

FPGA-Diff: Differential Bitstream Mechanisms for Efficient Secure Remote Updates of FPGAs

Sumiran Mehra*, Todor Stefanov*, Nele Mentens†

**Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands*

†*Department of Electrical Engineering, KU Leuven, Leuven Belgium*

*{s.mehra, t.p.stefanov}@liacs.leidenuniv.nl; †{nele.mentens}@kuleuven.be

Abstract—Integrating Field-Programmable Gate Arrays (FPGAs) in embedded systems enhances flexibility and performance. Remote reconfigurability of such systems enables post-deployment system updates. Existing approaches for remote FPGA reconfiguration rely on transmitting a full configuration bitstream, leading to long system update time and high energy consumption. Partial reconfiguration techniques reduce the update time and energy consumption but require partitioning of the FPGA into static and dynamic regions, thereby restricting the flexibility for post-deployment updates. To address these issues, we present FPGA-Diff, a framework for efficient and unrestricted secure remote FPGA reconfiguration. It introduces differential bitstream generation and application mechanisms to create a custom bitstream that contains the minimum data required to efficiently and securely update the remote FPGA configuration. The proposed framework is evaluated for several realistic FPGA systems with varying design updates and complexities. Compared to full reconfiguration, it achieves a reduction in update time and energy consumption by 97% and 70%, respectively, for minor design updates, and at least 15% and 10% reductions, respectively, for complex design updates.

Index Terms—FPGA, Remote Reconfiguration, Secure Updates

I. INTRODUCTION

The demand for lightweight and adaptable systems has been growing continuously in modern applications such as wireless sensor networks, edge-AI platforms, and industrial IoT setups. These environments require a low-cost, energy-efficient, and secure embedded System-on-Chip (SoC) that is also flexible for post-deployment functionality updates. Integrating a Field-Programmable Gate Array (FPGA) fabric within the SoC provides such flexibility due to its reconfigurable architecture. The FPGA fabric enables hardware acceleration and allows post-deployment functionality updates for evolving applications without physical hardware replacement. Such FPGA-integrated SoCs are often deployed remotely in the field and designed to operate with small energy budgets on battery power supply for years. Utilizing the flexibility of remotely deployed lightweight FPGA-integrated SoCs requires efficient remote reconfiguration of the FPGA by sending an updated FPGA configuration bitstream. Moreover, in some authentication-critical applications such as automotive, medical, or industrial automation, the transmitted configuration bitstream must also be authenticated and verified for integrity before FPGA reconfiguration to prevent transmission errors or malicious modifications.

The research work in [1], [2] addresses these requirements by proposing an approach that combines full bitstream compression for efficiency, and cryptographic computations for

authenticity and confidentiality. Other works [3]–[7] propose approaches for (secure) remote updates based on full FPGA reconfiguration. The main issue with the aforementioned approaches is that compressing the full bitstream for efficient bitstream transmission and performing full FPGA reconfiguration is not the most efficient way of updating the FPGA configuration. This is because typical updates of the system residing in the FPGA, such as updating constants/coefficients used in a functional block, adding/removing a functional block, or modifying the behavior of a functional block, are manifested as partial changes in the FPGA configuration bitstream, and only parts of the FPGA may need to be reconfigured.

There exist approaches [8]–[11] for efficient remote updates based on partial FPGA reconfiguration techniques that address the above issues. To utilize partial reconfiguration (PR), the FPGA fabric must be partitioned into static and dynamic regions before SoC deployment, and only the dynamically reconfigurable regions (DRRs) can be modified when the rest of the FPGA is still operational [12]. By confining the design updates within the DRRs, PR reduces the size of the transmitted configuration data and the FPGA reconfiguration time. However, such beneficial utilization of PR restricts the scope of post-deployment design updates because the updates must be confined within the DRRs and the boundary interfaces of these DRRs must not be changed [12]. If an update of the system residing in the FPGA includes, e.g., modifications of a functional block that cannot fit in the block's initial DRR or addition/removal of inputs/outputs in the boundary interfaces of the DRR, then such an update would require a full FPGA bitstream transmission, which is not efficient and desirable. Moreover, in some cases, the system designer cannot predict all possible post-deployment updates that will be required in order to properly partition the FPGA fabric in static and dynamic regions before SoC deployment.

Therefore, our work targets secure remote update scenarios where post-deployment design modifications are unrestricted, yet the remote updates are efficient without utilizing PR. We assume that the designer requires full flexibility over post-deployment system updates, i.e., the modifications are not limited by pre-allocated DRRs and fixed boundary interfaces of the DRRs. In addition, we assume systems where a full reconfiguration of the remotely deployed FPGA is performed with a bitstream stored in a serial non-volatile flash memory, since this is a common practice nowadays in low-cost FPGA-

integrated SoCs. Utilizing such memory for remote FPGA updates allows a system to reliably fall back to a secure golden configuration bitstream if an update fails.

