Low Power Scheduling For High-Level Synthesis

1st Rubayet Kamal
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
rubayet.kamal@stud.hshl.de

***Abstract*—Low power design has become a critical factor in the technical and commercial success of modern hardware systems. High Level Synthesis (HLS) involves transforming a behavioral description into a structural RTL-level netlist through scheduling, allocation, and binding [4] [1].The aim of this paper is to discuss integrated low power methods within the scheduling process of the HLS performed in [4]. The goal is to minimize switching activity and utilize low-power modules while meeting performance constraints, ultimately achieving a balance between size, performance, and energy efficiency. The scheduler discussed is known as the Power Scheduler [4] which identifies mutually exclusive operation paths, analyzes their activity profiles, and partitions them using a compatibility graph and clique search algorithm. Each resulting partition has a controlled activation or deactivation mechanism meaning they can be switched off when not used. Finally, the paper presents an example, where the methods discussed is implemented.**

***Index Terms*—high level synthesis, scheduling, dynamic and static power, clique search algorithm.**

I. Introduction

The demand of personal computing devices and wireless communications equipment are ever increasing. Therefore, the demand for designing low power consuming circuits has increased. Reducing power consumption has become the 3rd important parameter along with area and run-time as per [3].

Power management of low-power battery operated systems is a challenge for designers of application specific integrated circuit (ASIC) and system-on-chip (SoC). Designing for these low-power systems is a critical requirement for a chip's success. Failing to meet the power challenge can lead to increase in the cost of ownership of the chip and system. Three factors are the key drivers when it comes to designing ASIC and SoC: area, performance and power. However, in the early days, area and performance were given more importance when designing chips. The rise of design failures have lead to minimization of power being the most important objective to designers. The traditional approach of designing for low-power is time consuming and unreliable. Hence, a methodology [4] for low power design to save power at all digital levels is dicussed in this paper.

The varying workload of the system can be exploited to manage power. Turning off all inactive parts of the design, or to turn them into low-performance, low-power states is a good idea. Techniques such as clock gating, power down techniques or dynamic reduction of clock frequency and/or supply voltage are known to achieve this.

Two types of power dissipation can take place: static and dynamic. Logic gates that change state from Low to High or vice versa contribute to dynamic power dissipation. These switching of gates require charging of internal capacitors, therefore consuming power. On the other hand, static power dissipation takes place when the logic gates are at a fixed state (0 or 1) at which no power consumption is theoretically expected. However, in the real-world, due to leakage current passing through the transistors, certain amount of power is consumed.

Idleness of a device can be exploited to deal with the power consumption due to leakage current. This is done by turning off the power supply or increasing the threshold voltage of transistors during periods of inactivity. According to [4], such idle periods can be identified at higher levels of abstraction.

Dynamic power depends linearly on frequency and quadratically on voltage, therefore reducing these parameters locally as much as throughput allows will lead to reduction in power consumption. This may require introducing new voltage and clock domains.

All important issues regarding power must be addressed earlier in the cycle. That's why designers now focus on power closure early in the design cycle. Relying primarily on the reduction of physical attribites of transistors for power optimistation is not ideal anymore. As per [4], major improvements can be achieved prior to Register-Transfer-Level (RTL). At RTL, savings potential is much larger and design changes are more easily implemented.

The method presented in this paper, inspired by [4], aims to integrate low-power techniques into the scheduling process of High-Level Synthesis (HLS) by defining partitions. During HLS, a behavioral description is mapped to a Register Transfer Level (RTL) structure. Starting with a Control-Data-Flow Graph (CDFG), the proposed method employs standard scheduling techniques and path analysis on the graph to identify regions that can be grouped into partitions. The main goal of the developed low power approach is to partition the design during the scheduling task of the HLS. Each partition allows the integration of dedicated activation or deactivation mechanisms into the design. That means, if a partition is not active, it can be turned off to reduce power consumption.

The structure of the paper is as follows: Section II HLS along with various sources of power consumption are discussed. Additionally, an introduction to scheduling is provided before diving into the proposed power scheduler discussed

in section III. The section describes in detail the developed Power Scheduler including all executed tasks (scheduling, path analysis and partitioning). Finally, section IV concludes the paper and summarizes the main contributions.

II. Background knowledge

Reducing power consumption in VLSI circuits (Very-Large-Scale Integration) can be achieved by: reducing chip and package capactiance, scaling the supply voltage, better design techniques and power management strategies. However, reducing chip and package capactiance is expensive requiring cutting-edge fabrication process. Also, scaling supply voltage, such as lowering voltage, will contribute to reducing dynamic power consumption but is not ideal due to the need of extra circuits to convert voltage. Better design techniques are cost-effective as no additional hardware is required. Additonally, power management strategy such as dynamically turning off unused parts of a chip to save power can help achieving significant power savings. Therefore, HLS techniques are necessary now to target lower power dissipation in the circuit.

