IWLAS'97

Title:

Unified evaluation model for interconnection schemes used in behavioral synthesis

Authors:

Wander Oliveira Cesário
(presenter of the paper)
Email: Wander.Cesario@imag.fr
Laboratory TIMA-CMP
46 avenue Felix Viallet
38031 Grenoble Cedex FRANCE
Phone number: (+33) 4 76 57 44 64
Fax Number: (+33) 476 47 38 14

Polen Kission
Email: polenk@anacad.fr
Anacad
11A chemin de la Dhuy
38240 Meylan

Ahmed Amine Jerraya
Email: Jerraya@imag.fr
Laboratory TIMA-CMP
46 avenue Felix Viallet
38031 Grenoble Cedex

Philippe Guillaume
Email: pguillaum@rhin.imag.fr
Laboratory TIMA-CMP
46 avenue Felix Viallet
38031 Grenoble Cedex
FRANCE

Topic of interest is High Level Synthesis.

Unified evaluation model for interconnection schemes used in behavioral synthesis

W.O. Cesário, P. Kission, P. Guillaume and A.A. Jerraya

TIMA Laboratory
46, Av. Félix Viallet 38031 Grenoble Cedex France
e-mail: Wander.Cesario@imag.fr
Fax: (+33) 4 76 47 38 14

Abstract

This paper introduces a new methodology to evaluate datapath interconnection schemes at the behavioral level. This evaluation model is validated by comparison with RTL synthesis evaluation results. It has been implemented and used for architecture exploration in order to select the best interconnection scheme to use. A generic datapath model capable of representing a variety of interconnection schemes is presented. The efficiency of each scheme depends on the application under synthesis. In general, mux-based interconnections are more efficient for resources with low sharing factors. Contrarily, designs that employs a large controller or a large instruction set to enable high resource sharing factors give better results with bus-based interconnections. Experiments have shown that comparison using a behavioral level evaluation model is enough for quality measurement and qualitative classification.

I. Introduction

I.1 Motivation

Modern sub-micron technologies have established new compromises for the abstractions and methodologies commonly used in high-level synthesis (HLS). Deep sub-micron design must deal with interconnection delays greater than inside cell delays, which means that physical level information is becoming too important to be overlooked by high-level synthesis algorithms [1]. For instance, datapath interconnection styles and related synthesis/evaluation algorithms are directly affected by the availability of several routing layers.

Datapath generation has a crucial importance in all modern design methodologies. Experimental results using HLS tools [2] and standard cell current technologies show that datapath cell area corresponds to nearby 80% of the total cell area. Similar proportion could be verified on most current HLS tools. Moreover, about 70% of the circuit area is due to interconnections (routing wires). Of course, deep sub-micron technologies will considerably reduce interconnection costs. While cell and wiring area costs are decreasing continuously, the relative cost of datapath connection elements (multiplexes/tri-state drivers) is increasing. It means that new optimization constrains are being created for datapath synthesis. Old optimization practices, as minimizing bus number when tri-state driver area overhead is significantly larger, could make no sense nowadays. Until now, the interconnection optimization problem did not receive the attention it deserves.

Communication synthesis is tightly related with functional-unit and register allocation/binding [15][7]. Some HLS tools solve these problems simultaneously [3][4] and others sequentially [8][10]. Most existing HLS tools employ multiplex based interconnections [5][6]. Bus based interconnections are very popular with designers but not very used in HLS due to lack of heuristics capable of handling large examples. Of course, many studies have considered bus-based interconnections for the datapath [8][9][12]. Finally, very few works reported on mixed bus and mux architectures [3][13]. Interconnection optimizations are more effective when done in conjunction with functional unit and register allocation/binding tasks. The best interconnection scheme will depend of the application. This means that, in order to offer better design space exploration, the HLS tool must provide the designer a way to investigate diverse interconnection schemes.

I.2 Objectives

The purpose of this work is to present methods and tools for modeling, synthesis and evaluation of mux and bus-based datapath interconnection schemes. The goal is to provide a way to compare these schemes even on large designs. In order to achieve this objective, some steps are needed:

1. Datapath modeling adapted to both interconnection schemes.
2. Metrics allowing objective comparison between both schemes.

This paper presents the definition of a unified datapath model for the bus-based, mux-based and mixed bus-mux datapath interconnection schemes. Furthermore, we present the comparison between the most used interconnection schemes for some examples and we discuss the advantage of each approach. Our behavioral level evaluation model is validated by comparison with RTL evaluation results. It shows that the comparison at behavioral level is enough for quality measurement and qualitative classification.

