PERFORMANCE-ORIENTED FULLY ROUTABLE DYNAMIC ARCHITECTURE FOR A FIELD PROGRAMMABLE LOGIC DEVICE

by

Narasimha B. Bhat, Kamal Chaudhary, and Ernest S. Kuh

Memorandum No. UCB/ERL M93/42

1 June 1993

PERFORMANCE-ORIENTED FULLY ROUTABLE DYNAMIC ARCHITECTURE FOR A FIELD PROGRAMMABLE LOGIC DEVICE

by

Narasimha B. Bhat, Kamal Chaudhary, and Ernest S. Kuh

Memorandum No. UCB/ERL M93/42

U. S. Patent Pending

1 June 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

PERFORMANCE-ORIENTED FULLY ROUTABLE DYNAMIC ARCHITECTURE FOR A FIELD PROGRAMMABLE LOGIC DEVICE

by

Narasimha B. Bhat, Kamal Chaudhary, and Ernest S. Kuh

Memorandum No. UCB/ERL M93/42

U. S. Patent Pending

1 June 1993

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

In this report we present a new architecture for a Field Programmable Logic Device. The architecture is geared towards routing completion and predictable timing performance. The central principle of the new architecture is based on the concept of efficient use of silicon resources. It is performance-oriented, with predictable interconnect and logic delays, and has a guaranteed routability. Latency in the original circuit is exploited in such a manner as to make efficient reuse of interconnect resources. Specifically, the given circuit is topologically levelized and implemented in a folded-pipeline manner. The new architecture is CAD friendly, thereby eliminating the need for complex time-consuming place and route tools.

1 Introduction

In this report, we introduce a new architecture for a field programmable logic device (FPLD). The area of FLPDs and field programmable gate arrays (FPGAs) has been gaining prominence in the last few years. Many new chip architectures are being introduced in the market, along with related computer-aided design (CAD) tools to assist the user in the implementations of digital designs on these chips. One of the advantages of using FPGAs is fast turn around time, and this means that the CAD tool should complete the user's design implementation on the FPGA with little or no manual intervention. Designing FPGA chip architectures while keeping in mind the abilities of current CAD tools, is hence an area of active research.

We have christened our architecture "*Dharma*". *Dharma* is meant for use along with its CAD tool. *Dharma* is geared towards routing completion and predictable timing performance. The task of the CAD tool for this architecture is considerably simplified – to minimize the number of levels in the circuit, with the constraint that the number of nodes per level should not exceed a certain number (decided by the available resources on chip). The tasks of placement and routing are made almost trivial, because of the architecture.

In the next section, we motivate our work further. The central idea of our architecture is based on the concept of time-sharing of silicon resources. We assume single clock synchronous circuits with registered inputs and outputs. In such circuits, the combinational part of the circuit can be topologically levelized, and the levels are then implemented in folded-pipeline fashion. This idea is explained in Section 3. The architecture itself is described in Section 4. The architecture is illustrated with detailed examples in Appendix I. A brief analysis of the architecture is given in Section 5. Section 6 outlines various modifications to the generic architecture of Section 4 and Section 7 discusses mapping issues.

2 Motivation

When the FPGA chips made their entry into the market, the idea was to make possible fast prototyping. Many different architecture styles have cropped up [1]; differing mainly in type of logic blocks, technology used (like anti-fuse, SRAM), and routing resource availability and distribution. But they have depended on the same basic idea – to make the connections programmable.

Designing circuits using these kinds of architectures has proceeded very much along the lines of mask programmable gate arrays:

1. Map the circuit gates into the logic blocks of the FPGA chip.
2. Place the logic blocks on the FPGA plane.
3. Route the connections using the FPGA programmable routing resources.

All the above 3 steps are the responsibility of the CAD tools accompanying the architecture. Step 3 has tended to be the most difficult for certain architecture styles[2], as far as completion of routing is concerned. If performance (clock speed of the FPGA-implemented circuit) is made a

Circuit	Delay (in ns)	Routing delay per level (ns)
9sym	69.3	3.1
clion9	70.2	3.3
cs8	70.6	3.4
9symml	72.8	4.0
rd84	77.1	5.0
misex2	93.3	9.1
cex2	95.3	9.6
cdk16	102.8	11.5
b9	107.2	12.6

Table 1: Worst-case delays for some benchmark examples. All the circuits have 3 levels of logic each. Each circuit is placed and routed on the smallest Xilinx 3000 series chip that has enough number of CLBs to fit the design.

criterion too, at present there is no commercial FPGA chip and accompanying CAD tool which can guarantee a given clock speed, or perform the above 3 steps while taking into consideration both logic and routing delays. As a result, it is not yet possible to fully utilize a programmable chip in high-performance designs.

The synthesis step (mapping) for performance has proceeded along the lines of delay minimization by reducing the number of logic levels [6, 9]. By decreasing the number of logic levels, the number of logic blocks along the critical path is reduced, and it is hoped that this will produce faster circuits. However, this does not always turn out to be true, since the placement and routing tools may not succeed in keeping the interconnect delay low. In table 1 we have tabulated the worst-case delays for some combinational benchmark circuits, after place and route on Xilinx [15] 3000 series FPGA chips. Each of the circuits has 3 levels of logic, but we observe that the the delay varies by a factor of about 60%. This is because the routing delay per level (defined as total routing delay divided by number of levels) varies widely. In table 2 we have tabulated the worst-case delays for circuits with 4 levels of logic each. Again we observe wide variation in the delays. Further, we see that some circuits with 4 levels of logic (**cex7, cex5, cex4** and **cs420**) have lesser delay than circuits with 3 levels of logic (**cdk16, b9**).

The reason for poor performance characteristics of FPGA-implemented circuit designs is mainly due to the routing architecture. Since each routing track can carry only one signal, it is difficult for the CAD tool to program the connections in such a manner that both the routing completion requirement and the delay constraints are met at the same time.

So far, this problem has been tackled in the following manners:

- **Architecture approach** The chip architecture designers have essentially punted on the problem, by making available much more routing resources than needed. This eases the task of the router, and routing completion is usually possible. However, silicon resource has been wasted, and because of larger number of switches per track, the parasitic capacitance on the track is quite large, and hence causes poor timing performance.

Circuit	Delay (in ns)	Routing delay per level (ns)
cex7	86.3	4.3
cex5	94.0	5.8
cex4	99.5	6.9
cs420	102.0	7.4
f51m	109.3	8.8
clip	120.5	11.1
count	133.1	13.6
apex7	168.6	20.7

Table 2: Worst-case delays for examples with 4 levels of logic each. Compared to table 1, some circuits have lower delay than 3-level circuits.

- **CAD approach[2, 11]** The quality of the routing depends on the manner in which the circuit gates have been allocated to the FPGA blocks, in Step 1. Hence a CAD tool which combines steps 1, 2 and 3 above could be expected to perform better than those which split it into steps. But no sure-fire way of combining these steps has been published so far.

The way to go, it seems to us, is to combine the chip architecture design along with the CAD. In other words, the chip architect has to acknowledge that certain problems are too difficult to solve for the CAD tool, and hence design the programmable chip such that these problems do not occur. Research in the area of combining architecture and CAD is ongoing in the universities; but the efforts therein have been to modify existing FPGA architectures.

We present an entirely different architecture. It has programmable routing elements and programmable logic elements like current FPGA chips. However, the architecture has been designed such that steps 2 and 3 above are reduced to almost trivial steps, and the CAD tool's responsibility now lies only in performing step 1. This means that the mapping solution in itself decides the performance of the circuit.

3 Levelization: a means to share silicon

In this section, we develop the central idea on which *Dharma* is based. Our idea is to time-share routing resources. We start by describing signal propagation in logic circuit, and then show how levelization and the addition of latches can make it possible to time-share routing.

3.1 Signal propagation in logic circuits

In Figure 3.1 we show the general model of a sequential logic circuit. It consists of a combinational logic block and sequential elements(latches). The output of the latches are fed as input to the combinational logic block, and some outputs of the combinational logic block are fed back as inputs to the latches. In addition to the outputs from the latches, the combinational block also gets inputs

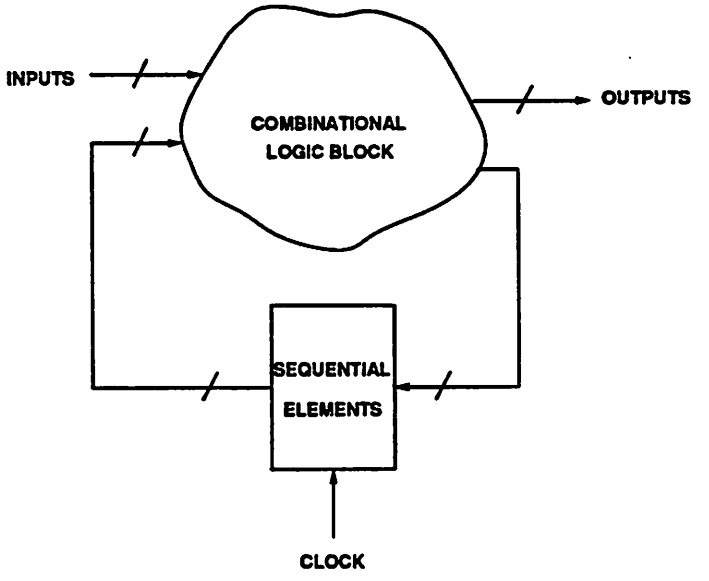


Figure 3.1: A general model of a sequential logic circuit.

directly; and some of its outputs need not feedback to the latches. The speed of this circuit is defined as the rate at which the latches can be clocked. For correct operation, if T_c is the time-period of the clock, D is the delay through the combinational block and T_{sd} is the latch set-up plus latch delay times,

$T_c \geq D + T_{sd}$.

What do we mean when we say that the combinational block has delay D? Let the inputs to the combinational block assume their correct value at time instant t_i. Then, if the time instant at which *all* outputs have assumed their correct values (as determined by the function the combinational block is implementing) is t_o, then $D = t_o - t_i$. Some of the outputs may assume their correct values earlier than the other outputs, but in the definition of the delay D, we require *all* the outputs to have settled to their correct values.

How do signals flow in such a circuit? Assume that at some instant of time t_i, the latch outputs and other inputs to the combinational block are ready. After D units of time, the outputs of the combinational block are valid, and after another T_s (latch set-up time) units of time, the values can be latched. After T_d (latch propagation delay) units of time, the latch outputs are valid, and the signal flow cycle repeats.

Let us now observe how signals propagate through the combinational block. When the number of inputs and outputs are large, the combinational block is typically implemented as a multi-level circuit, as shown in Figure 3.2.

The nodes of the directed acyclic graph (DAG) in Figure 3.2 represent single output combinational gates, and the edges represent the interconnection between them. The symbols within the nodes represent the delay through the gates and the interconnection delay is marked by symbols on the edges.

For ease of explanation, and without loss of generality, let us assume the gate delay to be the

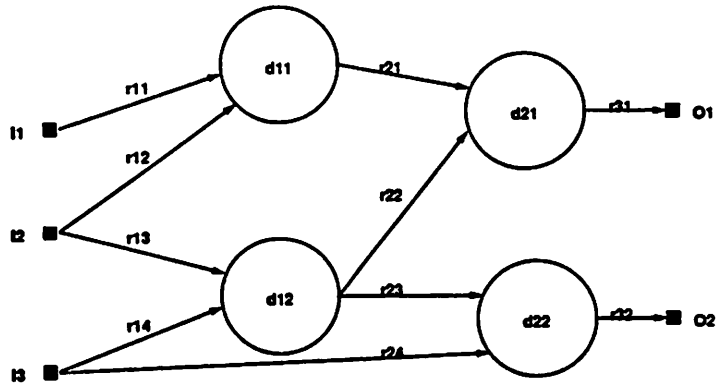


Figure 3.2: Multi-level implementation of combinational logic block.

same for all gates, and let it be δ. Similarly, let us assume the interconnect delay to be the same for every source-sink pair, and let it be γ. With these assumptions, we can immediately see that the delay of the entire combinational block, D is given by

$D = L\delta + (L+1)\gamma$,

where L is the maximum number of levels in the DAG.

