INTEGRATION OF ENERGY REDUCTION INTO HIGH-LEVEL SYNTHESIS BY PARTITIONING*

Achim Rettberg and Franz Rammig
Paderborn University
Paderborn, Germany
achim@c-lab.de, franz@upb.de

Abstract The optimization of power consumption at a very high design level is a critical step towards a power-efficient digital system design. The increasing usage of battery-powered and often wireless portable systems is driving the demand for IC and SoC devices consuming the smallest possible amount of energy. The aim of the method presented in this paper is to integrate low power methods within the scheduling process of the High-Level Synthesis by defining partitions. Starting from an Controlled-Data-Flow-Graph (CDFG) the proposed method uses standard scheduling techniques and path analysis on the graph to identify regions that can be combined to partitions. Each partition has a controlled activation or deactivation mechanism. That means, the partition can be switched off when it is not used. As an example design, a part of the MPEG-2 algorithm is used.

1. INTRODUCTION

The optimization of power consumption at a very high design level is a critical step towards a power-efficient digital system design. Furthermore creating optimal low power designs involves making tradeoffs such as timing-versus-power and area-versus-power at different stages of the design flow. Successful power-sensitive designs require engineers, having the ability to accurately and efficiently perform tradeoffs. In order to achieve this, engineers require access to appropriated low power analysis and optimization engines, which need to be integrated with and applied throughout the entire system design flow (see [7]).

We can distinguish between dynamic and static power dissipation. Dynamic power dissipation occurs in logic gates that are in the process of switching from one state to another. During the switching activity of the gates, any internal capacitance associated with the gates transistors has to be charged, thereby

*This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG) in SPP Verfahren zur Verlustarmen Informationsverarbeitung (VIVA), 322 1076

consuming energy. Static power consumption, however, is associated with the logic gates when they are inactive. In this case, these gates should theoretically not be consuming any power, but in reality, there is always some amount of leakage current passing through the transistors. That means those gates do consume a certain amount of power. In order to integrate optimization of power consumption at high design level it is necessary to modify the High-Level Synthesis (HLS) process of the system design flow.

2. RELATED WORK

For a low power behavioral synthesis system, automatic techniques must be developed to minimize the switching activity on globally shared busses and register files, to select low power modules while satisfying the timing constraints, and to schedule operations to minimize the switching activity from one cycle to another cycle. In the past, several algorithms for HLS have been developed. The major objective for all these algorithms was the minimization of the used resources to reduce chip area and the optimization of the system delay time [4]. An interesting approach for the integration of low power techniques into HLS is presented [10]. It focuses on the minimization of resources per cycle whereby energy consumption is reduced. This could be achieved by mapping the same operation types to a real resource. Most of the HLS perform scheduling of the Control and Data Flow Graph (CDFG) before the allocation of the registers and modules, like functional units, and synthesis of the interconnects (see [6] and [13]). Additionally, timing information for the allocation and assignment of various operations are provided. In other systems the resource allocation and binding, before scheduling, is performed to provide more precisely the timing information during the scheduling (see [5]). The work presented in [5] assumes that the scheduling of the CDFG has been done and performs the register allocation before the allocation of modules and interconnection. The work presented in [2] and [8] demonstrates that decisions at the behavioral level have a significant impact on power consumption of the final system implementation. The authors of [3] present a new technique for power optimization of control-dominated designs. This approach is an improvement of the existing techniques described in [9]. For control-dominated designs that typically consist of lots of sequential processes, scheduling is the most critical step during HLS. In our approach, we built intelligent partitions during scheduling. The developed approach is applicable to different target architectures. Especially, self-controlled architectures like the one presented in [12] are addressed. A self-controlled architecture has no global control unit; only distributed small control parts are present in the design. The data is assembled with so-called control information to direct the data content to the operation nodes by intelligent routers. Furthermore, our approach proposes

mapping possibilities, to reduce resources with the effect of reducing leakage current.

