2019 29th International Conference on Field Programmable Logic and Applications (FPL)

Quantifying the Benefits of Dynamic Partial Reconfiguration for Embedded Vision Applications

Marie Nguyen, Robert Tamburo, Srinivasa Narasimhan, James C. Hoe
Carnegie Mellon University
Pittsburgh, Pennsylvania

***Abstract*—Dynamic partial reconfiguration (DPR) allows parts of an FPGA to be reprogrammed at runtime (i.e., repurposed). Though DPR has been supported by commercial devices and tools for more than a decade, it has been underutilized, perhaps, due to a shortage of demonstrated use-cases and quantified benefits over static FPGA mapping (without DPR). In this paper, we quantify the benefits of dynamic FPGA mapping (with DPR) over traditional static FPGA mapping for two vision applications deployed on systems with area/device cost, power or energy constraints (i.e., smart car and smart robot). In both applications, the FPGA needs to accelerate multiple tasks at 60 fps. However, all tasks are not required at the same time. In this work, instead of mapping all tasks statically on a large FPGA, the set of tasks needed at a given time is (1) repurposed on a smaller FPGA and (2) still meets the functional and performance requirements (i.e., 60 fps). In the two application examples, we show that dynamic mapping on smaller FPGAs reduces logic resource utilization by up to 3.2x, device cost by up to 10x, and power and energy consumption by up to 30% in comparison with static mapping on larger FPGAs. These benefits are crucial for applications deployed on systems where reducing area/device cost, power and energy is as important as meeting performance requirement.**

I. Introduction

Motivations. Current embedded real-time vision applications need to support many tasks, and are deployed on systems with stringent area/device cost, power and energy constraints. In cost-constrained systems (e.g., smart automotive systems), minimizing manufacturing costs (area/device cost and power/-cooling cost) to maximize profit margin is a first-order priority. In energy-constrained systems that operate on batteries (e.g., robotic systems), energy savings are crucial to prolong battery life and system operation. For these systems, reducing area/device cost, power or energy is as important as meeting performance requirement.

Due to their power/energy efficiency, FPGAs have been increasingly used to accelerate embedded real-time vision applications. Figures 1 and 2 show two examples of vision applications with cost or energy constraints deployed on an FPGA. In both applications, the FPGA needs to support many tasks at 60 fps. However, all tasks are not needed at the same time. In the interactive application example deployed on an automotive Headlight system (Figure 1), the tasks needed at a given time are requested by the user depending on the environment (city vs highway & day vs night) which changes *infrequently (from minute to hour range)*. In contrast, in the navigation application example deployed on a robotic system (Figure 2), the task needed may change *very frequently (every tens of milliseconds)* depending on the number of objects

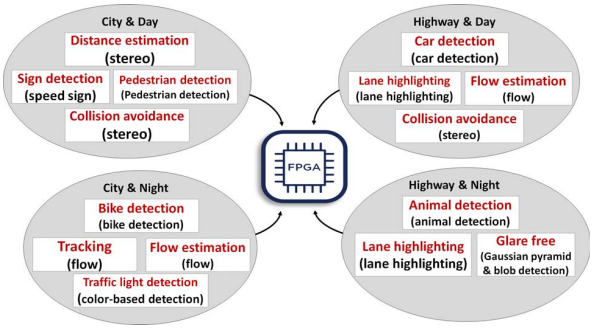

Fig. 1: Example of an interactive application deployed on an automotive Headlight system [1]. The tasks needed (in red) are requested by the user based on the environment (city vs highway & day vs night) which changes *infrequently*. Each task is accelerated by one module (in parenthesis) except for the glare-free task.

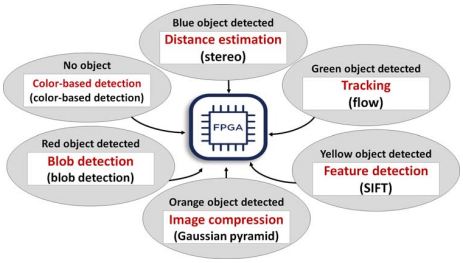

Fig. 2: Example of a navigation application deployed on a robotic system. The task needed (in red) depends on the objects encountered in the scene, and may change *very frequently*. Each task is accelerated by one module (in parenthesis).

encountered in the scene. In both applications, if the total number of tasks required over time is very high, it may not be possible to map all tasks statically on the FPGA. Even when the FPGA is large enough to map all tasks statically, it is inefficient and costly to do so, in terms of area/device cost, power and energy, given that *all tasks are not needed at the same time*. Not only does a larger FPGA dissipate more leakage power, but idle regions of the design also consume area, power and energy unnecessarily. (Leakage and non-leakage power refer to static and dynamic power, respectively.) This inefficiency represents a major problem in cost or energy-constrained situations.