In Figure 3.3 we show how the signals propagate with time. In the first time interval γ, the signals propagate through the first level of interconnect, following which they propagate through the first level of logic in δ time units. Hence, at the end of $\gamma + \delta$ units of time, after inputs are valid, the signals have reached the outputs of the first level of logic. Similarly, the signals propagate through the other levels of logic and finally reach the outputs of the Lth level after time $L(\delta + \gamma)$. One additional γ time interval is required to propagate through the interconnection to the output (or latch inputs).

Consider Figure 3.4, where we have introduced latches at all the inputs and at the outputs of all the nodes of Figure 3.2. For the present, let us assume that these latches are ideal, and have no set-up/hold times, nor any propagation delays. Let the inputs be latched at time instant t_i. After time interval $\gamma + \delta$, the outputs of the first level of logic are latched. At time instant $t_i + 2(\gamma + \delta)$, the outputs of the second level of logic are latched, and so on. Since the latches make available the signal value to higher levels of logic, the interconnect and logic resources at level i perform useful work only during the time interval $t_i + (i-1)(\delta + \gamma)$ to $t_i + i(\delta + \gamma)$ and are 'idle' (not producing anything new) for the rest of the time during a clock cycle t_i to $(t_i + D)$.

In Figure 3.3, we have used 2 different shades. Each resource, be it interconnect or logic, is shaded black when signals are propagating through it for the first time. At the end of the black shaded region, signals are available for use by the next resource (logic or interconnect, as the case may be). The light (grey) shade is used to show that the resource is, in some sense, 'idle', since it is not producing anything new. Our aim is to reuse the resource during its idle period.

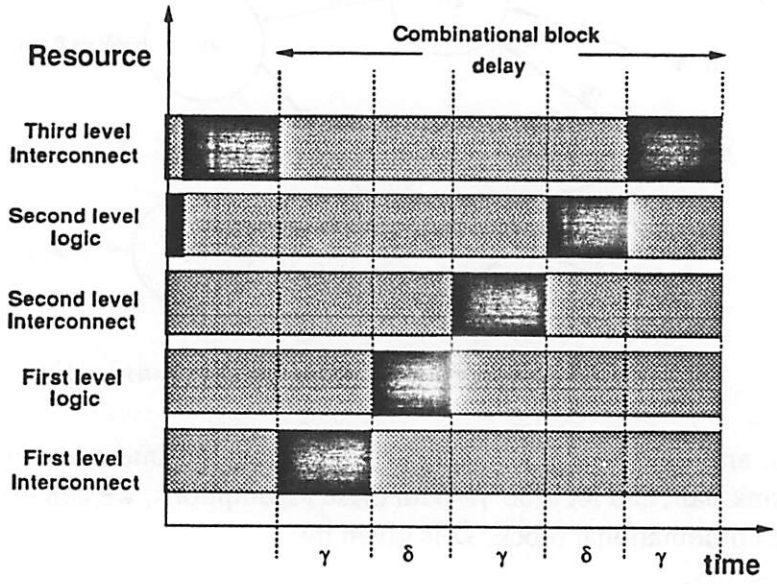


Figure 3.3: Signal propagation in Figure 3.2 and the identification of 'idle' resources. Lightly shaded regions correspond to time-intervals during which the logic or interconnect resource is not in use, when ideal latches are used to hold output logic levels, as illustrated in Figure 3.4

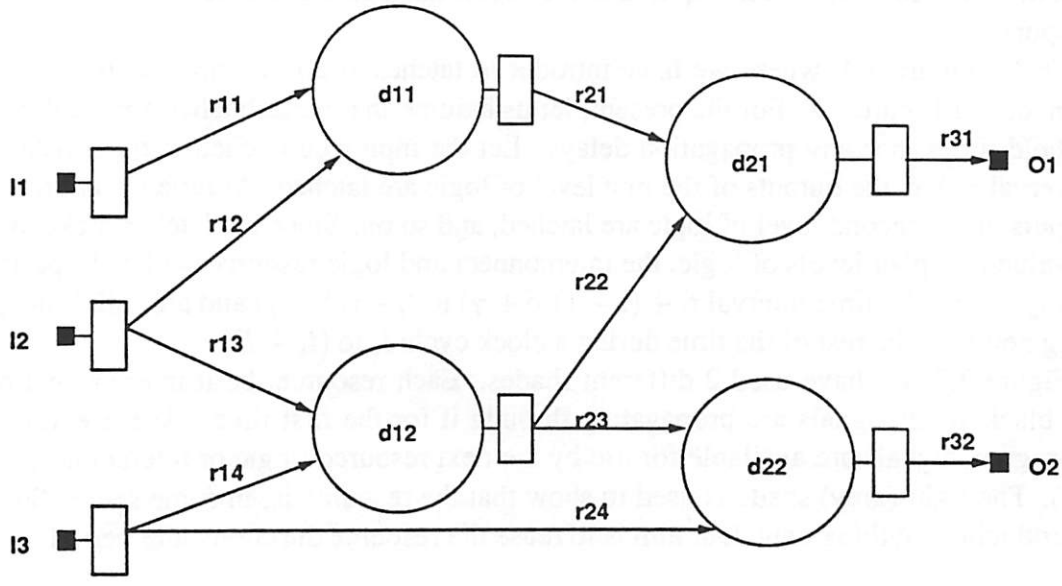


Figure 3.4: The DAG of Figure 3.2 is redrawn here with ideal latches at the outputs of every node and also at the inputs to the combinational logic block.

3.2 Levelize-and-hold

From the above observation, we can develop a strategy for time-sharing resources. We first perform a topological levelization of the DAG representing the combinational block of the input circuit. Nodes at the same level cannot be time-shared since they evaluate their outputs in parallel. Nodes at different levels can be time-shared. i.e., nodes at level $i+1$ can use the same logic resource used by nodes at level i, since the nodes at level $i+1$ cannot start their function evaluation until the nodes at level i have completed their evaluation. Similarly, the interconnect resource connecting level i to $i+1$ can be used to connect level $i+1$ signals to level $i+2$.

Hence, *Dharma* needs to have one level of interconnect, one level of logic, and latches to hold the values of signals generated. We also need a means to change the interconnection and logic function, when the next level of logic and interconnect are to be implemented. In the next section we describe how this is achieved in *Dharma*.

3.3 Sequential elements

So far we only described how the combinational block of Figure 3.1 can be implemented by time-sharing logic and interconnection. Do the sequential elements of Figure 3.1 pose a problem? No, they do not, since they are required to latch the values of the outputs of the combinational block, and make them available as inputs when the next clock period begins. In our levelize-and-hold scheme, this function is already being performed by the latches of *hold*. Thus, using levelize-and-hold, the only difference between combinational and sequential circuits is that the last level is fed back to the first level in the latter, but not in the former.

4 *Dharma* Architecture

In this section, we describe the new architecture. We first give an overview using a high-level block diagram and then go into the details of each block.

4.1 Overview

The central idea of *Dharma* is the time-sharing of silicon area, especially the interconnection resources. This is made possible because of the topological levelization of the input circuit being implemented on *Dharma*. *Dharma* consists of four main blocks (see Figure 4.1):

1. A one-dimensional array of dynamic logic modules (DLM)
2. A one-dimensional array of pass-buffers
3. Dynamic Interconnection Array (DIA)
4. Latches

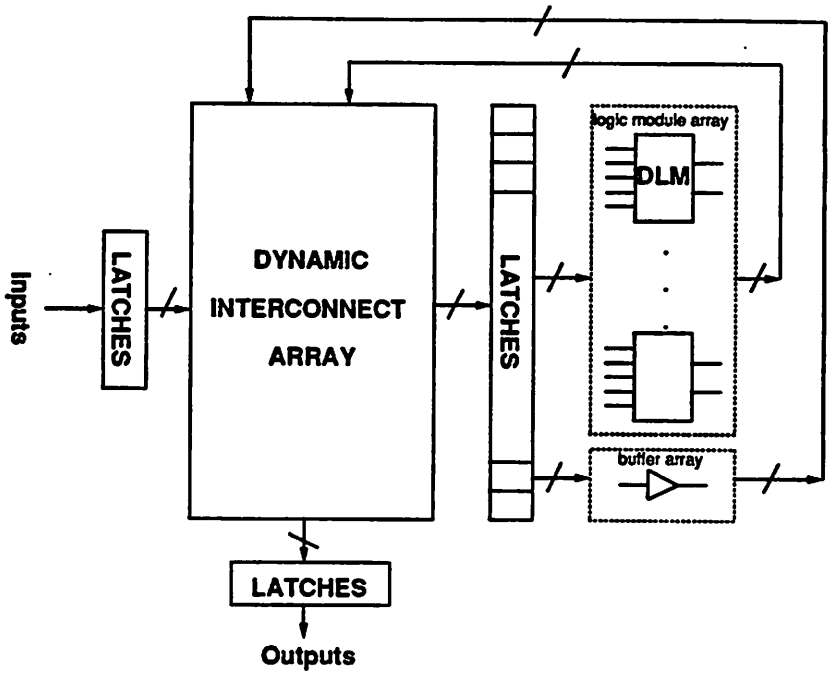


Figure 4.1: High level block diagram of *Dharma*

A circuit to be implemented on *Dharma* is first levelized. Let l be the number of levels. Then the implementation of this circuit on *Dharma* is a cyclic repetition of l internal cycles. In internal cycle i, the logic modules are configured to implement the functions in level i of the circuit, and the interconnection block is configured to implement the connections between level i and $i+1$. We use the term *dynamic*, to describe the logic modules and the interconnection, because the function implemented by the logic modules and the interconnection structure is dynamically re-configured at every level. Note that programming of a circuit onto *Dharma* consists of a one-time compilation of the values to be loaded into the DLMs and DIAs. These values are then loaded onto the chip at power-on or 'boot time'.

A DLM generates an output which is a combinational function of its inputs. To allow the possibility of realizing any function, the DLM is a RAM (look-up table). All the DLMs can perform a look-up (based on the input configuration) in parallel. Hence, each level of logic of the input circuit can be evaluated in parallel by the DLM array. (provided, of course, the number of functions being evaluated is less than the number of DLMs). The DIA connects the outputs of the DLMs and pass-buffers to the inputs of the next level of logic, through latches. To allow 100% routability, with predictable performance, the DIA block is designed such that all required connections are possible. The latches are used to store the signals between levels, and they also function as storage for state variables, in sequential circuit implementations. The pass-buffers are required to pass signals through levels; eg., if signal X is generated in level 2 and required in level 4, X will pass through level 3.

Based on the above four blocks, *Dharma* can be parameterized by the following variables (assigning different values to these variables will generate a chip-series).

1. Number of DLMs, C.
2. Number of inputs per logic DLM, K.
3. Number of pass-buffers, B.
4. Number of levels, L.

The number of internal latches would then be $KC + B$.
In addition to the four main blocks illustrated in Figure 4.1, *Dharma* has the following:

1. Level counting circuitry, consisting of
   (a) Level counter, to keep track of which level is currently evaluated.
   (b) Maximum level register, to store the value of the maximum level of the circuit being implemented.
   (c) A decision circuit, to control when the level counter loops back.
2. Internal clocking circuitry. The levels are changed at the rate of this internal clock.
3. Memory write circuitry, to program the chip for a particular circuit. This will write into the memory bits of
   (a) The DLMs
   (b) The DIA

A detailed block diagram of the chip is illustrated in Figure 4.3.

4.2 Dynamic Logic modules

The logic module has to be general enough that it can be programmed to perform many combinational function of its inputs. Among the known logic modules are PLAs, look-up tables and MUX based modules. Among these, only the look-up table based modules can perform all the combinational functions of its inputs. Xilinx series of chips have used look-up table based logic modules, in various configurations, and also with more than one output per module. For the purposes of illustration we have chosen the logic module to be a K-input, 1-output LUT. However, it should be noted that the architecture is general enough to allow other types of logic modules. The advantages of using more complex configurations would depend on how well logic modules can be utilized.

