Abstract—Static leakage power consumption is critical in modern FPGAs for many applications. Dynamic Power-Gating (DPG), in which parts of the FPGA in-use logic blocks are powered-down at run-time, is a promising technique to reduce the static power. Adoption of such emerging DPG enabled FPGA architectures remains challenging as the current tool-chains to program the FPGA does not support this type of power-gating. Moreover, manually identifying profitable power-gating opportunities in an application requires significant design expertise and is time consuming. In this paper, we propose a high-level synthesis-based design framework that exploits the dynamic power-gating feature of the FPGAs to minimize the static power dissipation. We use this framework on a set of CHStone benchmark suite and demonstrate that power-gating opportunities for hardware accelerators can be identified in an automatic way. Results show that up to 96% reduction in static energy is achieved for individual accelerators using dynamic power-gating technique.

Index Terms—Run-time/Dynamic Power Gating, High-level Synthesis, FPGA, Static Leakage Power

I. INTRODUCTION

Static-power dissipation has become an important concern for many FPGA applications, and significant work has been done developing low-power FPGAs and CAD [1]–[4]. Recently, an academic FPGA architecture [1] has demonstrated how parts of the chip can be turned off dynamically at runtime, depending on the behaviour of the application. This technique is called Dynamic Power Gating (DPG), and is especially effective for designs in which blocks have long idle times. If used correctly, DPG has the potential to significantly reduce the leakage power of many designs.

The ability to use DPG, however, depends on the designer’s ability to detect and exploit power gating opportunities in the design. In ASIC design, where power-gating is common, the designer would typically manually identify power-gating opportunities based on his or her knowledge of the application and domain. In the FPGA realm, where faster design cycles are desired, an automatic way of detecting power-gating opportunities is essential in order to take advantage of such an architecture.

 Automatically identifying power-gating opportunities as part of a FPGA CAD tool is difficult. Although it is possible to examine the dataflow graph or netlist of an application to try to find parts of the chip that are not needed for periods of time, these techniques typically identify very fine-grained power-gating opportunities, as short as several cycles [5]–[8]. Realistic DPG architectures, however, have significant overhead when parts of the chip are turned off and on; powering up internal capacitances requires significant power, and the need to throttle power gating circuitry to avoid in-rush current problem adds delay overhead [9]. Thus, these architectures are not well-suited to take advantage of the fine-grained power-gating opportunities uncovered by these previous CAD techniques.

To take advantage of power-gating technique in FPGAs, therefore, it is necessary to uncover much more coarse-grained opportunities – regions of the circuit that can be powered down for significant clock-cycles. Intuitively, these regions do exist in a circuit, but they are difficult to detect in an automatic way. In most cases, the high-level behaviour of the circuit will depend heavily on the temporal behaviour of input signals, and this is not encoded in the data-flow-graph (DFG), netlist, or any other form of circuit specification.

In this paper, we propose identifying power-gating opportunities at a higher level to exploit DPG-enabled FPGAs for saving static leakage power. In particular, we focus on SoC designs that are created using a High-Level Synthesis (HLS) methodology, and identify opportunities during HLS. A typical SoC design created using an HLS methodology, such as [10], contains a processor with one or more hardware accelerators used to speed up critical portions of the algorithm. We consider power gating at the accelerator-level as it is well-suited as a unit of granularity for power-gating architectures such as the one in [1]. HLS scheduling already has information about the temporal behaviour of the accelerator and so can make informed decisions regarding how often or how long an accelerator is likely to be idle. Specifically, the contributions of this paper are:

1) An HLS based design methodology for dynamic power-gated FPGAs that automatically identifies the profitable power-gating opportunities, automatically extracts the power-state controller to marshal the identified power-gating opportunities at run-time and finally maps the application to the power-gated FPGA architecture. A design tool framework that demonstrates the proposed methodology is discussed in Sec. III.

2) Quantification of the possible static energy savings achieved through power-gating in CHStone benchmark suite. Sec. IV discusses the experimental results.
II. CONTEXT

Although our work will apply to any FPGA which provides dynamic power control, and any high-level synthesis tool, we describe it in the context of the Dynamic Power-Gated FPGA architecture from [1] and the LegUp high-level synthesis tool from [10]. In this section, we present background about each.

A. Dynamic Power-Gating Architecture

The DPG architecture from [1] is a typical island-style FPGA in which the logic and routing fabric have been augmented with header switches and associated control logic that allow regions of the chip to be selectively powered-down, under the control of signals from elsewhere on the chip (typically from a power-state controller). Each of these regions consists of a small number of CLBs, as shown in Fig. 1. Each region is turned on or off as a unit. The flip-flops within each logic element are not turned off as this allows for a more rapid power-up sequence since state is retained. We also do not consider power-gating the memories or DSP blocks, although the architecture could easily be extended to incorporate this.

