CHAPTER 2

BACKGROUND AND RELATED WORKS

2.1 INTRODUCTION

In this section, preliminary information regarding the proposed works is provided. Starting with advantages and limitations of reconfigurable hardware a list of applications that have gained significant benefits are discussed. Basic concepts of online task placement and prior works in the task placement domain are discussed. Consequently fragmentation issues, basic of space filling curves and need for temperature aware design concepts are presented. Finally, the literature review is provided.

2.2 PARTIALLY RECONFIGURABLE FPGA ARCHITECTURE

The development of small, fast, power-efficient, low-cost, and flexible systems is still a challenge for chip designers. In order to ensure compatibility with the growing number of data protocols and multimedia applications in the mobile communication market, there is a need for high system flexibility. One way is to provide flexibility to integrate all application-specific hardware modules like dedicated protocol controllers and data stream decoders onto a single chip. However, large chip area is in contradiction to low-cost demands. If there are only a small number of hardware modules being executed at the same time, chip area and power consumption could be saved. Dynamically partially reconfigurable FPGA provide a solution. Sequential execution of modules in a time-division
multiplex manner enables the implementation of large designs into a moderately sized FPGA at the same chip location.

For prototype implementations of dynamically reconfigurable architectures, Xilinx FPGAs are used in almost all cases since they offer the highest logic density and performance compared to other fine grained dynamically reconfigurable devices (Brown et al 1995). Dynamic Partial reconfigurable (DPR) is not supported on all FPGAs. For example in Xilinx, Spartan 3, Virtex II, Virtex II Pro, and Virtex 4 and above series are supported.

Virtex FPGAs consist of Configurable Logic Blocks (CLBs), Input-Output blocks (IOBs), block RAMs, clock resources, programmable routing and configuration circuitry, which are placed in rows and columns. The functionality and interconnection of these elements can be (re)configured at run-time by means of a configuration bit stream which is loaded into the configuration memory of a Virtex chip either via its Boundary Scan or its Select-MAP interface.

The configuration memory can be visualized as a rectangular array of bits. The array is divided into vertical frames, which represent an atomic configuration unit that can be independently written or read. Frames are grouped together into larger units, called columns. Each column, except the columns for the block RAM, include the configuration bits for the logic and the interconnects. Unfortunately, there is no information available on which bits contain the interconnect data and which contain the CLB logic configurations.

Reconfigurable systems are typically divided into static and dynamic hardware. While the static hardware consists of components which are needed at any time (e.g., interface hardware or memory), the dynamic
hardware provides dynamically reconfigurable resources to which partially runtime reconfigurable modules can be loaded dynamically. The partitioning of the design into static and dynamic module is to be done manually by the designer. A synthesizable VHDL Register transfer logic (RTL) model is available for all possible configurations of the partially runtime reconfigurable module. The system configurations of both static module and partially runtime reconfigurable module are known in advance. Because the smallest configuration unit is a frame, the physical size of the partially runtime reconfigurable module must always be a multiple of a frame size.

During initialization of a Virtex FPGA a complete initial configuration containing the static and the default partially runtime reconfigurable module is loaded into the device. Upon completion, FPGA operation is started, but the configuration interface will be kept open. When run-time reconfiguration of the partially runtime reconfigurable module is to be performed, the partial bit stream for this module is loaded into the FPGA via the configuration interface without suspending the operation of the static module in the FPGA. This bit stream only replaces the configuration information of the partially runtime reconfigurable module. The Configurable Logic Block (CLB) interface macro ensures that the signals between the static and partially runtime reconfigurable modules remain properly connected and use the same routing resources.

In principle, all modules could be of arbitrary shape. Nevertheless, all implementations in the literature consider only modules with rectangular shapes. On the one hand, this is caused by the limited support for dynamic reconfiguration of currently available design tools, which only support rectangular shapes. On the other hand, it is much easier to solve the task of finding available and suitable resources on a two dimensional CLB array for rectangular modules rather than for arbitrarily shaped modules.
Figure 2.1 FPGA schematic with rectangular 2D module placement

Although modules always consist of a contiguous set of CLBs, the configuration data cannot necessarily be loaded into the FPGA continuously. Xilinx Virtex FPGAs can only be (re-)configured in a column-wise manner. This means that reconfiguring only a few CLBs (or IOBs, etc.) implicates a reloading of the configuration data of all other configurable elements in the affected columns. By loading module A to the FPGA as shown in Figure 2.1, the columns 6 to 10 have to be reconfigured. This implicates reloading the configuration data of the columns 5 to 10 of module B. This means, loading this module will also affect the columns 6 to 10 of module B, which currently uses columns 5 to 10. Thus, a partial bit stream has to be composed from the configuration data of two modules sharing identical columns during run-time.

In general, partial bitstreams consist of command and data sequences. The data sequences define the functionality of the reconfigurable resources while the command sequences define which resources will be reconfigured. Speaking of modules this means that the command sequences of an according bitstream define the position on the FPGA to which a module
will be loaded. This implies in turn that loading one module to two different positions requires two different bitstreams.

2.2.1 Communication Macros

Like static FPGA designs, dynamic modules can freely use the available resources within a given area. Exceptions are the routing resources used for signals, which are crossing the module’s borders. These signals, e.g., bus signals used for communication, have to be continued within neighbouring modules or static components. The position on a module’s border where a signal leaves or enters the module can be considered a module pin. To allow the exchange of modules during run-time, all pins used for communication or for dedicated signals have to be at the same positions for all modules. In the Xilinx design flow this can be realized by means of macros that explicitly define the routing resources to be used for a given signal. Only if the signals that are leaving or entering a module are realized with macros, and if the same macros are used for the implementation of all modules, a communication across module borders can be guaranteed among any neighbouring modules.

2.2.2 Advantages of Reconfigurable Hardware

The following are the benefits that come with Dynamic partial reconfigurable FPGAs

Diverse Applications: A wide range of applications ranging from network routers, Software Defined Radios (SDR), Encryption, space crafts, Robotics, automobiles, scientific data acquisition, and medical imaging systems can benefit from the reconfigurable computing (Garcia et al 2006). SDR are programmable to implement a variety of protocols, potentially even those not yet introduced (Rauwerda et al 2004).
Increased system performance: Although a portion of the design is being reconfigured, the rest of the system can continue to operate. There is no loss of performance or functionality with unaffected portions of a design (no down time).

The ability to change hardware: Partially reconfigurable FPGA can be updated at any time, locally or remotely. This feature allows easy service and updating of hardware in the field.

Hardware sharing: Because partial reconfiguration allows you to run multiple applications on a single FPGA, hardware sharing is realized. Benefits include reduced device count, reduced power consumption, smaller boards, and overall costs.

Shorter reconfiguration times: Configuration time is directly proportional to the size of the configuration bit stream. Partial reconfiguration allows making small modifications without having to reconfigure the entire device. By changing only portions of the bitstream as opposed to reconfiguring the entire device, the total reconfiguration time is shorter.

2.2.3 Challenges faced by Reconfigurable Hardware

Reconfigurable hardware faces a number of challenges if it is to become a commonplace in embedded systems. First, the reconfigurable computing is not a common technique in commercial hardware. This problem is gradually being overcome with the introduction of reconfigurable computing in certain embedded areas, such as network routers, high-definition video servers, automobiles, wireless base stations, and medical imaging systems. The other challenges are listed.
2.2.3.1 Technology and tools

Different FPGA devices offer different capabilities. To benefit from the maximum functional density while keeping sufficient performance the designer should select the most convenient technology for a specific application. Some FPGAs like Virtex-II can be reconfigured only whole columns at once. Software support, while improving, is not yet at the level required for widespread adoption of embedded reconfigurable computing. In most cases the computations to be implemented in software and the computations to be implemented in hardware must be specified separately in different languages, and compiled with different toolsets. Continued research in effective hardware-software co-design is essential to improve the ease of application design for embedded reconfigurable systems. The back-end tools also have some practical limitations.