3. POWER SCHEDULER

The aim of this work is to integrate low power methods within the scheduling process of the High-Level Synthesis (HLS). We call the developed system *Power Scheduler* (see [11]). From the input (CDFG), a scheduled CDFG is generated that supports low power saving. The idea is to have a partitioned graph, whereby each partition can be activated or deactivated by a guard. A guard could be implemented by using gated clocks, guarded evaluation or power down in our approach.
Following, a short overview of the developed method is given in details by specifying the different scheduling phases, but before the cost function for the *Power Scheduler* is described.

3.1 COST FUNCTION

As already known from the dynamic power dissipation is the product from the overall capacity is the square of the supply voltages and the frequency (see [7]).

The power of a node, for example a gate, is given by the following Equation: $P_{node} = \frac{1}{2} * C_L * V_{dd}^2 * f_{clk}$, whereas C_L is the capacity of the node and f_{clk} the clock frequency. When we multiply with the switching activity α_{node} of the node we get the dynamic power dissipation of a single node: $P_{dyn,node} = \frac{1}{2} * C_L * V_{dd}^2 * \alpha_{node} * f_{clk}$.

The switching activity is hard to predict during the HLS process, therefore, we take only the worst case into account $\alpha_{node} = 1$. Now we can calculate the total power dissipation of a design by summing up the dynamic power dissipation of all nodes. This is given in the following Equation: $P_{dyn,tot} = \sum_{i=1}^{n} P_{dyn,i}$, whereas n is the total number of nodes in the design. Later on, we will see that for our approach $P_{dyn,node}$ is used to calculate the dynamic power dissipation for a partition. A partition consists of a number of nodes. Therefore, the dynamic power dissipation of a partition p is given by: $P_{dyn,p} = \sum_{j=1}^{m} P_{dy,j}$, whereas m are the number of nodes inside the partition. It is necessary to add the costs for the components (partition control unit) needed to activate or deactivate the partition. This is illustrated in Figure 1. The left side of Figure 1 shows an un-partitioned design. In opposite to that, the right side shows how a control unit, a so-called guard, which activates or deactivates the execution of this design part, controls Partition 1.

The partition control unit is always active in the part of the circuit where the partition is embedded in opposite to the controlled partition. The power cost

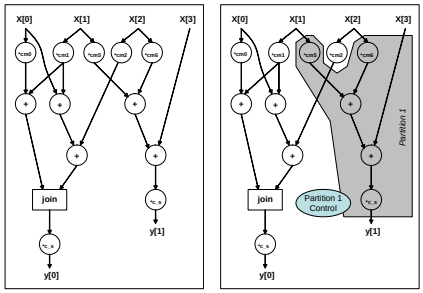

Figure 1. Partition with control unit.

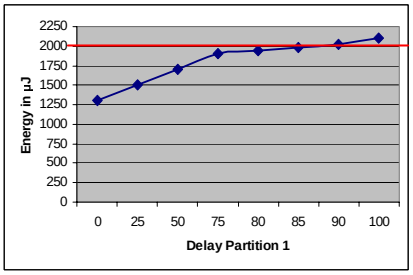

Figure 2. Energy reduction for example from Figure 1.

for a partition control unit (guard) of partition p is called $P_{gc,p}$. The power costs for $P_{gc,p}$ can be calculated by $P_{dyn,tot}$.

Now we can replace the number of nodes n in Equation for $P_{dyn,tot}$ by the number of partitions p of the entire design to calculate the total dynamic power dissipation of the design and adding the additional partitioning costs.

$$P_{design} = \sum_{j=1}^{p} P_{dyn,p} + P_{gc,p}. \tag{1}$$

Let us look at the example in Figure 1. What does equation 1 mean for the example? Let the dynamic power consumption of the entire design without partitioning $P_{exam} = 20\mu W/MHz$. Let $P_{dyn,Part1} = 8\mu W/MHz$. Thus the un-partitioned part has a dynamic power consumption of $P_{dyn,un-part} = 12\mu W/MHz$. In Equation 1 the un-partitioned part of the design is also considered as a partition ($P_{dyn,un-part}$), but without low power control. The power cost for the partition control is $P_{gc,Part1} = 1\mu W/MHz$. Here we assume the partition control consists of a gated clock.