Considering the above assumptions, we propose a novel framework, called **FPGA-Diff**, designed for efficient secure remote updates of FPGA configurations without limiting the scope of possible post-deployment design updates. More specifically, FPGA-Diff introduces a *differential bitstream generation* mechanism to create a custom bitstream (DIFF_BIT) containing the difference between two FPGA configurations. The DIFF_BIT bitstream contains the minimum data that must be transmitted to a remote FPGA-integrated SoC to properly update its FPGA configuration. In addition, FPGA-Diff adds an authentication tag and hash to the bitstream, and introduces a lightweight *differential bitstream application* mechanism to efficiently apply the DIFF_BIT onto the golden configuration, stored in the serial flash memory, after successful security verification. The specific novel contributions can be summarized as follows:

- 1) We present an algorithm to generate DIFF_BIT by identifying and encoding the difference between the last full bitstream (GOLDEN_BIT) used to configure the FPGA and a new full bitstream (UPDATE_BIT), which is the desired update of the FPGA configuration. The size of DIFF_BIT is much smaller than the size of UPDATE_BIT, thus sending DIFF_BIT instead of UPDATE_BIT to the remote SoC reduces the transmission and security verification time of the new FPGA configuration data as well as the memory and time needed to store the configuration data at the remote SoC.

- 2) We propose an efficient procedure, executed at the remote SoC, that utilizes the transmitted DIFF_BIT to partially overwrite the stored GOLDEN_BIT bitstream, before using it to reconfigure the FPGA. Such partial modification of GOLDEN_BIT reduces the total time and energy consumption of the FPGA configuration update process.

- 3) We evaluate our FPGA-Diff framework by comparing the runtime performance of FPGA configuration updates using the FPGA-Diff framework and a recent/relevant framework called UpGate [2]. This comparison is performed by using the state-of-the-art setup for secure remote FPGA reconfiguration, described in [2], and three realistic systems, namely Blinky LEDs, a RISC-V processor subsystem, and a Software-Defined Radio (SDR) receiver, implemented on the FPGA. Each system is updated a few times by variations of relevant design modifications in terms of size and complexity. The experimental results show that FPGA-Diff reduces the runtime and energy consumption by at least 15% and 10%, respectively, for large/complex design modifications, and up to 97% and 70%, respectively, for small design modifications.

The rest of the paper is organized as follows. The related work is presented in Section II. The FPGA-Diff framework and workflow are described in Section III. The experimental setup and use-case systems are elaborated in Section IV, and the experimental results and evaluations are presented in Section V. Finally, Section VI concludes the paper.

II. RELATED WORK

Existing research on efficient secure remote reconfiguration of FPGA-integrated SoCs broadly focuses on either full or partial reconfiguration of FPGAs. In the case of full FPGA reconfiguration, prior work primarily looks at improving data transfer efficiency through advanced bitstream compression techniques or optimized wireless communication protocols. In [2], the authors present an energy-efficient approach for secure over-the-air full FPGA reconfiguration using lossless compression algorithms such as GZip and LZ4 [13], thereby significantly reducing the bandwidth requirement of the bitstream transmission channel. Similarly, studies in [1], [14], [15] introduce alternative bitstream compression techniques to generate a compact bitstream for faster and more efficient transmission. However, despite the improved transmission efficiency, during a secure remote FPGA configuration update using a full bitstream, a major portion of the update time is spent in security verification of the full authentication tag and moving the decompressed full bitstream to the FPGA's configuration memory. This makes such an update less efficient for lightweight applications. In contrast, we propose the FPGA-Diff framework that uses a custom differential bitstream, which is much smaller than a full bitstream, for more efficient secure remote updates of the FPGA's configuration memory.

Several studies investigate wireless communication protocols for secure and power-efficient transmission of the configuration bitstream. The authors of [3] propose in-system reconfigurability of FPGA-MCU subsystems using Zipwire for automotive applications. The works in [4], [5] present a secure wireless reconfiguration protocol for embedded and edge systems. The works in [6], [7] address efficient secure remote reconfiguration of FPGA-integrated SoCs in cloud and IoT applications. Although the communication-centric optimizations improve the bitstream transfer efficiency, they still rely on full bitstream transmission, making them inherently inefficient for minor design updates that only alter a small portion of the FPGA configuration. Moreover, a full bitstream transmission requires a complete update of the stored golden bitstream in the flash memory, leading to longer FPGA update cycles and increased energy consumption.

To address these limitations, prior studies [8]–[11], [16], [17] explore partial or dynamic partial reconfiguration (PR) techniques. PR enables modification of a specific part of the design, residing in the FPGA, while keeping the remaining part operational, thereby reducing the FPGA reconfiguration time and the amount of configuration data to be transferred. However, as explained in Section I, PR relies on predefined reconfigurable partitions that split the FPGA fabric into static and dynamic regions. This design segmentation limits the flexibility for post-deployment design updates, particularly when updates extend beyond the boundaries of the dynamically reconfigurable regions (DRRs). In contrast, our work targets application scenarios in which post-deployment design modifications cannot be pre-determined, and full-device reconfiguration is required to preserve maximum modification flexibility. To support such scenarios, our proposed FPGA-Diff framework

introduces bitstream-level differential mechanisms based on our custom binary delta encoding to generate and apply a specific compact bitstream, containing only the modifications in the FPGA configuration, for efficient remote updates utilizing a serial non-volatile flash configuration memory.

Finally, it is worth mentioning that there exist several generic binary delta encoding algorithms, such as bsdiff, xdelta3, and Delta+++ [18], and remote data synchronization tools like rsync [19] that are used to efficiently build and apply patches to binary files. However, these generic algorithms and tools are not inherently designed to efficiently operate with binary files stored in serial flash configuration memories. Such memories are organized into multiple sectors and pages, and require sector-aware erase and page-aware program operations, imposing structural constraints on how memory content updates can be applied. Thus, applying the aforementioned generic algorithms and tools to serial FPGA configuration flash memories requires additional data pre-processing that introduces extra computational latency. To address these issues, the FPGA-Diff framework introduces serial-flash-memory-aware custom *differential bitstream generation* and *application* mechanisms to efficiently update the FPGA configuration stored in the serial flash memory. This custom bitstream aligns with the flash memory geometry, enabling efficient sector-wise updates while eliminating redundant erase and program operations. The proposed framework significantly reduces the amount of configuration data to be transferred, and more importantly, it allows a performance- and energy-efficient secure remote FPGA (re-)configuration without limiting the flexibility of possible post-deployment design updates.