I.3 Related work

Datapath modeling and evaluation on most HLS systems is based on a single interconnection style. A bus-based datapath interconnection model is used in [8]. The communication synthesis algorithm builds a graph where each node represents a point-to-point connection. There will be an edge linking two nodes if, and only if, the two associated connections will never be used simultaneously. After clique partitioning, one bus is allocated for each clique. This approach does not address switch minimization. One possibility for finding the optimal solution in terms of buses and switches is to use a dynamic programming approach as described in [9]. It is based on an efficient solution space enumeration using an auxiliary connection graph. Graph information is used to reduce the search space after each bus allocation. Even though, it remains an expensive solution on very large designs.

Special datapath models consisting of bus segments connected with switches are presented in [12]. In this particular case, the communication synthesis procedure must take into account the relative position of the datapath components. Register and functional-unit input binding could be exploited to find a better solution. The datapath produced has a small bus number but uses an excessive number of switches.

A further possibility is a mixed style where bus and multiplexer interconnections coexist. It is used in MABAL [13] and Cathedral-II [16]. MABAL uses a Greedy constructive approach that starts with an empty datapath and incrementally adds functional units, storage elements and interconnection units as necessary. Cathedral-II has a more restricted bus-mux interconnection model, where interconnections inside EXUs are mux-based and global connections are bus-based. For communication synthesis, it has an iterative busmerger that tries to reduce the bus number by merging the sporadically used buses with the most used ones. These algorithms lack switch minimization directives.

Elf [10] does a less restrictive bus-mux interconnection synthesis. It is based on the generalized interconnection model presented in [11]. The algorithm uses a greedy clique-partitioning algorithm to deal with a multi-level abstract bus model. Except for some special cases, there is not much interest in using multi-level bus interconnections. Normally, the delay is greater and there are more switches that in the single-level bus architecture. A major limitation pointed in [11] is that interconnection delays are not explicitly considered in the interconnection cost. In [14] is shown how to use RT-level floorplan information to take into account interconnect delays in HLS algorithms. Nevertheless, there is no good behavioral-level interconnection delay estimator yet.

What is missing is an objective comparison of datapath interconnection styles using realistic examples. It is always possible to obtain optimal solutions given a small design example or benchmark. But on large designs we need to use fast heuristics to evaluate each architectural possibility. In this case, the choice of the architectural style that better matches the data flow requirements is essential for the overall design optimization.

I.4 Structure of the paper

In the next section, we will present a unified datapath model able to handle both mux-based and bus-based interconnection schemes. The core of the paper is section IV, which presents the results of a comparative study of mux and bus-based datapath interconnection styles using a variety of examples.

II. Datapath model

In this section, we present a flexible and generic architecture representation that abstracts several architectural styles. Conceptually, a datapath may be defined by two aspects: the internal structure and the functional organization. The former corresponds to the types of allowed units. The latter is called transfer model and it is related to the types of accepted transfers, how they are executed and how communication with the controller is performed in both directions.

II.1 Datapath structural model

The datapath architecture is defined as a set of components associated to an interconnection topology. The basic conceptual architecture organization is depicted on the Figure 1.

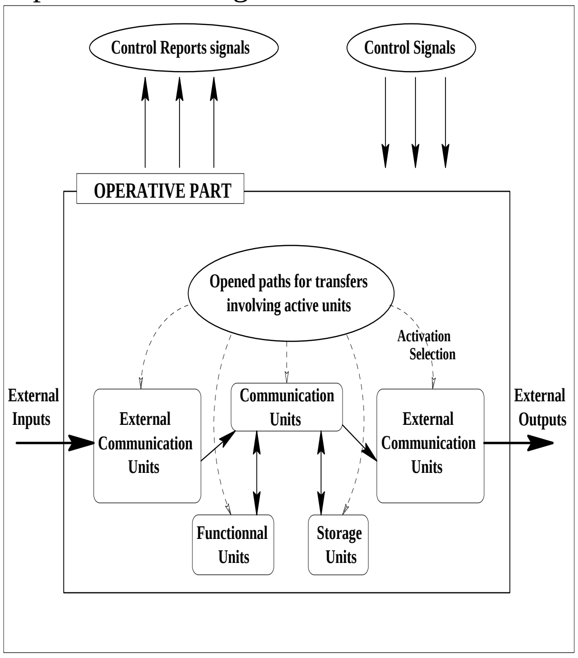
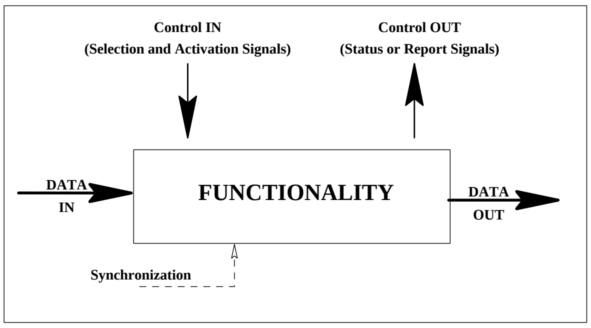