That means, $P_{exam'} = \sum_{j=1}^{p} P_{dyn,p} + P_{gc,p} = P_{dyn,un-part} + (P_{dyn,Part1} + P_{gc,p}) = 12 + (8+1)\mu W/MHz = 12 + 9\mu W/MHz = 21\mu W/MHz$, but $P_{exam'} = 21\mu W/MHz$ is greater than $P_{exam} = 20\mu W/MHz$. Within this calculation we have not considered the run-time. Therefore it is necessary to include the run-time of the system into the Equation.

By taking the run-time of the system into account, we will calculate the *delay* d of entire design and of each partition. In literature, it is well known to evaluate different implementations of a design w.r.t low power, by calculating the *power-delay* product. Equation 2 shows the *power-delay* product for a partition k.

$$PD_k = (P_{dyn,k} * d_k) + (P_{gc,k} * d_l), \tag{2}$$

whereas d_k is the delay of the partition and d_l is the delay of the part of the circuit where partition k is embedded. Let us go back to the example. When we assume that the entire system has a run-time of $100\ cycles$ than the delay for $P_{dyn,un-part}$ is $d_{un-part} = 100\ cycles$. That means, the total energy for the design without low power reduction is $20\mu W/MHz * 100\ cylces = 2000\mu Ws = 2000\mu J$. The delay for the low power control of the partition is also $100\ cycles$. Let us further assume that the partition is only $50\ cycles$ active at run-time, the delay of $d_{Part1} = 50\ cycles$ By including all this values into Equation 2 we get: $PD_{Part1} = P_{dyn,Part1} * d_{Part1} + P_{gc,p} * d_{gc} = 8\mu W/MHz * 50cycles + 1\mu W/MHz * 100cycles = 400\mu Ws + 100\mu Ws = 500\mu Ws = 500\mu J$ and for the un-partitioned part of the system $PD_{un-part} = P_{dyn,Part1} * 100cycles = 12\mu W/MHz * 100cylces = 1200\mu Ws = 1200\mu J$.

The *power-delay* product of the entire design can now be calculated by summing up the *power-delay* product of all partitions (see Equation 3).

$$TPD = \sum_{x=1}^{l} PD_x, \tag{3}$$

whereas l is the number of design partitions. For our example we get: $TPD = PD_{Part1} + PD_{un-part} = 500 + 1200\mu Ws = 1700\mu Ws = 1700\mu J$.

Remembering that, for the design without our low power reduction method we need $2000\mu J$, we got an energy reduction of $300\mu J$ (15 %). Obviously, the energy reduction depends on the run-time of the system (see Figure 2). If $d_{Partition1}$ is less than $88s$ we reduce the energy, for this case.

Generally a HLS system tries to minimize the delay D and area A of a design. With the *TotalPowerDelay* function we have another constraint power P minimized by a HLS system. Therefore, we use Equation 3 as a cost function for our approach.

3.2 SCHEDULING FLOW

The CDFG consists of two different graphs, a so called Control-Flow-Graph (CFG) and a Data-Flow-Graph (DFG). Generally, the design of digital systems bases on a high-level specification, which is transformed into algorithmic descriptions, like C or behavioral VHDL source code. The HLS transforms the behavioral descriptions into structural ones. During the HLS the algorithmic description (CDFG) is transformed into internal formats. The CFG represents the controller for the DFG which is itself the data-path of the given algorithm. The *Power Scheduler* starts with a CDFG, which describes the design, with each node corresponding to operations and control steps and each directed edge representing data dependency and control order.