2.2.3.2 Performance

The dynamic reconfiguration increases the functional density, but it is achieved with the cost of a decrease in the application performance. The designer must optimize these two parameters. The performance must overcome other implementation technologies (Digital Signal Processing (DSP), microcontrollers, etc.) at least in another important parameter (power dissipation, size, weight, cost, design time, etc.).

2.2.3.3 Low power

Many embedded devices are battery powered, increasing the importance of power efficiency. Computations on FPGAs typically consume less power than equivalent software running on embedded processors, but more power than ASICs. Several energy-efficient reconfigurable architectures
have been specifically developed to reduce power dissipation. Multiple supply voltages \((V_{dd})\) or threshold voltages \((V_t)\) can also improve energy-efficiency in Reconfigurable hardware. Reducing \(V_{dd}\) decreases dynamic power, while increasing \(V_t\) decreases leakage power.

### 2.2.3.4 Fault tolerance

Faults can be divided into two categories: permanent and transient. Fabrication faults and design errors are among the permanent faults. Transient faults, commonly called Single Event Upsets (SEUs), are brief incorrect values resulting from external forces, like cosmic rays, etc. Reconfigurable Hardware can increase tolerance of permanent physical faults because the hardware is modifiable to potentially compensate for these faults. Unfortunately, these SRAM bits, along with any other hardware used to provide flexibility, such as multiplexers, tri-state buffers, and pass transistors, are additional failure points not present in ASIC equivalent circuit implementations, and increase the chip area to present a larger target to radiation particles. Using properly-shielded radiation-hardened devices can minimize SEU errors. Unfortunately, these devices are expensive and generally use less advanced technologies than their unshielded counterparts. Triple Modular Redundancy (TMR) can detect and correct faults in circuits implemented in FPGAs. Other techniques include scrubbing and checksum of the configuration data.

### 2.2.3.5 Real time support

Many embedded systems require real-time operation. Hard deadlines represent tasks critical to system operation, causing system failure if missed. In dynamically reconfigurable systems, the operating system must take into account not only task types, deadlines, task location, resources and
task configuration time. Other techniques are configuration prefetching, configuration bitstream compression and relocation of tasks.

2.2.3.6 Design security

Developing high-quality hardware cores for embedded systems are time-consuming and expensive to develop and verify. Furthermore, since the hardware designs frequently reside in a configuration bitstream loaded at start-up or at runtime into the FPGA, designs can be intercepted and reverse-engineered. Both Altera and Xilinx have implemented configuration encryption in their commercial products ensure security of the intellectual property of core developers.

2.3 TASK PLACEMENT AND SCHEDULING

Current reconfigurable devices have the ability to reconfigure parts of their hardware resources without interrupting normal operation of the remaining fabric. The placement algorithms need to find locations for placing arrival tasks and to maximize the utilization of the resources. Careless placement of incoming tasks causes portions of chip area to be wasted because they are too small to hold another incoming task. Diessel et al (1999) was the first to identify several opportunities and aspects of task placement on an FPGA. Their technical report laid the foundation for most of the work in the area. They stated the reason behind the need for an operating system to control the function of reconfigurable devices. They also raised some questions and issues on scheduling and placing such as task sizing, task placement, pre-empting tasks, and inter-task communication. Most of the work done in this area targets the reconfigurable devices made by Xilinx, particularly the Virtex family of FPGAs.
2.3.1 System and task model

Figure 2.2 Reconfigurable operating system model

A model of a reconfigurable operating system is shown in Figure 2.2. The target architecture comprises a host processor and the reconfigurable device (Ahmadinia et al 2004b). Arrived tasks are stored in a queue where they await further processing. The resource management is implemented by three modules running on the host, the scheduler, the placer, and the loader (not shown in figure):

- **Scheduler**: The scheduler receives incoming tasks and uses a scheduling algorithm to assign starting times to the tasks. The scheduler relies on the placer as a sub function.

- **Placer**: The placer manages the free space on the device and tries to find feasible placements for the tasks. A poor placer leads to a badly fragmented device with many small areas scattered over the surface. A good placer achieves a lower fragmentation of the device surface which allows for the execution of more tasks in parallel.

- **Loader**: Whenever a suitable location for a task has been found, the loader is called to conduct all steps necessary for configuring this task onto the reconfigurable device.
The nexus of scheduling and placement is the main characteristics of the problem of mapping tasks to partially reconfigurable devices (Sherwani 1993). There is no guarantee that the placer finds a feasible placement for every task that has been selected for execution. As a consequence, the scheduling algorithm must incorporate a placement algorithm.

Most of the work in the area of reconfigurable design assumes a homogeneous FPGA area in which the task can be placed anywhere on the chip with no constraints. Many FPGA chips are not homogenous but there are some buses, RAM modules, DSP clocks and some multipliers between the CLBs (Koester et al 2006). The assumption of using homogeneous structure was made for several reasons. One of the reasons is that the scheduling problem on 2D model is a hard problem and it is useful to study an easier version of the problem first as assuming heterogeneous structure will make it harder.

The problem of scheduling and placing a set of tasks on an FPGA chip is similar to the classic bin packing problem that has been attacked by many researchers in the scheduling field where the goal is to pack a collection of objects into the minimum number of fixed-size "bins". Researchers have proposed many algorithms for placement under a variety of assumptions about tasks and the reconfigurable device. Some of the assumptions are based on FPGA area model and task parameters.

Usually a non-pre-emption model is assumed for the placement algorithm. The non-pre-emption model means that once a task has been loaded onto the device it runs to completion (Walder et al 2003). Although pre-emption is a very useful technique that yields efficient and fast online scheduling algorithms, it is too expensive for current reconfigurable technology. Pre-empting and resuming tasks require context switches. Compared to processor based systems where context switches involve saving
rather small task control blocks, including the program counter and other registers, the context of a reconfigurable task comprises the states of all state holding logic elements of the task’s circuit. A reconfigurable task’s context can easily amount to several Kbytes of data. Pre-empting and restoring such a task would not only introduce a considerable delay, but also require additional external memory for storing contexts.

2.3.2 FPGA Models used for Placement

According to the hardware task’s placement algorithm the FPGA can be modelled as below (Walder & Platzner 2003):

*Figure 2.3 Various FPGA Area Models*
**1D Variable Area Model:** Figure 2.3(a) shows 1D Variable model in which each task spans the full height of the reconfigurable device, but their width is variable. A module can have any width (with a minimum of one CLB column) and can be placed at any location. Current implementations of dynamically reconfigurable systems with 1D placement usually use widths of two or four CLB columns. Both of the internal and external fragmentation appears in this model. The internal fragmentation problem comes from the synthesis tools and platform related properties. The 1D variable model makes each placement more flexible than the 1D Fixed model, however, which also needs more complex placement algorithms.

**1D Fixed Area Model:** As shown in Figure 2.3(b), each hardware task occupies a fixed-size slot, and the slots span the full width of the reconfigurable device. The advantages of this model are their simplicity and fixed communication structures, and system developers can easily design an efficient placement algorithm to avoid external fragmentation problems. However, this model brings a limitation that to decide the slot size and the slot number are architecture dependent and application dependent. Another disadvantage of this approach is that it fixes the amount of resources available and reserved per module. While this wastes resources for smaller modules (leading to internal fragmentation), big modules cannot be implemented if they exceed the size of the slots.

