# *Emerging SoC Performance/Power Challenges and a Dozen Techniques*

Jerry Wong
Technical Marketing
Xilinx, Inc.
San Jose, USA
jerry.wong@xilinx.com

***Abstract*—Many emerging high performance embedded SoCs are based on accelerators ... This presents the unique problem with multiple peer processes and achieving the highest Performance/Power ... We will discuss a fitting architecture, and over a dozen techniques for increasing Performance/Power by using Power Management ... The examples will center around the Xilinx Zynq UltraScale+ MPSoC and Versal ACAP parts.**

***Keywords—System-on-Chip SoC Design Performance Power Software FPGA Hardware Xilinx MPSoC ACAP***

## I. INTRODUCTION

The topic deals with Performance/Power, and reveals over a dozen methods to achieve it … 16 methods to be exact.

Why didn't you mention my favorite Power Management feature (e.g. Linux Driver Runtime Power Management or DDR Self-Refresh)? In a 30-minute talk, the purpose is not to discuss an exhaustive list, but a selection chosen to spark your own innovation, and to highlight a few clever solutions.

So, why Performance/Power? Certainly, there are some edge applications that require it for considerations such as battery, power over Ethernet, and Energy Star compliance. But have you also considered Automotive, where air cooling demands low power modes when parked? Or considered the PCIe and other limits to the power spec? Or considered the total cost of ownership of a cloud server that includes the total power consumed and cooling costs?

Performance/Power is stated to be important to our users … In a survey of Xilinx customers, performance/power was the #3 reason why they chose Xilinx (Fig. 1) …

A modern SoC can be composed of a group of processors and a group of accelerators. This system can already have an inherent Performance/Power advantage due to its optimized accelerators. But to increase this advantage, the challenge can be management of shared resources (memory, clocks, I/O) (Fig. 2). Which processor should be aware of the state of all the others, and does that one master need to stay "On" to manage? This challenge

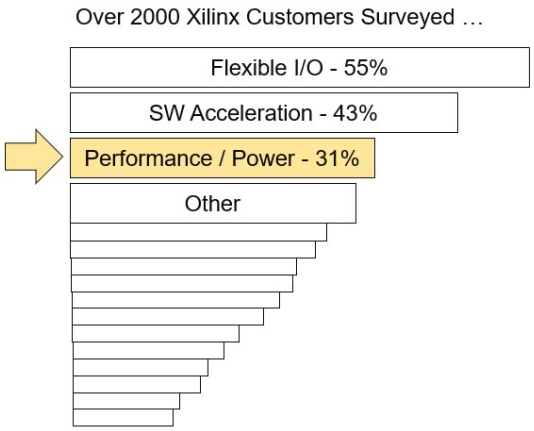


Fig. 1. Performance/Power one of the top customer reasons for choosing

will be addressed during the discussion of the various Performance/Power optimization methods.

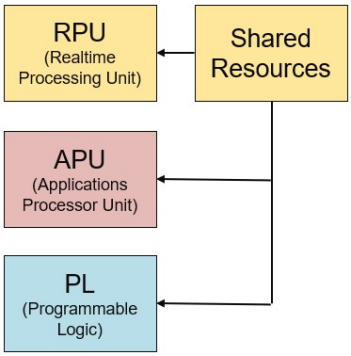


Fig. 2. Modern SoC needs to manage shared resources

We will start with 11 Processing Unit (PU) methods, followed by 5 Programmable Logic (PL) methods, and finally one "bonus" method.

## II. Eleven Processing Unit (PU) Methods

### A. Method 1 - Turn off what you are not using

Ideally this can be done automatically. Your Linux drivers can be written so that the driver "probe" would inform the system that the hardware should automatically be turned on.

```
zynqmp_gpd_probe
 pd->gpd.attach_dev = zynqmp_gpd_attach_dev;
 pd->gpd.detach_dev = zynqmp_gpd_detach_dev;
```

https://github.com/torvalds/linux/blob/master/drivers/soc/xilinx/zynqmp_pm_domains.c

### B. Method 2 - Consolidate decision making

By having one unit aware of the states of all other units, this simplifies the task for all the others. But that one unit must be "on" to control the others, so your system has that minimum power. Additionally, the reliability of that unit should match the required reliability of the system. Therefore, it makes sense to dedicate a low-power and triple-redundant processor to the task, a Platform Management Unit (PMU) (Fig 3).

### C. Method 3 - "Suspend" How far off is "Off"

By having an external unit manage the switching, we can achieve a lower processor power than its Wait For Interrupt (WFI). In this lower power "suspend" state, we can also turn off peripheral devices. Additionally, we do not need to worry about the processor receiving an interrupt. It can be fully turned off, with the PMU receiving the interrupt. On interrupt, the PMU would power the processor back "On" and inform it that it has received a "Resume" rather than a "Boot" (Fig 4).

