COMPACT HIGH SPEED CMOS CROSSBAR FOR SWITCHING FABRICS

B. Zerrouk, V. Reibaldi, D. Tranoudis and J.L Desbarbieux

Institut Blaise Pascal, URA 818-CNRS UPMC
Laboraoire MASI,
Université Pierre et Marie Curie
Couloir 55/65 - Etage 2,
4, place Jussieu 75252 PARIS CEDEX 05, France.
e-mail : zerrouk@masi.ibp.fr

Abstract: A crossbar is the central element in the design of a non blocking switching fabric. The most common implementation of a crossbar proposed in the litterature uses a non blocking multistage network based on elementary $k \times k$ switches where usual values of k are small (2 to 4). Generally, such elementary switches allow only unidirectional routing. In this paper we present the design semi-custom arbitrary bidirectional $n \times n$ synchronous crossbar macro-block that can be used both for complete or multistage emulation crossbar based router architectures. Several points related to the feasibility are pointed out: logical design, layout generation and performances.

1 Introduction

It is known that networking of computing systems has been and still is an important research and industrial activity. Examples of networking range from sample standard bus coupling for few processors based systems, to broad band ISDN coupling of geographically distant computer systems [8]. Novel architectures such as massively parallel computers or the integrated service ATM based distributed systems require suitable high performance communication support offering a high bandwidth for packet exchanges. In these systems the design of switching fabrics is necessary for routing packets from one terminal to an other.

The reader can find a large litterature on the design of switching fabrics. Non exhaustive examples are: the IBM Vulcan switch used in the SP-2 system [7], the ST-C104 switch [4], the RCube router prototype [9] developed at Pierre&Marie Curie University, the TIC chip from Sun MCC [3] and the Reliable Router developed at MIT [1]. Even if the overall architectures of such components are well understood, implementation details are subject to trade-offs that depend on the available technology and the expected performance for a given application. In a common device implementation several aspects must be specified, for example linking technology, buffering strategy, flow control technique, routing scheme, switching and contention resolution. This contribution deals with the last two aspects.

Switching in a router consists of circuit elements that enable path setting from any input port to any output port. A way to achieve this is the use of a complete crossbar or its emulation by mean of smaller crossbar elements arranged in a suitable (non-blocking) network. Different crossbar design styles may be adopted depending on architectural choices influenced by the implementation technology. These architectural choices affect mainly the crossbar controller and the implementation of the switching circuitry. Switching circuitry is simply the input to output multiplexing. Multiplexing may be done by mean of tristates or multiplexors [6]. In general, the control part consists of an allocation logic that controls the switching circuitry and an arbitration mechanism that resolves contention when two or more inputs of the router have to be switched to the same output. For concurrent self routing switches, decentralized control is preferable even for large crossbars [2].

In this paper we present a new design of arbitrary bidirectional $n \times n$ synchronous crossbar. The crossbar can be used both for complete or multistage emulation crossbar based router architectures. The idea is to provide a systematic method for automatic (or semi-automatic) layout generation of a crossbar as a compact and versatile ASIC macro-block. The target technology is CMOS but with some process independency making the design portable []. We have chosen CMOS technology because of its compactness, low cost and scalability. Submicronic CMOS is still a promising technology for the design of high performances communication application as demonstrated by the Gigabit serial link technology presented in [5].

The remaining sections are organized as follows : In Section II we present the model of the crossbars we are dealing with. The context and the different elements of the crossbar are specified. In Section III we specify the allocation (or control) units. In Section IV we give details of the arbitration logic used for contention resolution as defined in Section III. In Section V, we propose a layout placement and routing strategy for the switching part of the crossbar. Finally performance evaluation for generated crossbars is given in section VI.

2 Crossbar Model

We consider n independent identical input units (or ports) and n independent identical output units (or ports), where n is a power of 2 greater than 4. We number these units respectively from 0 to $n-1$.

Each output unit j ($0 \leq j \leq n-1$) is directly linked to a unique output bus. Each input unit i ($0 \leq i \leq n-1$) has an unique entry to access any output bus. We call this entry an input bus. Between each input bus i and output bus j there is a crosspoint (i, j) that consists of a switch. A bus may include data lines and all control lines necessary for a defined transfer protocol between an input unit to an output unit. Each output bus j has its own central allocation unit j. Figure 1 shows an example of a 2×2 crossbar that connects 2 input units to 2 output units. All the elements within the dashed rectangle constitute the crossbar.

