a dependency, so two pixels from the neighboring stripe are needed and that data needs to be row transformed. This requires an additional DWT

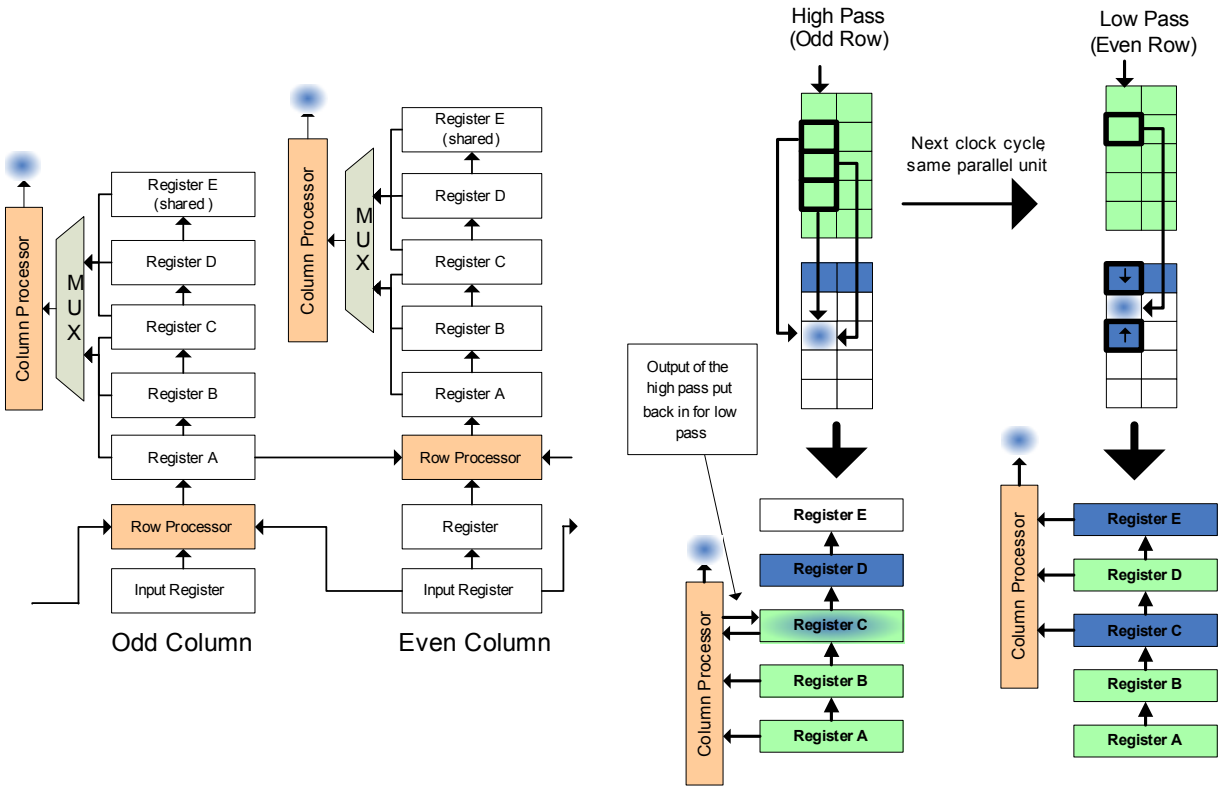


**Figure 6: 2D DWT Parallel System If we take a 1D parallel unit from Figure 5, then use a register chain and shift the 1D transformed data through it, we can then do another (5,3) transform and produce a 2D transform. 0**

processor. As mentioned earlier three extra pixels per row are needed, so three additional registers are needed on the bus. The bus will also have to run faster so it can keep up with filling the additional registers. Equation 3 lists the number of extra pixels per row required to stripe, given a width of the image and a width of the processor. Equation 4 lists the new clock speed the bus must run at to supply the additional registers.

$$ExtraPixelsPerRow = \lfloor \log_2(ImageWidth) - \log_2(ProcessorWidth) - 1 \rfloor \quad (3)$$

$$NewBusDataRate = \frac{(ProcessorWidth + ExtraPixelsPerRow) * OldBusDataRate}{ProcessorWidth} \quad (4)$$

The striping penalty would then be the increased power consumption as a result of running the bus faster, adding three registers to the bus, and adding an additional DWT processor.