B. LegUp High-Level Synthesis Framework

Our work is based on LegUp: A high-level open source synthesis framework [10]. LegUp automatically generates an SoC consisting of a MIPS processor and one or more accelerators from an application design expressed in C. The application is first profiled on a hardware profiler to identify the functions that would benefit from hardware implementation. Based on this information, the tool then compiles each identified function into a hardware accelerator. The portions of the algorithm that are not selected for acceleration are mapped to the MIPS processor and will run as software. These accelerators and a MIPS processor are then combined using an Avalon fabric, creating an accelerated version of the original C code.

III. PROPOSED DESIGN FRAMEWORK

Our proposed methodology to find the power-gating opportunities in an application has three phases: High-level Synthesis (HLS) Guided Phase, Pruning Phase and Mapping Phase, as shown in Fig. 2.

A. HLS Guided Phase

The goal of this phase is to identify all the idle time slots (the power-gating opportunities) in an application. There are two types of power-gating opportunities: accelerator-level opportunities and intra-accelerator opportunities. Accelerator-level opportunities occur when an entire accelerator (including all the sub-functions of this accelerator) is predicted to be idle for a period of time. Intra-accelerator opportunities occur when parts of an accelerator are predicted to be idle for a period of time. Due to the power and delay overhead of turning off a region, we focus on accelerator-level opportunities in this work.

In our flow, we identify the accelerator-level power-gating opportunities by combining the scheduling and trace information of each basic-block (BB) in a function. This gives the cycle count spent in each accelerator. We record the start and end time of each invocation of each accelerator, and use this information to calculate the idle time of each accelerator between invocations. Note that the idle time will depend on the pattern of inputs supplied to the algorithm and the impact of power-gating depends on the idle time duration.

B. Pruning Phase

The previous phase produces a list of all potential power-gating opportunities. However, many of these may not be profitable. Thus, for each power-gating opportunity, we must determine whether a power-gating event should be generated. To make this determination, we evaluate both the size of the accelerator and the idle time and the overhead suffered by each...
of these events. In general, an accelerator should be turned-off in its idle phase if energy with power-gating, $E_{PG}$, is less than the energy with no power-gating, $E_{NPG}$:

$$E_{PG} < E_{NPG} \quad (1)$$

In (1), the total energy consumed by an accelerator in power-gating mode, $E_{PG}$, is given by (2),

$$E_{PG} = E_{\text{turn-off}} + E_{\text{sleep}} + E_{SB} + E_{\text{turn-on}} + E_{PSC} \quad (2)$$

where $E_{\text{turn-off}}$ and $E_{\text{turn-on}}$ is the energy required to turn-off (and later on) an accelerator respectively. $E_{\text{sleep}}$ is the static leakage power of the accelerator when power-gated during idle time and $E_{PSC}$ is the overhead energy of the power-state-controller. These energies can be calculated as follows:

In (2), $E_{\text{turn-off}}$ is calculated by (3),

$$E_{\text{turn-off}} = P_{\text{turn-off}} \times T_{\text{turn-off}} \times \text{Accelerator size} \quad (3)$$

In (3), $P_{\text{turn-off}}$ and $T_{\text{turn-off}}$ are fixed architecture parameters. $\text{Accelerator size}$, expressed in terms of power-gated regions, is the number of power-gated regions occupied by an accelerator on the fabric. We find this by performing an initial mapping of the accelerator to the fabric, however, estimation techniques could also be used. Similarly, the energy required to turn-on an accelerator is calculated by (4),

$$E_{\text{turn-on}} = P_{\text{turn-on}} \times T_{\text{turn-on}} \times \text{Accelerator size} \quad (4)$$

In (2), $E_{\text{sleep}}$ is calculated by (5),

$$E_{\text{sleep}} = P_{\text{PGR-leakage-OFF}} \times (T_{\text{IDLE}} \times DF) \times \text{Accelerator size} \quad (5)$$

In (5), $P_{\text{PGR-leakage-OFF}}$ is the static leakage power of the power-gated region. $T_{\text{IDLE}}$ is the idle time of an accelerator which is extracted from the schedule, as discussed in Sec. III-A. The addition of power-gating support in an architecture incurs an area and delay overhead which degrades the performance. In order to account for this performance degradation, the execution and idle time period is increased by a Degradation Factor (DF) when calculating the accelerator energy in a power-gating mode. The value of degradation factor is architecture specific.

