Journal of Universal Computer Science, vol. 27, no. 11 (2021), 1174-1192
submitted: 11/2/2021, accepted: 9/8/2021, appeared: 28/11/2021

Temporal Accelerators: Unleashing the Potential of Embedded FPGAs

Christopher Cichiwskyj
(University of Duisburg-Essen, Duisburg, Germany
https://orcid.org/0000-0001-6126-5895, christopher.cichiwskyj@uni-due.de)

Gregor Schiele
(University of Duisburg-Essen, Duisburg, Germany
https://orcid.org/0000-0003-4266-4828, gregor.schiele@uni-due.de)

Abstract: When the complexity of a problem rises, its solution requires more hardware resources. A usual way to solve this is to use larger processors and add more memory. When using Field Programmable Gate-Arrays (FPGAs), which can instantiate arbitrary circuit designs, a larger, more costly and power hungry chip is used. In this paper we propose a different approach, namely to split the problem into a graph of interdependent smaller tasks and to reconfigure a small FPGA during runtime to execute each of these tasks efficiently sequentially. This can result in cheaper and more energy efficient systems that can execute very complex problems locally. We present a basic analytical model, evaluate its accuracy and discuss initial insight from it.

Keywords: IoT, Embedded, FPGA, Reconfigurable Hardware
Categories: B.6.0, C.3, C.5.3, C.5.4
DOI: 10.3897/jucs.77247

1 Introduction

The Internet of Things (IoT) consists of billions of cheap, low power devices that are embedded in everyday objects. They have very limited processing power and can execute only basic tasks. More complex tasks are usually offloaded to cloud services. This can lead to high latency as well as privacy and reliability risks. To mitigate this, researchers are looking into ways to make embedded IoT devices more powerful, allowing them to execute complex tasks locally.

To this end, recent years have seen a trend to augment IoT devices with embedded Field Programmable Gate Arrays (FPGA) [Intel 2019, Soliman et al. 2019], that allow to instantiate arbitrary hardware circuits at runtime. FPGAs can be used to execute tailor-made accelerators to efficiently perform complex calculations "in hardware", while having the capability to change the circuit when required. FPGA-based accelerators have been shown to increase the overall performance and efficiency significantly of applications in fields such as artificial intelligence (AI) and deep learning [Biswas et al. 2018, Han et al. 2015, Roth et al. 2020], networking [Varga et al. 2015, Zazo et al. 2015, Ruiz et al. 2019], cryptography [Chelton et al. 2008, Aysu et al. 2013, Koziel et al. 2016, Babaei et al. 2019], database systems [Ziener et al. 2016] or control systems [Monmasson et al. 2007, Montealegre et al. 2015].

These improvements can also be applied to applications on embedded system. An embedded FPGA, however, contains a comparatively small amount of resources for

circuit instantiation. This limits the size of accelerators and thus restricts the kind of application that can be supported. If a more complex accelerator is needed, the obvious solution is to use a larger FPGA with more resources, which increases the system cost as well as the power consumption.

In this paper, we propose to continue using a small FPGA and to take advantage of the reconfiguration capabilities of the FPGA. Instead of designing an accelerator as a single, monolithic entity, we propose to divide the accelerator into smaller, modular parts. The FPGA then executes the parts sequentially by reconfiguring itself for each part, passing intermediate results between the parts. We call this concept a *Temporal Accelerator*.

The paper is structured as follows. First, we discuss the idea of Temporal Accelerators in more detail, including what functional parts are needed to realise them. Then, we present an analytical model for the energy consumption of FPGA reconfigurations for different FPGAs. We briefly evaluate this model before we use it to derive interesting insights into the relationship between reconfigurations, number of reconfigurations and the amount of resulting resources for an application, showing the potential of this new approach. We wrap up the paper with a short conclusion and future work.

2 Temporal Accelerators

As discussed before, the basic idea of a Temporal Accelerator is to split an accelerator into a set of smaller, modular accelerators that are instantiated on an FPGA sequentially over time. Several aspects have to be considered to create and execute Temporal Accelerators. We will discuss these in the following section.

2.1 Design

The first step is to design a Temporal Accelerator. This can be done by a developer or by a task splitting algorithm based on an existing monolithic accelerator. A Temporal Accelerator consists of a set of interdependent *subtasks* (STs) that are arranged in a directed acyclic graph, often referred to as a *Task Graph* (TG) [Cordone et al. 2009, Knocke et al. 2014]. Each ST realises a function without side effects that consumes input data and produces output data. STs can be calculated independently and are implemented as self-contained bit files that can be instantiated on an FPGA individually. Edges in the TG represent data dependencies between STs such that if a ST produces output data that is consumed by another ST, both STs are connected with an edge in the TG.

While a Temporal Accelerator's TG can have a variety of shapes, for the remainder of the paper we focus the requirements for executing the simplest TG shape, i.e. a single, linear sequence of STs. An example can be seen in Figure 1.

We limit the following later evaluation to this simple TG structure to better understand the applicability of Temporal Accelerators in its simplest form. More complex TG containing branches are possible, but require a more sophisticated data management between reconfigurations and scheduling. We discuss the impact of more complex TGs further in Section 2.3 and 2.4.

Splitting circuit designs automatically is an open research question in itself and outside of the scope of this paper. For our purposes, we assume a VHDL developer is part of the development team to perform the necessary Temporal Accelerator designs manually. Creating a set of STs in a hardware description language such as VHDL is done using available development tools such as Xilinx' Vivado development suite [Xilinx 2021a].

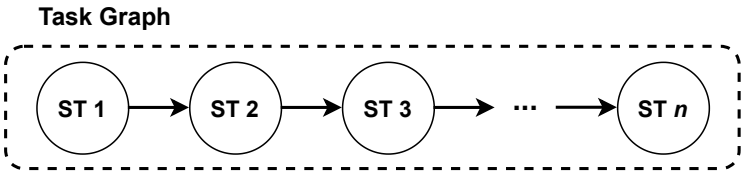

Figure 1: Example of a fully sequential Task Graph with n Subtasks (ST)

2.2 Execution model

To execute a Temporal Accelerator we first need to consider the underlying hardware platform In the following we give a brief overview of how we model the execution of a accelerators on a multi-core system containing an FPGA and how that can be extended to implement Temporal Accelerators.

Accelerators on heterogeneous multi-core system

While the general idea of using a Temporal Accelerator can be implemented using only an FPGA, the FPGA's power consumption is quite high compared to an embedded MCU. For most embedded applications not all parts require high performance hardware and as such those parts can be implemented more efficiently on a conventional MCU. Using a heterogeneous multi-core system that combines an MCU with a low-power FPGA can therefore lead to much more efficient applications [Burger et al. 2017, Burger et al. 2018, Schiele et al. 2019]. For this reason we developed our own board called the *Elastic Node* [Burger et al. 2017, Schiele et al. 2019], specifically designed for low-power embedded IoT applications.