Each logic module may have to perform a different function at each level. A K-input, 1-output LUT requires 2^K bits of memory to implement any function of K-inputs. If a single chip can be used to implement circuits of atmost L levels, then each logic module will require $L \times 2^K$ bits. A detailed diagram of the DLM is shown in Figure 4.2. There are many ways in which the level select lines and the K address lines can be combined together, and the figure shows one way. Which is the best one to use will depend on the area usage and memory read time, which would depend on the technology being used. Combining the level-selects and the K inputs together, is in essence making each DLM a $(K + log_2 L)$-input, 1-output LUT, and this LUT should be designed such that area usage is kept down while giving timing performance as close to the K-input LUT as possible.

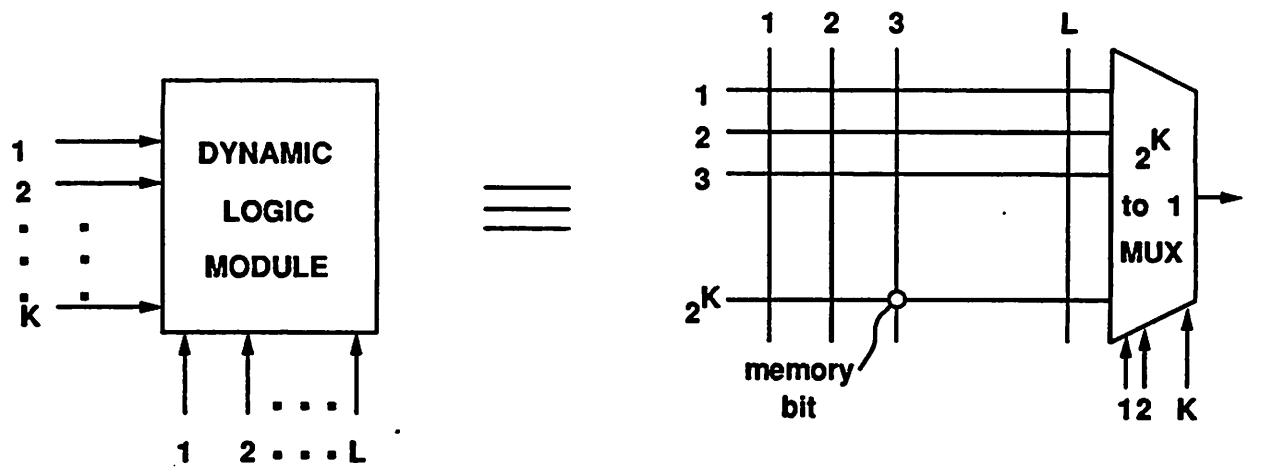


Figure 4.2: Dynamic Logic module architecture

4.3 Pass buffers

The pass buffers are used to pass signals through levels, as already described in the previous section. Although it is nice to think of them as buffers, in actuality they need not be anything more than a wire, since they just transfer the input back into the DIA block.

4.4 Dynamic Interconnection Array

The DIA block consists of two crossbar switches, as shown in the detailed diagram in Figure 4.4. This breakup into two crossbars reduces the number of routing switches.

The L-crossbar connects the outputs of the DLM and the buffers to the latches at the inputs of the DLM array. The B-crossbar connects the buffer outputs to the latches at the inputs of the buffer array. A DLM output which has to pass through the next level is directly connected to a unique buffer. This is a hard-wired connection. However, this connection passes through a multiplexer, allowing the buffer to be used to pass other signals, when the DLM at its input does not require to have its output passed through.

The circuit inputs enter the interconnection matrix, multiplexed with the DLM and buffer outputs. Input signals that are required at higher level in the circuit can be multiplexed with buffer outputs (those buffers that do not have their inputs multiplexed with DLM outputs). In such cases, the input can be used directly (by setting the multiplexer select line to choose the input instead of the buffer output) when that particular level is being evaluated. This would save buffer usage at lower levels. Thus $(B - C)$ input signals can be allocated for such usage. If there are more than these number of input signals being required at higher levels, they will have to pass through buffers at lower levels.

There are many different ways in which to implement the crossbars, so as to reduce the number of switches and area occupied. Several such techniques are listed in Section 5. Whatever be the technique used, the idea is the same, to be able to connect the outputs of level i to the inputs of level

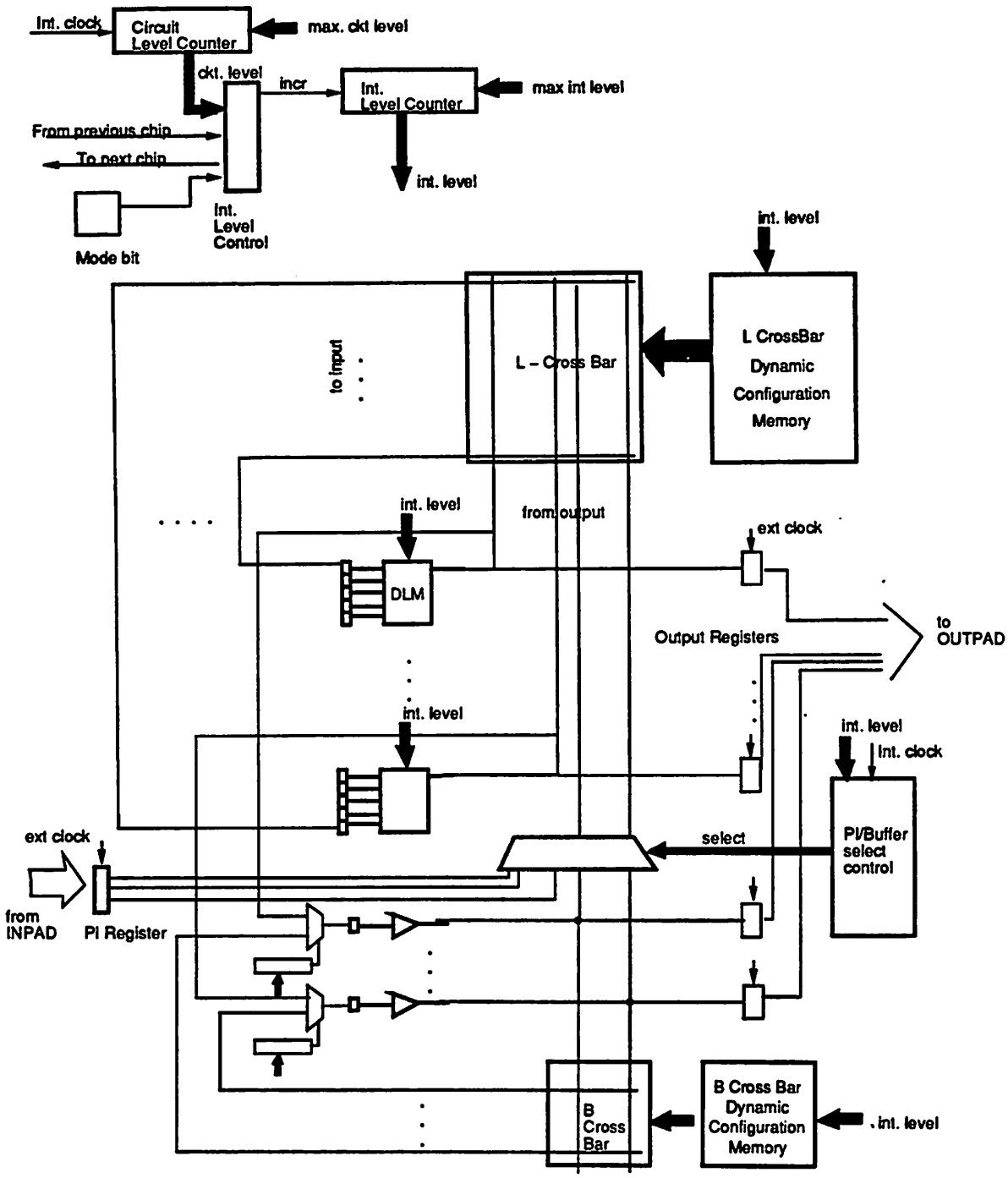


Figure 4.3: Detailed block diagram of *Dharma*

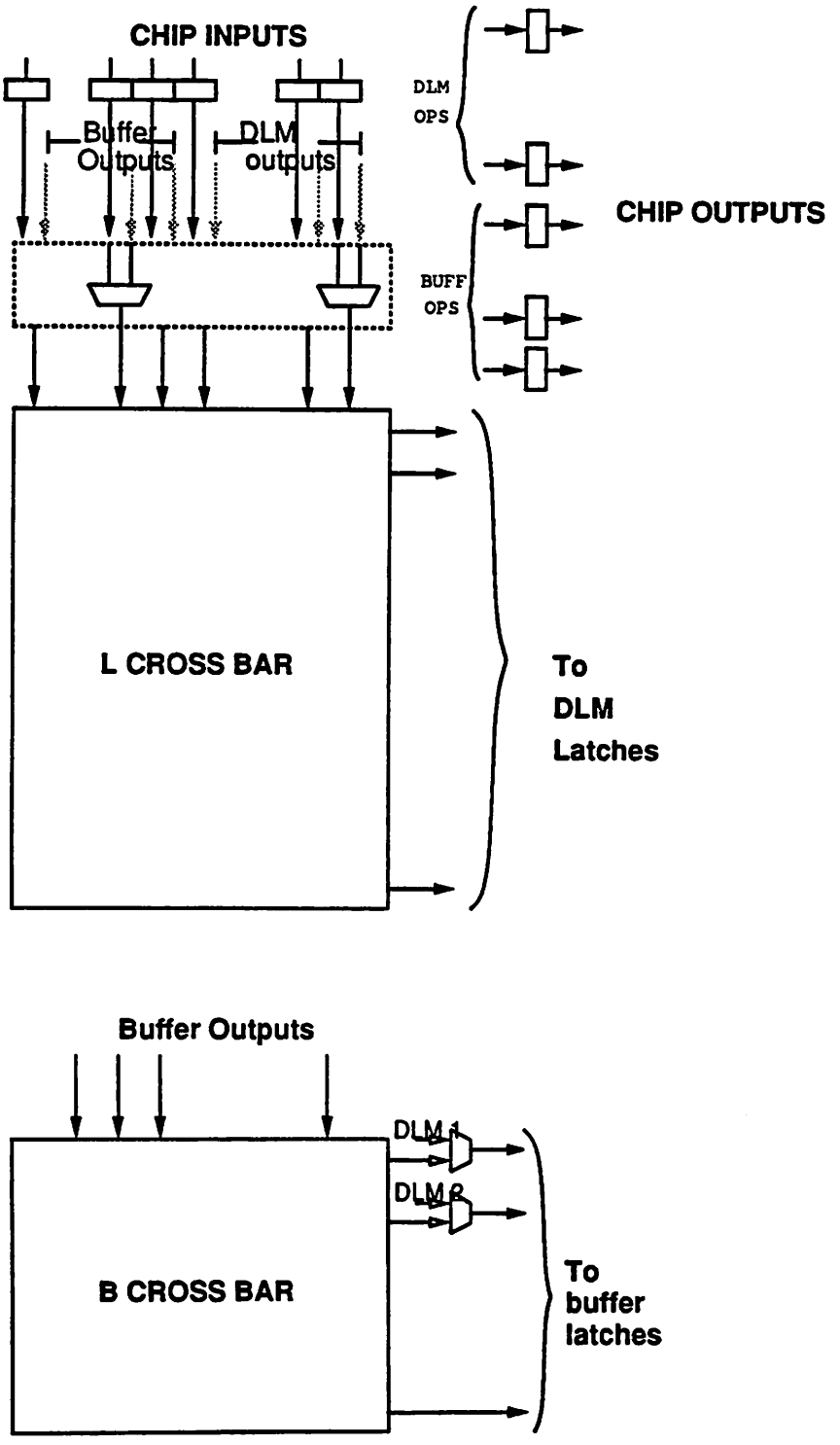


Figure 4.4: Dynamic Interconnection block architecture

$i + 1$, so that all nets are routed.

4.5 Latches

The number of internal latches on the chip equals the number of inputs to the DLM, i.e., KC. These latches serve 2 purposes:

1. To store the intermediate signals across levels
2. To store state variables of sequential circuits

The rate at which these latches are clocked depends on the delay for one level evaluation. This delay is given by

$T_{lvl} = T_{DLM} + T_{int} + T_{set\_up} + T_{lat}$,

where T_{DLM} is the DLM delay, T_{int} is the interconnection delay, and T_{set_up} T_{lat} are the set-up and propagation times for the latch. The latch value is unchanged during the level evaluation. In the case of sequential circuits, the rate at which the external clock can operate will be a multiple of the internal clock. Precisely, the external clock can operate at a frequency f_{IC}/l, where f_{IC} is the frequency of the internal clock, and l is the number of levels in the circuit.

