Communication Scheduling and Buslet-based Design: New Paradigms for High Level Synthesis

by

ENZO TARTAGLIONE
Laurea, Politecnico di Torino, Turin, Italy, 2013

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2015

Chicago, Illinois

Defense Committee:

Shantanu Dutt, Chair and Advisor
John Lillis
Mariagrazia Graziano, Politecnico di Torino

To my parents

always near

despite the distance.

ACKNOWLEDGMENTS

I would like to thank my advisor at the University of Illinois at Chicago, Prof. Shantanu Dutt, with whom I collaborated to perform the research described in this thesis. The work in this thesis is a result of this collaboration when I was a student at UIC from August 2014 to July 2015, as well as a funded research assistant from January 2015 to July 2015. This collaboration also allowed me to intellectually grow so much.

I would like to thank all my friends from Turin belonging to the group "The paladins of Medusa", old and new members, for letting me feel as I were in Italy despite the huge distance between us. A special thank goes to Massimo S., thanks to whom I was able to carry out part of the paperwork necessary to graduate.

Lastly, I deeply thank my parents for the vigorous and continuous support they have always given to me. Since my first year at the university they helped me to overcome difficult moments and certainly I was able to do whatever I did thanks to their continuous support.

ET

PREFACE

This project, as the final stage of my master's studies, gave me both the experience and abilities that I was willing to acquire in the field of VLSI CAD algorithms and tools development.

The chance of working with Prof. Dutt, in particular, provided me first with knowledge on state-of-the-art High Level Synthesis algorithms and then gave me the possibility of developing skills in problem solving and algorithm design. In the last three months the work was mainly focused on the study and design of interconnect models for dynamic power minimization. Furthermore, such an opportunity helped me to develop skills in object oriented coding.

The complexity of the analyzed problems, finally, gave me a brand new viewpoint on problem solving methods and approaches and enlarged my vision beyond a purely engineering perspective.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

ALAP	As Late As Possible
ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIC	Application-Specific Integrated Circuit
BP	Binding Possibility
CB	Chronological Binding
CBSR	Chronological Binding with Similarity Reduction
DFG	Data Flow Graph
FDB	Force Directed Binding
FDCS	Force Directed Communication Scheduling
FDS	Force Directed Scheduling
FSM	Finite State Machine
FU	Functional Unit
HLS	High Level Synthesis
HP	Hierarchical Partitioning
LA	Lookahead

LIST OF ABBREVIATIONS (Continued)

ML-RCS	Minimum Latency-Resource Constrained Schedule
MR-LCS	Minimum Resource-Latency Constrained Schedule
MST	Minimum Spanning Tree
PF	Predecessor Force
RCA	Ripple Carry Adder
RTL	Register Transfer Level
SF	Self-Force
SSI	Small Scale Integration
UIC	University of Illinois at Chicago
VLSI	Very Large Scale Integrated
WL	Wirelength

SUMMARY

Current nanoscale designs are highly interconnect dominated, taking about 70% of the chip area. Interconnects also consume a significant part of the dynamic power and are responsible of about the 60% of signal delays. It is, thus, important to be able to synthesize much lower interconnect-complexity designs than are possible with current high-level synthesis (HLS) tools and algorithms. Towards that end, we have developed the following new paradigms in the scheduling, binding and general architecture synthesis problems of HLS:

- Flexibly-structured that connect a few neighborhood functional units (FUs) instead of dedicated interconnects between pairs of FUs, thereby sharing interconnects among a number of FU pairs that need to communicate.
- Communication scheduling (followed by standard operation scheduling that respects the communication schedules) in which communications between FUs are scheduled at appropriate times to minimize the number of buslets needed, subject to buslet cardinality constraints (for the purpose of upper bounding signal delay).
- Buslet binding techniques, aiming to respect both buslet cardinality constraint and a constraint on maximum fanin and fanout for the functional units. These techniques will range from simple but effective approaches like chronological binding (CB) to more sophisticated ones, like the use of lookahead approaches and simultaneous binding of iso-scheduled communications (communications scheduled in the same clock cycle). Furthermore, in this direction, similar solutions detection mechanism was developed, in order to improve the

final quality of the result. Finally, also a force directed approach was used to solve the binding problem (FDB). All these techniques were implemented and compared in terms of both performance and complexity.

- Buslet power modeling. A number of configurations with multiple tri-state buffers for interconnecting FUs through a buslet were implemented, aiming to minimize the total power consumed using buslets. These range from techniques using minimum spanning trees to more sophisticated structures with constraints on maximum graph distance between connected FUs to hierarchical partitioning.

Using the aforementioned techniques, we obtain significant wirelength (WL) reduction, ranging between 35% and 71%, compared to conventional designs with dedicated interconnects between communicating FU-pairs. The total chip area, including total FU area, also reduces in our designs compared to conventional designs. The power, on the other side of the coin, will increase with buslet size, but sublinearly. Empirical results show that we are able to limit the increment of power consumed by buslets compared to dedicated-interconnect designs, to a logarithmic function of the maximum buslet cardinality.

CHAPTER 1

INTRODUCTION

Our world is full of integrated circuits. It is possible to find them practically everywhere, like in mobiles, cars, televisions etc.; hence, they are now the heart of the modern technology. The integrated circuit (IC) is a cluster embedding a number of very advanced electric circuits. In particular, it will contain a very large number of components like transistors, resistors and capacitors, all interconnected. Among these, the transistor is the fundamental electrical component, at the basement of modern technology. It was invented in 1947 by John Bardeen, Walter Brattain and William Shockley[1], it was revolutionary: it embedded together tiny dimensions, high speed, high reliability and efficiency. Decades after decades, larger and more complex circuits, but at the same time compact and efficient, were this way realized.From the first ICs commercialized in the early 60s, containing just a few of components on the same chip (for example, TAA320 by Philips had two transistors only) and, for this, called Small-Scale Integration devices (SSI), transistors scaling increased thanks to more accurate fabrication technologies (Medium-Scale Integration contained hundreds of transistors on the same chip, Large-Scale Integration hundreds of thousands). Nowadays, the most advanced ICs embed millions of components on an area no larger than a fingernail. The transistors on these chips are realized using up 14 nm technology (like for Intel Core i7-5550U). In order to understand these

sizes, we can say that we could fit thousands of transistors inside a red blood cell[1]. This is the era of Very Large-Scale Integrated Circuits (VLSI).

A very famous prediction in this field was done in 1965 by Gordon Moore, head of the research and development office at Fairchild Semiconductor. He speculated [3] that:

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years."

G. Moore, 1965

Today we have reached physical limits of transistor scaling: we need to handle problems like leakage power consumption, limited range of gate metals, limited available materials for modeling the channel due to scaling degree and, most important, atomic dimension about to be reached[2]. Keep on scaling transistors (trend known as *More Moore*) is unsustainable.
However, nowadays a huge work of research for allowing transistor scaling is done. One of the

---

[1]A human blood cell (erythrocyte) has approximately a 50µm^2 area[2]. From this, we can see that we have three magnitude orders of difference between the size of an erythrocyte and a modern transistor.

[2]Nowadays the used technology is 14 nm (as stated before) while we know that a silicon atom has a Van Der Waals radius of 220 pm. This means that a transistor is realized with less than a hundred of silicon atoms. Furthermore, we can not have transistors too much close each other in order to avoid mutual electric field influence.

most promising among these relies on spin state electron usage, called *spintronics*[4], but there are also other possibilities like tunnel junctions[5][6][7] and new studies on nano-wire geometries[8][9]. The use of these new technologies in order to produce new ICs and, for this, making the transistor scaling trend keep on increasing is known as *More than Moore*.

Hence, in rapidly shrinking technologies, interconnect design has become a significant issue to be tackled. Indeed, as we are scaling more and more transistors, the density of interconnections is hugely increasing. With this, interconnections consume the most of the dynamic power in the IC[10]. Further, performance in terms of maximum clock frequency is bottlenecked by interconnections because of self and inter-wire parasitic capacitance effects, some of them due to tight narrowness of wires like crosstalk [11], wire performance fall for effects like IR-Drop [12] and its reliability for the electromigration effect [12]. These, mixed with the increase of resistance due to both long wires and use of VIAs, will increase the RC time constant, reducing the cutoff frequency. A number of approaches like buffer insertion[13], wire shaping[14] and communication encoding[15] were proposed in order to minimize VLSI interconnects. However, all of these techniques focus their attention on the minimization of the delay on designs with a big number of interconnections. What if we try to directly minimize the number of the required interconnections or their length? In that case, as effect, we will obtain the effect of minimizing parasitic parameters making also designs less complex, making the routers algorithms running more easily and obtaining better results (in terms of average wirelength for a given design).
The problem, here, is how to address such an issue. In order to minimize the "wire cost", we need to work on such a minimization already at HLS phase. During this phase, indeed, FUs are

instantiated as well as their mutual interconnections: the true minimization of interconnections starts here.

Traditionally, in HLS one of the following two problems need to be solved:

- Minimum latency-resource constrained scheduling (ML-RCS): here we have a fixed number of hardware units and we need schedule operations on them to minimize the total latency.
- Minimum resource-latency constrained scheduling (MR-LCS): in this case we have a fixed latency and we need to schedule operations to satisfy the latency constraint and minimize the total number of resources.

As we aim to minimize the interconnect complexity, which is a problem tightly linked to resources minimization, we will solve the MR-LCS problem. Furthermore, this correlates to current problems of interest in both ASICs and embedded designs, where the paradigm of resource/power optimization for a target performance is nowadays dominant.

Some works already tried to address such a relevant issue. The approaches used were mainly two:

- Wirelength minimization.
- Minimization of the number of interconnects.

All the papers aiming to minimize the wirelength are interconnection-aware high level synthesis algorithms which will have a floorplanning[1] step during the HLS. In particular, during this, both scheduling[2] and binding[3] steps are performed together and the whole design is driven by the wirelength estimated by the floorplanner. As we can imagine, using such an approach, we will effectively be able to minimize the wirelength. However, as we are introducing a floorplan step, we are also introducing a non negligible computation effort: for each possible decision at the scheduling/binding step, in order to evaluate it, we need to run the floorplanner. Hence, such an approach is feasible for small designs only.

One of the approaches used was integrating interconnect power optimization into high-level synthesis[16]. The main strength of such a paper consisted not only in the reduction datapath unit power but also in the reduction of the power consumed by interconnects. Practically, such a work reduces unnecessary switching activity in the interconnects. This is accomplished by an interconnect-aware binding, which tries to allocate as close as possible hardware resources with a high communication rate. Furthermore, a gating technique is proposed to reduce spurious switching activity. Such a work, however, has the big drawback of running, together with the

---

[1]A *floorplan* of an IC is a schematic representation of the placement of its FUs. Floorplans are created during the so called *floorplanning* design stage, which usually is the first step in the physical design phase.

[2]We define *scheduling* as the step in the HLS in which we are deciding when a given operation should happen. It usually is the first step of HLS.

[3]We define *binding* as the step in HLS in which we map each operation to a given FU. It usually is performed after scheduling step.

binding algorithm, also the floorplanning stage. This will lead to a big overhead in the computation, also using an approximate power model.

Another proposed work uses physical information to drive the solution at the HLS stage of the VLSI CAD flow[17]. In particular, here it was explicitly designed an incremental floorplanning HLS algorithm, which improves at the same time designs schedule, binding and floorplan. Thanks to the incremental approach, the overhead introduced by the floorplan stage is reduced; however, performing both scheduling, binding and floorplan stages at the same time will make runtime hugely increasing.

Other works reducing interconnections in VLSI have, on the contrary, their focus on trying to minimize the amount of wires itself instead of their own length. They are able to do it detecting "common patterns" across the input[18]: if a sequence of same-type operations is observed in more than one point of the input, then they can be mapped to the same physical units, using exactly the same interconnections, which will result in the reduction of interconnect complexity. This approach is completely unaware of the wirelength because it is not the parameter to be optimized. For this, its execution is relatively fast and it obtains improvements in wiring complexity respect to a wire minimization unaware algorithm. However, such a model is not very accurate for two reasons:

- It is not estimating the wirelength at all: for this reason, there may be cases in which we prefer to have more short connections than few very long ones[1].
- It is not trivial to dynamically decide the granularity of the pattern detection: if this is fixed, then it may lead to non negligible suboptimal results; if it is dynamically determined, instead, it will result in a high computation effort.

Another approach used in such a direction was trying to optimize communication sharing in pipelined architectures[19]: in this way, the total number of interconnections reduces and their use increases. However, such a technique has some significant weaknesses:

- It works after HLS and, for this, it is limited to an already defined hardware architecture.
- It relies on pipeline architectures: this means that such an optimization is limited to the number of pipeline stages through a line: the more they are, the more the number of registers will be (higher power consumption) and the least they are, the least such an optimization is effective.

1.1 Flexible buslets: a new interconnection paradigm

Minimizing the number of interconnects is an interesting because it aims to maximize the use of the same interconnections. Let us define for a given wire w_i a parameter, called *wire efficiency*, as

$$\varepsilon(w_i) = \frac{T(w_i)}{\overline{\lambda}} \tag{1.1}$$

[1]Connecting distant FUs is more difficult than connecting close ones; for this, designs with few very distant connections may result in more complex interconnections than those with more but shorter.

where $T(w_i)$ is the number of clock cycles w_i is busy and $\overline{\lambda}$ is the total latency of the output design[1]. A good approach in order to minimize interconnections is the maximization of such a parameter. In this way, we are trying to reuse, where possible, the already existing interconnections.

However, in a typical design, such a parameter is usually very low[2] making a given wire idle for most of the time.

In a typical design, wire efficiency is usually very low (according to experimental data we present in in Table IV and Fig. 48, maximum wire efficiency for dedicated-interconnection based designs is below 0.1 or 10%), making a given wire idle most of the time. A natural solution to alleviate these issues is merging close-by wires which are busy in different clock cycles in a single mini-bus like structure which we call a *buslet*, and whose wirelength (WL) is smaller than the sum of the WL's of the merged wires. Such a buslet is a flexibly-structured mini-bus in that it connects a small subset of FUs placed potentially anywhere on the chip. However, we need to take into account the no-conflict constraint on a given buslet: we need to ensure that at most one FU per clock cycle will drive the buslet (which it will do via tri-state buffers that are enabled at the right clock cycle(s) by the datapath FSM controller, which all HLS design have). In order to ensure this, a proper scheduling algorithm (minimizing the number

---

[1]We define *latency* of a given design as the number of clock cycles needed for a complete execution.

[2]From our collected empiric data, even trying to maximize such a parameter, we obtained peak values below 0.1 (Table IV)

of buslets requested for such a constraint) as well as a binding one, need to be designed, as well as a study of the necessary constraints in order to ensure a good solution quality.

1.1.1 Previous work using bus-like structures

Finally, we note that the problem of minimizing design wirelength using "bus-like" structures was earlier proposed in [20]. This problem was solved using partitioned buses, each of which:

1. Spans across all the FUs.
2. Connects to the FUs in a linear order (determined based on the DFG in order to maximize the number of adjacent FU communication).

The number of such buses depends on the maximum number of simultaneous "overlapping" communication needed across the aforementioned linear order, and this depends on the operation scheduling performed to solve the ML-RCS. Further, it is also not clear that they can solve the more standard MR-LCS problem that we are solving, due to their technique not having any control on worst-case communication delay because of their long-spanning buses. While this was an innovative technique, one of the drawbacks is that the number of buses is not minimized in initial scheduling, and this number can be prohibitive for large designs—they only show that their technique works well in reducing total wirelength and area for two small DFGs compared to previous dedicated-interconnects based designs. Further, the worst-case length across which a communication can take place is linear in the number of FUs, leading to a slow design for larger DFGs. Also, since there is no constraint on the number of buses, the input and output capacitive loading of FUs can be large, leading to a further slow-down of their designs, as well

as to high dynamic power. Finally, apparently in order to reduce the interconnect complexity imposed by their linear order (each bus spans across all FUs), they use ALUs instead of FUs of different functionality, thus reducing the number of total resources. However, this can lead to higher leakage power (only one FU within an ALU can be active at any time, meaning that FU usage will be low, and thus they would generally need more total FUs—in spite of the constraint on number of ALUs—than a design of comparable speed that uses individual FUs). In contrast, we have flexibly structured buslets that each connect only a few FUs (thus not having a slow down or increased dynamic power due to unnecessary FU loading from buses). Further, we perform communication scheduling first in order to minimize the number of buslets needed, and thereby significantly reduce interconnect complexity for DFGs of any size.

1.2 Thesis outline

The rest of the thesis is organized as follows. In the next chapter an analysis of the constraints needed for designs without high interconnect delay and power consumption will be presented. In Chapter 3, I present an overview of the entire VLSI flow with focus on the general area of this thesis. Furthermore, some basic issues in scheduling are presented, including the well-known force-directed scheduling algorithm, which is the starting point for our scheduling algorithms. Chapter 3 discusses also our scheduling algorithms that aim to minimize, first, the number of buslets, and then the number of FUs. Chapter 4 discusses several new binding algorithms and Chapter 5 is entirely devoted to the design of different buslet interconnection structures in order to minimize dynamic power. The thesis ends with presentation and discussion of results in Chapter 6, and conclusions in Chapter 7.

CHAPTER 2

MAIN ISSUES IN BUSLET WIRED DESIGNS

The use of buslet in place of standard dedicated interconnections will bring evident benefits to the whole design in terms of overall wire structure to be routed as well as maximization of the already existing resources itself. However, a number of factors are involved in such a process: from the increment in dynamic power consumed during the communication for fanins to the increment of area due to multiplexers or demultiplexers needed. In this chapter all these aspects will be analyzed, defining the *main driving factors* of the HLS algorithm.

2.1 Maximization of wire efficiency

A design involving buslets will bring evident benefits on the efficiency of the design itself: as previously stated, it is literally possible to *merge* separate wires which are active in different clock cycles, obtaining benefits in the overall wirelength. Let us take, as example, Fig. 1. From this, we can see that we have to map 4 different communications between 4 different Functional Units (FUs). Unfortunately, in order to map all of these communications, using dedicated interconnections we will need to use four different interconnections, even if all of these will happen in different clock cycles. Now, let us use a buslet instead: as all of these communications will happen in different clock cycles, we will be able to map all of these in a wire structure simultaneously linking all of these FUs. This is possible because, when a

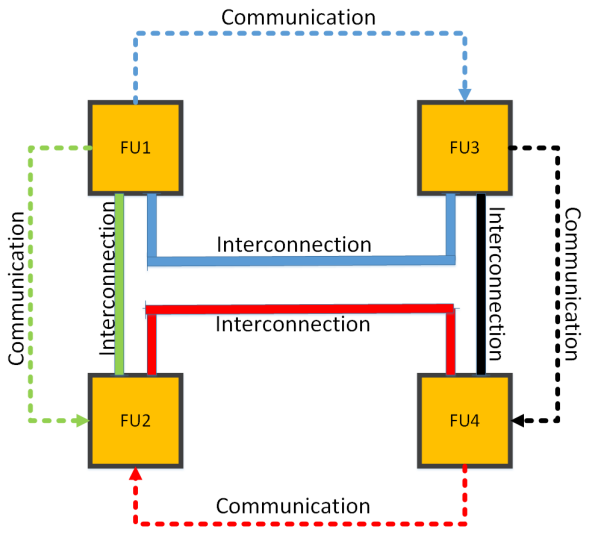

Figure 1: Example of dedicated interconnections design.
Dashed arrows indicate communications while continuous lines are implemented interconnection. Each color indicates a different clock cycle in which the communication happens.

communication between source functional unit FU_s and destination functional unit FU_d takes place, then all the others wired to the same buslet will ignore the signal, exactly how it does work in a regular bus, thanks to the use of tristate buffers. The same implementation of the problem presented in Fig. 1 with the use of buslets is sketched in Fig. 2.

Let us, now, analyze the difference in wirelength between the two designs. In the dedicated interconnections we will have the following wirelengths:

- blue wire: $d(FU_1, FU_3) + 0.25d(FU_2, FU_3)$
- red wire: $d(FU_1, FU_3) + 0.25d(FU_2, FU_3)$
- green wire: $d(FU_2, FU_3)$

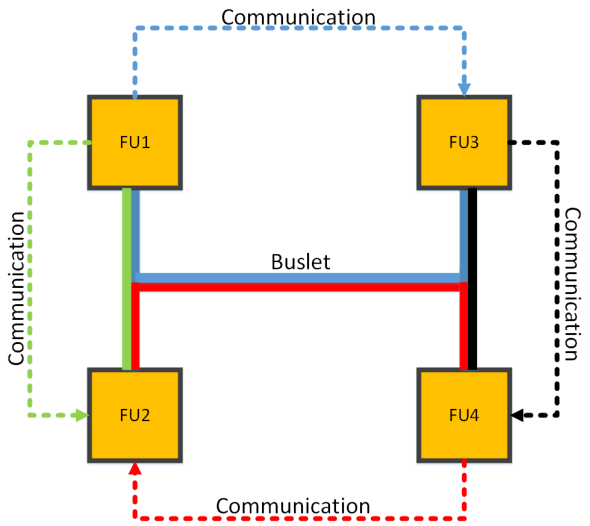

Figure 2: Example of buslet design. Dashed arrows indicate communications while the continuous line is the buslet. Each color indicates a different clock cycle in which the communication happens.

- black wire: $d(FU_2, FU_3)$

where $d(x, y)$ indicates the distance between x and y. So, we will have that the total wirelength for dedicated interconnections design in Fig. 1 is obtaining a relevant reduction in wirelength.

$$d_{ded\ interc} = 2d(FU_1, FU_3) + 2.5d(FU_2, FU_3)$$

For the buslet wiring, instead, it is evident it is

$$d_{buslet} = d(FU_1, FU_3) + 2d(FU_2, FU_3)$$

Let us, now, analyze the difference in wire efficiency between the two implementations. According to the definition of wire efficiency in (1.1), in the dedicated interconnections design we have $\frac{1}{4}$ the efficiency we will have in the buslet design. This evidences that having designs with buslets will have also a significant improvement in the overall wire efficiency. We can generalize this telling that the wire efficiency of a buslet is the sum of the dedicated interconnections merged in order to obtain it.

Let us say that $C(w_m)$ is the set of communications mapped in the dedicated interconnection w_m. We say that w_m respects the condition of *non conflicting communications* if, $\forall com_i \in C(w_m)$, $t(com_i) \neq t(com_j)$, with $com_i \neq com_j$, where $t(com_i)$ is the scheduling time for com_i. However, such a condition works assuming communication latency[1] λ_{com} unitary. If we have a parametric value of λ_{com}, we need to ensure the following conditions:

$$\begin{cases} t(com_i) + \lambda_{com} - 1 < t(com_j) & if\ t(com_i) \geq t(com_j) \\ t(com_j) + \lambda_{com} - 1 < t(com_i) & if\ t(com_i) \geq t(com_j) \end{cases} \tag{2.1}$$

which can be also written as

$$|t(com_i) - t(com_j)| \geq \lambda_{com} \tag{2.2}$$

which means that the scheduling times for the two communications must differ at least by the latency of the communication itself.

---

[1]We define *communication latency* the time interval necessary to complete the communication from its begin.

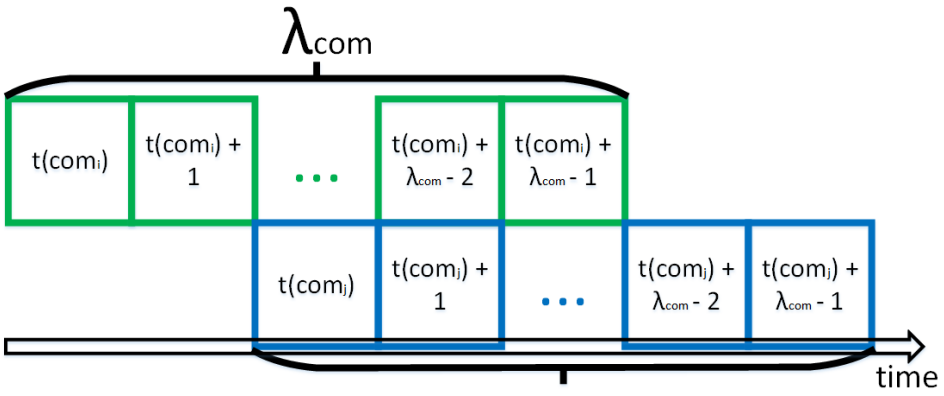

Figure 3: Example of overlapping communications.

Let us take, as example, Fig. 3. In this case, at a certain point, $t(com_j)$ has been scheduled while $t(com_i)$ has not finished yet. This obviously means that these communications will bever been scheduled to the same buslet. Now, let us analyze the case represented in Fig. 4. In this case, which is a limit case, the two scheduling times will exactly differ of λ_{com}. However, there will not be any conflict between them. In this way, they may be "mapped" to the same buslet. Now, if we decide to merge a set M of wires, we need to ensure the non conflicting communications condition: $\forall com_i \in \forall w_n \in M$ (2.2) must be true.

Let us say that M is a set of dedicated interconnections respecting the condition of non overlapping communications. We can here see that, merging buslets, we will always increase the wire efficiency (1.1)

$$\varepsilon_{buslet} = \sum_{\forall\ w_n \in M} \varepsilon_i \tag{2.3}$$

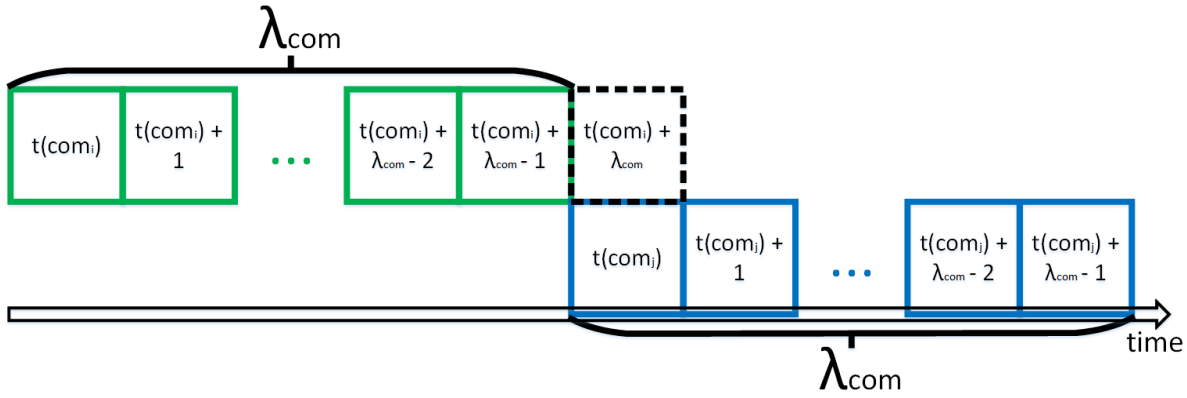

Figure 4: Example of non overlapping communications.

However, now comes the problem of how to maximize buslet efficiency. Let us assume we need to merge, in a certain way, some interconnections. Here we are using, for sake of simplicity, $\lambda_{com} = 1$. We have a design with 6 functional units (respectively A, B, C, D, E and F) and with the following communication scheduling times:

$$\begin{cases} t[com(A,B)] = t[com(A,C)] = t[com(E,F)] \\ t[com(C,D)] = t[com(C,B)] = t[com(D,E)] \end{cases}$$

Dedicated interconnections design is pictured in Fig. 5. In order to build buslets merging these wires, we have 6 possibilities.

For example, we can have an example of merging in Fig. 6. Here, in particular, wires w_1 and w_5 were merged creating buslet b_1, w_2 and w_4 generated b_2 and w_3 and w_6 were merged into

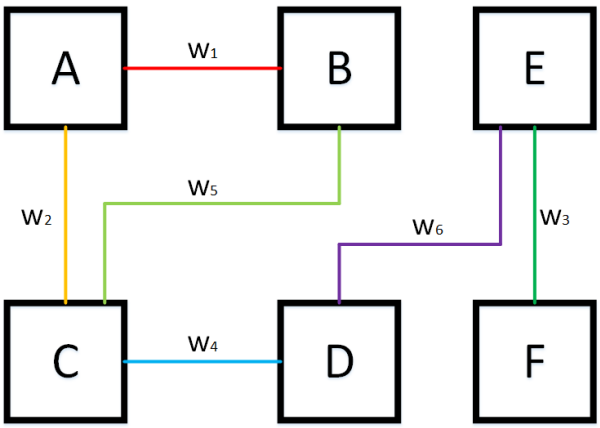

Figure 5: Example of dedicated interconnects implementation.

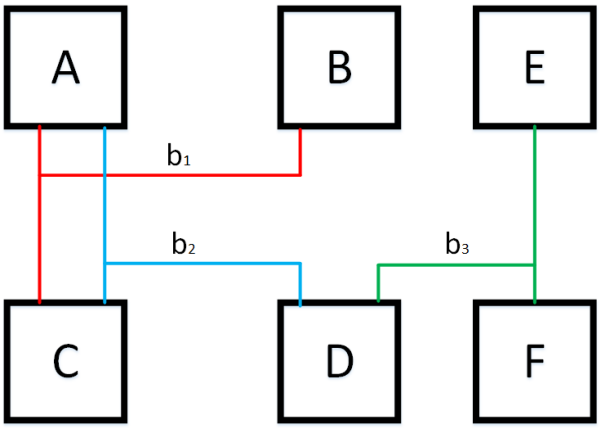

Figure 6: Example of buslet implementation.

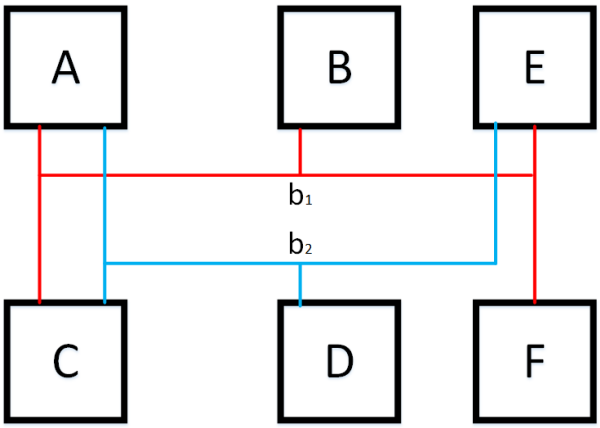

Figure 7: Second example of buslet implementation.

b_3. This may appear to be an optimal solution for the input design. However, in order to get such a solution, all the possible solutions were inspected, and this will take exponential time; so, for big designs, this is not a very good solution. For this, an heuristic to solve such a problem appears to be necessary. Furthermore, we can see that we are constrained by communications scheduling times: in fact, it is evident that the lowest possible number of buslets is the maximum number of overlapping communications scheduling times. Hence, if we could change the previous constraints on this like

$$\begin{cases} t[com(A,B)] = t[com(A,C)] \\ t[com(C,D)] = t[com(C,B)] \\ t[com(E,F)] = t[com(D,E)] \end{cases}$$

we would be able to reduce the total number of needed buslets. An example is provided in

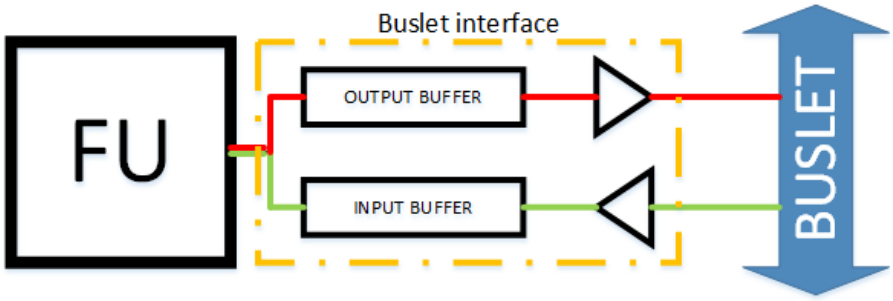

Figure 8: Functional unit interface with the buslet. It will have an output and an input tristate buffer, which will read/write from a proper buffer.