## 3.3 Data Distribution to the Parallel Units

The assumption for input data of the parallel design is the same as the reference pipelined design: image data will be delivered serially at a given data rate. Some sort of data delivery system is required that distributes the serial data to all of the parallel units. The simplest type of data distribution system is the bus shown in Figure 8(a) Each input register of a parallel unit is connected to the bus and the serial data is sent over the serial bus at the overall data rate. A global column counter is also sent to every parallel unit. When the global column counter equals the column number of a parallel unit, it latches in the data from the bus. As mentioned in Section 4.3, the $V_{dd}$ of the parallel units is reduced in order to reduce the power consumption. Moreover, it will be shown in the results section that the optimum operating voltage of these parallel units are well into the subthreshold voltage region. However, it is not possible to slow down the input registers of the parallel units as much as the DWT processors because the inputs to these registers change at a fast data clock rate and using slow registers will result in setup and hold errors. On the other hand, fast input registers consume a considerable amount of power, which squanders the gains from power reduction of DWT processors.

In order to solve this problem, we split the bus into 2 or more slower buses. This is the reverse of multiplexing and is done using a demux as shown in Figure 8. If the original data bus is split to 2 busses, for example, the speed of each of the new 2 busses is half of the original bus while the two bus collectively carry the data that is delivered through the original bus. The benefit of splitting the bus is that the input registers to the parallel units have more time to latch the data and can be slowed down in order to reduce the power consumption. It is noteworthy that when the demux selects one side, the other side goes to an indeterminate state, so a fast register is used to latch the value onto the sub bus. The area and power consumption of this data distribution module is the overhead associated with increasing the parallelism in our design. The more the number of units, the larger this overhead. In the results section, we provide the result of an analysis on the optimum number of bus splits.