III. THE FPGA-DIFF FRAMEWORK AND WORKFLOW

This section describes our novel FPGA-Diff framework and its mechanisms. It also describes a specific workflow utilizing FPGA-Diff in the context of a remote system containing a microcontroller unit (MCU), an FPGA, and a flash memory. The red dashed arrows in Fig. 1 show a typical workflow for secure remote updates of the FPGA configuration in such a remote system. The workflow involves: 1) the host-site that generates an updated FPGA configuration bitstream and transmits it via a public communication network, and 2) the device-site (i.e., the remote system) that receives the updated bitstream, stores it in the flash memory upon successful security verification of the received bitstream, and reconfigures the FPGA from the flash memory. The black solid arrows in Fig. 1 show the specific workflow utilizing our FPGA-Diff mechanisms. A detailed description of the framework, the specific workflow, and the system architecture is given in the following subsections.

A. FPGA-Diff Framework

The FPGA-Diff framework introduces bitstream-level differential-based mechanisms, called DIFF_BIT generation and DIFF_BIT application, that enable efficient updates of an FPGA configuration using minimal update data instead of a full configuration bitstream. FPGA-Diff identifies and encodes the difference between a previously deployed bitstream (GOLDEN_BIT) and an updated bitstream (UPDATE_BIT),

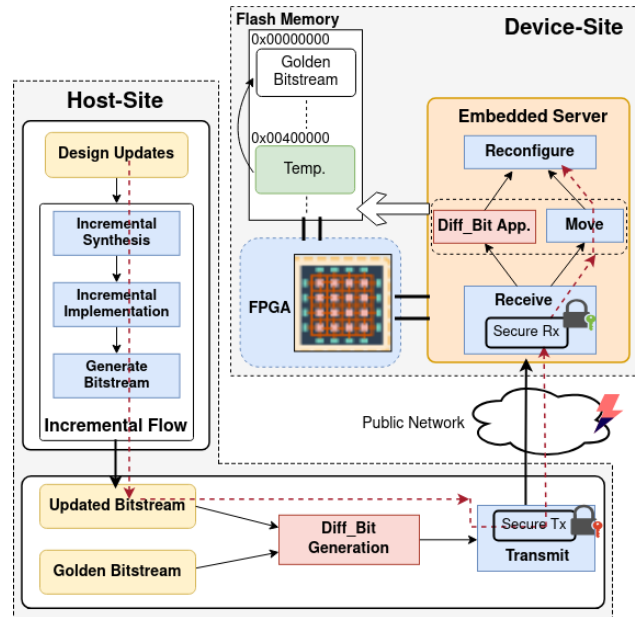


Fig. 1. Workflow for secure remote updates of FPGA configurations; the red dotted lines show the traditional workflow without our FPGA-Diff mechanisms

thereby generating a compact binary difference (DIFF_BIT). DIFF_BIT is a custom configuration bitstream that contains the minimal data required to properly update the previously deployed FPGA configuration. The FPGA-Diff mechanisms, illustrated as pink boxes in Fig. 1, are utilized in the specific 3-stage workflow, shown in Fig. 1. The workflow is as follows.

In **Stage 1**, on the host-site, the system designer has the HDL source files of the previously deployed FPGA design, its bitstream GOLDEN_BIT, and specific files that enable deterministic mapping between design updates and their configuration frames. The designer updates the previously deployed design by modifying the HDL files. Then, the updated design is synthesized and implemented by using an FPGA incremental compilation flow with the help of the specific files to ensure temporal and spatial correlation between design update iterations. This is followed by the generation of the updated bitstream UPDATE_BIT.

In **Stage 2**, on the host-site, the DIFF_BIT generation mechanism (Section III-B) is used to perform a byte-level comparison between GOLDEN_BIT and UPDATE_BIT to identify the differences and encode them into DIFF_BIT, which is then securely transmitted chunk-wise to the remote device-site over a public communication channel. Before transmission, the host-site system sends digitally signed metadata (such as a combination of data size, data version, and hash) to the device-site, and waits for a success acknowledgment. This ensures trust between the host-site and the device-site systems.

In **Stage 3**, on the device-site, an embedded server running on the MCU verifies the signed metadata and erases the corresponding portion of the flash memory required to store the incoming data, then sends a success acknowledgment to