Fig. 7 in which w_1, w_5 and w_6 were merged in b_1 and all the remaining created b_2. Hence, in order to have good quality solutions, we do not need to work on binding only, but scheduling times for communication play a fundamental role in the final solution quality.

2.2 Dynamic power consumption and signal delay: constraint on maximum cardinality of a buslet

Reduction of wirelength and increase of wire efficiency are two wonderful effects brought by buslets. However, we have also to take into account two relevant side effects of simultaneously connecting several functional units: each of these will have an input capacitance which will increase the overall dynamic power necessary to perform a single communication. Furthermore, the wirelength of the single interconnection (not of the overall design) will increase, which will also bring to an increment of the signal delay. This implies that we have to use a higher dynamic power compared to the dedicated interconnections case having, at the same time, a lower maximum working frequency. We need to carefully handle such a relevant issues. Let

us suppose that buslets are bidirectional communication structures (so, data can travel from any connected functional unit FU_a to any FU_b). In such a structure, any connected resource/FU needs two different "accesses" to the buslet[1]:

- one as its output, which needs to be a tristate buffer.
- one as its input.

Let us ignore the capacitive parasitics at the tristate buffer outputs (as they generally are much smaller than input capacitances) as well as capacitance for the interconnection itself (for the moment). We can see that the main capacitive load is given by the input capacitances. Let us define *buslet cardinality* $|b_j|$ of buslet b_j as the number of FUs connected to it. We know that, in the general case, some FUs can be connected to a buslet b_j as a fanout, without having it as a fanin. For this reason, defining $|fin(b_j)|$ as the number of FUs having b_j as a fanin, we know that $|fin(b_j)| \leq |b_j|$. Defining C_i as the input capacitance of FU_i (or that if a fanin multiplexer of b_j), the total buslet capacitance will be

$$C_{b_j} = \sum_{i=1}^{|fin(b_j)|} C_i \tag{2.4}$$

---

[1]Actually, this is not always true. In particular, let us assume FU_i, $FU_j \in b_n$ where b_n is a buslet. If b_n performs communication from FU_i to FU_j only, then FU_i will not need input tristate buffer and FU_j will not need output tristate buffer. Such a consideration will be taken into account in the estimation of the power consumed for communications.

If we say, for sake of simplicity, that every FU will have the same input capacitance C_{in}, we can approximate (2.4) to

$$C_{b_j} \approx \sum_{i=1}^{|fin(b_j)|} C_{in} = |fin(b_j)| \cdot C_{in} \tag{2.5}$$

According to (2.5), if we have a large number of FUs having a buslet as fanin and, according to this, allowing high buslet cardinality, we may have a high input load capacitance to a driving FU. Hence, one important parameter to be taken into account is an upper bound on buslet cardinality.

Buslet cardinality and length also affect signal delay, and thus limiting maximum buslet cardinality reduces the RC delay by both reducing wire resistance and capacitance, as well as capacitive load of FU inputs. However, signal delay is generally tackled as on upper bound constraint (e.g., signal delay between any two FUs should be at most 1 clock period) and not a minimization metric, and thus the ramification of buslets on signal delay is a little lighter than on power.

2.2.1 Maximum fanin and fanout per FU

All the functional units , in order to be connected to a buslet, need an input and an output access to it. However, in the general case, they will be connected to a number of buslets greater than one: we need extra structures in order to connect the input or the output access of the FU to the buslets. These structures are multiplexers for inputs and demultiplexers for the outputs. According to this, from the point of view of a single FU, it becomes important to limit the

number of buslets it is connected to, in order to minimize the signal delay in demultiplexers or multiplexers. Since a FU may have a buslet as only fanin or fanout, these two paramers will typically not be the same, and thus we need an upper-bound constraint on each. In the following, we motivate the need for such constraints:

- Signal delay: as already stated, if a FU has a high fanout, the amount of delay in the data transmission to the buslet is proportional to the number of 1:2 demultiplexers that a larger 1:m demultiplexer is constructed from the signal needs to cross. Assuming a tree structure, if the amount of time to cross a single 1:2 demultiplexer is t_{demux} and the fanout is m, the estimated delay is

$$\Delta t_{fanout} = \lceil \log_2(m) \rceil \cdot t_{demux} \tag{2.6}$$

  The same issue applies to fanin using multiplexers. Thus, fixing an upper bound on the fanin and on the fanout, we will ensure data transmission from FU to target buslet (and vice versa) will not be too slow.

- Area increment: multiplexers, as well as demultiplexers, have a cost, here expressed in terms of increment in area size. Assuming fanout m, tree structure for the 1:m demultiplexer and area for a single 1:2 demultiplexer A_{demux}, the expected increment of area is

$$\Delta A_{fanout} = (m-1) \cdot A_{demux} \tag{2.7}$$

which, unlike for the time delay increment in (2.6), is a linear increment and, for this, for high m, this has a non negligible effect on the whole design.

We know, indeed, that a 2:1 multiplexer needs 4 transistors to be implemented, as it uses transmission gates, while a full-adder requires 14 transistors[21]. For this reason, as increasing the input data size of h bits (we have 2 inputs of size h) we will increase hardware area cost of a factor h itself for both an h bit size 2:1 multiplexer and for a Ripple Carry Adder (RCA) (as for two h-bits inputs we need h 2:1 multiplexers and for adding two h-bits addenda we need h Full Adders (FAs)), we can approximately say that for a given h data size of any input, we have that the area of a multiplexer is about $\frac{1}{3}$ RCA's. The same issue applies for fanin and multiplexers.

CHAPTER 3

SCHEDULING ALGORITHMS

3.1 Introduction to the VLSI CAD flow

The electronics industry was able to achieve an outstanding growth over the last two decades, thanks to the huge improvements obtained in the field of large-scale systems design. Essentially, such a market took place thanks to the the advent of VLSI. For this, some well organized structure for designing VLSI circuits needed to be developed.

The design process, at each one of its levels, meets the same intermediate steps. Indeed, it starts from a given set of requirements; then, an initial design structure is developed according to some criteria and tested against the requirements. Whenever the requirements are not met, the design needs to be somehow improved. We can divide the nowadays VLSI CAD flow in the following steps:

1. *System specification*: In this stage global goals and constraints of the system are defined. In particular, it will be evidenced which one will be the functionality of the system (like input and output signals description) as well as boundaries on minimum speed or maximum power consumption, constraints on size and space occupied by the circuit or even more complex constraints like maximum temperature.

2. *High level synthesis*: Here we take as input a sequence of operations which will provide the desired output and it will give, as output, a general FU structure with all needed data

transfer between hardware blocks and time in which these FU are allowed to start their computation. Functional Units are here seen as black boxes which will perform a given computation.

3. *Logic design*: Choice of the proper logic gates structure for each functional unit needed in the high level synthesis stage. This phase can be seen also as RTL description of the FUs.

4. *Ciruit design*: All logic gates are here converted in transistors, according to the used technology. In particular, all the transistors are here sized in order to meet power and delay requirements (the transistor model has plenty of capacitances which are heavily taken into account in this synthesis stage).

5. *Physical design*: Here the circuit is converted into a *layout*, which is the physical implementation of the circuit. This will represent the actual fabrication result of the circuit. It is divided in a number of intermediate stages; however the two most significant are *floorplanning* (which is the placement on the board of the FUs) and *routing* (which is the creation of the physical wires interconnecting, where needed, the functional units). Both these synthesis stages can be iterated until physical constraints like delay and power consumption are not met.

6. *Physical verification and signoff*: In this stage all the constraints are checked. If some of these are not met, then synthesis stage(s) are iterated, depending on how the synthesis algorithms are designed.

7. *Fabrication*: Here layout is converted into masks for lithography, which will be used for the fabrication in the *tapeout* process.

8. *Packaging and testing*: Now, the circuit is physically tested and a verification of the constraints is performed.

We will mainly focus on the high level synthesis stage as it is our work domain.

3.2 High level synthesis

High-level synthesis (HLS) became a relevant issue in the VLSI design since the late 1970s. Because of the huge scaling trend happened in the last decades, a lot of the research was performed in order to bring some improvement at the design level. More, soon researchers realized that some decisions taken at in the earliest stages of the CAD flow will determine the quality of the final output. For this reason, a lot of research was done on the so called High Level Synthesis (HLS).

We can divide the high level synthesis in some steps, which can be also each other merged:

1. *Scheduling*: operation nodes, of the input DFG, will receive a scheduling time, according to some implemented heuristics, which usually aims to minimize total quantity of needed functional units for the given design or the total request time to accomplish the task the whole DFG aims to reach

2. *Binding*: here, operation nodes will be "mapped" to functional units (FUs) according to some heuristics. This means that we will decide which physical piece of hardware will execute the operation described by each operation node.

As these are the earliest steps in the whole physical design process, any decision taken here may affect quality of gained design in successive steps: for example, if we do not take into account chip area minimization problem, we may have huge designs with very distant FUs, indirectly affecting maximum distance between communicating functional units and, for this, making working frequency drop. For this reason, it is very important to properly take into account a number of metrics also in the early stages of physical design.

Through the history of high level synthesis, a number of algorithms were developed. Here follows the most significant, fundamental ones, in order to wholly understand the thesiswork.

3.2.1 As-soon-as-possible scheduling algorithm

This is the simplest scheduling algorithm. Let us suppose we wish to minimize the final design total latency regardless the number of hardware units we will need to use. In order to do this, what we can do is simply schedule any operation node in the first available clock cycle when all its predecessors in the dataflow graph ended their own computation. This gives this algorithm the name *as-soon-as-possible* (ASAP).

Let us define $\lambda(u_i)$ the latency of the operation node u_i. If $P(u_i)$ is the set of all predecessors of operation node u_i, the *asap schedule* of u_i is defined as

$$asap(u) = \max_{\forall v \in P(u)} \{asap(v) + \lambda(v)\} \tag{3.1}$$

Of course, the first nodes in the DFG are immediately scheduled as they do not have dependency with any other node.

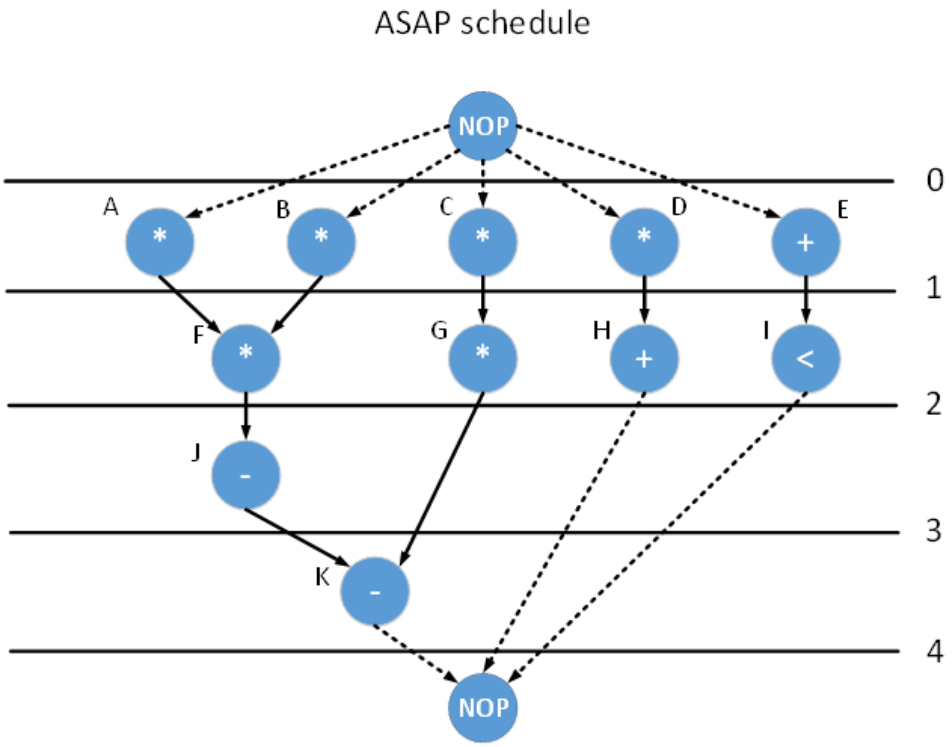

Figure 9: This is an example of as-soon-as-possible schedule.

An interesting property of this algorithm is that it provides the lowest possible execution time for the provided input. This is called *minimum latency for the input.*

Let us take, as example of as-soon-as-possible schedule, Fig. 9. Here we are assuming execution time for each operation node to be 1 clock cycle and communication latency to be negligible. Here we can see that, regardless the number of nodes working at the same clock cycles, we are scheduling all of them as soon as input data is ready. For example, we can schedule node F just when data will arrive from its predecessor nodes A and B. At the beginning, there are some nodes which have no predecessors (in this case, A, B, C, D and E) and are immediately

executed. Following this approach, we see that the clock cycle when the last operation nodes ends its execution (in this case K at clock cycle 5) corresponds also to the minimum possible latency for correctly executing the whole dataflow graph. We see that such an algorithm has a pull-up effect on the scheduling times (all of these are placed at the earliest possible clock cycle). This algorithm will analyze each node ($O(n)$) inspecting all its predecessors ($O(n)$). In the worst case, this runs in $O(n^2)$.

Algorithm 1 As-soon-as-possible scheduling

procedure ASAP(DFG)
  Topological sort (DFG) ▷ Here, for every directed edge $e(u_i, u_j)$ from u_i to u_j, u_i comes before u_j in the ordering
  for all $u_i \in DFG$ **do**
    $first\_sched = 0$ ▷ This is asap time, initialized
    for all $u_j \in P(u_i)$ **do**
      if $t(u_j) + \lambda(u_j) > first\_sched$ **then** ▷ We need to respect data dependency, as stated in (3.1)
        $first\_sched = t(u_j) + \lambda(u_j)$
    $t(u_i) = first\_sched$

3.2.2 As-late-as-possible scheduling algorithm

As-late-as-possible (ALAP) scheduling algorithm can be seen as the complementary than ASAP. Indeed, it will schedule the operation nodes in the latest available opportunity. However, such an algorithm will not provide the input latency as output, but it will ask for it as input.

Hence, the provided latency $\overline{\lambda}$ needs to be greater than the minimum latency for the input, provided by ASAP scheduling:

$$\overline{\lambda} \geq \lambda_{ASAP} \tag{3.2}$$

Contrarily than the ASAP schedule, the ALAP schedule for a given node u can happen when all its successors are scheduled: if we define $V(u_i)$ the set of all the successor nodes of u_i, the alap schedule for u_i is

$$alap(u) = \min_{\forall u_j \in V(u_i)} \{alap(u_j) - \lambda(u_i)\} \tag{3.3}$$

Here the latest nodes (those having no successors) will be scheduled at time $\overline{\lambda}$ minus their own latency. Let us also here analyze an example of as-late-as-possible schedule (proposed in Fig. 10). We underline that it is the same DFG as in Fig. 9. Here, unlike for asap, we start binding all the nodes which have no successors or, in a different way, all nodes not needing to send computed data (in this case, K, H and I) at the least possible clock cycle (which has to be provided as input) minus their own latency (in this case we are assuming all the operation nodes have the same unitary latency; so, we are scheduling these at $\overline{\lambda} - 1$). Then, we will schedule all the nodes which have to transmit data to these (K needs to receive data from both J and G, H from D and E from I) and so on. We see that such an algorithm has a pull-down effect on the scheduling times (all of these are placed at the least possible clock cycle).

The complexity follows the same analysis as done for asap schedule: it is $O(n^2)$. Such an algorithm is not that much useful by itself; however, it contributes in the definition of a very important parameter in our algorithms.

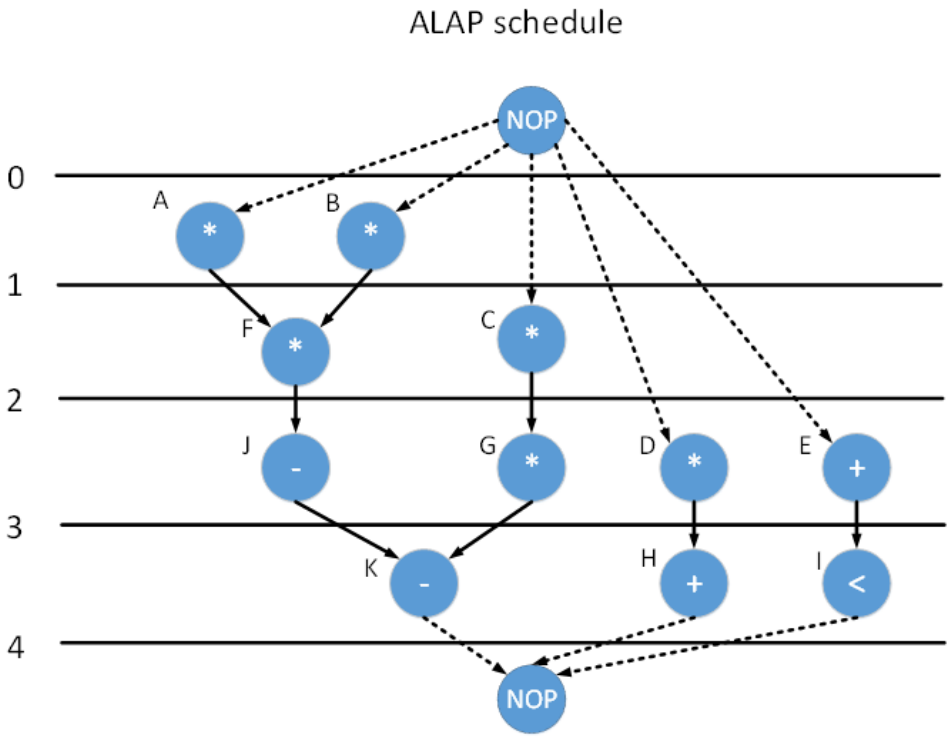

Figure 10: This is an example of as-late-as-possible schedule. It is the same DFG as in 9

Algorithm 2 As-late-as-possible scheduling

procedure ALAP(DFG, $\overline{\lambda}$)
Reverse topological sort (DFG)▷ Here, for every directed edge $e(u_i, u_j)$ from u_i to u_j, u_j comes before u_i in the ordering
for all $u_i \in DFG$ **do**
$last\_sched = \infty$
for all $u_j \in V(u_i)$ **do**
if $t(u_j) - \lambda(u_i) < first\_sched$ **then** ▷ Here we need to respect reverse data dependency (3.3)
$last\_sched = t(u_j) - \lambda(u_i)$
$t(u_i) = last\_sched$

3.2.3 Mobility range for an operation node

According to the definition of as-soon-as-possible (3.1) and as-late-as-possible (3.3) schedule, given a maximum execution time for out input $\overline{\lambda}$, we can determine a set of acceptable clock cycles in which a given node u can be scheduled. In particular, we can say that, if node u will be scheduled in any clock cycle between $asap(u)$ and $alap(u)$, it will certainly exist a solution such that the total latency will be lower or equal than $\overline{\lambda}$. The clock cycles between $asap(u)$ and $alap(u)$ are called *mobility range* of operation node u.

Hence, in order to respect latency constraints, we say that scheduling time $t(u)$ must be in its mobility range

$$t(u) \in [asap(u); alap(u)] \tag{3.4}$$

Further, from the mobility range of a node u we can know how much decisional power on its scheduling time we have. Such an estimation is obtained just counting all the valid scheduling times for node u. Now, as we know by (3.2) that $asap(u) \leq alap(u)$, we can determine a parameter, called *mobility* of node u, defined as

$$\mu(u) = alap(u) - asap(u) + 1 \tag{3.5}$$

which will always be a positive number. Let us take, as example, Fig. 11. This, in particular, encloses both the asap schedule in Fig. 9 and the alap in Fig. 9. Let us focus on the node G. Here we see that its asap schedule is at clock cycle 2 while its alap is at clock cycle 3. According to what stated for asap and alap schedules and to the maximum latency constraint here fixed

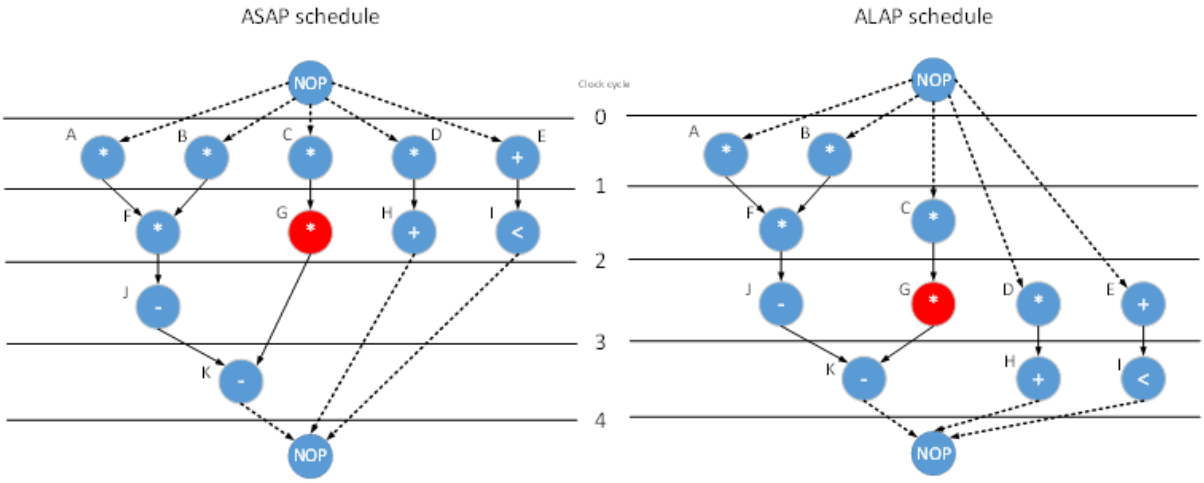

Figure 11: Example of mobility range for a node in a dataflow graph

to $\overline{\lambda} = 5$, we will never beein able to schedule G earlier than clock cycle 2 or later than clock cycle 3. Hence, we have just two possible schedulings for this and, for this, a mobility of 2.

3.3 Force directed scheduling algorithm

One well-known scheduling algorithm, known for its efficacy and efficiency, is forced directed scheduling, by Pierre G. Paulin and John P. Knight [22]. A wide number of MR-LCS algorithms used in High Level Synthesis comes from force directed scheduling (FDS). The main objective of FDS is minimizing the least number of FUs needed for the binding step. Such an objective is achieved uniformly distributing the operation nodes in all the available functional units. This will guarantee that functional units necessary in order to perform some operation in a given time range are efficiently used in any other time moment of the overall design latency, leading to a very high use of the FU and, according to the efficiency definition we provided, to a high

efficiency. Fundamental for FDS algorithm are asap and alap scheduling algorithms, because the mobility range for each operation node is an essential parameter.

We can divide this algorithm essentially in four different steps:

1. Computation of probabilities for operation nodes.
2. Creation of "distribution graphs".
3. Computation of forces and self forces
4. Inclusion of the effect on predecessors and successors

3.3.1 Probability estimation for an operation node

FDS algorithm assumes each operation node will have a uniform scheduling probability for any of the clock cycles in the mobility range and null in any other clock cycle. For this, we can say that the probability $p(u,k)$ of scheduling the operation node u in clock cycle k is

$$p(u,k) = \begin{cases} \frac{1}{\mu(u)} & if\ asap(u) \leq k \leq alap(u) \\ 0 & otherwise \end{cases} \tag{3.6}$$

Let us take, as example, the probability of scheduling node G in the DFG in Fig. 11. As shown in Fig. 12 and as already stated, the only two available clock cycles G can be scheduled are 2 and 3. According to the mobility definition, its μ is 2. Hence, as stated in (3.6), we have a uniform probability distribution for all clock cycles in range [2;3] which is $\frac{1}{2}$ for each of these clock cycles, while it is zero for all the others clock cycles (in this case, 1 and 4).

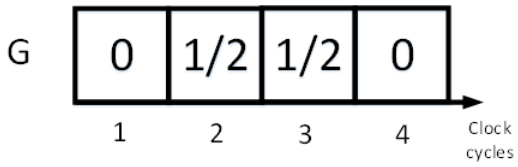

Figure 12: Probability for node G to be scheduled in any clock cycle of the DFG in Fig. 11.

3.3.2 Distribution graph

After computing all the probabilities for any clock cycle j in range $[0; \overline{\lambda}]$, we can see each clock cycles how many functional units of a certain type (like adders, multipliers...) are needed. If we say that J is the set of all the operations nodes u_i requiring the same type $g(u_i)$, we can say that, at the generic clock cycle j, the distribution graph for functional units of type $g(u_i)$ is

$$W(g(u_i), k) = \sum_{u_i \in J} p(u_i, k) \tag{3.7}$$

From this, we have an estimation of the peak of needed functional units: indeed, the minimum number of FUs for a design is the maximum number of functional units active in the same clock cycle. $W(k(u_i), k)$ indicates the distribution of the sum of scheduling a node of a certain type $g(u_i)$ in a given clock cycle k. Hence, if we minimize such a quantity, then we will try to schedule operation nodes in points in which the distribution graph is the lowest.

Let us try, for example, to build the distribution graph for multipliers in the DFG in Fig. 11. We have to take all the probabilities for all the latency of our DFG for any node performing multiplication and adding them through every single clock cycle, according to the formulation

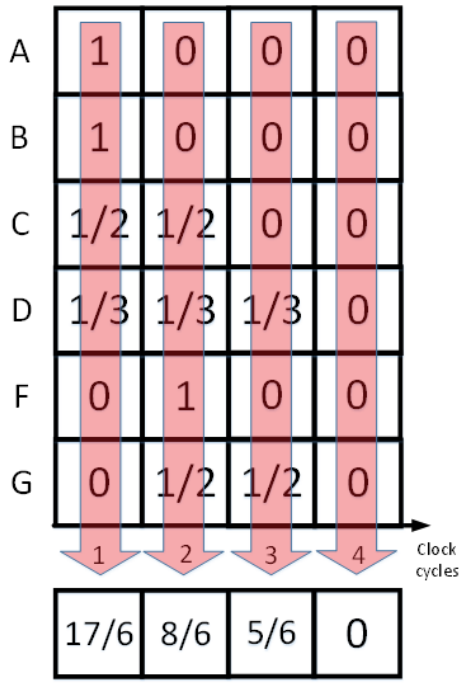

Figure 13: Example of distribution graph for multipliers in DFG in Fig. 11

given in (3.7). This is shown in Fig. 13. In particular, in this case we will have the highest value for our distribution graph in clock cycle 1 (it is $\frac{17}{6}$). Hence, we will have a lot of operation nodes that can be scheduled in clock cycle 1. As we are solving an MR-LCS problem, we want to minimize such a maximum value. Furthermore, in this way, we are implicitly maximizing the efficiency of the needed functional units and at the same time minimizing them. However, a formulation for this is necessary.

3.3.3 Force computation

Now we can compute the so called *force* for a given schedule[1]. This is the new parameter introduced by such a technique and it can be called the heart of this algorithm.

We define force of a given node u_i in clock cycle k as

$$F(u_i, k) = p(u_i, k) \cdot W(g(u_i), k) \tag{3.8}$$

This expresses a sort of "repulsive" force, like that of a spring. For this, what we aim to have is the scheduling of the generic operation node u_i in the clock cycle having the lowest force possible or, better, where the other forces are pushing to. For this, it is introduced the notion of *self force*, expressing how much a given force is weak against the others for scheduling the same operation node u_i:

$$SF(u_i, k) = \sum_{j=asap(u_i)}^{alap(u_i)} (\delta_{j,k} - p(u_i, k)) \cdot W(g(u_i), k) \tag{3.9}$$

where $\delta_{j,k}$ is a Kronecker delta.

In order to evaluate which is the best clock cycle to schedule a particular node u, we look at the lowest self-force value of it per every available clock cycle, because essentially it is the clock cycle in which we will make the total force being the lowest.

Let us analyze, for example, the self force for node G. According to the distribution graph,

---

[1]With the term *schedule* we identify a pair operation node - clock cycle in which it is scheduled.

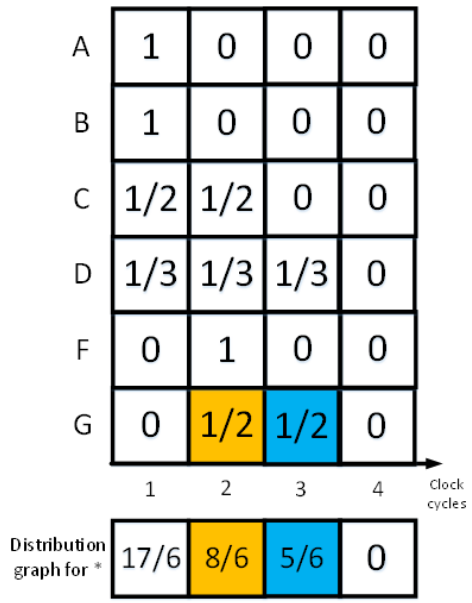

Figure 14: Example of self force computation. Here we want to focus on node G. The DFG is in Fig. 11

we have just two possible schedules (as we can see in Fig. 14). Let us suppose we want to schedule it in clock cycle 2. If we take such a decision, then we will have, as effect, that the distribution graph for cc 2 will increase of a value $1 - \frac{1}{2}$ while it will decrease for clock cycle 2 of $\frac{1}{2}$. So, we will have distribution graph value in 2 being $\frac{8}{6} + (1 - \frac{1}{2}) = \frac{11}{2}$ while in 3 being $\frac{5}{6} - \frac{1}{2} = \frac{2}{6}$. We can see a representation of this in Fig. 15a

Let us suppose, now, that we wish to schedule it in clock cycle 3. The distribution graph will decrease its value in 2 to $\frac{8}{6} - \frac{1}{2} = \frac{5}{6}$ while it will increase in 3 to $\frac{5}{6} + (1 - \frac{1}{2}) = \frac{8}{6}$. We can see a representation of this in Fig. 15b. So, at the end, we will have a lower maximum in the distribution graph if we schedule G in 3.

