Voltage Island Design in Multi-Core SIMD Processors

Sohaib Majzoub

Dept. of Computer Engineering, American University in Dubai, POBox 28282, Dubai, Dubai, UAE
smajzoub@aud.edu

ABSTRACT

Today, power management is a key design objective in chip fabrication. In this paper, we present a novel approach to reduce power consumption in SIMD based multi-core architectures. Voltage scaling technique is used, by implementing voltage islands, to optimize power and performance tradeoff for the cores. The number of islands and their respective voltage are selected based on the power-delay characteristics of each instruction: slow instructions run at the nominal voltage while fast instructions run at a lower voltage to save power. An image compression algorithm is mapped into the hardware to demonstrate the power reduction. The results show energy savings of 2.0X for the specified application.

1. INTRODUCTION

Today, many embedded systems and System-on-Chip (SoC) platforms are multi-core designs with many processing elements such as RISC processors, DSP blocks, image and video processors, as well as peripheral devices [2][16]. The multiprocessor System-on-Chip (MP-SoC) era has emerged from this trend in order to satisfy the ever-increasing demand for high-performance and flexible computer systems [13]. However, a major challenge that arises is the excessive power consumption of such platforms. One of the reasons for the focus on power is due to the requirements of mobile devices that contains these MP-SoCs with limited battery lifetimes [6][8][10].

With the recent technology improvements and the evolution of the SoC design, on-chip power management and control is used to optimize the power of a system while delivering the required performance [10]. These control units have the ability to adjust voltage levels of the chip dynamically to reduce power [11]. This includes shutting off some of the components, reducing the supply voltage for others, and changing the substrate voltage to control leakage [12]. The problem is to determine the optimal way to reduce the energy consumed by an application [2][9][12][13].

MP-SoC designs are available in both homogeneous and heterogeneous forms [6]. We consider the homogeneous form where an array of processors executes operations in parallel for a given application. The instructions that these processors execute have different power levels and delays [15]. Our approach is to equalize their delays so as to minimize the overall power. In order to accomplish this, we adjust the supply voltage, $V_{\text{DD}}$, of the associated processors. Slow instructions receive a higher $V_{\text{DD}}$ so that they can complete within the clock cycle, while fast instructions receive a lower $V_{\text{DD}}$ so that their power dissipation is reduced.

Processors are grouped column-wise into a set of voltage islands[2]. The supply voltage can be changed independently for each island [1][7][11]. The open question is whether this should be done statically or dynamically. Static assignments provide a different supply levels to different islands in the design before the execution of the application begins. This approach could be used but would require data transfers between islands that execute different instructions. The use of dynamic voltage scaling has been proposed to reduce power. In our approach, the supply voltage for each island is adjusted based on the instruction being performed. However, there is a significant time delay to switch from one supply voltage to another.

In this paper, we investigate such static and dynamic methods to reduce power. In Section 2, we provide the background of the research problem. In Section 3, we present a hardware emulator, the power/delay models, and the scaling methodology used for $V_{\text{DD}}$ selection for each instruction. In Section 4, we demonstrate the power saving by mapping a lossless image compression. Finally, in Section 5, we provide conclusions and future work.

2. BACKGROUND

Over the past few years, power reduction has been accomplished using a variety of methods. One successful approach has been through the use of so-called voltage islands [2]. The concept of voltage islands allows a finite number of supply voltages that can be applied to different blocks of a design, depending on the power requirements of the system. Figure 1 illustrates a design with multiple voltage islands [2][11].

![Figure 1 Multiple Voltage Island Design](image)

Different styles for voltage island design are still evolving according to the literature [1][2]. The level of complexity ranges from the simplest case where different voltage islands run at fixed
voltage levels for a given application, to the most complex case where the voltage supply can change dynamically during run time for different blocks, depending on the workload[7][3].

To provide flexibility in the design process, each core is given a choice of more than one supply voltage under the control of a power management unit. Dynamic power management (DPM) is the ability to change, assign, and manage the supply voltage as well as the substrate voltage, during operation in order to trade off performance and power[1][2][3]. DPM is a methodology that reassigns voltage levels to the underlying hardware to carry out the workload within the performance-power constraint. It selectively powers up or down islands to reduce power, or adjust their substrate bias to reduce leakage of the idle blocks. Figure 2 shows the DPM system assumed in our work[1][5][7].

![Figure 2 Power Management System Model.](image)

The motivation behind the DPM is the non-uniformity of the workload distribution during operation for different blocks in a SoC design. Shutting off and slowing down components in the system depends on the workload and the state of the specified component. In order to build a power management model, multiple power states can be defined[3]. For instance, Sleep mode, Idle mode, and Run mode are three possible states of the blocks in the design[1][7]. This is illustrated in Figure 3. In the Run state, the block is operating with its corresponding VDD value. In the Idle state, the island has no work assigned to it. In Sleep mode, the island is shut down for the application[5][7][13].