Each input unit can send a separate *request* signal to any allocation unit, at any time, asking for the allocation of the corresponding output bus. An input unit generates one (or several) request at a time. Several inputs may request the same output bus at the same time. An allocation unit ensures that its corresponding output bus is allocated to an unique input unit. In the case of several requests for the same output bus one, and only one, of the requests is selected by mean of an arbitration logic. This logic uses a turn around priority configuration that avoids *starvation*: any persistent request is eventually selected given that any input to output transfer takes a finite time.

When an allocation unit j has selected the request from the input unit i, it sends a backward *grant* signal to input unit i and, simultaneously, connects input bus i to output bus j by closing

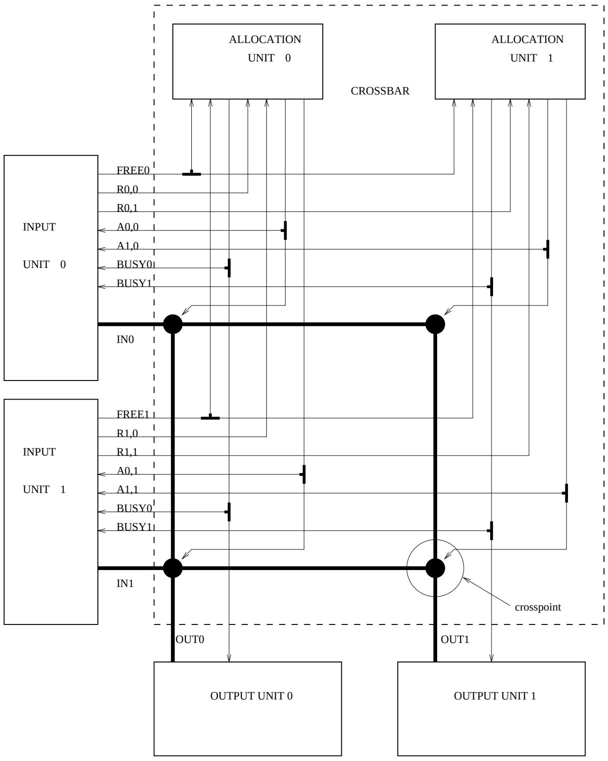

Figure 1: Overview of the crossbar.

the switch (i, j). It broadcasts a *busy* signal to all the input and output units. When the transfer is finished, input unit i broadcasts a *release* signal to all the allocation units. The allocation unit j, releases the output bus j by opening the switch (i, j). Output j is then ready to be allocated for another transfer. The list of signals at the interface of the crossbar is given in Table 1. We note that the list of signals and their functionalities is not exhaustive. It is simple to extend it to fit other requirements without altering the ideas we present here.

Signal Name	Function	Active logic level
$r_{i,j}$	Request signal of input i for allocation unit j	1
$a_{j,i}$	Allocation/Grant from the allocation unit j to the input i	1
	Controls the switch (i, j)	
$busy_j$	Status indicating wether ouput j is busy or not	1
in_i	Input bus i	
out_j	Output bus j	
$free_i$	Release signal from input i	1
$reset$	Reset signal	1
$clock$	Clock signal	