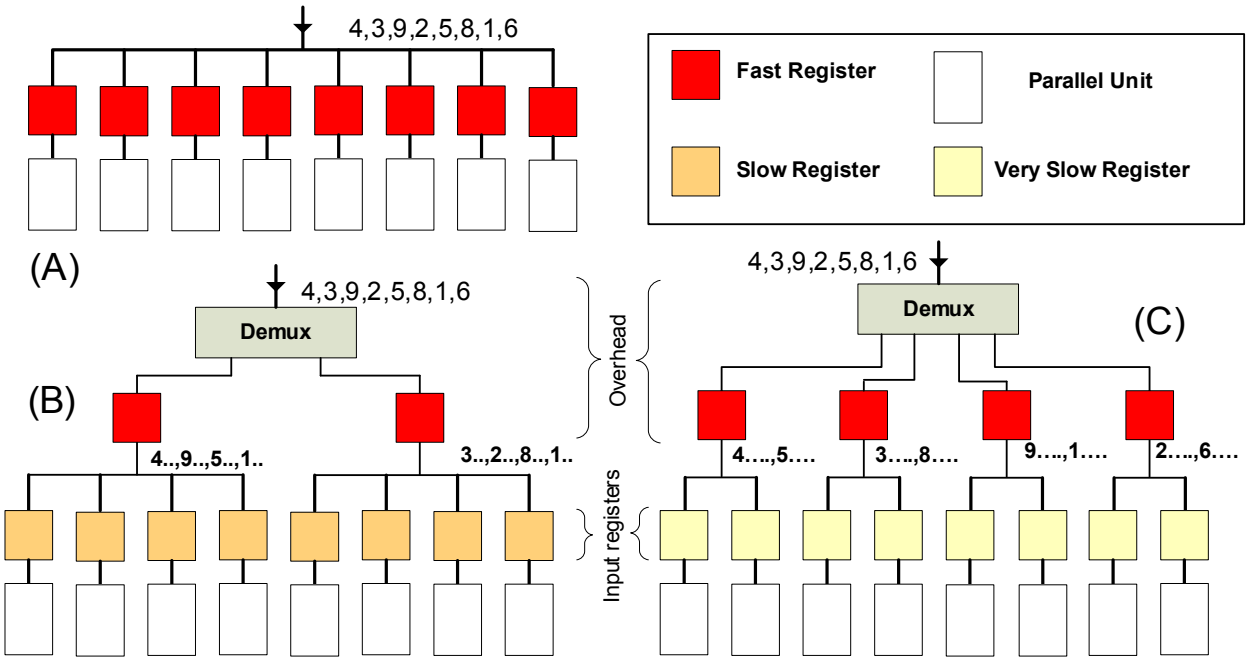


**Figure 8: Bus Splitting A shows a simple bus. B shows a single split bus. Figure C shows a bus split four times. As the bus gets split more and more, the $V_{dd}$ of the input registers can be reduced, saving power.**

# 4. METHODOLOGY

## 4.1 HDL Tool Chain

**Table 1: HDL Tool Chain**

| Design Aspect | Tool |
|---|---|
| HDL | Verilog |
| Simulator | Synopsys VCS |
| Standard Cell Library | OSU 0.18$\mu$m [1] |
| Synthesis | Synopsys Design Compiler |
| Power Measurements | Synopsys PrimeTime PX |

## 4.2 SRAM simulation

The pipelined design contains three memory banks. Modern memory banks are usually implemented as SRAM modules. The OSU standard cell library does not have an SRAM model and synthesizes memory structures inefficiently with standard flip-flops and primitive gates. To model these memory elements more realistically CACTI 4.2 was used to obtain power, area, and speed[23]. CACTI is an SRAM cache modeling tool that is used extensively in computer architecture research. To model the structure of a memory bank, the CACTI tool was setup to model a cache that is one-way set associative and has zero tag bits. These settings eliminate the tag portion of the cache. The CACTI tool only allows a minimum entry length of 8 bytes. Word length has a significant affect on power and area of memory. Therefore, curve fitting was applied to the CACTI data of memory structures with varying word lengths and the area and power of the memory unit was extrapolated.

## 4.3 Achieving Maximum Energy Efficiency

There are 2n DWT processors or n pairs of row and column processors (called DWT pairs) in our proposed parallel design where $n$ is the width of the design (defined in the previous section). As stated before, after the initial warmup time, each DWT pair transforms two pixels in each cycle. Therefore, the throughput of the parallel design can be calculated as shown in Equation :

$$ParallelThroughput = n * SpeedOfDWTProcessor \quad (5)$$

Since the throughput depends on both the width of the design and the speed of DWT processors, it is possible to change their values and still maintain the same throughput. The goal of our design is to maximize energy efficiency for a given throughput. In order to achieve this, we compute the power consumption of the parallel design while varying n and maintaining the same throughput[3]. As we increase n, the speed of the DWT processors is reduced. We use voltage scaling techniques in order to achieve this. Figure 9 shows the algorithm used to find the most energy efficient design.

[3] The DWT can be performed on an image of any width, but one of the driving applications of the wavelet transform is JPEG2000. In the JPEG2000 compression process, the output of the wavelet transform goes into an encoder that uses Embedded Block Coding with Optimal Truncation (EBCOT). The maximum size of a code block in JPEG2000 EBCOT is 256, so the image has to be split into blocks that are at most 256x256. Because of this requirement, for the purposes of this paper, the maximum size of an input image will be 256.

1. For *n* = 2, 4, 8, 16, 32, 64, 128, 256
   *DWT_speed* = *Throughput* / *n*
   a) Use voltage scaling trends to determine the operating voltage, *V*, of DWT processor to make it run at *DWT_speed*
   b) Compute the power consumption of the DWT processor at *V* as *DWT_power*
   c) Record the power consumption of the system from *DWT_power* x *n*
2. Identify *n* that, after including overhead, results in minimum power consumption of the system.