**2D Variable Area Model:** The 2D variable model depicted in Figure 2.3(c) is more flexible than the 1D model, which can allocate an arbitrary rectangle task at any position of the device. Owing to 2D area model only suffers the external fragmentation, better hardware utilization is desirable, but to design an efficient scheduling and placement method becomes more complex.

**2D Fixed Area Model:** This model shown in Figure 2.3(d) is less flexible than the 2D variable model. This method places tasks inside, already
partitioned blocks. This may lead to wasted area if the size of the task is less than the size of the block. Discovering the possible location is faster compared to old methods.

2.4 INTRODUCTION TO FRAGMENTATION

The online placement algorithms need to find locations for placing arrival tasks and to maximize the utilization of the resources. Careless placement of incoming tasks causes portions of chip area to be wasted because they are too small to hold another incoming task. Consequently, area fragmentation is one of the biggest obstacles of obtaining good utilization of chip resources. In software domain this problem occurs in memory allotment technique for which efficient techniques have been developed. Due to dynamic addition and deletion of tasks the empty area on the runtime partially reconfigurable FPGA will be split into a large number of small sized regions which cannot accommodate even an average sized task. This phenomenon is called fragmentation. Fragmentation leads to lower area utilization, more task rejection and slower execution of tasks.

Figure 2.4 shows two fragmentation states having the same number of empty slots, but with different distribution. Clearly seen, the first case will provide a better chance of placement for the incoming task.

Figure 2.4 Two Fragmentation cases having same number of vacant spaces
Fragmentation can be classified into several types (Handa & Vemuri 2004b, Pellizzonni & Caccamo 2006, 2002). They are

Internal fragmentation: This is the empty area created inside the boundary of rectangular task by trying to fit the actual logic of the task into a rectangular area.

External fragmentation: This is the fragmentation of empty area outside the boundary of the rectangular tasks. When the sum of the remaining spaces is sufficient for the placement, but the placer rejects the task due to the remaining spaces are not contiguous. This has been already explained in Figure 2.4.

Virtual fragmentation is a situation in which the placement algorithm fails to locate contiguous empty space, even though such a space exists. This can be solved by improving the efficiency of placement algorithm.

Partition fragmentation: This will occur only if FPGA surface is divided into a finite number of predefined sub-areas called blocks and only a single task can be placed in any of these blocks. Empty area occurs if the task placed in smaller than the block size. It is difficult to find an optimal size of the block. In the literature several works have been attempted with fixed and variable sized blocks, splitting and merging of blocks etc.

An empty area is represented by a list of overlapping maximal rectangles. The classical model assumes that fragmentation exists only when a placement request is rejected. (But fragmentation is always there). Fragmentation is the ability to detect the aptness of FPGA area to accommodate future tasks. Some metrics mix occupation degree with fragmentation. An FPGA with a high occupation, but with all free area
concentrated as a single rectangle cannot be considered as fragmented as some algorithm classifies.

2.5 THERMAL MANAGEMENT ON CHIP

As the modern reconfigurable devices consume more power compared to ASIC solutions, the researchers spent effort to tackle this issue. One of the most important defects of the power is the heat that produced within the chip. As this heat increase, a number of problems that affect the final application implementation occur. Among them, the reliability of the system, the packaging of the chip and the lifecycle of the final product are some critical issues. Moreover, as the FPGA power consumption continues to increase, low-power FPGA circuitry, architectures, and CAD tools need to be developed.

The total power consumption in the FPGA can be defined as: $P_t = P_{st} + P_{sc} + P_{dy}$, where $P_{st}$ denotes the static power that a circuit consumes even when it is in the standby mode, $P_{sc}$ denotes the short-circuit power, which is due to the short-circuit current flowing from the power supply to ground when both p-network and n-network are ON, and $P_{dy}$ denotes the dynamic power consumption that occurs when the output signal of a CMOS logic cell makes a transition.

Traditionally the Integrated Circuit (IC)'s power consumption was due to dynamic power consumption. Consequently, in order to decrease the frequency while increasing ICs throughput, the solution was to parallelize processes. Leakage power consumption is strongly correlated to the circuit area. Therefore a new trade-off between parallelism and power consumption should be found. Leakage power represents a significant share of the total power dissipation, especially for 65 nm technology and below. If in the past, FPGAs were mainly concerned with dynamic power consumption, it remains
no more true with new technologies (65 nm and below) and with the large
number of transistors inside an FPGA circuit: the leakage power consumption
becomes more and more important both in the used and unused part of the
component.

Field-Programmable Gate Arrays (FPGAs) like Xilinx’s Virtex5
and Altera’s Stratix III, based on a 65 nm design process and 12 copper
interconnect layers. The post fabrication flexibility provided by these devices
is implemented using a large number of prefabricated routing tracks and
programmable switches. These interconnects can be long, and can consume a
significant amount of power. In addition, the programmable switches add
capacitance to each wire-segment, which further increasing their power
dissipation. Finally, the generic logic structures consume more power than the
dedicated logic in ASICs. The power consumed by the FPGA will lead to heat
generation and in turn to the rising of the overall die temperature. This
underlines again the importance of embedding temperature reduction
techniques in the main design flow.

Four factors determine the temperature of an FPGA die: total power
dissipation, package thermal resistance, ambient temperature, and airflow.
These factors must be managed to stay below the maximum junction
temperature of the die. The designer usually has control over at least one of
these factors. If it is determined by thermal analysis that the die will exceed
its maximum temperature limits, the designer may choose to reduce the clock
speed, improve airflow, change the package, or add a heat sink. Given the
temperature inside the case or rack \( T_{\text{amb}} \), the total power dissipation of the
FPGA \( P \) and the thermal resistance of the package \( R_{\text{th}} \), the junction
temperature \( T_J \) may be calculated using Equation 2.1.

\[
T_J = T_{\text{amb}} + (P \times R_{\text{th}}) \quad (2.1)
\]
This shows that the die will experience a higher temperature than the ambient temperature. The total power dissipation (P), multiplied by the thermal resistance of the package (R_{th}), is the difference between junction and ambient temperature. The goal of thermal planning is to keep the die temperature below the maximum rated value. For commercial grade parts, the maximum value is 80°C and for industrial grade, 100°C. Exceeding this temperature will result in a degradation of performance; timing values in the data sheet will no longer be guaranteed. For both commercial and industrial grade parts, there is an absolute maximum temperature rating of 125°C. If this temperature is exceeded, device reliability will be compromised.

Consider a design with a power dissipation of 5 watts. Let the ambient temperature be 40°C and thermal resistance of the package is 10.0°C/Watt. Substituting in Equation 2.1, the die temperature will be 90°C, which exceeds commercial device maximum temperature rating. Fans or heat sink may be used to decrease the effective thermal resistance from the die to the environment. Therefore, temperature management is very important.

\[ T_j = 40°C + 5 \text{ Watts} \times (10.0°C/\text{Watt}) = 90°C \]

The fact that FPGA functionality is programmable implies that a diverse range of possible thermal gradients could exist during operation. Different temperatures in different parts of the chip cause the switching speed of the transistors to vary, which may cause timing difficulties. Pecht (1991) has stated that every 10 degree rise in temperature will decrease reliability by half. Because FPGAs are manufactured to accommodate a large number of possible designs, it is not possible to know the locations of hot spots apriori during manufacturing time as hot spot locations depend on the design and the design’s layout configured into the FPGA.
The temperature of an IC depends on the function of the chip as well as the data it is operating upon. For example, if an IC is capable of decompressing JPEG as well as PNG (Portable network Graphics) images, depending on which image type needs to be decompressed, different areas of the chip will get heated up. Thus the occurrence of a hotspot is not a certainty, but a matter of probability dependent upon some factors that are out of the designer’s control. Hence, temperature aware placement design is crucial.