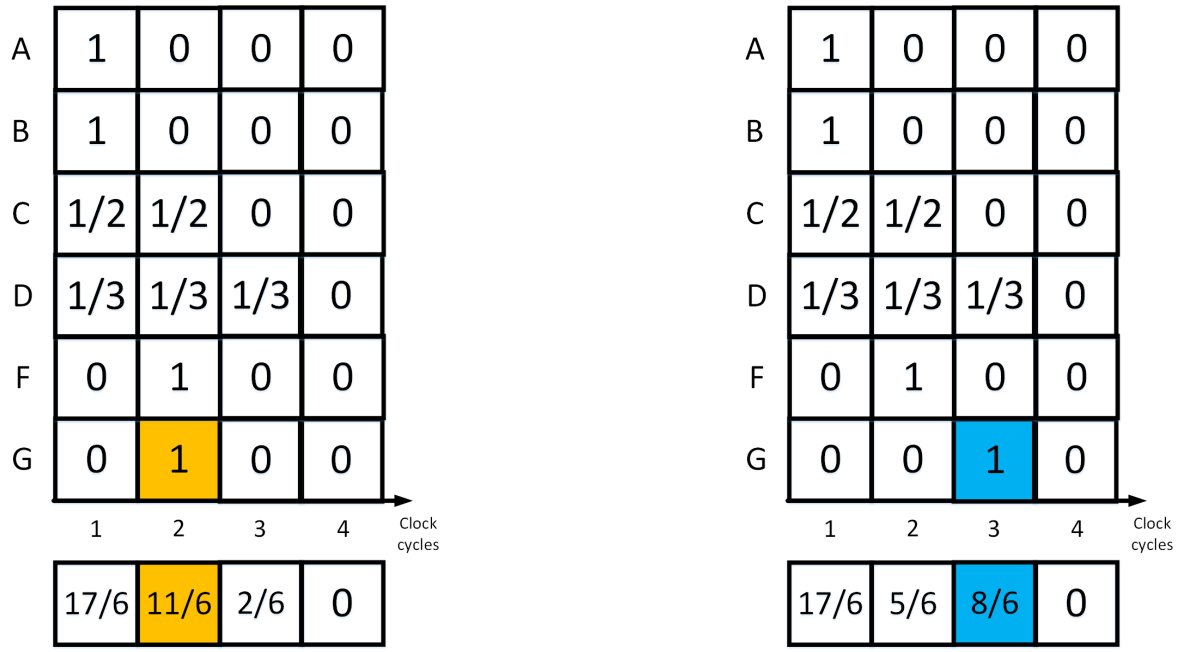

Figure 15: Example of self force computation. Here we are scheduling node G in clock cycle 2 (a) or 3 (b). The DFG is in Fig. 11

The self force formulation in (3.9) performs exactly such an analysis: it will evaluate at the same time the increment of the distribution graph at a certain point and the decrease in all the other possible schedules. For this, choosing the lowest value of self force for performing our scheduling, we will at the same time minimize both the value of the distribution graph in the scheduling point and in all the other points in the mobility range.

3.3.4 Predecessor and successor forces

Scheduling the operation node u_i in a given clock cycle k will affect mobility range of adjacent nodes in the dataflow graph. In particular, we will have the effect of modifying the

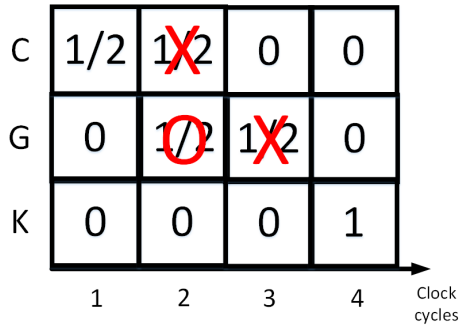

Figure 16: Reduction of mobility range for predecessor of G if scheduled at clock cycle 2. The DFG is in Fig. 11

mobility range for the predecessors of u_i modifying $alap(P(u_i))$ and the mobility range of its successors modifying $asap(V(u_i))$.

Let us take the example of scheduling node G in clock cycle 2. According to the input DFG (in Fig. 11), it has as predecessor node C and as successor K. If we perform such a scheduling, it means that it needs to receive input data at most by clock cycle 2. For this, C needs to produce its output data at most by cc 2 and, for this, it has to be scheduled earlier. This results in the reduction of its mobility range and, for instance, in the reduction of its mobility, which will modify the distribution graph itself. This is visible in Fig. 16 This may be a non-negligible effect for the quality of our solution because it may lead to suboptimalities. For this, we have to take in consideration the effect of the scheduling on predecessors and successors for the scheduled node, together with self force in the choice of scheduling to be performed.

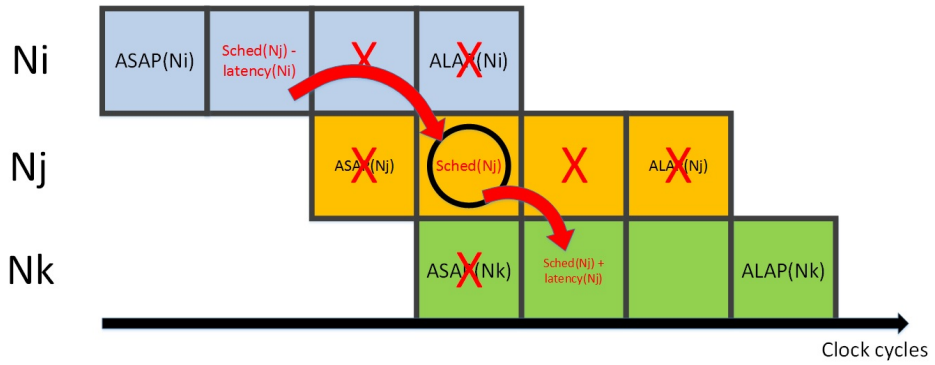

Figure 17: Effect of scheduling node N_j on predecessor N_i and successor N_k

The predecessor force $PF(u_i, k)$, corresponding to scheduling u_i at clock cycle k, is

$$PF(u_i, k) = \sum_{u_h \in P(u_i)} \sum_{j \in MR(u_h)} [\Delta_{pred}(j, k, u_h)]\, W(j) \tag{3.10}$$

where

$$\Delta_{pred}(j, k, u_h) = \begin{cases} p_{new}(u_h, j) - p(u_h, j) \\ if\ j + \lambda(u_h) \\ +\lambda(com) \leq k \\ \\ -p(u_h, j) \\ otherwise \end{cases} \tag{3.11}$$

$$p_{new}(u_h, j) = \frac{1}{\mu_{new}(u_h)} \tag{3.12}$$

$\lambda(u_h)$ is the latency u_h-type node requires and $\mu_{new}(u_h)$ is the new mobility of u_h due to $alap(u_h)$ reduction to $k - \lambda(com) - \lambda(u_h)$ in case it originally was $>= k - \lambda(com) - \lambda(u_h)$ (k is the clock cycle for which the force effect of the current node of interest u_i is being determined).

Instead, the successors force $VF(u_i, k)$ is

$$VF(u_i, k) = \sum_{u_h \in V(u_i)} \sum_{j \in MR(u_h)} [\Delta_{succ}(j, k, u_h)]\, W(j) \tag{3.13}$$

where

$$\Delta_{succ}(j, k, u_h) = \begin{cases} p_{new}(u_h, j) - p(u_h, j) \\ if\ j - \lambda_{com} \leq k \\ \\ -p(u_h, j) \\ otherwise \end{cases} \tag{3.14}$$

Let us analyze more in depth the example previously discussed and pictured in Fig. 16. In particular, let us analyze the predecessor force contribution to the scheduling of G to cc 2. In particular, it can no longer be scheduled in clock cycle 2. For this, we will apply a similar formulation already described for self force. For this, we will have positive contribution

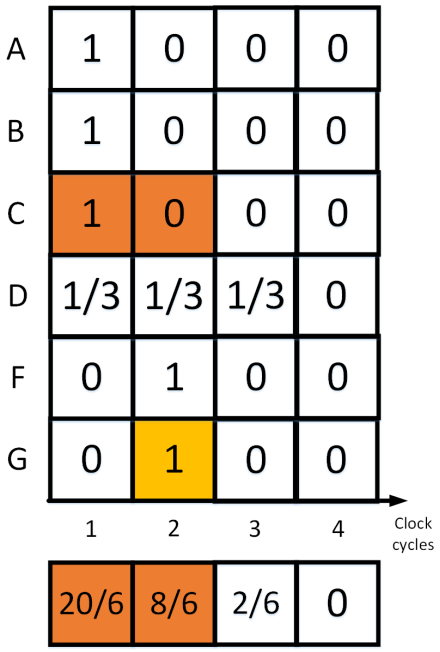

Figure 18: Reduction of mobility range for predecessor of G if scheduled at clock cycle 2 and its effect on the distribution graph. The DFG is in Fig. 11.

(making the evaluation of the current scheduling worse) from all the schedules which will not be available while a negative one from those still available. In this case, in particular, we will have for clock cycle 2 a contribution $1 - \frac{1}{2} \cdot \frac{8}{6}$ which indicates that the current solution will bring the distribution graph having a higher value in 2. As we can see, this is conform to the formulation provided in (3.10).

Finally, the schedule being chosen (u_i, k) is the one having the lowest value, combining self force and predecessors/successors contributions.

$$(u_i, k) = \min_{\forall k, \forall u_i} \{SF(u_i, k) + PF(u_i, k) + VF(u_i, k)\} \tag{3.15}$$

Algorithm 3 Force directed scheduling

procedure FDS(DFG, $\overline{\lambda}$)
  REM_NODES ← NODES(DFG) ▷ Load all the nodes in remaining nodes
  while REM_NODES $\neq \emptyset$ **do** ▷ While there still is one node to be scheduled
    Update asap and alap times for all REM_NODES
    Compute distribution graph for all clock cycles ▷ According to (3.7)
    for all $u_i \in REM\_NODES$ **do** ▷ For all unscheduled nodes compute forces
      for all $ck \in [asap(u_i); alap(u_i)]$ **do**
        Compute $SF(u_i, ck)$ ▷ The computation follows (3.9)
        $PSF(u_i, ck) = 0$
        for all $u_j \in P(u_i), V(u_i)$ **do** ▷ For all the predecessors and successors of u_i
          for all $cc \in [asap(u_j); alap(u_j)]$ **do**
            if cc valid schedule **then** ▷ This follows the formulation in (3.10)
              $PSF(u_i, ck)$-= $SF(u_j, ck)$
            else
              $PSF(u_i, ck)$+= $SF(u_j, ck)$
    Schedule u_{sch}, having min$\{SF(u_i, ck) + PSF(u_i, ck)\}$
    Remove u_{sch} from REM_NODES ▷ You remove it as it is now scheduled
  return (sched(NODES)) ▷ Return all nodes scheduling times

3.3.5 Complexity

Let us first analyze this algorithm without predecessors and successors. In such a condition, we will analyze every node every scheduling step (as we need to compute self forces), this will totally take $O(n^2)$ time. Including predecessors and successors, as their maximum number can be $n-1$, we have an overhead of $O(n)$, making the total complexity being $O(n^3)$.

3.4 Scheduling algorithm for buslet-based designs

The topic here discussed involves the scheduling of operation nodes. As previously seen, a good scheduling algorithm used for minimizing the minimum amount of FUs for a given design is FDS. However, such an approach assumes that an arbitrary number of interconnections is available for the final design and their contribution to the design is negligible. In general, this is not true, as a big number of interconnections will make the design at the routing phase being a very complex problem to be solved.

3.4.1 Communication with dedicated interconnections

For any design using dedicated interconnections, if we schedule node u_i at time $t(u_i)$ and one of its successors v_i at $t(v_i)$ (in this way, if $P(v_i)$ is the set of v_i predecessors and $V(u_i)$ is the set of successors of u_i, we can say that $u_i \in P(v_i)$ and $v_i \in V(u_i)$), the communication $com(u_i, v_i)$ between these can happen in the time frame

$$t[com(u_i, v_i)] \in [t(u_i) + \lambda(u_i); t(v_i) - \lambda_{com}] \tag{3.16}$$

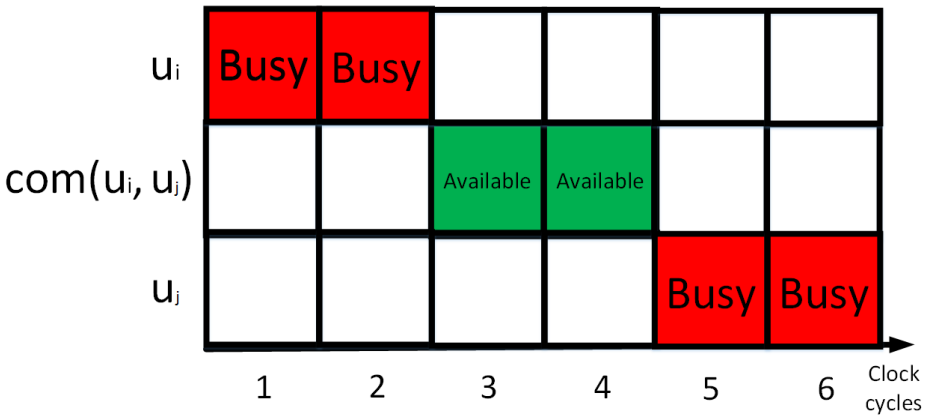

Figure 19: Example of time frame in which the communication between u_i and u_j can happen

where λ_{com} is the latency of the communication, usually

$$\lambda_{com} = 1 \tag{3.17}$$

and deciding when a given communication will take place will not appreciably change the hardware structure as well as the total power consumed[1]. For this reason, such a problem is not taken into account by scheduling, as it involves the management of registers and their allocation. Timing for communications, in this way, will happen after the binding phase.

Let us suppose u_j is a successor of u_i. Further, u_i will have data ready from clock cycle 3 and u_j is scheduled for clock cycle 5 (as pictured in Fig. 19). If $\lambda_{com} = 1$ we can schedule

[1]However, we have to state that, changing communication schedule, we may have effect on peak temperature on the realized device. Anyway, such a problem can not be taken directly into account at high level synthesis stage, as routing still does not exist.

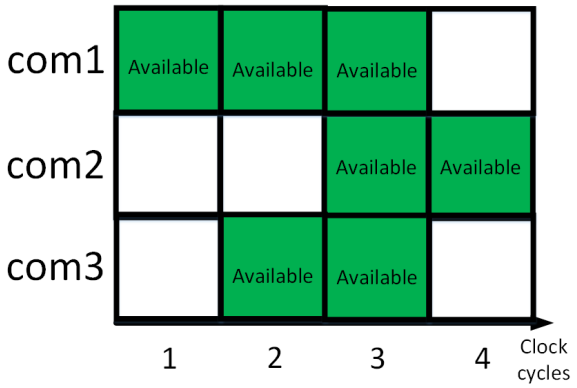

Figure 20: Example of communications possible concurrency.
Such an issue is relevant for our buslet-based design.

the communication to both clock cycle 3 or 4 and, in case it is $\lambda_{com} = 2$, just in clock cycle 3. However, from this we can see that, when we will perform the binding, we really need an interconnection between the functional units u_i and u_j will be bound to and, finally, such a communication needs be performed in any case[1].

3.4.2 Force directed communication scheduling

Here we wish to use wiring structures which will connect at the same time a number of functional units. For this, in order to minimize their minimum needed number, we have to reduce as much as possible their concurrency.

Let us suppose to have three communications having some timeframes (computed as previ-

[1]This statement is always true if u_i and u_j need to be mapped to different types of FUs. If they can be mapped to the same FU, such a communication will never take place as data is already in internal registers of such an FU. We are assuming such an optimization will automatically be performed.

ously stated previously and stated in Fig. 19) as pictured in Fig. 22. As we are going to use buslets, we need to ensure that, in a given buslet, conflicts do not have to happen. For this, one communication at a time can happen on each buslet. In case we decide to schedule all the communications in clock cycle 3, then we need 3 buslets. However, as it is possible to see, assuming $\lambda_{com} = 1$, it is easy to minimize such a number scheduling communications in different clock cycles. So, if our aim is here to minimize the minimum number of needed buslets, the scheduling must happen not on the nodes in the DFG (operation nodes) but on the arcs (communications between operation nodes).

For this, let us assume that operation nodes are not scheduled yet. we can say that the mobility range for communication $com(u_i, v_i)$ between operation node u_i and v_i is

$$\mu[com(u_i, v_i)] \in [asap(u_i) + \lambda(u_i); alap(v_i) - \lambda_{com}] \tag{3.18}$$

An example of communication mobility range can be found in Fig. 21. However, once we will schedule a communication, we will have an effect on the mobility range of the operation nodes. Let us decide to schedule communication $com(u_i, u_j)$ from u_i to u_j in $t[com(u_i, u_j)]$. The new alap time for u_i will be

$$alap(u_i) = t[com(u_i, u_j)] - \lambda(u_i) \tag{3.19}$$

while the new asap time for v_i will be

$$asap(v_i) = t[com(u_i, u_j)] + \lambda_{com} \tag{3.20}$$

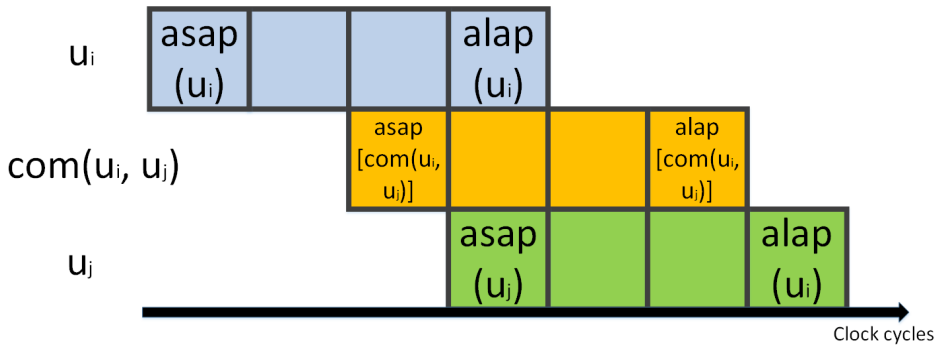

Figure 21: Mobility range for communication between u_i and u_j.
We are assuming here $\lambda(u_i) = 2$ and $\lambda_{com} = 1$.

In general, the situation presented in (3.19) and (3.20) is pictured in Fig. 22. This happens because we need to ensure data from the source being ready by the beginning of the communication from the source u_i and data to be computed by u_j has already been transmitted when u_j is scheduled. For this reason, we need to take into account a reduction of the mobility range for operation nodes. However, as our main objective here is minimizing concurrency between communications, we will perform force directed scheduling for communications (FDCS) independently from the regular FDS. However, we will not have the warranty that mobility for every operation node will drop to 1.

For this, regular FDS has to be performed on the modified mobility ranges after FDCS, in order to get a complete schedule for both communications and operation nodes.

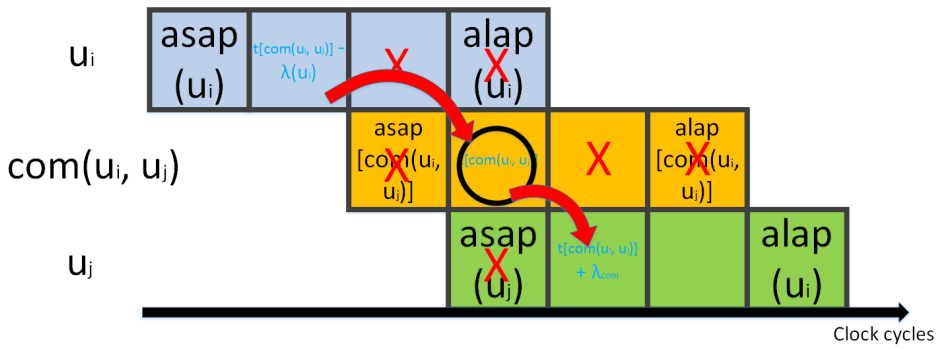

Figure 22: Example of communication scheduling and its effect on mobility ranges for functional units

Algorithm 4 Force directed scheduling

procedure FDCS(DFG, $\overline{\lambda}$)
  REM_COMS ← EDGES(DFG) ▷ Load all the edges of the input DFG
  while REM_COMS $\neq \emptyset$ **do** ▷ While there still is one com to be scheduled
    Update asap and alap times for all REM_COMS
    Compute distribution graph for all clock cycles ▷ According to (3.7)
    for all $e_i \in REM\_COMS$ **do** ▷ For all unscheduled coms compute forces
      for all $ck \in [asap(e_i); alap(e_i)]$ **do**
        Compute $SF(e_i, ck)$ ▷ The computation follows (3.9)
        $PSF(e_i, ck) = 0$
        for all $e_j \in P(e_i), V(e_i)$ **do** ▷ For all the predecessors and successors of e_i
          for all $cc \in [asap(e_j); alap(e_j)]$ **do**
            if cc valid schedule **then** ▷ This follows the formulation in (3.10)
              $PSF(e_i, ck)$-= $SF(e_j, ck)$
            else
              $PSF(e_i, ck)$+= $SF(e_j, ck)$
    Schedule e_{sch}, having min$\{SF(e_i, ck) + PSF(e_i, ck)\}$
    Remove e_{sch} from REM_COMS ▷ You remove it as it is now scheduled
  return (sched(COMS)) ▷ Return all communications scheduling times

3.4.3 Complexity

Let us first analyze this algorithm without predecessors and successors. In such a condition, we will analyze every edge (as we are talking about communications) every scheduling step (as we need to compute self forces), this will totally take $O(e^2)$ time. Including predecessors and successors, as their maximum number can be $e - 1$, we have an overhead of $O(e)$, making the total complexity being $O(e^3)$.

3.5 Summary of scheduling algorithms

Our scheduling is articulated in two different parts, sequentially executed:

1. Scheduling for communications: applies exactly the same model than FDS (3.9) to communications, including communication predecessors/successors analysis (3.10)
2. Scheduling for operation nodes, according to remaining mobility ranges

The complexity for FDCS is $O(e^2)$ while the complexity for FDS is $O(n^2)$. As these algorithms are applied sequentially and we can not say anything on the input DFG connectivity, the total complexity for the scheduling is $O(n^2 + e^2)$.

Figure 23: General dataflow for the scheduling technique used

CHAPTER 4

BINDING ALGORITHMS

4.1 General concepts to be taken into account

After communication and node scheduling, we need to bind each operation node to a FU and each communication between operation nodes to a target buslet (these bindings also lead to binding of FU connections to buslets). From FDCS and FDS we obtained:

- *Scheduling times for operation nodes* $t(u_i)$
- *Scheduling times for communications* $t[com(u_i, u_j)]$
- *Lower bound on the final number of functional units.* We may finally need more FUs than those determined during scheduling because of the fanin/fanout constraints we impose on binding that were not imposed during scheduling (we cannot determine fanin/fannout during scheduling).

  Let us assume we need to bind operation node u_i to a target functional unit. This operation node requires a FU able to perform the operation needed by u_i. Let us identify the type of functional units able to perform the operation described by operation node u_i as $k(u_i)$ and let us say the set of FUs of type $k(u_i)$ is $FU[k(u_i)]$. We identify the subset $FU_{avl}(u_i)$ of available functional units as the set of functional units $r_l \in FU[k(u_i)]$ which

are idling in the clock cycles range $[t(u_i); t(u_i) + \lambda(u_i) - 1]$. More formally we can say that

$$r_l \in FU_{avl}(u_i) \left| \prod_{k=t(u_i)}^{t(u_i)+\lambda(u_i)-1} \delta_{ck}(r_l, k) = 1 \right. \tag{4.1}$$

where

$$\delta_{ck}(r_l, k) = \begin{cases} 1 & if\ r_l\ idling\ in\ k \\ 0 & otherwise \end{cases} \tag{4.2}$$

Further, we also observe that we have

$$FU_{avl}(u_i) \subseteq FU(k(u_i)) \tag{4.3}$$

Let us here define, for a given functional unit r_l, the set of all the buslet in fanin as $fin(r_l)$ and the set in fanout as $fout(r_l)$. From this, we know that the fanin of r_l is the cardinality of $fin(r_l)$ and that the fanout is the cardinality of $fout(r_l)$. We define fanin for the functional unit u_i as $|fin(r_l)|$ and fanout as $|fout(r_l)|$. Because of the constraint on maximum allowable fanin and fanout on any functional unit, we need also to ensure that, after binding, for any r_l in our design, we will have

$$|fin(r_l)| \leq max\_fanin \tag{4.4}$$

and

$$|fout(r_l)| \leq max\_fanout \tag{4.5}$$

It may happen that, if we will not have any $r_l \in FU_{avl}(u_i)$ also respecting (4.4) and (4.5), we may need to add some functional unit of type $k(u_i)$ to $FU[k(u_i)]$ and, consequently, to $FU_{avl}(u_i)$.

- *Lower bound on the final number of buslets.* The number of buslets can also increase beyond this value during binding due to the max buslet cardinality constraint on it that we cannot determine during scheduling.

  Let us assume that the set of buslets in our design is B. In this way, any buslet in the design $b_k \in B$. Furthermore, the number of total buslets in our design is given by the cardinality of B. For this, we say that our design has $|B|$ buslets. Let us assume we need to bind $com(u_i, u_j)$. Let us define *set of available buslets* $B_{avl}[com(u_i, u_j)]$ as set of buslets which are idling in clock cycles window $[t[com(u_i, u_j)]; t[com(u_i, u_j)] + \lambda_{com} - 1]$. More formally,

$$b_i \in B_{avl}[com(u_i, u_j)]| \prod_{k=t[com(u_i,u_j)]}^{t[com(u_i,u_j)]+\lambda_{com}-1} \delta_{ck}(b_i, k) = 1 \tag{4.6}$$

  where $\delta_{ck}(b_i, k)$ is defined in (4.2).By consequence, we also have

$$B_{avl}[com(u_i, u_j)] \subseteq B \tag{4.7}$$

  Let us assume that

$$FU_{avl}(u_i) \neq \emptyset \tag{4.8}$$

and that

$$FU_{avl}(u_j) \neq \emptyset \tag{4.9}$$

If $\forall r_l \in FU_{avl}(u_i), \forall r_m \in FU_{avl}(u_j), \forall b_k \in B_{avl}[com(u_i, u_j)]$, after binding, we will have

$$|b_k| > max\_buslet\_cardinality \tag{4.10}$$

then we need to add another buslet to B and, consequently, to $B_{avl}[com(u_i, u_j)]$, in order to use it to bind $com(u_i, u_j)$.

At the end of binding process, what we aim to reach is to satisfy buslet cardinality constraint as well as maximum fanin and fanout. Moreover, just using these conditions is not sufficient in order to have a design in which we have low wirelength: this has to be a driving factor in our whole binding process. For this reason, what we want to use is a wirelength estimation model giving us an idea on which solution, among all those available, may be better to choose.

4.1.1 Wirelength model

As we need a very quick-to-be computed wirelength model, without performing any kind of floorplanning, we estimate wirelength for a given buslet b_k based on only two parameters:

1. current buslet cardinality $|b_k|$
2. maximum allowed buslet cardinality $\max\{|b|\}$

We assume for simplicity that all FUs have the same square shape and same area. We then estimate the buslet WL as the sum of Manhattan distances of all FUs in it from the center of

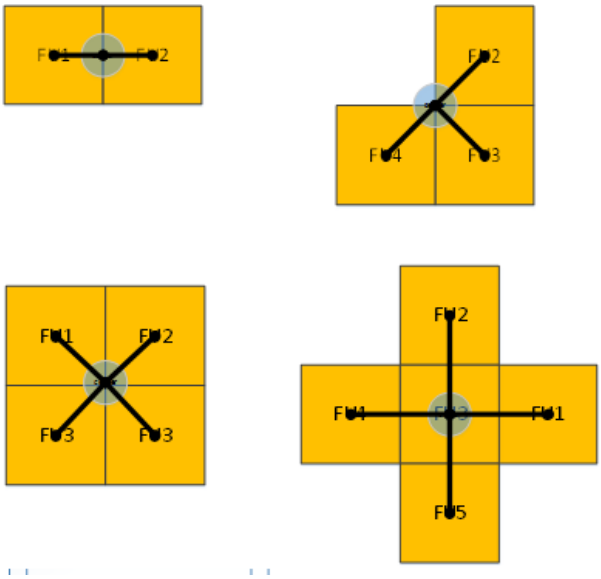

Figure 24: Examples of computation of wirelength for buslet cardinalities from 2 to 5

gravity of the most "compact" area that the buslet and its FUs lie in. This is a obviously lower bound on the WL of the buslet, since due to FUs being connected to multiple buslets, not all buslets can be located in their most compact areas determined independent of other buslets. However, as empirical evidence shows (Fig. 50 and Table III), this simple WL model works well in yielding a final design with low floorplanning determined WL.

4.1.2 Effect of fanin/fanout constraints on binding

As previously stated, during our binding, we will have to deal with fanin and fanout constraints for ach functional unit. Let us say that FU_l is a generic functional unit required from

our binding. We need to ensure that the number of buslets it is connected as fanin will not exceed a given threshold level max_fanin

$$fanin(FU_l) \leq max\_fanin \tag{4.11}$$

Same issue will apply to fanout:

$$fanout(FU_l) \leq max\_fanout \tag{4.12}$$

We need to ensure this during binding. However, such a problem is not straightforward. Let us say that we want to bind operation node u_i to functional unit FU_l, which, as it is connected to the operation node u_j, bound to FU_m, will require a link with it through a buslet b_k. When we perform such an operation, we have to ensure that, at least,

$$fanin(FU_l) + \sum_{\forall k(u_n) \in P_{unb}(u_i)} \delta_{conn}(FU_l, k(u_n)) \leq max\_fanin \tag{4.13}$$

where

$$\delta_{conn}(FU_l, k(u_n)) = \begin{cases} 1 & if\ FU_l\ not\ connected\ to\ any\ FU\ of\ type\ k(u_n) \\ 0 & otherwise \end{cases} \tag{4.14}$$

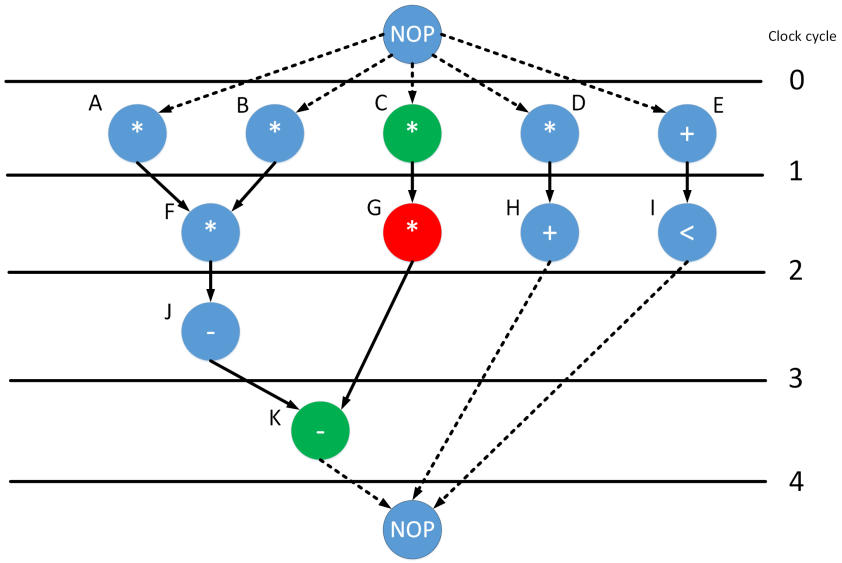

Figure 25: Example of fanin/fanout constraint to be taken into account if binding G (in red). FU which binds G needs to be able to be connected to the connected to the ones the green op nodes will be mapped to.

and $P_{unb}(u_i)$ is the set of unbound predecessor nodes of u_i. This is because we need to ensure that, after binding u_i to FU_l, we will be able to respect maximum fanin constraint. Such a constraint will apply to the fanout constraint:

$$fanout(FU_l) + \sum_{\forall k(u_n) \in V_{unb}(u_i)} \delta_{conn}(FU_l, k(u_n)) \leq max\_fanout$$

where $V_{unb}(u_i)$ is the set of unbound successor nodes of u_i.

