Resource-Efficient Dynamic Partial Reconfiguration on FPGAs for Space Instruments

Alexander Dörflinger, Björn Fiethe, Harald Michalik
Institute of Computer and Network Engineering (IDA)
TU Braunschweig
Braunschweig, Germany
{doerflinger, fiethe, michalik}@ida.ing.tu-bs.de

Sándor P. Fekete, Phillip Keldenich, Christian Scheffer
Department of Computer Science
TU Braunschweig
Braunschweig, Germany
{s.fekete, p.keldenich, c.scheffer}@tu-bs.de

***Abstract*—Field-Programmable Gate Arrays (FPGAs) provide highly flexible platforms to implement sophisticated data processing for scientific space instruments. The dynamic partial reconfiguration (DPR) capability of FPGAs allows it to schedule HW tasks. While this feature adds another dimension of processing power that can be exploited without significantly increasing system complexity and power consumption, there are still several challenges for an efficient DPR use. State-of-the-art concepts concentrate either on resource-efficient implementations at design time or flexible HW task scheduling at runtime. In this paper we propose a balanced algorithm that considers both optimization goals and is well suited for resource-limited space applications.**

I. Introduction

The demand for on-board processing capabilities of space missions has been increasing steadily in recent years. State-of-the-art remote sensing instruments on spacecrafts deliver vast amounts of high-resolution data at very high data rates, but the telemetry rate is very limited, especially for deep-space missions. Thus, classic ground-processing steps need to be performed already on board the spacecraft. Final physical values have to be extracted and processed by an autonomous, intelligent and reliable application, adapting itself to the changing needs. The benefits of an adaptable processing platform are a superior data yield and a reduced risk of total instrument loss [1]. An additional advantage of adaptability is the possibility to time-share resources, when dedicated functions are not needed at the same time. All this promises more efficient hardware and power utilization, which is a design driver for future robotic missions and planetary landers. This challenge can be handled by combining modern reconfigurable hardware with sophisticated optimization techniques.

Radiation-tolerant, space-suitable SRAM-based FPGAs with large logic density provide a highly flexible platform to implement highly reliable data processing for scientific space instruments and have already been proven in many space missions. The ability of SRAM-based FPGAs to support dynamic reconfiguration allows a very flexible use of the available HW platform within the very tight constraints of scientific space missions [2]. A wide range of methodological developments on FPGAs are available to combine the performance of an ASIC implementation with the flexibility of software realizations. One important development is Dynamic Partial Reconfiguration (DPR) during runtime. In this paper we evaluate the usability and capabilities of the DPR for a typical space application with its resource limited HW/SW platform.

With DPR, different hardware modules can be used in the same region of the FPGA over time, which enables further optimization of FPGA usage. However, this mechanism is also quite costly compared to static system designs. Firstly, active runtime of a reconfigurable region is reduced due to reconfiguration times. Secondly, flexibility of running different tasks on the same region is afforded at the expense of increased resource demands and resource underallocation. Resource underallocation at design time (small reconfigurable modules assigned to large reconfigurable regions) and runtime (unassigned reconfigurable regions due to scheduling stalls) is still a sophisticated optimization problem today. Particularly when scheduling HW tasks of very different sizes and diverse resource demands (logic-/memory-/arithmetic-intensive), the given restrictions limit the performance and flexibility gain of DPR. Most state-of-the-art concepts optimize for either most efficient placement at design time or highest scheduling flexibility at runtime. In this paper we propose an algorithm that balances both.

The paper is organized as follows. Problems that arise when partitioning a reconfigurable area into regions are described in Section II. In Section III, we analyze existing concepts for bindings between Reconfigurable Regions (RRs) and Reconfigurable Modules (RMs). Section IV introduces our new approach for optimum resource efficiency and scheduling flexibility. Section V provides a detailed description of the algorithm. In Section VI, we evaluate the quality of our new approach in comparison to existing strategies.

II. Problem Description

A typical SRAM-based FPGA is composed of logic (flip-flop/latch and lookup table) contained in Configurable Logic Blocks (CLBs), Block RAMs (BRAMs), Digital Signal Processors (DSPs), clock-, and I/O-related components. For simplicity, we first distinguish only three physical resource types in this paper: CLB, BRAM, and DSP. These are the most common reconfigurable resources available for space usage, such as in Xilinx 4/5-series and UltraScale+ FPGAs or the Altera Stratix V. Different HW tasks require different compositions of these three resource types, e.g., an arithmetic-

978-1-5386-3439-4/17/$31.00 ©2017 IEEE

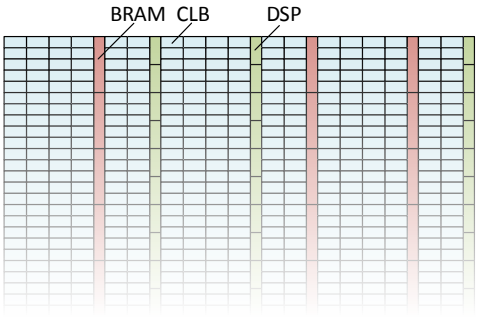

Fig. 1. Typical FPGA fabric layout.

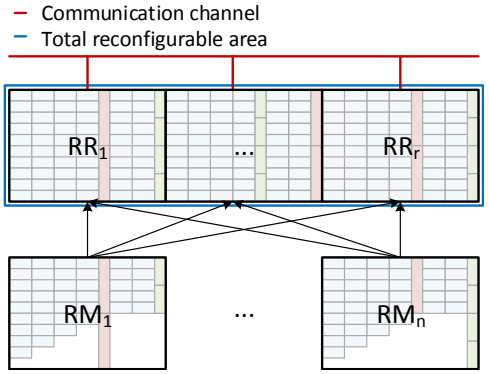

Fig. 2. Fixed uniform RR size.