An application on a system such as the Elastic Node consists of software components deployed on the MCU handling the application control flow, and several FPGA accelerator circuit designs that can be instantiated on demand. The structure can be seen in Figure 2. Because of the complex interactions required to manage the control flow and data management, we use a system software called the *Elastic Node Middleware* [Burger et al. 2017, Schiele et al. 2019].

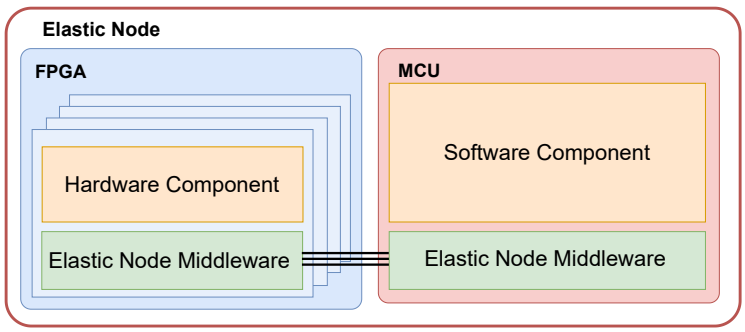

Figure 2: A Temporal Accelerator application on the Elastic Node

Applications are written in a bare-metal approach, i.e. without operating system

Listing 1: Example stub implementation to access an AI accelerator on the FPGA

```
#define LOC_AI 0x0
#define INPUT_ADDR 0x100
#define RESULT_ADDR 0x101
uint16_t cnn_execute(uint8_t input){
    elasticnode_fpgaPowerOn();
    elasticnode_reconfigure(LOC_AI);
    elasticnode_writeData(INPUT_ADDR,
        input, sizeof(input));
    elasticnode_execute();
    while(!elasticnode_isDone()){}
    uint8_t result;
    elasticnode_read(RESULT_ADDR,
        &result, sizeof(result));
    elasticnode_fpgaPowerOff();
    return result;
}
```

support, in embedded C and using our middleware components as libraries. Code listing 1 shows a simple example of using the Elastic Node Middleware to execute a single AI accelerator. This Middleware is aimed to be platform independent but was mainly developed for the Microchip AVR MCU family, compiled and tested using the AVR-GCC toolchain [Free Software Foundation 2021].

Temporal Accelerators on the Elastic Node

The Temporal Accelerator can be modeled as a set of multiple sequential calls to the middleware instantiating and then executing an accelerator one after another. The different STs are then interconnected solely by passing the result of one ST as input to the next ST. The application flow for such an execution can be seen in Figure 3.

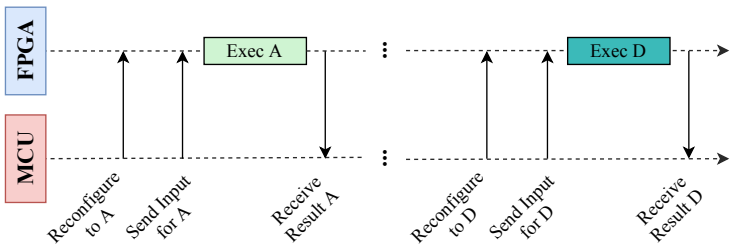

Figure 3: Execution of a Temporal Accelerator on the Elastic Node

As the FPGA does not retain any internal state during a reconfiguration, the intermediate results of a ST cannot be stored internally on the FPGA but *have to be offloaded* before a reconfiguration, and *loaded back* onto the FPGA afterwards.

2.3 Scheduling

In a first step towards using Temporal Accelerators, we currently assume that the composition of a TG is chosen at design time. Finding a corresponding schedule to deploy each ST of a TG is therefore possible at design time. Executing a TG means loading the corresponding bit file and reconfigure the FPGA accordingly, one ST at a time. Finding a schedule for our sequential TG from Figure 1 is simple as it is exactly the same order as in the TG.

In contrast, for a more complex TG containing branches, a single linear execution sequence still has to be found as only one ST can be executed at a time. This means linearising the complex TG back into a single sequential schedule that adheres to the data dependencies, often allowing for multiple valid sequences. While possible to find and execute the very first sequence, depending on the TG structure and composition, sequences may exist that have the potential to avoid unnecessary reconfigurations. By scheduling STs requiring the same bit file right after each other the necessary circuit design is already instantiated, eliminating the need for a reconfiguration. This, however, is active research.

Using the Elastic Node in a networked setting and communicating with other devices, such as in [Burger et al. 2020], introduces the possibility for the Elastic Node to receive offloading tasks from other devices. In such a scenario, we will not be able to efficient reconfiguration aware schedules at design time, but will need a dynamic runtime scheduler. This, however, is a still ongoing active research topic.

2.4 Data Management

As mentioned in Section 2.2 to execute a TG any intermediate results of a ST have to be offloaded to the MCU before a reconfiguration and loaded back onto the FPGA after it as the next input.

For a sequential TG this means that at any point in time, there is only one intermediate result that has to be stored on the MCU. Any data that is buffered will be directly used by the next scheduled ST and can freed or overwritten directly after loading it back onto the FPGA. While the amount of intermediate results is in this case exactly one, we need to allocate an appropriate amount of memory. We currently assume that the result size can be known at design time or at least an upper bound can be estimated depending on the input size of a ST.

In the case of more complex TGs containing branches this problem becomes more complex. In contrast to a sequential TG, in any case where a branching data dependency is present the intermediate result of the first ST may not directly be consumed as input by the next ST, but instead has to be buffered. This requires the MCU to handle a more complex mapping of input data to specific ST instances to ensure the correct input is sent back to the FPGA.

Due to this, we require a more sophisticated data management strategy that can handle intermediate results of complex TGs efficiently. This is another topic of active research.

2.5 Chip Cost

One of the major benefits we see in Temporal Accelerators is the ability to choose smaller chips that cost less, leading to a reduced production cost and device price for embedded applications. To illustrate this we conducted a short price comparison of the different

FPGAs from the Xilinx Spartan 7 chip family in Table 1. These values are the result of a web search [Mouser Electronics 2021] and as such may vary.

Table 1: Pricing comparison for FPGAs of the Spartan 7 chip family

	XC7S6	XC7S15	XC7S25	XC7S50	XC7S75	XC7S100
Price (€)	12.45	18.26	24.49	49.09	79.03	105.83

It shows, that the larger an FPGA is, i.e. the more resources it contains, the more expensive it becomes, increasing between 1.34-2 times for each next larger chip. When switching from a design on the XC7S25 to a Temporal Accelerator on a XC7S15 we can already reduce the FPGA cost by ∼25%.