1946-1488/19/$31.00 ©2019 IEEE
DOI 10.1109/FPL.2019.00029

Area, Power and Energy Benefits of DPR. Dynamic partial reconfiguration (DPR) allows FPGA regions, called reconfigurable partitions, to be reprogrammed at runtime (i.e repurposed). Though DPR has been supported by commercial devices and tools for more than a decade, it has been underutilized, perhaps, due to a shortage of demonstrated use-cases [2], [3], [4], [5] and quantified benefits over static FPGA mapping (without DPR). Prior works [6], [7], [8] have shown that vision applications tolerate current DPR overhead (tens of milliseconds to reconfigure a partition) and still achieve realtime rates (30+ fps).

In this paper, we focus on classes of vision applications that benefit from dynamic FPGA mapping (with DPR). These applications share two main characteristics: (1) all tasks are not needed at the same time, and (2) the goal is to meet performance requirement while reducing area/device cost, power or energy. We develop a framework using DPR for dynamic mapping of vision applications on FPGAs. In the framework, multiple tasks can be mapped and accelerated simultaneously on the FPGA. More importantly, the FPGA can be repurposed with different sets of tasks over time. In this work, instead of mapping all tasks required by an application statically on a large FPGA, the set of tasks needed at a given time is (1) repurposed on a smaller FPGA and (2) still meets the functional and performance requirements (i.e., 60 fps). We quantify the benefits of dynamic mapping on smaller FPGAs, in terms of area/device cost, power and energy, over static mapping on larger FPGAs for two applications built on top of the framework: (1) an interactive application (Figure 1) in which the total reconfiguration time is negligible compared to the total execution time and (2) a navigation application (Figure 2) in which the total reconfiguration time is significant compared to the total execution time. Both applications are representative of many rapidly emerging vision and AI-driven applications that could benefit from DPR.

Contributions. Our results support the following conclusions:

- Applications that benefit from dynamic mapping on smaller FPGAs have two main characteristics: (1) all tasks are not needed at the same time, and (2) the goal is to meet performance requirement while reducing area/device cost, power or energy.
- Smaller FPGAs can be used for dynamic mapping, resulting in area/device cost reduction (device cost savings are greater than area savings) over static FPGA mapping.
- Dynamic mapping on smaller FPGAs consumes less power/energy than static mapping on larger FPGAs since (1) the smaller FPGA dissipates less leakage power than the larger FPGA, and (2) the dynamic design is smaller than the static design, uses fewer clocking resources, and therefore, dissipates less non-leakage power. In this paper, we refer to statically mapped and dynamically mapped design as static and dynamic design, respectively.
- When the total reconfiguration time is significant compared to the total execution time (the ratio of reconfiguration to compute may be almost 1:1), the dynamic design still consumes less energy than the static design because the total reconfiguration energy is very low.
- Tasks accelerated individually in our framework have comparable performance to static designs, and are an order of magnitude faster than a software implementation.

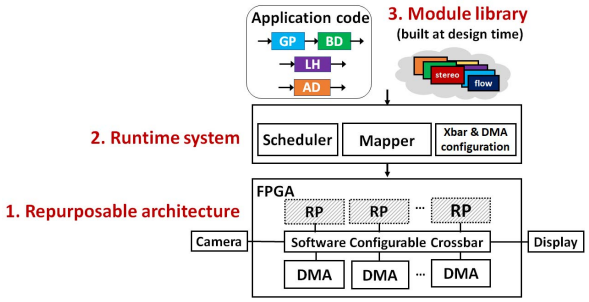

Fig. 3: The framework is composed of (1) a repurposable architecture, (2) a runtime system, and (3) a module library.

II. Framework for Vision

We build two applications on top of our DPR framework. In this section, we briefly present the design and the runtime operation of this framework.