A practical example of this is presented by Fig. 25. Here, when we decide to bind the operational node G to a given functional unit FU_i, we also need to ensure it will be able to be connected to at least one more functional unit (that K will be mapped to): in fact, C can potentially be

mapped to the same FU as it is of the same type. However, such a constraint may be too tight. Let us assume we have $u_n \in V_{unb}(u_i)$. When binding u_i to FU_l, we have FU_l already connected to the functional unit FU_m, with $k(u_n) = k(FU_m)$. However, when binding u_n, we may have the case that FU_m has already been bound from another operation node. This means that we may not been able to satisfy maximum fanout constraint. Same issue applies to predecessors and fanin constraint. In order to be sure to satisfy fanin/fanout constraints, we need to use the loose constraints

$$fanin(FU_l) + |P_{unb}(u_i)| \leq max\_fanin \tag{4.15}$$

$$fanout(FU_l) + |V_{unb}(u_i)| \leq max\_fanout \tag{4.16}$$

This may oversize the constraint but this will certainly allow us to satisfy maximum fanin and fanout constraints.

4.2 Chronological binding

The first finding algorithm we are going to introduce is called *chronological binding* (CB). This is because it will perform binding moving chronologically: if we are binding $com(u_i, u_j)$, then it means that we will have already bound all the communications $com(u_k, u_l)$ having $t[com(u_k, u_l)] < t[com(u_i, u_j)]$. This is a greedy strategy that will ensure us to bind the absolute locally best solution for the current communication. This technique, in its semplicity, will bring with itself two huge benefits:

1. Makes reuse of hardware available more natural
2. Reduces the solution space to be explored

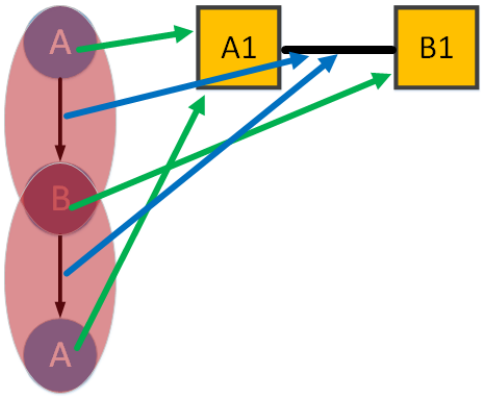

Figure 26: Example of reuse of previously bound buslets.
In blue communications binding, while in green operation nodes binding.

4.2.1 Reuse of already available hardware

Binding communications in chronological way, we will be able to more easily reuse already existing hardware. Let us suppose we have bound $com(u_i, u_j)$ to b_k, binding also u_i to the functional unit r_0 and u_j to r_1. When its communication successor $com(u_j, u_l)$ needs to be bound, considering that u_j has already been bound to r_1, if resource type $k(u_l) = k(u_i)$ and r_0 is an available FU, then it is possible to bind such a communication without any resource adding. We have also to add that it is very likely for both r_0 and b_k to be available as we are moving chronologically.

Let us analyze an example to better understand this. Looking at Fig. 26, we see that, moving chronologically, we will first bind $com(A, B)$. Let us assume A is bound to the FU A1 while B to B1. In order to perform the communication between these functional units, a certain interconnection is created (as we will use buslets, it will be through one of these). When

we will need to bind successor communication, if it is with an "A-type" operation node, we can reuse such a communication. Furthermore, as we are moving chronologically and if we realistacally assume $\lambda_{com} = 1$, if such a connection exists, it can certainly be reused. This naturally contributes int the wire efficiency increment.

4.2.2 Reduction of solution space

In the general case (except at the beginning of our binding process), when we need to bind a communication $com(u_i, u_j)$, we have already bound u_i to a target functional unit, as we have also bound the predecessor communication $com(u_h, u_i) \in P[com(u_i, u_j)]$. If we consider the space of the solutions as combination of communications, buslets, source and destination functional units, each binding step not only we are analyzing one communication at a time but we already know which is the functional unit the source is bound to. For this, as the total number of buslets will be at most the number of edges e of the DFG and the quantity of functional units is the amount of nodes n of the DFG, then we say that CB will scale the size of solutions to be analyzed from n^2e^2 to ne.

An example of such an issue can be observed in Fig. 27. Let us suppose we already bound com(A, B) with, of course, operation nodes to functional units. When we are going to bing the successor communication com(B, A), we already find B being bound to B1. This, as it is here evidenced, reduces the solution space to be explored. This is something constant in all the binding execution, significantly reducing the amount of possibilities to be computed each step.

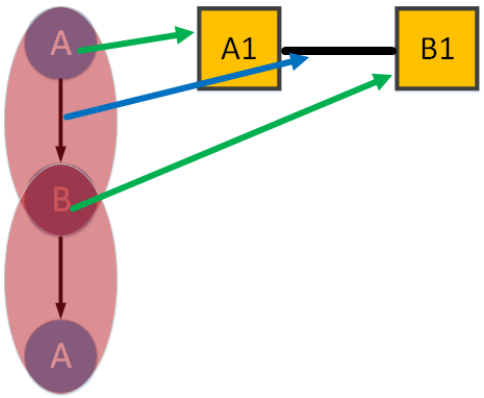

Figure 27: Example of binding solution space reduction.
In blue communications binding, while in green operation nodes binding.

Algorithm 5 General chronological binding algorithm

procedure CHRONOLOGICAL_BINDING(sched(NODES, COMS))
Sort_growing_sched_time(COMS)
partial_soln_set = ∅
initial_empty_binding = initialize(buslets, FUs) ▷ Initialized empty design
push(partial_soln_set, initial_empty_binding) ▷ Initialized list of partial solutions
for all $com_i \in COMS$ **do**
com_bound = false
while com_bound = false **do**
Compute all possibilities and put them in Local_solns
if Local_solns = ∅ **then** ▷ If no possibility available, then we lack of FU or buslet
Add needed resource (FU or buslet)
else
com_bound = true
if similar_soln_pruning = true **then**
Local_solns = Similar_solns_cutter(Local_solns) ▷ This function will be explained in Sec. 4.3
Sort_in_decreasing_WL_order(Local_solns)
partial_soln_set = first_n_solns(Local_solns, soln_held_per_step) ▷ We take the best n solutions
return (first(partial_soln_set)) ▷ We return at the end the design having the best wirelength

4.2.3 Complexity

About the binding, as we are binding e different communications to b buslets and to r functional units, a first estimation of the complexity could be $O(ebr^2)$. However, according to our former analysis, source nodes are most of the times already bound. For this, the complexity will reduce to reduces to $O(ebr)$. We can also state that certainly $b \leq e$ (as at most we will have one buslet per communication) and $r \leq n$ (as at most we will have one functional unit per node and in the worst case we will have only one resource type). According to this, the complexity for CB will be $O(ne^2)$. Finally, let us say that we want to propagate each binding step at most S best solutions: the complexity becomes $O(Sne^2)$.

4.3 Chronological binding with similarity reduction

Chronological binding with similarity reduction (CBSR) is a natural extension of simple CB. When we decide to hold a number of locally "optimal" solution, in fact, there may be the case that we are holding some *similar* solution. We can say that two solutions are similar if they will bring, at the end, to the same buslet structures. This means that, at the end of our binding process, if we say that solution S_1 is similar to solution S_2, then the set of buslets $B_1 \in S_1$ will be exactly the same as the set of buslets $B_2 \in S_2$. More formally, we can say that

$$\forall b_i \in B_1 \exists b_j \in B_2 \,|\forall r_k \in b_i \Rightarrow r_k \in b_j \tag{4.17}$$

We can see an example in Fig. 28. Here we see that Solution A is similar to Solution B as buslet b1 and b2 are inverted and functional units bind exactly the same operation nodes in the

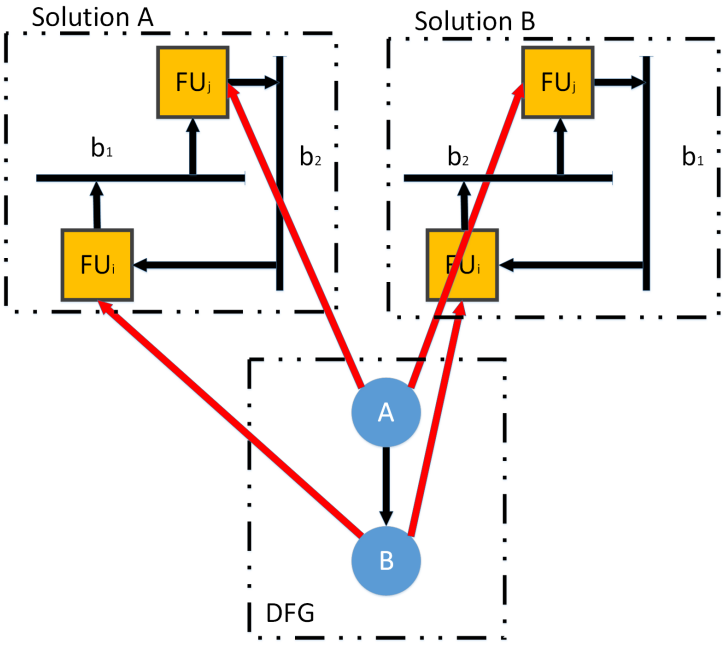

Figure 28: Example two solutions similar differing for buslets labels.
In red operation nodes bindings, in black buslets and in yellow functional units.

DFG. However, such a definition seems to be weak for r_k: in fact, potentially any $r_i \in FU[k(r_k)]$ can have the same schedule of r_k in a different design: it is like swapping labels.

In the example presented by Fig. 29, we not only have buslet labels swapped, but also those for functional units. We are here assuming FU_i and FU_j are of the same type. We see that the schedules for both are overlapping, as well as the structure of the buslets they are connected to. Here it is not necessary that they are bound exactly to the same operation nodes, provided

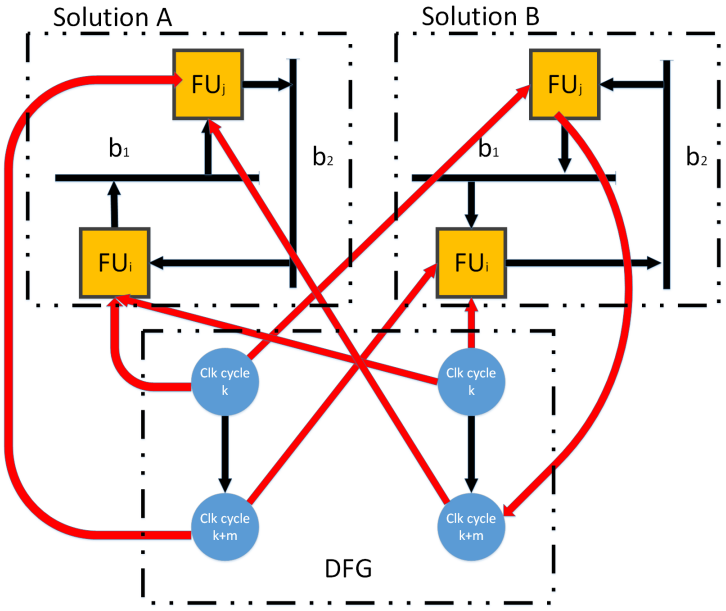

Figure 29: Example two solutions similar with different binding for both buslets and functional units. In red operation nodes bindings, in black buslets and in yellow functional units.

that the buslets they are connected to are the same and their busy clock cycles are exactly identical. For this, we should modify statement (4.17) allowing any resource in b_j to match r_k:

$$\forall b_i \in B_1 \exists b_j \in B_2 \left| \forall r_k \in b_i \exists r_l \in b_j \right| \forall ck \in [0; \overline{\lambda} - 1] \delta_{ck}(r_k, ck) = \delta_{ck}(r_l, ck) \tag{4.18}$$

In general we can say that, if (4.18) is true, then S_1 and S_2 are similar. Let us give, now, an extensive formulation of this.

Let us suppose we need to bind $com(u_i, u_j)$. We know that the set of available FUs to be

mapped to u_i is $FU_{avl}(u_i)$ and for u_j is $FU_{avl}(u_j)$. Moreover, the set of available buslets is $B_{avl}(com(u_i, u_j))$. If we say that

$$b_k \in B_{avl}(com(u_i, u_j)) \tag{4.19}$$

$$r_l \in FU_{avl}(u_i) \tag{4.20}$$

$$r_m \in FU_{avl}(u_j) \tag{4.21}$$

we can say that the quantity of binding possibilities $|S(com(u_i, u_j))|$ is

$$|S(com(u_i, u_j))| = \sum_{\forall b_k} \sum_{\forall r_l} \sum_{\forall r_m} \delta_R(r_l, b_k) \cdot \delta_R(r_m, b_k) \cdot \delta_B(r_l, r_m, b_k) \tag{4.22}$$

where, having

$$\Delta_{R(fanout)}(r_l, b_k) = \begin{cases} 1 & if\ b_k \notin\ fanout(r_l) \\ 0 & otherwise \end{cases} \tag{4.23}$$

it is

$$\delta_R(r_l, b_k) = \begin{cases} 1 & if fanout(r_l) + \Delta_{R(fanout)}(r_l, b_k) \leq max\_fanout \\ 0 & otherwise \end{cases} \tag{4.24}$$

and in the same way we have

$$\Delta_{R(fanin)}(r_m, b_k) = \begin{cases} 1 & if\ b_k \notin\ fanin(r_m) \\ 0 & otherwise \end{cases} \tag{4.25}$$

$$\delta_R(r_m, b_k) = \begin{cases} 1 & iffanin(r_m) + \Delta_{R(fanin)}(r_m, b_k) \leq max\_fanin \\ 0 & otherwise \end{cases} \tag{4.26}$$

and, saying that

$$\Delta_B(r_l, r_m, b_k) = \begin{cases} 2 & if\ (r_l \wedge r_m) \notin\ busletset(b_k) \\ 1 & if\ \exists!(r_l, r_m) \notin\ busletset(b_k) \\ 0 & otherwise \end{cases} \tag{4.27}$$

we have

$$\delta_B(r_l, r_m, b_k) = \begin{cases} 1 & if\ |b_k| + \Delta_B(r_l, r_m, b_k) \leq max\_bus\_cardinality \\ 0 & otherwise \end{cases} \tag{4.28}$$

From this, we can see that we may potentially have several binding possibilities, resulting with the same estimated wirelength, which are "similar": for example, assuming that we are just beginning out binding (so, all buslets are empty and all FUs available), if we need to bind $com(u_i, u_j)$, which one may be the difference between binding it to buslet b_k or b_{k+1}, as both of them are empty?

If we desire to save more than one binding possibility for every step, as those "similar" solutions will have exactly the same wirelength, it may be necessary to implement a *similar solutions detector*, in order to prune all solutions leading to same final solutions.

In order to find solution S^* is similar to S^{ref}, we need to look at the following issues:

1. Each buslet $b^* \in S^*$ needs to have a corresponding buslet $b^{ref} \in S^{ref}$ such that

$$|b^*| = |b^{ref}| \tag{4.29}$$

2. Each buslet $b^* \in S^*$ needs to have a corresponding buslet $b^{ref} \in S^{ref}$ so that, for each clock cycle m, defining, for the generic resource[1] r

$$\delta_{ck}(r, m) = \begin{cases} 0 & if\ r\ idling\ in\ m \\ 1 & otherwise \end{cases} \tag{4.30}$$

we need to have

$$\delta_{ck}(b^*, m) = \delta_{ck}(b^{ref}, m) \tag{4.31}$$

$\forall m$ value.

3. Then, assuming b_k^{ref} has been found similar to b_k^*, $\forall$ functional unit $FU_i^{ref} \in b_k^{ref}$, we need to find a functional unit $FU_i^* \in b_k^*$ such that

   (a) Has same busy clock cycles and, in particular, $\forall m$

$$\delta_{ck}(FU_i^{ref}, m) = \delta_{ck}(FU_i^*, m) \tag{4.32}$$

---

[1]Here with resource we mean both FU or buslet.

(b) Has same fanin and fanout:

$$\begin{cases} fanin(FU_i^{ref}) = fanin(FU_i^*) \\ fanout(FU_i^{ref}) = fanout(FU_i^*) \end{cases} \tag{4.33}$$

4.3.1 Justification of similarity-based pruning

Let us imagine two solutions S_1 and S_2 are similar at step $l-1$ but, at step l, while binding $com(u_i, u_j)$, S_1 will have, among all its binding possibilities, a solution with a better best wirelength than S_2. We can have this when we need to add FU_i to a buslet b_k in S_2 while we do not do the same in S_1. This happens if:

1. $\nexists b_1 \in S_1, b_2 \in S_2 |\ |FU_{avl}(u_i) \in b_1| = |FU_{avl}(u_i) \in b_2|$

   If we have not the same number of same type FUs available connected to the same buslet, then we may need to add one more of such a FU to the buslet, resulting in a cardinality increment and, consequently, in a wirelength increase. However, this goes against condition (4.32).

2. $\nexists FU_1 \in S_1,\ FU_2 \in S_2 \mid k(FU_1) = k(FU_2)$,

   $|b_{avl}[fin(FU_1)]| = |b_{avl}[fin(FU_2)]|$

   If we need to use a FU as source and the available different fanin buslets are different, then we need to add one more buslet to $fanin(FU_2)$. However, as we have ensured (4.33) for all FUs linked to buslets having (4.29), this can not happen.

3. $\nexists FU_1 \in S_1,\ FU_2 \in S_2 \mid k(FU_1) = k(FU_2),$

   $|b_{avl}[fout(FU_1)]| = |b_{avl}[fout(FU_2)]|$

   Same as previous point.

Algorithm 6 Similarity detector algorithm

```
procedure SIMILAR_SOLNS_CUTTER(Input_solns) ▷ Here I take as input the set of solutions to verify
    Output_solns = ∅
    for all S_i ∈ Input_solns do
        similar = false
        for all S_j ∈ Output_solns do
            if all buslets in S_i have a similar in S_j then
                if all FUs in S_i have a similar in S_j then
                    similar = true
        if similar = false then push(Output_solns, S_i)
    return (Output_solns)
```

4.3.2 Complexity

Similarity reduction will certainly improve the quality of solutions. However, it may introduce some overhead in the computation of the final solution. It will concur, at the end of each binding step, in the formation of the set of the Y partial solutions to be propagated in the next binding steps. For this, for each binding step ($O(e)$) we will analyze the Y top ranked (according to our driving metrics which, in this case, is wirelength) solutions ($O(Y)$). For each of these, we will try to match buslets ($O(e)$) and functional units ($O(n)$). Let us say $s_n \in Y$.

y_0 will be matched with $Y-1$ other solutions, y_1 with $Y-2$ and so on. With this, we have a number of matchings to be tested which is $O(Y^2)$[1]. So, the total complexity added from simularity reduction will be $O(Y^2ne)$. With this, we will have that total complexity for SRCB will be $O(Yne(e+Y))$.

4.4 Chronological lookahead binding

A possible development of chronological binding algorithm is to evaluate the fouture implications for a given binding solution chosen. In fact, if we perform choices in function of the current design structure without taking care of future effects of it, then we may easily fall in suboptimal solutions.

Let us assume we are at binding step $i-1$ and our current solution is S_{i-1} (with this we mean the set of both functional units and buslets already bound). We know that each communication

---

[1]We know that

$$\sum_{i=0}^{N-1} i = \frac{N(N-1)}{2}$$

can be bound to a buslet only, as well as to a source FU and a destination one. Let us say that such a quatern of values is identified by $q_k \in Q_i$[12]. We see that

$$S_i = S_{i-1} + q_{best} \tag{4.35}$$

where $q_{best} \in Q_i$ is the quatern of values which will make the increase of total wirelength the lowest. Each $q_k \in Q_i$ may significantly modify the sets Q_{i+f} with f positive integer. Let us assume our q_{best} involved the binding of $com(u_h, u_j)$ to buslet b_k and u_j to r_m. When we will bind its successor communication $com(u_j, u_l)$, we will already have u_j bound to r_m and, if no a connection is available between r_m and any functional unit in $FU_{avl}(u_l)$, then extra wiring cost will be needed. However, if we will be able to detect this in advance, then we may be able to save extra wirelength.

So, let us say we need to compute the partial solution at the i-th step allowing the inspection of the next k levels. Our partial solution will be

$$S_i = S_{i-1} + q_{best}(i + k - 1) \tag{4.36}$$

---

[1]In particular, we can extensively write

$$q_k = (com_i, b_j, r_l, r_m) \tag{4.34}$$

where com_i is the bound communication, b_j is the buslet com_i has been bound to, r_l is the source FU while r_m is the destination one.

[2]Note here that, as we are moving in chronological order, we will analyze each time all the possible binding solutions for the same communication com_i. For this reason, we will have as Q sets as it is the total latency $\overline{\lambda}$

where $q_{best}(i+k-1)$ is the quatern producing, to the $i+k-1$ binding step, the best wirelength.

Algorithm 7 Chronological binding with lookahead

procedure CHRONOLOGICAL_BINDING_WITH_LOOKAHEAD(sched(NODES, COMS))
    sort_coms_per_ascending_sched_time() ▷ From lowest to highest scheduling time
    root_soln = initialize_binding_soln()
    for all $com_i \in coms\_to\_bind$ **do**
        best_LA_wl = ∞
        for all available buslets b_k, source FU_m, destination FU_n **do** ▷ Any combination of these
            local = bind(S_j, com_i, b_k, FU_m, FU_n) ▷ Here we will bind the quatern (com_i, b_k, FU_m, FU_n) to S_j
            LA_wirelen = wl_lookahead(local, com_i, coms_to_bind, LA_level) ▷ We call lookahead function
            if LA_wirelen < best_LA_wl **then** ▷ This was implemented holding 1 solution only
                best_local_soln = local
                best_LA_wl = LA_wirelen
        root_soln = best_local_soln
    return (root_soln)

4.4.1 Complexity

We already know that usual chronological binding will take $O(ne^2)$. However, here we need to take into account that, when we are performing the lookahead search, we will try any possible combination between any buslet ($O(e)$), any source ($O(n)$) and any destination ($O(n)$). All of this will be performed for the next, let us say, k steps, each binding step.

The overhead introduced by lookahead technique is, this way, $O(ken^2)$, bringing the total algorithm complexity to $O(kn^3e^3)$.

Algorithm 8 Wirelength lookahead

procedure WL_LOOKAHEAD(binding_soln, last_bound_com, coms_to_bind, LA_level) ▷ This performs the lookahead and returns the best wirelength after k binding steps.
  if LA_level = 0 **then**
    return wirelength(binding_soln) ▷ This function will compute wirelength for a given solution
  current_com = next_com(last_bound_com, coms_to_bind) ▷ We compute the chronologically next com of last_bound_com
  best_LA_wl = ∞
  for all available buslets b_k, source FU_m, destination FU_n **do**▷ any combination of these
    local = bind(S_j, com_to_bind, b_k, FU_m, FU_n)
    LA_wirelen = wl_lookahead(local, current_com, coms_to_bind, LA_level-1) ▷ Recursive call: we decrease LA level and we pass the new bound com and the new partial solution (local)
    if LA_wirelen < best_LA_wl **then**
      best_LA_wl = LA_wirelen
  return (best_LA_wl) ▷ LA function will return just the best wirelength value found

4.5 Simultaneous binding of iso-scheduled COMs

According to the algorithm developed so far, what we are going to do in our design is binding the best solution(s) moving from one communication to another. Communications are sorted in ascending scheduling time order; so, we are binding communications according to this. However, what shall we expect to do if more than one communication has the same scheduling time? How to sort them? Can this lead us to suboptimal solutions?

If we assume a generic communication latency $\lambda_{com} > 0$, for any nodes in the DFG u_i, u_j, u_k, u_l, if we have $|t[com(u_i, u_j)] - t[com(u_k, u_l)]| < \lambda_{com}$, then the two communications can not be scheduled to the same buslet. However, depending on the order they are bound, we may have different effects on the resulting output.

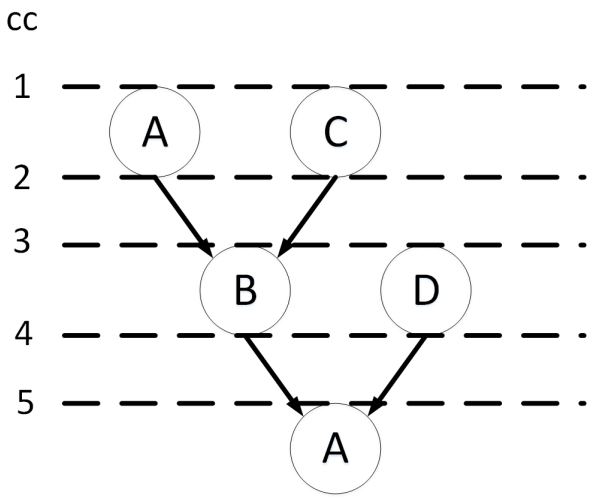

Figure 30: Scheduling example in order to figure out why simultaneous binding matters. Here we are assuming latency for both FUs and communications 1.

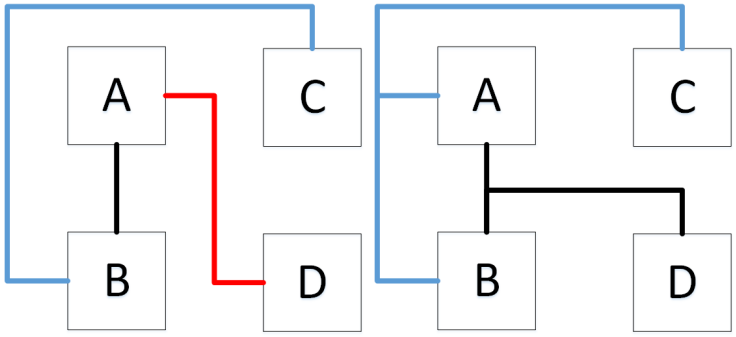

Figure 31: Binding obtained by using CB (on the left, (a)) or simultaneous binding of iso-scheduled communications (on the right, (b)).
Input scheduling is taken by Fig. 30.

Let us assume we have, as input, the scheduling in Fig. 30. Further, let us also assume we have a maximum cardinality constraint of 3. If we use Chronological binding technique, when we will bind com(B, A), we will reuse the already existing interconnection between these FUs (in Fig. 31a, it is the black wire). However, in order to bind com(D, A), as we can not use the black wire because already busy in clock cycle 4 and we can not use the blue wire because we would violate the buslet maximum cardinality constraint, we need to add another buslet (the red one). However, if we analyze simultaneously scheduled communications, we can be able to find better configurations, satisfying the constraints and saving on the number of buslets (and, in this way, increasing the wire efficiency). This is pictured in Fig. 31b. Here we have bound com(B, A) to the blue buslet, connecting A to it ans com(D, A) to the black one, adding D to it.

Hence, for this simple example we can formalize as follows: if we have that

$$\forall com(u_i, u_j) \in DFG \exists com(u_k, u_l) \in DFG | t[com(u_k, u_l)] < t[com(u_i, u_j)] + \lambda_{com} \tag{4.37}$$

then we have to take into account all the edges being bound in any possible order (this means that the complexity will be $e!$). However, in order this to happen, some conditions need to be satisfied. We can primarily say that, if we have

$$\forall u_i \in DFG \exists k(u_i) | \lambda[k(u_i)] \geq \lambda_{com} \tag{4.38}$$

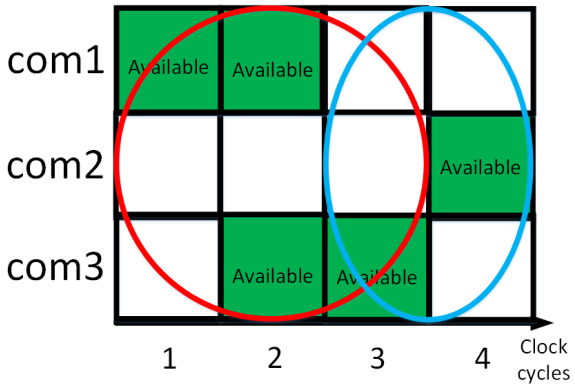

Figure 32: Example of domino effects when considering overlapping communications. Here we are assuming $\lambda_{com} = 2$.

Furthermore, we usually have

$$\lambda_{com} \leq \lambda[t(u_i)] \forall u_i \in DFG \tag{4.39}$$

Let us assume (4.39) is verified. At such a point, the maximum number of communications to be taken into account will be the maximum number of overlapping communications, which is an output of FDCS. However, for this, we have a domino effect which may let the number of communications to be bound huge.

Let us analyze the case proposed in Fig. 32. Here we want to bind com1. In order to do it, assuming that we have $\lambda_{com} = 2$, we want to respect (4.37). For this, we need to bind together all the communications within the red circle. However, we see that the same condition applies to all the other communications, generating the overmentioned domino effect. In our

computation case, we will assume $\lambda_{com} = 1$, which is a realistic parameter which, more, will satisfy (4.39). In this case, thanks to FDCS, we will have that the maximum number of overlapping communications ϕ will be

$$\phi = \left\lceil \frac{\max\{t[com(u_i, u_j)]\} - \min\{t[com(u_k, u_l)]\}}{e} \right\rceil \tag{4.40}$$

$\forall u_i, u_j, u_k, u_l \in DFG$. Hence, as this is a special case, the overmentioned domino effect does not take place. Usually such a number is very low and, thanks to this, computing all the dispositions between these will not bring to any significant overhead. An example of graph with 3 overlapping communications can be found in 33.

4.5.1 Improving the average computation time

We define *binding level* the group of binding solutions in which the same number of communication has already been bound.

Let us suppose we have ϕ overlapping communications. For the first communication to be bound, we have ϕ possibilities. In each of them there will still be $\phi - 1$ communications to be bound; so, at the second binding level, we will have $\phi \cdot (\phi - 1)$ possibilities. If we step to the last level which has to be the ϕ^{th} as we need to bind all ϕ communications, we will have $\phi \cdot (\phi - 1) \cdot ... \cdot 1$ possibilities, which is $\phi!$.

What we want to do now is trying to reduce this number of possibilities to be computed as much as possible. In order to do this, we can use the *detection of similar patterns heuristics* (Sec. 4.3), applied at each binding level: in this way, we are able to prune all similar solutions.

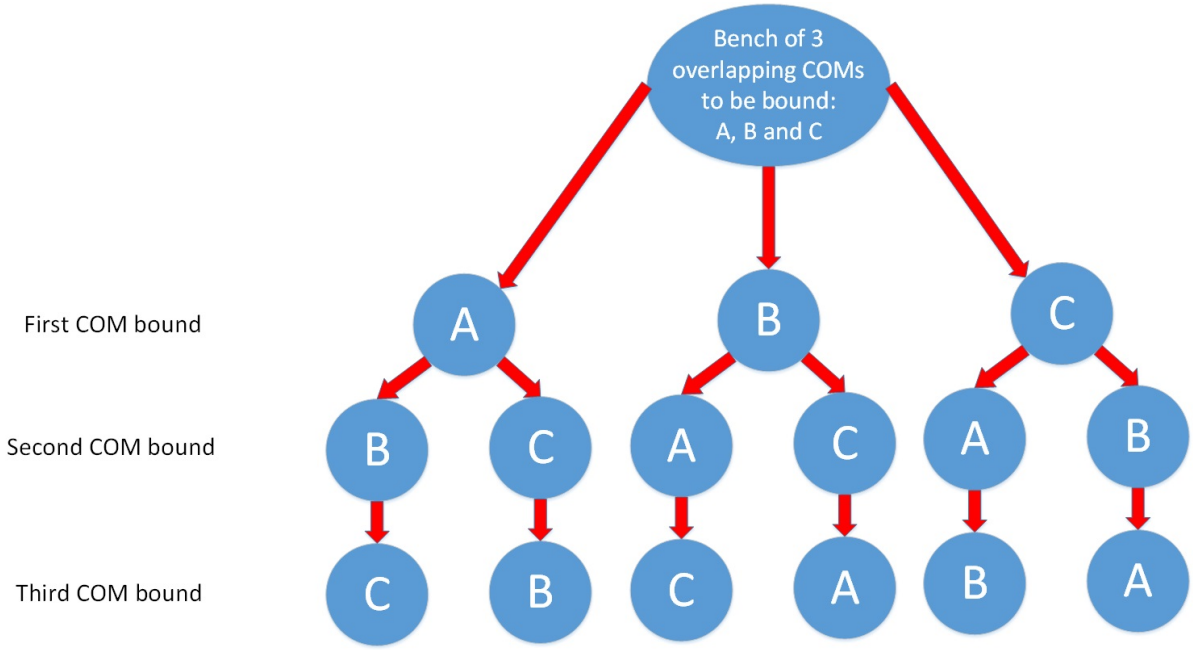

Figure 33: Analysis of possibilities for 3 overlapping communications

At this point, however, if we find solutions moving per binding levels (e.g. we first find solutions for first binding level, then for second...) we will not be able to erase any similar pattern, as we need to arrive at the last level in order to correctly explore solution space[1]. For this, in order to make us able to prune some solutions, we may apply here Depth-First Search algorithm: as we have a final result, we may be able to prune branches in the "possibilities tree" we have. Here a question comes naturally out: is this really necessary? Can we go in depth just for one

[1]at a certain level k we may prune a solution because worse than others for Δ_1WL but, in the next $\phi - k$ levels, while this pruned solution worsened of a factor Δ_2, all the solutions held at level k worsened of factor $\Delta_3 > \Delta_1 + \Delta_2$

solution and then, applying other algorithms, like Breadth-first search, prune all suboptimal solutions (referring to the first initially found)?