Table 1: Legend of the Crossbar Interface Signals

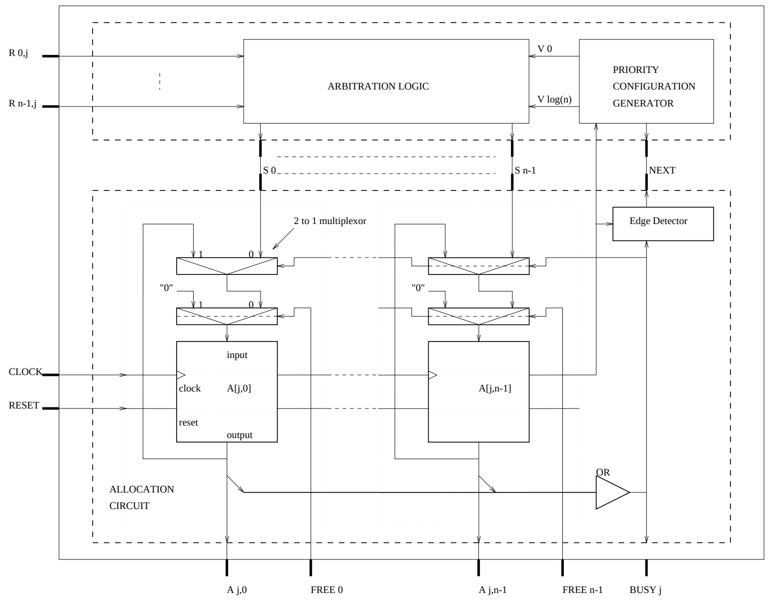

Figure 2: Allocation Unit j

3 Allocation Unit

The structure of an allocation unit consists of three parts: an arbitration circuit, a priority configuration generator and an allocation circuit (see Figure 2).

The priority configuration generator is a $\log(n)$ bit counter (or a pseudo random generator) that is increased if and only if the signal *next* is high. The arbitration logic serves to arbitrate between several requests for the same output bus. Its inputs are the outputs of the configuration generator and the requests signals $r_{i,j}, i = 0, \cdots, n-1$. When a request $r_{i,j}$ is selected, a corresponding selection signal s_i is set to 1 otherwise it stays at 0.

The allocation circuit contains an n bit allocation register a_j (we use without ambiguity the same name as for grant signals). Table 2 gives the important signals of the allocation unit j as shown in figure 2 .

Signal Name	Function
s_i	Selection signal of the request from input i
$a_{j,i}$	Allocation/Grant Signal
	It is the output of bit i of a_j
$free_i$	Release signal from input i
	When active it set bit i of a_j to 0
$busy_j$	Logical OR of all the bits of a_j
$clock$	Clock signal
$reset$	Reset signal
$next$	Priority Configuration Change Signal

Table 2: Signals Of The Allocation Unit j

The initial state (after a reset) of the allocation circuit is such that each bit of the register a_j is set to 0. The signal $busy_j$ is then at 0 which enables inputs to the allocation register.

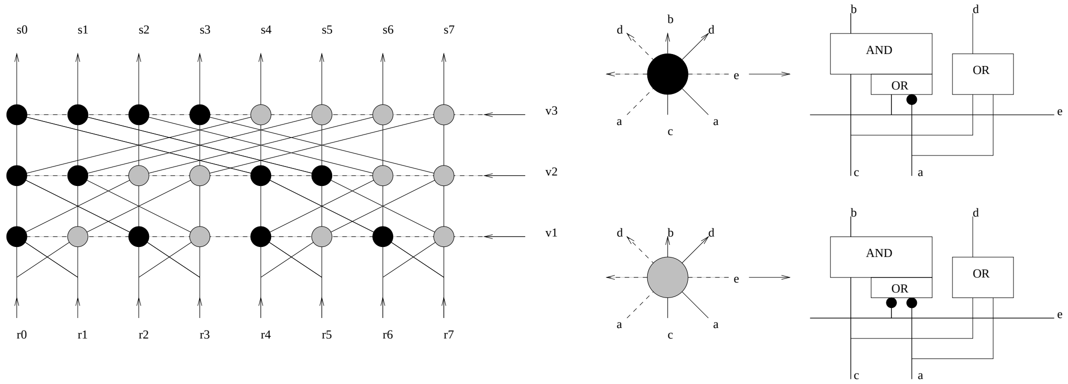

Figure 3: Possible Implementation Of The Arbitration Tree

At this state and after the active arbitration requests , only one bit of the allocation register is set to 1. The signal $busy_j$ passes to 1 and locks the allocation register. This ensure mutual exclusion. A rising edge of the busy signal triggers the generation of a one clock cycle pulse on the *next* signal (this is done by an edge detector, see figure 2), and increments the priority configuration counter.

4 Arbitration Logic

The arbitration logic is based on a priority binary tree. We describe here the algorithm of the logic.