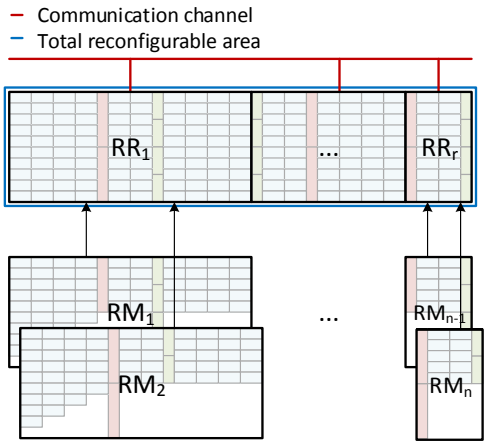

Fig. 3. Fixed non-uniform RR size.

intensive task demands a higher DSP proportion compared to a logic-intensive task.

Fig. 1 depicts a typical FPGA fabric with a uniform vertical but non-uniform horizontal layout. This non-uniform characteristic is one of the constraints that need to be considered when partitioning the Total Reconfigurable Area (TRA). For an efficient assignment of RMs to RRs at runtime, different optimization goals can be formulated. The ratio between required resources of an RM and provided resources of an RR is to be maximized (resource efficiency). For high scheduling flexibility, it has to be possible to assign an RM to multiple RRs. The arrangement of RRs should allow an easy communication channel to connect them (communication complexity).

Several concepts exist for optimum partitioning. However, when dealing with HW tasks that consist of diverse resource demands, such as the space application introduced in Section VI, these concepts still achieve poor results. Main reason for drawbacks in this use case is the assumption of a uniform FPGA fabric or uniform resource demands of RMs.

III. Related Work

Existing concepts for assigning RMs to RRs can be classified into three categorizations that favor different optimization goals. The main attributes of these categories, such as resource efficiency, scheduling flexibility, communication- and bitstream complexity are briefly discussed in the following; for a more general overview, see [3].

A. Fixed Uniform RR Size

The simplest approach for selecting RRs divides the TRA into regions of uniform size. As it should be possible to schedule each RM into any RR, the RR size must suffice to the most resource-demanding RM with respect to of the different resource types. This requirement leads to resource underallocation at design time. Furthermore, the fair distribution of resources restricts the number of RRs on a given FPGA fabric: the example in Fig. 2 allows an even partitioning of RAM and DSP resources only for three horizontally arranged RRs. On the other hand, this approach allows each RM to be scheduled in any RR and therefore offers high scheduling flexibility per RR. Resource management and scheduling strategies of this approach are discussed in [4] and [5]. The fixed structure allows a simple interfacing of the RRs to a communication channel that can be routed in the static part of the FPGA. When selecting RRs of identical resource arrangement, bitstream relocation can be applied [6] and only one bitstream per RM needs to be generated and stored. When applied, this results in low bitstream complexity.

B. Fixed Non-Uniform RR Size

The disadvantage of underallocation of resources at design time in the previous approach can be avoided by using a fixed non-uniform RR size, as depicted in Fig. 3. Several RMs with similar resource demands are clustered and assigned to one RR, which improves resource efficiency. [7] presents an algorithm for optimum clustering; for alternative approaches to the related packing problems, see [8] or [9].

The bilateral assignment between RM and RR results in a low bitstream complexity. The communication channel can be routed in the static part of the FPGA again and the number of interfaces is fixed, therefore communication complexity is low. However, with only one given RR per RM, scheduling flexibility is limited. An RM cannot be scheduled anymore once its RR is already occupied, even though there may be other free RRs, resulting in runtime underallocation.

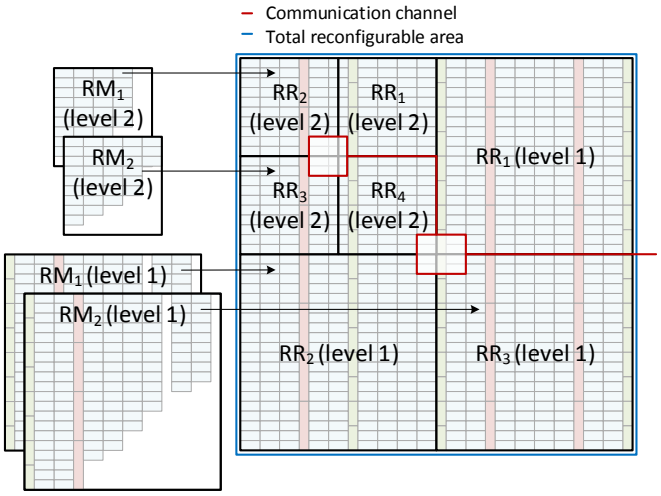

Fig. 4. Adaptive RR size.

As small modules also fit into regions reserved for larger ones, this restriction could be weakened by populating large regions with small modules. For large modules, however, it will always be difficult to provide alternative regions. This fixed non-uniform RR size approach is still quite popular in state-of-the-art (dynamic) partial reconfiguration systems. [10] discusses scheduling of HW tasks under such constraints. [11] proposed an algorithm for optimized partitioning of the FPGA fabric w.r.t. different resource types, but only static reconfiguration problems are targeted and therefore the algorithm is not suited for dynamic scheduling of HW tasks.

C. Adaptive RR Size

A more general approach allows adaption of RR size during operation, as depicted in Fig. 4. A quadtree (as proposed in [12]) or a 2D mesh (as proposed in [13]) can react to changing runtime requirements and provide RRs of flexible size. A proper scheduling algorithm minimizes fragmented free space and therefore offers maximum flexibility for placing new RMs during runtime. As the RR size is adaptable, RMs can be placed resource efficiently at design time. Therefore, this academic approach promises optimized results w.r.t. both resource efficiency and scheduling flexibility.