If we act this way, we have the disadvantage of having just one reference solution (until we do not reach leaves in our solutions graph): if we use DFS algorithm, we are constantly trying to reach leaves in our solution graph, having best solution been updated quicker than in BFS. In particular, here we are using a "left hand" algorithm.

The pseudocode is divided into three different procedures.

Simultaneous_COM_binding_caller will just find out which will be the next communications to be bound and it will pass both the current solution and the communications to be bound to *Simultaneous_COM_binder*, which will produce the best solution (in terms of wirelength).

Algorithm 9 Simultaneous com binding caller

```
procedure SIMULTANEOUS_COM_BINDING_CALLER
    root_soln = initialize_binding_soln()
    while not all coms bound do                          ▷ While there still is a com not bound
        coms_to_bind = lowest_sched_time_unbound_coms (coms)          ▷ Look for the set of unbound coms with the lowest scheduling time
        root_soln = Simultaneous_COM_binder(coms_to_bind, root_soln)          ▷ Call Simultaneous_COM_binder: it will return the best solution binding all the coms_to_bind
    return (root_soln)
```

Simultaneous_COM_binder is really the heart of the algorithm: it will analyze all the possible solutions binding communications in any possible order and will choose the best. It is a recursive

function. This also calls *solution_binder* which, taking as input a set of binding solutions and the com to be bound, it will produce the set of all possible binding solutions for the given inputs.

Algorithm 10 Simultaneous COM binder

procedure SIMULTANEOUS_COM_BINDER(coms_to_bind, binding_solns)
  if coms_to_bind = $\emptyset$ **then** ▷ Recursive function: if no more coms to be bound, then return just the best among the passed binding solutions
    return (best(binding_solns)) ▷ best=lowest wirelength
  for all $com_i \in coms\_to\_bind$ **do**
    intermediate_bindings = solution_binder(com_i, binding_solns)
    coms_list_next_level = coms_to_bind - com_i ▷ coms_list_next_level contains coms_to_bind excluding com_i
    binding_solns_harvest = Simultaneous_COM_binder(coms_list_next_level, intermediate_bindings) ▷ Recursive call: it will be passed all the computed solutions to this moment and the remaining coms to be bound
    best_binding_soln = best(binding_solns_harvest, best_binding_soln) ▷ best=lowest wirelength
  return (best_binding_soln)

4.5.2 Complexity

This algorithm adds an overhead to standard chronological binding ($O(ne^2)$). Such an overhead is given by the computation of all possible solutions for overlapping communications. If we say that the maximum number of overlapping communications is ϕ, usually much lower than e, and that $\gamma = 2^{\phi}$ (γ, in this case, represents the average computations for all the

Algorithm 11 Solution binder

procedure SOLUTION_BINDER(com_to_bind, binding_solns) ▷ Inputs are scheduling times (both coms and nodes), max buslet cardinality, max fanin and fanout for FUs and how many solutions to propagate each binding step. Finally, we can choose to use similar solution pruning enabling the proper flag.
  for all $S_j \in$ binding_solns **do**
    for all available buslets b_k, source FU_m, destination FU_n **do** ▷ any combination of these
      local = bind(S_j, com_to_bind, b_k, FU_m, FU_n) ▷ We perform binding, starting from solution S_j, for the quatern (COM_i, B_k, FU_m, FU_n)
      push(Local_solns, local) ▷ Add local to Local_solns
  return (Local_solns)

combinations, having first a reference solution to be compared with intermediates). The final complexity will be, this way, $O(\gamma n e^2)$.

4.6 Force directed binding

Here what we aim to present is an alternative to chronological binding. In fact, as chronological binding has evident benefits, unfortunately, shows also the non negligible drawback of having a too local view of the whole dataflow graph. If we focus just on binding a single communication per time (or, by extension, to a restricted set of them, considering overlapping communications or also taking into account lookahead mechanisms), we will not be able to take into account the whole DFG structure, seeing which will be the effect of our binding on the whole final solution. For this, it may be reasonable to use a force directed-like approach also for binding, with the aim of minimizing final wirelength.

Let us define a solution possibility q as a combination of four different elements:

- Communication $com(u_i, u_j)$ to be bound
- Buslet b_k to which $com(u_i, u_j)$ will be bound
- Functional unit r_l to which u_i will be bound
- Functional unit r_m to which u_j will be bound

If, at step n during execution of our algorithm, we have obtained a binding solution BS_{i-1}, the new binding solution BS_i will be obtained applying the chosen solution q_i to BS_{i-1}:

$$BS_i = BS_{i-1} + q_i \tag{4.41}$$

Now comes naturally the question: how to choose q_i, if we are not moving chronologically anymore?

4.6.1 Formulation of probability for a given solution

As we aim to minimize the overall wirelength of our design and our wirelength model deals with buslet cardinality (in particular, wirelength is directly proportional to current buslet cardinality), it is very likely for us to try to minimize it. It may sounds natural to elaborate a force using the probability of binding a given solution q_k is inversely proportional to the cardinality b_k will have after binding FU_i and FU_j to it.

According to the definition of probability, in order to compute the probability for a particular q_k, we need to have the future buslet cardinalities for any solution binding the communication

$com(u_i, u_j)$ taken into exam.

We can model, this way, the probability of binding the possibility $q(com_i, b_k, r_l, r_m)$ as

$$p[q(com_i, b_k, r_l, r_m)] = \frac{\frac{1}{1+\Delta WL(S_{n-1}, q(com_i, b_k, r_l, r_m))}}{\sum_{all\ q_s\ for\ com_i} p(q_s)} \tag{4.42}$$

where $\Delta WL(S_{n-1}, q(com_i, b_k, FU_m, FU_n))$ is the difference between the wirelength for $S_n = S_{n-1} + P_r$ and wirelength for S_{n-1}.

4.6.2 Formulation of weight function for a given solution

What we aim here to formulate is a factor determining a sort of distribution graph in which we may aim to minimize peaks. However, it is not a straightforward issue to determine which one could be the parameter we may apply such an issue.

The first idea we may have is to directly plug whole solution's wirelength into it. In fact, the highest such a parameter is for a given solution, the least we are likely to choose it. So, as first instance, we may decide to define weight function in our FDB algorithm as

$$w(q_k) = \Delta WL(S_{n-1}, q_k)$$

With these, we will compute the self force as

$$SF(q_k) = (1 - p(q_k)) * w(q_k) - \sum_{\forall q_s \neq q_k\ \ with\ \ com_i} p(q_s) * w(q_s) \tag{4.43}$$

After computing all the self forces, it will be chosen the binding solution with the lowest one.

4.6.3 Predecessor/successor forces

According to what said previously for the force directed-like approaches (FDS, our FDCS), after we take a decision we will have an effect on predecessors as well as on successors. Does such an issue apply also on force directed binding?

Let us assume we want to bind the quatern described in q_k which was defined in (4.34). Then, we will have, on all the other binding possibilities, the following effects:

1. $b_j \in q_k$ will not be anymore available $\forall com_h || t(com_h) - t(com_i)| < \lambda_{com}$
2. r_l will not be available $\forall u_h | k(u_k) = k(u_i), |t(u_h) - t(u_i)| < \lambda(k(u_i))$
3. r_m will not be available $\forall u_h | k(u_k) = k(u_j), |t(u_h) - t(u_j)| < \lambda(k(u_j))$
4. b_j may increase its cardinality
5. r_l and r_m may increase their own fanin/fanout

Let us analyze all these effects. For the first effect, as we can not have any conflict on the buslet, we have a direct effect on the forces for all the communications scheduled at the same time the currently bound communication is. About the availability of the the functional units, we will certainly have an effect on all the nodes in the DFG having a scheduling time which overlaps the currently marked as busy clock cycles for the current functional units. This will certainly affect the force for all the communications between nodes having such a type. These are the effects not taking into account the constrains on the buslet cardinality and on functional unit fanin/fanout.

Analyzing the buslet variation of cardinality, we will have an effect on the force for all the

communications which can be bound to buslet b_j. Unfortunately, this means that we will have an effect on all the forces for our DFG and, for this, we need to analyze all of them. Further, the effect on fanin/fanout will affect all the operation nodes of the same type as the currently bound ones through the whole design. Synthesizing, we need to analyze the variation for all the forces in the design.

4.6.4 Efficacy of this algorithm

Such an algorithm will show a very peculiar behavior. For its formulation, it will take a communication between two operation nodes, bind it and, then, keep on binding other communications between the same kind of functional units as much as possible. So we can say that, after binding the quatern q_k, it will bind all the other quaterns q_h such that

- It will not create conflicts on the buslet $b_j \in q_k$.
- Functional units r_l and r_m will not be busy.

This happens because the distribution graph will make the force dropping for these (as it will use an already available structure, the value for it is zero). According to this, any effect on predecessors/successors appears being unuseful (as all this contribution is ignored from the distribution graph structure). However, this may lead to suboptimalities. A similar example was already discussed in Sec. 4.5. In that case, the self force to bind com(B, A) will be the lowest as an interconnection is already present. This, as seen, will drive the solution to be suboptimal.

For this, we expect such a technique not to produce the best absolute result but, as it globally

looks at all the communications, not producing the worst results either.

Down here it is presented a very synthetic version of FDB which, however, illustrates all main issues to be tackled.

Algorithm 12 FDB algorithm

procedure FORCE_DIRECTED_BINDING(sched(COM, NODES))
    root_soln = initialize_binding_soln()
    while not all coms bound **do**
        prev_wl = wirelength (root_soln) ▷ Wirelength before next binding step
        for $com_i \in coms\_to\_bind$ **do**
            for all available buslets b_k, source FU_m, destination FU_n **do** ▷ Any combination of these
                wl_after[com_i][b_k][FU_m][FU_n] = wirelength(bind(root_soln, com_i, b_k, FU_m, FU_n))▷ With this we can identify a 4D variable. The same will apply to weight, probability and SF
                weight[com_i][b_k][FU_m][FU_n] = wl_after[com_i][b_k][FU_m][FU_n] - prev_wl
            for all available buslets b_k, source FU_m, destination FU_n **do**
                compute probability[com_i][b_k][FU_m][FU_n] ▷ See 4.42
            for all available buslets b_k, source FU_m, destination FU_n **do**
                compute SF[com_i][b_k][FU_m][FU_n] ▷ See 4.43
        lowest_SF = ∞
        for $com_i \in coms\_to\_bind$ **do**
            for all available buslets b_k, source FU_m, destination FU_n **do**
                if SF[com_i][b_k][FU_m][FU_n] < lowest_SF **then** ▷ Look for the lowest SF
                    lowest_SF = SF[com_i][b_k][FU_m][FU_n]
                    best_bind = (com_i, b_k, FU_m, FU_n)
        root_soln = bind(root_soln, best_bind)
    return (root_soln)

4.6.5 Complexity

Here the complexity is comparable to that already computed for FDS algorithm, as the basic technique is the same. However, in FDS the solution space to be explored was n only. Here it will be $O(n^2e^2)$ (complexity coming from the quatern of values composing each binding possibility). For this, the total complexity of such a technique without predecessors and successors will be $O(n^4e^4)$. However, if we want to also add predecessors and successors contribution, the total expected complexity will be $O(n^6e^6)$.

CHAPTER 5

BUSLET POWER MODELING

Once we perform scheduling and binding, we will have, as output, a bench of functional units and interconnections between them. Furthermore, we also have the correct sequence for both data transmissions (performed using buslets) and data computation (performed in the functional units). Having these data, commonly outputted after the high level synthesis step in VLSI CAD (Sec. 3.1), we need to go through the rest of the CAD flow. Let us assume we will jump straight to physical design (floorplanning). In order to estimate the power consumed using buslets and to compare it with dedicated interconnections, some modeling is necessary. Indeed, we can use some technique to minimize all the metrics involved in wiring power consumption. All the models here presented assume floorplanning stage has already been performed: for this, all the FUs are already placed on chip.

5.1 Metrics involved in buslet power consumption

Let us assume a generic buslet b_i connects $|b_i|$ functional units (this is its cardinality). Further, let us assume that the two functional units $FU_j, FU_k \in b_i$. Whenever a communication between these two functional units has to be performed, two main factors contribute to the power consumption (in this case, this is *dynamic power*):

1. Wire length to be driven. The wire has its own capacitance and it has to be driven. From this, we can say that, the lowest this is, the lowest this power contribution is. An idea could be "slicing" the circuit such that just the minimum wirelength will be driven.

2. Input buffer capacitance. Each FU will have, as input, a buffer. This has a non negligible input capacitance to be driven. Furthermore, if we wish to insert some buffers in the middle of the circuit, this contribution depends on the number of buffers met. This goes against the statement evidenced in the previous point and, for this, some trade-off is necessary.

Furthermore, we have also a static power contribution, coming from the buffers in the design, due to leakage current. All of these factors need to be taken into account while designing the buslet structure itself.

5.2 Minimum spanning tree

The first approach we can try to use is creating a minimum spanning tree (MST), connecting all the functional units. Let us build a completely connected[1] undirected[2] graph G, having as nodes all the functional units connected to the same buslet b_j. Let us associate to each edge $e(FU_i, FU_j)$ connecting $FU_i, FU_j \in G$ the weight $w[e(FU_i, FU_j)]$, which will be the Manhattan distance between the two functional units. Now, we will use an algorithm allowing us to have

---

[1]We define *completely connected* a graph G having n nodes, each of them directly connected to all the other $n-1$ nodes. Such a graph will have $\frac{n^2-n}{2}$ edges in total.

[2]We define a graph G being *undirected* when, for each couple of nodes $u_i, u_j \in G$, if exists an edge connecting u_i to u_j, then it will also exist an edge connecting u_j to u_i.

a connected graph[1] with the least total minimum wirelength. We define it being a *minimum spanning tree*. Here follows a quick description of two standard algorithms used to solve this problem.

5.2.1 Prim's algorithm

This algorithm was designed by Robert Prim in 1957[23]. Prim's algorithm uses a greedy approach that computes an MST in a weighted undirected graph. So, it will look for all the edges which will create a tree including each and every node, in which the total of the edges' weights is the minimum. The algorithm builds such a tree one node at a time, from an arbitrary chosen one, and each step it will add the cheapest possible connection from such a built the tree to any other non connected node.

The algorithm, however, may be modified to start with any particular vertex. The total complexity of this algorithm is $O(n^2)$.

5.2.2 Kruskal's algorithm

This algorithm was designed by Joseph Kruskal 1956[24]. Kruskal's algorithm is another minimum-spanning-tree algorithm.

It will work finding, each step, the cheapest edge which will connect any two trees in the forest. Also this uses a greedy approach and it will compute the MST starting from a connected graph. Hence, it will find the edges, which will form a tree, which will include all the nodes, minimizing the total of the edges' weights.

---

[1]We define a graph G being *connected* if, $\forall u_i, u_j \in G$, exists a path connecting them.

Algorithm 13 Prim algorithm

```
procedure PRIM(G)                          ▷ G is the input graph, containing both nodes and edges
    Q = ∅
    F = ∅                                  ▷ F will be our minimum spanning tree
    for each node u_i ∈ G do
        C[u_i] = min_{∀u_j ∈ P(u_i)}{w[e(u_i, u_j)]}
        Q = Q + u_i                        ▷ When initialized Q contains all the nodes in G
    while Q not empty do
        u_best = first(Q)
        for all u_i ∈ Q do
            if C[u_i] < u_best then
                u_best = u_i               ▷ Here we will select the edge with the lowest cost in Q
        F = F + u_best
        Q = Q − u_best                     ▷ Here we will exclude u_best from Q
    return (F)                             ▷ We return the MST
```

The complexity of this algorithm is $O(e \log(e))$: this means that this algorithm is better than Prim's if the number of arcs from the starting graph is not high. Hence, if the graph is fully connected, Prim runs in lower time.

5.3 Standard buffer placement

According to the common structure for any bus, through the line we will not have any buffer. The signal, in this way, will be propagated through the whole wirelength of the buslet, reaching all the functional units connected to it. Of course, we will minimize their number according to the fact that a given FU will need to read or to write only data from/to the buslet. This is completely deterministic as it is a known data from any of our HLS algorithms. For example, if a given functional unit $FU_j \in b_j$ will drive the buslet never reading it, we will insert the output buffer only. This will improve both static and dynamic power, reducing also signal delay. We

Algorithm 14 Kruskal algorithm

```
procedure KRUSKAL(G)                    ▷ G is the input graph, containing both nodes and edges
    Q = ∅
    F = ∅
    for each edge e_i ∈ G do
        Q = Q + e_i                     ▷ When initialized Q contains all the edges in G
    Sort_ascending(Q)                   ▷ Sort all the edges in weight ascending order
    while Q not empty do
        e_best = first(Q)
        F = F + u_i + u_j
        for all e_i ∈ Q do
            if u_i ∈ F and u_j ∈ F then
                Q = Q − e_i
    return (F)                          ▷ We return the MST
```

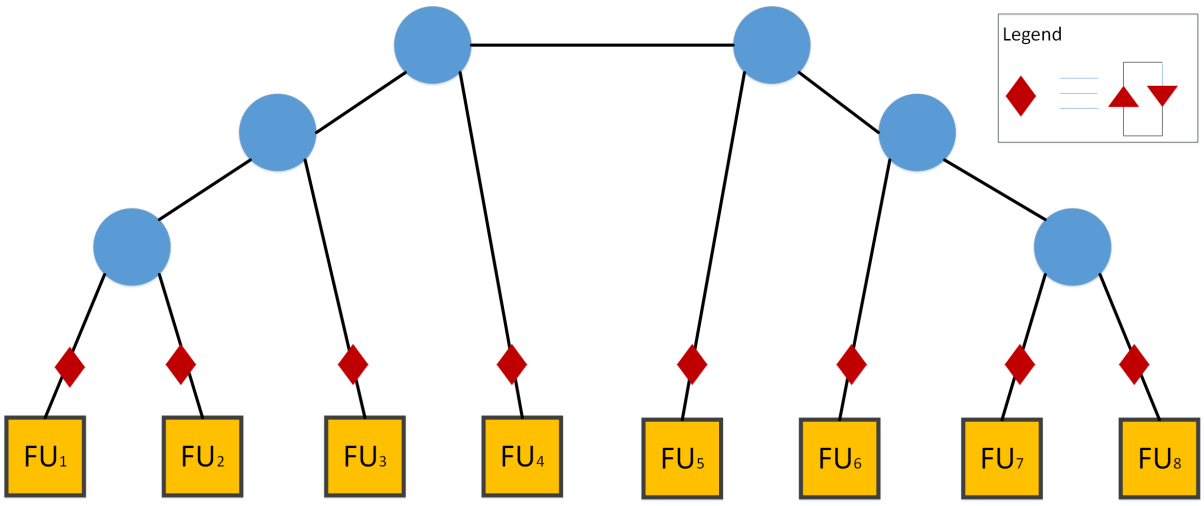

Figure 34: Example of standard buffer placement model for buslet of maximum cardinality 8. FUs are the squares, diamonds are bidirectional buffers (connected as shown in the legend) and blue nodes are steiner points.

have benefits on the dynamic power because we will decrease the total buslet capacitance to be driven. We know that, for estimating dynamic power, we have two contributions. In this case, the contribution from the wirelength C_w is fixed for any communication happening and it is

$$C_w(b_j) = WL(b_j) \cdot C_{F/m} \tag{5.1}$$

where $WL(b_j)$ is the total wirelength for b_j and $C_{F/m}$ is characteristic line capacitance per length unit. The contribution from the buffers, instead, is

$$C_{tri}(b_j) = C_{tri-in} \sum_{\forall FU_i \in b_j} \delta_{fanin}(FU_i, b_j) \tag{5.2}$$

where

$$\delta_{fanin}(FU_i, b_j) = \begin{cases} 1 & if\ FU_i\ reads\ data\ from\ b_j \\ 0 & otherwise \end{cases} \tag{5.3}$$

and C_{tri-in} is typical input capacitance for tristate buffers. For this reason, minimizing fanout buffers will have here a contribution on saving static power only, but fanin buffers will also save static power. An example of such an interconnection type is shown in Fig. 34

To wrap-up, the final dynamic power for a single buslet in the design b_j will be

$$P_{dyn}(b_j) = f_w \cdot V_{dd}^2 \cdot \frac{COMS(b_j)}{\overline{\lambda}} (C_w(b_j) + C_{tri}(b_j) + 1) \tag{5.4}$$

Algorithm 15 Standard MST power model

```
procedure MST-POW(G, D, λ̄)
    tot_pow = 0
    for all buslet b_j ∈ D do
        F = Prim(G)
        WL = total_length(G)
        CFIN = total_fanin_buffers(b_j)
        this_buf_pow = P_dyn(b_j)                    ▷ It is (5.4)
        this_buf_pow = this_buf_pow + static_power
        tot_pow = tot_pow + this_buf_pow
```

5.3.1 Complexity

In order to build the MST we use here Prim's algorithm ($O(n^2)$). Then, in order to decide the buffer placement, we will take another $O(n)$ time and, finally, to compute the total static and dynamic power we just need to know their quantity ($O(1)$). The most dominant part is here the creation of the MST and, for this, the total final complexity is $O(n^2)$.

5.4 MST-2B

The main problem with standard buffer placement is that we have to drive all the buslet for any communication, even if two functional units are each other close.

We can see an example of such a behavior in Fig. 35. In order to avoid this, what we can do is literally "slicing" the buslet, using tristate buffers which may enable/disable the propagation of the signal according to the target functional unit position in the MST.

Let us call *routing nodes* all the nodes in the MST which are not FUs. Each wire $w(u_i, u_j)$ in the MST connecting two nodes which are both routing nodes will have, at both its ends, a

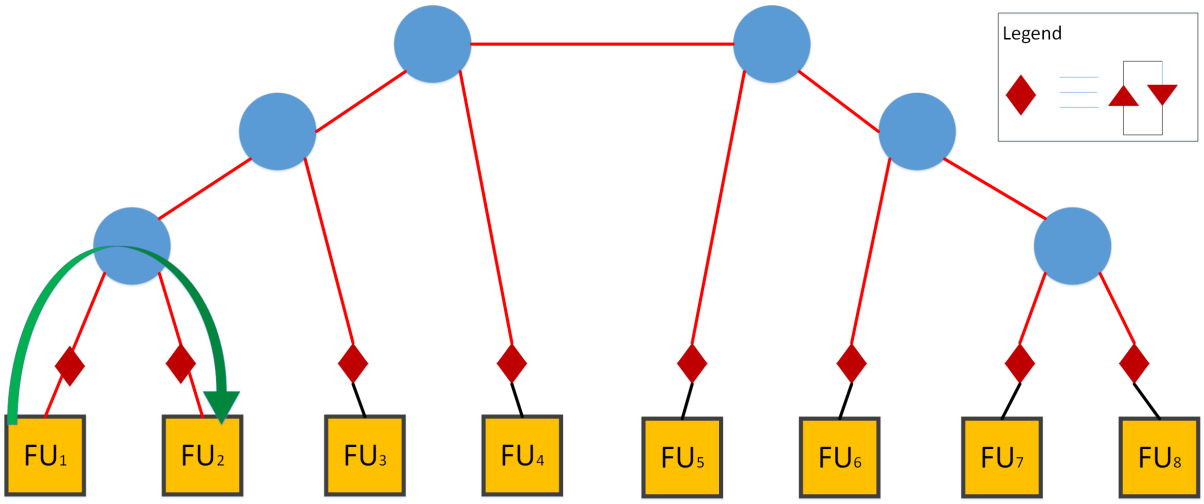

Figure 35: Example of communication between FUs in the case of standard interconnections. Here, even if the two FUs needing to communicate are close (in this case, FU_1 is sending data to FU_2, the signal will propagate through all the buslet (it is evidenced in red).)

bidirectional tristate buffer[1]. Using these, it is possible to minimize the total wirelength to be driven. We can see this implementation in Fig. 37. In that case, just the minimum wirelength will be driven.

However, such an approach has some drawbacks:

- When the signal reaches routing node R_k, the fanin input capacitance at that particular node will be not only that of the next wire it will cross, but also the sum of the input capacitances of all the other wires connected to R_k. An example of this is in Fig. 36.

[1]We define *bidirectional tristate buffer* the pairing of two tristate buffers in parallel, oriented in the two opposite directions.

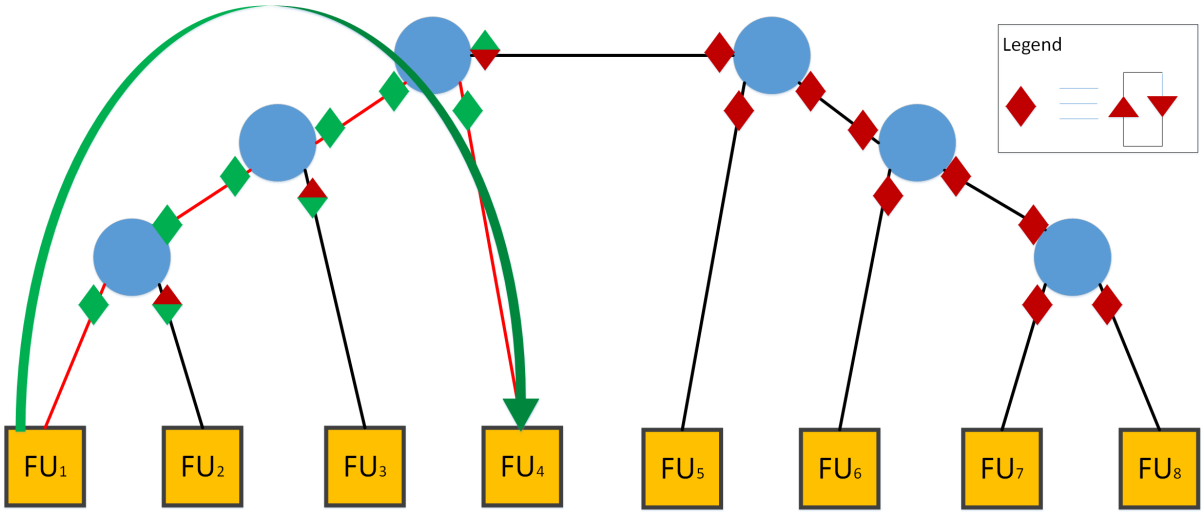

Figure 36: Example of communication between FU_1 and FU_4 in the MST-2B model. In this case, even if the wirelength is the lowest possible (in red), the number of input capacitances to be loaded (evidenced in green) is very high, much greater than in the standard implementation.

Let us call the extra capacitance to be driven (that not belonging to the path the signal will follow) $C_{fin-extra}(R_k)$. It will be

$$C_{fin-extra}(R_k) = C_{tri-in} \cdot \left(\sum_{\forall w_j \ connected \ to \ R_k} \delta_{wirefanin}(w_j, R_k) - 1 \right) \tag{5.5}$$

where

$$\delta_{wirefanin}(w_j, R_k) = \begin{cases} 1 & if \ w_j \ reads \ data \ from \ R_k \\ 0 & otherwise \end{cases} \tag{5.6}$$

- For all the wire slices crossed by the signal, we will have a contribution from the fanin tristate buffers. Let us call such a contribution $C_{fin-route-2B}$. Let us also assume we have the route Y which is a bunch of wires w_j connecting routing nodes only. We can say that wire $w(R_k, R_l)$ will connect routing nodes R_k and R_l. From this, the contribution of $C_{fin-route-2B}(Y)$ can be expressed as

$$C_{fin-route-2B}(Y) = \sum_{\forall w_j(R_k,R_l)\in Y} 2 + \delta_{wirefanout}(w_j, R_k) + \delta_{wirefanin}(w_j, R_l) \tag{5.7}$$

where $\delta_{wirefanin}(w_j, R_l)$ was already defined in (5.6) while

$$\delta_{wirefanout}(w_j, R_k) = \begin{cases} 1 & if\ w_j\ writes\ data\ to\ R_k \\ 0 & otherwise \end{cases} \tag{5.8}$$

Always referring to Fig. 36, when performing com(FU_1, FU_4), input capacitances for FU_2, FU_3 and another wire connecting steiner points are loaded.

So, such an approach will minimize capacitance from wire with a higher cost coming from tristate buffers.

5.4.1 Shortest path problem: Dijkstra's algorithm

Dijkstra's algorithm is used in a graph, directed or undirected, to find the shortest path between two nodes. It was designed by Edsger W. Dijkstra in 1959[25].

From the source node, it will iteratively look for the lowest cost edge, and it will bring at the destination node the sum of the crossed edges. It will end when it will reach the destination

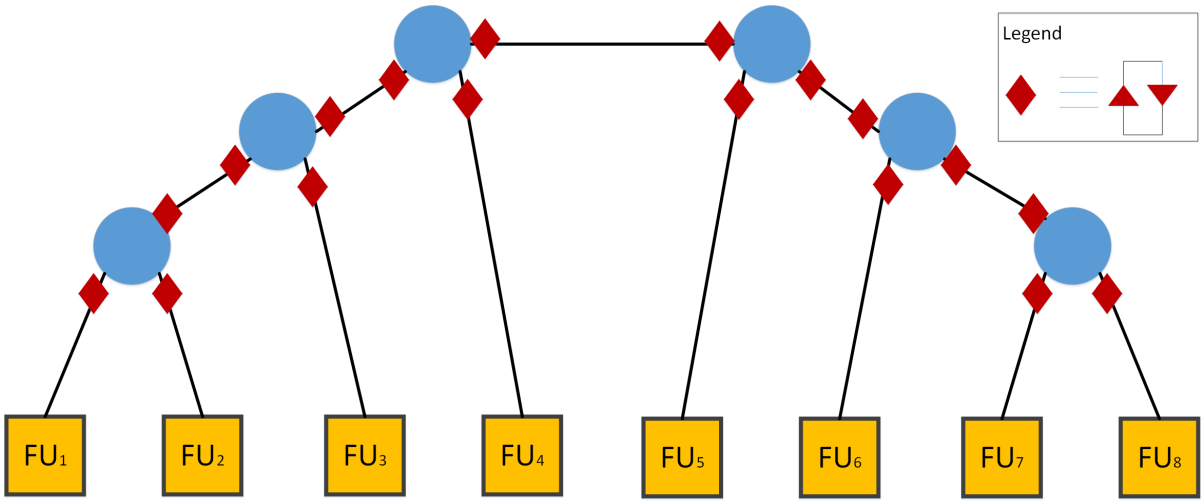

Figure 37: Example of implementation of MST-2B.

node.