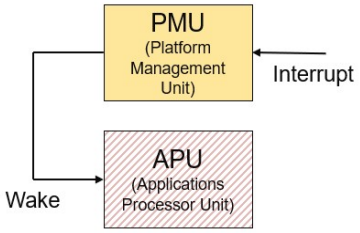


Fig. 3. "Suspend" lets PMU receive interrupt, a lower power solution

### D. Method 4 - Domain Switching saves static power

So how do you save static power? If an entire logical block (processor and peripherals) is not used, and that logical block can be turned off at its Voltage Regulators (e.g. PMICs) it can be considered a Power Domain (Fig 5). This requires some foresight in the partitioning into Power Domains, but can save significant power.

### E. Method 5 - Heterogeneous Architecture

A Heterogeneous Architecture means that a task can be accomplished by dissimilar Processors and/or Accelerators. This allows the designer to mix-and-match performance according to the optimum type for each sub-task. The innovation here is to architect the SoC so the processors and accelerators use the same multi-ported DDR. By sharing the same DDR, the data does not need to be buffer-copied when we wish to "hand off" the sub-tasks between Processors and Accelerators (Fig 6).

### F. Method 6 - CPU Hotplug

Multi-core processors can shut down unused cores. This familiar CPU concept has a place on an SoC. The SoC can also take advantage of the Linux "CPU Idle" feature that automatically turns "Off" cores when they are not in use (Fig 7).

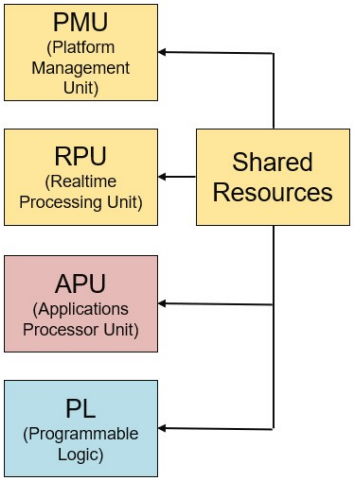


Fig. 5. Platform Managemen Unit (PMU) consolidates decision making

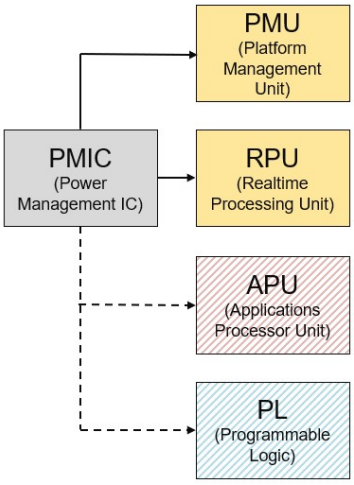


Fig. 4. Power Domains can be turned off at the voltage regulators to save static power

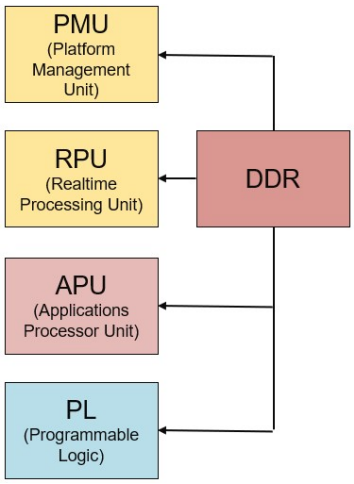


Fig. 7. Heterogeneous systems sharing the DDR can avoid buffer copies when performing parts of the same task

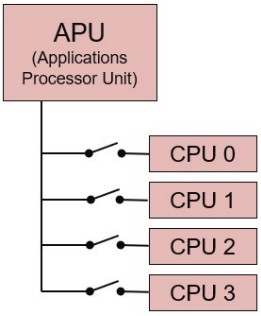


Fig. 8. Idle processor cores can be turned off manually or automatically

### G. *Method 7 - Frequency Scaling*

Frequency scaling slows the processors when performance is not needed. The innovation here is to design your dividers to be glitch-less, so switching frequencies can be safely done while processing (Fig 8).

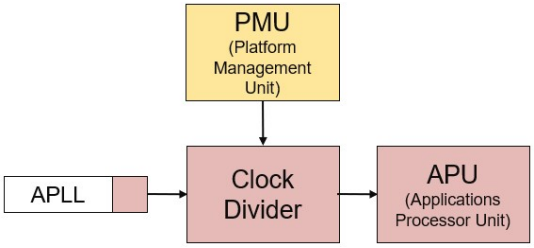


Fig. 6. Glitch-less dividers for switching frequencies while processing

### H. *Method 8 - Wait For Interrupt (WFI)*

This needs to be mentioned, since it is often missed in coding corner cases. In general, plan for the power management features you need, and remember to code their corner cases.