If we can replace e.g. the XC7S100 with the XC7S50 we can reduce the cost by 46%. Temporal Accelerators can therefore significantly improve the economic viability of FPGA-based accelerator systems.

3 An Analytical Model for Reconfiguration

To be able to understand the viability of Temporal Accelerators we are currently developing an analytical model. This model allows us to gain more insights into if and when a Temporal Accelerator is a viable solution compared to using a larger chip.

In this paper we focus on analysing the reconfiguration overhead in terms of its energy consumption. Intuition dictates that, more reconfigurations during an accelerator execution should lead to more overhead, specifically due to the additional time and energy overhead for reconfigurations. This could lead to a Temporal Accelerators being so slow and energy-consuming that it becomes infeasible.

However, we have to take into account two important aspects. First, many battery-operated embedded device use a periodic sleep schedule. They will wake up e.g. once per minute, perform some actions like processing some sensor data with the accelerator, and go back to sleep. Since modern SRAM-based FPGAs cannot maintain their state during sleep, the device has to perform at least one reconfiguration each time it wakes up. This is true independently of the FPGA's size, i.e. it is necessary regardless of whether we use a classical monolithic accelerator or a Temporal Accelerator.

Second, when using a Temporal Accelerator, we are replacing a big FPGA with a smaller one. Because of its fewer resources, the corresponding bit file used to store an FPGA circuit design on the smaller FPGA is shorter as well. This in turn leads to faster reconfiguration times, which, in combination with a lower static power consumption makes reconfiguring a smaller FPGA more efficient. If an FPGA with 50% of the resources of a larger one requires less than 50% of the energy for reconfiguring than the larger FPGA, then using the smaller FPGA and reconfiguring it twice instead of once (which is the lower minimum as discussed before) actually may lead to a lower total energy consumption.

To explore this potential effect, our analytical model calculates as a first step, how high the overhead of reconfigurations are across different chips of the Xilinx Spartan 7 chip family. Based on the information provided by Xilinx' official device documentation [Xilinx 2018a, Xilinx 2018b, Xilinx 2018c, Xilinx 2019a] and the Xilinx Power Estimator [Xilinx 2020], we determined what elements influence the reconfiguration overhead. The

energy consumption for a single reconfiguration of a chip (for the Spartan 7 chip family) is dependent on three parameters: (i) the FPGA chip in question with the determined amount of resources it contains, (ii) the clock speed of the reconfiguration interface and (iii) the bus width of the reconfiguration interface. This leads to a straight-forward analytical model that we show in Equations 1 to 3.

$$RE(chip, clk, buswidth) = PR(chip, clk, buswidth) \cdot t_{bitfile}(chip, clk, buswidth) \tag{1}$$

Equation 1 describes the reconfiguration energy overhead RE as the simple product of the FPGA chip's power consumption PR during reconfiguration times the required time $t_{bitfile}$ for the FPGA to read in the corresponding bit file.

$$PR(chip, clk, buswidth) = PS(chip, clk) + PIO(chip, buswidth) \tag{2}$$

Equation 2 shows how we can model PR, i.e. the power consumption during the reconfiguration process. It consists of the chip's static power consumption PS and the power consumption of the I/O interface PIO used for the reconfiguration interface. We derived these two values from estimations provided by [Xilinx 2020]. It shows that while the interface power consumption is near identical, the static power consumption for a chip is higher, the more resources it contains.

$$t_{bitfile}(chip, clk, buswidth) = \frac{bitfileSize(chip)}{buswidth \cdot clk} \tag{3}$$

The more interesting and influential component, however, is the time $t_{bitfile}$ it takes to reconfigure an FPGA to a given bitfile. Our model for this is shown in Equation 3. We define $t_{bitfile}$ as the time required to read each bit of a bit file of size $bitfileSize(chip)$ at a clock speed clk through a hardware interface with a certain $buswidth$.

While the supported clock speeds clk and $buswidth$ are all identical for the Spartan 7 chips, the size of the bit files are directly related to how large the chip is, i.e. how many resources the FPGA contains to instantiate circuitry. That means, the more resources a chip has, the longer it takes to reconfigure it compared to a smaller counter part, with the implication that the same reconfiguration interface is used to read in the bit file.

Due to the embedded nature of the Spartan 7 chips and the prototype device we used to evaluate these numbers, we limited the reconfiguration interface to the serial SPI interface, which – for the Spartan 7 chips – supports 1-SPI, 2-SPI and 4-SPI. This means that the reconfiguration interface consists of one data line, two lines or four lines, respectively, over which the bit file itself is read through. Additionally various interface clock speeds are supported ranging from 12 MHz to 150 MHz.

The analytical model may be extended in the future with chips from other Xilinx chip families, assuming that their power supply is similarly structured to that of the Spartan chip family. However, for more performant chips it will be necessary to extend the configuration interface model to include e.g. PCIe [Xilinx 2019b] to model PC- and server-grade systems appropriately.

FPGAs from other manufacturers could potentially be included as well, if they follow a similar structure for their chips' power supply. However, more research and likely additional work would be required to appropriately represent them in this model.

3.1 Model vs. Hardware Experiment

To confirm whether our analytical model can provide appropriate predictions we ran a number of experiments on our experimental hardware platform, the Elastic Node v4 [Burger et al. 2017, Schiele et al. 2019]. We used a Spartan 7 XC7S6, a Spartan 7 XC7S15 and a Spartan 7 XC7S25 and measured the energy consumption per reconfiguration through built-in sensing circuitry. Due to the lower sampling rate of the current sensors on the Elastic Node and the comparatively fast reconfiguration times we instructed the FPGA to reconfigure itself 100 times consecutively and measured the power during that time frame. Based on the average power consumption of 100 reconfigurations we compared the energy consumption per reconfiguration against the predictions provided by our analytical model using the same reconfiguration interface settings: a Double SPI interface running at 50 MHz. The results can be seen in Figure 4.