Overview. The framework has three main components (Figure 3): (1) a repurposable architecture in which reconfigurable partitions (RPs) are connected to the external I/O via a software configurable crossbar and DMA engines, (2) a runtime system which manages the repurposing of the FPGA with tasks specified in an application code, and (3) a module library which contains pre-built vision modules that are used to repurpose the reconfigurable partitions. A reconfigurable partition hosts one module at a time. Modules from the library have the same AXI-based I/O interfaces as the reconfigurable partitions.

Runtime operation. In the framework, the process for dynamically executing a set of tasks needed by an application is managed by the runtime system: (1) the runtime system first parses the application code that contains the set of tasks to accelerate, (2) the runtime system assigns the reconfigurable partitions to all tasks, where each task is accelerated by one or more modules from the module library, (3) the reconfigurable partitions are repurposed with selected modules from the module library, (4) the software configurable crossbar and the DMA engines are configured, (5) the runtime system starts the modules. This process is repeated for each set of tasks to accelerate in an application.

III. Implementation

Experimental Setup. For this study, we deploy our framework on the Zynq 706 and the Zynq 702 boards [9] to accelerate the interactive and the navigation application, respectively. Dynamic FPGA mapping provides most benefits if the FPGA has just enough resources to fit the set of concurrent tasks using the highest amount of resources in an application. The Zynq 706 board has an XC7Z045 FPGA which is large enough to fit the set of tasks using the highest amount of resources in the interactive application. The Zynq 702 board has an

XC7Z020 FPGA (smaller than the XC7Z045 FPGA) which is large enough to fit the module using the highest amount of resources in the navigation application.

Two ARM CPUs are available on each board. We use one CPU that runs Linux to load input test images into on-board DRAM, and to save the output of vision applications. The other bare-metal CPU executes the runtime system and the application code which specifies the sets of tasks to accelerate in an application. Each task is accelerated by one or more modules from the module library. The module library is loaded into DRAM at system boot up. The reconfigurable partitions are reconfigured through the processor configuration access port (PCAP). Data is streamed between DRAM and the modules.

Dynamic Design Area. To quantify the area benefits of dynamic FPGA mapping over static FPGA mapping, we first characterize the resource utilization of the dynamic design. Table I reports the resource utilization of the repurposable architecture on the (1) the Zynq 706 board with an XC7Z045 FPGA and (2) the Zynq 702 board with an XC7Z020 FPGA. On the XC7Z045 FPGA, the I/O infrastructure comprises a software configurable crossbar with 16 endpoints and eight DMA engines. We implement four reconfigurable partitions which occupy more than 80% of the FPGA resources. Three reconfigurable partitions have similar size and contain approximately the same amount of logic and memory resources; the fourth partition has approximately three times more resources than the other partitions to fit the largest module needed in the interactive application. When the set of tasks using the highest amount of resources is mapped on the FPGA, the dynamic design (including the I/O infrastructure) consumes 74%, 58%, and 18% of LUT, BRAM, and DSP resources, respectively.

On the smaller XC7Z020 FPGA, the I/O infrastructure comprises two DMA engines. We implement a single reconfigurable partition that is large enough to fit the largest module needed in the navigation application. When the largest module is mapped on the FPGA, the dynamic design (including the I/O infrastructure) consumes 62%, 33%, and 38% of LUT, BRAM, and DSP resources, respectively.

In both cases, most of the FPGA resources are used for compute. The I/O infrastructure uses less than 16% of logic resources. Note that a similar amount of logic resources would be needed to build the I/O infrastructure in a static design.

Individual Module Performance. To accelerate the tasks required in our two applications, we implement the 12 modules shown in Figures 1 and 2 using Vivado HLS [10]. A significant amount of effort was placed into implementing and into ensuring the functional correctness of these modules. More information about the modules can be found in [1] and [11].

The 11 modules used in the interactive application are all clocked at 169 MHz in the framework deployed on the Zynq 706 board. The 11 modules achieve a throughput that ranges between 152 and 167 MPixels/s, which is sufficient to achieve 60 fps for full HD frames. Similarly, the six modules used in the navigation application are all clocked at 169 MHz in the framework deployed on the Zynq 702 board.

In our frameworks, the *stereo* and the *flow* modules achieve a throughput of 152 and 161 MPixels/s, respectively. Our CPU implementations of the *stereo* and the *flow*, based on OpenCV and running on a i7-3770@3.40 GHz, achieve a throughput of 13 and 10 MPixels/s, respectively. In [12] and [13], the authors report a throughput of 148 and 165 MPixels/s for their static FPGA designs of the *stereo* and *flow* modules, respectively, which is comparable to the performance we get.