Algorithm 1: DIFF_BIT Generation

Input: $G[], U[]$
Output: $D_{\text{diff}}[]$

```

1  $D_{\text{diff}}[] \leftarrow \emptyset;$ 
2 foreach  $i \in [0, N/S_{\text{size}} - 1]$  do
3   foreach  $j \in [0, S_{\text{size}}/P_{\text{size}} - 1]$  do
4      $b_g \leftarrow G[S_i, P_j]; \quad b_u \leftarrow U[S_i, P_j];$ 
5      $Q_g[] \leftarrow \text{SplitToQuads}(b_g); \quad Q_u[] \leftarrow \text{SplitToQuads}(b_u);$ 
6      $k = 0;$ 
7     while  $k < |Q_g|$  do
8       if  $Q_g[k] \neq Q_u[k]$  then
9          $idx = k; \quad \text{off} = 1;$ 
10        while  $\text{RUNCONDITION}(k, \text{off})$  do
11           $\text{off} = \text{off} + 1;$ 
12           $H = (S_i, P_j, idx, \text{off});$ 
13           $D_{\text{diff}}[] \leftarrow D_{\text{diff}}[] \# \{(H, Q_u[idx : idx + \text{off}])\};$ 
14           $k = idx + \text{off};$ 
15        else
16           $k = k + 1;$ 
17 return  $D_{\text{diff}}[]$ 
18 Definition:  $\text{RUNCONDITION}(k, \text{off})$  returns true if:
19    $k + \text{off} < |Q_g|$  and  $Q_g[k + \text{off}] \neq Q_u[k + \text{off}]$ 

```

the host-site. Further, the device-site receives DIFF_BIT chunks and, upon successful security verification, stores them in a temporary location of the flash memory. After verifying and storing all DIFF_BIT chunks, the MCU uses the DIFF_BIT application mechanism (Section III-C) to update GOLDEN_BIT, stored in the flash memory, and triggers the FPGA configuration using the updated GOLDEN_BIT.

B. DIFF_BIT Generation Mechanism

The DIFF_BIT generation mechanism is formally described in Algorithm 1. It takes as input GOLDEN_BIT $G[]$ and UPDATE_BIT $U[]$, and outputs DIFF_BIT $D_{\text{diff}}[]$. The algorithm is designed to align with the organization of the device-site flash memory, where typically the memory is divided into sectors and each sector is further subdivided into pages. In the algorithm, the size of the bitstream is denoted as N , the sector identifier as S_i , the size of the sector as S_{size} , the page identifier as P_j , and the size of the page as P_{size} .

The algorithm performs sector-wise and page-wise comparison as shown in Line 2-16. First, part b_g of the golden bitstream for a particular sector S_i and page P_j is extracted from $G[]$ and the corresponding part b_u is extracted from $U[]$, and both are split into groups of 4 bytes called quads $Q_g[]$ and $Q_u[]$, respectively, as shown in Line 4-5. The quad splitting ensures an even number of 4-byte groups in each flash memory page. Next, consecutive differing quads are grouped into *segments*, as shown in Line 7-16, where the number of segments is less than or equal to the number of quads per page $|Q_g|$ (Line 7). More specifically, for segment k , quads $Q_g[k]$ and $Q_u[k]$ are compared (Line 8), and if they are different, then index idx is set with value k (Line 9), showing the position of the update in that particular page. Also, the offset off is iteratively and conditionally incremented, using function $\text{RUNCONDITION}()$ in Line 10-11, to determine the number of consecutive quads that

Algorithm 2: DIFF_BIT Application

Input: $D_{\text{diff}}[], G_F[]$
Output: Updated $G_F[]$

```

1  $S_{\text{imp}} = -1; \quad \text{Buf}[] \leftarrow \emptyset;$ 
2 foreach  $(H, Q_u[]) \in D_{\text{diff}}[]$  do
3    $P_{\text{off}} = H.P_j \times P_{\text{size}}; \quad B_{\text{off}} = H.idx \times 4;$ 
4   if  $(S_{\text{imp}} \neq H.S_i)$  and  $(S_{\text{imp}} \neq -1)$  then
5      $\text{ERASE: } G_F[S_{\text{imp}}]; \quad \text{PROGRAM: } G_F[S_{\text{imp}}] \leftarrow \text{Buf}[];$ 
6      $S_{\text{imp}} = -1;$ 
7   if  $(S_{\text{imp}} = H.S_i)$  or  $(S_{\text{imp}} = -1)$  then
8     if  $S_{\text{imp}} = -1$  then
9        $\text{Buf}[] \leftarrow G_F[H.S_i]; \quad S_{\text{imp}} = H.S_i;$ 
10    foreach  $Quad \in Q_u[]$  do
11       $\text{Buf}[P_{\text{off}} + B_{\text{off}}] \leftarrow Quad;$ 
12       $B_{\text{off}} = B_{\text{off}} + 4;$ 

```

are different and that belong to segment k . A header H is attached to the consecutive differing quads $Q_u[t]$ that belong to segment k , where $t \in [idx, idx + \text{off}]$, and header H and the quads $Q_u[t]$ are appended to $D_{\text{diff}}[]$ (Line 12-13). Header H encodes the sector identifier (S_i), the page identifier (P_j), the position (idx) of the update in the page, and the number (off) of consecutive quads (Line 12). By traversing all sectors S_i and pages P_j in $G[]$ and $U[]$ and comparing their quads, the algorithm returns the final differential bitstream $D_{\text{diff}}[]$ that is required to update the golden configuration bitstream, stored in the flash memory, on the remote device-site. Moreover, since Algorithm 1 operates entirely on the raw binary data in a bitstream, it remains independent of vendor-specific bitstream formats and is therefore portable to any FPGA that supports configuration from flash memory.

C. DIFF_BIT Application Mechanism

The DIFF_BIT application mechanism is formulated in Algorithm 2. It takes as input DIFF_BIT $D_{\text{diff}}[]$ and GOLDEN_BIT $G_F[]$ stored in the flash memory on the device-site, and as an output it updates the golden configuration bitstream $G_F[]$. The algorithm uses the same notations for sectors and pages, S_i , S_{size} , P_j and P_{size} , and for the header-quads pair, $(H, Q_u[]) \in D_{\text{diff}}[]$, as used in Algorithm 1.