### I. *Method 9 - Duty Cycle*

Does your use case allow you to turn off the processor between processing? A process that is "Off" 90% of the time can save considerable power. The platform should be architected to minimize the suspend and resume latencies (Fig 9).

### J. *Method 10 - Consolidate PLLs*

Complex SoCs can have several PLLs for design flexibility. If you can design with multiples of the same frequencies, you can use fewer PLLs. Each PLL saved can be 12 mW to 20 mW (Fig 10). Switching PLLs can be architected to be glitch-less, but changing PLL frequencies requires settling, so is not inherently glitch-less. The PLL frequency can still be changed while processing by switching all users to a stable clock source while the PLL frequency is settling.

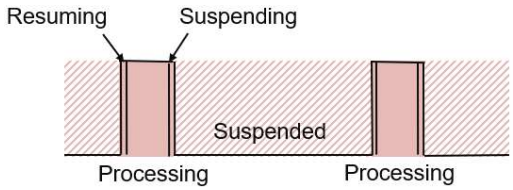


Fig. 9. Duty cycle possible by minimizing suspend and resume latencies

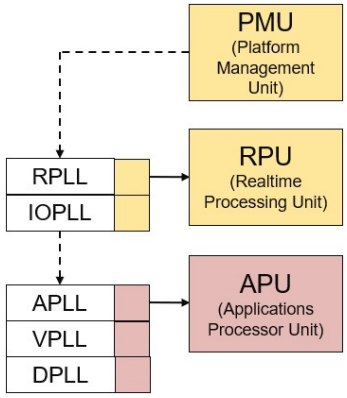


Fig. 10. Multiple PLLs can be configured while processing with glitch-less switching

### K. *Method 11 - Interconnect performance*

Advanced SoCs have multiple clock domains. Interconnects allow data to communicate between clock domains. These interconnects use a moderate amount of power just for clocking, even when no data is flowing. If your system state requires little data flow, the interconnect clocks can be scaled down until needed. If the interconnect clock is scaled down during suspend, it can be scaled up again during the process of resume. This trades off a good suspend power with a good resume latency (Fig 11).

To conclude the Processing Unit (PU) methods, we have a table of typical PU power states. Note the variable power benefit, and resume latency times.

TABLE I. TYPICAL PU POWER STATES

| PU State | Parameters | |
|---|---|---|
| | ***Power*** | ***Undo Latency*** |
| Full Performance ZU9 | 3240mW 100% | |
| Turn off Unused Cores | 1920mW 81% | 86ms |
| Frequency Scaling | 1800mW 75% | 70us |
| APU Suspend | 955mW 40% | 15ms |
| FPD Off | 345mW 14% | 79ms |
| RPU Suspend | 325mW 13% | immeasurably small time |
| Deep Sleep | 30mW 1% | 129us |

[a.] Note: Measurements taken on a single ZCU102 board running the example code

Here is a link to a pre-built design example with source code. "Generated binaries for reference … Petalinux 2019.1 generated images"

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841757/ZU+Example+-+Typical+Power+States

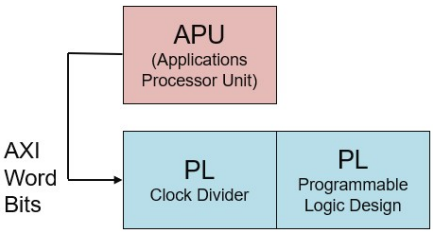


Fig. 13. Clock domain interconnects can be scaled down if maximum throughput is not needed

## III. FIVE PROGRAMMABLE LOGIC (PL) METHODS

### A. *Method 12 - Partial Reconfiguration Dynamic Function eXchange (DFX)*

DFX allows you to change your Programmable Logic (PL) design. This can be beneficial if you can (1) fit into a smaller part and benefit from the lower static power, or (2) swap a design to a lower power "standby design" (e.g. less resolution), or (3) eliminate an entire module by repurposing an existing module. The innovation with DFX is the ability to switch out a portion of the design while the rest of the design continues to run (Fig 12).

### B. *Method 13 - Hardened Cores*

Optimized "Hardened Cores" can save power over new designs if the generic solution is good enough. The dynamic power is dependent on switching so it may be like the dynamic power of a good custom design (Fig 13).

### C. *Method 14 - PL Clock (Frequency) Scaling*

If full performance is not needed, dynamic power can be saved by reducing the clock frequency. This can be implemented by having a clock divider in PL. The control bits of the clock divider can then be mapped to the bits of an Advanced eXtensible Interface (AXI) so they can be controlled by a processor (Fig 14).