Repurposing Overhead. The overhead for repurposing the FPGA with a single module is the sum of the time for reprogramming a partition with the module, and configuring the crossbar links and the DMA engines. In practice, the time for configuring the crossbar links and the DMA engines (in the order of hundreds of microseconds) is negligible compared to the time spent for reprogramming a partition. The time for reprogramming a partition is proportional to its size.

In the framework deployed on the Zynq 706 board, the time to reprogram (1) the three smaller partitions is approximately 22.2, 25 and 32.3 ms and (2) the largest partition is 73.4 ms. In the framework deployed on the Zynq 702 board, the time to reprogram a partition is 19.7 ms. Despite the non-trivial time to reconfigure a partition, both applications meet their performance requirements (more details in next section).

IV. Evaluation

In this section, we quantify the benefits of dynamic mapping on smaller FPGAs over static mapping on larger FPGAs for the interactive and the navigation application examples. We show that dynamic FPGA mapping reduces logic resource utilization by 2.5x and 3.2x, device cost by approximately 10x and 4x, and power/energy consumption by 28% and 30% (with impact on cooling cost) compared to static FPGA mapping for the interactive and the navigation application, respectively.

A. Case Study 1: Interactive Application

Overview. In this application, the user can pick one of the four sets of tasks shown in Figure 1. In each set, up to four tasks are requested simultaneously; each set is accelerated by up to four modules. In the framework, when the FPGA is spatially shared by up to four modules, each module achieves 60 fps for full HD images. In this application, the FPGA is repurposed with a different set of tasks within minute to hour range depending on the user's selection. *The total reconfiguration time is negligible over the total execution time.* Therefore, the reconfiguration time, power and energy overheads will not be considered for this evaluation.

Area Model. Before presenting our results, we offer a simple model to reason about the potential area benefits of dynamic FPGA mapping over static FPGA mapping. We consider an application with a total number N_{modules} of modules to accelerate on the FPGA. For this simple model only, we assume that one module is needed at a time. We do not take into account the area utilization of the I/O infrastructure. In the static design, all modules are mapped simultaneously on the FPGA; we use the smallest FPGA on which the static design fits. For the dynamic design, we use the smallest FPGA that fits

TABLE I: Resource utilization of the repurposable architecture on (1) the Zynq 706 and (2) the Zynq 702 board. The percentage of resource utilization is given in parenthesis. In both cases, most of the FPGA resources are used for compute.

	Zynq 706 with four partitions				**Zynq 702 with one partition**			
	I/O infrastructure			**Available for compute**	**I/O infrastructure**			**Available for compute**
	Crossbar	DMA engines	Misc		Crossbar	DMA engines	Misc	
LUT	5580 (3%)	18,270 (9%)	2428 (1%)	185,400 (85%)	0	7388 (14%)	1035 (2%)	26,000 (49%)
BRAM	0	36 (7%)	7 (1%)	500 (92%)	0	14 (10%)	6 (4%)	80 (57%)
DSP	0	0	0	840 (93%)	0	0	0	120 (55%)

TABLE II: Resource utilization of six modules used in the interactive application after place & route on the XC7Z045 FPGA. The percentage of resource utilization is given in parenthesis. The six modules are not evenly sized.

	lane highlighting	**speed sign detection**	**car detection**	**pedestrian detection**	**animal detection**	**bike detection**
LUT	1185 (1%)	16,576 (8%)	84,258 (39%)	80,756 (39%)	81,432 (37 %)	37,122 (17%)
BRAM	16 (3%)	50 (9%)	104 (19%)	99 (18%)	99 (18%)	59 (11%)
DSP	0	10 (1%)	126 (14%)	156 (17%)	234 (26 %)	70 (8%)

TABLE III: Resource utilization of six modules used in the navigation application after place & route on the XC7Z020 FPGA. The percentage of resource utilization is given in parenthesis. The six modules are almost evenly sized (in terms of LUT).

	stereo	**flow**	**color-based detection**	**SIFT**	**blob detection**	**Gaussian pyramid**
LUT	18,396 (34.6%)	17,509 (33%)	16,501 (31%)	23,513 (44%)	20,855 (39%)	19,135 (36%)
BRAM	61 (44%)	60 (43%)	25 (18%)	26.5 (19%)	33 (24%)	47 (34%)
DSP	0	39 (18%)	0	83 (38%)	41 (18%)	55 (25 %)

the largest module requiring the highest amount of resources in the application.