Algorithm 2 begins with initializing the current sector identifier S_{imp} and the local sector buffer $\text{Buf}[]$ (Line 1). The algorithm updates the flash memory in a sector-wise manner by sequentially processing each pair $(H, Q_u[]) \in D_{\text{diff}}[]$ (Line 2-12). First, it interprets header H and computes the page offset P_{off} and byte offset B_{off} (Line 3), determining the write address of the incoming data quads $Q_u[]$. Next, the current sector identifier S_{imp} and the sector identifier from the header $H.S_i$ are compared to check whether the quads $Q_u[]$ belong to the currently processed sector S_{imp} or the sector is a new one (Line 7). In case of a new sector ($S_{\text{imp}} = -1$), first, the golden bitstream $G_F[]$ at sector $H.S_i$ is copied to $\text{Buf}[]$, and S_{imp} is set to $H.S_i$ (Line 8-9). Then, all updated quads $\in Q_u[]$, corresponding to sector S_{imp} are copied to $\text{Buf}[]$ (Line 10-12), where each quad of 4-byte updated data is written at buffer position $P_{\text{off}} + B_{\text{off}}$, thereby replacing the old configuration data

(Line 11). Before processing a new pair $(H, Q_u[]) \in D_{\text{diff}}[]$, the algorithm checks whether the pair belongs to a sector different from the currently processed sector, as shown in Line 4. If this is the case, it means that all updated quads belonging to $Q_u[]$ of the current sector S_{imp} are copied to buffer $\text{Buf}[]$, and the algorithm finalizes the current sector (Line 5-6). That is, it performs an ERASE operation of sector $G_F[S_{\text{imp}}]$ in the flash memory followed by a page-wise PROGRAM operation on the same sector using the updated data in buffer $\text{Buf}[]$ (Line 5). The current sector identifier S_{imp} is then set to its default value (Line 6) for the next sector update cycle.

D. System Architecture

IV. EXPERIMENTAL SETUP

A. Setup

Fig. 2. Block diagram of the system architecture considered in FPGA-Diff

On the host-site (Fig. 1), the Xilinx Vivado Design Suite 2025.1 is used to design the FPGA use-case systems targeting Artix-A7 FPGA, and the Vivado standard incremental compilation flow is used to generate the configuration bitstream for each design. The incremental compilation flow preserves synthesis, placement, and routing information for each compilation run in the form of design checkpoint (.dcp) files. In addition, our DIFF_BIT generation mechanism (Algorithm 1) is integrated with the software implementation of the update controller secure transmission protocol used in the UpGate framework [2]. This protocol is used for secure transmission of DIFF_BIT to the device-site using Ed25519 for digital signature verification and SHA-256 for hashing. On the device-site, the embedded server running on the MCU is implemented using the software infrastructure of the UpGate framework, i.e., the Receive, Move, and Reconfigure blocks in Fig. 1. The Receive software implements the security algorithms accelerated by the ARM CryptoCell hardware accelerator, to check the authenticity and integrity of the received data. The Move software contains specific drivers to (re-)program the FPGA flash memory, whereas the Reconfigure software includes drivers to (re-)configure the FPGA using a bitstream stored in the flash memory. Moreover, the embedded server is extended with a software implementation of our DIFF_BIT application mechanism (Algorithm 2).

B. FPGA Use-Case Systems

For every use-case system, an initial baseline design (**Baseline**) is synthesized and implemented using the Vivado incremental compilation flow, and the corresponding GOLDEN_BIT configuration bitstream and .dcp files are obtained. Furthermore, for every use-case system, three post-

TABLE I
RESOURCE UTILIZATION AND DIFF_BIT SIZES

Designs	Updates	LUT	Reg.	DSP	MUX	Slice	I/O	BUFG	FPGA-Diff		UpGate	
									KB	%	KB	%
Blinky LEDs	Baseline	3534	7980	0	0	2127	10	1	-	-	-	-
	Case1	-94	-3	0	0	-20	-2	0	108.50	97.09	102.41	97.23
	Case2	+11	+1	0	0	+5	0	0	28.90	99.23	113.36	96.97
	Case3	+1	-1	0	0	-5	0	0	13.60	99.64	115.17	96.92
RISCV	Baseline	11414	4007	0	78	3505	180	2	-	-	-	-
	Case1	+3425	-8	0	-16	+1090	0	0	652.25	82.54	544.37	85.43
	Case2	-41	+142	0	+2	+71	0	0	484.00	87.05	433.70	88.39
	Case3	-304	-4	+2	-58	-41	0	0	366.00	90.20	430.13	88.49
SDR	Baseline	19144	28620	36	0	8481	77	2	-	-	-	-
	Case1	-947	-342	-18	0	-305	0	0	778.50	79.16	810.43	78.31
	Case2	-2	0	0	0	-60	0	0	144.00	96.15	829.77	77.79
	Case3	+1	0	0	0	0	0	0	0.50	99.99	823.87	77.95

TOTAL RESOURCES: LUT:63400, REG.:126800, SLICE:15850, DSP:240, IO:210, BUFG:32; FULL BITSTREAM: 3.65MB

deployment updated designs are created. Each of these updated designs is compiled incrementally by using the .dcp files of the **Baseline** design to generate the updated configuration bitstream (UPDATE_BIT). The details of every **Baseline** design and the specific design update/modification cases to obtain its three updated designs are described below:

1) **Blinky LEDs**: The baseline design is a parametrized LED blinking controller driven by an oscillator array, a timing sequencer, and an LED mapper. The controller drives eight LEDs at a fixed blinking frequency. The three incremental update cases are: i) **Case1** - the number of active LEDs is changed from eight to six by modifying one parameter in the HDL design files; ii) **Case2** - a constant in the design HDL that controls the blinking period is adjusted to update the blinking frequency; iii) **Case3** - the internal control logic of the FSM-based sequencer is optimized to a more timing-stable design, while keeping the design update small and localized. The above update cases reflect design modifications ranging from major design tuning to small logic restructuring.