5 Architecture Analysis

Since we are proposing a new architecture, we must analyze the pros and cons, and compare with existing architectures. We perform the analysis under the following sections:

1. Area analysis
2. Timing analysis
3. Experimental verification

5.1 Area analysis

We have to determine how effectively silicon is being used by *Dharma* as compared to existing commercial architectures. We consider the area usage for logic and interconnect separately. Consider the implementation of a circuit, which has l levels and N logic modules.

Let us compare this with a Xilinx-style architecture, where we have a 2 dimensional array of CLBs. Let the number of CLBs be X. If there are C LUTs and L levels in *Dharma*, then we have actually got CL CLBs. Let X above, be such that $X = CL$. In this case, silicon used for the purposes of logic modules is almost the same, except that in *Dharma*, since we can re-use the K-to-1 decoder, we would need only C K-to-1 decoders, instead of CL. So there is a net area benefit of C(L-1) K-to-1 decoders.

In the Xilinx-style, any circuit with $N \leq X$ can fit in. This is far too optimistic, since usually only 50% to 75% of the CLBs can be used, to make routing completion possible. In this case, the number of levels l has no consequence on area.

clear at this point, simulation studies have to be performed to arrive at a definite answer – the reduced crosspoints implies that the signals have to be connected to lesser switch inputs, and this will reduce the parasitic capacitance, and could improve speed).

5. The above discussion has centered on using the crossbar to implement a real connection, i.e., the interconnect switch connects its inputs to its outputs (say through transistors). This type of connection is *required* in the case of analog signals, where the input signals can have different values. However, in our case we are dealing with digital signals, which take on only two values, either 0 or 1. So, the switch will still be functioning properly if its outputs are set to the value of the inputs they are supposed to be connected to. In essence, the switch is required to perform a one time copy of a selection of the inputs (those that are to be connected to the outputs) to the outputs. Can we take advantage of this fact to speed up the connection and reduce interconnect area? One answer to this question is delineated in the next paragraph.

   Consider the case of just one DLM input. It can be connected to any one of the $(B + C)$ signals (switch inputs) coming from the previous level. That is, its value should be set to the value of one of the $(B + C)$ signals. This is equivalent to a memory read operation, where the memory consists of $(B + C)$ bits, and one of them has to be read onto the switch output. So, the selection can be performed by using precharge circuitry and sense amplifiers. The $(B+C)$ signals correspond to one column of memory, and one of the cell select transistors is chosen by means of a decoder or local memory bit. There are both area and speed advantages in using this style of implementation. The cell select transistors are of minimum size as opposed to the large pass transistors that would have been required if a direct connection between switch input and output were required. And using the sense amp, the read operation can be performed fast. Further, the precharge time of the memory can be interleaved with the DLM look-up time, thus shortening the interconnect delay even further. This would mean that the interconnect delay would be of the same magnitude as the logic module delay (since the logic module delay is also a memory read operation), and could be even lesser. (The logic module is a $2^K L$ bit memory, and the interconnect consists of $KC(B + C)$ bit memories).

6. So far, we only discussed fully connected switches, where it is always possible to connect any switch input to any switch output. These switches tend to be expensive in terms of area. It could be possible to decrease the area requirement of the DIA, by giving up the full connection flexibility. We discuss below a smart way of reducing the number of crosspoints, while maintaining a high probability of routing completion.

   For the sake of this discussion, assume there are C DLMs, each with K inputs and 1-output. We divide the DLMs into K classes, with $\lfloor C/K \rfloor$ modules per class. A module in class j, $j = 1 \ldots K$, is connected to the j^{th} input of each DLM (instead of to every input). This will decrease the number of crosspoints and memory bit by a factor of K. It has been found that for all the benchmark circuits tested, it is always possible to achieve complete connectivity using this connection strategy. Note that, we could still have some DLMs connected to every input, just in case it is not possible to complete the connections.

7. A further decrease in crosspoints is possible by combining modifications 3 and 6 above.

7 Mapping Designs onto *Dharma*

In the following paragraphs we briefly outline how logic circuits can be mapped onto *Dharma*. We assume that *Dharma* has L levels and C modules. We assume that the logic network being implemented is a feasible network (i.e., each node in the network has $\leq K$ inputs). If not, logic synthesis techniques can be used to transform it into one. Let the combinational part of the feasible network be topologically levelized. Let P be the number of levels, S be the total number of modules and G be the maximum number of modules in any level.

If $P \leq L$ and $G \leq C$, then the mapping is straightforward, with one level of the circuit being evaluated in every internal cycle of *Dharma*.

If $P > L$, with $S \leq CL$, then it may be still possible to implement the circuit on *Dharma* by grouping together modules across several levels. Each such group can then be realized in more than one internal cycle, before switching over to the next group.

If $G > C$, and $S \leq CL$, then the design could still be realized within a single chip, at the expense of increasing the number of levels.

When we deal with designs that do not fit on a single chip, $S > CL$, we will have to split it across several chips. If the partitions are independent (i.e., they do not share any common signals), then the split does not pose any problems. But consider the case where we have P levels of combinational logic, with C modules per level, to be implemented on *Dharma*-style chips, with only L ($L < P$) levels of C modules each. Then, we will require $\lceil PL \rceil$ chips, and a mechanism to synchronize their level counting circuitry. In Figure 4.3, we show two signals, **To next chip** and **From previous chip** meant for this purpose. These signals are connected in daisy chain fashion, so that chip C_i connected after chip C_{i-1} begins its level counter only after chip C_{i-1} has reached its maximum level.

If the design is such that $G > C$, then splitting across chips could necessitate duplication of logic.

8 Conclusions

In this report we have presented a new architecture for a Field Programmable Logic Device. A new concept of dynamically changing modules and interconnections has been introduced to implement the given circuit in a folded-pipeline fashion. The dynamic architecture allows the reuse of silicon resources and makes the device completely routable. The performance of the device is fully predictable during synthesis and the architecture eliminates the need for complex place and route tools. This reduces time-consuming design iterations usually required for getting the desired performance.

Several alternatives for the architecture have also been delineated in the report. A particular variation may be chosen depending on the technology of the implementation. It may also be possible to use the concept of dynamically changing resources as a part of a larger programmable device.

Acknowledgements

We would like to thank Prof. R. K. Brayton and Prof. J. Rabaey for many useful discussions and encouragement. This work is supported in part by the Semiconductor Research Corporation under Grant No. 93-DC-008, the National Science Foundation under Grant No. MIP 9117328 and the State of California MICRO program under Grant Nos 92-078 and 92-079.

References

[1] R. C. Alford, *Programmable Logic Designer's Guide*, Howard W. Sams and Co., 1989.

[2] N. B. Bhat and D. D. Hill, "Routable Technology Mapping for LUT FPGAs,"*Proc. ICCD '92*, pp. 95-98, Oct. 1992.

[3] J. Cong and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Design," *Proc. Int. Conf. CAD (ICCAD-92)*, pp. 48-53, Nov. 1992.

[4] M. Fujita and Y. Matsunaga,"Multi-level minimization based on minimal support and its application to the minimization of look-up table type FPGAs,"*Proc. Int. Conf. CAD (ICCAD-91)*, pp. 560-563, Nov. 1991.

[5] R. J. Francis, "A Tutorial on Logic Synthesis for Lookup- Table Based FPGAs," *Proc. Int. Conf. CAD (ICCAD-92)*, pp. 40-47, Nov. 1992.

[6] R. Francis, J. Rose and Z. Vranesic, "Technology Mapping of Lookup Table-Based FPGAs for Performance," *Proc. Int. Conf. CAD (ICCAD-91)*, pp. 568-571, Nov. 1991.

[7] R. Francis, J. Rose and Z. Vranesic, "Chortle-crf: Fast technology mapping for lookup table-based FPGAs," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 227-233, 1991.

[8] C-H Horng, Vice-President, Research and Development, I-Cube Design Systems, Inc. Personal Communication. I-Cube markets large (80 by 80) single-chip crossbar switches.

[9] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli,"Performance Directed Synthesis for Table Look Up Programmable Gate Arrays,"*Proc. Int. Conf. CAD (ICCAD-91)*, pp. 572-575, Nov. 1991.

[10] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli,"Improved Logic Synthesis Algorithms for Table Look Up Architectures,"*Proc. Int. Conf. CAD (ICCAD-91)*, pp. 564-567, Nov. 1991.

[11] M. Schlag, J. Kong and P. K. Chan, "Routability-Driven Technology Mapping for LookUp-Table-Based FPGAs," *Proc. ICCD*, pp. 86-90, Oct. 1992.

[12] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, A. L. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," *Proc. ICCD*, pp. 328-333, Oct. 1992.

[13] S. Trimberger and M. Chene,"Placement-Based partitioning for Lookup-table-based FPGAs,"*Proc. ICCD '92*, pp. 91-94, Oct. 1992.

[14] N. Woo, "A heuristic method for FPGA technology mapping based on edge visibility," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 248-251, 1991.

[15] *Xilinx: The Programmable Gate Array Data Book*, Xilinx Inc., 1989.

[16] Masson, G. M., "Binomial switching networks for concentration and distribution," *IEEE Transactions on Communication*, COM-25, no.9, Sept.77, pp. 873-883.

Appendix I

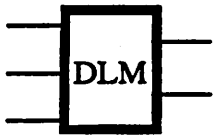
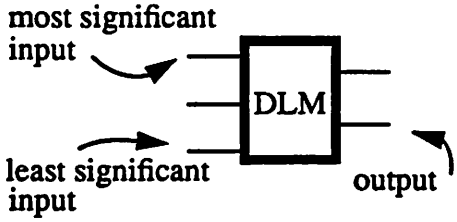
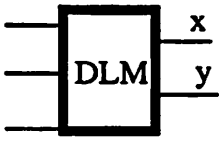
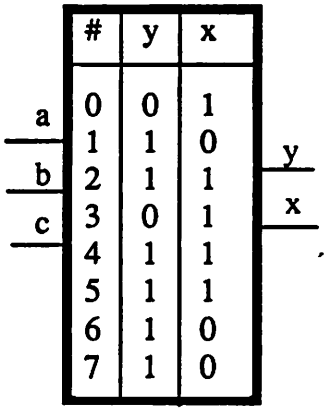
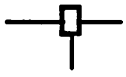
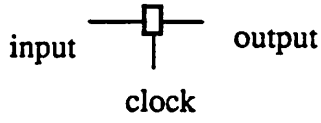
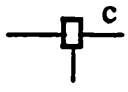
Here we illustrate Dharma by means of two simple examples. We have chosen combinational circuits with a small number of logic elements, and we demonstrate by means of figures, how this circuit is programmed onto *Dharma*. For the purposes of the illustration, we have simplified *Dharma*. The simplified architecture is discussed in the next section. Note that we have retained the main concept, i.e., levelizing and time sharing of routing resources, in the simplified version. However, we have kept the number of DLMs and the size of the DIA small, so as to help in understanding the concepts better. Section 3 describes the example circuits chosen. The circuits are presented both in terms of equations and circuit diagrams. The last section shows how the example circuits are implemented on the simple *Dharma*. Notations used for the diagrams are described in Table 1 and Table 2.

2 Simple *Dharma*

Dharma consists of DLMs, DIA, buffers and latches. The number of inputs and outputs per DLM, the number of DLMs, the number of buffers, the interconnection structure of the DIA, the number of levels that can be implemented, etc., can all be set to different values so as to yield a family of *Dharma* chips. For this simple illustration, we use a hypothetical, simple *Dharma* architecture. This simple architecture has the following features:

- All DLMs are identical
- Each DLM has 3 inputs (K = 3) and 2 outputs
- There are 3 DLMs (C = 3)
- There are 5 pass-buffers (B = 5)
- The B-crossbar and L-crossbar are implemented as full crossbars
- There are 5 chip inputs and 11 chip outputs
- L = 5, levels of logic can be implemented [i.e., each crossbar point has 5 bits of memory to choose the configuration, and each DLM has $2^3 * 2 * 5 = 80$ bits to store the logic function].