When transferring this concept to silicon, limitations arise. The overhead for the communication infrastructure connecting all RRs grows with the number of RR levels. In contrast to the fixed RR size concepts, the communication channel is now part of the reconfiguration area, which limits synthesis optimization. Both the non-uniform FPGA fabric and the non-uniform resource requirement of RMs make it difficult to define a standard minimum RR size. Fig. 4 illustrates this effect, because it is not possible to guarantee any BRAM or DSP resources in an RR of level 2. Because it must be possible to place an RM in any RR of corresponding level, bitstream generation and management becomes a sophisticated task. Each RM of level i can be placed in 2^i RRs, which results in a high bitstream complexity.

IV. Merging RR Size

As discussed in the previous section, state-of-the-art approaches neglect one of the concurrent optimization goals (efficiency, flexibility, complexity). In the following we propose a novel algorithm that balances all three of them and is well suited for HW tasks with diverse resource demands.

The basic idea extends the fixed non-uniform RR size approach with the possibility to merge adjacent regions. Merging adjacent regions additionally increases the size distribution of available sites for placing RMs, which promises high resource efficiency. Because RMs with high resource demands can be placed either in a large RR of sufficient size or in multiple small merged RRs, scheduling flexibility is increased. By defining the maximum number of RRs at design time, complexity of the communication channel can be confined. It also bounds the placement possibilities for each RM and thus bitstream complexity. Entrenching the communication channel into the TRA as shown in Fig. 5 keeps routing paths short and improves timing behavior.

Because the communication channel itself is part of the TRA, multiple operation modes of the system with different sets of HW tasks are possible. When transitioning from one operation mode to another, the TRA needs to be cleared first. The communication channel for the new operation mode with different locations or number of interfaces is loaded in the second step. Now the system is ready again to schedule HW tasks and place RMs. This process allows to optimize the distribution of RRs for each operation mode separately.

Merging adjacent RRs requires hierarchical reconfiguration, which is not supported by the standard Xilinx toolchain [14]. Several different solutions can still be applied: hierarchical reconfiguration is possible when using the GoAhead partial reconfiguration framework [15]. Alternatively, adjacent RRs can be connected to each other with a universal interface. RMs using multiple RRs need to be split hierarchically accordingly. Furthermore, reconfiguring the complete FPGA when changing level of RR hierarchy, is most simple but also restricts scheduling.

The challenge for this approach to be successful is to find an optimum partitioning of the TRA. The position of the RR boundaries significantly influences resource efficiency and scheduling flexibility. We have developed an algorithm that finds an optimum layout on a given reconfiguration area for a given set of HW tasks with given resource demands.

V. Algorithm

Our input consists of a rectangular TRA of width w, height h and maximum dimension $D := \max\{w, h\}$, a list of n RMs $M_1, \ldots, M_n$ and m resource types $R_1, \ldots, R_m$, such as DSP or memory resources. Each module is characterized by the amount $R_0(M_i)$ of logic cells and the amount $R_j(M_i)$ of cells of every other resource R_j required for realizing it. For each resource R_j, there are s_j resource strips $S^1_{R_j}, \ldots, S^{s_j}_{R_j}$. They can either be horizontal or vertical, but all strips of all resources share the same orientation and do not intersect.

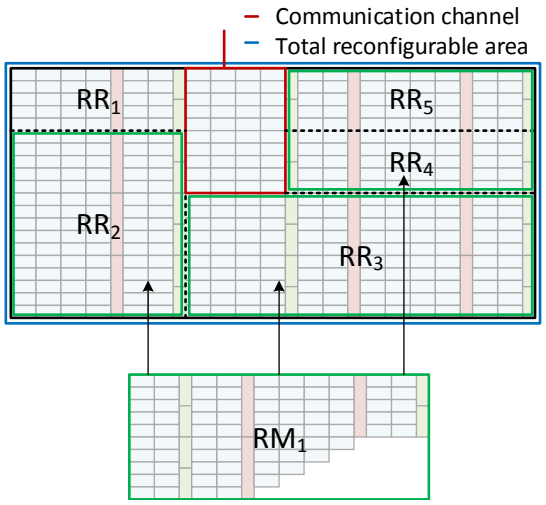

Fig. 5. Merging RR size.

Each resource strip has a position $c(S^i_{R_j})$, which is the x-coordinate of its left side (in case of vertical resource strips) and the y-coordinate of its top side (in case of horizontal resource strips). Each resource strip has a given thickness $h(R_j)$ and is subdivided into cells of size $w(R_j)$. We assume that these values are identical for all strips of a resource R_j. Moreover, the TRA has a communication channel consisting of a rectangular region; without loss of generality, this is located at the top. In the following, let c_x denote the x-coordinate of the left side of the communication channel and let c_w and c_h be its width and height. For an overview of the parameters, refer to Fig. 6.

It is important to note that, in general, the given list of modules will not fit into the TRA. Instead, the system has to react to a given *request sequence* that only arrives after the system has been configured. Each request in the sequence asks to add an RM to or remove an RM from the current system configuration. Therefore, just computing a static mapping of RMs to RRs is insufficient. Instead, we have to compute a decomposition of the TRA into non-overlapping rectangular RRs. In order to achieve low-latency communication with the outside world, we require each RR to touch the communication channel. Moreover, as we want to be able to control the complexity of the communication infrastructure, we have a parameter $N_{\mathcal{R}}$ that acts as an upper bound on the number of RRs we are allowed to use.

Additionally, for each module M_i, we have to produce a given number $p(M_i)$ of possible placements $M_{i,1}, \ldots, M_{i,p(M_i)}$. Each placement $M_{i,j}$ maps module M_i to a region of the TRA consisting of at least one RR of the decomposition. It is allowed to merge several adjacent RRs into a larger region for a module placement. For each module placement $M_{i,j}$, the resource requirements of module M_i must be satisfied, i.e., for every resource R_k, the region used by $M_{i,j}$ must contain at least $R_k(M_i)$ cells of that resource. In the following, this output will be referred to as *layout*.