### 2.6 SPACE FILLING CURVES

A space-filling curve can be thought of as a map from one-dimensional space onto a higher-dimensional space. Space filling curves is used in DBMS, geographical research, computer art, etc. In 1890, Peano found a curve that maps the unit interval \([0, 1]\) into the plane with the image having positive Jordan content. This curve has been called the first Space-Filling Curve (SFC). Other SFCs soon followed with Hilbert’s in 1891, Moore’s in 1900, Lebesgue’s in 1904, Sierpinski’s in 1912, and Polya’s in 1913. To be able to characterize their properties, Mokbel & Aref (2001) concentrated on the nature of the segments that connect adjacent points in the space-filling curve by cataloguing them as a Jump, Contiguity, Reverse, Forward, and Still. Different applications will require different space-filling curves. For accessing a database by an index in which the order is relevant, a space-filling curve that preserves the order of the dimension will probably be the best (row reverse, i.e., the Peano curve). It turns out that there are identifiable relationships between the types of segments in each SFC.

Abel & Mark (1990) have identified desirable qualitative properties of spatial orderings. Locality means that a SFC never leaves a region at any level of refinement before traversing all points of that region. Thus
neighbouring data in a multidimensional space remain neighbours after linearization. This clustering of data is important in exploiting cache memory during computation and for the partitioning of computational grids. The recursive behaviour of SFCs allows for the linearization of recursive hierarchical data structures. Space filling curves are classified into recursive and non-recursive SFC. A Recursive Space Filling Curve (RSFC) is a Space filling curve that can be recursively divided into four RSFC of equal size. The 2-regular curves are obtained by subdividing d-dimensional hypercube into $2^d$ subcubes, and 3-regular curves obtained by subdividing d-dimensional hypercubes into $3^d$ subcubes. The important space filling curves are Peano curve, Z-order curve, Hilbert curve, RBG curve, snake curve, scan curve, Serpenski curve, dragon curve etc.

Some other quantitative properties of curves are: the total length of the curve, the variability in unit lengths (path between two cells next in order), the average of the average distance between 4 neighbours, and the average of the maximum distance between 4 neighbours. Goodchild (1989) proved that the expected difference of 4-neighbour keys of a n by n matrix is $(n+1)/2$ for Peano, Hilbert, row, and row-prime (snake) orderings. This does not appear to be a very discriminating property. Therefore, Faloutsos & Roseman (1989) derived a better measure for spatial clustering called average (Manhattan) maximum distance of all cells within N/2 key value of a given cell on an N*N grid. Another measure for clustering is the average number of clusters for all possible range queries. A cluster is defined as a group of cells with consecutive key value.

The four curves are used in this work is shown in Figure 2.5. They are chosen because their curves have good clustering and match with the requirement of the space filling curves. The curves used are Z-order curve,
Hilbert curve, Reflected Binary Gray curve and Peano curve. The properties of each curve and the generation of curves are explained below.

Figure 2.5 Important space filling curves

The first curve used in this work is Z-order curve. It is also called Morton code, N code, quad code, bit interleaving code. Z value of a cell is calculated interleaving the row and column indices, which are represented in binary as shown in Figure 2.6. The bits of row index fall in the odd positions of the binary representation of Z; the bits of column index fall in the even position. The basic step contains four points in the four quadrants of space. The second iteration contains four blocks of first iteration visited in the same order at finer resolution. The points are visited in ascending order according to their binary number representation. The Z-order SFC is due to its definition almost everywhere differentiable. On the contrary, it lacks the locality property all previous curves possess, meaning that it jumps across the unit-square.

Figure 2.6 Two dimensional Z-order and RBG curve encoding
A small variation lead to another code called Reflected binary gray code. In this the row and column indices are encoded in gray code. After interleaving the gray code is converted to binary code for getting the decimal value. Unlike Z-order curve, the first and the fourth blocks in second iteration has same orientation as in first iteration but second and third are constructed by rotation the basic block by 180 degrees. The templates and the translation scheme used to generate higher order Hilbert curves are shown in Figure 2.7. The last one is a Peano curve, which is a 3 regular Space filling curve that can be recursively divided into nine RSFC of equal size.

![Translation scheme for the four templates](image)

![Generation of Hilbert curve](image)

**Figure 2.7 Method to generate higher order Hilbert curve**

### 2.7 LITERATURE REVIEW ON TASK PLACEMENT

Current strategies dealing with task placement are divided into two categories: offline placement and online placement. In an offline solution, the
scheduling decision is optimized when the application is compiled. In an online solution, the information of each task (e.g., execution time, configuration time) is unknown until it arrives. The online solution provides more adaptivity to various applications and avoids the application profile step, which is time-consuming (Gerez 1999). The online scheduler should, at runtime, assign a required time period to the arrival task. During this time period the task can be loaded and execute on the FPGA. The efficiency of the online scheduler will directly impact the overall performance of the whole system.

Offline problems assume that all information about all tasks is available initially, and the algorithm is looking for the best placements for all of them to satisfy certain criteria. Fekete et al (2001) has shown the offline placement problem for a general set of tasks (tasks with no constraints) is NP-complete, so it is unlikely that an algorithm exists to obtain an optimal solution to the problem in less than exponential time. Since a solution to the online scheduling problem implies a solution to the offline problem, it can be concluded that the online problem is also NP-complete.

### 2.7.1 Prior Works in Offline Task Placement

By considering the placement of hardware tasks as rectangular items on a hardware device as rectangular unit, several approaches for resolving the two-dimensional packing problem are proposed. Baker et al (1980) suggested the Bottom-left offline algorithm, which packs each hardware task in the bottom left position. Lodi et al (1997) investigated different offline approaches to resolve a hardware task placement as 2D bin packing problem. For instance, the Floor-Ceiling algorithm that considers alternate directions for packing tasks, either from left to right when their bottom edges touch the level floor or from right to left; when their top edges are on the top of the level floor. The Knapsack packing algorithm was also
suggested (Lodi et al 1999) which initializes each level by the tallest unpacked item and completes it by packing tasks as the associated Knapsack problem that maximizes the total area within level.

Martello & Vigo (1998) used an enumerative offline approach to exact solution for 2D bin-packing. Their algorithm is based on a two-level branching scheme: the outer branch-decision tree that assigns tasks to the bins without specifying their position and the inner branch-decision tree that enumerates all possible patterns.

Bazargan et al (2000b) defines 3D templates depicting the tasks in time and space dimensions and uses slow heuristics: simulated annealing and greedy research, and KAMER-BF to perform a high-quality placement in terms of resource utilization and task rejection.

An offline approach of the packing of a set of items into a single bin using a graph-theoretical characterization was presented by Fekete et al (2001). Tasks are presented as three-dimensional boxes and the feasible packing is decided by the orthogonal packing problem within a given container. Their approach considers packing classes, precedence constraints, and the edge orientation to solve the packing problem. Similarly, Teich et al (2001) defines the task placement as higher dimensional packing problem. Tasks are modelled as 3D polytopes with two spatial dimensions and the time of computation. Based on packing classes as well as on a fixed scheduling, they search a feasible placement on a fixed-size chip to accommodate the set of tasks. The resolution is performed by Branch and Bound technique to the optimality of dynamic hardware reconfiguration.