The figure on the next page (Figure 1) shows a block diagram of the simple Dharma architecture. The crosspoints in the L-crossbar and the B-crossbar are shown as shaded circles. These shaded circles represent the switch at the crosspoints and the memory bits required to configure the switch (to be either on or off). The chip inputs enter the interconnection array via multiplexers. Each multiplexer has a separate select signal. 5 bits of memory are required to control each of these select signals. Outputs of the DLMs are connected to the buffer inputs by means of multiplexers. Each of these multiplexers is also separately selected (again, 5 bits of memory are required). The DLM inputs and the buffer inputs are latched; and these latches are clocked by an internal clock. The multiplexer selection bits have not been shown in the figure, to keep the diagram simple.

Notation	Description
	A 2-output, 3-input DLM 
	A DLM configured to implement logic functions x and y
	The values stored in a DLM configured to implement x and y. The column '#' represents the addresses. x and y are functions of a, b, c. The '0' and '1' at each address are stored in such a manner so that the DLM implements functions x and y.
	A latch. 
	A latch holding the value of c
	A pass-buffer

#	y	x
0	0	1
1	1	0
2	1	1
3	0	1
4	1	1
5	1	1
6	1	0
7	1	0

Table 1

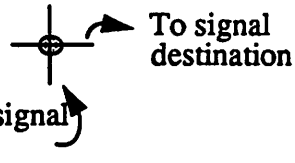
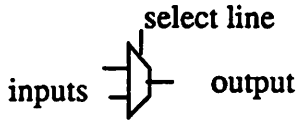
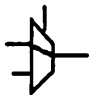
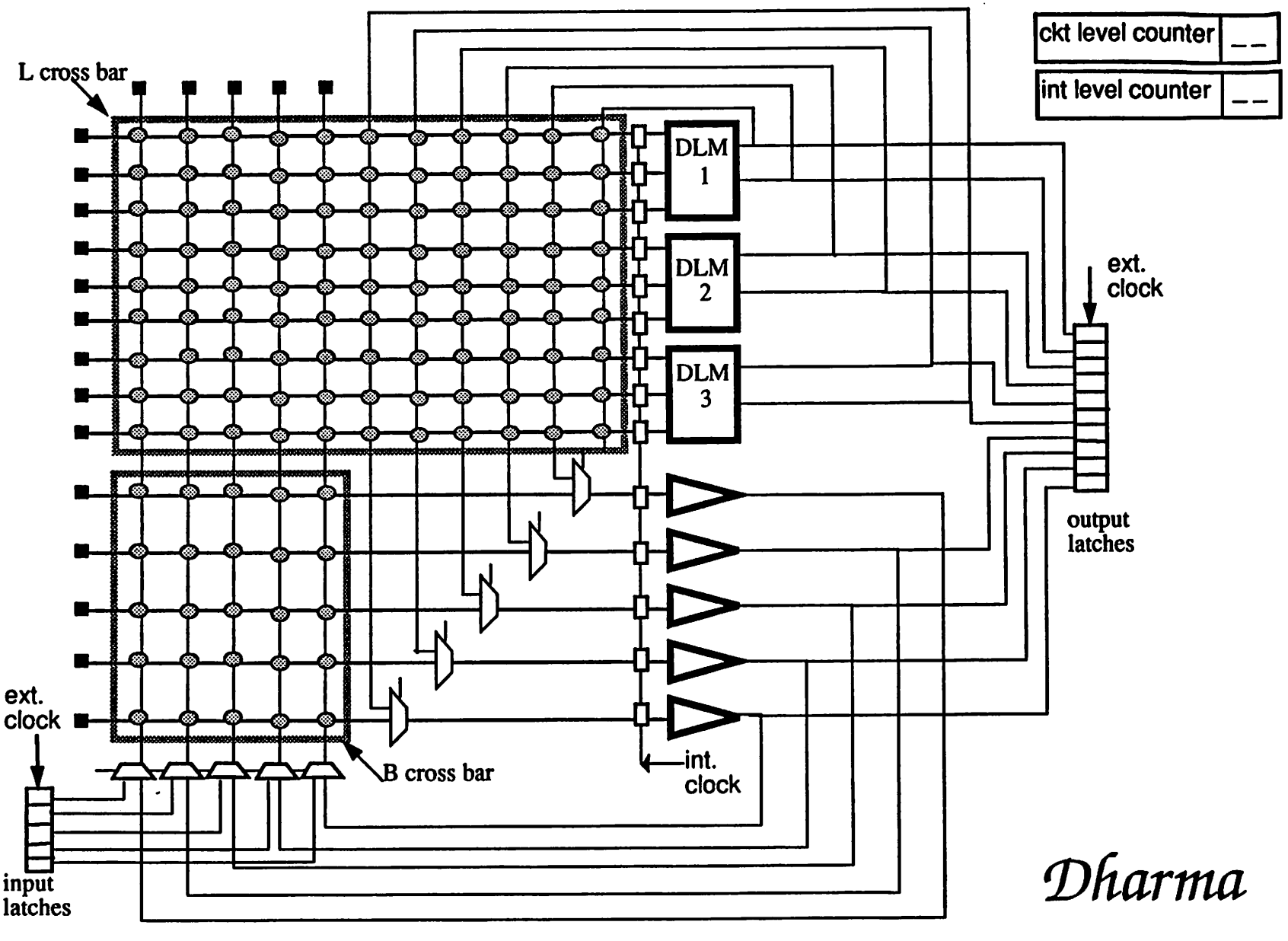
Notation	Description
	A configurable crossbar connection, with its associated memory bits 
	Crossbar connection configured so as to perform a connection
	Crossbar connection configured so as to disconnect the wires
	A 2-to-1 multiplexer. 
	The multiplexer is configured so as to connect the top input to output.
	The multiplexer is configured so as to connect the bottom input to the output.
	A 2-to-1 multiplexer shown in sideways position. Notation similar to above is used to show how it is configured to connect input to output.
	Latched pad (at chip inputs and outputs)

Table 2


ckt level counter
int level counter
L cross bar
DLM 1
DLM 2
DLM 3
ext. clock
output latches
ext. clock
B cross bar
int. clock
input latches
Dharma

3 Examples

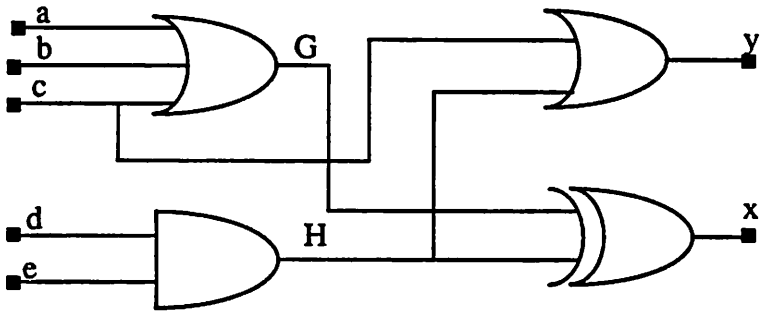
We have selected 2 examples for the purposes of this illustration. Both are combinational circuits. The first example is extremely simple. The second example is a little more complex. In the textual description, we use the symbol '+' to denote the OR operator, '*' to denote the AND operator and an '!' symbol to denote Inversion.

Example 1

This example has 5 inputs a, b, c, d and e and 2 outputs, x and y. There are 2 levels of logic.

```
G = a + b + c
H = d * e
x = G * !H + !G * H
y = c + H
```

The figure below shows schematic of Example 1.

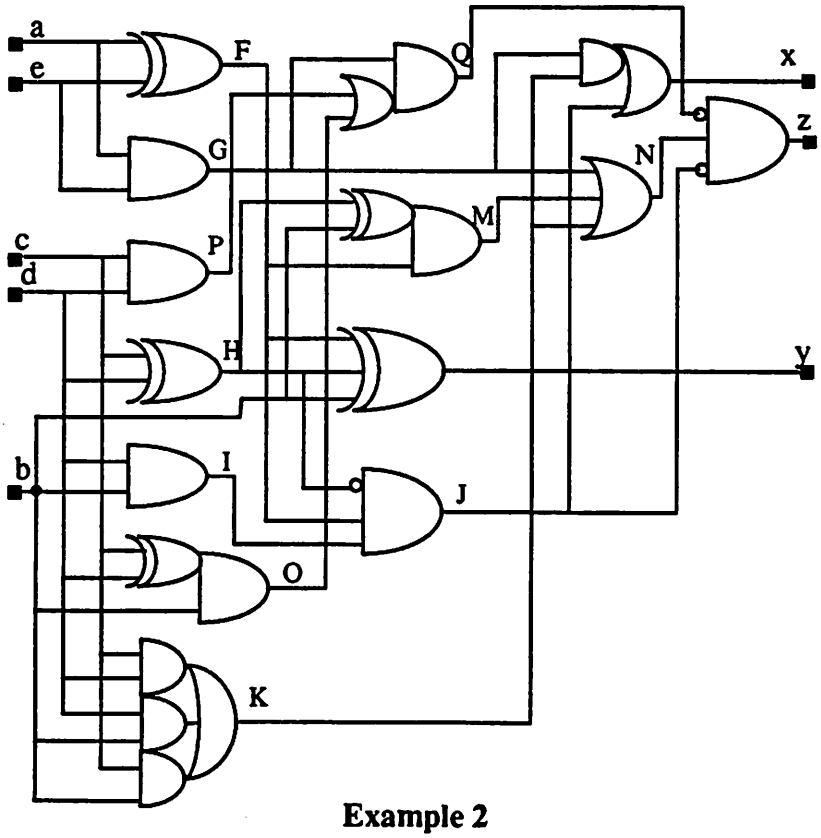


Example 2

This example has 5 inputs a, b, c, d and e and 3 outputs, x, y and z. There are 4 levels of logic.

```
G = a * e
H = c * !d + !c * d
I = b * d
F = a * !e + !a * e
J = I * !H * F
K = b * c + b * d + c * d
x = G * K + J
P = c * d
y = !H * !F * b + !H * F * !b + H * !F * !b + H * F * b
M = F * H * !b + F * !H * b
N = K + G + M
O = b * !c * d + b * c * !d
Q = G * P + G * O
z = N * !J * !Q
```

The figure below shows schematic of Example 2.



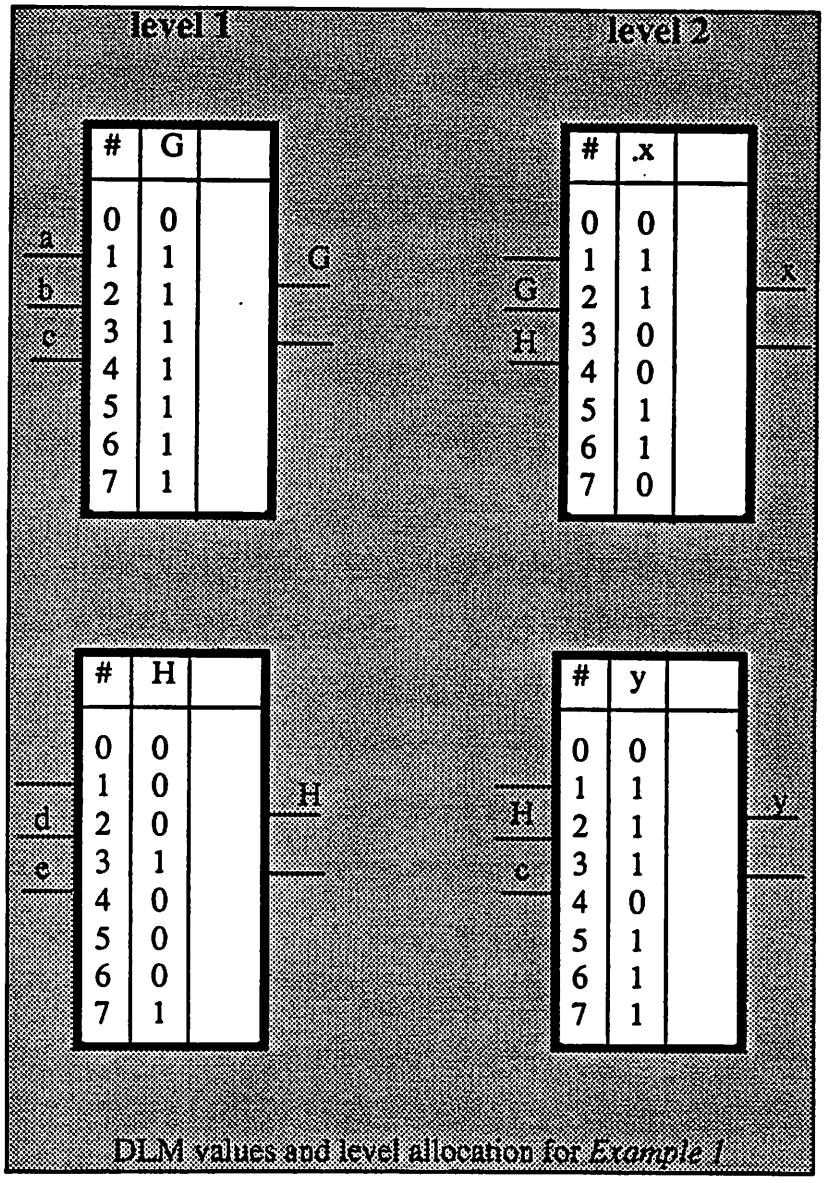
Example 2

4. Implementation

We show the implementation of the two examples, by showing the values of the memory bits that change during each level of implementation.