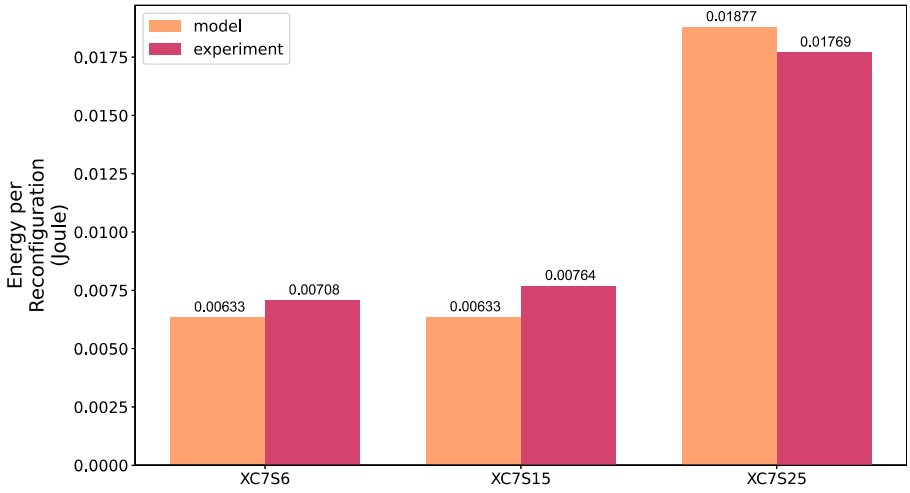

Figure 4: Model prediction vs. preliminary hardware experiment results

As the energy consumption is the product of the average power, we examined the variance on the collected power samples to ensure that the reconfigurations behave alike. The average power and variance for the three FPGAs can be found in Table 2. Differences in the variance are the result of the manual assembly process of our prototypes leading to minimal differences in the measurement accuracy, however these can be considered negligible.

Table 2: Average power consumption and Variance based on 100 reconfigurations

	XC7S6	XC7S15	XC7S25
Average Power (W)	0.1345	0.1497	0.1487
Variance	$2.76 \cdot 10^{-5}$	$4.04 \cdot 10^{-5}$	$2.07 \cdot 10^{-5}$

While not perfectly accurate we see that our model can provide us already with general insights about the trend of the reconfiguration overhead. In all cases the model does not exactly match the measured results. It is 10.5% and 17.2% lower for the XC7S6 and XC7S15 respectively, and 5.7% higher for the XC7S25. We are still analysing the causes of these differences in our predictions. One possibility is that the switching activity of the IO pins of the reconfiguration interface is only providing an average power consumption value.

We are further investigating how to improve our model to represent the power consumption during the bit file reading process more accurately.

While this model still differs from the experimental results, we believe that the model is still able to provide the general trends of an FPGA's energy consumption, or more interestingly the trend regarding the gap in energy consumption between different FPGAs.

The predicted increase in energy consumption from an XC7S15 to an XC7S25 is 2.9 times while the experimental results are 2.3 times. This allows us to narrow down potential FPGA pairs for experimental evaluations of Temporal Accelerator applications.

4 Energy efficiency of a Temporal Accelerator

To see whether our assumptions about the equal or better energy efficiency of a Temporal Accelerator has the potential to hold up, we now use our analytical model to calculate the energy consumption for all reconfiguration interface settings supported by the Spartan 7 chip family. After that we compare how many resources can be made available cumulatively when using Temporal Accelerators on a smaller chip, compared to a single reconfiguration on a larger chip. Finally, we discuss the implications of these considerations and how these resources can be used.

4.1 Analytical results

As mentioned in Section 3, intuition would dictate that the introduced overhead for additional reconfigurations makes Temporal Accelerators unviable. To disprove this, we use our analytical model to compare the energy consumption per reconfiguration for different chips of the Xilinx Spartan 7 FPGA family. For each FPGA we vary the reconfiguration interface used for reading the bit file: We compare the three available bus width settings 1-SPI, 2-SPI and 4-SPI, together with the different interface clock speeds, that are supported, ranging from 12 MHz to 150 MHz. The results can be seen in Figure 5. Although we only evaluated the model for a 2-SPI reconfiguration interface at 50MHz in Section 3.1 we wanted to investigate the potential impact of other interface settings on the energy consumption to get a better understanding of the combinations. Even if we take into consideration the inaccuracies of the model, this analysis can still provide an insight into the relationship of the different components influencing the energy consumption here.

Unsurprisingly, we can see that a reconfiguration takes less energy, if it uses more data lines for loading the bit file at a higher clock speed. More interestingly, however, if we compare identical interface settings across the different FPGA chips, in all cases the smaller chip requires less energy than the bigger ones. This effect is especially significant for fast interface settings. In these cases, the energy consumption for the smaller chip drops enough to make it viable for multiple reconfigurations. If we e.g. compare the XC7S15 to its next larger chip XC7S25 running a 4-SPI interface at 100MHz, the XC7S15

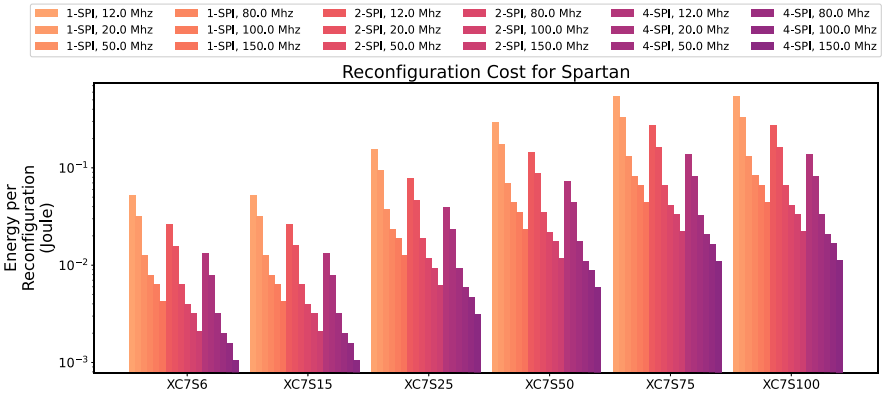

Figure 5: Cost per reconfiguration for chips of the Xilinx Series 7 Spartan family

requires 66.25% less energy for a single reconfiguration. This would allow a developer to perform up to three reconfigurations with a XCS15 before it is as costly as reconfiguring once on the XC7S25. We analyse this in more detail in Section 4.2. We can see that the faster the reconfiguration interface is, the less the overall chip's power consumption comes into play and the more the cost becomes dependent on the bit file length of a chip. As these tend to increase with a factor of approx. 2, as do the amount of resources, for larger chip this presents huge opportunities to replace those with a smaller FPGA.

It is also interesting to note that the energy consumption for reconfiguration of the XC7S6 and XC7S75 are identical to that of their direct next larger chip, the XC7S15 and the XC7S100 respectively. While both chips contain less resources than their larger counterparts [Xilinx 2018a] in both cases they use an identically long bit file [Xilinx 2018b], which results in identical bit file loading times, when using the same available interface configuration. This means that in both cases they can only provide a lower number of resources for each reconfiguration. The implications with regards to their potential feasibility to run Temporal Accelerators will be discussed in Section 4.2.