2) **RISCV**: The baseline design features a RISCV processor subsystem based on the lowRISC Ibex core architecture. It implements the secure Ibex variant with the RV32IM instruction set, an RV32M_slow multiplier/divider unit, and no write-back stage or bit-manipulation support. The three incremental update cases are: i) **Case1** - Bitmanip (B) extension is integrated to provide support for new arithmetic/logic operations and instruction decoding; ii) **Case2** - a write-back stage is added to the processor pipeline, thereby modifying the data-path latency and the register control logic; iii) **Case3** - the RV32M_slow multiplier/divider unit is replaced with a high-performance RV32M_fast variant to speed up the arithmetic operations. These update cases introduce both structural and functional modifications to the RISCV core micro-architecture.

3) **SDR**: The baseline design is a parameterized software-defined-radio baseband receiver system including frequency generation and mixing stages, a multi-rate decimation filter, automatic gain control logic, and a Bluetooth Low Energy (BLE) demodulator. The three update cases are: i) **Case1** - BLE demodulator is replaced with an IEEE 802.15.4 demodulator, thereby changing the communication protocol; ii) **Case2** - Quadrature Phase Shift Keying demodulation chain is replaced

with a 2-Frequency-Phase Shift Keying chain to enable narrow-band frequency-shift keying; iii) **Case3** - the baseband signal parameters, including the decimation ratio and the symbol sampling rates, are adjusted within the BLE demodulation.

Collectively, the aforementioned design update cases represent a diverse range of typical post-deployment design modifications from parameter-level updates and architectural changes to complex design functionality updates. This allows a comprehensive evaluation of the FPGA-Diff framework in the context of realistic post-deployment FPGA reconfiguration scenarios.

V. EXPERIMENTAL RESULTS AND EVALUATION

This section presents the results from a quantitative performance evaluation of our proposed FPGA-Diff framework by analyzing the impact of small to complex design modifications on the size of our custom differential bitstream (DIFF_BIT) and on the runtime and energy consumptions of the FPGA configuration update cycle. It also presents a performance comparison of FPGA-Diff with the state-of-the-art UpGate framework [2]. For a fair comparison, all experiments are conducted on the experimental setup described in Section IV-A.

A. Impact of Design Modifications on DIFF_BIT Size

We analyze the impact of the complexity of FPGA configuration updates on the size of the differential bitstream DIFF_BIT, generated by Algorithm 1, across all FPGA use-case systems and their incremental update cases described in Section IV-B. The complexity of an FPGA configuration update is indicated by the difference in FPGA resource utilization after each design modification/update. Table I summarizes the resource utilization differences in terms of number of LUTs, Registers, DSPs, MUXs, Slices, Input/Outputs (I/O), and Buffers, along with the DIFF_BIT size in Kilobytes (KB) in column FPGA-Diff. For each incremental update case, the +/- numbers in the table show the differences in utilized resources compared to the baseline design.

The result shows that the size of DIFF_BIT scales proportionally with the complexity of the FPGA configuration update. Low-complexity updates result in a very small differential bitstream, while high-complexity updates lead to a larger size of the differential bitstream. For example, Blinky LEDs update

TABLE II
RUNTIME OF FPGA UPDATE CYCLE WHEN USING UPGate AND FPGA-DIFF

Designs	Time Components (seconds)	Case 1		Case 2		Case 3	
		UpGate	FPGA-Diff	UpGate	FPGA-Diff	UpGate	FPGA-Diff
Blinky LEDs	Initialize Time	11.839	0.423	11.882	0.238	11.958	0.247
	Decompress Time	1.026	-	1.035	-	1.045	-
	Store+Hash Verify Time	32.190	15.206	36.844	4.047	36.998	1.896
	Move/Diff_App. Time	32.443	11.599	32.492	8.625	32.602	7.486
	FPGA Config. Time	4.000	4.000	4.000	4.000	4.000	4.000
	Total Runtime	84.499	31.228	86.252	16.910	86.603	13.929
RISCV	Initialize Time	11.198	2.131	11.646	1.571	11.719	1.163
	Decompress Time	2.76	-	1.827	-	0.911	-
	Store+Hash Verify Time	94.104	94.243	78.747	70.013	79.189	53.003
	Move/Diff_App. Time	31.898	21.511	32.762	21.138	32.655	16.588
	FPGA Config. Time	4.000	4.000	4.000	4.000	4.000	4.000
	Total Runtime	143.960	121.885	128.982	96.722	128.474	73.591
SDR	Initialize Time	11.960	2.556	11.739	0.619	11.640	0.247
	Decompress Time	1.001	-	1.001	-	0.999	-
	Store+Hash Verify Time	133.295	112.368	136.343	20.751	135.868	0.101
	Move/Diff_App. Time	32.690	25.355	32.610	10.447	32.691	1.020
	FPGA Config. Time	4.000	4.000	4.000	4.000	4.000	4.000
	Total Runtime	182.945	144.279	185.693	35.817	184.197	5.368

Case2 and Case3 involve minor design modifications that result in small changes in the resource utilization of the updated design and in the internal interconnection of logic blocks compared to the Baseline design. This, in turn, leads to a small differential bitstream with sizes 28.9KB and 13.6KB, respectively. On the other hand, in Blinky LEDs Case1, a significant modification in the internal interconnection of logic blocks leads to changes in a large portion of the Baseline configuration bitstream and thus, generates a larger DIFF_BIT of size 108.5KB compared to Case2 and Case3.