The *first* step of the *Power Scheduler* is to read the CDFG that consists of a

CFG and DFG, and store the graphs into the internal data format. The *second* step of the *Power Scheduler* is to use As-Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) scheduling on the DFG. ASAP scheduling means that all operations are scheduled as soon as they can processed in the sense of time. Vice versa ALAP means that all operations are scheduled as late as possible. If both scheduling methods are processed it is possible to calculate the mobility of each operation within the DFG. Mobility means the degree of freedom the operation has within the scheduling.
The next important step, the *third* one of the *Power Scheduler*, is to calculate the pathes within the DFG. This step consists of three different phases. The first phase examines all disjoint paths of the DFG. Eventually some of these paths are not active during the entire run-time of the system. In the second phase fork and join nodes of the DFG will be examined. The different paths between the fork and join nodes are the basis for the partitioning construction, because they are alternatively active during the run-time. In the third phase control nodes of the CFG are examined. That means, if it is applicable to schedule them as soon as possible, different paths can be identified which are alternatively active during run-time. The examined paths are the basis of the partitioning and they could be, again combined to partitions. All paths are nodes in a so called compatibility graph.
The *fourth* step of the *Power Scheduler* is to build the partitions by combining the paths that are calculated in the second step To do this it is necessary to examine if there are so-called conflicts between the paths. Two paths are in conflict to each other if they are, for example, both depending from the same fork, join or control node. That means, paths with a conflict have no edge between each other in the compatibility graph. Then a clique search algorithm [1] is used to find cliques in the compatibility graph. Finally, a clique builds a partition than can be activated or deactivated during the run-time to save energy.
From the perspective of the reader of this paper, two questions are opened. Why is it important to build partitions to save energy instead of controlling each single node? Why not use each calculated path as a partition? The answer to both questions is the same. The insertion of activation or deactivation mechanisms into a circuit has also power costs in the data-path as well as in the controller. To reduce these additional costs it is necessary to combine the paths to partitions. Nevertheless, it could be possible that after the clique approach a partition contains only one path.

3.3 PATHS ANALYSIS

As described before the path analysis consist of three different phases. In the first one we examine disjoint pathes, followed by path analysis between

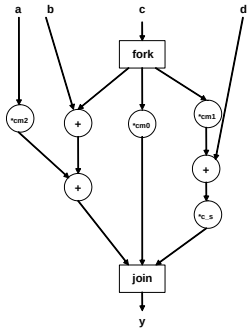

Figure 3. Paths between fork-join nodes.

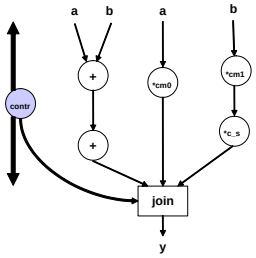

Figure 4. Control nodes path analysis.

fork and join nodes. In the third phase the scheduling of control nodes are important. A path is defined as follows:

DEFINITION 3.1 (PATH) *A path p_{v_s,v_e}, with $i \in I\!N$, is a connection from a source v_s to a destination node v_e to transport a data-word within the DFG $G = (V_d, E_d)$. Whereas v_s, $v_e \in V_d$. All nodes $v_i \in V_d$ between v_s and v_e and all nodes that are necessary to provide the correct operation of the path are objects: $p_{v_s,v_e,i} = \{v_j, \cdots, v_n\}$ with j, $n \in I\!N$ and index $i \in I\!N$.*

The index is necessary if different alternatives paths between the corresponding nodes exist. Furthermore, we need the time of each path p_{v_s,v_e}, which is defined as follows:

DEFINITION 3.2 (PATH-TIME) *Let time($p_{v_s,v_e,i}$) the time necessary to send one data-word from the start node v_s to the end node v_e of $p_{v_s,v_e,i}$.*

The path identification can be realized by *Depth-First-Search* (DFS), starting from source to the destination node and to the primary inputs or outputs that are necessary to realize the path.
To find all disjoint paths we modify the DFG slightly by including a virtual source and destination node. The source node is connected by edges with the primary inputs and constants of the DFG, because these are the only elements from where data goes into the circuit. In similarity, all primary outputs are connected by edges to the destination node, because output data goes only via the primary outputs to the environment where the circuit is embedded in.
For pathes between fork and join nodes, we start with the analysis from the fork towards the join node and add all visited nodes to the path. If we found nodes on the path with additional inputs or outputs we follow them to their primary inputs or outputs and add all visited nodes to the path. If we examine an already visited node, we stop with the determination for the path. For the example given in Figure 3 the following pathes for $p_{fork,join,i}$ (with $i =$

$1 \cdots 3$) can be identified: $p_{fork,join,1} = \{*_{cm2}, +, +\}$, $p_{fork,join,2} = \{*_{cm0}\}$ and $p_{fork,join,3} = \{*_{cm1}, +, *_{c_s}\}$. By assuming that operation $*$ needs two and operation $+$ one timestep the paths times are: $time(p_{fork,join,1}) = 4$, $time(p_{fork,join,2}) = 2$ and $time(p_{fork,join,3}) = 5$.