Figure 1: Conceptual organization of the datapath architecture

Data exchanges between units are done using communication units. Transfers follow the paths opened by the control signals. They obey to a transfer model that includes the instruction set, defining the dependencies between source and destination for a transfer, and the generation rules that define the realization of a transfer. This point will be discussed latter in this section. Each unit within the datapath is more or less built on the generic scheme represented on Figure 2.

Figure 2: Generic model of a datapath unit

Roughly, each unit manipulates data



and transforms, transmits or stores these data. Therefore, each type of unit will differ from the others principally by its functionality, while the connection with the surrounding components will be done following similar schemes. The main components of the datapath are [15][2]:

1. The functional units (**FU**s) aimed to execute the operations of the behavioral description.
2. The storage units (**SUs**), such as registers, register files, RAMs and ROMs, are the components which can store the variable values or provide constants specified in the initial behavioral description.
3. The external communication units (**ECUs**) link the operative part with the external world.
4. The communication units (**CUs**) or transmission units are used to control data transfers between other units, through nets and buses which are the supports of these data transfers. Figure 3-a depicts the generic form of a transmission unit. The most used types of communication units are switches and multiplexers (Figure 3-b and Figure 3-c respectively) but this model is not restricted to them.

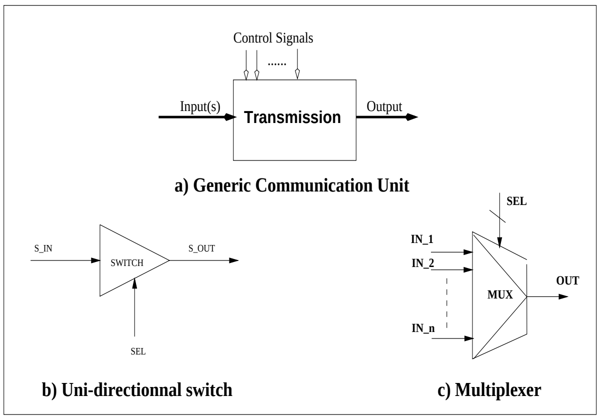


Figure 3: Basic communication units' structure

II.2 Datapath transfer model

At each basic clock cycle, the datapath executes an instruction, itself possibly composed of a set of parallel transfers. The computation power of a datapath can be fixed by the transfer model associated to it, and by the amount of parallelism it allows [17]. The "instruction set" may be defined by the set of transfers that can be executed by the datapath. Therefore, in addition to the structural information, the datapath definition must include a transfer model.

Transfers are defined by three components: the source, the sink or destination and the path. During a transfer, data are sequentially exchanged, transformed, or both. This definition involves a transfer path between source and destination units; a path can be composed of others datapath units such as **CUs**, but also **FUs**. A transfer can therefore be represented as a graph, where nodes are the datapath components, and the edges are the physical links between them. Each transfer may then be decomposed into several atomic data exchanges from the source unit to the destination unit. In order to fix how to distinguish source and destination units from path units, one can define the source unit as the first unit involved after the beginning of the clock cycle, and the destination unit as the last unit reached before the end of the transfer's last clock cycle. If we consider the previous component sets, the general form of a transfer is the following:

TRANSFER

Source ⇒ *Path* ⇒ *Destination*

[***ECU***, ***SU***, ***FU***] → ***CU*** → { [***FU***, ***CU***] → } [***ECU***, ***SU***, ***FU***]

where:

[a,b]*: means select one a or b*

{a}*: means any repetition of the string a*

It is obvious that this model may apply for bus-based and mux-based interconnection styles. This model may be easily extended to other existing models such as multi-level mux model or mixed bus-mux model. In the following, we will consider the most used models: single-level mux-based and the single-level bus-based.