Let $\{r_0, \cdots, r_{n-1}\}$ be n requests. The goal is to define a priority scheme in order to elect one request. For simplicity we consider that n is a power of 2. We construct a n complete binary tree T where leaves, are non-valued vertices representing the requests (for clarity we consider a standard representation of a tree where leaves are placed left to right according to the index). The intermediate vertices, are $0-1$ valued. We call this kind of tree a P-Tree.

$0-1$ valued vertices in a P-Tree specify the priority relation between nodes as follow:

Let v be any valued vertex of T. We note L_v (resp. R_v) the left (resp. the rigth) subtree of T rooted by the left (resp. right) son of v. Priorities are defined recursively for any requests $r \in R_v$ and $r' \in L_v$ as follow : $r' > r$ if $value(v) = 1$ otherwise $r > r'$. Note that there are $n-1$ valued vertices in a complete binary tree and $\log(n)$ levels. Hence, it is easy to show that only $\log(n)$ bits (one bit per valued vertices level) are needed to specify a complete suitable priority configuration. Indeed, a necessary property we need is that for each request it is possible to generate a priority configuration with highest priority for that request. Remember that priority configuration bits are generated by mean of a $\log(n)$ counter. The corresponding circuit is of logarithmic time complexity. As an example, the set of priority configurations for 4 requests are:

$$r_0 > r_1 > r_2 > r_3$$

$$r_1 > r_0 > r_3 > r_2$$

$$r_2 > r_3 > r_0 > r_1$$

$$r_3 > r_2 > r_1 > r_0$$

To implement the arbitration logic we need the set of n requests and the $\log(n)$ configuration bits as inputs, and outputs which identify one of the active requests to be satisfied. Outputs are defined simply by associating to each request $r_i, i = 0 \cdots n-1$ a boolean s_i where :

$s_i = 1$ means that r_i is elected.

$s_i = 0$ means that r_i is not elected.

Let $\{v_1, \ldots, v_{\log n}\}$ be the set of valued nodes on the path from a request r_i to the root in the P-TREE. The equation of a $s_i, i = 0, \cdots, n-1$ is formulated by :

$$s_i = r_i \wedge \bigwedge_{j=1}^{\log n} \overline{f_j \wedge (\vee_{r \in g_j} r)}$$

where f_j is a boolean such that :

$$f_j = \begin{cases} v_j & \text{if } r_i \in R_{v_j} \\ \mathrm{not}(v_j) & \text{otherwise} \end{cases}$$

and g_j is a set of requests such that :

$$g_j = \begin{cases} \{r : r \in L_{v_j}\} & \text{if } r_i \in R_{v_j} \\ \{r : r \in R_{v_j}\} & \text{otherwise} \end{cases}$$

The implementation of the arbitration tree is based on a butterfly multistage network of cells as shown in Figure 3 for the case $n = 8$. As shown by the same figure, this structure minimizes cell fanout and its area complexity is an $O(n \log n)$ function with a small contant factor.

5 Implementation Of The Switching Part

As noted in section 2, crosspoints consist of switches. In our design each switch is a set of tristate buffers. The number of tristate buffers at each crosspoint depends of the size of the bus. The switching part (the set of crosspoints) is a layout problem. Its implementation depends on the layout strategy. The one we propose here is datapath oriented and it is based on the following assumptions :

- Use of an, at least, two metal layers CMOS technology. The first level metal is vertical and the second is horizontal.
- Use of a datapath tristate cell of height T_h and width T_w, with virtual connectors (over the cell terminals) and a number (say N_t) of horizontal transparencies (or feed-through). See figure 4.
- Use of a datapath cell that contains only transparencies (and power supplies). This cell is of the same dimensions as the tristate buffer cell and for simplicity, has also N_t transparencies.
- An appropriate relative placement of the different parts of the crossbar.
- Different signals : coming from, crossing or going into the crossbar on the datapath structure are routed using use horizontal transparencies and vertical channels.