Panainte et al (2005) models the problem of resource allocation as a 0-1 integer linear programming problem, which aims necessarily at minimizing the resources area which is reconfigured at runtime. They
consider an application with known sequential execution trace. Hence, the huge configuration latency is tackled by reducing the overlapped areas between tasks. Belaid et al (2010) suggested a three level resource management that improves placement quality by reducing task rejection, configuration overhead and optimizing resource utilization. They formulated the problem into a constrained optimization problem and solved using branch and bound technique.

### 2.7.2 Prior Works in Online Task Placement

The most intuitive placement method is the brute force algorithm, which considers all possible locations in a device for an incoming task, but its time complexity depends on the width (W) and height (H) of the reconfigurable device and the number of running tasks (n), that is, $O(W \times H \times n)$. Moreover, the bin-packing heuristic methods of placement algorithms are First-Fit (FF), Best-Fit (BF), and Worst-Fit (WF). Chazelle et al (1983) implemented a bottom-left heuristic for two dimensional bin-packing. They proved that their algorithm can pack (insert) ‘n’ rectangles in an infinite height vertical bin of fixed width with the bottom-left heuristic in time $O(n^2)$, using $O(n)$ space. This algorithm complexity is for insertion only without deletions.

Brebner (1996) introduced the idea of an operating system with the ability to manage FPGAs. Wigley & Kearney (2002) discussed a partitioning algorithm that includes placement and routing. Placement algorithms to find out a near optimal solution were based on simulated annealing algorithm and the genetic algorithm. Although a high quality solution can be achieved, they consume much time. Also they were not suitable for online task placement algorithms.
In literature due to limitation of the FPGA hardware the existing works used various assumptions. One of the main assumptions is regarding the FPGA model used. Some of them used one dimensional model, which matches well with FPGA having column based frames while others assumed the two dimensional model for which hardware support was not truly available. Similarly to reduce complexity, concepts like relocation, pre-emption of task, arbitrary shape of tasks, rotation of tasks etc. were not used much. Most of the works focus only on placement assuming that routing can be done.

2.7.2.1 Models based on one dimensional FPGA models

Most of these algorithms were targeting the Xilinx FPGA which had column based frame structure for which one dimensional area model will suit well. Steiger et al (2003a) was the first to present two heuristics for online scheduler: the Horizon and Stuffing algorithms. They first presented the schedulers to work on the 1D model, and then they showed how to change these algorithms to work on the 2D model. The Horizon guarantees that arriving tasks are only scheduled when they do not overlap in time or space with other scheduled tasks. The Stuffing schedules arriving tasks in arbitrary free areas that will exist in the future by imitating future task terminations and starts. The authors reported that the Stuffing algorithm outperforms the Horizon algorithm in scheduling and placement quality. The drawback of this algorithm is it always tends to place task on the left most of its free space.

Walder & Platzner (2003a) introduced their online scheduler for 1D reconfigurable device, which were statically divided into several width-fixed blocks and schedules tasks according to several non-pre-emptive and pre-emptive policies. The length of a hardware task configuration is not considered in the 1D area model. In the 1D reconfiguration model, it is easier to decide where to place a hardware task. However, the 1D model cannot
exploit the full reconfigurable chip area, and thus, results easily in external fragmentation and internal fragmentation when the length of a task is small.

Chen & Hsiung (2005) modified the stuffing algorithm (Steiger et al. 2003a) by classifying incoming tasks before scheduling and placement. This is called as 1D Classified Stuffing as it performs better than the original 1D Stuffing. The Classified Stuffing can place a task on the leftmost or rightmost of its free space based on the task Space Utilization Rate (SUR). SUR is the ratio between the number of columns required by the task and its execution time. High SUR tasks (SUR > 1) are placed starting from the leftmost available columns of the FPGA space, while low SUR tasks (SUR ≤ 1) are placed from the rightmost available columns.

Marconi et al. (2008b) introduced their 1D Intelligent stuffing to solve the problems of both the 1D Stuffing and Classified Stuffing. The main difference between their algorithm and the previous 1D algorithm is the additional alignment flag of each free segment. The flag determines the placement location of the task within the corresponding free segment. By utilizing this flag, the 1D Intelligent Stuffing outperforms the previously mentioned 1D algorithm. Lu et al. (2010) introduced their 1D Reuse and Partial Reuse (RPR). The algorithm reuses already placed tasks to reduce reconfiguration time. As a result, the RPR outperforms the 1D Stuffing.

2.7.2.2 Models based on two dimensional FPGA models

As older FPGA devices did not allow fine-grained 2D placement, researchers have made some assumptions and designed their 2D schedulers based on them. Bazargan et al. (2000a) was the first to accomplish this using a fast online placer, but that placer was not an optimal one. Their method
included a partitioning manager for insertion and deletion and a search engine and a bin-packing rule. The partitioning method cuts the empty region into several free rectangles. These free rectangles were candidates for accommodating an incoming task. The favoured location for an incoming task was based on the bin-packing rule. They used two kinds of free space management methods, namely maximal empty rectangles and non-overlapping rectangles. An illustration of both schemes is shown in Figure 2.8. Finding maximal rectangles takes quadratic space in terms of the number of running tasks, whereas finding non-overlapping rectangles needs linear space.

![Figure 2.8 Illustration of KAMER and KNER algorithm](image)

Walder & Platzner (2002) suggested online placement using footprint transform. When a task arrives, the placer first searches for a direct placement using First-Fit or Best-Fit. If no direct placement is possible, then the placer tries footprint transform to modify the shape of the tasks. Compared to relocation, footprint transforms are complex requiring re-routing of larger number of wires. On an average, footprint transform improved execution time by 8.7% over the First-Fit algorithm. They also introduced a hash matrix data structure to maintain the free space. Using the hash matrix
they were able to find the feasible location in constant time (Walder et al 2003b).

Steiger et al (2003b) improved the above method using an On The-Fly (OTF) algorithm. As mentioned earlier, the wrong split decision can lower the placement quality of KNER. To avoid such wrong decisions, they modified KNER by delaying the split decision until a next arrived task placed on the FPGA. However, for this modification, they need to resize several rectangles on a task insertion. They also suggested a hash matrix approach to find placement in constant time, but updating this data structure is time consuming. A free rectangle of size a x b is associated with the entry [a, b] in this array. Every entry consists of a pointer to a list of free rectangles of the corresponding size. Four types of fitter are considered; Best-Fit, Worst-Fit, Best fit with exact fit and Worst-Fit with exact fit. Among all rectangles which can accommodate task BFEF/WFEF choose the smallest/biggest rectangle which has exactly the same width or height as the tasks.

Ahmadinia et al (2003) introduced temporal task clustering algorithm. The algorithm minimizes the number of frames to be reconfigured by clustering the modules that have to be reconfigured at the same time in consecutive frames. There are two algorithms a) close run time clustering b) communication trade off clustering. It is not possible to combine both because their properties are against each other.

Ahmadinia & Teich (2003) also suggested the Least Interference Fit (LIF) algorithm. In order to reduce the reconfigurable overhead due to the limitation of currently available FPGA technology in that time (column-wise reconfigurable capability), LIF places tasks at the position where the tasks interfere with the currently running tasks as little as possible.
Ahmadinia et al (2004a) suggested a method in which two horizontal lines are maintained, i.e., one above, and another below the running tasks. The authors claim 20% improvement over Bazargan KAMER method. Ahmadinia et al (2004b) presented the Nearest Possible Position (NPP) algorithm. They manage the occupied space rather than the free space, because the set of empty rectangles grows much faster than the set of placed rectangles. The Impossible Placement Region (IPR) of an arriving task relative to a placed task is the region near the placed task where it is impossible to place this arriving task without overlapping the placed task. The Possible Placement Region (PPR) is the area where it is possible to place the arriving task without overlapping any placed tasks. In order to find the best position on the PPR for placing an arriving task, they compute the routing cost based on Euclidean distances and place an arriving task at the optimal point, where routing cost is minimized. If they cannot find the optimal point on the PPR, they will find the Nearest Possible Position (NPP) for placing the task. This algorithm does not take into account area fragmentation or time and data constraint during scheduling.