II.3 Datapath styles

For the bus-based interconnection model, internal connections between units are made up of buses and switches. If we assume the use of unidirectional switches only, some basic transfers allowed by the interconnection model are detailed on Figure 4. In this figure, ***[Sw]*** represents an optional switch. Complex transfers can be realized by combining several basic transfers (see Figure 4-b).

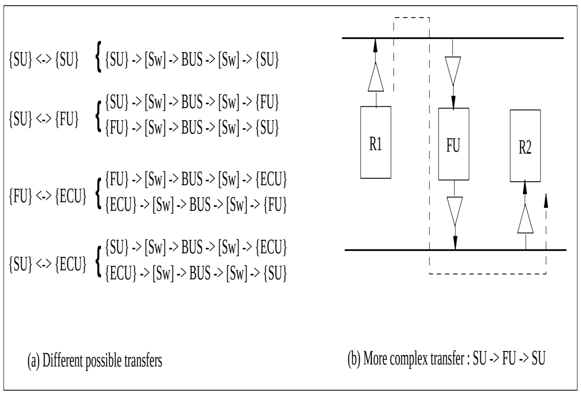

{SU} <-> {SU}
{SU} -> [Sw] -> BUS -> [Sw] -> {SU}
{SU} <-> {FU}
{SU} -> [Sw] -> BUS -> [Sw] -> {FU}
{FU} -> [Sw] -> BUS -> [Sw] -> {SU}
{FU} <-> {ECU}
{FU} -> [Sw] -> BUS -> [Sw] -> {ECU}
{ECU} -> [Sw] -> BUS -> [Sw] -> {FU}
{SU} <-> {ECU}
{SU} -> [Sw] -> BUS -> [Sw] -> {ECU}
{ECU} -> [Sw] -> BUS -> [Sw] -> {SU}
R1
FU
R2
(a) Different possible transfers
(b) More complex transfer : SU -> FU -> SU

Figure 4: Datapath model for a single-level bus-based architecture

A similar datapath transfer model for the mux-based interconnection model is shown in Figure 5.

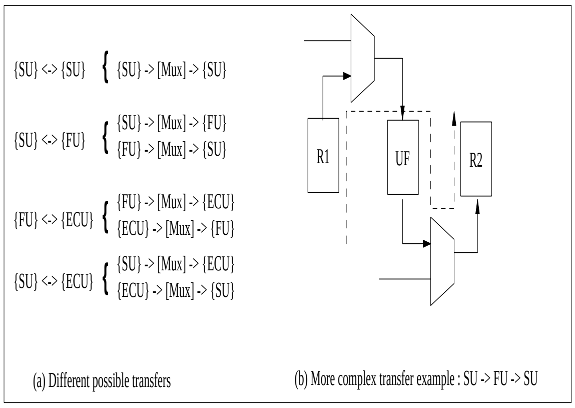


Figure 5: Datapath model for a single-level mux-based architecture

Communication units are abstractions of physical components. From a conceptual point of view, the communication unit is an object that can execute a data transfer. This object has a set of properties associated with it: bit size of its inputs and outputs, the delay of transmission, the availability on each synthesis cycle, etc.

Communication synthesis could be treated in a more general way when the physical details of the communication units are abstracted. In addition, it provides more flexibility over the type of connection elements used by the synthesis tool. This generalized model could accommodate mixed bus-mux architectures, interconnection through variable bit-width paths and through functional unit's internal paths.

II.4 Datapath estimation

The datapath area is divided in two parts by traditional estimation algorithms: active area and wiring area. Active area is the cell area of functional units (**FUs**), storage units (**SUs**), external communication units (**ECUs**) and communication units (**CUs**). Wiring area corresponds to the routing space. Our experiments have shown that wiring area is approximately a constant factor of the cell area. The value of this factor depends of the technology and the layout routing style. From these considerations, circuit's area estimation could be done using formula (1).

$$S_T = \alpha \cdot (A_D + A_C) \qquad (1)$$

where:
***S_T**: circuit's area* ***A_D**: datapath cell area*
***A_C**: controller cell area* ***α**: routing area factor*

From the information presented in synthesis library, it is easy to obtain an accurate estimation of datapath active area by adding the area values of the each datapath cell, as shown in formula (2).

$$A_D = \sum_i A_{FU}(i) + \sum_j A_{SU}(j) + \sum_k A_{ECU}(k) + \sum_l A_{CU}(l) \qquad (2)$$

where:
***A_D**: datapath cell area*
***$A_{FU}(i)$**: FU(i) cell area* ***$A_{SU}(j)$**: SU(j) cell area*
***$A_{ECU}(k)$**: ECU(k) cell area* ***$A_{CU}(l)$**: CU(l) cell area*

As will be shown later (section IV), the routing area may be estimated as a constant factor (around 70%) of the cell area. The estimation of the controller is much more difficult, since it is described as a FSM with symbolic state encoding. In our case, we use a technique similar to the one presented in [15] to estimate the size of the controller. It will be also shown (in section IV) that the size of the controller is not a determinant factor for the selection of the datapath interconnection scheme.