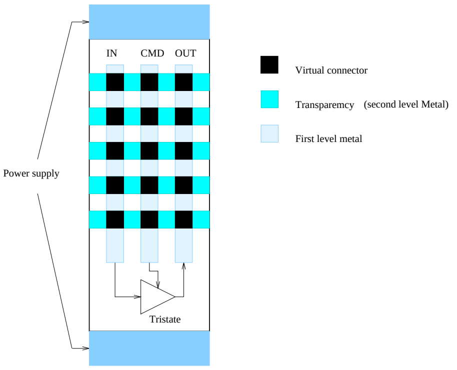

Figure 4: Tristate Data Path Cell

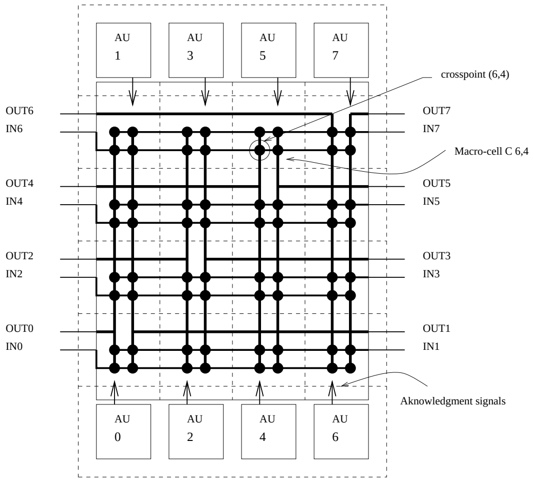

Figure 5: The Crossbar Floor Plan And Bus Layout Strategy

Hereafter, we present the layout implementation of the switching part:

The relative placement of the different units for a 8×8 crossbar is shown in figure 5. Input and output units of the same number are supposed to be grouped successively from bottom to top into pairs at the left (even numbered pairs) and right (odd numbered pairs) sides of the crossbar. The AUs (Allocation units) are located top (odd numbered AU) and bottom (for even numbered AU) sides of the crossbar. The adopted buses routing strategy is shown in figure 5. Input buses are routed horizontally and output buses are routed vertically and horizontally. The switching part can be split into $\frac{n}{2} \times \frac{n}{2}$ regular four crosspoints macro-cells $C_{i,j}$, where i is the index of the left input/output units index and j is the bottom allocation unit index (i and j are even integers). A $C_{i,j}$ consists of:

- tristates that connect input bus i to output buses j and $j + 1$.
- tristates that connect input bus $i + 1$ to output buses j and $j + 1$.

A row of macro-cells $C_{i,*}$ forms a data path. There are two kinds of cells:

- diagonal macro-cells ($C_{i,j}$ such that $i = j$): they contain vertical and horizontal routing of a pair of output buses and horizontal routing of a pair of input buses. Horizontal routing of output buses in the diagonal cells is to the left for a left output unit and to the right for a right output unit.
- non-diagonal macro-cells: they contain horizontal routing of a pair of input buses and vertical routing of a pair of output buses and horizontal bypassing routing of another pair of output buses.

Crosspoints within the same macro-cell are arranged so that tristates of the same crosspoint are stacked together vertically. Resulting stacks are placed horizontally. The height of a data path is then $H_d \times T_h$ where H_d includes the bus width b_w and eventual empty cells for routing or vertical abutment of power supplies. T_h is the height of a tristate cell. The order of crosspoints in the macro-cell $C_{i,j}$ is from left to right (i, j), $(i + 1, j)$, $(i, j + 1)$ and then $(i + 1, j + 1)$. Vertical channels for output buses are respectively between crosspoints (i, j) and $(i + 1, j)$ for outbus j and between crosspoints $(i, j + 1)$ and $(i + 1, j + 1)$ for output bus $(j + 1)$ (see figure 6).