![Figure 3 Three-State “System Component Status” Model.](image)

The time required to transition between the different states is important in determining a suitable power management strategy[1][5]. The corresponding times are represented as T1, T2, and T3 in the figure. If these times are prohibitively large, then a static assignment of supply voltages to processors is required. However, if they are all relatively small, then dynamic assignment during runtime is feasible.

Each state has an associated power consumption level. The minimum power is in the Sleep mode with the complete shut down[1]. The Run state has multiple power dissipation levels depending on the supply voltage of the island. The Idle state is used to keep an island available for use later in the application. However, its substrate bias level is adjusted to reduce leakage current while it is waiting[1][5][7].

Careful consideration should be taken when the transition policy[1][5][7]. Typically, the number of processors required will be determined by the application. If this number is less than the total available processor count, then the unused processors will be put to sleep. The rest of the processors will either be running or idle, depending on the instructions. The active processors will have a high or low VDD value. The inactive processors will have a high substrate bias to increase the threshold voltage, thereby decreasing the leakage[16]. Our research objective is to determine the appropriate power management strategy, either dynamic or static, for a set of applications on a reconfigurable MP-SoC platform.

3. POWER MODELING of a SIMD Multi-Core

MorphoSys[4] is one of the few coarse-grain reconfigurable platforms with multiple processors. We view it as a reconfigurable MP-SoC platform. Figure 4 shows the block diagram and architecture of MorphoSys M1 chip.

![Figure 4 MorphoSys Block Diagram](image)

Its structure follows the master-slave processor design. The master processor is a RISC processor called TinyRISC. The slave processors form the reconfigurable part which consists of an 8x8 array of small reconfigurable processors (RPs)1. The other supporting blocks are: the context memory, the frame buffer, and the DMA controller. The frame buffer and the context memory provide the data and instructions, respectively, in a parallel fashion to the RP array. The role of the frame buffer is to improve the data flow between the main memory and the RP array. The context memory contains the context words that are sent to configure the data input, the interconnection, and the operation to be executed in every RP in the array.

The relevant part of the MorphoSys architecture for this paper is the RP array. It has a parallel structure, with a hierarchical communication bus scheme. It is suitable for operations that have certain level of parallelism, regularity, and intensive computations. The execution of the RP array can be configured to be row-wise or column-wise to provide the parallelism needed for the targeted application. The reconfigurability feature in each RP lies in the multiplexing of the data inputs. The RP array has been scaled up from 8×8 in the M1 MorphoSys to 16×16 in our work. The settings on the multiplexers control the data inputs and operations performed, and it is in this sense that the processors are reconfigurable[4].

---

1 In the original papers, the RP array is referred to as the RC array.
In order to evaluate different power reduction schemes, a hardware emulator for this MP-SoC was developed. The emulator is a cycle-based simulator that functionally executes each instruction. The user may specify different $V_{DD}$ and $V_T$ values to the tool and it will report the number of cycles needed and the power dissipated for the mapped application. The emulator is written in C++.

It is imperative that the emulator be accurate enough to carry out meaningful simulations. However, there is a runtime penalty for a high degree of accuracy. In order to balance the runtime vs. accuracy tradeoff, we developed a methodology based on existing tools such as PrimeTime\textsuperscript{TM}, PrimePower\textsuperscript{TM} and HSPICE\textsuperscript{TM}.

### 3.1. Voltage Island Partitioning

Reconfigurable architectures have the promise of high-speed and adaptable hardware, but the dissipation levels are very high in current designs. Lower power levels can be achieved if the characteristics of the application are known in advance. In our approach, the RP array is subdivided into voltage islands, with regions of high-$V_{DD}$ and low-$V_{DD}$, and regions of high-$V_T$ and low $V_T$. For example, for an application requiring a total of 192 processors, 128 can be set to a high-$V_{DD}$ while another 64 are set to low-$V_{DD}$. These 192 processors would have a low-$V_T$ setting. However, whenever a subset of them is idle, they would be given a high-$V_T$ setting to reduce leakage power. Unused processors would be put in the Sleep mode. Such a partitioning of the array is illustrated in Figure 5. The selection of the number of processors depends on the inherent parallelism in the application.

The number of supply voltages is based on the type of instructions being executed in the application and how often they are executed. If an instruction is fast, it can benefit from a lower supply. Slow instructions should use the nominal supply voltage. To categorize the different instructions in this way, the RP unit was implemented in RTL and then synthesized to the gate level. The delay vs. instruction plot for the RP is shown in Figure 6. The plot was obtained with $V_{DD}=1.8V$ for a 0.18µm CMOS technology using PrimeTime\textsuperscript{TM}.