III. Communication synthesis for the datapath

This section outlines the communication synthesis algorithms used to compare the datapath interconnection schemes. Performance considerations for these algorithms are beyond the scope of this paper.

Communication synthesis aims to transform the abstract communication units of the datapath model defined in the previous section into physical components as wires, buses and multiplexers. New datapath elements will be specially created to realize interconnections. The elements created by resource allocation could be used with routing function if there is no usage conflict. This section outlines two algorithms for the synthesis of bus-based and mux-based interconnections. They have been implemented and integrated in the HLS tool AMICAL [2].

For mux-based interconnections, we used a simple communication synthesis algorithm, it generates multiplexers in an as-need basis. This means, every time a data destination is shared by more than one data source a multiplex is introduced in the transfer's path. This straightforward algorithm is used by most HLS tools [15]. One can note that for large designs this kind of algorithm may lead to very large multiplexers and specific optimizations may be needed in order to tradeoff area and speed.

For bus-based interconnections, we developed a new communication synthesis algorithm that uses a bus-merge approach, though the goal is to reduce the switch number and not the bus number. Figure 6 shows connection diagrams where the set of transfers executed by the datapath are represented as arrows between the set of data sources, the set of buses and the set of destinations. The algorithm is interactive, it starts with an initial solution that has one bus for each data destination as shown in Figure 6-a. In this solution, there are switches only from the data sources to the shared buses.

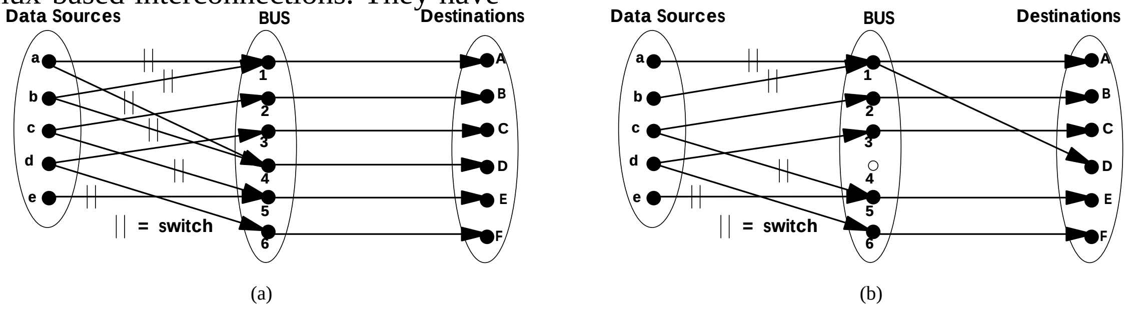


Figure 6: Bus-merge example

The time concept remains implicit in Figure 6. The set of data sources is linked to the set of data destinations through the set of buses. If in a given cycle we need to do *n* transfers, we will need *n* buses because the transfers are done in parallel. The minimum size of the set of buses is equal to the maximum value of *n*. Switches must be added in two cases:

1. for the data sources that share a bus.
2. for the data destinations connected to more that one bus.

Moreover, they can be avoided in the following cases:

1. for the buses connected to only one data source.
2. for the data destinations connected to only one bus.

The bus-merge heuristic starts with a reduced number of switches but a large number of buses (see Figure 7). After this initial allocation, the buses are merged to get a solution with fewer switches. For example, Figure 6-b shows that two switches could be eliminated by merging buses **1** and **4**. To avoid local minimums the merge process is done while it is possible, even if sometimes it increases the switch number. Intermediate solutions are all stored. Due to efficiency considerations, the set of impossible merges is stored for the next algorithm iteration. The number of switches can be calculated using formula (3). It must be evaluated for all the possible bus merges. When no more merge is possible or the minimum bus number is reached the merge process ends and the best solution is picked.

$$Sn = \sum_{i=1}^{BN} ds_i + \sum_{j=1}^{DDN} db_j \qquad (3)$$

where:
***Sn**: switch number*
***BN**: bus number*
***ds$_i$**: number of data sources for bus **i** (if greater then 1 else is 0)*
***DDN**: number of data destinations*
***db$_j$**: number of buses for the destination **j** (if greater then 1 else is 0)*