Example 1

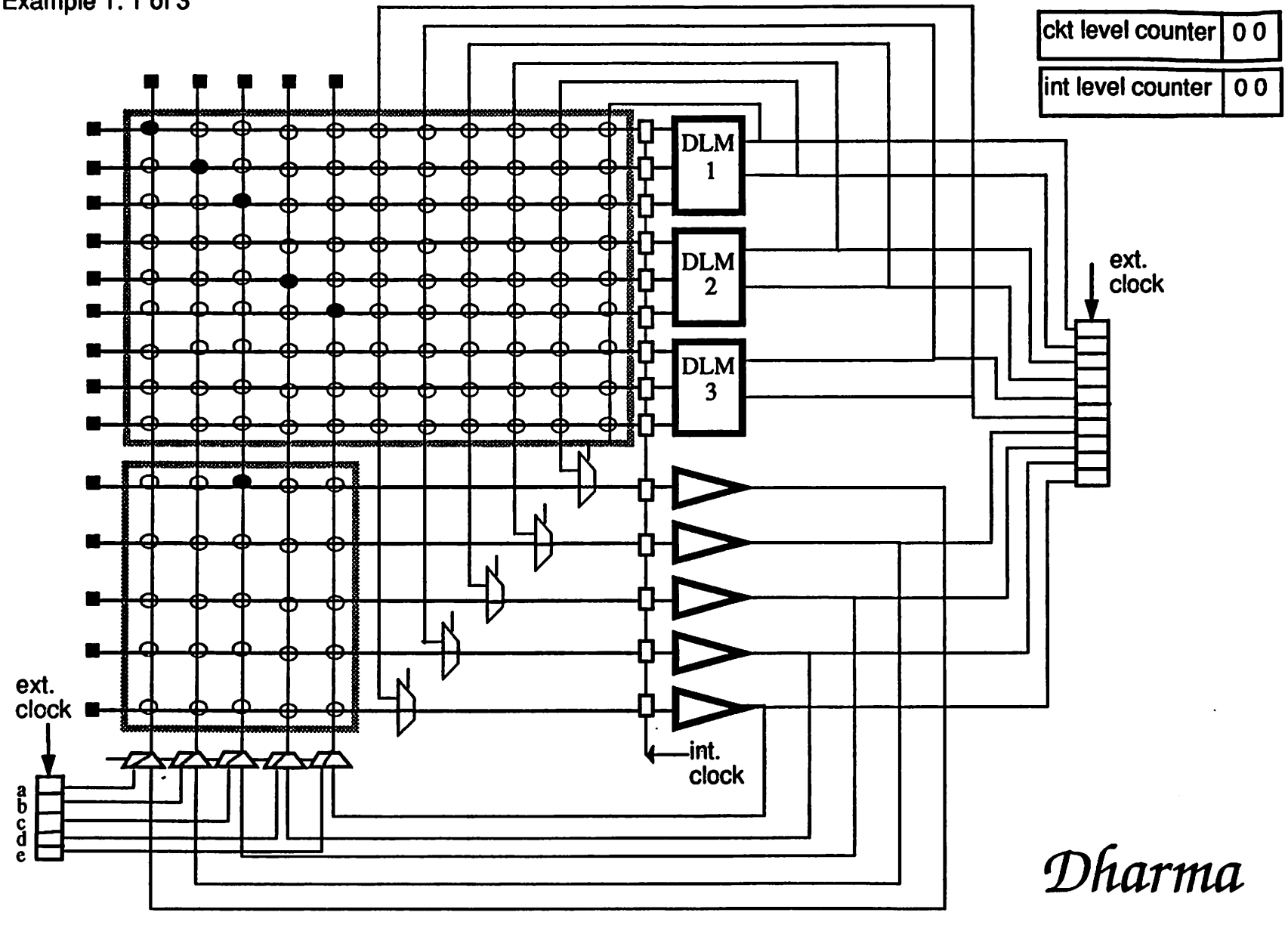
This has 2 levels of logic. The first level connects inputs a, b, c, d and e to the DLMs to generate signals G and H. The second level connects G, H and c to the DLMs to generate x and y. The values stored in DLM1 and DLM2 are as shown below. The interconnection structure is shown in the



DLM values and level allocation for *Example 1*

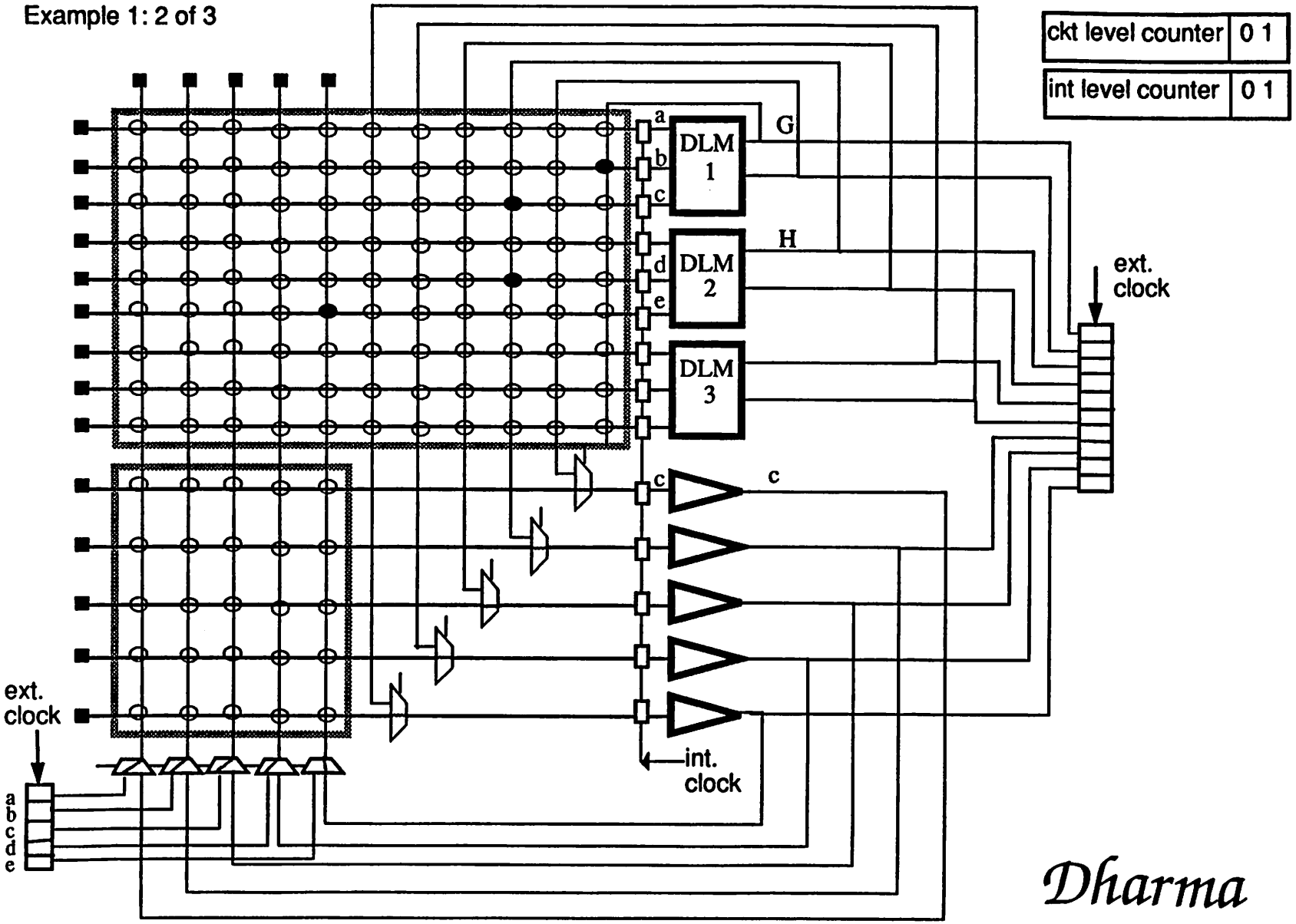
next 3 pages.