We define $R_{\text{S/D}}$ as the ratio of the amount of resources available on the FPGA used for static mapping to the amount of resources available on the FPGA used for dynamic mapping. A larger ratio represents a greater saving. In the best-case scenario, all modules are evenly sized (i.e. they consume approximately the same amount of resources). In this case, $R_{\text{S/D-bestcase}} = N_{\text{modules}}$. In the worst-case scenario, modules are not evenly sized; the largest module is significantly larger than all other modules combined. In this case, $R_{\text{S/D-worstcase}} = 1$. Finally,

$$1 < R_{\text{S/D}} \leq N_{\text{modules}}$$

In the interactive application, $R_{\text{S/D}}$ is close to the worst-case scenario since modules are not evenly sized (Table II). In the navigation application, even though modules are evenly sized (Table III), $R_{\text{S/D}}$ is not close to the best-case scenario due to the quantization of FPGA sizes. In practice, using the smallest possible FPGAs to map the static and the dynamic designs may not be feasible.

Static Design Area. Since no FPGA is large enough in the Zynq family to map the static design, we use the XC7VH870T FPGA from the Virtex 7 family which is the smallest FPGA on which the static design fits. The XC7VH870T FPGA uses the same process node (28 nm) as the XC7Z045 FPGA. The static design maps 13 modules to accelerate all tasks shown in Figure 1, and has a software configurable crossbar with 25 endpoints, eight DMAs and additional control logic for enabling/disabling module combinations. We clock the design at 169MHz (same frequency as in the dynamic design). Table IV shows the resource and percentage utilization of the static design after place & route on the XC7VH870T FPGA. The XC7VH870T FPGA has 2.5x more LUT and BRAM resources, and 2.8x more DSP blocks than the XC7Z045 FPGA. The XC7VH870T FPGA is approximately 10x more expensive than the XC7Z045 FPGA.

Power Results. We estimate the power consumed by the static and by the dynamic FPGA designs when one set of tasks is active at a time (Figure 1). We use the Xilinx Power Estimation tool [14] for estimating the power consumption of both designs based on their resource utilization after place & route assuming (1) nominal voltage, (2) same switching activity and frequency, and (3) clock gating the inactive part of the static design. Maximum effort for clock gating is applied so that amount of non-leakage power consumed by the inactive part of the design is minimized. (We perform many power measurements on the actual Zynq 706 board when the design is running in steady-state, calibrate the model with our measurements, and find that our power estimations match with the power measurements within 3%.)

Figure 4 reports our estimations of the total power consumed by (1) the static design and (2) the dynamic design for each set of tasks with corresponding percentage error. The power consumption is broken down into the leakage and non-leakage power. The static design consumes 28%, 29%, 30.5%, and 30.5% more total power than the dynamic design for the *City & Day*, *Highway & Day*, *City & Night*, and *Highway & Night* environment, respectively. The static design consumes more power/energy than the dynamic design for two reasons: (1) the larger FPGA dissipates more leakage power than the smaller FPGA and (2) the static design is larger than the dynamic design and uses more clocking resources, resulting in extra non-leakage power consumed. On average, the static

TABLE IV: Resource utilization of the statically mapped design after place & route on the XC7VH870T FPGA and on the XC7Z035 FPGA for the interactive and navigation application, respectively. The percentage of resource utilization is given in parenthesis. In both cases, the statically mapped design consumes more resources than available on the FPGA used for dynamic mapping.

	interactive			navigation		
	I/O infrastructure only	**Modules only**	**Total**	**I/O infrastructure only**	**Modules only**	**Total**
LUT	36,278 (7%)	449,026 (82%)	485,304 (89%)	8423 (5%)	115,899 (67.5%)	126,322 (72.5%)
BRAM	43 (3%)	835 (59%)	878 (62%)	20 (2%)	252.5 (28%)	272.5 (30%)
DSP	0 (0%)	770 (29%)	770 (29%)	0	218 (43.6%)	218 (43.6%)