4.2 Available resources for Temporal Accelerators

With the higher energy efficiency for a single reconfiguration, we have established that smaller chips have the potential to provide a cheaper alternative to larger chips, if they can provide similar amounts of resources. Resources in this case means the registers, Lookup-Tables (LUT), Blockram blocks (BRAM) and Digital Signal Processor slices (DSP) contained in an FPGA to instantiate a given circuit design. The more are available, the more complex a single design can be.

For this reason we extended our analytical model to calculate, how many resources can be offered cumulatively with a Temporal Accelerator using multiple reconfigurations compared to all larger chips, before its energy consumption becomes as high as a single reconfiguration of the larger chip.

We show the results in Table 3. In it we compare how many resources a Temporal Accelerator deployed on a specific base FPGA can perform before it consumes as much energy as a larger FPGA and how many cumulative resources it then offers in % of the amount of resources that the larger FPGA contains. We excluded the XC7S100 as base

chip due to it being the largest chip in the Spartan 7 family and no other chip could be replaced in this group by the XC7S100.

Table 3: A base chip's available resources as Temporal Accelerator vs. larger FGPAs

Base	Larger	Reconfs	%Registers	%LUTs	%BRAM 18kb	%DSPs
XC7S6	XC7S15	1	46.88	46.88	50.00	50.00
XC7S6	XC7S25	3	77.05	77.05	33.33	37.50
XC7S6	XC7S50	6	69.02	69.02	40.00	50.00
XC7S6	XC7S75	10	78.12	78.12	55.56	71.43
XC7S6	XC7S100	10	58.59	58.59	41.67	62.50
XC7S15	XC7S25	3	164.38	164.38	66.67	75.00
XC7S15	XC7S50	6	147.24	147.24	80.00	100.00
XC7S15	XC7S75	10	166.67	166.67	111.11	142.86
XC7S15	XC7S100	10	125.00	125.00	83.33	125.00
XC7S25	XC7S50	2	89.57	89.57	120.00	133.33
XC7S25	XC7S75	4	121.67	121.67	200.00	228.57
XC7S25	XC7S100	4	91.25	91.25	150.00	200.00
XC7S50	XC7S75	2	135.83	135.83	166.67	171.43
XC7S50	XC7S100	2	101.88	101.88	125.00	150.00
XC7S75	XC7S100	1	75.00	75.00	75.00	87.50

We see from these results that there are several pairs of Temporal Accelerator enabled small FPGAs that can provide a similar, or more interestingly, an even greater amount of resources for the same reconfiguration energy overhead. However, it also shows that this depends on both the original target FPGA and the type and amount of resources that a circuit design needs to be able to decide whether to replace that target FPGA with a smaller FPGA running a Temporal Accelerator.

If we e.g. look at an XC7S15 running a Temporal Accelerator, see second group in Table 3, it can offer more registers and LUTs compared to an XC7S25, but less BRAM and DSPs for the same reconfiguration energy cost. For applications that require more BRAM and DSPs, the XC7S25 with a classical circuit design deployment may therefore be more efficient and preferable. Only when replacing an XC7S75 or XC7S100 the XC7S15 can offer more resources for the same energy cost. If the original target FPGA is an XC7S50 and the circuit design requires more BRAM and DSPs, it can be more energy efficient to instead replace it with a Temporal Accelerator on an XC7S25. Additionally, as it can provide even more of those resources, it can in theory deploy an even more complex design using the now newly available resources to improve its performance.

The XC7S50 poses an especially interesting case. Using a Temporal Accelerator, it would provide more resources of any type than both larger FPGAs in the Spartan 7 family. Thus, it could replace both of them and provide more resources for a more complex solution or higher energy efficiency. At the same time, it would be cheaper.

Another very interesting effect can be seen for the XC7S6 and XC7S75, seen in the first and last group of Table 3. Since they consume the same amount of energy per reconfiguration than their next larger counterparts (see Section 4.1), they represent potentially undesirable candidates for the Temporal Accelerator deployment and should most likely be replaced with the XC7S15 and XC7S100 respectively. Note that these observations do not take into account the lower static and dynamic power consumption as well as price. In the case of long running tasks the smaller FPGAs could still outperform their larger counterparts due to their lower dynamic power consumption during normal operation. Our numbers however indicate that these will most likely be fringe cases.

4.2.1 Sensitivity Analysis

Due to the analytical model's current differences in prediction compared to our experimental results, we performed a sensitivity analysis with regards to the amount of available resources based on varying the energy consumption per reconfiguration.

We assume for this analysis that the model may be off by 20%. Because we compare pairs of FPGAs, adjusting the energy consumption has an impact on the number of reconfigurations depending on the difference between both FPGAs. To see if Temporal Accelerators may still be viable in the "worst case" we compared the resources for chip pairs, where reconfiguration energy consumption for the smaller base FPGA is 20% higher, while for the larger comparison FPGA it is 20% lower. This results in a smaller difference, which generally leads to fewer possible reconfigurations. The results can be seen in Table 4.

Table 4: Sensitivity analysis of resource consumption vs. original results (see Tab. 3)

Base	Larger	Reconfs (vs. Original)	%Registers	%LUTs	%BRAM 18kb	%DSPs
XC7S6	XC7S15	1 (1)	46.88	46.88	50.00	50.00
XC7S6	XC7S25	2 (3)	51.37	51.37	22.22	25.00
XC7S6	XC7S50	4 (6)	46.01	46.01	26.67	33.33
XC7S6	XC7S75	7 (10)	54.69	54.69	38.89	50.00
XC7S6	XC7S100	7 (10)	41.02	41.02	29.17	43.75
XC7S15	XC7S25	2 (3)	109.59	109.59	44.44	50.00
XC7S15	XC7S50	4 (6)	98.16	98.16	53.33	66.67
XC7S15	XC7S75	7 (10)	116.67	116.67	77.78	100.00
XC7S15	XC7S100	7 (10)	87.50	87.50	58.33	87.50
XC7S25	XC7S50	1 (2)	44.79	44.79	60.00	66.67
XC7S25	XC7S75	2 (4)	60.83	60.83	100.00	114.29
XC7S25	XC7S100	2 (4)	45.62	45.62	75.00	100.00
XC7S50	XC7S75	1 (2)	67.92	67.92	83.33	85.71
XC7S50	XC7S100	1 (2)	50.94	50.94	62.50	75.00
XC7S75	XC7S100	1 (1)	75.00	75.00	75.00	87.50

In the general case we can see a reduced amount of available reconfigurations, which in return means that less resources can be made available by a Temporal Accelerator for the same energy cost.