Intuitively speaking, the output of the algorithm should maximize the resource efficiency and flexibility of the assignment to be able to react well to the module request sequence. This can be formalized using different goal functions. For instance, one could try to maximize the minimum number of requests that can always be served, regardless of the request sequence. Another possibility would be to maximize the number of insertion-only request sequences that can be served. Moreover, one could also try to maximize the critical resource usage, i.e., the fraction of resources that has to be used before the algorithm can fail to serve a request. However, these objective functions would require us to produce a packing strategy together with the output. Additionally, they are extremely hard to evaluate for a given layout, which makes this direct approach computationally infeasible. Therefore, we have to resort to optimizing a different goal function that can be evaluated more easily for a given layout and that is a good heuristic for the flexibility of a layout. For this purpose, we propose to minimize the *overlap depth*, i.e., the weighted number of module placements that overlap in any region of the layout. To formalize this, for each reconfigurable region $\mathcal{R}$ in the layout, let $o_i(\mathcal{R})$ be the number of module placements of module M_i using this region. Then, the *overlap depth* of a layout is

$$o := \max_{\mathcal{R}} \sum_{i=1}^{n} \frac{o_i(\mathcal{R})}{p(M_i)}.$$

An algorithm that computes a layout minimizing o can also be used to find a packing of the modules into the TRA if they fit, by requesting a layout with overlap depth 1, where each module has only one placement option.

The problems arising when optimizing over assignments of RMs to regions in the TRA are variants of geometric packing problems that are known to be computationally hard. In their general form, these problems are usually intractable in practice, even when dealing with instances of moderate size. This is in part due to the fact that solving them typically requires a representation of the packing area as a large grid.

Minimizing the overlap depth o in our setting is NP-hard[1], even when only considering logic resources. One can reduce the NP-complete problem PARTITION to it by using the communication channel to subdivide the TRA into two regions of equal height h and width w. A PARTITION instance $z_1, \ldots, z_n$ can then be solved by asking for an overlap depth $o = 1$, with n modules, using logic resource requirements $wz_1, \ldots, wz_n$ and height $h = \frac{1}{2}\sum z_i$. Therefore, on the theoretical side, an algorithm computing an optimal layout cannot be efficient in general. In practice, the restricted structure of the problem can be leveraged to compute optimal solutions for practical instances in reasonable time. In particular, the requirement that each RR touches the communication channel effectively eliminates one dimension and thus can be used to avoid representing the TRA as a grid. In this section, we present a practical algorithm for this problem, based on a formulation

[1]For an introduction to complexity theory, we refer to [16].

of the problem as an Integer Linear Program (ILP) that is constructed from the input and then solved using an integer linear program solver like CPLEX [17] or GLPK [18].

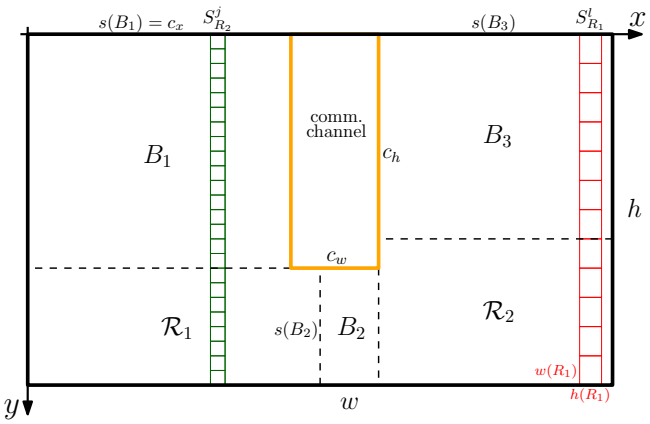

Fig. 6. The situation under consideration by the algorithm.

A. Decomposition into Regions

The communication channel subdivides the TRA into several rectangular regions, some of which may be empty if the communication channel touches more than one side of the TRA. There are three rectangular regions that may contain more than one rectangle.

1) A region B_1 to the left of the communication channel,
2) a region B_2 below the communication channel, and
3) a region B_3 to the right of the communication channel.

Moreover, there are two further rectangles $\mathcal{R}_1, \mathcal{R}_2$ in the lower left and right corners of the TRA. If $c_h = h$, these rectangles are empty and can be ignored. Otherwise, they have to touch the communication channel, implying that their width $w(\mathcal{R})$ and height $h(\mathcal{R})$ satisfy the following:

$$h(\mathcal{R}_1) = h - c_h, \quad w(\mathcal{R}_1) \geq c_x \text{ or} \tag{1}$$
$$h(\mathcal{R}_1) \geq h - c_h, \quad w(\mathcal{R}_1) = c_x, \tag{2}$$

and, analogously,

$$h(\mathcal{R}_2) = h - c_h, \quad w(\mathcal{R}_2) \geq w - c_x - c_w \text{ or} \tag{3}$$
$$h(\mathcal{R}_2) \geq h - c_h, \quad w(\mathcal{R}_2) = w - c_x - c_w. \tag{4}$$

We want to further subdivide each of the regions B_1, B_2, B_3 into rectangular RRs. One of the dimensions in these regions is always fixed, as B_1 and B_3 have fixed width and B_2 has fixed height, see Fig. 6. Let $s(B_k)$ be the size of this fixed dimension of region B_k, and $S(B_k)$ be size of the other dimension of B_k, which is fixed once the sizes of the rectangles $\mathcal{R}_1, \mathcal{R}_2$ are chosen. In the following sections, we will describe the integer linear program used by our algorithm.

B. Decomposition into Rectangles

We start by giving the variables and constraints of the ILP that describe the shape of the rectangles $\mathcal{R}_1, \mathcal{R}_2$ and the decomposition of the regions B_1, B_2, B_3 into RRs.