The total complexity of this algorithm is $O(n^2)$, but there are some other implementations making this $O(e + n\log(n))$.

5.4.2 Complexity

The total complexity of such an algorithm is tightly linked to the initial computation of the MST (Prim's algorithm, $O(n^2)$), computation of the dynamic power (number of communications, e, times shortest path computation, $O((n+r)^2)$, where r is the number of routing nodes) and buffer placement ($O(w)$, with w number of wires, because each wire will need to handle its own buffers, depending on read/write operations). In total, our complexity will be $O(n^2) + O((n+r)^2) + O(w)$. As $O(n^2) \leq O((n+r)^2)$, we can say that the complexity is

Algorithm 16 Dijkstra algorithm

procedure DIJKSTRA(G, s, d)▷ G is the graph, s is the source node and d is the destination
$S = \emptyset$
for all $u_i \in G$ **do**
if $u_i \neq s$ **then**
$d(u_i) = \infty$ ▷ The distance between s and u_i is unknown
$p(u_i) = \infty$ ▷ The minimum distance node between u_i predecessors and s is unknown
$S = S + u_i$
while $S \neq \emptyset$ **do**
n = node with min d
$S = S - n$
for all u_j predecessors and successors of n **do**
$temp = d(n) + e(u_j, n)$
if $temp < d(u_j)$ **then**
$d(u_j) = temp$
$p(u_j) = n$

Algorithm 17 MST with two buffers per line slice power model

procedure MST-2B(G, D, $\overline{\lambda}$)
tot_pow = 0
for all buslet $b_j \in D$ **do**
F = Prim(G)
this_buf_pow = 0
for all $c_i \in CS(b_j)$ **do** ▷ For all communications scheduled (CS) in buslet b_j
R = Dijkstra(G, u_i, u_j)
CFIN = $C_{fin-extra}(R, G) + C_{fin-route-2B}(R)$ ▷ $C_{fin-extra}(R, G)$ defined in (5.5) while $C_{fin-route-2B}(R)$ defined in (5.7)
$this\_buf\_pow = P_{dyn}(b_j)$▷ It is (5.4), using as capacitance $CFIN$ and as distance $d(R)$
this_buf_pow = this_buf_pow + static_power
tot_pow = tot_pow + this_buf_pow

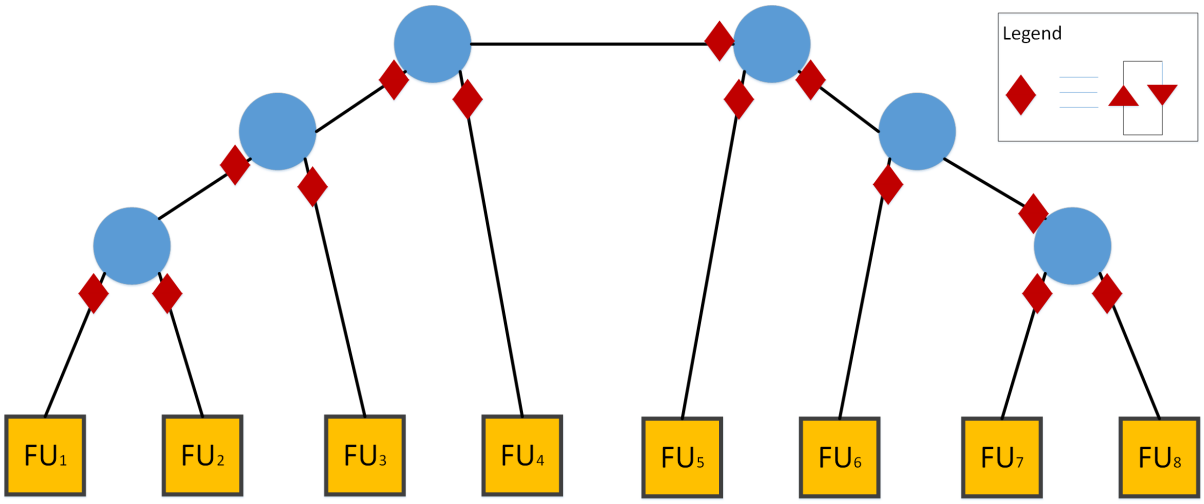

Figure 38: Example of implementation of MST-1B.

$O((n+r)^2+w)$. However, as we are in a minimum spanning tree, we also know that $w = n+r-1$. For this, we rewrite the complexity as $O((n+r)^2+n+r-1)$ and, finally, $O((n+r)^2)$.

5.5 MST-1B

In MST-2B, we are routing signals through the shortest path of our MST. This is performed completely isolating this path from the rest of the buslet, thanks to the use of tristate buffers at both the ends of each wire. However, we can reach such a result using a couple of tristate buffers only.

A structure example of this can be found in Fig. 38. MST-1B differs from MST-2B exactly from this. In particular, it has the big advantage of halving $C_{fin-route}$ contribution, as it will be met one couple of tristate buffers per wire only.

Let us try to perform the same communication as in Fig. 36 (MST-2B). With this implemen-

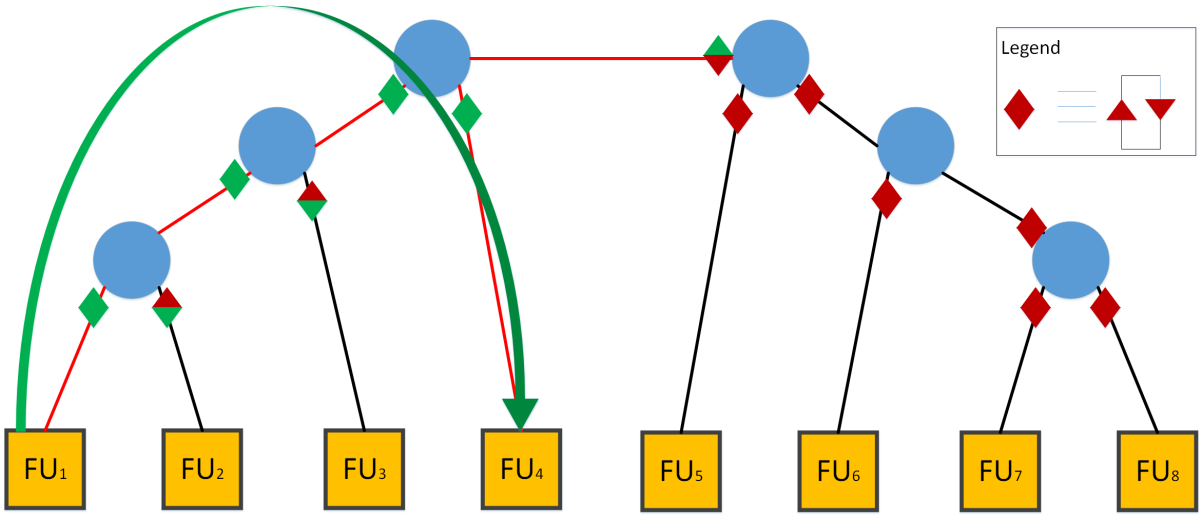

Figure 39: Example of communication between FU_1 and FU_4 in the MST-1B model. In this case, even if the wirelength is the lowest possible (in red), the number of input capacitances to be loaded (evidenced in green) is very high, much greater than in the standard implementation.

tation, the number of the buffers crossed almost halves. We find this in Fig. 39.

Assuming route Y, the contribution of $C_{fin-route-1B}(Y)$ can be expressed as

$$C_{fin-route-1B}(Y) = \sum_{\forall w_j (R_k, R_l) \in Y} 1 + \delta_{wirefanin}(w_j, R_l) \tag{5.9}$$

where $\delta_{wirefanin}(w_j, R_l)$ was already defined in (5.6).

The other side of the coin of using this technique, however, is that the signal will not propagate through the shortest path only. If we look at Fig. 39, indeed, we see that the signal, for a small piece of line, will overpropagate.

5.5.1 Placing tristate buffers

Here comes naturally a question: where it is more convenient to place the pair of buffers? It depends on the communications flow itself. Let us assume we are taking this decision on wire $w(R_k, R_l)$. Here, we can determine the *communication rate*, defined as the ratio between the number of communications between two nodes over the total latency

$$r(R_k, R_l) = \frac{\#\ coms\ from\ R_k\ to\ R_l}{\overline{\lambda}} \tag{5.10}$$

Note: $r(R_k, R_l) \neq r(R_l, R_k)$. Let us assume we place the buffer at distance x from R_k. In this case, the total dynamic power consumed by $w(R_k, R_l)$ will be

$$\begin{aligned} P_{wiredyn}[w(R_k, R_l)] = \quad & [r(R_k, R_l) + r(R_l, R_k)] \cdot C_{F/m} \cdot d[w(R_k, R_l)] + \\ & [\eta_1 - r(R_k, R_l)] \cdot C_{F/m} \cdot x + \\ & [\eta_2 - r(R_l, R_k)] \cdot C_{F/m} \cdot \{d[w(R_k, R_l)] - x\} \end{aligned} \tag{5.11}$$

where η_1 is defined as

$$\eta_1 = \sum_{\forall n_i \in U(R_k, R_l)} r(n_i, R_k) \tag{5.12}$$

where $U(R_k, R_l)$ is the set of all neighbors nodes of R_k excluding R_l, and η_2 is similary defined. The contribution depending on x is $[\eta_1 - r(R_k, R_l)] \cdot C_{F/m} \cdot x + [\eta_2 - r(R_l, R_k)] \cdot C_{F/m} \cdot \{d[w(R_k, R_l)] - x\}$. In order to minimize the power, we have two possibilities:

$$x = \begin{cases} 0 & if r(R_k, R_l) - r(R_l, R_k) + \eta_1 - \eta_2 \geq 0 \\ d[w(R_k, R_l)] & otherwise \end{cases} \tag{5.13}$$

Furthermore, thanks to this analysis, we can accurately evaluate the total power consumed by each slice of the buslet. This makes the total dynamic power estimation easier.

When we have to compute the contribution of the buffer input capacitances to the dynamic power, we have to distinguish the two cases. Let us say that C_1 is the buffer oriented from R_k to R_l and C_2 is oriented in the opposite direction. Both will be loaded when signal will entirely cross the wire (in both directions), while just one will be loaded when neighbor nodes will communicate. According to this, we can state that

$$\begin{cases} P_{C_1 dyn} = [r(R_k, R_l) + r(R_l, R_k) + \eta_1] \cdot C_{tri-in} \\ P_{C_2 dyn} = [r(R_k, R_l) + r(R_l, R_k) + \eta_2] \cdot C_{tri-in} \end{cases} \tag{5.14}$$

5.5.2 Complexity

The complexity of this algorithm is the same as MST-2B adding the overhead of taking the decision of where to place the buffer. This, in particular, analyzes the neighbor nodes ($O(n+r)$)

Algorithm 18 MST with one buffer per line slice power model

procedure MST-1B(G, D, $\overline{\lambda}$)
  tot_pow = 0
  for all buslet $b_j \in D$ **do**
    F = Prim(G)
    this_buf_pow = 0
    for all $c_i \in CS(b_j)$ **do** ▷ For all communications scheduled (CS) in buslet b_j
      R = Dijkstra(G, u_i, u_j)
      track(R, G) ▷ Mark each wire composing R as crossed in G
    for all edges $e_i \in G$ **do** ▷ G is composed by edges, already owning the number of times being crossed
      Place_buffers(e_i) ▷ According to (5.13)
      Compute $wire\_dpow(e_i)$ ▷ (5.11)
      Compute $buf\_dpow(e_i)$
      $this\_buf\_pow = wire\_dpow(e_i) + buf\_dpow(e_i) + static\_buf(e_i)$
      this_buf_pow = this_buf_pow + static_power
    tot_pow = tot_pow + this_buf_pow

per each edge ($O(n + r - 1)$), and adding an overhead $O((n + r)^2)$. However, complexity for MST-2B was already $O((n + r)^2)$; so, MST-1B complexity is $O((n + r)^2)$.

5.6 MST-LOGD

All the previous approaches (Standard, MST-2B and MST-1B) are approaches based on minimum spanning tree. This means that, as they are built, they ensure the lowest possible wirelength for the buslet. However, this may lead to situations in which, for a given communication, the total number of nodes to be crossed may be very high. In the worst case, we may need to cross the whole buslet, making power consumption (for MST-2B and MST-1B) higher than in the standard approach (this is due to the extra buffers inserted in the line).

In order to ensure this, we may fix a maximum graph distance between, let us say, a root node,

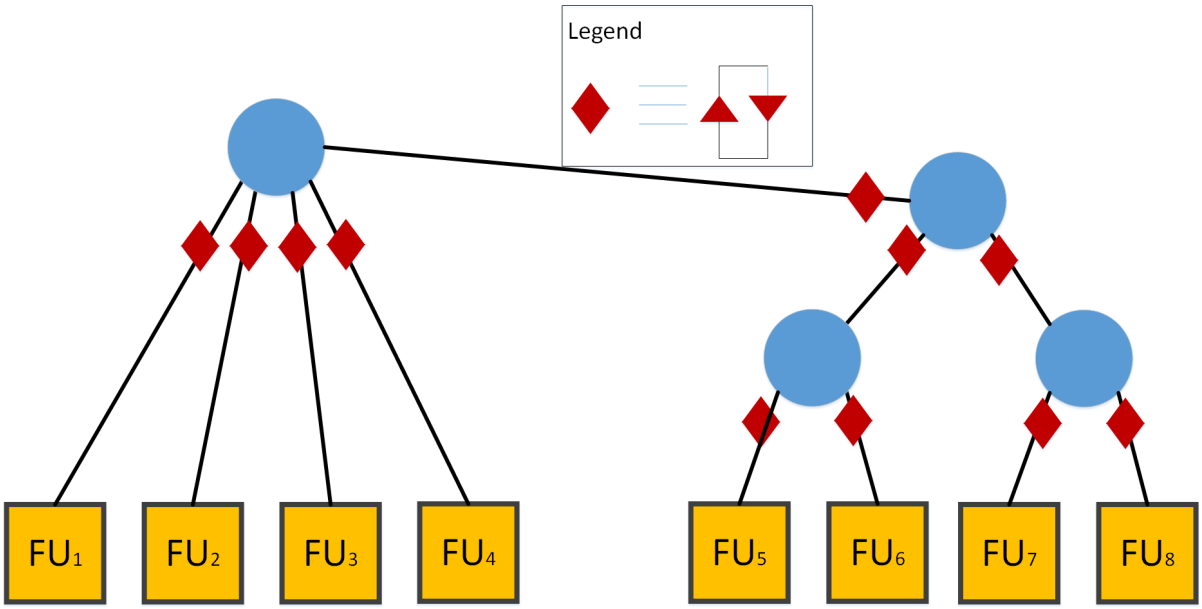

Figure 40: Example of implementation of MST-LOGD.

and any other node in the buslet. Let us define *hop distance* h_d as the distance, expressed as number of nodes, between two nodes in a graph.

If we impose this to be at most $\log_2(n)$ from a root node, which can be randomly chosen, then we are, at the same time, ensuring that the maximum distance between any two nodes in such a graph will be, at most, $2\log_2(n) + 1$. As we will build our MST starting from a fully connected graph, such a solution will always exist and will have, as result, a more compact buslet structure. What we need to do is just modifying Prim algorithm adding as constraint the maximum distance from a root node. For any other characteristic, this will be just an evolution of MST-1D.

Algorithm 19 Prim algorithm with max hop distance constraint

procedure PRIM_LOGMOD(G, D)
  $Q = \emptyset$
  $F = \emptyset$
  for each node $u_i \in G$ **do**
    $C[u_i] = \min_{\forall u_j \in P(u_i)}\{w[e(u_i, u_j)]\}$
    $Q = Q + u_i$ ▷ When initialized Q contains all the nodes in G
  $root = \emptyset$
  while Q not empty **do**
    $u_{best} = first(Q)$
    for all $u_i \in Q$ **do**
      if $C[u_i] < u_{best}$ **then**
        if $hops(u_i, root) \leq D$ **then** ▷ If the number of hops separating u_i and $root$ respects the constraint
          $u_{best} = u_i$
    if $root = \emptyset$ **then**
      $root = u_{best}$
    $F = F + u_{best}$
    $Q = Q - u_{best}$

5.6.1 Complexity

We are just adding an overhead to the computation of the MST obtained satisfying logarithmic distance from root node. For this, each time we have to test the distance between the potentially new node and the root. However, as all the nodes, once added, will store the information of their own distance from the root, such a test will be $O(1)$, non affecting the overall complexity, remaining, as in the case of MST-1D, $O((n + r)^2)$.

5.7 MST-BF

The problem with MST-LOGD is that we do not have any constraint on the maximum number of wires connected to a single node. In this way, we may potentially have high undesired

Algorithm 20 MST with one buffer per line slice power model and max log distance

procedure MST-LOGD(G, D, $\overline{\lambda}$)
  tot_pow = 0
  for all buslet $b_j \in D$ **do**
    F = Prim_logmod(G, D)
    this_buf_pow = 0
    for all $c_i \in CS(b_j)$ **do** ▷ For all communications scheduled (CS) in buslet b_j
      R = Dijkstra(G, u_i, u_j)
      track(R, G) ▷ Mark each wire composing R as crossed in G
    for all edges $e_i \in G$ **do** ▷ G is composed by edges, already owning the number of times being crossed
      Place_buffers(e_i) ▷ According to (5.13)
      Compute $wire\_dpow(e_i)$ ▷ (5.11)
      Compute $buf\_dpow(e_i)$
      $this\_buf\_pow = wire\_dpow(e_i) + buf\_dpow(e_i) + static\_buf(e_i)$
      this_buf_pow = this_buf_pow + static_power
    tot_pow = tot_pow + this_buf_pow

capacitances to be loaded. We want to also impose a constraint on this. Let us call *branching factor* the maximum number of wires connected to a single node and let us say it to be f. Each node will have at most f wires connected to it, limiting the number of extra undesired capacitances to be loaded.

5.7.1 Complexity

We are just adding an overhead for the introduction of the branching factor. However, such a test will be $O(1)$ only. The overall complexity, in this way, will keep on being $O((n+r)^2)$.

5.8 MAX-D

Let us, now, focus on the minimization of the longest path between FUs. Such a kind of optimization makes sense because we may want to minimize as much as possible wire dynamic

Algorithm 21 Prim algorithm with max branch factor constraint

```
procedure PRIM_MAXBRANCHFACTOR(G, B)
    Q = ∅
    F = ∅
    B = ∅
    for each node u_i ∈ G do
        C[u_i] = min_{∀u_j ∈ P(u_i)}{w[e(u_i, u_j)]}
        B[u_i] = 0
        Q = Q + u_i                          ▷ When initialized Q contains all the nodes in G
    while Q not empty do
        u_best = first(Q)
        for all u_i ∈ Q do
            if C[u_i] < u_best then
                acceptable = true
                for all neighbors nodes u_j of u_i do
                    if B[u_j] ≥ B then       ▷ If the number of branches of u_i does not meet the
constraint
                        acceptable = false
                    if acceptable = true then
                        u_best = u_i
        for all neighbors nodes u_j of u_best do
            B[u_j] = B[u_j] + 1
        B[u_best] = B[u_best] + 1
        F = F + u_best
        Q = Q − u_best
```

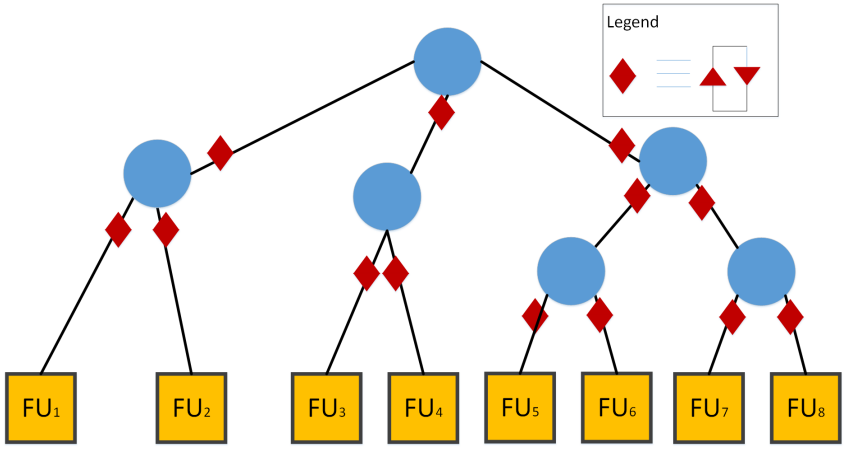

Figure 41: Example of implementation of MST-BF with maximum branching factor allowed 3.

Algorithm 22 MST with one buffer per line slice power model and max branching factor

procedure MST-LOGD(G, B, $\overline{\lambda}$)
  tot_pow = 0
  for all buslet $b_j \in D$ **do**
    F = Prim_maxbranchfactor(G, B)
    this_buf_pow = 0
    for all $c_i \in CS(b_j)$ **do** ▷ For all communications scheduled (CS) in buslet b_j
      R = Dijkstra(G, u_i, u_j)
      track(R, G) ▷ Mark each wire composing R as crossed in G
    for all edges $e_i \in G$ **do** ▷ G is composed by edges, already owning the number of times being crossed
      Place_buffers(e_i) ▷ According to (5.13)
      Compute $wire\_dpow(e_i)$ ▷ (5.11)
      Compute $buf\_dpow(e_i)$
      $this\_buf\_pow = wire\_dpow(e_i) + buf\_dpow(e_i) + static\_buf(e_i)$
      this_buf_pow = this_buf_pow + static_power
    tot_pow = tot_pow + this_buf_pow

power consumption. In order to do this, what we have to do is to fix an upper bound for such a distance and each other connect the other Functional units accordingly. In order to do this, we can first connect the two most distant FUs and use their distance as upper bound. Then, we can connect the other FUs to the so obtained wire through the closest Hannan point[1]. However, it may happen that some FUs can not be directly connected to it. At this point, they are connected to the closest Hannan point (distance computed as Manhattan distance). Then, there may be FUs that, even if connected in such a way, can not respect the maximum distance imposed. If this happens, then they are connected at the end, such that the maximum distance will be minimized.

Finally, as the first two FUs can be connected in a number of ways, as shown by the dashed lines in Fig. 42, a number of interconnections are attempted and we will hold just the best (i.e. that having the least maximum distance).

5.8.1 Complexity

This technique first looks for the two most distant FUs ($O(\frac{n^2}{2})$). Then, for each FU, it will compute Hannan points ($O(n)$) and check if max distance constraint is respected ($O(n)$, because just one path exists between any connected FU). Finally, All the FUs not respecting such a constraint will be connected to the closest Hannan point ($O(n^2)$). Hence, the total complexity here is $O(\frac{n^2}{2}) + O(n^2) + O(n^2) = O(n^2)$.

---

[1]In this case, we define Hannan point the intersection between the already placed wire and the wire representing the shortest distance between such a wire and the target FU.

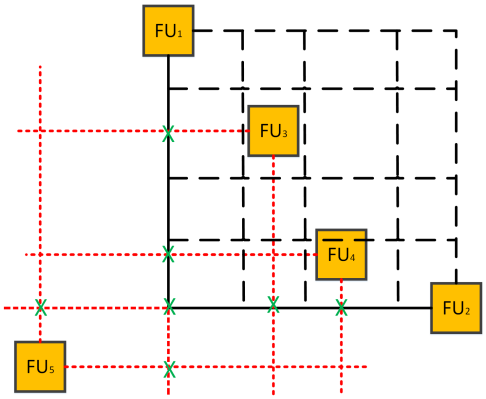

Figure 42: MAX-D algorithm at work. First it defines that max distance is between FU_1 and FU_2, then it tracks one among the possible min Manhattan distance paths between them (the other paths are dashed), then Hanna points are computed (green "X") and finally, where distance constraints respected, FUs are connected (the example of final design is in Fig. 43).

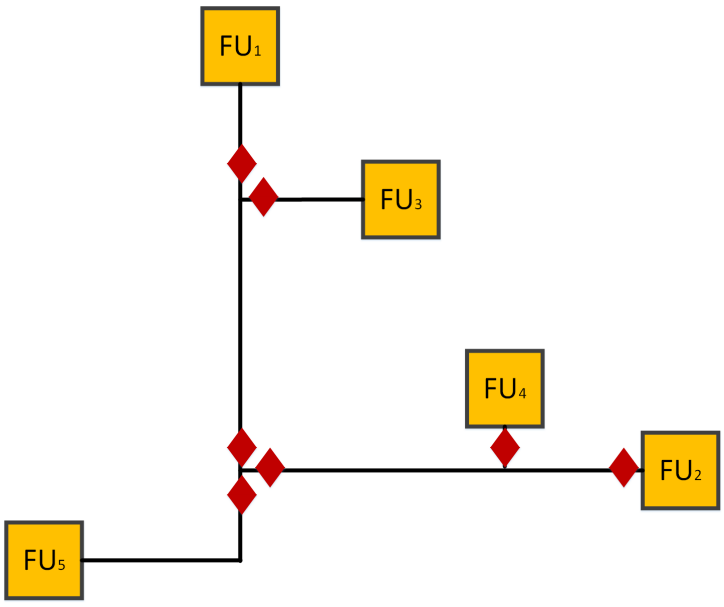

Figure 43: Example of final interconnections obtained with MAX-D.

Algorithm 23 Maximum distance constrained buslet generation

procedure MAX-D(G)
tot_pow = 0
for all buslet $b_j \in G$ **do**
B = $\emptyset$ ▷ B is the final structure of the buslet
R = FUs(b_j) ▷ R contains the remaining FUs to be connected to the buslet
Q = compute_clique(G, b_j) ▷ Here we create a clique from G having as nodes all the FUs in b_j having as weight of the arcs the Manhattan distance between the nodes
first_edge = $\emptyset$
for all edges $e_i \in Q$ **do**
if $w[e_i] \geq w[first\_edge]$ **then**
first_edge = e_i
B = first_edge
R = R - FUs(first_edge) ▷ Remove from the set of FUs to be connected those connected through first_edge
for all $r_i \in R$ **do**
next_wire = Hannan(r_i, first_edge)▷ Here we compute the closest wire connecting r_i to first_edge
if R+next_wire respects max distance w[first_edge] **then**
B = B+next_wire ▷ We add wire connecting r_i to first_edge
R = R-r_i ▷ We remove r_i from the list of the to be connected FUs
for all $r_i \in R$ **do** ▷ For all the FUs not connected because not respecting max distance constraint
next_wire = shortest_edge (r_i, B)▷ We compute the shortest edge connecting r_i to B
B = B+next_wire ▷ We add wire connecting r_i to first_edge
R = R-r_i ▷ We remove r_i from the list of the to be connected FUs
this_buf_pow = 0
for all $c_i \in CS(b_j)$ **do** ▷ For all communications scheduled (CS) in buslet b_j
R = Dijkstra(G, u_i, u_j)
track(R, G) ▷ Mark each wire composing R as crossed in G
for all edges $e_i \in G$ **do** ▷ G is composed by edges, already owning the number of times being crossed
Place_buffers(e_i) ▷ According to (5.13)
Compute $wire\_dpow(e_i)$ ▷ (5.11)
Compute $buf\_dpow(e_i)$
$this\_buf\_pow = wire\_dpow(e_i) + buf\_dpow(e_i) + static\_buf(e_i)$
this_buf_pow = this_buf_pow + static_power
tot_pow = tot_pow + this_buf_pow

5.9 MAX-D-MAXHOP

MAX-D technique intrinsically has a big drawback: in the first tracked line, there will be a number of steiner points, which will make huge the power consumption due to buffer load. A solution to this problem is to include also a constraint to the maximum number of steiner points a given communication needs to cross: Thanks to this, we my more difficultly meet the max distance constraint; however, we can fix the maximum number of buffers to be loaded for a single communication.

5.9.1 Complexity

Here we are adding the overhead of checking hop distance to MAX-D algorithm. However, when we are computing max distance, this is an $O(1)$ operation. Fot this, the total complexity remains $O(n^2)$.

5.10 Hierarchical partitioning

Let us use, now, a completely different approach. Now, we aim to build a binary tree-like structure. In order to do this, we hierarchically partition the floorplan computing centers of gravity. Then, when we will have partitions of one functional unit each, we will build our binary tree, connecting the previously computed centers of gravity (CG).

We can find an example of hierarchical partitioning in Fig. 44. More formally, let us say we have a set of functional units S to be connected through a buslet. We compute the center of gravity of it

$$CG(S) = \left[\frac{\sum_{\forall FU_i \in S} x(FU_i)}{|S|}; \frac{\sum_{\forall FU_i \in S} y(FU_i)}{|S|}\right] \tag{5.15}$$

Algorithm 24 Maximum distance constrained buslet generation with maximum hop distance constraint

procedure MAX-D(G, max_hops)
  tot_pow = 0
  for all buslet $b_j \in G$ **do**
    B = $\emptyset$ ▷ B is the final structure of the buslet
    R = FUs(b_j) ▷ R contains the remaining FUs to be connected to the buslet
    Q = compute_clique(G, b_j) ▷ Here we create a clique from G having as nodes all the FUs in b_j having as weight of the arcs the Manhattan distance between the nodes
    first_edge = $\emptyset$
    for all edges $e_i \in Q$ **do**
      if $w[e_i] \geq w[first\_edge]$ **then**
        first_edge = e_i
    B = first_edge
    R = R - FUs(first_edge) ▷ Remove from the set of FUs to be connected those connected through first_edge
    for all $r_i \in R$ **do**
      next_wire = Hannan(r_i, first_edge)▷ Here we compute the closest wire connecting r_i to first_edge
      if R+next_wire respects max distance w[first_edge] **then**
        if R+next_wire respects max hop distance max_hops **then**
          B = B+next_wire ▷ We add wire connecting r_i to first_edge
          R = R-r_i ▷ We remove r_i from the list of the to be connected FUs
    for all $r_i \in R$ **do** ▷ For all the FUs not connected because not respecting max distance constraint
      next_wire = shortest_edge (r_i, B)▷ We compute the shortest edge connecting r_i to B
      B = B+next_wire ▷ We add wire connecting r_i to first_edge
      R = R-r_i ▷ We remove r_i from the list of the to be connected FUs
    this_buf_pow = 0
    for all $c_i \in CS(b_j)$ **do** ▷ For all communications scheduled (CS) in buslet b_j
      R = Dijkstra(G, u_i, u_j)
      track(R, G) ▷ Mark each wire composing R as crossed in G
    for all edges $e_i \in G$ **do** ▷ G is composed by edges, already owning the number of times being crossed
      Place_buffers(e_i) ▷ According to (5.13)
      Compute $wire\_dpow(e_i)$ ▷ (5.11)
      Compute $buf\_dpow(e_i)$
      $this\_buf\_pow = wire\_dpow(e_i) + buf\_dpow(e_i) + static\_buf(e_i)$
      this_buf_pow = this_buf_pow + static_power
    tot_pow = tot_pow + this_buf_pow

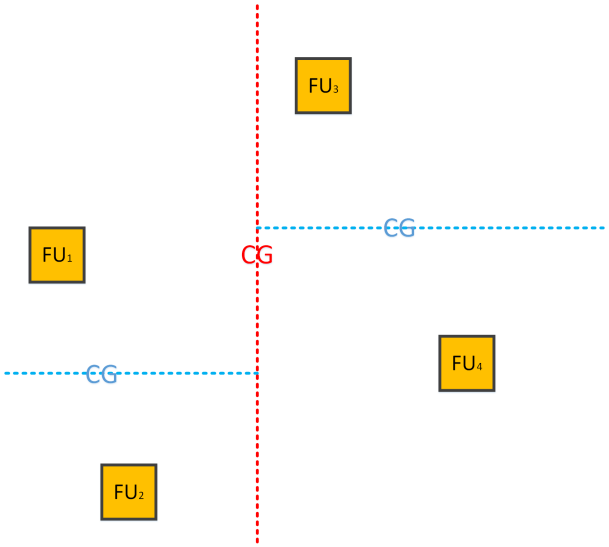

Figure 44: Example of hierarchical partitioning. Here we are recursively partitioning the floorplan (first partitioning in red, second in blue and, finally, connect all the centers of gravity in reverse order to that they were created.).

where $x(FU_i)$ is x value of the position of FU_i while $y(FU_i)$ is the y position. Now, we decide if we want to divide S partitioning vertically or horizontally. If we have

$$\forall FU_i, FU_j \in S \max\{d[x(FU_i), x(FU_j)]\} > \max\{d[y(FU_i), y(FU_j)]\} \tag{5.16}$$

then we will partition S vertically, otherwise horizontally. We will use, this way, a divide and conquer-like approach. Furthermore, after tracked the buslet, we can apply the buffer placements in MST-2B and MST-1B.