The delays can be clearly grouped into fast and slow instructions. In the first group are the normal arithmetic and logic operations that require less than 10ns of delay. The second category has instructions that involve multiply and accumulate operations with approximately 15ns of delay.

Our task is to determine suitable operating voltages for each group of instructions. Our overall goal was to develop a more
generalized approach that would allow us to scale the delay as a function of $V_{DD}$ for any design. For this purpose, we analyzed a chain of inverters using HSPICE\textsuperscript{TM}. The supply voltage was varied from 0.1V to 1.8V and the delay versus voltage was plotted. These simulations were carried out on 30 different inverter chains, starting with 1 inverter and continuing up to 30 inverters. Figure 7 shows the curves of all 30 inverter chains.

![Figure 5 Power Mask for RP Array](image)

![Figure 6 Delay vs. Instruction](image)

![Figure 7 Inverter Chain Delay versus Voltage Level](image)

![Figure 8 Normalized Delay-Voltage Plot](image)
All the curves were normalized with respect to their respective delays for $V_{DD}=1.8V$. Figure 8 shows the normalized delay versus normalized supply voltage. Interestingly, the normalized delay ratio is independent of the number of the inverters in the chain. In other words, the effect of the voltage scaling on the delay can be determined from the one curve. We can fit this curve to a suitable polynomial function and extract the $V_{DD}$ values for each instruction if we know the delay of the reference point.

After obtaining the reference values from PrimeTime™ and using the fitted polynomial, the required $V_{DD}$ value was obtained for each instruction. In our case, the target delay for the overall system is 15ns. Using this target value, the results shown in Figure 9 were generated.

From this figure, it is reasonable to implement the design using two voltage levels. The first will be at roughly 1.2V and the second at 1.8V. The 1.2V supply will be used for 23 instructions, and the 1.8V supply will be used for the other 7 instructions (those that require 1.2 V or more). Other levels could be chosen, for instance 1.1V, 1.3V, 1.7V and 1.8V. However, the actual implementation cost will determine the maximum possible number of $V_{DD}$ values for the RP array. In our case, the number of voltage islands will be set to two in subsequent experiments.

Using the voltage scaling based on 1.2V and 1.8V, all instructions will have a critical path delay of roughly 15ns.

The average static power dissipation can be obtained for the RP using PrimePower™ or HSPICE™ for the nominal $V_T$ value. The value of $I_o$ can be extracted and entered into the emulator. The static power can be computed for different values of $V_{DD}$ and $V_T$ using Eqn. (2). Note that an RP will either be active in which case the dynamic power will be computed using Eqn. (1), or it is inactive in which case Eqn. (2) will be used. If it is in the sleep mode, the power is set to 0.

### 4. APPLICATION MAPPING

In this section, we will demonstrate the management of the power masks for the multiprocessor array. The power mask features high $V_{DD}$ for slow instructions, low $V_{DD}$ for fast instructions, and high $V_T$ for the idle processors, and finally complete shutdown for the unused processors.

The assignment of voltages to the different RPs will be shown for both the static and dynamic cases. Static assignment is carried out before the application is executed. The supply voltage for each voltage island is established column-wise and remains fixed for the duration of the application. Instructions are directed to the appropriate type based on their required $V_{DD}$ values. However, this implies that data may need to be transferred from one island to another at various points during the execution of the application. Dynamic voltage scaling is another alternative where each island is adjusted as needed based on the next instruction. However, this cannot be carried out quickly enough to switch levels between consecutive instructions. Therefore, it is desirable to have many instructions executed at one voltage level before switching to another voltage level.

In order to compare different power saving schemes, an image processing application, lossless image compression, is mapped manually onto the system. We used dynamic voltage islands in order to determine the maximum possible power improvement. Then, a static approach was used and compared against the dynamic approach. Ultimately, the compiler should handle the power mask definition to determine if a static or dynamic approach is appropriate. In other words, the mapping within the RP array should be carried out by a power-aware compiler. Such as compiler would have knowledge of the RP states of Figure 10. Every state has two variables: the transition time to the specified state and the power consumed while in that state. From the previous section, there are two Run modes: $V_{ddl}=1.8V$ and $V_{ddl}=1.2V$. The idle state savings is realized by switching the threshold voltage to a higher level to save on static power as well.
as data gating to save on dynamic power. It is implemented automatically in the operand field. The Sleep state is a complete shutdown of the RP.