For the rectangle $\mathcal{R}_1$, we introduce a Boolean variable $O_{\mathcal{R}_1}$ that is zero iff Equation (1) holds. If Equation (1) holds, the height of $\mathcal{R}_1$ is fixed to $h - c_h$ and its width is described by the integer variable $S_{\mathcal{R}_1}$. Otherwise, its width is fixed to c_x and its height is described by the integer variable $S_{\mathcal{R}_1}$. Similarly, for $\mathcal{R}_2$, we have variables $O_{\mathcal{R}_2}$ and $S_{\mathcal{R}_2}$. For $\mathcal{R}_1$ and $\mathcal{R}_2$, we introduce auxiliary variables $w_{\mathcal{R}_1}, h_{\mathcal{R}_1}, a_{\mathcal{R}_1}$ and $w_{\mathcal{R}_2}, h_{\mathcal{R}_2}, a_{\mathcal{R}_2}$ encoding the width, height and area of $\mathcal{R}_1$ and $\mathcal{R}_2$. We introduce constraints ensuring that these auxiliary variables are fixed to their correct value if $O_{\mathcal{R}_1}$ and $O_{\mathcal{R}_2}$ are integral. The constraint $w_{\mathcal{R}_1} + w_{\mathcal{R}_2} \leq w$ ensures that the rectangles fit into the TRA.

For each region B_k, we have $N_{\mathcal{R}} - 2$ potential rectangles. Moreover, for each potential rectangle $\mathcal{R}^k_i, 1 \leq i \leq N_{\mathcal{R}} - 2, k \in \{1, 2, 3\}$, we have an integer variable P^k_i describing its position. For the rectangles in B_1 and B_3, this is the y-coordinate of the top side of $\mathcal{R}^k_i$. For rectangles in B_2, this is the x-coordinate of the left side of $\mathcal{R}^k_i$.

Rectangles in B_1 have to satisfy $0 \leq P^1_i \leq h - h_{\mathcal{R}_1}$. Similarly, for rectangles in B_2, $w_{\mathcal{R}_1} \leq P^2_i \leq w - w_{\mathcal{R}_2}$ has to hold, and $0 \leq P^3_i \leq h - h_{\mathcal{R}_2}$ for rectangles in B_3. Moreover, for each rectangle $\mathcal{R}^k_i$, there is a Boolean variable x^i_k that will be zero if $\mathcal{R}^k_i$ is not used, i.e., that has zero area. For these rectangles, we enforce $P^k_i \geq P^k_{i-1}, 1 < i \leq N_{\mathcal{R}} - 2$ in order to break symmetries. This also enables us to express the size S^k_i of rectangle $\mathcal{R}^k_i$ as $P^k_{i+1} - P^k_i$; for the size of the last rectangle in each region, the boundary of the region must be used. For each rectangle $\mathcal{R}^k_i$ we also include a decision variable x^k_i that is zero iff the rectangle's size is zero, enforced using the constraints

$$S^k_i \leq x^k_i \cdot D, \quad S^k_i \geq x^k_i. \tag{5}$$

We reduce symmetries by enforcing $x^k_i \leq x^k_{i-1}$ and make sure that only $N_{\mathcal{R}}$ RRs are used by imposing the constraint

$$2 + \sum_{k=1}^{3} \sum_{i=1}^{N_{\mathcal{R}}} x^k_i \leq N_{\mathcal{R}}. \tag{6}$$

C. Amount of Resources Available

For each RR $\mathcal{R}^k_i$, we need a way to express the amount of resource R_j that is available. Depending on the orientation and k, we are in one of two situations.

(I) A strip of resource R_j crosses the region B_k orthogonal to the orientation of the rectangles. For vertical resource strips, as depicted in Fig. 6, this is the case for regions B_1 and B_3; for horizontal strips, this is the case for region B_2. In this situation, resources from the strip are available in every rectangle in the region, their amount depending on the size of the rectangle.

(II) A strip of R_j crosses the region parallel to the orientation of the rectangles. For vertical resource strips, this is the case for region B_2. For horizontal strips, this is the case for regions B_1 and B_3. In this situation, resource R_j is only available in rectangles placed in the region in such a way that they intersect the strip. This introduces

new constraints on the possible placements of module M_i in B_k.

In case (I), the amount of resources can be expressed as follows. Let $N_{R_j}(k)$ be the number of strips of R_j that cross region B_k; regardless of the size of the regions, this number is known in advance. Then the amount of R_j-resources available in rectangle $\mathcal{R}_i^k$ is

$$R_j(\mathcal{R}_i^k) := N_{R_j}(k)\frac{S_i^k}{w(R_j)}. \tag{7}$$

We denote the amount of logic space available in $\mathcal{R}_i^k$ by $R_0(\mathcal{R}_i^k) := l(\mathcal{R}_i^k)$, which is defined as

$$S_i^k \cdot s(B_k) - \sum_{i=1}^{m} N_{R_j}(k) S_i^k h(R_j). \tag{8}$$

In case (II), we have to introduce additional auxiliary variables. In particular, for each rectangle $\mathcal{R}_i^k$ and each strip $S_{R_j}^l$ potentially crossing it, we have a Boolean variable $z_{j,l}^{i,k}$ indicating whether $\mathcal{R}_i^k$ is crossed by $S_{R_j}^l$. Analogous variables $z_{j,l}^1$ and $z_{j,l}^2$ model whether the strip $S_{R_j}^l$ crosses the rectangles $\mathcal{R}_1, \mathcal{R}_2$. We group the rectangles into three regions U_l, U_b, U_r, where U_l contains the rectangles from B_1 and $\mathcal{R}_1$, U_b contains the rectangles from B_2 and $\mathcal{R}_1, \mathcal{R}_2$ and U_r contains the rectangles from B_3 and $\mathcal{R}_2$. Within each region $U \in \{U_l, U_b, U_r\}$, we require each parallel resource strip to cross exactly one rectangle in U.