After the analysis of the fork and join nodes control nodes, are examined. They are scheduled as soon as possible to allow the identification of alternative paths (see also [3] and [9]). Once more, those paths are the basis of the partitioning and they are combined to partitions. Figure 4 gives an example for the path analysis for control nodes. The node *contr* controls the join node (in this case join corresponds to a multiplexer) and selects which of the inputs are directed to the output. If we can schedule and execute *contr* before all other nodes an additional timestep is needed, but we are able to activate only the used input path of the multiplexer. Therefore, we get three paths for our analysis: $p_{+,join,1} = \{+, +\}$, $p_{*_{cm0},join,1} = \{*_{cm0}\}$ and $p_{*_{cm1},join,1} = \{*_{cm1}, *_{c_s}\}$. The path time for these paths are: $time(p_{+,join,1}) = 2$, $time(p_{*_{cm0},join,1}) = 2$ and $time(p_{*_{cm0},join,1}) = 4$.

Obviously, depending on the characteristics and timing requirements of the design it may be not possible to schedule a control node by extending the run-time. Furthermore, all independent graphs in the CDFG forms a path for the partitioning. Before, we discuss the partitioning of the *Power Scheduler* a design example is introduced in the next section.

3.4 PARTITIONING

The example that is used to illustrate the *Power Scheduler* steps is part of the MPEG-2 algorithm. Figure 5 shows the DFG of the vertical and horizontal conversion used for the motion estimation of MPEG-2. Besides this, the DFG depicted in Figure 5calculates the conversion of the image format CIF 4:2:2 to CIF 4:2:0. This conversion halved the chrominance values of the MPEG-2 picture.

The path analysis for the example examines nine paths (named A to I) displayed in Figure 5. These paths are used for the partitioning of the design. Hence, that independent graphs in the CDFG forms also a partition. To build the partitions by combining the paths that are calculated by the path analysis, we construct a compatibility graph, which is defined as follows:

DEFINITION 3.3 (COMPATIBILITY GRAPH (CG)) *Let $G = (V_k; E_k)$ be a undirected graph, with the set of nodes $V_k = v_1, v_2, \cdots, v_n$ equal to the number of paths of the CDFG. An edge (a_s, a_t) with $a_s, a_t \in V_k$ exists in E_k if there is no conflict between the nodes.*

Each path of the path analysis is a node in the compatibility graph. An edge between two nodes exists if they have no conflict and not the same start and end node. Two paths are in conflict to each other if they are depending from the

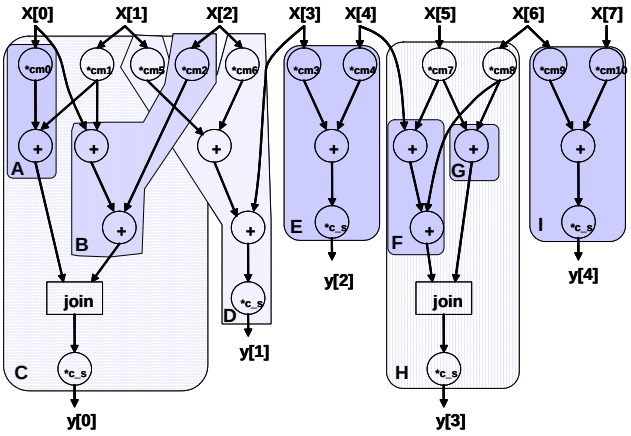

Figure 5. Partitioned design example.

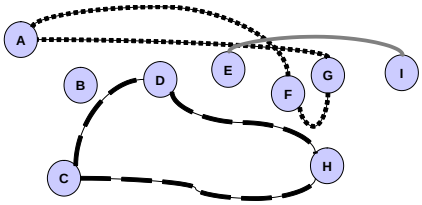

Figure 6. Compatibility graph.