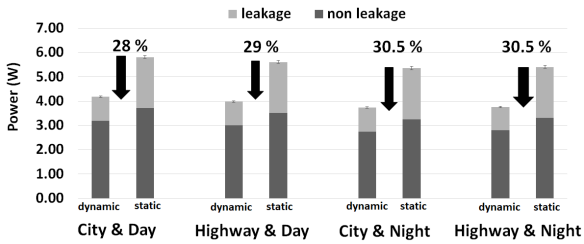

Fig. 4: Power consumed when the design is mapped (1) dynamically and (2) statically in four environments shown in Figure 1. The power savings mainly result from a reduction of leakage power.

design dissipates 50% more leakage power and consumes 14.5% more non-leakage power than the dynamic design.

B. Case Study 2: Navigation Application

Overview. We build a navigation application on top of the framework running on the Zynq 702 board (Figure 2). Figure 5 illustrates the execution of this application during a fixed time interval when the design is mapped statically and dynamically on the FPGA. In the reference static design (Figure 5 (a)), the system monitors the scene for changes. When an event happens (i.e. detection of a colored object), the system processes this event (by executing one of the five possible tasks) and then returns to its monitoring state. In the dynamic design (Figure 5 (b)), when an event happens, an idle phase starts. The active module is turned off before starting the reconfiguration of the partition with one of the five possible modules. The idle phase ends when the partition is reconfigured. In this application, an event needs to be processed in less than 34 ms. Though the static design can return to its monitoring state faster than the dynamic design, the dynamic design meets the application requirement. (The time to reconfigure a partition is 19.7 ms and the average time to process a frame is 12.2 ms.)

Depending on the number of objects to detect in the scene, *the FPGA may be repurposed very frequently.* The total reconfiguration time may not be negligible compared to the total execution time. In the worst-case scenario, the ratio of reconfiguration to compute is 1:1. In this evaluation, we take into account the total energy spent for reconfiguration by the design when mapped dynamically as a function of the number of reconfigurations per second. For a fair comparison, we also consider the conservative case where the compute time is identical in both the static and the dynamic designs (Figure 5 (c)).

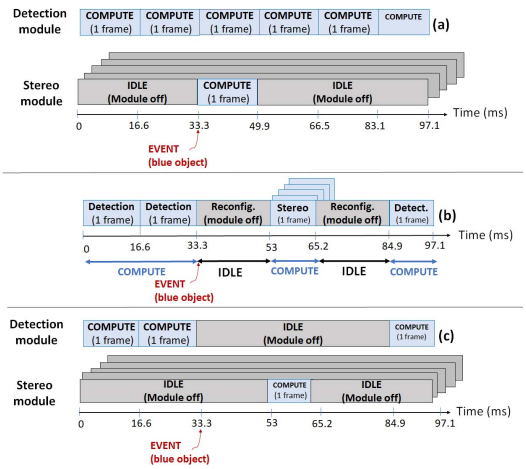

Fig. 5: Example execution timeline of the navigation application when the design is mapped (a) statically (reference case), (b) dynamically and (c) statically (equalized case).

Static Design Area. We use the XC7Z035 FPGA to map the static design since it is the smallest FPGA from the same family as the XC7Z020 FPGA on which the static design fits. The static design maps six modules used in this application, has two DMA engines and a software configurable crossbar with eight endpoints to connect the six modules and the DMA engines. We clock the design at 169MHz (same frequency as in the dynamic design). Table IV shows the resource and percentage utilization of the static design after place & route on the XC7Z035 FPGA. The XC7Z035 FPGA has 3.2x, 3.6x and 4x more LUT, BRAM and DSP resources, respectively, than the XC7Z020 FPGA. The XC7Z035 FPGA is approximately 4x more expensive than the XC7Z020 FPGA.

Energy Model. We develop an energy model to breakdown the total energy into relevant components when designs are mapped statically and dynamically. The model considers the energy expended during a fixed time interval of length t_{interval}. In this time interval, we enforce that the static and the dynamic designs process the same number of events.