$$\forall U \in \{U_l, U_b, U_r\}, S_{R_j}^l : \sum_{\mathcal{R}_i^k \in U} z_{i,k}^{j,l} = 1. \tag{9}$$

This prevents strips from being split by a rectangle, which would be a waste of resources and would complicate the computation of the remaining logic area. The amount of resource R_j available in rectangle $\mathcal{R}_i^k$ and the available amount of logic space can then be expressed as

$$R_j(\mathcal{R}_i^k) := \sum_l z_{j,l}^{i,k} \cdot \left\lfloor \frac{s(B_k)}{w(R_j)} \right\rfloor, \tag{10}$$

$$l(\mathcal{R}_i^k) := S_i^k \cdot s(B_k) - \sum_{j=1}^{m} \sum_l z_{j,l}^{i,k} s(B_k) h(R_j). \tag{11}$$

For $\mathcal{R}_1, \mathcal{R}_2$, neither dimension is known in advance. Therefore, to avoid quadratic constraints, the amount of resources available has to be modeled differently. Let

$$s_{\mathcal{R}_1} := \begin{cases} w_{\mathcal{R}_1}, & \text{if resources are horizontal,} \\ h_{\mathcal{R}_1}, & \text{otherwise.} \end{cases}$$

For each resource strip $S_{R_j}^l$, we introduce auxiliary variables $v_{j,l}^1, v_{j,l}^2$ representing the width (for horizontal resource strips) or height (for vertical resource strips) of $\mathcal{R}_1, \mathcal{R}_2$ crossed by $S_{R_j}^l$.

$$v_{j,l}^1 \le z_{j,l}^1 \cdot D, \tag{12}$$

$$v_{j,l}^1 \le s_{\mathcal{R}_1}, \tag{13}$$

$$v_{j,l}^1 \ge s_{\mathcal{R}_1} - (1 - z_{j,l}^1) \cdot D. \tag{14}$$

Again we have analogous constraints for $\mathcal{R}_2$. Using these variables, the amount of resources and logical space available in $\mathcal{R}_1$ can be expressed as follows.

$$R_j(\mathcal{R}_1) := \sum_l \frac{v_{j,l}^1}{w(R_j)}, \tag{15}$$

$$R_0(\mathcal{R}_1) := l(\mathcal{R}_1) := a_{\mathcal{R}_1} - \sum_{j=1}^{m} \sum_l v_{j,l}^1 h(R_j). \tag{16}$$

D. Mapping of Modules to Rectangles

For each module $M_i, 1 \le i \le n$, each requested positioning option $M_{i,j}, 1 \le j \le p(M_i)$ and each rectangle $\mathcal{R}_h^k$, we add a decision variable $y_{i,j}^{h,k}$ that is set to 1 if placement option $M_{i,j}$ uses rectangle $\mathcal{R}_h^k$. Analogously, for the rectangles $\mathcal{R}_1, \mathcal{R}_2$, we have decision variables $y_{i,j}^1, y_{i,j}^2$, indicating whether $\mathcal{R}_1, \mathcal{R}_2$ are used by $M_{i,j}$. Empty rectangles may not be used for a mapping. This is enforced by setting $y_{i,j}^{h,k} \le x_h^k$. As merging rectangles is allowed, we have to ensure that the region allocated to a placement option $M_{i,j}$ is contiguous. This is done by enforcing, for all placement options $M_{i,j}$ and each pair $\mathcal{R}_g^k, \mathcal{R}_h^\ell$ of RRs, that if both are used for the same placement, all rectangles $B(\mathcal{R}_g^k, \mathcal{R}_h^\ell)$ between the two regions must either be empty or used, leading to

$$y_{i,j}^{g,k} + y_{i,j}^{h,\ell} - \frac{\sum_{\mathcal{R}_f^r \in B(\mathcal{R}_g^k, \mathcal{R}_h^\ell)} y_{i,j}^{f,r} + (1 - x_f^r)}{\left|B(\mathcal{R}_g^k, \mathcal{R}_h^\ell)\right|} \le 1. \tag{17}$$

E. Resource Requirements

Finally, we have to ensure that each module placement option $M_{i,j}$ satisfies the resource requirements of the module. Thus, for each module placement option $M_{i,j}$ and each resource R_l with $R_l(M_i) > 0$, we have to ensure that the required amount of resource R_l is available to $M_{i,j}$. To avoid quadratic constraints, we need auxiliary integer variables $R_{i,j,l}^{g,k}$ that model the amount of resource R_l contributed to the placement option $M_{i,j}$, i.e.,

$$R_{i,j,l}^{g,k} \le y_{i,j}^{g,k} \cdot D^2, \tag{18}$$

$$R_{i,j,l}^{g,k} \le R_j(\mathcal{R}_g^k), \tag{19}$$

$$R_{i,j,l}^{g,k} \ge 0, \tag{20}$$

$$R_{i,j,l}^{g,k} \ge R_j(\mathcal{R}_g^k) - (1 - y_{i,j}^{g,k}) \cdot D^2. \tag{21}$$

The integrality of these variables enforces that RR boundaries do not split resource cells. Analogous constraints and auxiliary variables $R_{i,j,0}^{g,k}$ can be used for the logic area. Then sufficient resources for each module placement can be guaranteed as

$$\sum_{g,k} R_{i,j,l}^{g,k} \ge R_l(M_i). \tag{22}$$

F. Objective Function

For the objective function, we need another variable o that models the maximum depth of the overlap we want to minimize.

$$o \ge \sum_{i,j} \frac{1}{p(M_i)} \cdot y_{i,j}^{g,k}. \tag{23}$$

VI. Evaluation

For evaluation of the novel algorithm and comparing the merging RR size approach to others, we first define benchmark parameters and then compare the results of an exemplary robotic space application.