This section is organized as follows: In subsection II-A HLS in general is discussed. Furthermore, subsection II-B contrasts traditional low-power design methods with modern, system-level approaches that use HLS. The emphasis is on how power optimization should start at the system level, rather than waiting until RTL or gate-level stages. A brief introduction to power dissipation sources is discussed in subsection II-C. Finally, subsection II-D, defines graphs crucial to the scheduling algorithm such as CDFG. Additionally, subsection II-D provides a brief overview of different scheduling algorithms.

A. High Level Synthesis

The HLS process is basically a translation function from behavioral to a structural description [1].During analysis phase, the behavior of a given circuit is studied where the focus is on understanding the behavior of the given structure. On the other hand, during the synthesis phase, for the given behavior, the goal is to determine and analyze the structure of the circuit in order to design it. The so called behavior of the system refers to the ways the system or its component interact with their environment. And the structure refers to the set of interconnected components, usually described by a netlist, that constitute the system.

HLS is quite similar to the construction of compilers that translates high-level languages to assembly. As per [2] HLS process is to take the specifications of the behavior required for a system and a set of constraints and goals to be satisfied, and to find a structure that implements the behavior while satisfying the goals and constraints. The HLS is different from logic synthesis. In the case of logic synthesis, the system is specified in terms of logic equations, which must be optimized and mapped into a given technology. Logic synthesis is applied after HLS in a design process. Fig. 1 shows the design phases of HLS. Usually the input to the High-Level Synthesis is a control data flow graph, called CDFG. More about the design flow will be discussed in upcoming subsections and specially in the section III. The HLS process consists of three phases which can be performed in any order depending upon the design flow:

- Scheduling: determines when each operation in the algorithm will be executed—i.e., which clock cycle.
- Allocation: decides what resources (e.g., adders, multipliers, registers) are needed and how many.
- Binding (assignment): maps operations to specific, allocated resources.

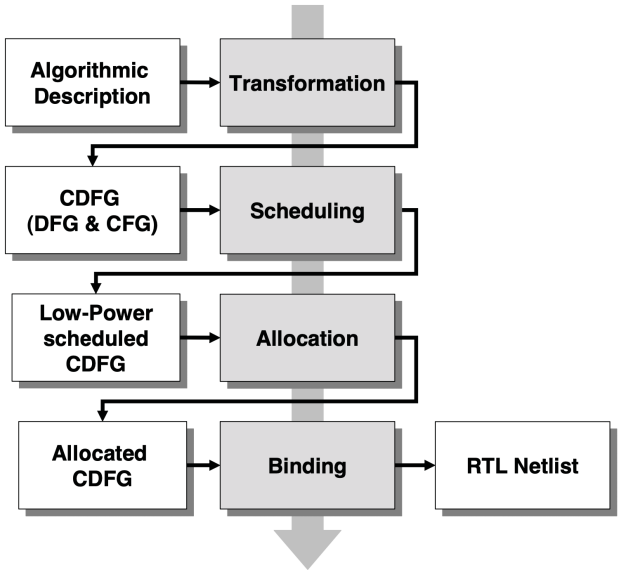

Fig. 1. HLS Design Flow [4]

As many different structures can be used to realize a given behavior, one important task of the synthesis process is to find the structure that best meets the constraints, such as area, cycle time, power, cost, etc. The aim of a low power synthesis is to meet the power constraint and to satisfy the design constraints. Synthesis can take place at various levels of abstraction. Fig. 2 displays the structuring according the Gajski Y-Chart where Gajski introduces the six design levels starting from system level to layout level. There exist three views on each level, that are namely the geometry, structure, and behavior. Looking at this fig. 2, it can said that the task of the HLS is to transform a behavioral description on algorithmic level to a structural description on register-transfer level [1].

B. Comparison of Traditional and Modern Synthesis

In traditional approach, power estimation is usually done after RTL/gate-level design, requiring multiple redesign iterations to meet power targets. Simple datapath tweaks may reduce power by 30% (but still take weeks) and reducing power consumption by 75% need full architectural/algorithmic changes, which may take months and impact cost/performance. Analysis of final physical layout helps identify ”hot spots” (i.e., transistors with high power density) but does not reduce chip-wide power. The modern approach focuses on