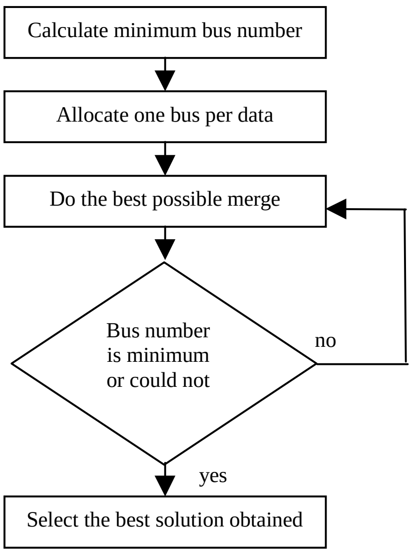

Calculate minimum bus number
Allocate one bus per data
Do the best possible merge
Bus number is minimum or could not
no
yes
Select the best solution obtained

Figure 7: Bus-merge heuristic

The communication synthesis algorithms presented in this section are quite fast. They are able to handle large designs within a reasonable CPU time, i.e., a few seconds on a SPARC 20.

IV. Comparative study

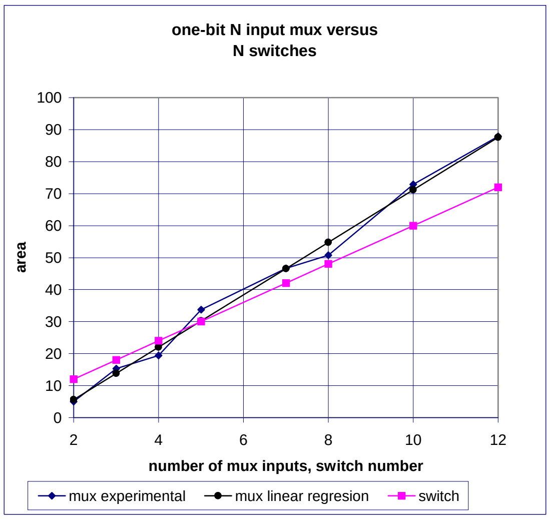
In this section, bus and mux-based datapath interconnection schemes are evaluated and compared to determine which solution is most suited for a given type of application. Ideally, this evaluation must be done at the behavioral level since we are targeting large designs. The behavioral-level area evaluation model presented in section II.4 will be used.

What makes the difference between the bus-based and mux-based interconnection styles is the area of communication units (**CUs**) and inter cell routing. The performances (area, delay) of multiplexes and switches are discussed in

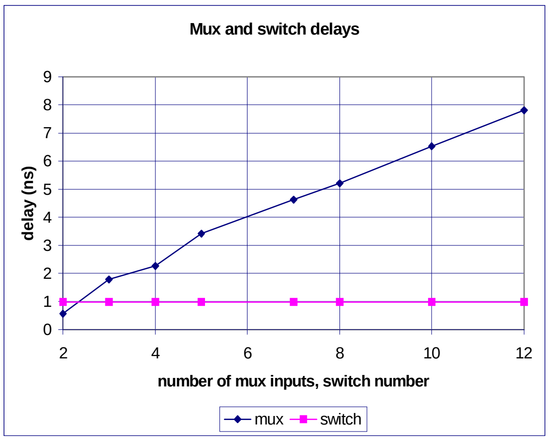
Figure 8. These results were obtained using a 0.8-micron two metal layers standard cell technology and industrial standard RTL synthesis tool. A two input multiplexer size is smaller than two switches for this technology. But for five or more multiplexed inputs this relation is reversed, i.e., the equivalent switch area is smaller than the multiplexer area.

Figure 8-b supposes bus-based interconnections; the worst cell delay introduced by **CUs** is that of going through two switches. While this delay remains fixed as the number of multiplexed inputs increase, the delay introduced by the multiplexers will grow proportionally. Consequently, as the number of multiplexed inputs goes higher the bus-based interconnection scheme becomes more interesting than the mux-based one. In our behavioral level evaluation model, technology dependent data (as showed in

Figure 8-a) is used to calculate **CUs** cell area using linear regression equations.