**Figure 9: Finding the most energy efficient parallel design The above algorithm shows how to find the number of parallel units that results in the most energy efficient design. The throughput is constant, so the power consumption corresponds to energy consumption.**

## 4.4 Supply Voltage scaling and splitting the bus

For this experiment, the power consumption of the bus after splitting will be determined for a number of bus widths. The width of the bus is defined as the number of input registers attached to it. First, the power consumption of the simple bus at nominal voltage running at its maximum speed will be determined. The supply voltage($V_{dd}$) of the bus will then be scaled down until the bus's maximum frequency is the target throughput. This $V_{dd}$ will be referred to as the *Throughput* $V_{dd}$. The bus will be split according to the procedure shown in Figure 8. The section labeled overhead in Figure 8 cannot be $V_{dd}$ scaled any further so the supply voltage to it must remain at the throughput $V_{dd}$. The $V_{dd}$ of the input registers can be scaled down further because the effective data rate on the sub bus they are attached to is one half of the original data rate. The input registers are $V_{dd}$ scaled until their maximum frequency is equal to the effective data rate on the sub bus. The bus is then split again and the experiment is repeated. The bus is split until the number of splits equals the width of the bus. This experiment is performed for bus widths of 8,16,32,64,128, and 256.

# 5. RESULTS

## 5.1 Synthesis Results

Table 2 presents the basic synthesis results for the reference pipelined implementation excluding the SRAM module as well as a pair of parallel units excluding the data distribution module. The delay and power consumption reported are from operation at nominal voltage.

## 5.2 Analysis of voltage scaling the parallel units

Figure 10 shows the power consumption of the parallel DWT units, for a given number of parallel units. The throughput is constant at 200MHz, and the graph does not include the data distribution module. Figure 11 shows the $V_{dd}$ that each parallel unit is running at for a given number of parallel units. The power consumption shows diminishing returns as more parallel units are added. However, for these graphs,

| | Pipelined Design | Parallel Design |
|---|---|---|
| Notes | Not Incl. SRAM | 2 parallel units |
| Critical Delay | 1.67ns | 2.97ns |
| Power | 39.3mW @ 200 MHz | 71.8mW @ 336MHz |
| Area | 0.00426mm$^2$ | 0.00716mm$^2$ |

**Table 2: Pipelined and Parallel Results at Nominal Voltage (1.8V)**

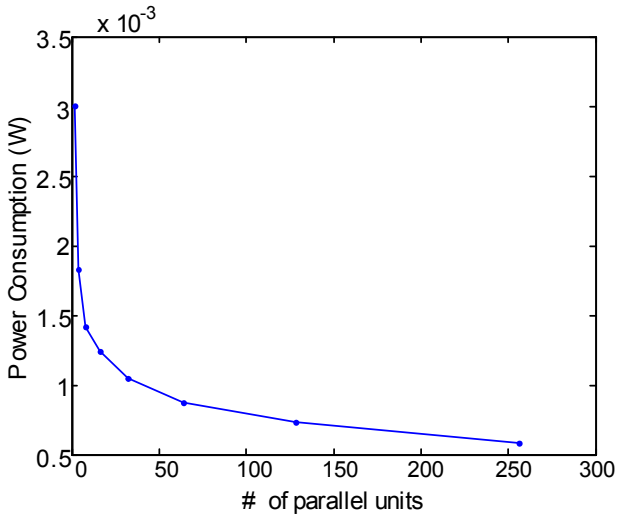


**Figure 10: Power consumption of the parallel units for a given number of parallel units. No data distribution overhead and throughput is constant at 200MHz, which matches the throughput of the pipelined design.**

increasing the number of parallel units is always beneficial as we are not yet including the overhead associated with the data distribution module.

## 5.3 Analysis of voltage scaling the data distribution module

Figure 12 shows the number of splits on the bus vs. the power consumption, for a variety of bus widths. It shows that there is an optimal power consumption for a given number of splits on the bus. Each time a bus is split, a lower $V_{dd}$ can be used to power the registers. Figure 13 shows the $V_{dd}$ of the different buses, for a given number of bus splits. Running a number of different $V_{dd}$s across a design would end up being prohibitive because of routing issues and the need for level converters. For this design, we will find a bus that can run at the same $V_{dd}$ as the parallel units. This will be referred to as the $V_{dd}$ Matched Split Bus. Because of this $V_{dd}$ restriction, there are only two options: the simple bus, which runs at nominal voltage, and the $V_{dd}$ Matched Split bus. Figure 14 shows the power consumption of both a simple bus and the $V_{dd}$ matched bus for parallel unit counts from 1 to 256. This graph shows that the $V_{dd}$ Matched Split Bus is always more optimal no matter the number of parallel