Example 1: 1 of 3



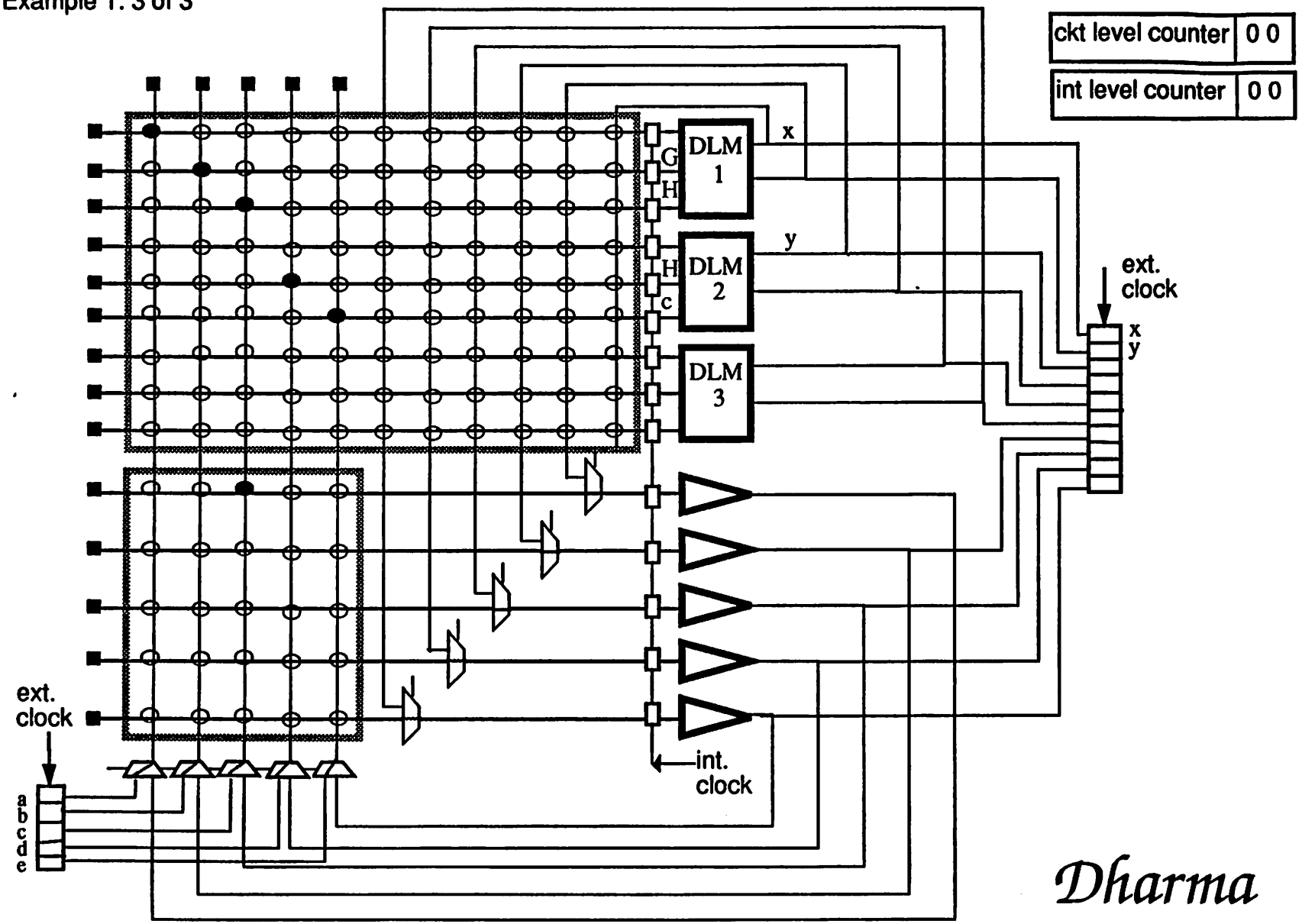
Dharma

Example 1: 2 of 3



Dharma

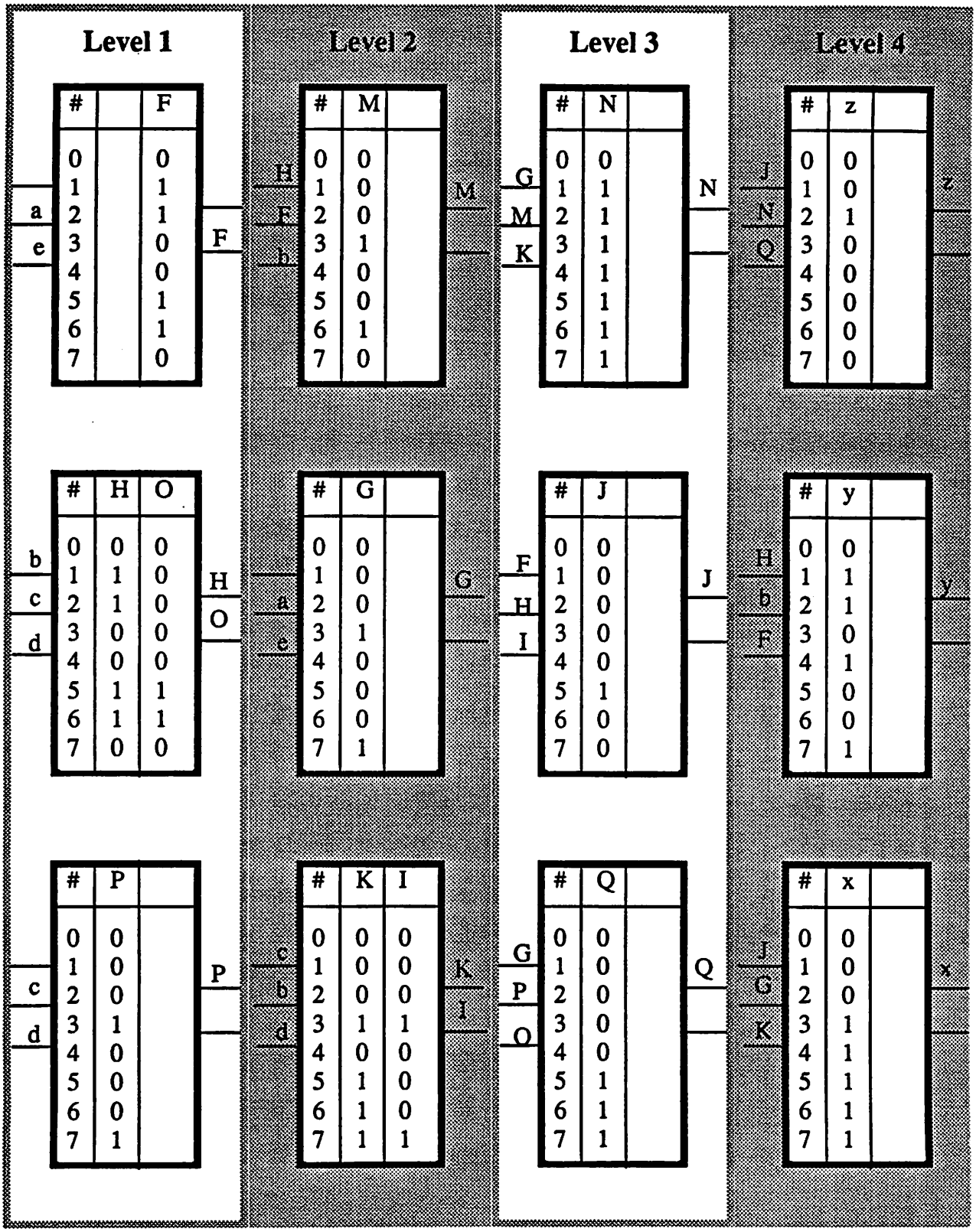
Example 1: 3 of 3



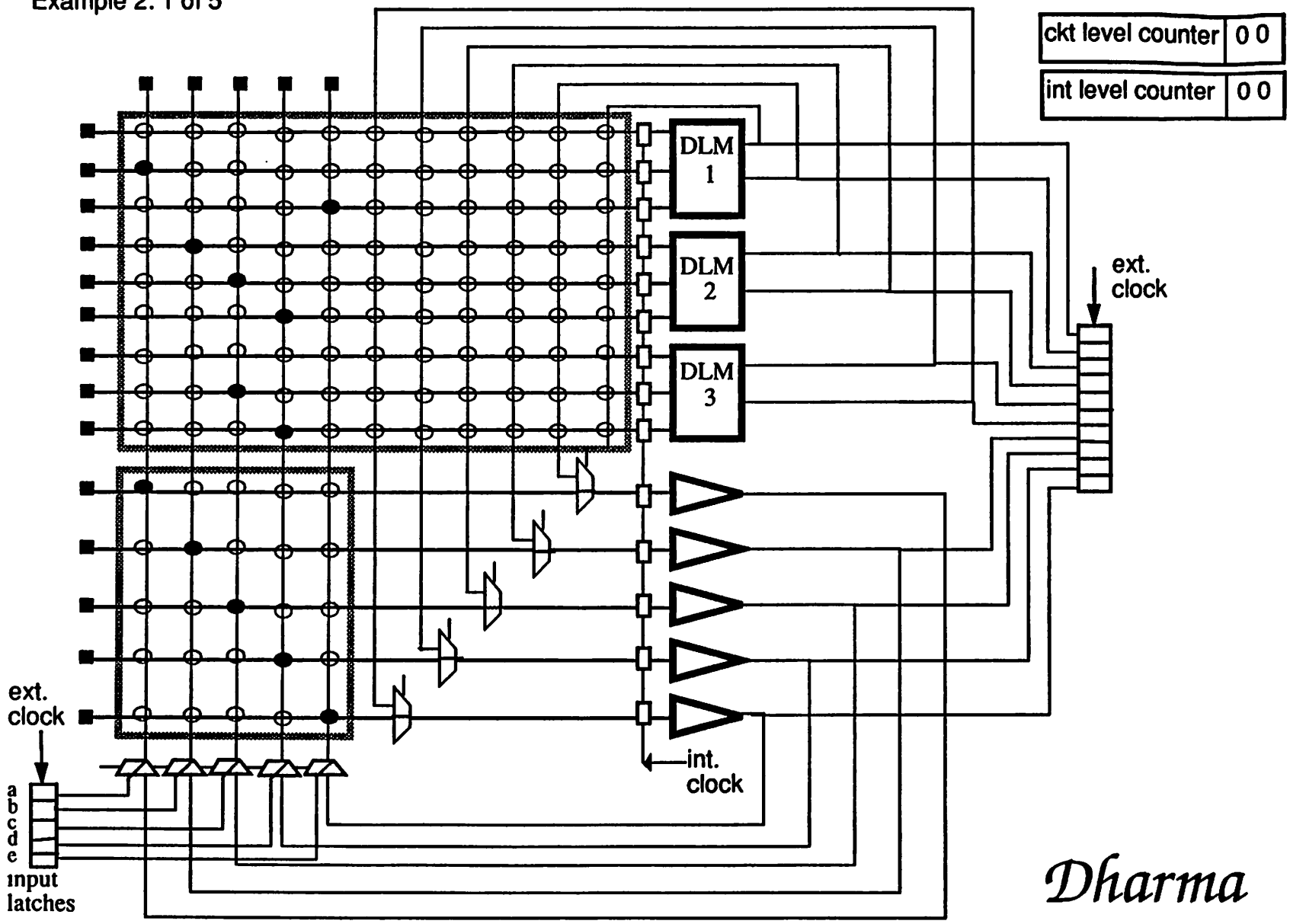
Dharma

Example 2

DLM values are shown below and the interconnection configuration is on following pages.