$E_{SB}$, in (2), is the static leakage power of switch-blocks (SBs), that can be dynamically turned-off, in an accelerator. Significant portion of FPGA static power can be saved by turning-off these SBs in a design which can be found by mapping the accelerator to fabric. $E_{SB}$ is given as,

$$E_{SB} = P_{\text{SB-leakage-OFF}} \times (T_{\text{IDLE}} \times DF) \times SB_{\text{Dyn.OFF}} \quad (6)$$

The last component in (2) is the overhead energy consumed by the power-state controller (PSC) which is calculated by estimating the number of CLBs occupied by the controller. The logic blocks occupied by PSC always remain powered-on and hence dissipate static energy.

Once the profitable power-gating opportunities have been identified, a power-state controller (PSC) is extracted from the accelerator’s schedule which controls the sleep signal of the individual accelerator at run-time. The output of the pruning phase is the RTL of the PSC.

C. Mapping Phase

In the mapping phase, the entire system (including the datapath and power-state controller) is mapped to the DPG FPGA architecture. We used Altera’s Quartus II to generate BLIF files from the LegUP output, ODIN-II [11] for elaboration and analysis of power state controller, ABC for logic synthesis and a modified version (5.0) of the VPR FPGA tool [12] to target DPG enabled FPGA architecture, as shown in Fig. 2.

Since we have not fabricated an FPGA with our power-gating architecture, we do not generate a bitstream. However, the mapping described allows us to estimate the power consumption of the resulting design in order to evaluate the effectiveness of our techniques.

IV. EXPERIMENTAL RESULTS

We use the CHStone benchmarks suite [13] which was developed for C-based high-level synthesis (HLS). The C-based source of each benchmark was provided as input to Legup HLS framework which performs optimization passes. Functions in C code are then synthesized to hardware accelerators. We model the impact of power-gating to these hardware accelerators and quantify the energy savings. Five of the twelve CHStone benchmarks have sufficient idle time in their schedules to warrant power gating; we present results only for these five.

A. Power-Gating Potential

Each benchmark comes with a test set that stimulates the design using realistic input traffic. To determine the potential to reduce static power for our benchmarks, we measure the idle time in each benchmark; the results are shown in Table I. The column labeled Exe.Cyc shows the activity cycles of an accelerator. The column labeled Idl.Cyc shows the number of cycles in which the accelerator remains idle and hence can be potentially power-gated. The idle percentage column, idl%, shows the significant amount of idleness observed in our benchmarks.