In the dynamic example, we assumed that there is no transition time delay between certain states. That is, switching between \( V_{\text{ddL}} \) and \( V_{\text{ddH}} \) and the Idle modes can be realized in the same cycle. This ideal case will be considered as upper bound for maximum power saving that can be realized for this specific application. However, switching the voltage supply from 0 to 1.8V will take considerable overhead delay. Thus, we assumed that the transition to and from the Sleep mode, \( T_3 \), is not feasible.

The minimum number of RPs per voltage island will be one column, or 16 RPs. Since the RP array has a SIMD architecture with column-row execution, it is more practical to have the minimum granularity of the voltage island as one column.

The lossless image compression is a highly-parallel application that consists of a sequence of multiplications and additions. We considered 24-bit colored bmp image. The equation used in the mapping is:

\[
Q_{(n,i)} = P_{(n,i)} - [0.9P_{(n-1,i)} + 0.07P_{(n-2,i)} + 0.03P_{(n-3,i)}]
\]

where \( Q \) is the new pixel, \( P_{(n,i)} \) is the current pixel and other pixels are the neighboring pixels. Every pixel is represented in 3 bytes so it needs to be processed in 3 RPs. The number of pixels to be processed in parallel in the RP array is \( 4 \times 16 \).

Figure 11 Dynamic Voltage Islands Activity within the RP Array

In the dynamic switching case, the execution required 12 clock cycles to complete. Figure 11 illustrates the cycle-by-cycle switching during the execution. The supply voltages were controlled column-wise between the different states. The figure shows also the columns switching from \( V_{\text{ddL}} \) high to \( V_{\text{ddL}} \) low. Four columns of the RP array remained off during the operation since they were not needed in the application.

Table 1 shows the number of RP columns assigned to each power mode in each cycle, from 1 to 12. Cumulatively, there were 108 columns that used \( V_{\text{ddL}} \) and 36 that used \( V_{\text{ddH}} \). The total power consumption can be calculated by the emulator using these values.

**Table 1. The mode definition for every column per cycle**

<table>
<thead>
<tr>
<th>Cycle Number</th>
<th>Run Mode</th>
<th>Idle Mode</th>
<th>Sleep Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>108</strong></td>
<td><strong>36</strong></td>
<td><strong>48</strong></td>
</tr>
</tbody>
</table>

We now consider the static power mask case. Figure 12 shows the single mask that defines the island assignment used during the execution. This mask was defined based on characteristics of the dynamic activity masks. The \( V_{\text{ddH}} \)-high sandwich was used since there are fewer instructions with the high \( V_{\text{dd}} \) and the transfer of data between islands is facilitated by having \( V_{\text{ddL}} \)-low islands on either side. In the static case, it required 14 cycles to complete instead of 12 cycles as in the ideal case. We now compute the effective energy for the three cases. To illustrate the calculations used in the emulator, we normalized the power for the columns as follows: \( P_{\text{RP column}}(V_{\text{dd}}=1.8V) = 1 \) and \( P_{\text{RP column}}(V_{\text{dd}}=1.2V) = 4/9 \).

**Figure 12 Static Power Mask**

The reference energy value is based on all columns running at the high \( V_{\text{dd}} \) value, \( V_{\text{dd}} =1.8V \), for all 12 cycles:

\[
E(V_{\text{dd}}=1.8) = 16 \text{ columns} \times P_{\text{RP column}}(V_{\text{dd}}=1.8) \times 12 \text{ cycles} = 192
\]

The energy for the dynamic case:

\[
E_{\text{dynamic}} = 108 \text{ columns} \times P_{\text{RP column}}(V_{\text{dd}}=1.2V) + 36 \times P_{\text{RP column}}(V_{\text{dd}}=1.8) = 84
\]

The energy for the static case:
\[ E_{\text{static}} = [9 \text{ columns} \times P_{\text{Power}}(V_{DD} = 1.2) + 3 \times P_{\text{Power}}(V_{DD} = 1.8)] \times 14 \text{ cycles} = 98 \]

Considering the reference energy, the maximum possible energy saving in the dynamic case is 2.3X based on the assumptions given earlier. However, for the static case, it is approximately 2.0X. This demonstrates that static voltage islands are capable of providing the level of power improvements close to the dynamic case for this particular application.

5. CONCLUSION

In this paper, we presented static and dynamic voltage island partitioning technique for a reconfigurable MP-SoC design. We used a novel voltage scaling method to determine the voltages needed for each instruction to operate at the same frequency. We described a methodology for power modeling in our emulator. To demonstrate the power saving, we mapped the lossless image compression into the RP array. We showed that, for one application, it is possible to achieve energy savings close to the case using dynamic voltage islands even with a static voltage island assignment.

REFERENCES