This trend also remains consistent across the RISCV and SDR systems and their update cases. For RISCV, a complex architectural enhancement in Case1 leads to a large change in the design logic resources and thus, a large differential bitstream of size 652.25KB. A relatively small update in Case2 leads to a smaller DIFF_BIT of size 484KB. On the other hand, a localized design modification in Case3 leads to a very small change in the logic resource utilization and thus, generates the smallest size of DIFF_BIT, compared to Case1 and Case2. Similarly, the SDR update Case1 leads to a large change in the system internal architecture and thus, generates the largest differential bitstream of size 778.5KB. In contrast, update Case3 involves a very small design modification, which in turn results in the smallest differential bitstream of size 512B.

Moreover, Table I also compares the size of our generated DIFF_BIT with the size of the configuration bitstream transferred when using the state-of-the-art UpGate framework for remote FPGA configuration updates. UpGate uses the LZ4 compression technique to reduce the size of the bitstream because LZ4 outperforms most of the state-of-the-art compression techniques in terms of compression rate and update performance [2]. In Table I, the size of the LZ4 compressed bitstream is given in column UpGate, and the size of DIFF_BIT in column FPGA-Diff, together with the corresponding reduction ratios (in %) with respect to the full uncompressed bitstream size for all FPGA use-case systems and their update cases. The results

clearly suggest that utilizing DIFF_BIT instead of a compressed full bitstream for remote update of the FPGA is more efficient in terms of the amount of data that must be transferred during the update cycle, especially for smaller design modifications such as the updates in Case3 for all use-case systems.

B. Time and Energy Consumption

In this section, we compare the runtime and energy consumption of the FPGA configuration update cycle on the remote device-site, when using FPGA-Diff and UpGate, across all use-case systems and their design update cases.

Table II shows the runtime results, where the total runtime is broken down into five components: 1) **Initialize Time**: the time taken to perform signature verification and to erase the flash memory in order to store the incoming data; 2) **Decompress Time**: the time taken to decompress the incoming LZ4 compressed full bitstream in case of UpGate; 3) **Store + Hash Verify Time**: this is the total time taken to store the incoming data, i.e., the full bitstream in case of UpGate and the DIFF_BIT bitstream in case of FPGA-Diff, at a temporary location in the flash memory and to verify the hash; 4) **Move/Diff_App. Time**: the time taken to copy the verified incoming data from the temporary location to the location where the golden bitstream is stored, i.e. in case of UpGate, the time taken to move the uncompressed full bitstream from the temporary location to the golden bitstream location thereby updating the golden bitstream by fully overwriting it, and in case of FPGA-Diff, the time taken to copy/apply the DIFF_BIT bitstream to the golden bitstream location (as described in Algorithm 2), thereby updating the golden bitstream by partially overwriting it; and 5) **FPGA Configuration (Config.) Time**: this is the time taken to reconfigure the FPGA with the updated golden bitstream, which is the same for both UpGate and FPGA-Diff. Once the golden bitstream is updated, the FPGA is reconfigured with the updated golden configuration bitstream.

The results in Table II show that the FPGA-Diff framework delivers a significant reduction of the total runtime compared

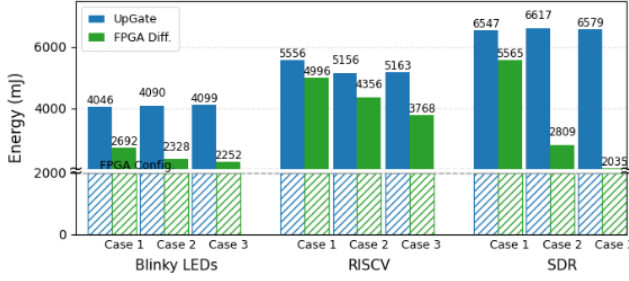


Fig. 3. Energy consumption during the FPGA update cycle

to the UpGate framework. The largest time reductions are in the following components: i) Initialize Time, where fewer flash memory sectors needs to be erased to store DIFF_BIT compared to the full bitstream in case of UpGate; ii) Store Time, where the full bitstream takes longer to be stored in the temporary location of the flash memory than the DIFF_BIT; and iii) Move/Diff_App. Time, where copying/applying DIFF_BIT takes less time than moving the full bitstream to the golden bitstream location in the flash memory. The FPGA-Diff consistently outperforms UpGate by reducing the total runtime by at least 15% for complex design modifications such as RISCv Case1. For small design modifications, such as in the Blinky LEDs Case3 and SDR Case3, FPGA-Diff achieves up to a 97% runtime reduction. Moreover, for large design modifications, such as in RISCv update Case2 and Case3 and SDR Case1, FPGA-Diff still delivers 15-25% total runtime reduction compared to UpGate.