Ahmadinia et al (2004c) demonstrated the Routing-Conscious Placement (RCP). In order to reduce free-space management to a single point, they expand inserted modules and concurrently shrink the FPGA area and an arriving task by half both in width and height. To choose the position for placing an arriving task, they choose a position, such that the weighted communication cost computed based on the Manhattan distances is minimized. Finally they suggested a fast placement algorithm having a complexity $O(n\log n)$. Their approach (Ahmadinia et al 2007) is by considering occupied space as ‘$n$’ placed rectangular modules using contour of union of rectangle (CUR) algorithm.
Handa & Vemuri (2004a) presented an algorithm which aims at finding empty space fast and efficiently. They made use of two data structures namely area matrix and staircase data structure to model the FPGA resources and to manage empty space, respectively. In an area matrix representation, each cell of the matrix was assigned a negative value for occupied space and a positive integer number for empty space. A positive number in a cell gives the number of contiguous empty cells above and including that cell in that column and a negative number in an occupied cell gives the remaining width of the task measured in the right direction. Worst case space complexity is $O(\min(x, y))$ and time complexity is $O(xy)$ where $x$ and $y$ are the number of columns and rows in an FPGA.

Tabero et al (2004) managed the free space for devices with a vertex list set that stored the free space fragments in an FPGA. They achieved the goal of minimizing fragmentation by two heuristics, namely the adjacency-based heuristic and the fragmentation-based heuristic. A vertex list set describes all the available FPGA free space, with a different vertex list for each FPGA Holes. This is shown in Figure 2.9. Vertex selector checks whether a feasible position exists where the task could be mapped. The vertex is chosen among candidates according to the selected heuristics. The candidates can be bottom left, bottom right, top left and top right. Then the task is inserted and vertex list set is updated by the vertex list updater.

Huang & Vemuri (2005) introduced online synthesis for partially reconfigurable FPGAs. The main idea is to include physical placement information into high level synthesis to ensure that synthesised design can be placed in the available empty area on the FPGA.
Tomono et al (2006) suggested an online FPGA algorithm that does not only take into consideration the degree of fragmentation, but also the speed of I/O communication computed based on the Manhattan distances. The purpose of their algorithm is to balance the degree of fragmentation and the speed of I/O communication. They use the same area matrix data structure as used by the Staircase algorithm (Handa & Vemuri 2004a) with additional I/O communication constraints, so they increase the degree of fragmentation in order to gain the speed of I/O communication. Because of this additional consideration, they need to check the status of each communication channel during staircase creation.

In order to implement task placement algorithm for heterogeneous reconfigurable architectures Koester et al (2006) suggested a new technique. The basic idea of this algorithm is to avoid placing an arriving task with many feasible positions in areas that can be used by tasks with few feasible positions whenever possible. A Fragmentation-Aware Placement (FAP)
algorithm was introduced by ElFarag et al (2006, 2007b). They introduced a fragmentation metric that gives an indication to the continuity of occupied (or free) space on the reconfigurable device and not the amount of occupied (or free) space. The algorithm places each arriving task at the location where the fragmentation metric is smallest. All empty spaces have to be tested before it can select one that causes the lowest fragmentation.

Zhou et al (2006) demonstrated their 2D Window-based stuffing to tackle the drawback of 2D Stuffing. By using time windows instead of the time events, the 2D Window-based Stuffing outperforms previous 2D stuffing. The drawback of their 2D Window-based Stuffing is its long execution time. To reduce this runtime cost the authors presented the Compact Reservation (CR), which is the computation of the Earliest Available time (EA) matrix for every incoming task. The EA matrix contains the earliest starting times for scheduling and placing the arriving task. The CR outperforms the original 2D Stuffing and 2D Window-based Stuffing.

Cui et al (2007a, 2007b) and Gu et al (2009) introduced the Scan Line Algorithm (SLA) algorithm for finding MERs. The authors use the same area matrix as the staircase algorithm with a different encoding to represent the FPGA area. Every free area CLB is represented by a positive number that gives the number of continuous empty cells on the left including that cell. Every occupied CLB is represented by a zero. In SLA, the area matrix is used for finding Maximum Key Elements (MKEs) and finally these MKEs are scanned for finding all MERs. A key element is an empty cell with an occupied cell as its right hand neighbour. A column that contains one or more key elements is called as a scan line. The SLA algorithm can find empty rectangles that are not MERs. They proposed Enhanced SLA algorithm
(ESLA) to solve this problem. The algorithm records a set of scanned positions during each MKE scanning to avoid duplicated scanning.

Cui et al (2007b) combined the scan line algorithm with Fragmentation Matrix (FM) to create a Cell Fragmentation (CF) algorithm. Fragmentation Matrix (FM) is used to represent the area of the FPGA. For empty cells, each cell is labelled with the number of contiguous empty cells in horizontal, in vertical direction, and a zero. For occupied cells, each cell is labelled with the number of contiguous occupied cells in horizontal, in vertical direction, and the finish time of the task. In order to place a task on the FPGA, CF calculates the Time-Averaged Area Fragmentation (TAAF) for all MERs that are large enough to accommodate the task and then places the task into one of the MERs which has the largest TAAF. A MER with large TAAF means it has more impact on the overall degree of fragmentation, which is why CF places the arriving task to a MER that has the largest TAAF.

Lu et al (2007) introduced an Immediate Fit (IF) algorithm. The 2D FPGA surface is initially partitioned into various size blocks based on application requirements. There are three types of split and merge operations. The initial partition is adjusted according to different application according to incoming tasks. A split only process splits a large size block into smaller size blocks. If all ‘A’ size block is occupied, when another A size task arrives. A ‘2A’ size block can be split into two A size block, which can be used for the new input task. The reverse is merge operation. If all ‘2A’ size blocks are occupied and another ‘2A’ size task arrives then split a ‘3A’ block and merge with a neighbouring A block to create two ‘2A’ size blocks, which can accommodate the new task. The authors (Lu et al 2007) also suggested a new model (PQM-IC) to assess the placement quality. The real part and imaginary
part of the complex number format indicates resource wastage and task rejection, respectively.

Deng et al (2008) introduced 2D and 3D adjacency techniques. The algorithm places the tasks densely so that the resulting vacant space is a large continuous area. Morandi et al (2008) developed a Routing Aware Linear Placer (RALP) algorithm. The algorithm is a modified version of KNER algorithm with an additional routing cost consideration between dependent tasks. Tasks are placed on empty rectangles that have the least Manhattan distances between dependent tasks to minimize routing cost. The algorithm can reduce routing cost compared to KNER but has a lower placement quality due to the negative impact of its routing consideration.

Marconi et al (2008a) investigated three techniques for speeding up online placement algorithms: Merging Only if Needed (MON), Partial Merging (PM), and Direct Combine (DC) were proposed. Furthermore, Intelligent Merging (IM) algorithm was developed by applying these techniques. Instead of merging all fragmented free space blocks, the algorithm only merges on demand to decrease the runtime overhead. The algorithms that use splitting and merging in managing free area need to merge free area for accommodating arrival tasks.