### D. *Method 15 - Clock Gating*

You can switch clocks to portions of your circuit. Dynamic power is saved if that portion of the design is not being clocked and is not switching. Typical applications include uni-directional data only requiring half the design, or lower

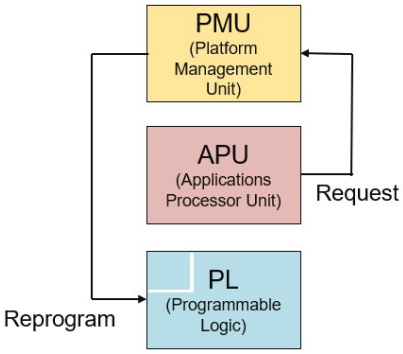


Fig. 11. DFX allows reprogramming a portion of PL while the rest continues to run

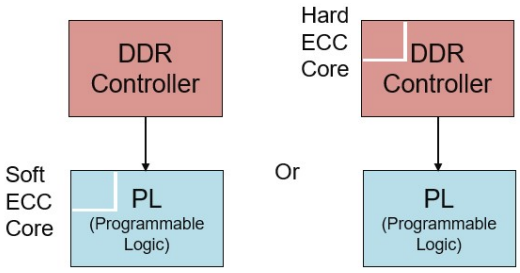


Fig. 12. Optimized hardened cores can save a little

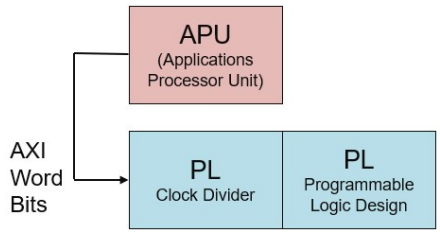


Fig. 16. PL clock divider controlled by AXI

resolution requiring a narrowed bit width. Again, this clock gating can be controlled by a processor and the bits of an AXI (Fig 15).

To conclude the Programmable Logic (PL) methods, we have a table of typical PL power states. Note the variable power benefit, and the implied scaling of the performance.

TABLE II. TYPICAL PL POWER STATES

| PL State | Parameters | |
|---|---|---|
| | *Power* | *Notes* |
| Full Performance ECC | 1970mW 100% | 200MHz |
| Hardened Cores | 1840mW 93% | Few seconds DFX |
| Frequency Scaling | 1120mW 57% | 50MHz |
| Clock Gating | 994mW 50% | ¼ being clocked |
| Idle | 770mW 39% | Static PL power |
| PLD Off | 0mW 0% | Can reload from RAM |

[b] Note: Measurements taken on a single ZCU102 board running the example code

Here is a link to a pre-built design example with source code. Section 4.1: 2019.1 ZCU102 (PL) Download

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841681/Zynq+UltraScale+MPSoC+Power+Advantage+Tool+part+1+-+Introduction+to+the+Power+Advantage+Tool

## IV. ONE "BONUS" METHOD

### *A. Method 16 - New performance/power modes reached*

The final method is to architect a specific design for a family of applications. Moore's Law is slowing down the benefit of shrinking silicon, so there is an increased focus on squeezing more performance by design. The following example of the Xilinx Versal ACAP AI Engine (AIE) solves significant vector processing problems with 6x – 10x performance and with an incremental improvement to performance/power (Fig 16).

AIE achieves these benchmarks by being an optimized Vector Processing System (all memory local), and by being adjacent to Programmable Logic (PL). This allows the bulk of a problem to be efficiently solved by the AIE, and its corner cases solved efficiently by PL (Fig 17).

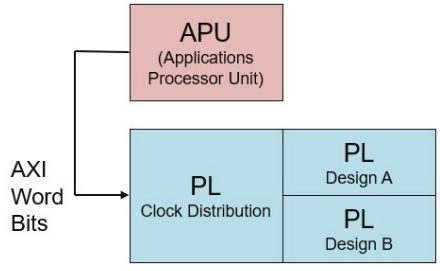


Fig. 17. PL clock gating controlled by AXI

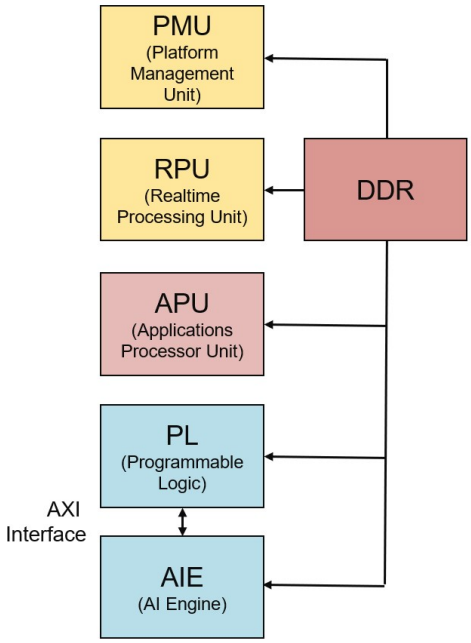


Fig. 14. New accelerators to achieve 6x-10x existing performance with incremental improvement to power