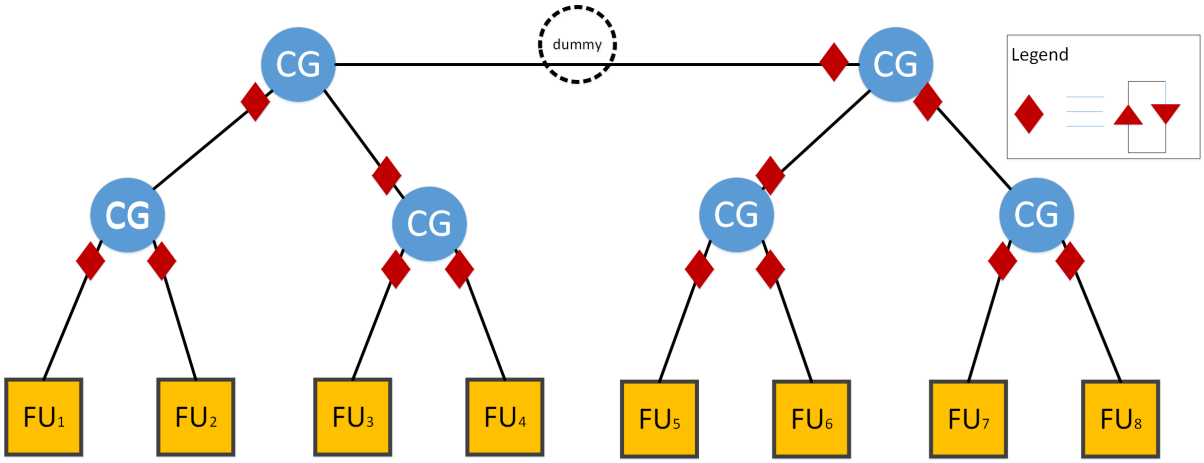

Figure 45: Example of implementation of hierarchical partitioning. In blue we have the centers of gravity. We see that the last one is a dummy nose as it connects two wires only and, for this, it can not be considered a steiner point.

Algorithm 25 Hierarchical partitioning wire generation

```
procedure HP(G)
    if size(G) = 1 then return G
    F = ∅
    Compute CG(G)                          ▷ According to formulation in (5.15)
    Compute maxx(G)
    Compute maxy(G)
    if maxx > maxy then
        L = left_partition(G, CG)
        R = right_partition(G, CG)
        F = F + HP(L) + HP(R) + CG
    else
        U = up_partition(G, CG)
        D = down_partition(G, CG)
        F = F + HP(U) + HP(D) + CG
    return F
```

5.10.1 Complexity

Computing the first center of gravity will take $O(n)$. After partitioning it, we will have two sets of approx $\frac{n}{2}$ size each, which will need to be partitioned and so on. We will perform such a division $\log_2(n)$ times; so, the time for dividing will be $O(nlog(n))$. The number of generated CG will be $n-1$; so, conquering will take $O(n)$ time. For this, we can say that the generation of such a buslet will take $O(nlog(n))$ time. However, we have to perform also buffer placement, which will be dominant as it is $O((n+r)^2)$. Here we know r as it is $n-1$. For this, we can easily say that the complexity will be $O(4n^2)$.

CHAPTER 6

EXPERIMENTAL RESULTS

All the algorithms presented in the previous sections were implemented in C++. In order to evaluate both final results and runtimes, several runs were performed, from the mediabench benchmark suite (with number of nodes and edges ranging from (11, 7) for DFG `hal` to (197, 196) for DFG `smooth_color_z_triangle`) [26]. In order to evaluate both final results and runtimes, several runs were performed, with 14 different DFGs, For each FU type, aspect ratios 1:1, 3:4 and 1:2 were used in the floorplanning step, in which we used the well-known floorplanner Parquet[27] to obtain the wirelengths and areas of the final designs. We assume the following areas of resources: add/sub 8 μm^2, multiplier/divider 128 μm^2, memory 512 μm^2, comparator 8 μm^2. For any run of our algorithm, where we define *run of our algorithm* the execution of our algorithm giving as input a particular combination of the following parameters: input DFG, maximum buslet cardinality, maximum FUs' fanin, maximum FUs' fanout, maximum DFG's latency, a set of 20 solutions were produced, in which the aspect ratio of any FU was randomly chosen. Then, for all of these solutions power estimation was performed for all the patterns we have presented and then these values were averaged. Communication delay (FU-to-FU signal delay) is assumed to take 1 clock cycle, and the communication as well as operations scheduling done assume this delay for all communication. All the runs were performed on a 4th Generation Intel Core i7-4720HQ Processor (2.60 GHz with 1600 MHz 6 MB cache) processor machine with 16 GB RAM.

6.1 Performance for scheduling and binding algorithms

The most significant metrics we determined were area size (Fig. 51) and wirelength (Fig. 50) shows the wirelength interchange average obtained from 14 different DFGs for different buslet cardinality constraint, comparing four different algorithm settings:

- No similarity solution pruning, propagating from each binding step 1 best partial solution only (NSP1).
- No similarity solution pruning, propagating from each binding step 20 best partial solutions (NSP20).
- Use of similarity solution pruning, propagating from each binding step 1 best partial solution only (YSP1).
- Use of similarity solution pruning, propagating from each binding step 20 best partial solutions (YSP20).
- Force Directed Binding (FDB).
- No similarity solution pruning, propagating from each binding step 1 best partial solution only with lookahead of next com to be bound (LA1).
- Simultaneous iso-scheduled communications binding (SIMCOM).

6.1.1 Comparison with another HLS algorithm

In order to evaluate how fair is the solution we obtain for dedicated interconnections (cardinality > 2), which will be used to evaluate the effect on wirelength of buslets, we have compared

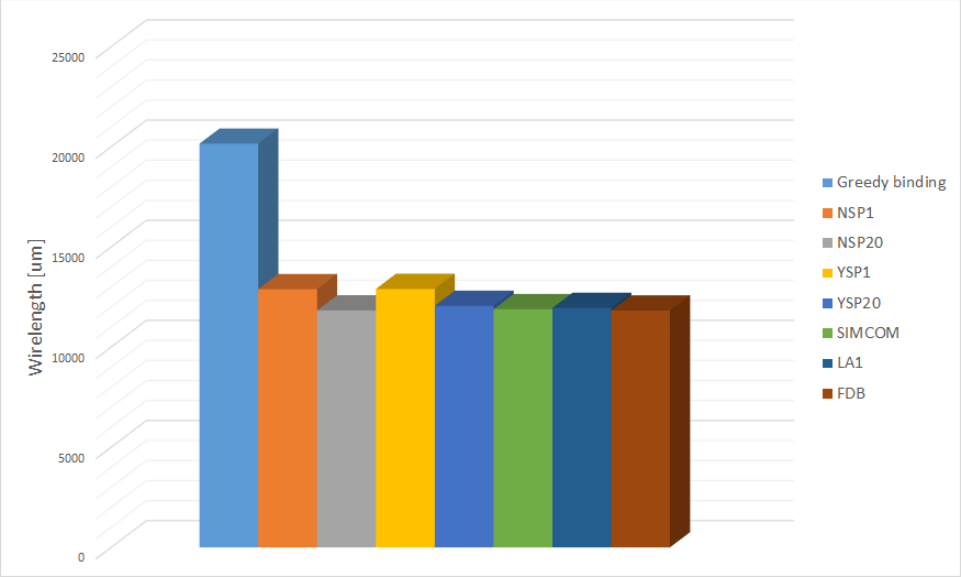

Figure 46: Comparison between greedy-binding method and ours for buslet cardinality 2.

these results with those obtained from another common sense algorithm. This is structured as follows: as scheduling algorithm it will use FDS only while, as binding, a greedy approach which tries to maximize as much as possible the reuse of the same already created interconnections. However, the special case of our algorithm for buslet cardinality of 2 achieves lower wirelenghts than the latter FDS and greedy-binding method by about 55% on an average, and thus we compare the buslet results (cardinality > 2) to our version of the dedicated interconnect results (Fig. 46). This is possible thanks to the efficacy of our FDCS algorithm, which allows us to minimize the number of interconnections needed in the binding step. So, all the comparisons will be performed here on will be fair.

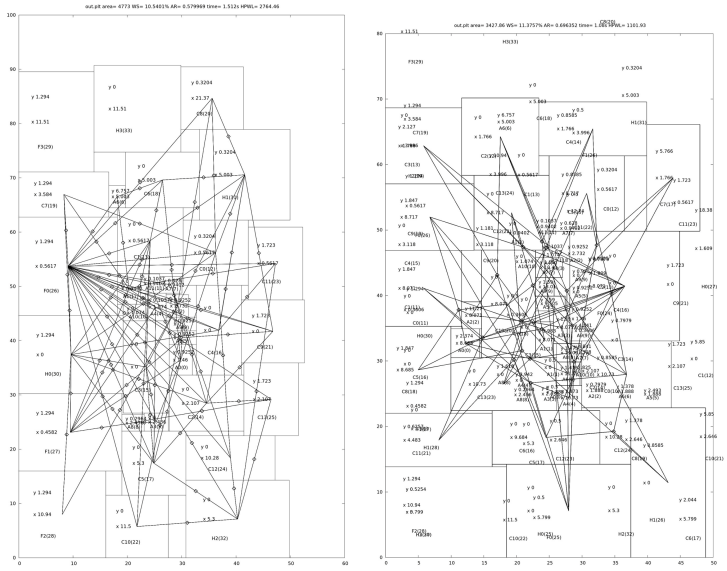

Figure 47: Comparison between two different output chip layouts for `smooth_color_z_triangle` (197 nodes, 196 edges) with max FU fanin and fanout constraints of 8. (a) Dedicated-interconnections based design (area: 4788.06 µm^2, FUs: 34, WL: 2847.29 µm). (b) Buslet-based design with max cardinality 5 (area: 3147.09 µm^2, FUs: 27, WL: 1169.09 µm).

6.1.2 Discussion of results

For all the computed solutions a latency constraint of 1.6x DFG's asap time was imposed. Maximum fanin and maximum fanout constraints were set to 8. λ_{com} was set to 1 clock cycle. With these constraints, using buslets instead of dedicated interconnections, we obtained a wirelength reduction ranging between 35% and 71% as shown in Table III. According to these data, the wiring complexity was significantly reduced. We can visually notice (Fig. 47) this comparing the output of the floorplanner for max buslet cardinality of 5 (Fig. 47b) with the dedicated interconnections solution (Fig. 47a). Let us focus, for example, on the FU "H1":

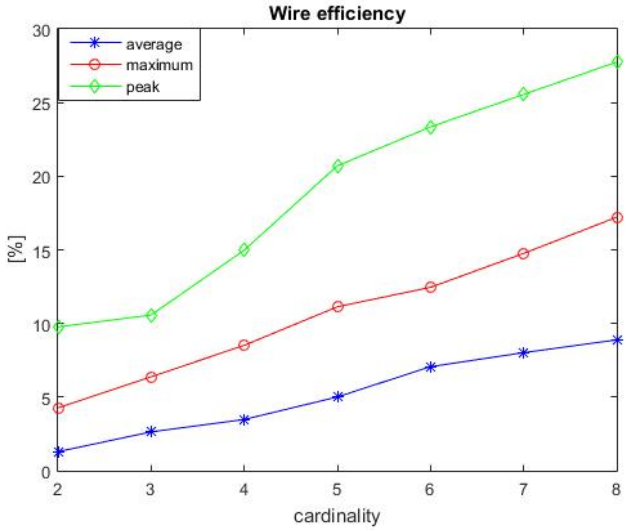

Figure 48: Average, maximum and peak wire efficiencies gained using NSP1, $\lambda_{tot} = 1.6 \cdot asap(DFG)$, max FU fanin/fanout = 8.

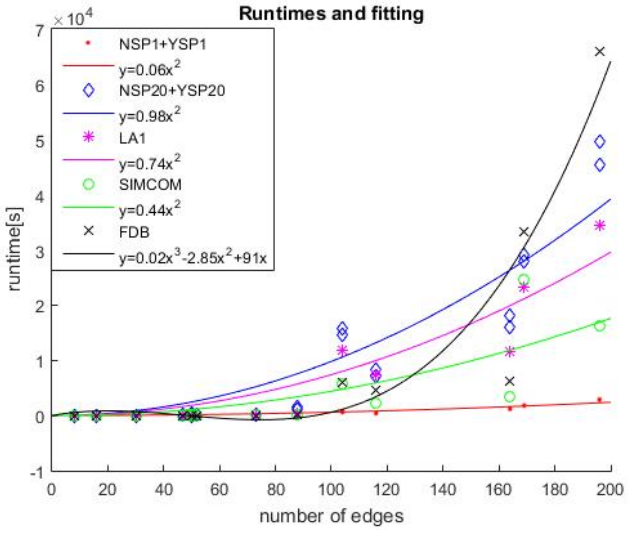

Figure 49: Runtime data and curve fitting for the designed HLS algorithms

in Fig. 47a it is placed on the top right of the floorplan and we can count 7 wires it is connected to, while in Fig. 47b, where it is placed in the low left, it is connected through 3 buslets only. Such a reduction happens for all the FUs. Same thing happens for "F0", which is the most connected FU in Fig. 47a: its number of interconnections scales of a factor greater than 2 in Fig. 47b. We can also see a wirelength reduction trend with the increase of maximum buslet cardinality in Fig. 50 and Table III. According to this, if we look at how average wire efficiency (Table IV and Fig. 48) changes as a function of buslet cardinality, we see that it has a growing trend as buslet cardinality increases. In particular, we can see that best results are obtained propagating more than one solution: comparing NSP20 to NSP1 (Table III), we have a reduction in wirelength up to 8%. Empirically we can clearly see that we obtained the best results using SIMCOM binding, which is followed by LA1. We can explain this thinking that LA1 will look at the next communication only, while, in a design, there may be more than two overlapping communications. This empirically states that simultaneously binding iso-scheduled communications is the best strategy, leading to the best results.

If we look at runtimes (Table I), we notice that runtimes are mainly affected by the number of edges in the graph as opposed to the number of nodes:

- For NSP1 and YSP1 runtime $t = 0.06e^2$, where e is the number of edges of the input DFG
- For NSP20 and YSP20 runtime $t = 0.98e^2$
- For LA runtime $t = 0.74e^2$
- For SIMCOM runtime $t = 0.44e^2$
- For FDB runtime $t = 0.02e^3 - 2.85e^2 + 91e$

This was also stated during the theoretical analysis of the designed algorithms in Ch. 3 and Ch. 4: in particular, as our scheduling and binding algorithms are buslet and, for instance, communication-centric, the number of edges in the DFG, which represents also the number of communications to be both scheduled and bound, is the main parameter which affects the runtime. If we look at the runtime in Fig. 49, we see that all the designed techniques, apart from FDB, empirically have a quadratic best fit. This is consistent with the theoretical analysis: we expected all of these being $O(kne^2)$ where k is a factor differing for all the designed algorithms. Also FDB behavior is consistent with the theoretical analysis as it was predicted being $O(n^4e^4)$. Among these, we can say that the algorithm providing the best wirelength (SIMCOM) is also the second in speed (the fastest is simple CB). FDB has a cubic best fit and, for this, it is fast for low number of edges but, beyond 120 edges, increases rapidly, already having, for $e \approx 200$, the worst runtime.

Looking at total chip area (Table II, Fig. 51), we can observe that, for buslet cardinality lower than 6, increasing buslet cardinality, a decreasing trend occurs. This can be explained by observing that the number of FUs used is higher for lower buslet cardinalities. This is an effect of the maximum fanin/fanout constraint. If we are using dedicated interconnections, we can connect a given FU_i to at most $max\_fanin + max\_fanout$ other FUs. However, if we use buslets of maximum cardinality $\max\{|b|\}$, as the fanin/fanout constraint applies to buslets, we will be able to connect at most $(max\_fanin + max\_fanout) \cdot (\max\{|b|\} - 1)$ FUs to FU_i while respecting the actual fanin/fanout constraint. For this, a reduction of the number of FUs and

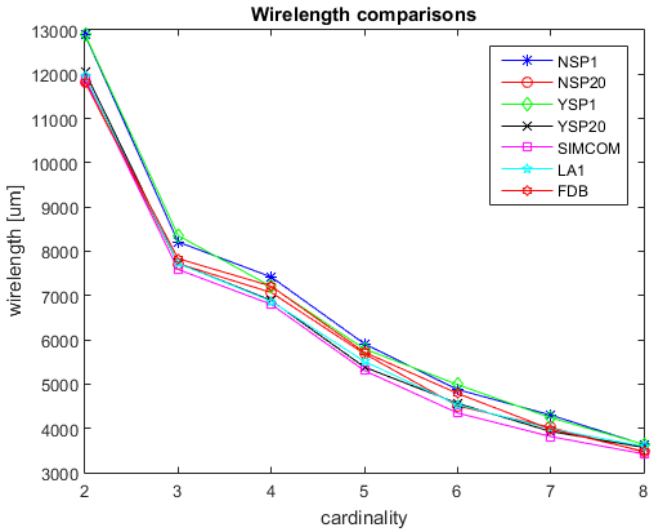

Figure 50: Total wirelength averages

thus layout area is expected.

6.2 Power estimation

All the following power estimations are performed on the outputs of NSP1. When computing the power, we are assuming supply voltage 1.2 V and working frequency 1 GHz. We have analyzed the following approaches:

- *SBP*: standard buffer placement
- *MST-2B*: MST with2 buffer placed per wire
- *MST-1B*: MST with 1 buffer placed per wire

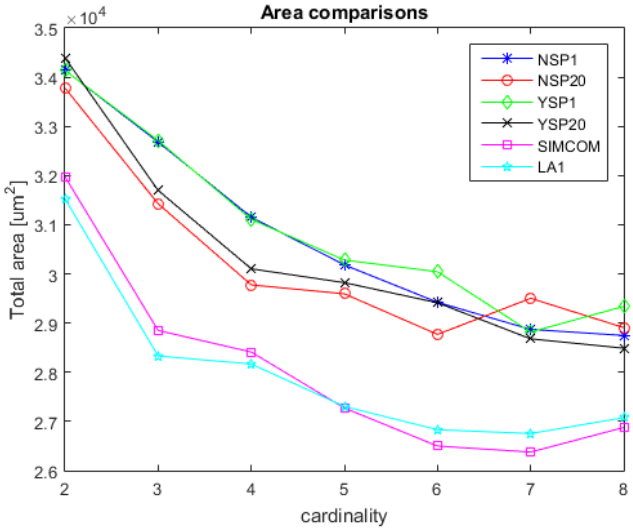

Figure 51: Total area averages

- *MST-LOGD*: MST with constraint on the maximum hop distance between FUs, which has to be logarithmic
- *MST-BF3*: MST with branching factor 3 (branching factor set to 3)
- *MAX-D*: buslet with constraint on maximum physical distance between each FU
- *MAX-D-MAXHOP3*: buslet with constraint on maximum physical distance between each FU and maximum hop distance 3
- *HP*: hierarchical partitioning

The technology node assumed is 22 nm with unit length wire capacitance 0.11 $\frac{\text{fF}}{\mu\text{m}}$[28], input capacitance for an inverter (C_L^{inv}) 0.08 fF[29] and leakage current (I_{leak}^{inv}) 3.8 pA[29]. In order

to compute the dynamic power to drive an FU we first define its *dynamic energy per operation* as

$$DE_{op}^{FU} = \frac{1}{2} \cdot V_{dd}^2 \cdot C_L^{FU} \cdot p(0 \rightarrow 1) \tag{6.1}$$

where

- V_{dd} is the supply voltage.
- C_L^{FU} is the sum of the input loads of all the logic gates of the FU.
- $p(0 \rightarrow 1)$ is the probability that the driving signal will indeed be a 0 to 1 transiting signal. Probabilities of 0 and 1 of a signal being uniform, $p(0 \rightarrow 1) = \frac{1}{4}$; however, since this assumption is generally incorrect, and depends on the logic of the path generating the signal, in our general formulation, we set $p(0 \rightarrow 1)$ to be a little higher at $\frac{1}{3}$.

From (6.1) we can define the dynamic power for a given FU as

$$DP^{FU} = DE_{op}^{FU} \cdot f \cdot \varepsilon_{FU} \tag{6.2}$$

where

- f is the working frequency.
- ε_{FU} is the fraction of cc's over the latency period in which it will be "busy" (potentially driven by a $0 \rightarrow 1$ transiting signal).

Leakage power is computed as

$$P_{leak}^{FU} = V_{dd} \cdot I_{leak}^{FU} \tag{6.3}$$

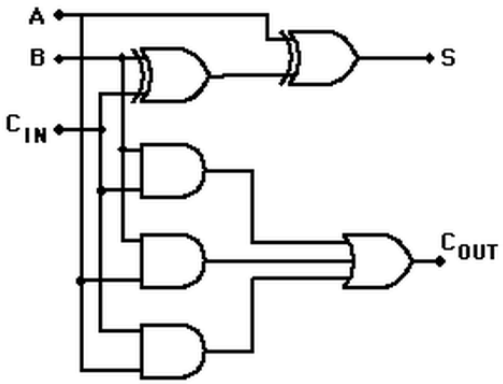

Figure 52: Logic gates composing a FA

where I_{leak}^{FU} is the leakage current.

Tristate buffers are realized with two inverters and a transmission gate (TG) cascaded. For this, we can say that the total input load of tristate buffers is $3{\cdot}C_L^{inv}$ of an inverter (input for the two inverters and for the transmission gate) while the leakage power consumed is $2{\cdot}P_{leak}^{inv}$ (we ignore the leakage current of a T-gate since we estimate that it has much smaller leakage than an inverter because its path to ground goes through the gate of at least one transistor).

Let us derive the leakage and dynamic power also for a full adder (FA). We know that a FA is made of two 2-inputs XOR gates, three 2-inputs AND gates and one 3-inputs OR gate. For each of these, we know that the dynamic power depends on the number of fanins of each gate. For this, we can say that, according to the FA design we take into account (Fig. 52), the input load for a FA is $15 \cdot C_L^{inv}$. From this, dynamic power is obtained using (6.1) and (6.2). Talking about leakage power, we can see that the equivalent resistance will always be comparable to that of

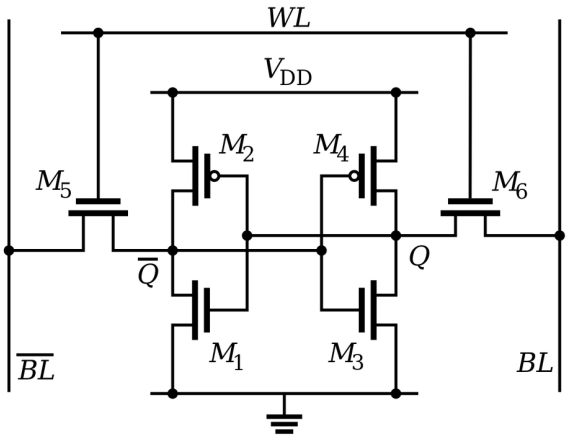

Figure 53: SRAM cell

the inverter, as $R_{off} \gg R_{on}$, where R_{off} is the drain-source resistance when the transistor is off and R_{on} is the drain-source resistance when transistor is on. For this, for each gate, we have 1x the leakage power consumed by an inverter. As we have 6 different gates, at the end we will have a leakage power consumption of $6 \cdot P_{leak}^{inv}$.

Let us assume we are working with 16-bit data. Assuming that for arithmetic operations we will use a ripple carry adder (RCA), its total input load will be $240 \cdot C_L^{inv}$ and its leakage power $96 \cdot P_{leak}^{inv}$. If we want to determine the leakage and dynamic power for an array multiplier, we know that it requires, in our case, 16^2 FAs. Hence, its input load will be $3840 \cdot C_L^{inv}$ and its leakage power $1536 \cdot P_{leak}^{inv}$.

Finally, let us analyze an SRAM cell power consumption: as it is made of two inverters and two pass transistors, its power consumption is the same as the tristate buffer. Now, let us assume we will use 16x256 memories. Its input load in write will be $48 \cdot C_L^{inv}$ (as we will write 16 bits

only), its input load in read $16 \cdot C_L^{inv}$ (pass transistors on the address line activated only) while the leakage power will be $8192 \cdot P_{leak}^{inv}$.

Let us compute, now, the lowest allowable switching time t_{sw}^{min}. From our former analysis, we can see that the most critical FU we will use is the multiplier, as it will have a critical path crossing 32 FAs. Let us assume here that the total wirelength of the interconnections in a FA to be $\left(\frac{1}{10}\right)^{th}$ the average wirelength of a buslet. From experimental data, we know that it is 80 µm. Furthermore, as we know that R_{wire} is 17 $\frac{\Omega}{\mu m}$[30], the total delay in a single FU due to wiring is $t_{wire}^{FU} = 11.9$ ps. Now, in the entire multiplier, this will be $t_{wire}^{mul} = 380.8$ ps. Let us analyze, now, transistors switching time. We know that, in a FA, the critical path crosses 2 logic gates. Furthermore, the longest switching time is $t_{rd} = 0.48$ ps[29]. Hence, the total transistors switching time in a FA is $t_{tran}^{FA} = 1.44$ ps and, for the multiplier, it will be $t_{tran}^{mul} = 46.08$ ps. Hence, the lowest switching time we may theoretically have is $t_{sw}^{min} = 426.88$ ps. From this, we can deduct that the theoretical highest frequency is $f_{max} = 2.34$ GHz. We want to use a frequency which is below the theoretical maximum one: this is why we want to choose $f = 1$ GHz.

Let us analyze the total power consumption (Table XI, Fig. 54) for different buslet designs. From this, we can see that SBP, for any buslet cardinality, is the worst approach for the power metric. We could expect this because, with such a structure, the signal propagates through the whole buslet wirelength. We can see that, in general, the best approaches are here MST-2B, MST-1B and MST-LOGD. Hence, here looks like that the total wirelength plays a fundamental role for the final power consumption (Table XIII, Fig. 55).

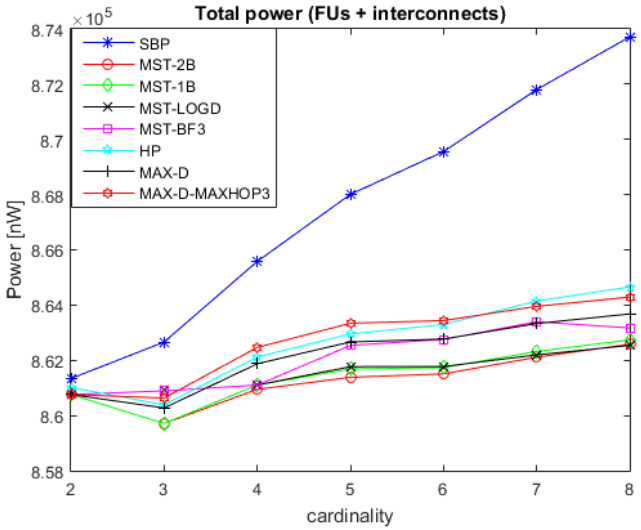

Figure 54: Total power consumption

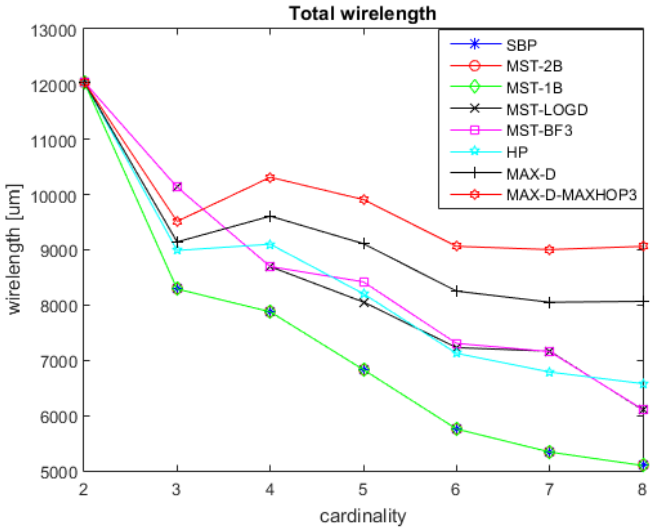

Figure 55: Total wirelengths

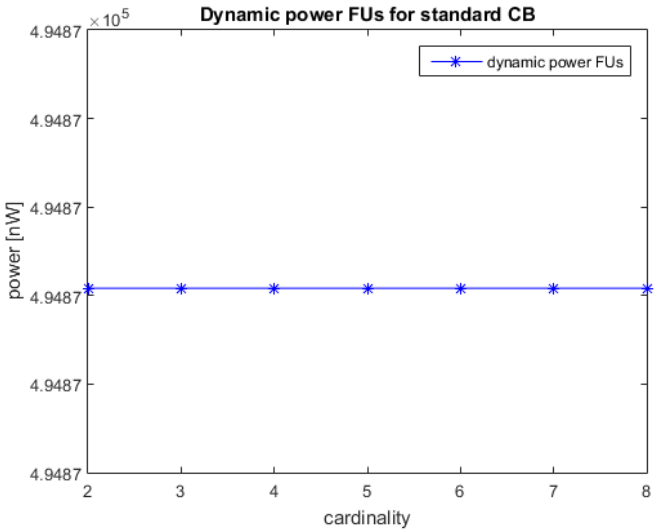

Figure 56: Dynamic power contribution of FUs

Let us analyze the contribution of FUs power. Dynamic power (Table IX, Fig. 56) is constant through all the buslet cardinalities. We have, as input for all our algorithms, a DFG, whose essential components are operation nodes. Each operation node represents an operation to be executed, regardless the number of bound FUs, within the total design latency $\overline{\lambda}$. Even if we minimize the number of FUs, all the operation nodes will be bound to a FU; so, all of them will always be executed. Hence, dynamic power consumed by FUs does not change for any buslet cardinality or FU fanin/fanout constraint. However, leakage power (Table X, Fig. 57) depends on the number of the bound FUs. As stated in Par. 6.1, the number of FUs decreases when the buslet cardinality increases. Hence, the leakage power decreases and, as dynamic power is constant, also total power consumed by FUs decreases.