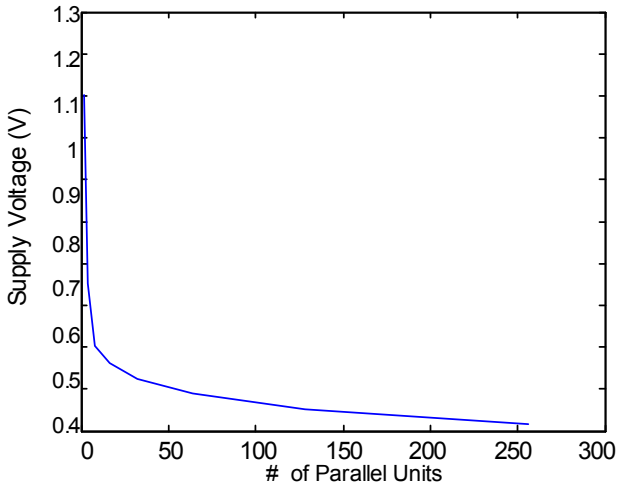


**Figure 11: $V_{dd}$ of the parallel units in Figure 10.**

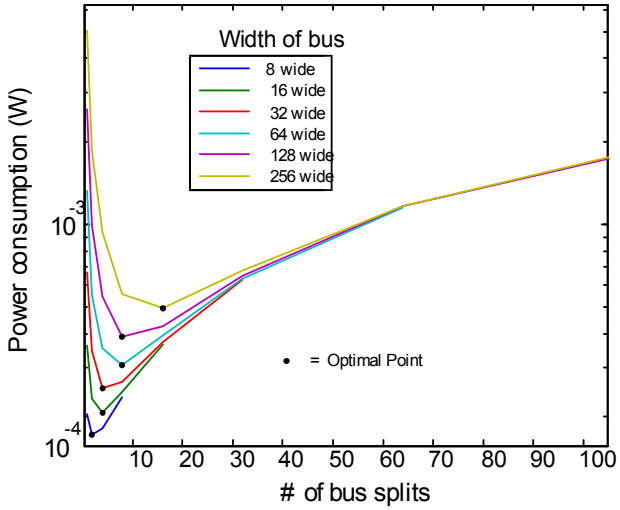


**Figure 12: Number of bus splits vs. power consumption, for a variety of bus widths.**

units. Because of this, the $V_{dd}$ Matched Split Bus will used for all total power consumption calculations.

## 5.4 Striping Penalty

Figure 15 shows the result of the striping penalty in terms of power consumption. It can be seen that as the number of parallel units or width of the design increases, less striping is needed and therefore, less power is wasted for this purpose.

## 5.5 Total Power

Two different total power consumption numbers are calculated. The first power consumption includes the striping penalty, and the second one does not. Equation 6 shows how the total power is calculated when striping is included.

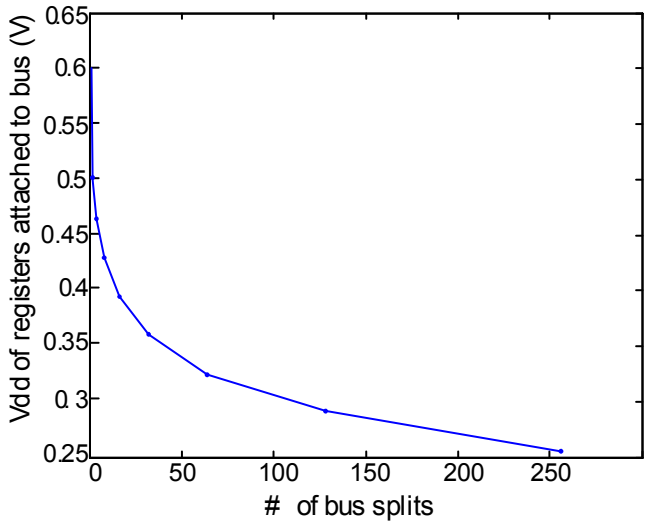


**Figure 13: Optimal $V_{dd}$ of the registers attached to the buses in Figure 12. The $V_{dd}$ is dependant on the number of bus splits, not the width of the bus.**