However, even with the reduced amount of resources certain pairs of chips exist where a Temporal Accelerator on the smaller FPGA can provide a viable alternative to the larger FPGA, such as the XC7S15. Here however it is more viable for circuit designs that require more on registers and LUTs than BRAM blocks and DSPs. In the case of the XC7S25 it may not be efficient enough to fully replace larger chips, however, depending on the application scenario it may still be a viable alternative. Imagine a design that may exceed the DSP resources of an XC7S25, but only requires about 50% of the registers and LUTs of an XC7S75. Using a Temporal Accelerator would enable device designers and developers to be more flexible in reducing the overall device cost (see Section 2.5).

Even in a worst case scenario, where the Temporal Accelerator is less applicable as a full fledged replacement, it can still offer developers more flexibility in designing cheaper, tailor-made systems by extending the range of scenarios a smaller FPGA can remain useful.

4.3 Impact on the execution performance

The potential benefits presented in Section 4 come with a certain implication about how a circuit design can or cannot be structured to be converted into a Temporal Accelerator. As mentioned before, with Temporal Accelerators we can offer a cumulative amount of resources that can be equal or higher than that of larger chips. However, being a cumulative amount means that only a certain portion of the resources are available at the same time. Depending on the structure of a given circuit design, this may result in a design being impossible to convert into a Temporal Accelerator. Alternatively it may provide reduced performance, be it execution speed or data throughput of the resulting Temporal Accelerator.

Imagine an application where an image, taken by a camera, is processed by several independent components in parallel on the FPGA to provide a unified computer vision analysis of the image contents, with the implication that the components have similar execution times. While the different components can operate independently on the same image, it would be possible to split the components into two halves for a smaller FPGA and reconfigure after processing the first half. Having to do so, however, doubles the execution time and may halve the throughput of images. This may be unacceptable for low latency applications, that require a minimum amount of new information per second.

On the other hand in circuit designs with sequentially aligned components such as layers in a neural network an additional reconfiguration in between layer executions may overall not impact the latency of the execution itself and the overall execution time including reconfigurations may even be lower, depending on the circuit size.

The impact of parallelisability of components in the original circuit design on a Temporal Accelerator's performance is an open question and is currently being investigated in more detail.

4.4 Future experiments

While these initial results already show the potential of Temporal Accelerators we still need to further investigate what possibilities this technique can offer for current FPGA based applications.

We want to explore under which conditions it is more energy efficient, faster, or cheaper, respectively, to use Temporal Accelerators rather than just deploy accelerators on larger FPGAs. Here we will focus on using and extending our analytical model to explore the range of parameters and confirm these with practical experiments using our Elastic Node platform. Additional aspects that we see relevant to this are presented in the following:

Communication overhead

We described in Section 2.4 that Temporal Accelerators require an offloading off the FPGA of any intermediate results between reconfigurations. While we have shown in previous work, that the communication overhead for conventional accelerator executions is negligible compared to the performance gains, Temporal Accelerators require much more data exchanges. Here we want to investigate just how many reconfigurations, and with it intermediate results can still be offloaded before it outweighs the performance gains from the shorter reconfiguration times.

Additionally we are investigating a dual-port SRAM design that would allow the FPGA to buffer the data at higher speeds on it instead of sending it all to the MCU at it's, lower, clock speed.

Dynamic Scheduling

Making Elastic Nodes *and* Temporal Accelerators accessible to a larger system is one of the major goals we pursuing at our department. Allowing other devices to offload tasks in the form of TG onto an Elastic Node however will require a more dynamic scheduler. To reduce the reconfiguration overhead, due to the similarity of the problem space, we are looking into incorporating DNA sequencing techniques to align the TG sequences to group STs that use the same bit file.

Here we are interested in what cases we can reduce the amount of reconfigurations, which includes an analysis of TG compositions and the degree of similarity between TGs, i.e. how many STs share the bit file they require.

Memory Management

To support more complex TGs we are exploring the requirements necessary to handle the intermediate results of STs that could be "in parallel" and that have to be stored and managed across multiple STs executions before they are used.

5 Related Work

FPGA systems have a successful track record, not only in their original application area to simplify hardware design prototyping [Gschwind et al. 2001, Ray et al. 2003], but acting as a reconfigurable hardware accelerator. They have been deployed in application areas such as web based services [Brzoza-Woch et al. 2016], databases [Dennl et al. 2012, Becher et al. 2014], space applications [Montealegre et al. 2015], sensor systems [Le et al. 2004], cryptography [Chelton et al. 2008, Aysu et al. 2013, Koziel et al. 2016, Babaei et al. 2019], medical applications [Schmid 2015]. With the rise of AI and deep learning, FPGAs are also used here to improve the execution times with tailor-made designs [Han et al. 2015, Rastegari et al. 2016, McDanel et al. 2017, Biswas et al. 2018, Molanes et al. 2018, Wang et al. 2019, Roth et al. 2020].

To support more performant applications more complex circuit designs are used which need more FPGA resources to be instantiated. A common approach is to use an FPGA that can provide this amount of resources.

The reconfiguration capabilities, while often cited as a major advantage of FPGAs, are rarely used, often only in a firmware update like manner or e.g. to introduce fault tolerance [Montealegre et al. 2015]. In many cases it is not seen as a mean to expand the amount of resources.

Splitting an accelerator into multiple configurations has been done in fpgaConvNet [Venieris et al. 2016, Venieris et al. 2017, Venieris et al. 2019]. By using multiple bit files, they can recreate parts of their applications with more resources, resulting in a more performant overall execution, compared to a single bit file design with reduced complexity. However, because the split design is run on the same FPGA as the single bit file design, the reconfiguration overhead is significantly high and only amortised when performing a batch-processing of a large enough data set. They do not compare what effects using a smaller FPGA with less resources could have.

Modelling the execution components of a task into a directed graph has been known to help and schedule these tasks both on a single processing unit as well as across multi-processor systems [Hwang et al. 1993].

In various application domains partial reconfiguration has improved performance [Becher et al. 2014, Dennl et al. 2012, Koch et al. 2011, Vipin et al 2014]. By using partial reconfiguration the URUK tool chain [Aklah 2017] allows developers to compose accelerators at run-time using pre-defined and pre-synthesised building blocks and creating a Just-In-Time Assembly approach, conceptually similar to a Temporal Accelerator. However their bit streams are not for entire circuit designs but only small portions of partially reconfigurable areas. While partial reconfiguration can be quite beneficial, it does not explore applications that would require switching to a completely different circuit design, or directly analyse very resource constrained devices. Additionally, many, especially embedded, FPGAs do not support partial reconfiguration yet.

6 Conclusion and Future Work

Temporal Accelerators provide an opportunity to design high performing yet cheap embedded IoT devices. Contrary to the intuition the multiple reconfigurations used in Temporal Accelerators do not introduce an additional time or energy overhead for the concept, as smaller FPGAs require much less time to load their shorter bit files. This can reduce the cost of devices, since a smaller, cheaper FPGA can be used to create the same or even more complex applications.