The total energy consumed by the reference static design (i.e. the system is never idle) $E_{\text{total, static reference}}$ during t_{interval} is equal to the total energy spent for compute $E_{\text{compute, static reference}}$.

$$E_{\text{total, static reference}} = E_{\text{compute, static reference}}$$

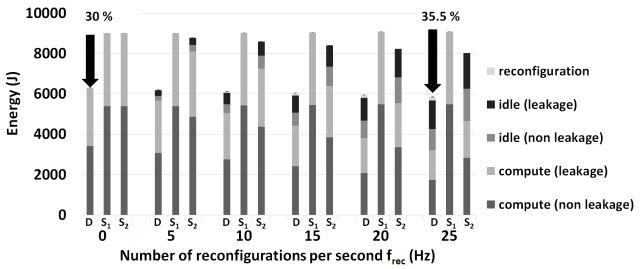

Fig. 6: Total energy consumed when the design is mapped dynamically (D) and statically (in the reference case (S_1), in the equalized case (S_2)) vs f_{rec} ($t_{\text{interval}} = 3600$ s).

The total energy consumed by the equalized static design (i.e. the system can be idle) $E_{\text{total, static equalized}}$ during t_{interval} has two contributions: (1) the total energy spent for compute $E_{\text{compute, static equalized}}$ and (2) the total energy spent when the design is idle $E_{\text{idle, static equalized}}$.

$$E_{\text{total, static equalized}} = E_{\text{compute, static equalized}} + E_{\text{idle, static equalized}}$$

The total energy consumed by the dynamic design $E_{\text{total, dynamic}}$ during t_{interval} has three contributions: (1) the total energy spent for compute $E_{\text{compute, dynamic}}$, (2) the total energy spent when the design is idle $E_{\text{idle, dynamic}}$, and (3) the total energy spent for reconfiguration E_{reconfig}.

$$E_{\text{total, dynamic}} = E_{\text{compute, dynamic}} + E_{\text{idle, dynamic}} + E_{\text{reconfig}}$$

E_{reconfig} depends on the number of reconfigurations during the time interval considered and is rewritten as

$$E_{\text{reconfig}} = E_{\text{rec,partition}} \times f_{\text{rec}} \times t_{\text{interval}}$$

$E_{\text{rec,partition}}$ is the energy for reconfiguring a single partition. $f_{\text{rec}} \times t_{\text{interval}}$ is the number of partition reconfigurations during the time interval considered. f_{rec} is the number of reconfigurations per second in Hz.

Energy Results. We use the same methodology as in the first study for power/energy estimations. Figure 6 reports the total energy consumed in Joules (J) during a fixed time interval of length $t_{\text{interval}} = 3600$ s when the design is mapped (1) dynamically $E_{\text{total, dynamic}}$, (2) statically (reference case) $E_{\text{total, static reference}}$, and (3) statically (equalized case) $E_{\text{total, static equalized}}$ for different number of reconfigurations per second f_{rec} (in Hz). We report the total energy spent (leakage and non leakage) (1) for compute and (2) when the design is idle. We also report the total reconfiguration energy E_{reconfig}. When no reconfiguration happens, $E_{\text{total, static reference}}$ and $E_{\text{total, static equalized}}$ are greater than $E_{\text{total, dynamic}}$ since (1) the statically mapped design dissipates more leakage power than the dynamically mapped design due to the use of a larger FPGA and (2) the statically mapped design uses more clocking resources than the dynamically mapped design resulting in a higher non-leakage power consumption.

As f_{rec} increases, we observe that both $E_{\text{total, static reference}}$ and $E_{\text{total, static equalized}}$ remain greater than $E_{\text{total, dynamic}}$ even when the compute to reconfiguration ratio is almost 1:1 (i.e. for $f_{\text{rec}} = 25$ Hz, $t_{\text{compute}} = 1827$ s and $t_{\text{rec}} = 1773$ s). Table V breaks down the total energy consumed when the design is mapped statically and dynamically for $f_{\text{rec}} = 25$ Hz. We observe that E_{reconfig} is very small compared to $E_{\text{compute, dynamic}}$ and $E_{\text{idle, dynamic}}$ even for a large number of reconfigurations. (The total number of reconfigurations is 90,000.) The dynamically mapped design consumes 30% and 35.5% less total energy than the statically mapped reference design in the best-case scenario ($f_{\text{rec}} = 0$ Hz) and in the worst-case scenario ($f_{\text{rec}} = 25$ Hz), respectively. (The modules used for event processing consume slightly more power/energy than the detection module.) The dynamically mapped design consumes 30% and 26% less total energy than the statically mapped equalized design in the best-case scenario ($f_{\text{rec}} = 0$ Hz) and in the worst-case scenario ($f_{\text{rec}} = 25$ Hz), respectively.