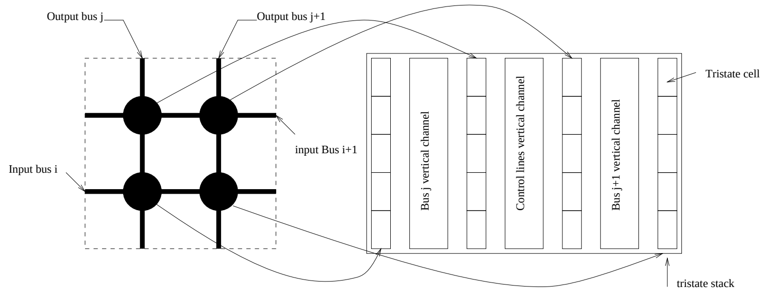

Figure 6: Crosspoint Macro-cell Arrangement

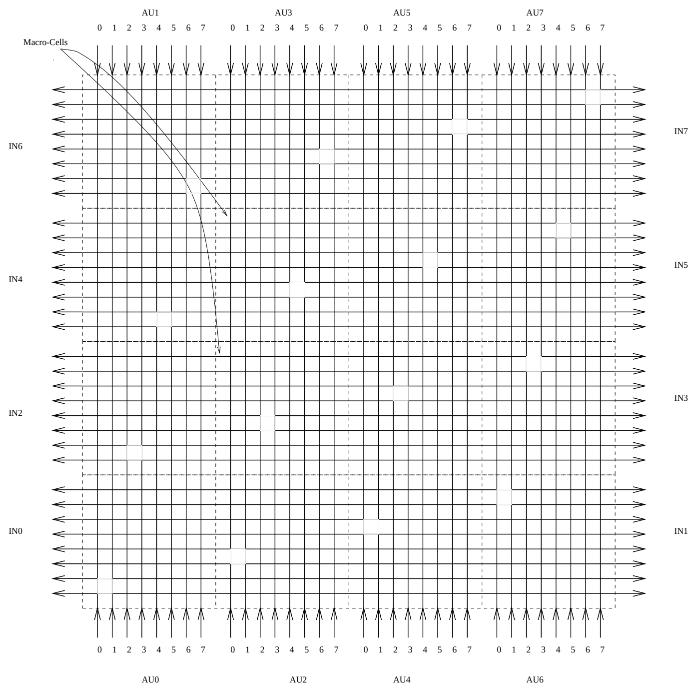

Figure 7: Routing Patterns Of Grant Signals

A channel between crosspoint $(i+1, j)$ and $(i, j+1)$ is reserved for vertical routing of control signals. We call this channel the vertical control lines channel. According to the tristate cell layout model and the floor plan, in each slice two horizontal metal routing tracks are reserved respectively for two input bit bus lines (one for the left and the other for the right one), a third track is shared by two output bit bus lines and two additional tracks are reserved for tristate driver-bus connections. Figure 8, shows how tristate drivers are connected to the buses. Note that the total number of horizontal bus tracks in a datapath row is $5 \times b_w$ and the number of available tracks is $N_t \times H_d$.

The routing of the remaining (control) signals depends on the type of a signal which is one of:

- One to all signals (broadcast): this is the case of *busy* and *free* signals.
- One to one signal (unicast): this is the case of *request* and *grant* signals.

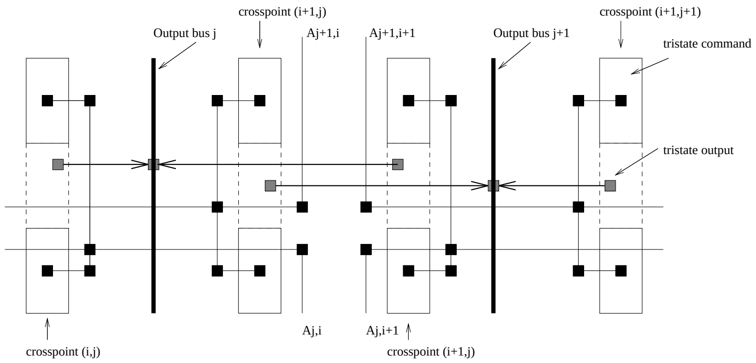

Figure 8: Output Buses Driving And Grant Signals Connecting in a Macro-cell $C_{i,j}$

Control signals are routed through vertical control channels and horizontal tracks using the available transparencies of the tristate cells. Control signals define also an optimal regular routing layout that can be derived from the placement.