This is more than a theoretical possibility. We already successfully applied our Temporal Accelerator approach to deploy a Convolutional Neural Network (CNN) that detects anomalies in ECG-data [Cichiwskyj et al. 2020]. We were able to split and execute the CNN as a Temporal Accelerator using two reconfigurations on an XC7S15 at a lower overall energy consumption compared to a “classical” deployment on a XC7S25. In this scenario our approach was not only more energy efficient but also faster in its overall execution, including reconfiguration, loading data and the execution itself. This confirms that it is possible to apply our approach efficiently in a real application scenario.

For future work, we plan to examine further application cases, such as a voice-recognition based person tracking. Additionally, we are developing new scheduling algorithms to reduce the number of reconfigurations based on DNA sequence alignment

techniques as well as a new memory management algorithms for a more efficient forwarding of intermediate data to support complex TGs. We also want to explore whether a dual-port SRAM chip, managed by the MCU could act as an intermediate buffer to further reduce the communication overhead. Finally, we want to further investigate the impact of the forced sequential split in circuit designs and how the execution pattern of the circuit design, i.e. its degree of parallelism can influence the execution performance of Temporal Accelerators.

References

[Aklah 2017] Aklah, Z.T.: "A hybrid partially reconfigurable overlay supporting just-in-time assembly of custom accelerators on FPGAs"; University of Arkansas, 2017.

[Aysu et al. 2013] Aysu, A., Patterson, C., Schaumont, P.: "Low-cost and area-efficient FPGA implementations of lattice-based cryptography"; 2013 IEEE int. symposium on hardware-oriented security and trust (HOST), 2013, 81-86.

[Babaei et al. 2019] Babaei, A., Schiele, G.: "Physical unclonable functions in the internet of things: State of the art and open challenges"; Sensors, 19(14), 2019, 3208

[Becher et al. 2014] Becher, A., Bauer, F., Ziener, D., Teich, J.: "Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration"; 24th Int. Conf. on Field Programmable Logic and Applications (FPL), 2014, 1-8.

[Biswas et al. 2018] Biswas, A., Chandrakasan, A.P.: "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks"; IEEE Journal of Solid-State Circuits, 54 (1), 2018, 217-230.

[Brzoza-Woch et al. 2016] Brzoza-Woch, R., Nawrocki, P.: "FPGA-Based Web Services–Infinite Potential or a Road to Nowhere?" IEEE Internet Computing, 20 (1), 2016, 44-51.

[Burger et al. 2017] Burger, A., Cichiwskyj, C., Schiele, G.: "Elastic nodes for the internet of things: a middleware-based approach". 2017 IEEE Int. Conf. on Autonomic Computing (ICAC), 2017 (Jul), 73-74.

[Burger et al. 2018] Burger, A., Schiele, G.: "Demo abstract: deep learning on an elastic node for the Internet of Things"; 2018 IEEE Int. Conf. on Pervasive Computing and Communications Workshops (PerCom Workshops), 2018 (Mar), 424-426.

[Burger et al. 2020] Burger, A., Cichiwskyj, C., Schmeißer, S., Schiele, G.: "The Elastic Internet of Things-A platform for self-integrating and self-adaptive IoT-systems with support for embedded adaptive hardware"; Future Generation Computer Systems, 113, 2020, 607-619.

[Chelton et al. 2008] Chelton, W.N., Benaissa, M.: "Fast elliptic curve cryptography on FPGA"; IEEE Trans. on very large scale integration (VLSI) systems, 16 (2), 2008, 198-205.

[Cichiwskyj et al. 2020] Cichiwskyj, C., Qian, C., Schiele, G.: "Time to Learn: Temporal Accelerators as an Embedded Deep Neural Network Platform"; IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning, Springer Cham., 2020, 256-267.

[Cordone et al. 2009] Cordone, R., Redaelli, F., Redaelli, M.A., Santambrogio, M.D., Sciuto, D.: "Partitioning and scheduling of task graphs on partially dynamically reconfigurable FPGAs"; IEEE Trans. on computer-aided design of integrated circuits and systems, 28 (5), 2009, 662-675.

[Dennl et al. 2012] Dennl, C., Ziener, D., Teich, J.: "On-the-fly composition of FPGA-based SQL query accelerators using a partially reconfigurable module library"; IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, 2012 (Apr), 45-52.

[Free Software Foundation 2021] Free Software Foundation, Inc.: "AVR-GCC - GCC Wiki"; https://gcc.gnu.org/wiki/avr-gcc, 16 March 2021.

[Gschwind et al. 2001] Gschwind, M., Salapura, V., Maurer, D.: "FPGA prototyping of a RISC processor core for embedded applications"; IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 9 (2), 2001, 241-250.

[Han et al. 2015] Han, S., Mao, H., Dally, W.J.: "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding"; arXiv preprint arXiv:1510.00149, 2015.

[Hwang et al. 1993] Hwang, D.J., Cho, S.H., Kim, Y.D., Han, S.Y.: "Exploiting spatial and temporal parallelism in the multithreaded node architecture implemented on superscalar RISC processors"; 1993 Int. Conf. on Parallel Processing (ICPP'93), 1, 1993 (Aug), 51-54.

[Intel 2019] Intel Corporation: "Internet of Things - Accelerating the IoT with Intel FPGAs and SoCs"; https://www.intel.de/content/www/de/de/internet-of-things/products/programmable/overview.html, 21 October 2019.

[Knocke et al. 2014] Knocke, P., Görgen, R., Walter, J., Helms, D., Nebel, W.: "Using early power and timing estimations of massively heterogeneous computation platforms to create optimized HPC applications"; 12th IEEE Int. Conf. on Embedded and Ubiquitous Computing, 2014 (Aug), 162-169.

[Koch et al. 2011] Koch, D., Torresen, J.: "FPGASort: A high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting"; Proc. of the 19th ACM/SIGDA Int. Symp. on Field programmable gate arrays, 2011 (Feb), 45-54.

[Koziel et al. 2016] Koziel, B., Azarderakhsh, R., Kermani, M.M., Jao, D.: "Post-quantum cryptography on FPGA based on isogenies on elliptic curves"; IEEE Trans. on Circuits and Systems I: Regular Papers, 64 (1), 2016, 86-99.

[Le et al. 2004] Le, C., Chan, S., Cheng, F., Fang, W., Fischman, M., Hensley, S., Johnson, R., Jourdan, M., Marina, M., Parham, B., Rogez, F.: "Onboard FPGA-based SAR processing for future spaceborne systems"; Proc. of the 2004 IEEE Radar Conference, 2004 (Apr), 15-20.