A. Benchmark Parameters

Resource efficiency, scheduling flexibility, and bitstream-/ communication complexity can be measured according to the following benchmarks, with m being the number of resource types, n being the number of modules, r being the number of regions, and p being the number of RM-RR pairs.

Resource efficiency represents the proportion of resources R required by an RM compared to resources P provided by an RR. The resource efficiency of each RM-RR pair is weighted with the corresponding RR size, so the calculated value refers to the average of the TRA. Furthermore, in this context we calculate the arithmetic mean of all resource types.

$$\text{Resource efficiency} := \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{p}\frac{R_i(p_i)}{P_i(p_i)}. \tag{24}$$

For expressing scheduling flexibility, all possible insertion-only request sequences s of RMs to be placed in the reconfigurable area are examined (permutations of RMs). For each sequence, the ratio of successfully placed RMs n_p compared to the total number of RMs n is calculated. An optimum scheduling flexibility of 1 allows all modules to run in parallel, which corresponds to a non-DPR design.

$$\text{Scheduling flexibility} := \frac{1}{s}\sum_{i=1}^{s}\frac{n_{p,i}}{n}. \tag{25}$$

Bitstream complexity represents the size of the bitstream repository and required storage. Each RM-RR pair requires a bitstream of size b, corresponding to its RR (not accounting for possible compression). For better comparability, bitstream complexity is normalized to the bitstream size of the TRA.

$$\text{Bitstream complexity} := \frac{1}{b(\text{TRA})}\sum_{i=1}^{p} b(\text{RR}_i). \tag{26}$$

A rough estimate for communication channel complexity is the number of possible interfaces and therefore RRs. This applies in particular for networks-on-chip that scale well with the number of nodes.

$$\text{Communication complexity} := r. \tag{27}$$

B. Exemplary Space Application

The following space-instrument setup currently investigated at IDA is used for exemplary evaluation. The space instrument is equipped with a stereo camera system, which will be used for 1) *object recognition* and 2) high-quality *image acquisition*. Both operation modes can be divided into smaller tasks that can either be executed in SW or scheduled as an RM in HW. An exemplary estimate of resource demands for corresponding RMs is given in Table I. The estimates represent a typical wide spread of resource demands. Logic resources (slices) are generously estimated because an overhead of about 10% is required for successful placement and routing. Some RMs need to be scheduled concurrently in multiple RRs due to functional reasons (e.g., running the same algorithm on left and right image) or safety reasons (critical modules such as motor control will run in redundant or Triple Modular Redundancy mode). In this example, a TRA of 5700 slices, 60 BRAMs, and 100 DSP blocks is assumed, which correlates to available resources of two clock regions in the Zynq XC7Z020 FPGA. The resource demands of the RMs of each operation mode exceed the resources provided by the TRA, so DPR has to be performed. It is assumed that the communication channel has a resource requirement of 100 slices per interface. The remaining resources can be divided into at most two RRs of identical size for the fixed uniform approach. The fixed non-uniform approach allows a division of the TRA into multiple RRs. In this example, four RRs have been selected. For the adaptive approach, the TRA is divided into only two parts on the first level instead of four. This allows mapping the large RMs *Stereo match* and *CCSDS 122* for image data compression to the first level. The second level follows the quadtree layout again and subdivides the remaining areas into four parts.

TABLE I
Resource Requirements of HW Tasks

Mode	*Reconf. Module*	*Slices*	*BRAM36*	*DSP48*
Object recognition	Debayer (2x)	200	2	0
	Rectifier (2x)	500	30	10
	Stereo match	2500	30	10
	Disparity	1000	15	30
	Flex-SURF[1]	1000	0	10
	Motor Control (3x)	200	0	0
Image acquisition	FPN correction[2]	100	0	1
	Dark field corr.	200	1	10
	FFT	800	7	16
	Bad pixel/spike	100	2	4
	CCSDS 122	2500	12	0
	Binning	300	4	6
	Hough Transform.	1800	14	15
	Median Filter	800	0	0

[1] IP-core for feature detection using the "Speeded-Up Robust Features" algorithm.
[2] IP-core for Fixed-Pattern Noise suppression.

Table II illustrates the average benchmark results of both operation modes for the different RR size approaches that are discussed in this paper. As expected, the fixed uniform RR size approach yields very low resource efficiency. Schedulability is also comparably low in this example, which can be explained with the low number of RRs. The fixed non-uniform approach yields better results, in particular the tailored size of RRs contributes to a relative high resource efficiency. When choosing adaptive RR size, scheduling flexibility is improved, but because arithmetic- and memory-type resources cannot be

TABLE II
BENCHMARK RESULTS OF DIFFERENT RR SIZE CONCEPTS

Reconf. region concept	*resource efficiency*	*scheduling flexibility*	*bitstream compl.*	*comm. compl.*
fixed uniform	24%	23%	9	2
fixed non-uniform	44%	24%	2.18	4
adaptive	30%	30%	9	10
merging, 5 RRs	**42%**	**51%**	**3.14**	**5**
merging, 10 RRs	**52%**	**57%**	**3.20**	**10**

guaranteed to be part of an RR of deep level, only a low resource efficiency is achieved.

When dividing the TRA into 5 RRs using the merging RR size concept introduced in this paper, the algorithm yields a similar resource efficiency as the fixed non-uniform approach, but already provides higher scheduling flexibility. Bitstream complexity and thus the required size for memory is relatively low, only outperformed by the fixed non-uniform approach.

When increasing the number of defined RRs, the TRA can be divided into more finely granular areas. Thus, resource efficiency and scheduling flexibility increases. The results for the merging RR size approach with 10 RRs confirm these expectations. However, the price for this improvement is higher communication complexity. This impacts resource requirements and possibly speed of the communication channel. Also, bitstream complexity increases marginally as more placement options for an RM need to be provided.