(a)



(b)

Figure 8: Mux and switch comparison

IV.1 Results and evaluation

Data on this section comes from HLS tool AMICAL and standard RTL synthesis tool. RTL synthesis results were used to validate our behavioral evaluation model. Three design examples and three worst case designs were synthesized. The design examples are the GCD (greatest common divisor), a fixed point calculation unit (FPU) that do addition, subtraction, multiplication and division and the bubble sort algorithm. The worst case designs are called ***RegN*** (Reg5, Reg6, and Reg11) meaning that they have *N* registers and each register exchange data with all others. When realized with mux-based interconnections, each register will have an (*N-1)*-input (4, 5 and 10 respectively) multiplexer in its input. This structure simulates very large designs like complex controllers and microprocessors with large instruction sets. In fact, it is often the case that the registers are normalized in this kind of design. Consequently, each register value could be transferred to all other registers. Table 1 is a compilation of the RTL synthesis results for these examples using both datapath interconnection styles.

Switch, equivalent two-input multiplexes and net numbers are shown only to give an idea of the interconnection requirements of each design. Cell area and speed (critical path) come from the structural information obtained after RTL synthesis.

	mux-based solution				**bus-based solution**				**mux ÷ bus cell area**
	# 2 input mux	**# nets**	**critical path**	**cell area**	**# switches**	**# nets**	**critical path**	**cell area**	
GCD	5	125	9.17	220	12	154	8.34	340	65%
Bubble	21	234	14.73	538	25	254	14.72	811	66%
FPU	25	801	43.42	2329	26	876	31.77	3111	75%
Reg5	22	147	8.54	438	9	121	4.64	405	108%
Reg6	31	172	9.93	574	10	131	4.99	454	126%
Reg11	100	283	12.18	1530	15	181	6.76	698	219%

Table 1: Datapath RTL synthesis results

From Table 1, it is easy to see that there is no style good for all design examples. The mux-based style gives better results for the two first examples while the bus-based style is better for the three last examples, which are more control oriented. In general, datapaths with high average component sharing factor are more suitable to bus-based mapping. On the other hand, designs such as GCD, FPU and Bubble that have a low resource sharing factors achieve better results when mapped to a datapath with mux-based interconnections. In the case of the FPU design example, the mux-based solution gives a better result in terms of area while the bus-based solution provides a better solution in terms of delay.

In order to measure the liability of our estimation methods and to investigate the opportunity to use them for architecture exploration, we compared the results of our estimation tools with the results of RTL synthesis. The results are given in Table 2. For each example, we give the area estimation produced by RTL synthesis for the datapath, the controller and the routing. We give also the results produced by our estimation tool according to section II.4. The results are given for both mux-based and bus-based interconnection schemes.

	mux-based solution						**bus-based solution**						**behavioral**	
	RTL				**behavioral**		**RTL**				**behavioral**			
	datapath (% total)	**controller (% total)**	**routing (% total)**	**total**	**datapath (% total)**	**total**	**datapath (% total)**	**controller (% total)**	**routing (% total)**	**total**	**datapath (% total)**	**total**	**mux ÷ bus datapath**	**mux ÷ bus total**
GCD	220 (19%)	93 (8%)	829 (73%)	1142	196 (23%)	842	340 (21%)	101 (6%)	1151 (73%)	1592	310 (24%)	1268	63%	72%
Bubble	538 (17%)	313 (11%)	2233 (72%)	3084	499 (17%)	3012	811 (21%)	303 (8%)	2732 (71%)	3846	728 (18%)	3954	69%	79%
FPU	2329 (24%)	218 (3%)	7015 (73%)	9562	2223 (26%)	8639	3111 (26%)	245 (2%)	8758 (72%)	12114	3042 (26%)	11640	73%	75%
Reg5	438 (22%)	112 (6%)	1415 (72%)	1965	419 (22%)	1868	405 (24%)	114 (7%)	1178 (69%)	1697	395 (22%)	1763	106%	106%
Reg6	574 (23%)	116 (4%)	1834 (73%)	2524	523 (21%)	2508	454 (24%)	125 (7%)	1299 (69%)	1878	443 (21%)	2160	118%	120%
Reg11	1530 (21%)	369 (5%)	5267 (74%)	7166	1251 (16%)	7915	698 (21%)	339 (10%)	2330 (69%)	3367	686 (15%)	4635	182%	167%

Table 2: Area estimation results

Table 2 shows two important results of this work:

1. Datapath cell area estimation is quite accurate when compared to RTL synthesis estimation. The difference between the estimated area at the behavioral level and the area estimated at the RTL level is about 10%.
2. The datapath estimation may be used as an objective criterion for selecting architectural solutions during behavioral synthesis. The two last columns in Table 2 shows the ratio between estimations at behavioral level of mux-based and bus-based datapaths. They are very close to the ratio computed after RTL synthesis (last column of Table 1). The fact that the cell area of the datapath is sufficient for qualitative classification of the result of HLS synthesis may be explained by two facts. First, the routing area seems to be a constant factor in relation to the whole circuit (around 70%). Second, the size of the controller is almost the same for both datapath interconnection styles.