[McDanel et al. 2017] McDanel, B., Teerapittayanon, S., Kung, H.T.: "Embedded binarized neural networks"; arXiv preprint arXiv:1709.02260, 2017

[Molanes et al. 2018] Molanes, R.F., Amarasinghe, K., Rodriguez-Andina, J., Manic, M.: "Deep learning and reconfigurable platforms in the internet of things: Challenges and opportunities in algorithms and hardware"; IEEE Industrial Electronics Magazine, 12 (2), 2018, 36-49.

[Monmasson et al. 2007] Monmasson, E., Cirstea, M.N.: "FPGA design methodology for industrial control systems-A review"; IEEE Trans. on industrial electronics, 54 (4), 2007, 1824-1842.

[Montealegre et al. 2015] Montealegre, N., Merodio, D., Fernandez, A., Armbruster, P.: "In-flight reconfigurable FPGA-based space systems"; 2015 NASA/ESA Conf. on Adaptive Hardware and Systems (AHS), 2015, 1-8.

[Mouser Electronics 2021] Mouser Electronics: "Electronic Components Distributor - Mouser Electronics Germany"; https://www.mouser.de/, 15 June 2021.

[Rastegari et al. 2016] Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: "Xnor-net: Imagenet classification using binary convolutional neural networks"; European Conf. on computer vision, 2016 (Oct), 525-542, Springer, Cham.

[Ray et al. 2003] Ray, J., Hoe, J.C.: "High-level modeling and FPGA prototyping of microprocessors"; Proc. of the 2003 ACM/SIGDA eleventh Int. Symp. on Field programmable gate arrays, 2003 (Feb), 100-107.

[Roth et al. 2020] Roth, W., Schindler, G., Zöhrer, M., Pfeifenberger, L., Peharz, R., Tschiatschek, S., Fröning, H., Pernkopf, F., Ghahramani, Z.: "Resource-efficient neural networks for embedded systems", arXiv preprint arXiv:2001.03048, 2020.

[Ruiz et al. 2019] Ruiz, M., Sidler, D., Sutter, G., Alonso, G., López-Buedo, S.: "Limago: An fpga-based open-source 100 gbe tcp/ip stack"; 2019 29th Int. Conf. on Field Programmable Logic and Applications (FPL), 2019 (Sept), 286-292.

[Schiele et al. 2019] Schiele, G., Burger, A., Cichiwskyj, C.: "The elastic node: an experimentation platform for hardware accelerator research in the internet of things"; 2019 IEEE Int. Conf. on Autonomic Computing (ICAC) , 2019, (Jun), 84-94.

[Schmid 2015] Schmid, M.: "Rapid Prototyping for Hardware Accelerators in the Medical Imaging Domain"; https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/6573, 2015, 26. March 2020.

[Soliman et al. 2019] Soliman, S., Jaela, M.A., Abotaleb, A.M., Hassan, Y., Abdelghany, M.A., Abdel-Hamid, A.T., Salama, K.N., Mostafa, H.: "FPGA implementation of dynamically reconfigurable IoT security module using algorithm hopping"; Integration - the VLSI Journal, 68, 2019 (Sept), 108-121.

[Varga et al. 2015] Varga, P., Kovács, L., Tóthfalusi, T., Orosz, P.: "C-GEP: 100 Gbit/s capable, FPGA-based, reconfigurable networking equipment"; IEEE 16th Int. Conf. on High Performance Switching and Routing (HPSR), 2015, 1-6.

[Venieris et al. 2016] Venieris, S.I., Bouganis, C.S.: "fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs"; IEEE 24th Annual Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), 2016 (May), 40-47.

[Venieris et al. 2017] Venieris, S.I., Bouganis, C.S.: "fpgaConvNet: A toolflow for mapping diverse convolutional neural networks on embedded FPGAs"; arXiv preprint arXiv:1711.08740, 2017.

[Venieris et al. 2019] Venieris, S.I., Bouganis, C.S.: "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas"; IEEE Trans. on neural networks and learning systems, 30(2), 2018, 326-342.

[Vipin et al 2014] Vipin, K., Fahmy, S.A.: "DyRACT: A partial reconfiguration enabled accelerator and test platform"; 24th Int. Conf. on field programmable logic and applications (FPL), 2014 (Sept), 1-7.

[Wang et al. 2019] Wang, E., Davis, J.J., Cheung, P.Y., Constantinides, G.A.: "LUTNet: Rethinking inference in FPGA soft logic"; IEEE 27th Annual Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), 2019 (Apr), 26-34.

[Xilinx 2018a] Xilinx, Inc.: "7 Series FPGAs Data Sheet: Overview"; https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf, 2018, 07. April 2020.

[Xilinx 2018b] Xilinx, Inc.: "7 Series FPGAs Configuration User Guide - UG470"; https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf, 2018, 07. April 2020.

[Xilinx 2018c] Xilinx, Inc.: "Power Methodology Guide - UG786 (v14.5)"; https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug786_PowerMethodology.pdf, 2018, 07. April 2020.

[Xilinx 2019a] Xilinx, Inc.: "Spartan-7 FPGAs Data Sheet: DC and AC Switching Characteristics"; https://www.xilinx.com/support/documentation/data_sheets/ds189-spartan-7-data-sheet.pdf, 2019, 07. April 2020.

[Xilinx 2019b] Xilinx, Inc.: "7 Series FPGAs Integrated Block for PCI Express v3.0 LogiCORE IP Product Guide"; https://www.xilinx.com/support/documentation/ip_documentation/pcie_7x/v3_0/pg054-7series-pcie.pdf, 2019, 10. June 2021.

[Xilinx 2020] Xilinx, Inc.: "Xilinx Power Estimator (XPE)"; https://www.xilinx.com/products/technology/power/xpe.html, 2020, 26. March 2020.

[Xilinx 2021a] Xilinx, Inc.: "Vivado Design Suite"; https://www.xilinx.com/products/design-tools/vivado.html, 16 March 2021.

[Zazo et al. 2015] Zazo, J.F., Lopez-Buedo, S., Audzevich, Y., Moore, A.W.: "A PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances"; 2015 Int. Conf. on ReConFigurable Computing and FPGAs (ReConFig), 2015 (Dez), 1-6.

[Ziener et al. 2016] Ziener, D., Bauer, F., Becher, A., Dennl, C., Meyer-Wegener, K., Schürfeld, U., Teich, J., Vogt, J.S., Weber, H.: "Fpga-based dynamically reconfigurable sql query processing"; ACM Trans. on Reconfigurable Technology and Systems (TRETS), 9 (4), 2016, 1-24.