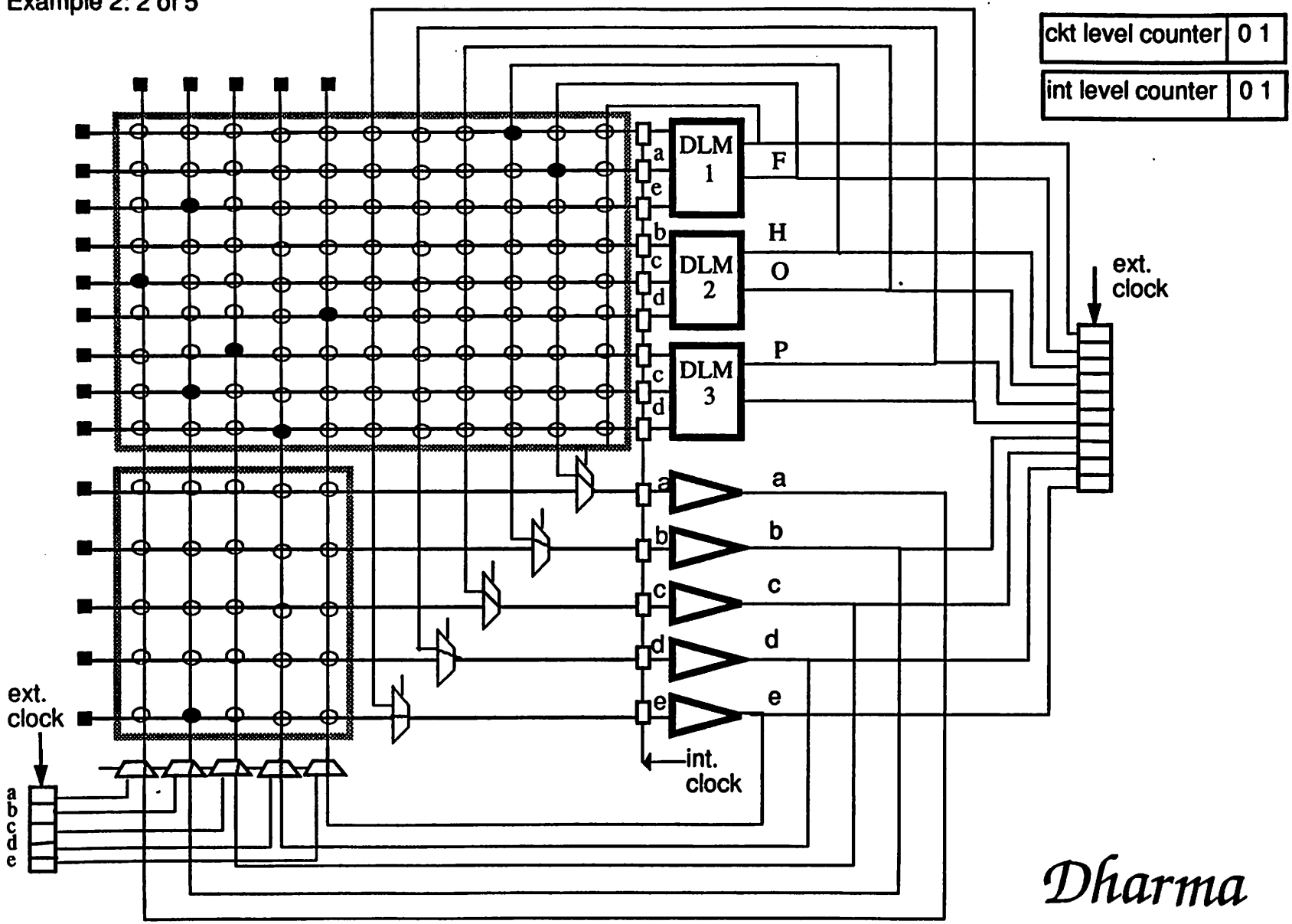


Example 2: 1 of 5



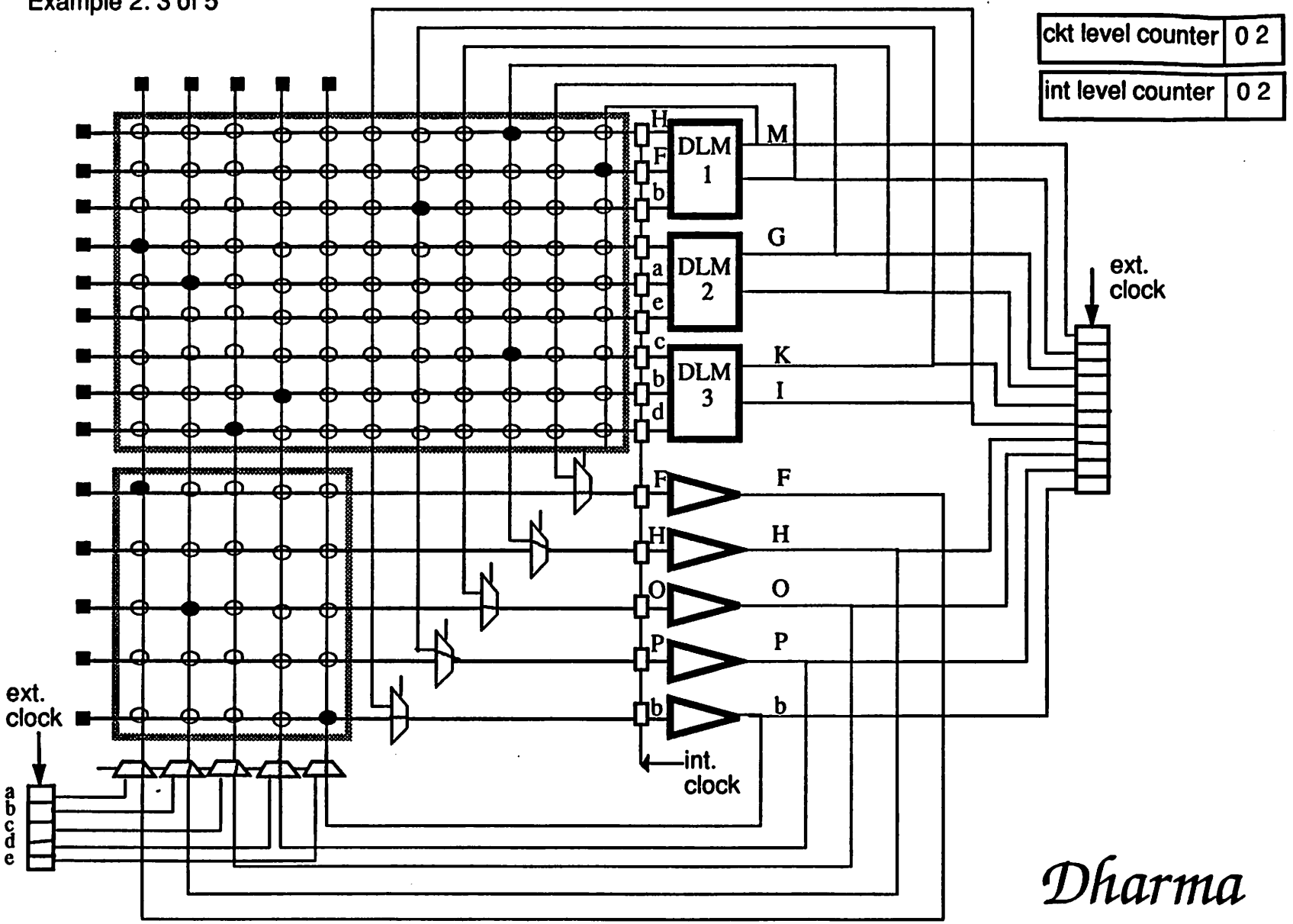
Dharma

Example 2: 2 of 5

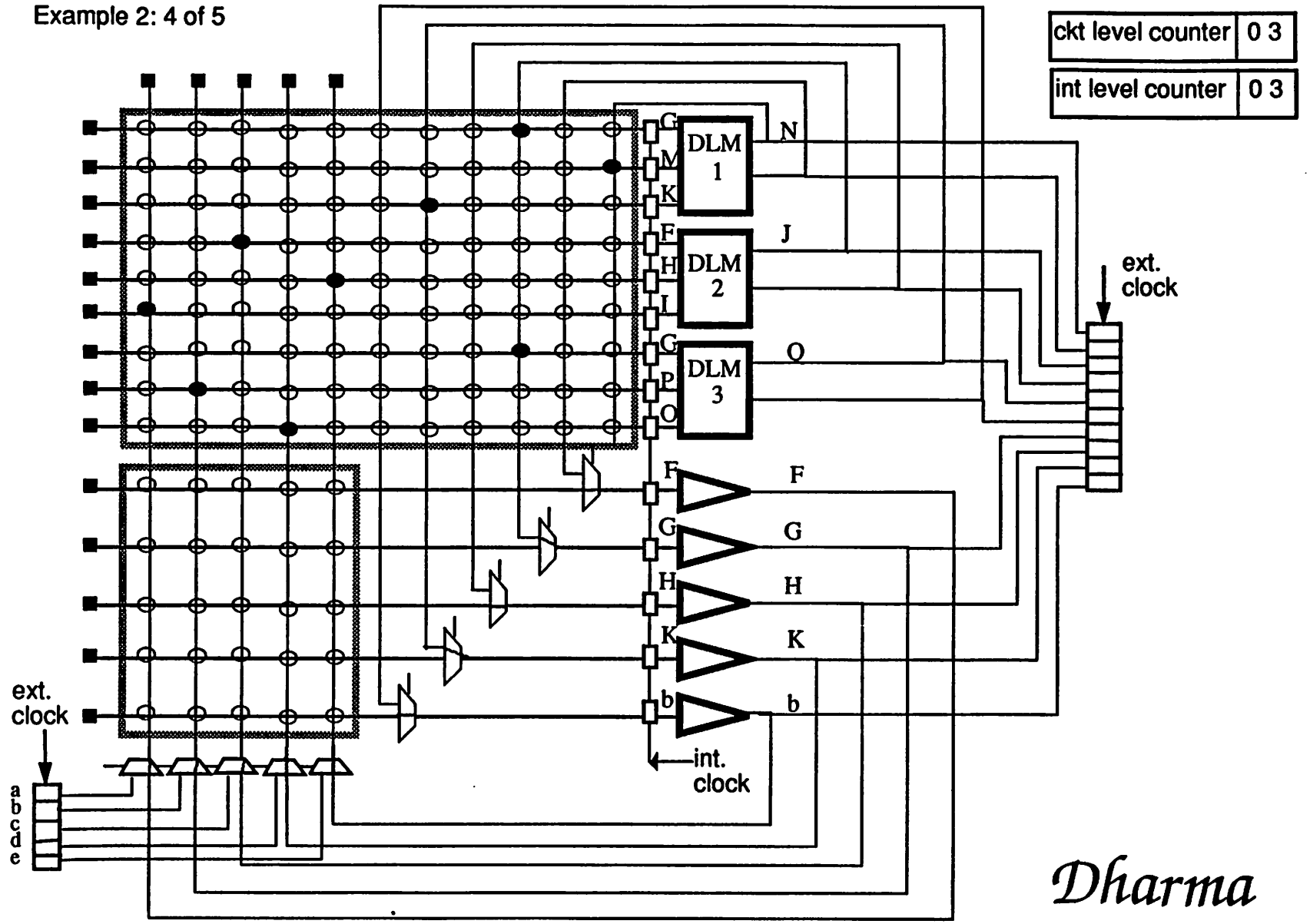


Dharma

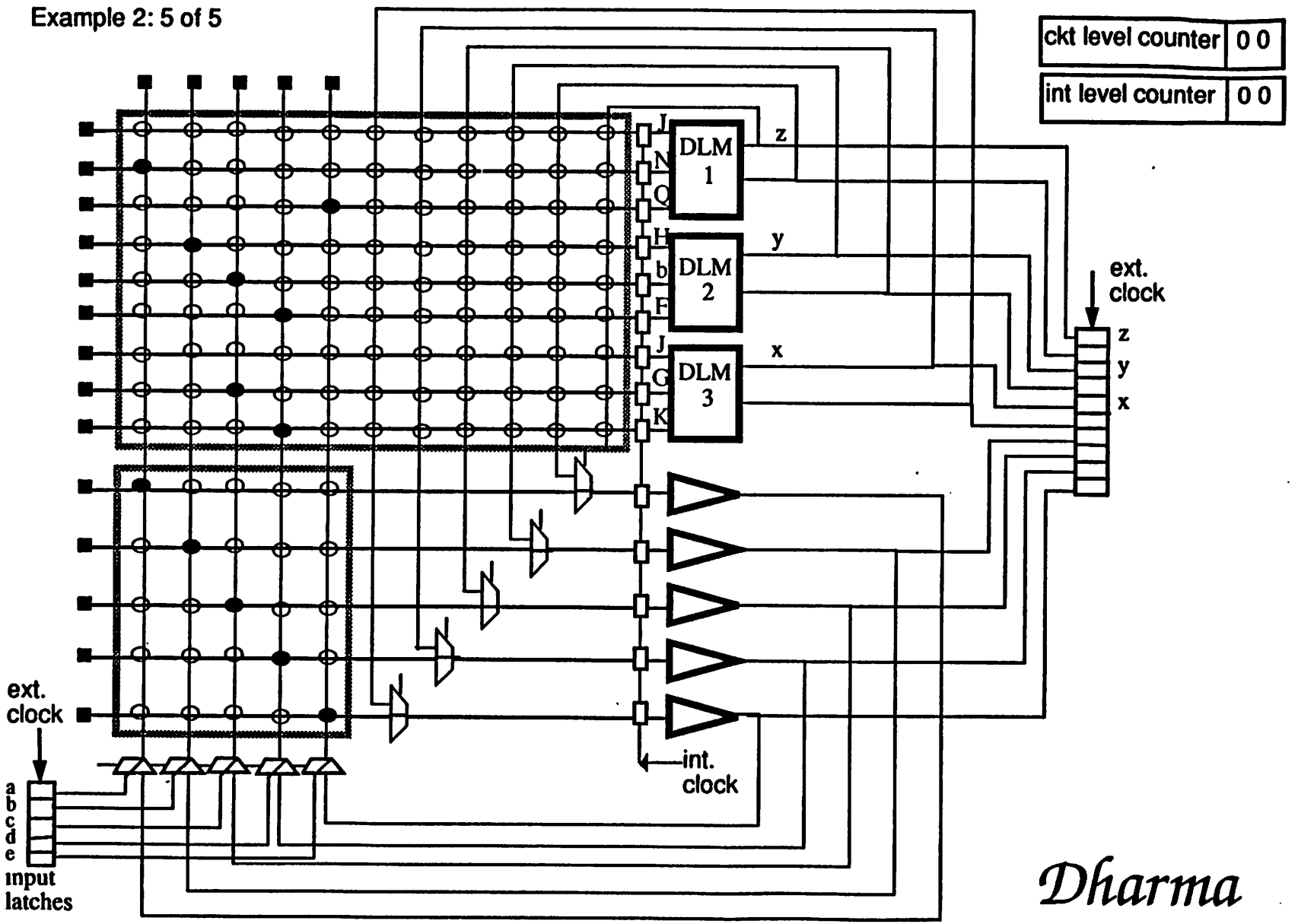
Example 2: 3 of 5



Example 2: 4 of 5



Dharma


Example 2: 5 of 5
ckt level counter
0 0
int level counter
0 0
DLM 1
DLM 2
DLM 3
J
N
Q
H
b
F
J
G
K
z
y
x
ext. clock
z
y
x
ext. clock
int. clock
a
b
c
d
e
input latches
Dharma