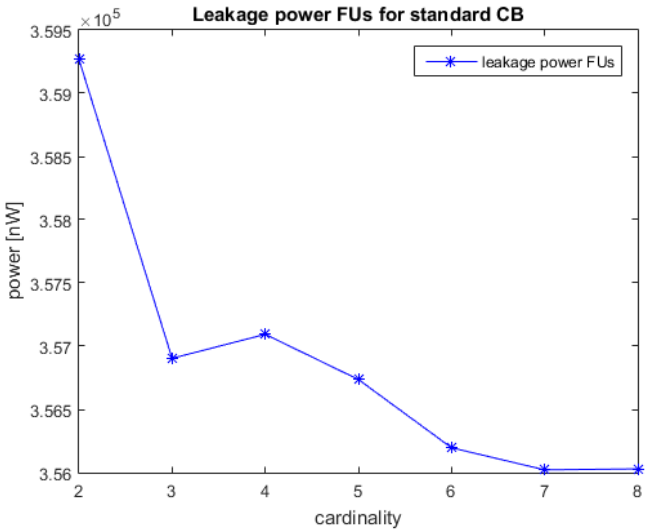

Figure 57: Leakage power contribution of FUs

Let us analyze now, in detail, all the power contributions of the buslet. Dynamic power, in particular, receives contributions from both wire capacitance and buffer input capacitance. If we analyze the contribution of wire capacitance (Table VI, Fig. 58), we see, as we could expect, that we have the highest contribution from SBP (as previously stated, the signal will be propagated through all the buslet). This is the upper bound for MST-based interconnect structures. Then, we see that we have the lowest contribution for MST-2B. As it will isolate just the shortest path the signal needs to propagate through, this is the lower bound for MST-based approaches.

Let us analyze, now, the contribution to dynamic power given by tristate buffers (Table VII, Fig. 60). Here we see that SBP is the lowest (we expect this as their number is in the worst

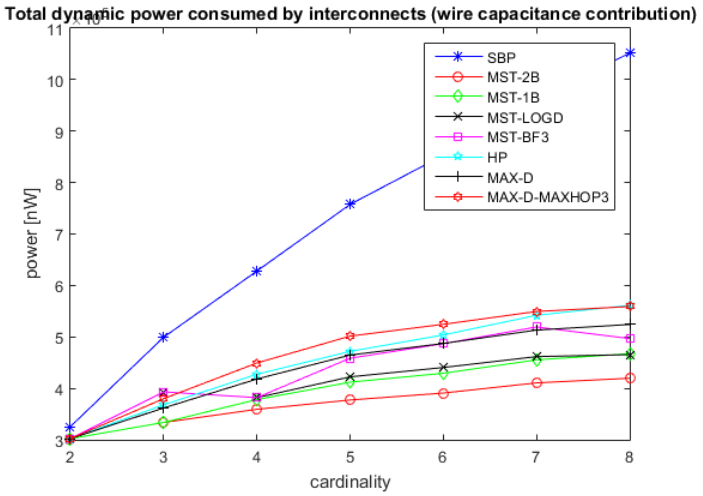

Figure 58: Dynamic power contribution of wire

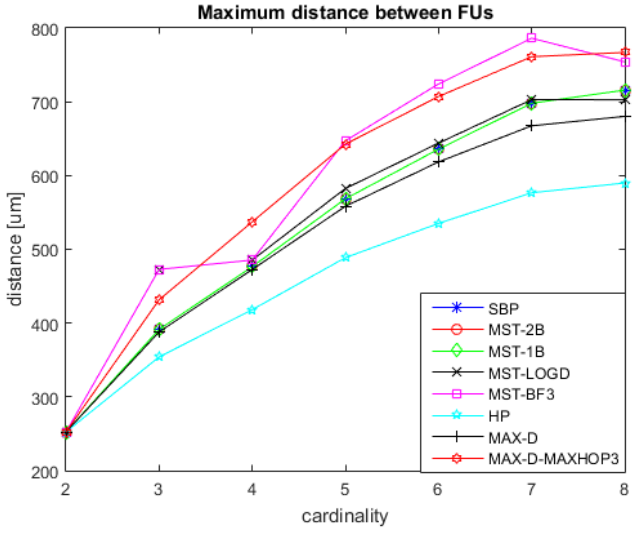

Figure 59: Maximum distance between any two connected FUs

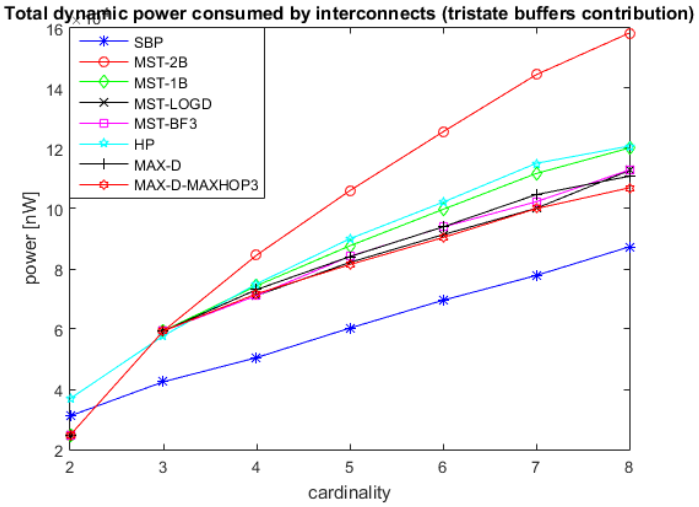

Figure 60: Dynamic power contribution of tristate buffers input load

case the number of FUs connected through a buslet) while the worst is MST-2B.

Analyzing leakage power (Table VIII, Fig. 61), which is here the leakage power consumed by tristate buffers, we see that we have the lowest contribution in SBP while the highest in both MST-2B and HP. leakage power is here directly proportional to the number of buffers. We can see that HP, which had a lower dynamic power consumption than MST-2B, here shows similar power values. This is due to the branching factor each steiner node has. In HP it will always be 3 and, for this, every time a signal crossing a steiner node will load 3 buffers while, in MST-2B, such a parameter is not precised; hence, dynamic power may be higher. Furthermore, with HP we will meet, through the signal shortest path, at most $2\log_2(n)$ steiner points while, for MST-2B, it will be, in the worst case, $2(n-1)$.

The total power consumed by the buslet is the sum of all of these three contributions (we

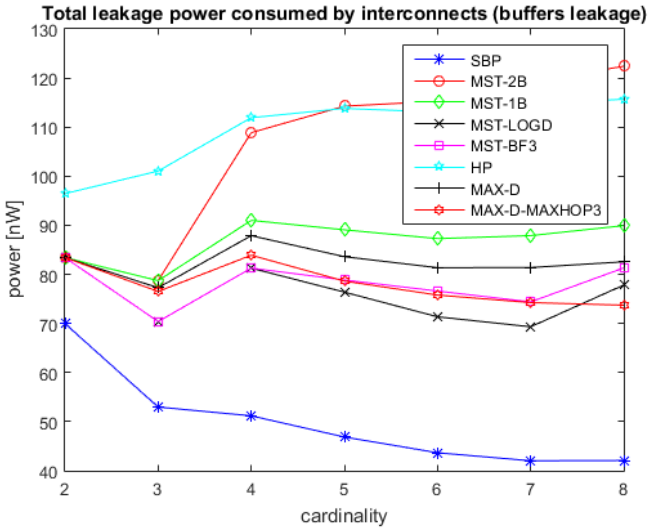

Figure 61: Leakage power contribution of tristate buffers

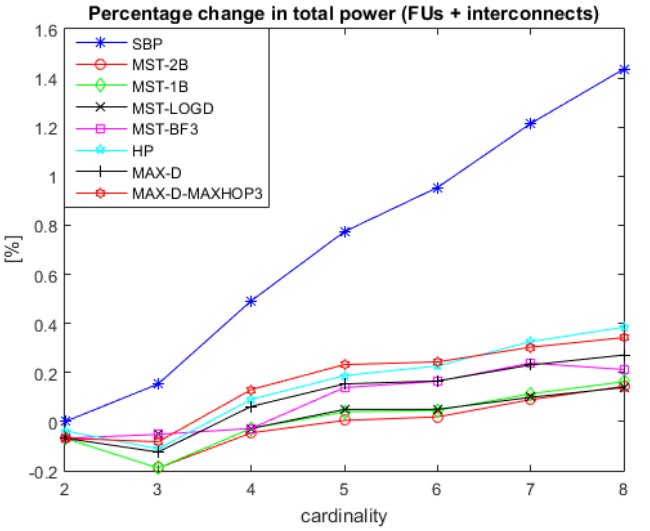

Figure 62: Percentage change in total power

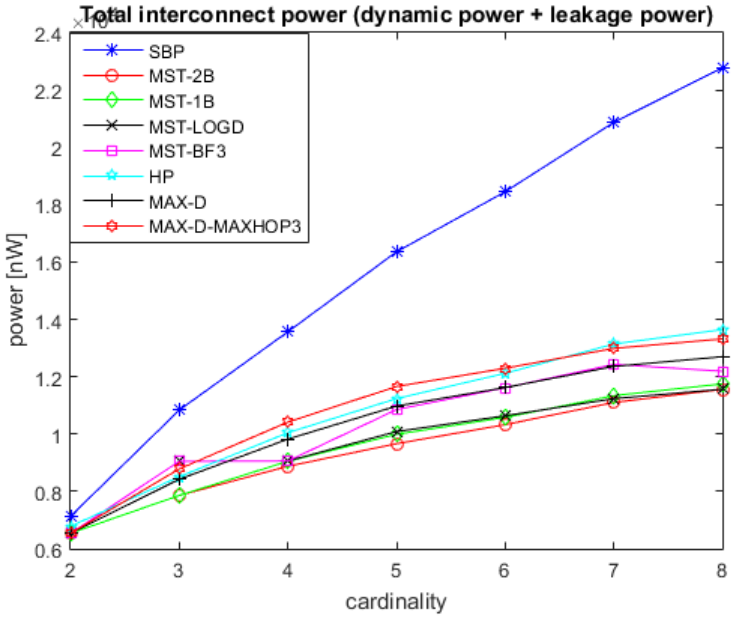

Figure 63: Total interconnect power

can see this in Table XI and Fig. 63). Hence, we see that the worst values are held by SBP while the best are for MST-LOGD and MST-2B, as the wire power consumption contribution is here the most relevant.

If we want to analyze the percentage change of power from the dedicated interconnects case (Table XII, Fig. 62), we see that, for cardinality 3, we have a reduction of 0.19% on the total power, for MST-2B and MST-1B. This is due to the fact that buslet power consumption does not increase as much as FUs leakage power decreases. Apart from this, however, we have a sublinear increment in the total power, due to the fact that FUs leakage power does not decrease significantly. We have, for cardinality 8, an increase in total power consumption of

about 0.8% for all the techniques apart from SBP, whose power increment is 1.43%. Regarding the wirelength (Fig. 55), we see that as expected SBP has the best values while MST-BF is second best ranked.

CHAPTER 7

CONCLUSIONS

In this thesis, we proposed a novel and innovative interconnection paradigm for HLS designs, based on the use of flexibly-structured buslets in place of dedicated interconnections. Moreover, a new scheduling algorithm, Force Directed Communication Scheduling, was developed to perform communication scheduling (to the best of our knowledge, the first time such an operation has been performed in HLS) with the goal of minimizing the number of buslets. A number of buslet-centric binding algorithms were also proposed, with the aim of minimizing wirelength under maximum buslet cardinality and maximum FU fanin/fanout constraints. A "similar solutions pruning" heuristic was also developed, whose task is to avoid the propagation of similar solutions during binding. Finally, a number of buslet designs with multi-tri-stated points and different routing techniques were developed and implemented with the aim of minimizing either the total power consumed by buslets or the maximum communication distance (in order to meet latency constraints without increasing the number of FUs or buslets). Experimental results show that the use of buslets results in an average wirelength reduction in range of 35% to 71% across a set of 14 tested DFGs. This clearly establishes the efficacy of buslet-based interconnections in reducing interconnect complexity. Finally, power dissipation grows sublinearly with maximum buslet cardinality. In particular, for high cardinalities, we have found a buslet design technique that results in logarithmic growth in power with buslet cardinality. Further challenges in this direction involve the optimization of dynamic power

consumption also for buslets with low cardinalities (as opposed to achieving logarithmic power growth asymptotically, i.e., for buslets with high cardinalities).

APPENDIX

EXPERIMENTAL DATA COLLECTED

Here we provide all the tables with various empirical data that have been referenced in Chapter 6.

TABLE I: AVERAGE RUNTIMES FOR DIFFERENT DFG SIZES

		Average runtimes [s]						
DFG name	nodes/ edges	NSP1	NSP20	YSP1	YSP20	LA1	SIMCOM	FDB
`hal`	11/8	0.06	0.169	0.015	0.015	0.122	0.068	0.004
`horner`	18/16	0.171	3.42	0.249	0.608	0.98	0.552	0.105
`arf`	28/30	0.452	11.159	0.734	9.063	3.009	3.213	0.162
`ewf`	34/47	1.921	32.609	3.156	46.163	29.579	20.509	1.004
`smooth`	51/52	1.78	36.926	2.626	46.095	17.491	11.948	0.414
`motion`	53/50	1.982	29.277	3.015	23.55	30.429	13.753	1.449
`feedback`	53/50	12.834	246.226	20.348	296.99	187.059	88.734	20.066
`collapse`	56/73	15.361	186.836	24.611	228.616	66.494	79.581	8.409
`write`	106/88	94.048	1438.797	91.209	1194.191	379.791	237.063	41.464
`interpolate`	108/104	565.312	15717.174	826.798	14555.136	11780.99	5773.47	6027.027
`matmul`	109/116	336.527	8193.599	542.706	7172.703	7361.628	2294.239	4492.863
`idctcol`	114/164	1164.966	18102.956	1222.529	16008.73	11513.62	3501.898	6324.749
`jpeg`	134/169	1728.064	29170.12	1848.222	27840.8	23334.3	24555.86	33360.879
`smooth`	197/196	2829.357	49515.27	2941.344	45300.13	34534.22	16134.999	65808.893

APPENDIX (Continued)

TABLE II: AVERAGE AREA

	Average area [µm^2]						
buslet card	NSP1	NSP20	YSP1	YSP20	LA1	SIMCOM	FDB
2	34140	33780	34138	34393	31527	31976	28520
3	32682	31427	32710	31698	28330	28850	27240
4	31154	29776	31110	30103	28168	28412	26931
5	30183	29595	30281	29822	27306	27273	26329
6	29423	28769	30044	29413	26832	26500	26044
7	28871	29504	28818	28679	26754	26378	25906
8	28749	28908	29340	28490	27077	26877	25849

TABLE III: AVERAGE WIRELENGTH

	Average wirelength [µm]						
buslet card	NSP1	NSP20	YSP1	YSP20	LA1	SIMCOM	FDB
2	12890.23	11829.41	12903.97	12056.56	11951.75	11891.36	11822.00
3	8210.82	7721.33	8355.09	7728.88	7724.66	7588.27	7834.83
4	7419.55	7069.31	7203.90	6889.79	6872.57	6801.19	7212.15
5	5911.16	5684.90	5795.64	5384.46	5513.70	5305.71	5718.99
6	4877.09	4506.85	4989.47	4564.82	4534.18	4348.19	4789.79
7	4301.18	4021.56	4240.95	3930.40	3991.34	3815.23	3981.63
8	3632.94	3467.75	3639.42	3569.15	3595.23	3422.67	3480.06

TABLE IV: AVERAGE WIRE EFFICIENCY

buslet card	Average wire efficiency [%] avg(dfg_j(avg($\varepsilon(b_i)$)))	Max wire efficiency [%] avg(dfg_j(max($\varepsilon(b_i)$)))	Peak wire efficiency[%] max(dfg_j(max($\varepsilon(b_i)$)))
2	0.6	4.3	9.8
3	0.7	6.8	10.6
4	0.9	8.5	14.9
5	1.8	11.1	20.7
6	3.5	12.4	23.3
7	4.1	14.7	25.5
8	3.6	17.2	27.7

APPENDIX (Continued)

TABLE V: AVERAGE WIRELENGTH FOR DIFFERENT BUSLET STRUCTURES

	Average wirelength [µm]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	12043.56	12043.56	12043.56	12043.56	12043.56	12043.56	12043.56	12043.56
3	8290.99	8290.99	8290.99	10135.06	10135.06	8988.84	9143.08	9514.66
4	7877.12	7877.12	7877.12	8695.93	8695.93	9101.65	9609.30	10314.31
5	6830.58	6830.58	6830.58	8055.14	8419.98	8205.02	9111.87	9910.66
6	5756.40	5756.40	5756.40	7229.11	7305.61	7125.79	8252.74	9063.80
7	5338.84	5338.84	5338.84	7164.80	7157.40	6787.59	8051.16	9004.20
8	5094.44	5094.44	5094.44	6103.21	6109.72	6577.24	8066.88	9062.46

TABLE VI: TOTAL DYNAMIC POWER CONSUMED BY INTERCONNECTS - WIRE CAPACITANCE CONTRIBUTION

	Power [nW]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	325614.48	302265.00	302265.00	302265.00	302265.00	302265.00	302265.00	302265.00
3	499326.47	333705.21	333705.21	393170.09	393170.09	368010.09	361843.91	379925.31
4	627904.34	359291.88	378154.14	381928.38	381928.38	427339.17	418111.59	449245.76
5	757494.36	377306.02	411995.67	422277.06	458542.19	471866.75	464719.65	501538.98
6	852962.00	390877.94	429326.25	440535.92	486823.15	503781.09	487184.72	524452.74
7	965694.90	410624.50	455353.79	461668.42	519260.68	542027.72	513271.87	549435.31
8	1051381.50	419680.27	467384.43	465556.54	496738.44	561435.89	524144.84	559028.99

TABLE VII: TOTAL DYNAMIC POWER CONSUMED BY INTERCONNECTS - TRISTATE BUFFER CONTRIBUTION

	Power [nW]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	31166.38	24774.01	24774.01	24774.01	24774.01	36912.77	24774.01	24774.01
3	42498.09	59350.18	59350.18	59350.18	59350.18	57788.22	59350.18	59350.18
4	50518.95	84559.50	74394.82	71130.52	71130.52	75012.09	73117.42	71737.93
5	60317.44	105972.78	87644.58	82131.07	84163.36	89995.06	84064.25	81427.78
6	69564.78	125518.43	99759.03	91352.18	93900.22	102141.99	93949.53	90398.45
7	77806.02	144498.12	111680.10	100120.85	102304.29	115057.36	104669.25	100012.91
8	87322.46	158367.21	120174.26	112690.69	112834.14	120794.21	110792.24	106931.97

APPENDIX (Continued)

TABLE VIII: TOTAL STATIC POWER CONSUMED BY INTERCONNECTS - TRISTATE BUFFER LEAKAGE

	Power [nW]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	69.97	83.42	83.42	83.42	83.42	96.45	83.42	83.42
3	52.98	78.72	78.72	70.39	70.39	101.05	77.28	76.52
4	51.19	108.88	91.00	81.27	81.27	111.92	87.89	83.90
5	46.87	114.27	89.08	76.39	78.86	113.81	83.60	78.63
6	43.66	115.26	87.29	71.36	76.60	113.06	81.36	75.79
7	42.03	119.14	87.89	69.32	74.39	114.63	81.39	74.25
8	42.09	122.41	89.93	77.83	81.32	115.69	82.56	73.73

TABLE IX: DYNAMIC POWER CONSUMED BY FUS

	Power [nW]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04
3	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04
4	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04
5	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04
6	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04
7	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04
8	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04	494868.04

TABLE X: STATIC POWER CONSUMED BY FUS

	Power [nW]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	359267.20	359267.20	359267.20	359267.20	359267.20	359267.20	359267.20	359267.20
3	356903.60	356903.60	356903.60	356903.60	356903.60	356903.60	356903.60	356903.60
4	357093.60	357093.60	357093.60	357093.60	357093.60	357093.60	357093.60	357093.60
5	356736.40	356736.40	356736.40	356736.40	356736.40	356736.40	356736.40	356736.40
6	356196.80	356196.80	356196.80	356196.80	356196.80	356196.80	356196.80	356196.80
7	356022.00	356022.00	356022.00	356022.00	356022.00	356022.00	356022.00	356022.00
8	356029.60	356029.60	356029.60	356029.60	356029.60	356029.60	356029.60	356029.60

APPENDIX (Continued)

TABLE XI: TOTAL POWER

	Power [nW]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	861340.83	860759.44	860759.44	860759.44	860759.44	861015.24	860759.44	860759.44
3	862661.11	859711.47	859711.47	860892.43	860892.43	860388.66	860272.80	860633.67
4	865581.29	860947.54	861103.62	861104.09	861104.09	862120.58	861874.11	862465.22
5	868007.55	861384.28	861686.32	861768.99	862537.41	862955.49	862663.72	863342.41
6	869559.04	861508.03	861733.83	861773.96	862755.91	863296.36	862768.89	863437.65
7	871802.09	862111.64	862318.61	862195.15	863395.73	864146.38	863330.26	863953.26
8	873713.81	862581.00	862738.75	862540.42	863170.41	864657.93	863678.95	864290.59

TABLE XII: PERCENTAGE CHANGE IN TOTAL POWER

	Power [%]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	0.00	-0.07	-0.07	-0.07	-0.07	-0.04	-0.07	-0.07
3	0.15	-0.19	-0.19	-0.05	-0.05	-0.11	-0.12	-0.08
4	0.49	-0.05	-0.03	-0.03	-0.03	0.09	0.06	0.13
5	0.77	0.01	0.04	0.05	0.14	0.19	0.15	0.23
6	0.95	0.02	0.05	0.05	0.16	0.23	0.17	0.24
7	1.21	0.09	0.11	0.10	0.24	0.33	0.23	0.30
8	1.44	0.14	0.16	0.14	0.21	0.39	0.27	0.34

TABLE XIII: MAXIMUM DISTANCE BETWEEN FUS

	distance [µm]							
buslet card	SBP	MST-2B	MST-1B	MST-LOGD	MST-BF3	HP	MAX-D	MAX-D MAXHOP3
2	252.4321	252.4321	252.4321	252.4321	252.4321	252.4321	252.4321	252.4321
3	391.4565	391.4565	391.4565	472.7913	472.7913	354.1469	387.966	431.638
4	476.5574	476.5574	476.5574	485.4096	485.4096	418.2369	472.6802	537.204
5	568.4289	568.4289	568.4289	582.4692	647.0399	488.6677	558.275	642.5311
6	635.792	635.792	635.792	643.9562	724.1355	535.0732	618.4402	707.064
7	697.9911	697.9911	697.9911	702.9376	786.1211	576.6508	667.4153	761.1182
8	715.8234	715.8234	715.8234	702.7463	754.152	589.9182	680.1763	766.8425

CITED LITERATURE

1. Shockley, W.: The path to the conception of the junction transistor. Electron Devices, IEEE Transactions on, 23(7):597–620, 1976.

2. Turgeon, M. L.: Clinical hematology: theory and procedures, volume 936. Lippincott Williams & Wilkins, 2005.

3. Moore, G. E.: Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp. 114 ff. Solid-State Circuits Society Newsletter, IEEE, 11(5):33–35, 2006.

4. Wolf, S., Awschalom, D., Buhrman, R., Daughton, J., Von Molnar, S., Roukes, M., Chtchelkanova, A. Y., and Treger, D.: Spintronics: a spin-based electronics vision for the future. Science, 294(5546):1488–1495, 2001.

5. Moodera, J. S., Kinder, L. R., Wong, T. M., and Meservey, R.: Large magnetoresistance at room temperature in ferromagnetic thin film tunnel junctions. Physical Review Letters, 74(16):3273, 1995.

6. Yuasa, S., Nagahama, T., Fukushima, A., Suzuki, Y., and Ando, K.: Giant room-temperature magnetoresistance in single-crystal fe/mgo/fe magnetic tunnel junctions. Nature materials, 3(12):868–871, 2004.

7. Ikeda, S., Miura, K., Yamamoto, H., Mizunuma, K., Gan, H., Endo, M., Kanai, S., Hayakawa, J., Matsukura, F., and Ohno, H.: A perpendicular-anisotropy cofeb–mgo magnetic tunnel junction. Nature materials, 9(9):721–724, 2010.

8. Stockman, M. I.: Nanofocusing of optical energy in tapered plasmonic waveguides. Physical review letters, 93(13):137404, 2004.

9. Wang, Z. L., Gao, R. P., Pan, Z. W., and Dai, Z. R.: Nano-scale mechanics of nanotubes, nanowires, and nanobelts. Advanced Engineering Materials, 3(9):657, 2001.

10. Liu, D. and Svensson, C.: Power consumption estimation in cmos vlsi chips. Solid-State Circuits, IEEE Journal of, 29(6):663–670, 1994.

CITED LITERATURE (Continued)

11. Altunyurt, N., Aygun, K., Doran, K., and Mekonnen, Y.: Mitigation of far-end crosstalk induced by routing and out-of-plane interconnects, July 4 2013. WO Patent App. PCT/US2011/067,976.

12. Bhooshan, R., Kuve, S., and Puvvada, V.: Electro-migration (em) and voltage (ir) drop analysis of integrated circuit (ic) designs, January 1 2008. US Patent 7,315,992.

13. Wu, B. and Sherwani, N. A.: Effective buffer insertion of clock tree for high-speed vlsi circuits. Microelectronics journal, 23(4):291–300, 1992.

14. Fishburn, J. P.: Shaping a vlsi wire to minimize elmore delay. In Proceedings of the 1997 European conference on Design and Test, page 244. IEEE Computer Society, 1997.

15. Benini, L., De Micheli, G., Macii, E., Sciuto, D., and Silvano, C.: Address bus encoding techniques for system-level power optimization. In Design, Automation and Test in Europe, 1998., Proceedings, pages 861–866. IEEE, 1998.

16. Zhong, L. and Jha, N. K.: Interconnect-aware low-power high-level synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(3):336–351, 2005.

17. Gu, Z., Wang, J., Dick, R. P., and Zhou, H.: Unified incremental physical-level and high-level synthesis. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 26(9):1576–1588, 2007.

18. Herrmann, D. and Ernst, R.: Improved interconnect sharing by identity operation insertion. In Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design, pages 489–493. IEEE Press, 1999.

19. Park, N. and Kurdahi, F. J.: Module assignment and interconnect sharing in register-transfer synthesis of pipelined data paths. In Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers., 1989 IEEE International Conference on, pages 16–19. IEEE, 1989.

20. Ewering, C.: Automatic high level synthesis of partitioned busses. In Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on, pages 304–307. IEEE, 1990.

CITED LITERATURE (Continued)

21. Vigneswaran, T., Mukundhan, B., and Reddy, P. S.: A novel low power, high speed 14 transistor cmos full adder cell with 50% improvement in threshold loss problem. Enformatika Trans. Eng. Comp. Tech, 13:82–85, 2006.

22. Paulin, P. G. and Knight, J. P.: Force-directed scheduling for the behavioral synthesis of asics. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 8(6):661–679, 1989.

23. Prim, R. C.: Shortest connection networks and some generalizations. Bell system technical journal, 36(6):1389–1401, 1957.

24. Kruskal, J. B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical society, 7(1):48–50, 1956.

25. Dijkstra, E. W.: A note on two problems in connexion with graphs. Numerische mathematik, 1(1):269–271, 1959.

26. Lee, C., Potkonjak, M., and Mangione-Smith, W. H.: Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, pages 330–335. IEEE Computer Society, 1997.

27. Ng, A. N., Markov, I. L., Aggarwal, R., and Ramachandran, V.: Solving hard instances of floorplacement. In Proceedings of the 2006 international symposium on Physical design, pages 170–177. ACM, 2006.

28. Nakamura, N., Oda, N., Soda, E., Hosoi, N., Gawase, A., Aoyama, H., Tanaka, Y., Kawamura, D., Chikaki, S., Shiohara, M., et al.: Feasibility study of 70nm pitch cu/-porous low-k d/d integration featuring euv lithography toward 22nm generation. In Electron Devices Meeting (IEDM), 2009 IEEE International, pages 1–4. IEEE, 2009.

29. Prathima, A., Bailey, K., and Gurumurthy, K.: Impact of device parameters of triple gate soi-finfet on the performance of cmos inverter at 22nm. International Journal of VLSI Design & Communication Systems, 3(5):79, 2012.

30. Nakamura, N., Takigawa, Y., Soda, E., Hosoi, N., Tarumi, Y., Aoyama, H., Tanaka, Y., Kawamura, D., Ogawa, S., Oda, N., et al.: Design impact study of wiring size and barrier metal on device performance toward 22 nm-node featuring euv lithogra-

CITED LITERATURE (Continued)

phy. In Interconnect Technology Conference, 2009. IITC 2009. IEEE International, pages 14–16. IEEE, 2009.

31. Goldstine, H. H. and Goldstine, A.: The electronic numerical integrator and computer (eniac). Mathematical Tables and Other Aids to Computation, pages 97–110, 1946.

32. Kilby, J. S.: Invention of the integrated circuit. Electron Devices, IEEE Transactions on, 23(7):648–654, 1976.

33. Wolfe, T.: the tinkerings of robert noyce. Esquire (December 1983), pages 346–374, 1983.

34. Moore, G. E. et al.: Progress in digital integrated electronics. IEDM Tech. Digest, 11, 1975.

35. Donze, R., Erickson, K., Hovis, W., Sheets, J., Tetzloff, J., and Zumbrunnen, L.: Fin fet diode structures and methods for building, September 17 2004. US Patent App. 10/944,624.

36. Fried, D. M., Nowak, E. J., Rainey, B. A., and Sadana, D. K.: Fin fet devices from bulk semiconductor and method for forming, November 4 2003. US Patent 6,642,090.

37. Chang, J., Guillorn, M., Haensch, W., and Saenger, K.: Fin field effect transistor devices with self-aligned source and drain regions, November 26 2013. US Patent 8,592,280.

38. Cea, S., Mehandru, R., Shifren, L., and Kuhn, K.: Wrap-around contacts for finfet and tri-gate devices, June 23 2011. US Patent App. 12/646,651.

39. Bernstein, K., Sleight, J., and Yang, M.: Hybrid crystal orientation cmos structure for adaptive well biasing and for power and performance enhancement, December 8 2009. US Patent 7,629,233.

40. Adkisson, J., Dunbar, T., Gambino, J., and Leitch, M.: Graphene field effect transistor, June 20 2013. WO Patent App. PCT/US2012/063,508.

41. Jego, C., Casseau, E., and Martin, E.: Interconnect cost control during high-level synthesis. In Proceedings of Design Circuits & Integrated Systems Conference, pages 507–512, 2000.

CITED LITERATURE (Continued)

42. Krishnan, V. and Katkoori, S.: Clock period minimization with iterative binding based on stochastic wirelength estimation during high-level synthesis. In VLSI Design, 2008. VLSID 2008. 21st International Conference on, pages 641–646. IEEE, 2008.

VITA

Name: Enzo Tartaglione

Education: Master of Science in Electrical and Computer Engineering

University of Illinois at Chicago, USA, July 2015

Master of Science in Electronic Engineering

Politecnico di Torino, Italy, July 2015

Alta Scuola Politecnica student

Italy

Bachelor of Science in Electronic Engineering

Politecnico di Torino, Italy, October 2013