Input/Output Buses	Grant	Request	Free	Busy	Total
$5 \times b_w$	$n + b_w$	n	2	n	$3 \times n + 6 \times b_w + 2$

Table 3: Needed Horizontal Tracks Per Datapath

Figure 7 shows routing of the *grant* signals (at each allocation unit, *grant* signals are numbered according to their input unit destination). Note that n vertical routing tracks in each vertical control channel and n routing tracks in each data path are needed to route the *grant* signals. When routing a grant signal horizontally, it is also connected to the control inputs of the corresponding stack of tristate. This connection is done as shown in figure 8. Globally this consumes four additional vertical tracks in the vertical channels per macro-cell and b_w additional horizontal track per datapath. In the same way, *Request* signals routing need n vertical tracks in the control channel and n horizontal tracks in each datapath. Also, 2 horizontal tracks in each datapath and n vertical tracks in a vertical control channel are needed to route the *free* signals. The same thing happens in the other direction for the *busy* signals. Table 3 and 4 summarize this. It follows that the condition for horizontal routability is:

$$n \leq (H_d \times N_t - 6 \times b_w - 2)/3$$

where $b_w \leq H_d$.

Output Buses Channels	Control Channel	Total
$2 \times b_w$	$3 \times n + 6$	$2 \times (n + b_w) + 6$

Table 4: Needed Vertical Tracks Per Macro-Cell $C_{i,j}$

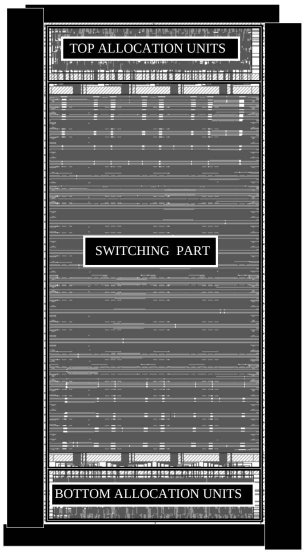

Figure 9: Example Of Layout Assembly

6 Size And Performance Evaluation

A mixed methodology is used for layout generation of a complete $n \times n$ crossbar. In one hand we use a standard-cell methodology for the allocation unit. On the other hand we developed a specific custom layout generator for the switching part which inputs are the specification of a tristate cell (this include T_w, T_h and N_t, the bus width b_w and the the number of input/output units n.

Once these parts are generated, assembly is done using a floor plan router. The use of a standard-cell strategy for the generation of the allocation unit allows to merge all the allocation units of the same side in a single block which horizontal dimension approximatly fits with the switching part. An example of assembly is given by Figure 9 (the layout corresponds to an 8×8 crossbar with 12 bit bus width).

The size estimation of a switching part can be derived from the previous section. The exact expression of its width S_w and length S_h are:

$$S_h = n/2 \times H_d \times T_h$$

and

$$S_w = n/2 \times (4 \times T_w + c_{dr} \times (3 \times n + 2 \times b_w + 6))$$

where c_{dr} is a layout constant design rule for the vertical metal. This constant includes metal width and spacing with respect to vias (it is called the pitch). Dimensions of switches of different sizes their corresponding worst case timing (CPT is the critical path timing) in a $0.5\mu m$ CMOS technology are given in table 5. They are expressed in $lambda$'s (λ): a technology independent symbolic CMOS transistor channel width. Figure 10 gives a plot of the silicon area as a function of crossbar size and bus width for a $0.5\mu m$ CMOS technology.

These results give a precise idea on the feasibilty and expected performance of a complete router. For example, using a 0.5μ CMOS technology a 16×16 full crossbar with 32 bit data

n	b_w	$H_d(\lambda)$	$S_h(\lambda)$	$S_w(\lambda)$	C_h	timing (ns)
4	1	2	240	520	1138	4
	2	4	480	540	1344	
	4	4	480	580	1285	
	8	8	960	660	1667	
	16	16	1920	820	2489	
	32	32	3840	1140	4249	
8	1	4	960	1280	2630	5
	2	4	960	1320	2580	
	4	6	1440	1400	2967	
	8	8	1920	1560	3290	
	16	16	3840	1880	4977	
	32	32	7680	2520	8528	
16	1	6	2880	3520	5030	6
	2	8	3840	3600	5943	
	4	8	3840	3760	5853	
	8	10	4800	4080	6655	
	16	16	7680	4720	9284	
	32	32	15360	6000	16621	
32	1	12	11520	10880	14678	8
	2	12	11520	11040	14632	
	4	14	13440	11360	16464	
	8	16	15360	12000	18223	
	16	20	19200	13280	21787	
	32	32	30720	15840	32889	