Marconi et al (2009) also suggested new strategy for online placement algorithm on 2D partially reconfigurable devices, termed the Quad-Corner (QC). They spread hardware tasks close to the four corners of the devices. To accommodate tasks, algorithm search four task lists: upper left corner for very large tasks, upper right corner for large task, lower bottom corner for medium tasks and lower right corners for small tasks. There are two main advantages of this strategy: (1) it reserves a lot of free area in the middle of the device; (2) it solves splitting free area problem. As a result, both
the reconfigurable device utilization and the system performance will be increased.

Lu et al (2009) has also presented a flow scan model for online task placement. They use four different linked lists; general edge linked list, in edge and out edge linked list, rectangular well linked list. For each placed task the lower and upper Y coordinate indicate in edge and out edge, respectively. The default values are the bottom and top of the FPGA. Rectangular wells are temporary rectangle without top line created during the scanning process. A rectangular well that can be expanded only in the upward direction is called formed rectangular well. A maximum free rectangle is a rectangle whose four edges cannot be expanded. There are two scan procedures: in edge processing happen when a scanning flow reaches in edge and out edge processing occur while leaving the out edge. The authors claim that on an average this algorithm is 1.5x faster than staircase and 5x times faster that SLA.

Elbidweihy & Trahan (2009) focussed on an online placement algorithm that manages both Maximal Horizontal Strips (MHS) and Maximal Vertical Strips, called the Maximal Horizontal and Vertical Strips (MHVS) algorithm. MHSs are rectangles generated by partitioning free area using top and bottom boundaries of running tasks; whereas MVSs are rectangles that result from free area partitioning using left and right boundaries of running tasks. In this algorithm, the First-Fit rectangle is used for placing an arriving task. The algorithm can run faster compared to KAMER but has a lower placement quality than KAMER.

Redaelli et al (2009) developed three variations of module mapping based on the knowledge of the workload. Offline-online Collaborative scheduling algorithm is used if complete static knowledge of task sequences is available. Static grid scheduling requires knowledge of maximum resource
requirement by any task expected to be executed. The last method Dynamic NoC grid scheduling requires no prior knowledge of any kind. All these are well suited for Network on chip (NoC) paradigm. The authors further developed ILP formulation to solve task scheduling program (Redaelli et al 2008).

Lee et al (2009, 2010) developed an adaptive free space management technique. The method includes two procedures to search candidate space for online placement. The first procedure (C-Look) searches rectangular candidate space that has to meet the form of incoming task. The second procedure Continuous Space of Arbitrary Form (CSAF) searches nonrectangular candidate that has to equal or exceed the area of incoming task.

A Multi-Objective Hardware Placement (MOHP) algorithm was introduced by Lu et al (2010). Incoming tasks are classified into three groups with different treatments. The first group is for independent tasks that need to be executed urgently due to short remaining time to the deadline. To handle tasks in this group, the algorithm uses the First-Fit heuristic for fast allocation. The second group is for independent tasks that do not need urgent treatment. For this group, the algorithm adopts a vertex-list approach. The third group is for dependent tasks. In this group, the routing between dependent tasks needs to be shortened. For that reason, the algorithm utilizes the NPP approach for tasks in this group.

Lu & Marconi (2010) introduced a Communication Aware online task Scheduling Algorithm (CASA) that has three steps. When scheduling an arriving task the modified flow scan algorithm is used to find all the FTSS (Free Time Space Slot) in the strip model. Then these FTSS are checked and adjusted if any conflict with the configuration port is found. Finally the task is scheduled into a FTSS according to its communication requirement. The
CASA use locked communication scheduling. If the task is successfully scheduled, all the needed communication buses have to be assigned to the task during the required period (locked).

Virtual strip packing was introduced by Angermeier et al (2010) where modules loaded onto the reconfigurable device may be exchanged at runtime. It can be seen from the literature survey that the performance of the placement algorithm depends on the FPGA area model. This necessitates the creation of a new data structure that can represent the free space compactly as well as can be updated easily. Bassiri & Shahhoseini (2011) pointed out that configuration reusing can reduce reconfiguration overhead considerably. The reusing based scheduling algorithm divide the FPGA area into two virtual partitions and schedule arriving task on different partition according to their significance. Simulation resulted in low rejection ratio and reconfiguration overhead.

Iturbe et al (2011) suggested an EAC (Empty Area Compaction) First-Fit algorithm aimed at preserving MER and manages the device as a single resource instead of splitting it into non-realistic independent pieces of area. This functioning is favoured especially when the device is partially damaged. Hong et al (2011) increased acceptance rate on a permanent damage chip using EAC-First-Fit method.

Marconi (2013) suggested a technique called pruning moldable in which multiple selectable hardware implementation for each hardware request is incorporated. The selection reflects trade-off between required reconfigurable resources and the task runtime performance. The method starts with slowest hardware version. Then a starting time matrix is constructed. If the smallest starting time is greater than the deadline the next faster hardware
version will be taken. This will ensure that the fragmentation will be always less.

A heuristic algorithm based on Particle swarm optimization was introduced for online task placement (Premalatha et al 2013) on two dimensional homogeneous FPGA. Iturbe et al (2013) focused on a 2D model and developed a finishing aware earliest deadline first algorithm that consider the plastic time between finish time and deadline. Finally a Communication aware Maximum adjacent edges based on a new 2D model is proposed by Sheng et al (2014). They demonstrated that the communication cost has been reduced by 17%.

2.7.3 Literature review on Fragmentation

In the literature several methods exist for fragmentation estimations. These works were mainly developed for memory fragmentation. Some of these methods are adapted for the partially reconfigurable placement scenario. When the vacant area on the FPGA can be represented as a run-length format, one dimensional fragmentation metrics can be used. ElFarag et al (2007) developed a method in which $F = \frac{1}{m_1} + \frac{1}{m_2} + \cdots$ where $m_i$ indicates the size of $i^{th}$ free block. The same technique can be extended to 2D area model by applying the formula to both rows ($F_r$) and columns ($F_c$) and take the sum as shown in Equations (2.2, 2.3 and 2.4). For an 8x8 array the maximum and minimum fragmentation values are 64 (checker board pattern) and 2 (fully vacant case), respectively.

$$F_r = \frac{1}{m_1} + \frac{1}{m_2} + \cdots$$  \hspace{1cm} (2.2)
\[ F_c = \frac{1}{n_1} + \frac{1}{n_2} + \cdots \]  \hspace{1cm} (2.3)

\[ F = F_r + F_c \]  \hspace{1cm} (2.4)

Gehr & Schneider (2009) suggested the following methods. The simplest one is shown in Equation 2.5. The parameter \( n \) indicates the number of free blocks. However, the size of each free block is not considered. Because of these two scenarios having the same number of free spaces but different free block size will show the same level of fragmentation.

\[ F = 1 - \frac{1}{n} \]  \hspace{1cm} (2.5)

\[ F = 1 - \frac{f_{\text{max}}}{\sum_{i=1}^{n} f_i} \]  \hspace{1cm} (2.6)

\[ F = 1 - \frac{f_{\text{avg}}}{\sum_{i=1}^{n} f_i} \]  \hspace{1cm} (2.7)

\[ F = 1 - \frac{\sum_{i=1}^{n} f_i^p}{\left(\sum_{i=1}^{n} f_i\right)^p} \]  \hspace{1cm} (2.8)

The limitation of the above method is overcome by Equation 2.6, by considering \( f_{\text{max}} \) which is defined as the size of the largest free block. The disadvantage is that a single block of very large size will influence the fragmentation levels. The average free block size \( f_{\text{avg}} \) is used in Equation 2.7.
instead of the maximum size. This is mathematically identical to first one
since \( f_{avg} = \frac{1}{n} \sum_{i=1}^{n} f_i \). Equation 2.8 is less sensitive to the influence of large
free blocks. Parameter \( p \) can be 1, 2…n.