For the given example, the new algorithm developed achieves best scores for scheduling flexibility together with high resource efficiency results without significantly increasing bitstream complexity. Communication complexity can easily be controlled by limiting the number of RRs. Therefore our new algorithm provides a good approach for the given robotic space application in particular, and for reconfigurable modules of diverse resource demands in general.

VII. CONCLUSION

In this paper, we presented a new approach and algorithm for subdividing a reconfigurable area into regions efficiently. For a given robotic space application we evaluated the results. Compared to existing concepts we achieved better results regarding resource efficiency and schedulability for HW tasks with diverse resource demands.

The algorithm can be applied for any DPR problem, given that the underlying FPGA fabric is 1D-homogeneous. So far we concentrated on the three main resource types of FPGAs (logic, memory, and arithmetic). Our model does not cover additional reconfigurable resource types yet, such as clocking- and I/O-related, which are free to reconfigure in the Xilinx UltraScale+ architecture. Such an extension is planned for the future. We also plan to consider correlations between reconfigurable modules, such as co-occurrence or mutual exclusion, which allows further optimization.

ACKNOWLEDGMENT

This work is part of the DFG Research Group FOR 1800 “Controlling Concurrent Change”. Funding for the Department of Computer Science was provided under grant number FE 407/17-2, while funding for IDA was provided under grant number MI 1172/3-1.

REFERENCES

[1] L. Fossati and J. Ilstad, “The Future of Embedded systems at ESA: Towards Adaptability and Reconfigurability,” in *Proc. 2011 NASA/ESA Conf. Adaptive Hardware and Systems (AHS)*, June 2011, pp. 113 –120.

[2] B. Fiethe, F. Bubenhagen, T. Lange, H. Michalik, and H. Michel, “Dynamically Reconfigurable Processing Module (DRPM) and its Application on a Space Science Instrument,” in *Proc. 2013 Workshop on Reconfigurable Computing (WRC)*, January 2013.

[3] A. Ahmadinia, J. Angermeier, S. P. Fekete, D. Göhringer, T. Kamphans, D. Koch, M. Majer, N. Schweer, J. Teich, C. Tessars, and J. C. van der Veen, “ReCoNodes – Optimization methods for module scheduling and placement on reconfigurable hardware devices,” in *Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications*. Springer Netherlands, 2010, pp. 199–221.

[4] A. Al-Wattar, S. Areibi, and F. Saffih, “Efficient on-line hardware/software task scheduling for dynamic run-time reconfigurable systems,” in *2012 IEEE 26th Int. Parallel and Distributed Processing Symp. Workshops PhD Forum*, May 2012, pp. 401–406.

[5] A. Rodriguez, J. Valverde, E. de la Torre, and T. Riesgo, “Dynamic management of multikernel multithread accelerators using dynamic partial reconfiguration,” in *Proc. 9th Int. Symp. Reconfigurable and Comm.-Centric Systems-on-Chip (ReCoSoC)*, May 2014, pp. 1–7.

[6] A. Lalevée, P. H. Horrein, M. Arzel, M. Hübner, and S. Vaton, “AutoReloc: Automated Design Flow for Bitstream Relocation on Xilinx FPGAs,” in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 14–21.

[7] Y. Ma, J. Liu, C. Zhang, and W. Luk, “HW/SW partitioning for region-based dynamic partial reconfigurable FPGAs,” in *2014 IEEE 32nd Int. Conf. on Computer Design (ICCD)*, Oct 2014, pp. 470–476.

[8] S. P. Fekete, J. C. van der Veen, A. Ahmadinia, D. Göhringer, F. Hurtado, and J. Teich, “Offline and online aspects of defragmenting the module layout of a partially reconfigurable device,” *IEEE Trans. VLSI Systems*, vol. 16, no. 9, pp. 1210–1219, 2008.

[9] A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. C. van der Veen, “Optimal free-space management and routing-conscious dynamic placement for reconfigurable computing,” *IEEE Trans. Comput.*, vol. 56, pp. 673–680, 2007.

[10] G. Charitopoulos, I. Koidis, K. Papadimitriou, and D. Pnevmatikatos, *Hardware Task Scheduling for Partially Reconfigurable FPGAs*. Cham: Springer International Publishing, 2015, pp. 487–498. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16214-0_45

[11] S. Fekete, B. Fiethe, S. Friedrichs, H. Michalik, and C. Orlis, “Efficient reconfiguration of processing modules on FPGAs for space instruments,” in *Proc. 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, July 2014, pp. 15–22.

[12] S. P. Fekete, J.-M. Reinhardt, and C. Scheffer, “An efficient data structure for dynamic two-dimensional reconfiguration,” *Journal of Systems Architecture*, vol. 75, pp. 15–25, 2017.

[13] T. Cervero, J. Dondo, A. Gómez, X. Peña, S. Lopez, F. Rincon, R. Sarmiento, and J. C. Lopez, “A resource manager for dynamically reconfigurable FPGA-based embedded systems,” in *2013 Euromicro Conference on Digital System Design*, Sept 2013, pp. 633–640.

[14] *Vivado Design Suite User Guide Partial Reconfiguration UG909*, v2016.4 ed., Xilinx, Inc., 2016.

[15] C. Beckhoff, D. Koch, and J. Torresen, “Go Ahead: A Partial Reconfiguration Framework,” in *2012 IEEE 20th Int. Symp. Field-Programmable Custom Computing Machines*, April 2012, pp. 37–44.

[16] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York, 1979.

[17] “IBM ILOG CPLEX Optimizer,” https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer, accessed: 2017-03-01.

[18] “GNU Linear Programming Kit,” https://www.gnu.org/software/glpk, accessed: 2017-03-01.