B. Power-Gating Savings

Both the size of the accelerator (logic-blocks and switch-blocks) and its idle time determine how much leakage power could be saved if the accelerator is power-gated during its idle phase. To estimate these savings, we first find the number of dynamically power-gated regions (PGRs) and switch-blocks (SBs) for each accelerator in a benchmark by mapping it to the target architecture. This accelerator size information is then combined with its schedule to find the profitable power-gating opportunities using the criteria and equations in Sec. III-B. In Table I, the column $\text{Accelerator $(PGR:SB)$}$ shows the name of the accelerator followed by its number of PGRs and SBs that can be dynamically turned-off. For each accelerator in the benchmark, we estimate three quantities:
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Accelerator (PGR:SB)</th>
<th>Exe.Cyc</th>
<th>Id.Cyc</th>
<th>Idle%</th>
<th>$E_{PG}$ (%)</th>
<th>$E_{ACTIVE}$ (%)</th>
<th>$E_{NPG}$ (%)</th>
<th>Sav%</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>KeySchedule (69:755)</td>
<td>1798</td>
<td>7801</td>
<td>81.26</td>
<td>5.37E-07</td>
<td>2.59E-06</td>
<td>2.26E-06</td>
<td>79.28</td>
</tr>
<tr>
<td></td>
<td>ByteSub_ShiftRow (21:64)</td>
<td>240</td>
<td>9359</td>
<td>97.40</td>
<td>2.93E-08</td>
<td>3.86E-07</td>
<td>3.22E-07</td>
<td>92.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>240</td>
<td>9359</td>
<td>97.40</td>
<td>2.05E-08</td>
<td>2.06E-07</td>
<td>1.65E-07</td>
<td>90.06</td>
</tr>
<tr>
<td></td>
<td>InversShiftRow_ByteSub (16:12)</td>
<td>240</td>
<td>9359</td>
<td>97.40</td>
<td>2.05E-08</td>
<td>2.06E-07</td>
<td>1.65E-07</td>
<td>90.06</td>
</tr>
<tr>
<td>ADPCM</td>
<td>upzero (26:28)</td>
<td>4344</td>
<td>19458</td>
<td>81.40</td>
<td>2.39E-07</td>
<td>8.98E-07</td>
<td>7.3E-07</td>
<td>73.32</td>
</tr>
<tr>
<td>Motion</td>
<td>Flush_Buffer (12:94)</td>
<td>6268</td>
<td>89</td>
<td>99.01</td>
<td>2.36E-07</td>
<td>2.39E-07</td>
<td>2.06E-07</td>
<td>1.16</td>
</tr>
<tr>
<td></td>
<td>Get_motion_code (20:193)</td>
<td>61</td>
<td>6297</td>
<td>99.01</td>
<td>1.72E-08</td>
<td>4.55E-07</td>
<td>3.95E-07</td>
<td>96.21</td>
</tr>
<tr>
<td>SHA</td>
<td>sha_update (115:1079)</td>
<td>256526</td>
<td>1000</td>
<td>0.39</td>
<td>1.04E-04</td>
<td>1.04E-04</td>
<td>9.05E-05</td>
<td>96.36</td>
</tr>
<tr>
<td></td>
<td>sha_transform (63:478)</td>
<td>901</td>
<td>20664</td>
<td>99.64</td>
<td>1.55E-06</td>
<td>4.90E-05</td>
<td>4.29E-05</td>
<td>96.9</td>
</tr>
<tr>
<td>JPEG</td>
<td>Write4Blocks (144:1389)</td>
<td>70560</td>
<td>1159088</td>
<td>94.26</td>
<td>5.10E-05</td>
<td>6.36E-04</td>
<td>5.31E-04</td>
<td>91.97</td>
</tr>
<tr>
<td></td>
<td>decode_block (216:1356)</td>
<td>1021315</td>
<td>208261</td>
<td>16.93</td>
<td>5.9E-04</td>
<td>6.69E-04</td>
<td>5.68E-04</td>
<td>96.4</td>
</tr>
<tr>
<td></td>
<td>huff_make_dhuff.tb (33:312)</td>
<td>3850</td>
<td>1225843</td>
<td>99.69</td>
<td>4.50E-06</td>
<td>1.42E-04</td>
<td>1.24E-04</td>
<td>96.84</td>
</tr>
</tbody>
</table>

$E_{PG}$: This represents the total leakage energy consumed by an accelerator in power-gating mode, in which the accelerator is turned off if it is deemed profitable, on an architecture supporting power-gating. The energy dissipated in all execution and idle phases are added together to estimate $E_{PG}$. By design, our architecture suffers a 10% performance degradation due to the presence of power-gating circuitry; to account for this, we increase the idle and execution time periods by this degradation factor when calculating $E_{PG}$.

$E_{ACTIVE}$: This represents the total leakage energy consumed by an accelerator if it were to run in a non-power-gating mode; keeping the accelerator powered-on during idle phase, on an architecture supporting power-gating. Comparing $E_{ACTIVE}$ to $E_{PG}$ gives an indication of the effectiveness of our power-gating algorithm.

$E_{NPG}$: This represents the total leakage energy consumed by an accelerator if it were to run on a non-power-gated FPGA, keeping the accelerator powered-on during idle phases. Comparing $E_{NPG}$ to $E_{PG}$ gives an indication of the effectiveness of power gating.

All the above mentioned energies are expressed in Joules. As can be observed from column A vs B of Table I, leakage energy savings of up to 96% can be achieved by power-gating an accelerator during its idle phase. It is to be noted that an accelerator must stay in the sleep mode, for certain number of clock cycles, such that the leakage savings compensates the energy penalty for mode transition. The column A vs C of Table I shows the percentage savings of power-gating as compared to the non-power-gating architecture. The negative results for flush_buffer and sha_update accelerators are due to their limited idleness and power-gating overhead. However, for the accelerators with more idleness, $E_{PG}$ savings of up to 96% can still be achieved when compared with $E_{NPG}$.

V. CONCLUSION

We have proposed a high-level synthesis (HLS) based design methodology to uncover the profitable leakage savings opportunities that may exist in an application. In doing so, we have shown an automatic way of extracting the workload idleness using HLS and use this information to build a power-state-controller that marshals the power-gating sequence at run-time. Through our design framework, we demonstrate how these power-saving opportunities exploit the power-gating feature of the target architecture to save static power. Our results show the promising potential of power-gating in real-world benchmark applications.

REFERENCES