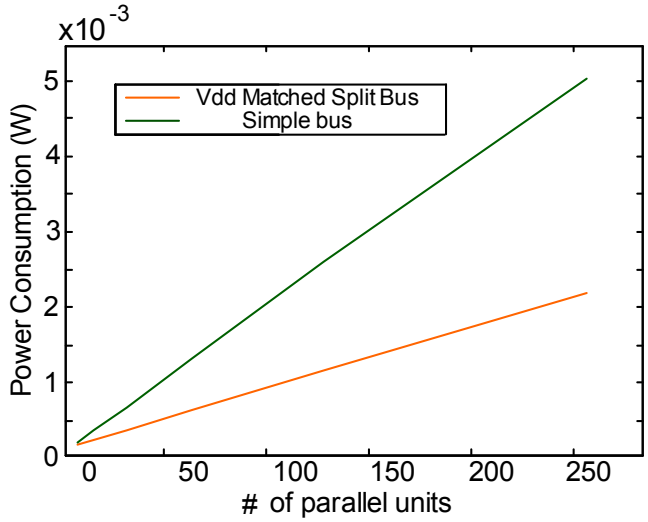


**Figure 14: Simple Bus vs. Split Bus The green line shows the power consumption of a simple bus with no splits for a given number of parallel units. The orange line shows the power consumption of a split bus that is matched to the $V_{dd}$ of the parallel units.**

$$TotalPower = ProcessorPower + VddMatchedSplitBusPower + StripingPenalty \quad (6)$$

## 5.6 Comparisons

Table 3 shows the results of the optimal parallel design, with the striping penalty included, compared to a pipelined design that stripes and one that does not. The $V_{dd}$ scaled parallel design gives a 26.3x reduction in power compared to

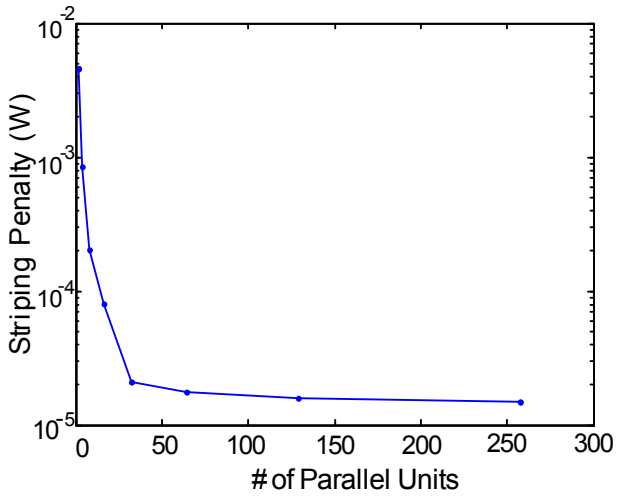


**Figure 15: Striping penalty As the number of parallel units increases, less stripes are required so the striping penalty decreases.**

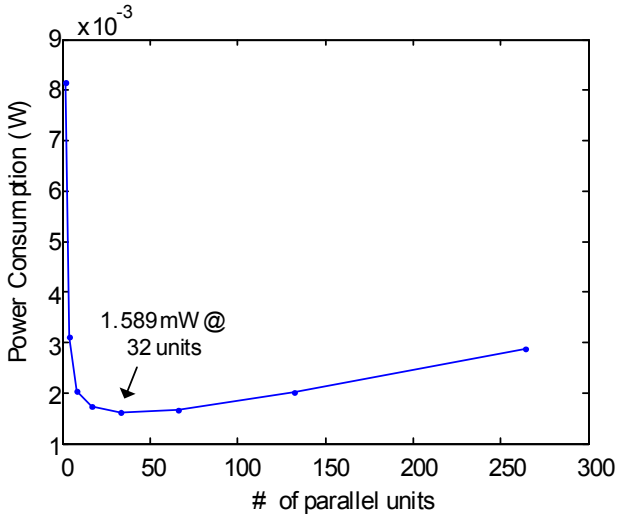


**Figure 16: Total power consumption of the proposed parallel architecture. The throughput of the parallel system is matched to the pipelined system that is not striping (and uses a large RAM bank), so the bus of the parallel system runs faster to match it.**

the reference pipelined implementation. If the pipelined design is not striping and uses a 256 word memory bank, then the optimal parallel design takes up roughly half the area of the pipelined design. If the pipelined design is striping with 32 pixel wide stripes, then the area of the optimal parallel design is roughly 4 times that of the pipelined design.