Similarly, Fig. 3 illustrates that the FPGA-Diff framework also reduces the overall energy consumption during the full FPGA update cycle compared to the UpGate framework. However, a significant amount of the energy, ≈ 2 J represented by the dashed bars in the figure, is consumed to configure the FPGA fabric, which consequently limits the overall energy reduction. As a result, the relative reduction in energy is smaller than the observed runtime improvement. Across all update cases, FPGA-Diff delivers an energy reduction of $\approx 10\%$ for large/complex design modifications, while for small/minor design modifications the reduction reaches up to 70%. All presented and discussed results demonstrate that FPGA-Diff is effective in providing an overall reduction of the runtime and energy consumption for secure remote FPGA configuration updates.

VI. CONCLUSIONS

We propose the FPGA-Diff framework for efficient and unrestricted secure remote FPGA configuration updates. By transmitting only modified configuration data instead of a full configuration bitstream, the framework reduces the overall FPGA configuration update time and energy consumption. It achieves a reduction in the update time and energy consumption by 97% and 70%, respectively, for minor configuration updates, and at least 15% and 10% reduction, respectively, for complex configuration updates. Furthermore, FPGA-Diff specifically targets applications that use a lightweight FPGA-MCU system with a serial flash memory to store a golden

FPGA configuration bitstream, and where post-deployment updates of the design residing in the FPGA are unrestricted as well as the system can reliably fall back to a secure golden configuration bitstream if an update fails.

VII. ACKNOWLEDGMENT

The author would like to express their gratitude to NimbleAI project. NimbleAI is funded by the EU's Horizon Europe Research and Innovation programme (Grant Agreement 101070679), and by the UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee (Grant Agreement 10039070).

REFERENCES

- [1] J. Vliegen *et al.*, "A single-chip solution for the secure remote configuration of FPGAs using bitstream compression," in *2013 International Conference on ReConFig*, 2013, pp. 1–6.
- [2] A. Abadie *et al.*, "UpGate: Low-Power and Secure Over-the-Air Hardware Reconfiguration for Deployed FPGAs," in *2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE, 2025, pp. 28–35.
- [3] K. Cho *et al.*, "An FPGA-based ECU for remote reconfiguration in automotive systems," *Micromachines*, vol. 12, no. 11, p. 1309, 2021.
- [4] S. Drimer *et al.*, "A protocol for secure remote updates of FPGA configurations," in *International Workshop on Applied Reconfigurable Computing*. Springer, 2009, pp. 50–61.
- [5] S. M. Aziz *et al.*, "Remote reconfiguration of FPGA-based wireless sensor nodes for flexible Internet of Things," *Computers and Electrical Engineering*, vol. 100, p. 107935, 2022.
- [6] O. Machidon *et al.*, "Remote SoC/FPGA platform configuration for cloud applications," in *2014 International Conference on OPTIM*. IEEE, 2014, pp. 827–832.
- [7] A. Braeken *et al.*, "Secure remote reconfiguration of an FPGA-based embedded system," in *6th International Workshop on ReCoSoC*, 2011, pp. 1–6.
- [8] D. Mesquita *et al.*, "Remote and Partial Reconfiguration of FPGAs: tools and trends," in *Proceedings International Parallel and Distributed Processing Symposium*. IEEE, 2003, pp. 8–pp.
- [9] K. Vipin *et al.*, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–39, 2018.
- [10] F. Liu *et al.*, "A Novel Partial Dynamic Reconfiguration Image Sensor Node for Wireless Multimedia Sensor Networks," in *2012 IEEE 14th International Conference on High Performance Computing and Communication*, 2012, pp. 1368–1374.
- [11] B. Sellers *et al.*, "Bitstream compression through frame removal and partial reconfiguration," in *2009 International Conference on Field-Programmable Logic and Applications*. IEEE, 2009, pp. 476–480.
- [12] "Vivado Design Suite User Guide, Partial Reconfiguration UG909 (v2015.2)," 06 2015.
- [13] Y. Collet, "LZ4 Github Repository," in *Available:https://github.com/lz4/lz4*, 2024. [Online].
- [14] C. Fritzsch *et al.*, "Reduction of bitstream size for low-cost ice40 FPGAs," in *2022 32nd International Conference on Field-Programmable Logic and Applications*. IEEE, 2022, pp. 117–122.
- [15] Xilinx, "Spartan-6 FPGA Configuration User Guide," *Document on. UG380 (v2.11)*, pp. 1–174, 22 March 2019.
- [16] J. Vliegen *et al.*, "Secure, remote, dynamic reconfiguration of FPGAs," *ACM TRET*, vol. 7, no. 4, pp. 1–19, 2014.
- [17] T. Friedrich *et al.*, "A flexible co-processing approach for soc-fpgas based on dynamic partial reconfiguration and bitstream relocation methods," in *2015 10th International Symposium on ReCoSoC*. IEEE, 2015, pp. 1–7.
- [18] M. J. May, "Donag: Generating efficient patches and diffs for compressed archives," *ACM TOS*, vol. 18, no. 3, pp. 1–41, 2022.
- [19] A. Tanpure *et al.*, "Rsync over https for linux and windows with seamless data transfer," *IJSRP*, pp. 582–285, 2015.
- [20] Digilent, "Arty A7 Reference Manual," in *Available:https://digilent.com/reference/programmable-logic/arty-a7/reference-manual*, 2023. [Online].
- [21] S. Riedel, "Lowriscv Github Repository," in *Available:https://github.com/lowRISC*, 2025. [Online].