Table 5: Dimensions And Timing Of Switches For Typical Values ($T_h = 60\lambda$, $T_w = 40\lambda$,$c_{dr} = 5\lambda$)

wide busses support a worst case clock frequency of 163 Mhz achieving a bandwidth of 670Mo per second per bus. This corresponds to the use of serial links running at 5.36 Gigabit/s per direction (!). The dimensions of this crossbar is about $0.85cm$ high and $0.80cm$ wide. Given a chip die of 1.4cm by 1.4cm, it occupies only 35% of the silicon area.

7 Conclusion

We have presented a datapath oriented design and a parametrable layout generation of a crossbar as a building block for switching fabrics (routers). We described precisely the main parts: switching part and allocation units, including arbitration. The switching part is based on tristate datapath gates which provides compactness. Thanks to the logarithmic time complexity of the arbitration tree, the control part still achieves good timing performances while the number of inputs increases. The accurate precision of the results helps making trad-offs when designing high performance switching fabrics. Also, for a great number of communication channels routers, these results demonstrate first that a crossbar unit is not necessarily an architectural critical part. Second, very high bandwidth necessary for todays systems can be achieved using standard CMOS technology.

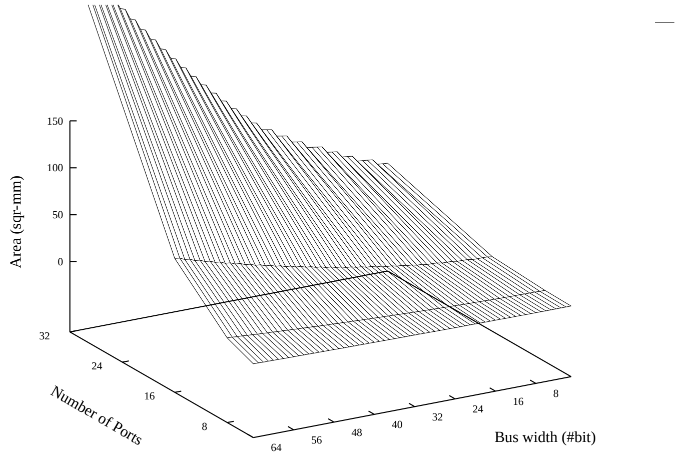

Figure 10: Area vs Number of Ports And Bus Width

References

[1] W. J. Dally *et al*, Architecture and implementation of the Reliable Router, *Proc. of Hot Interconnects II*, Stanford, August 1994.

[2] J. Ghosh, A. Varma and N. Krishnamurthy, Distributed control schemes for fast arbitration in large crossbar networks, *IEEE. Trans. On VLSI Systems*, vol. 2, pp. 54–67, March 1994

[3] A. Nowatzyk *et al*, S-Connect: from netwroks of workstations to supercomputer performance, *ACM Proc. of ISCA'95*, pp. 71–82, Santa Margherita Ligure, Italy 1995.

[4] M. D. May, P. Thompson and P. H. Welsh, editors, *Networks, Routers and Transputers*, IOS Press, 1993.

[5] R. Marbot *et al*, Integration of Multiple Bidirectional Point-to-Point Serial Links in the Gigabits per Second Range, *in IEEE Hot Interconnects Symposium*, Stanford, August 1993.

[6] H. J. Shin and M. J. Immediato, An experimental 5-Gb/s 16×16 Si-Bipolar crosspoint switch, *IEEE J. Of Solid-State Circuits*, vol. 27 pp. 1812–1818, December 1992.

[7] C. B. Stunkel *et al*, The SP2 high-performance switch, *IBM Systems J.* , vol. 34, pp. 185-204, Fabruary 1995.

[8] F. A. Tobagi, Fast packet switch architectures for broad-band Integrated services digital networks, *Proc. Of The IEEE*, vol. 78, pp. 133–167, January 1990.

[9] B. Zerrouk *et al*, Un routeur de messages adaptatif, to appear in the Proc. of *4ème Symposium ANM*, Rennes, December 1995.