## 6. RELATED WORK

[22] presented a 180mv FFT architecture that runs in the subthreshold region. The entire architecture runs at subthreshold voltage and the resulting throughput is very low

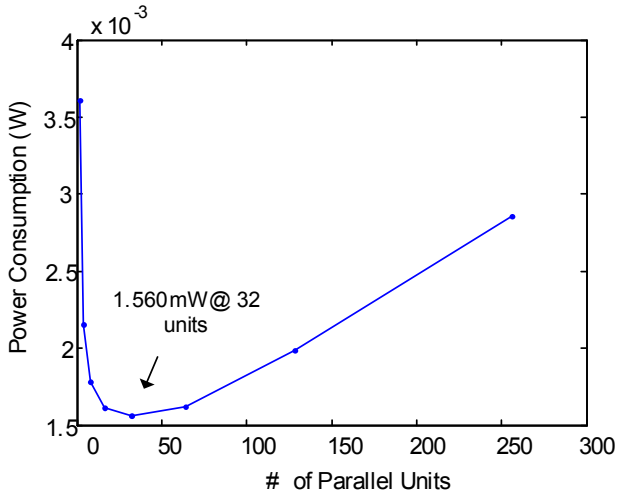


**Figure 17: Total power consumption of the proposed parallel architecture. The throughput of the parallel system is matched to the pipelined system that is striping (uses a small RAM bank). Since they are both striping, they both run at the same clock speed.**

**Table 3: Optimal Parallel Design and the Pipelined Design, Throughput = 200MHz**

| | Parallel Design | Pipelined No Striping | Pipelined Striping |
|---|---|---|---|
| # of Units | 32 | 1 | 1 |
| Processor Power | 1.59mW | 39.3mW | 39.3mW |
| SRAM Power | n/a | 9.80mW | 1.72mW |
| Total Power | 1.59mW | 49.1mW | 41.0mW |
| Processor Area | 0.121mm$^2$ | 0.0426mm$^2$ | 0.0426mm$^2$ |
| Bus Area | 0.0630mm$^2$ | n/a | n/a |
| SRAM Area | n/a | 2.129mm$^2$ | 0.258mm$^2$ |
| Total Area | 1.208mm$^2$ | 2.172mm$^2$ | 0.300mm$^2$ |

due to a 164 Hz clock cycle at 180mV and 10kHz at 350mV. Our design, on the other hand, parallelizes the processing to achieve a throughput typical of nominal voltage signal processing ASICs. [4] presents a parallel DWT architecture, but they do not consider power consumption as a design factor.

## 7. CONCLUSION

In this paper, we presented a new parallel architecture of a 2D lifting-based discrete wavelet transform processor. By increasing the number of parallel units and implementing voltage scaling, this design consumed 1.59mW compared to the 39.3mW of a single state of the art pipelined DWT processor, with both running at the same throughput. This is a 26x reduction in power and the operating voltage of our design was in the subthreshold region of 386mV. The data delivery to the parallel system was also explored. It was found that by adding registers and a demultiplexer to the data delivery system, the supply voltage to certain portions

of the bus could be run at the $V_{dd}$ of the parallel units, and the overall power consumption could be reduced further. Future work would include doing a complete ASIC design and taping out a chip. This would provide a more accurate real world model of how a scaled down supply voltage affects the power consumption. Moreover, we intend to explore the same approach of allowing subthreshold voltage usage in other parallel applications.

# 8. REFERENCES


[1] Oklahoma state university standard cell library

[2] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *HICSS '95: Proceedings of the 28th Hawaii International Conference on System Sciences(HICSS'95)*, page 288, Washington, DC, USA, 1995. IEEE Computer Society.

[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473-484, Apr. 1992.