TABLE V: Energy breakdown when $t_{\text{interval}} = 3600$ s and $f_{\text{rec}} = 25$ Hz. When the ratio of reconfiguration to compute is almost 1:1, $E_{\text{total, dynamic}}$ is smaller than $E_{\text{total, static reference}}$ and $E_{\text{total, static equalized}}$ due to E_{reconfig} being very small.

		dynamic	static reference	static equalized
E_{compute} (J)	non-leakage	1736	5491	2741
	leakage	1462	3600	1827
	total	3198	9091	4568
E_{idle} (J)	non-leakage	1064	N/A	1596
	leakage	1418	N/A	1773
	total	2482	N/A	3369
E_{reconfig} (J)	non-leakage	177	N/A	N/A
	leakage	N/A	N/A	N/A
	total	177	N/A	N/A
E_{total} (J)	non-leakage	2977	5491	4337
	leakage	2880	3600	3600
	total	5857	9091	7937

V. Conclusion

In this paper, we show that dynamic mapping on smaller FPGAs reduces logic resource utilization by 2.5x and 3.2x, device cost by approximately 10x and 4x, and power/energy consumption by 28% and 30% compared to static mapping on larger FPGAs for two applications with cost or energy constraints. These applications benefit from dynamic FPGA mapping since (1) all tasks are not needed at the same time, and (2) reducing area, power or energy is as important as meeting performance requirement.

VI. Acknowledgments

This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We thank Xilinx for their FPGA and tool donations.

References

[1] R. Tamburo, E. Nurvitadhi, A. Chugh, M. Chen, A. Rowe, T. Kanade, and S. G. Narasimhan, "Programmable automotive headlights," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 750–765, Springer International Publishing, 2014.

[2] M. Ullmann, M. Huebner, B. Grimm, and J. Becker, "An fpga run-time system for dynamical on-demand reconfiguration," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pp. 135–, April 2004.

[3] D. Koch and J. Torresen, "FPGASort: A High Performance Sorting Architecture Exploiting Run-time Reconfiguration on Fpgas for Large Problem Sorting," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, (New York, NY, USA), pp. 45–54, ACM, 2011.
[4] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 109–116, May 2014.
[5] A. Sudarsanam, R. Barnes, J. Carver, R. Kallam, and A. Dasu, "Dynamically reconfigurable systolic array accelerators: A case study with extended kalman filter and discrete wavelet transform algorithms," *IET Computers Digital Techniques*, vol. 4, pp. 126–142, March 2010.
[6] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, "The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based Computer," *J. VLSI Signal Process. Syst.*, vol. 47, pp. 15–31, Apr. 2007.
[7] C. Claus, W. Stechele, and A. Herkersdorf, "Autovision a run-time reconfigurable mpsoc architecture for future driver assistance systems (autovision eine zur laufzeit rekonfigurierbare mpsoc architektur fr zuknftige fahrerassistenzsysteme)," vol. 49, pp. 181–, 05 2007.
[8] M. Nguyen and J. C. Hoe, "Time-shared execution of realtime computer vision pipelines by dynamic partial reconfiguration," in *28th International Conference on Field Programmable Logic and Applications, FPL 2018, Dublin, Ireland, August 27-31, 2018*, pp. 230–234, 2018.
[9] Xilinx, *Zynq-7000 All Programmable SoC Technical Reference Manual*, 2017.
[10] Xilinx, *Vivado Design Suite User Guide: High-Level Synthesis (UG902)*, 2017.
[11] Itseez, *The OpenCV Reference Manual*, 2.4.9.0 ed., April 2014.
[12] J. Hegarty, J. Brunhaver, Z. DeVito, J. Ragan-Kelley, N. Cohen, S. Bell, A. Vasilyev, M. Horowitz, and P. Hanrahan, "Darkroom: Compiling high-level image processing code into hardware pipelines," *ACM Trans. Graph.*, vol. 33, pp. 144:1–144:11, July 2014.
[13] J. Hegarty, R. Daly, Z. DeVito, J. Ragan-Kelley, M. Horowitz, and P. Hanrahan, "Rigel: Flexible multi-rate image processing hardware," *ACM Trans. Graph.*, vol. 35, pp. 85:1–85:11, July 2016.
[14] Xilinx, *Xilinx Power Estimation*, 2014.