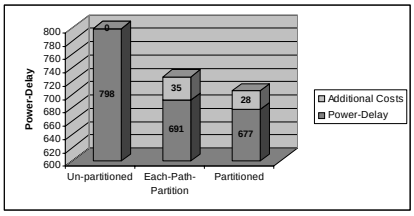

Figure 7. Results for design example.

same fork, join or control node. Therefore, those paths have no edge between each other in the compatibility graph. Furthermore, the path time is recognized. Then a clique search algorithm [1] is used to find cliques in the compatibility graph. The clique algorithm found three cliques for our design example (see Figure 6). Therefore, partitions $P_1 = \{A, F, G\}$, $P_2 = \{C, D, H\}$ and $P_3 = \{E, I\}$ can be activated or deactivated during run-time to save energy. Finally, node B builds an own partition.

4. RESULTS

First results of our proposed method are promising. For the used example, we achieved an energy saving of 15 % in comparison to a not partitioned design (see Figure 7). Also in opposite to a solution where each path build a partition, Figure 7 show, that we achieve a better result. For measurement we used the cost function given in Section 3.1. Furthermore, we implemented the example from the partitioned CDFG in synthesizable VHDL and used the

Power Compiler from Synopsis to prove our measurements. For other parts of the MPEG-2 approach, we achieve similar results.

5. CONCLUSION

In this paper, we presented an approach for low power driven synthesis. Thus, we use standard scheduling algorithms and implement a special partitioning algorithm based on clique search. The implemented *Power Scheduler* contains all developed methods and allows the integration of power saving at a high abstraction level. The presented example, part of the MPEG-2 algorithm, demonstrates the effectiveness of the method.

REFERENCES

[1] Carraghan, R. and Pardalos, P. M. (1990). An exact algorithm for the maximum clique problem. In *In Operations Research Letters 9*, pp 375–382.

[2] Chandrakasan, A., Potkonjak, M., Rabaey, J., and Broderson, R. (1992). Hyper-lp: A system for power minimization using architectural transformations. In *Proc. of ICCAD*.

[3] Chen, C. and Sarrafzadeh, M. (2002). Power-manageable scheduling technique for control dominated high-level synthesis. In *Proc. of Design Automation and Test in Europe (DATE)*.

[4] Gajski, D. D. and Ramachandran, L. (1994). Introduction to high-level synthesis. In *IEEE Design and Test of Computers*.

[5] Knudsen, P. and Madsen, J. (1996). Pace: A dynamic programming algorithm for hardware/software partitioning. In *Proc. of IEEE International Workshop on Hardware/Software Codesign*.

[6] Kurdai, F. and Parker, A. (1987). Real: A program for register allocation. In *Proc. of the IEEE-ACM Design Automation Conference*.

[7] Magma-Design-Automation (2004). *Enabling Low Power Design Within an RTL-to-GDSII Implementation Flow*. White Paper.

[8] Mehra, R. and Rabaey, J. (1994). Behavioral level power estimation and exploration. In *Proc. of IWLPD*.

[9] Monteiro, J., Devadas, S., Ashar, P., and Mauskar, A. (1996). Scheduling technique to enable power management. In *Proc. of the 33rd Design Automation Conference, Las Vegas, NV*.

[10] Monteiro, J., Devadas, S., and Li, B. (1994). A methodology for efficient estimation of switching activity in sequential circuits. In *Proc. of the 31st Design Automation Conference, San Diego, CA*.

[11] Rettberg, A. and Rammig, F. J. (2006). A new design partitioning approach for low power high-level synthesis. In *Third IEEE International Workshop on Electronic Desing, Test and Applications (DELTA 2006)*, Kuala Lumpur, Malaysia.

[12] Rettberg, A., Zanella, M. C., Lehmann, T., Dierkes, U., and Rustemeier, C. (2003). Control Development for Mechatronic Systems with a Fully Reconfigurable Pipeline Architecture. In *Proc. of the 16th Symposium on Integrated Circuits and System Design (SBCCI)*, Sao Paulo, Brazil.

[13] Tseng, C. and Siewiorek, D. (1986). Automated synthesis of data paths in digital systems. In *IEEE Transactions on CAD*, pp 5(3): 379–395.