V. Conclusion

This paper presented a comparative study of bus and mux-based datapath interconnection schemes. It is based on the general datapath model presented on section II. This high-level synthesis model is designed to represent a large subset of architectural styles. It is clear that the choice of the interconnection scheme will greatly influence the efficiency of the synthesis result. Our methodology allows deciding at the behavioral level, which is the best scheme to use. This is essential for very large designs since RTL synthesis time could be an important part of the complete design cycle time.

The results prove that interconnection optimizations play a fundamental role on the overall design optimization. As the relative cost of communication units is increasing in respect to the other datapath cells, new communication synthesis algorithms as presented in section III are needed. They must minimize the number of communication units used in order to achieve a better architectural solution.

This study has shown that the efficiency of each interconnection scheme depends on the characteristics of the design under synthesis. For large datapaths with small controllers, the mux-based solution seems to be more effective. On the other hand, when the controller includes a large instruction set, the bus-based model seems to produce a better solution.

VI. Bibliography

[1] R. Camposano, Behavioral Synthesis, *Proc. 33th Design Automation Conference*, June 1996.

[2] A.A. Jerraya, H. Ding, P. Kission, and M. Rahmouni, *Behavioral Synthesis and Component Reuse with VHDL*, Kluwer Academic Publishers, 1997.

[3] J.Septién, D.Mozos, F.Tirado, R.Hermida, M.Fernández, Heuristics for Branch-and-Bound Global Allocation, *EuroDAC*, 1992.

[4] B. Landwehr, P. Marwedel, and R. Dömer. OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming. *EuroDAC*, 1994.

[5] P. Lippens, J. van Meerbergen, A. van der Werf, W. Verhaegh, B. McSweeney, J. Huisken, O. McArdle, PHIDEO: a Silicon Compiler for High Speed Algorithms, *EuroDAC*, 1991.

[6] D. Knapp, T. Ly, D. MacMillen, and R. Miller, Behavioral Synthesis Methodology for HDL-Based Specification and Validation, *Proc. 32th Design Automation Conference*, June 1995, pp. 286-291.

[7] Y. Nakamura, K. Oguri and N. Nagoya, Synthesis from Pure Behavioral Descriptions, in Camposano and Wolf Editors, *High-Level VLSI Synthesis*, Kluwer Academic Publishers, 1990.

[8] C.J. Tseng abd D.P. Siewiorek, Automated Synthesis of Data Path in Digital Systems, *IEEE Trans. on Computer-Aided Design*, Vol. 5, N°. 3, pp 379-95, July 1986.

[9] H.C. Torng and N.C. Wilhelm, The optimal interconnection of circuit modules in microprocessor and digital system design, IEEE Trans. on Comput. 26 (5) (1977) 450-45.

[10] E.F. Girczyc & J.P. Knight, An ADA to standard cell hardware compiler based on graph grammars and scheduling, *Proc. of ICCD*, 1984.

[11] T.A. Ly, W. Loyd Elwood, and E.F. Girczyc, A generalized interconnect model for data path synthesis, *Proc. 27th Design Automation Conference*, June 1990, pp. 168-173.

[12] C. Ewering, Automatic high level synthesis of partitioned busses, *Digest of Technical Papers of the Int. Conference on Computer Aided Design*, November 11-25, 1990, pp. 304-307.

[13] K. Küçükçakar and A.C. Parker, Data Path Tradeoffs using MABAL, *Proceedings of the 28th Design Automation Conference*, pp. 511-516, 1990.

[14] F. J. Kurdahi, P. Jha, C. Ramachandran, and N. Dutt, Towards More Realistic Physical Design Models for High Level Synthesis, Workshop on Synthesis And System Integration of MIxed technologies, 1993.

[15] D. Gajski, N. Dutt, A. Wu, and Y. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, Boston, Massachusetts, 1992.

[16] J. Vanhoof, K. V. Rompaey, I. Bolsens, G. Goossens, and H. De Man. *High-Level synthesis for Real-Time Digital Signal Processing*, Kluwer Academic Publishers, 1993.

[17] F. Anceau, The Architecture of Microprocessors, Addison-Wesley Publishing Company, 1986.