[4] C. Y Chen, Z. L. Yang, T.-C. Wang and L.-G. Chen A Programmable Parallel VLSI Architecture for 2-D Discrete Wavelet Transform *J. VLSI Signal Process. Syst.* , 28(3):151–163, 2001.

[5] I. Daubechies The wavelet transform, time-frequency localization and signalanalysis *Information Theory, IEEE Transactions on* 36(5):961–1005, Sept. 1990.

[6] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *J. Fourier Anal. Appl.*, 4(3):245-267, 1998.

[7] R. G. Dreslinski, B. Zhai, T. Mudge, D. Blaauw, and D. Sylvester. An energy efficient parallel architecture using near threshold operation. In *Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on*, pages 175-188, Sept. 2007.

[8] S. Hanson, M. Seok, D. Sylvester, and D. Blaauw. Nanometer device scaling in subthreshold circuits. In *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, pages 700–705, San Diego, CA, USA, June 2007.

[9] N. Jayakumar and S. P. Khatri. A variation tolerant subthreshold design approach. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 716Ű719, New York, NY, USA, 2005. ACM Press.

[10] J. Kao, S. Narendra, and A. Chandrakasan. Subthreshold leakage modeling and reduction techniques. In *Proc. International Conference on Computer-Aided Design*, Nov. 2002.

[11] J. Keane, H. Eom, T.–H. Kim, S. Sapatnekar, and C. Kim. Subthreshold logical effort: a systematic framework for optimal subthreshold device sizing. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 425–428, New York, NY, USA, 2006. ACM Press.

[12] C. H. I. Kim, H. Soeleman, and K. Roy. Ultra-low-power DLMS adaptive filter for hearing aid applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*,11(6):1058–1067, Dec. 2003.

[13] T.–H. Kim, H. Eom, J. Keane, and C. Kim. Utilizing reverse short channel effect for optimal subthreshold circuit design. In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 127-130, New York, NY, USA, 2006. ACM Press.

[14] J. D. Meindl and J. A. Davis. The fundamental limit on binary switching energy for terascale integration (TSI). In *IEEE JSSCC vol. 35*, Feb. 2002.

[15] L. Nazhandali, M. Minuth, B. Zhai, J. Olson, T. Austin, and D. Blaauw. A second-generation sensor network processor with application-driven memory optimizations and out-of-order execution. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 249-256, New York, NY, USA, 2005. ACM.

[16] L. Nazhandali, B. Zhai, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, T. Austin, and D. Blaauw. Energy optimization of subthreshold-voltage sensor network processors. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 197—207, Washington, DC, USA, 2005. IEEE Computer Society.

[17] A. Raychowdhury, B. Paul, S. Bhunia, and K. Roy. Computing with subthreshold leakage: device/circuit/architecture co-design for ultralow-power subthreshold operation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(11):1213–24, 2005/11/.

[18] A. N. Skodras, C. A. Christopoulos, and T. Ebrahimi JPEG2000: The upcoming still image compression standard. In *Proceedings of the 2000 ACM workshops on Multimedia*, pages 45–49, 2000.

[19] V. Sze, R. Blazquez, M. Bhardwaj, and A. Chandrakasan. An energy efficient sub-threshold baseband processor architecture for pulsed ultra-wideband communications. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 3, Toulouse, 2006.

[20] K. C. B. Tan and T. Arslan. Low power embedded extension algorithm for lifting–based discrete wavelet transform in JPEG2000. *Electronics Letters*, 37(22):1328–1330, Oct. 2001.

[21] H. Varshney, M. Hasan, and S. Jain. Energy efficient novel architectures for the lifting–based discrete wavelet transform. *IET Image Processing*, 1:305-310, Sept. 2007.

[22] A. Wang and A. Chandrakasan. A 180-mv subthreshold FFT processor using a minimum energy design methodology. *IEEE Journal of Solid-State Circuits*, 40(1):310–319, Jan. 2005.

[23] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model, 1996.

[24] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 868–873, New York, NY, USA, 2004. ACM Press.

[25] B. Zhai, R. G. Dreslinski, D. Blaauw, T. Mudge, D. Sylvester Energy efficient near-threshold chip multi-processing. In *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*, pages 32–37, Portland, OR, USA, 2007, ACM Press.