When the FPGA is represented as a two dimensional model,
there are several techniques. These techniques are closely linked with the
placement algorithm and the data structure used to represent the vacant space.
For algorithms in which vacant space is represented as empty rectangles,
Wigley & Kearney (2002) considered the average size of Maximal Square
fitting into MERs. They calculated the mean of the distribution of shorter size
of empty rectangles. This method has less discrimination because it gives
same value for several fragmentation situations. Walder & Platzner (2002)
considered a set of non-overlapping rectangles used to cover an empty area
denoted by \( a_i \). Fragmentation is given by Equation 2.9.

\[
F = 1 - \frac{\sqrt{\sum_{i} a_i^2}}{\sum_{i} a_i} \tag{2.9}
\]

Handa & Vemuri (2004b) calculated FCCx and FCCy (Equation
2.10 and 2.11) for each cell in the MER. Using these Total Fragmentation
Contribution of a Cell (TFCC) is calculated by adding FCCx and FCCy.

\[
FCC_x = 1 - \frac{v_x}{L_x - 1} \quad \text{if } v_x \leq L_x \text{ otherwise } FCC_x = 0 \tag{2.10}
\]

\[
FCC_y = 1 - \frac{v_y}{L_y - 1} \quad \text{if } v_y \leq L_y \text{ otherwise } FCC_y = 0 \tag{2.11}
\]

where \( L_x(L_y) \) are average width of the tasks being placed and \( v_x(v_y) \) is the
number of consecutive empty cells in the horizontal and vertical direction of
the current cell. Total fragmentation is the average value of TFCC of all cells
in the MER. The task is placed in any of the corners of MER having the
largest total fragmentation and corner chosen to maximize the total
fragmentation of task sized rectangle. The authors does not mention about the
overall fragmentation index.

Vertex list method. Using equation 2.12 they measured fragmentation of each
hole of empty cells after task placement which may have more than four
corners, instead for each MERs.

\[ F = 1 - \prod_{l} \left[ \left( \frac{4}{V_l} \right) \times \left( \frac{A_l}{A_f} \right) \right] \quad (2.12) \]

\( V_l \) is the number of vertices of hole, \( A_l \) is the hole area size and \( A_f \) is the total
size of the free area. In the expression \( \frac{4}{V_l} \) represents suitability of hole \( H_l \) to
accommodate rectangular tasks. Any hole with 4 vertices will have best
suitability. The relative hole normalised area is represented by \( \frac{A_l}{A_f} \). This
method penalizes holes with irregular shapes and small sizes.

Ejnioui & DeMara, (2005) suggested another metric for
fragmentation. Let the FPGA chip have \( N \times N \) cells. Here Fragmentation is
defined as Equation 2.13

\[ F = 1 - \left( \prod_{i=1}^{j} f_i \right) \quad (2.13) \]

\( f_i = \frac{k}{N^2} \), corresponds to a hole \( i \) consisting of \( k \) cells. Using this method the
smallest possible fragmentation will be an empty chip which consists of
single hole having \( N^2 \) cells. \( F=1-f_i =1- \frac{N^2}{N^2}=0 \). Highest possible fragmentation
resembles a checkerboard pattern. If $N$ is even, there will be $\frac{N^2}{2}$ holes where each hole occupies a single cell. Therefore, $F = 1 - \frac{1}{(N^2)}$. The value of $F$ approaches one as $N$ gets larger. In this algorithm also, two different fragmentation states may give the same fragmentation factor.

Septien et al (2006, 2008) proposed a Perimeter quadrature based metric. Assuming that an ideal hole should have a perfect square shape, they estimated how far its shape is near perfect square by dividing area $A$ by the area of a perfect square having same perimeter $p$ using Equation 2.14

$$A_q = \left(\frac{p}{4}\right)^2$$

(2.14)

$F = 1 - Q$, where $Q$ is relative quadrature defined by $Q = \frac{A}{A_q}$. The scenario with the smaller $F$ indicates less fragmentation. In case of multiple holes $P = \sum P_i$ and $A = \sum A_i$ is used.

Gu et al (2009) introduced a metric based on MER that takes into account the probability distribution of width and height of future arrivals instead of average values in Handa & Vemuri (2004b). Also, they calculated the time averaged area fragmentation and use look ahead technique to choose a location with minimum TAAF among all candidates at the current time and next few points when some tasks complete.

2.7.4 Task Relocation and Defragmentation

Diessel & Wigley (1999) tackled the fragmentation problem in partially reconfigurable FPGAs by performing task rearrangement techniques denoted as local repacking and ordered compaction. The local repacking
method attempts to repack the tasks within a part of the chip so as to accommodate the waiting task as well. A quad tree decomposition of the free space on the chip is used and a depth-first search of the tree allows promising parts to be identified and evaluated. This operation requires $O(n^3 \log n)$ where $n$ is the number of currently running tasks on the chip. Diessel & Wigley (1999) also presented an ordered compaction heuristic that moves some tasks to one side of the chip and places the waiting task at the freed location. Ordered compaction therefore has the effect of moving the running tasks that are to be compacted closer together while preserving their relative order. To select the moved tasks a direct dependency graph is built and depth-first traversal is applied to some candidate cells to check the minimum cost movement place. This operation requires $O(n^3)$ where ‘$n$’ is the number of currently running tasks on the chip.

Crompton et al (2002) discussed a hardware modification to the FPGA that provides task relocation and transforms to reduce fragmentation. Task transforms consist of a series of rotation and flip operations. Veen et al (2005) described an offline algorithm to maximize the available contiguous free space to optimality. But the algorithm can only be used at recurring idle times where the task configuration of the system is constant for a reasonably long time. Koester et al (2006) suggested a defragmentation method, which is suitable for heterogeneous FPGAs. If a requested task cannot be placed, the algorithm basically relocates hardware tasks, which are placed at a feasible position of the requested task. Relocating a hardware task at run-time requires a suitable mechanism to preserve the internal states of the hardware task during the relocation process. Therefore a context saving and restoring mechanism is depicted, which is suitable for Xilinx Virtex-II/-Pro FPGAs.

Gericota et al (2006) introduced a new method to access performance degradation introduced by relocation of functions during
defragmentation. The same authors have previously developed replication mechanism for CLBs to perform defragmentation without affecting the running functions. (Gericota et al 2002). ElFarag et al (2007b) studied different compaction strategies and found their effects on FPGA area utilization and performance. They proposed a compaction algorithm, Blind Compaction, which is one dimensional (compaction of tasks is done in one direction) and the order of tasks is preserved after compaction. They also introduced a two dimensional compaction algorithm, one-corner compaction, in which the tasks are compacted in two dimensions toward one corner of the chip. Finally, they extended the corner compaction algorithm to work with respect to the four corners of the chip, 4-corner compaction. In this algorithm, tasks that are closer to a certain chip corner are compacted toward that corner using the corner compaction algorithm. Their results show an improvement of about 16% in area utilization and 15% in allocation time over the blind compaction.

2.8 SUMMARY

In this chapter the basics of reconfigurable hardware were discussed. It was shown that a lot of domains can benefit from the reconfigurable hardware. However, the only limitations are the software and hardware maturity level to exploit these features. A general overview of online task placement and prior works done in that domain is presented. This is followed by a discussion fragmentation and the necessity of temperature management on a chip. The next chapter explains the proposed online task placement techniques.