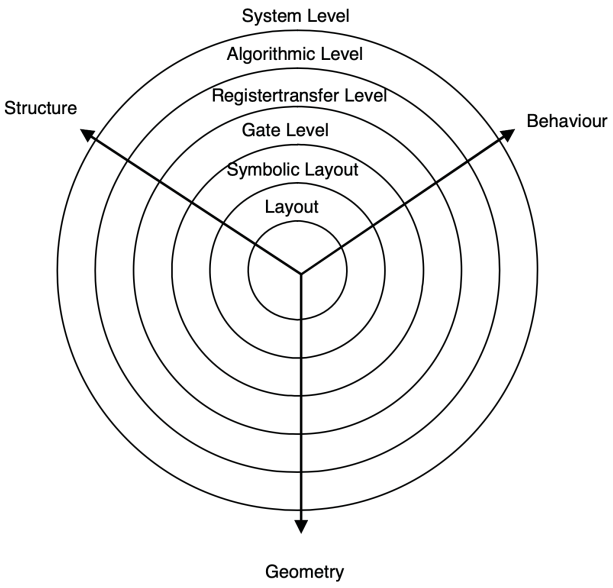

Fig. 2. Gazzky Y Chart [4]

system-Level design which starts at a high-level specification (in C/C++ or SystemC). Algorithms are analyzed and optimized for function-level power which allows architectural exploration before synthesis, reducing design time and improves power efficiency.

In system-level design, it is necessary to develop and use power-optimal algorithms and architectures. It should be noted that system-level optimization targets dynamic power consumption, which is calculated from four factors, namely clock frequency, the square of the supply voltage, load capacitance and average switching activity. Short-circuit power and leakage power are optimized at lower levels of abstraction. More about power dissipation is discussed in subsection II-C.

C. Power Dissipation

It has already been discussed in section section I about two different types of power dissipation: dynamic and static. Fig. 3 shows sources of these power dissipation [3].

Leakage current arises due to imperfections in the transistor when it is technically supposed to be "off." Even when a transistor is not conducting, some tiny amount of current leaks through due to subthreshold conduction and reverse-biased diodes. Standby current is the DC current that continuously flows from the power supply (V_{dd}) to ground, even when the circuit is idle or not switching. The equation 1 displays the total static power dissipation where n is the number of transistors:

$$P_{\text{static}} = \sum_{i=1}^{n} I_{\text{leakage,i}} \cdot V_{\text{dd}} \tag{1}$$

During the brief moment when a CMOS gate switches from one state to another (e.g., from 0 to 1), both the PMOS and NMOS transistors can be partially on, creating a short direct path from V_{dd} to ground. Although short-lived, repeated transitions can add up and waste significant power, especially in high-speed circuits. As per [6], short-circuit power dissipation accounts for less than 20% of dynamic power in a well-designed circuit. Therefore, the largest contributor to total power in active CMOS circuits is due to capacitive switching.

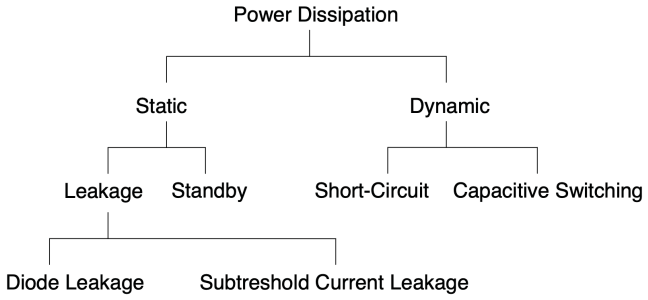

Fig. 3. Source of power dissipation

Capacitive switching power dissipation is caused by charging and discharging of parasitic capacitance in the circuit [3]. It is given by the following equation:

$$P_{\text{dynamic}} = \frac{1}{2} \cdot C_L \cdot V_{dd}^2 \cdot \alpha \cdot f \tag{2}$$

where C_L is the load capacitor, V_{dd} is the supply voltage, a is the average or expected number of transitions per clock cycle also known as the switching activity, and f is the clock frequency. Therefore, varying these parameters can affect dynamic power as well as overall energy consumption.

However, these parameters are dependent on one another. This means some sort of trade-off has to happen in order to come to an optimised solution.

D. Graphs and Scheduling Algorithms

The CDFG consists of two inter-weaved graphs, a so called Control-Flow-Graph (CFG) and a Data-Flow- Graph (DFG). The graphs are defined as follows:

Definition 1 (Control-Flow Graph (CFG))**.** *Let $G = (V_c, E_c)$ be a directed graph, with the set of nodes $V_c = \{v_1, v_2, \ldots, v_n\}$. Each v_i is a control object of a given algorithm. An edge (a_s, a_t) with $a_s, a_t \in V_c$ describes the transition from one control object to another. This can also be interpreted as the transition from one state of the system to the following state.*

Definition 2 (Data-Flow Graph (DFG))**.** *Let $G = (V_d, E_d)$ be a directed graph, with the set of nodes $V_d = \{v_1, v_2, \ldots, v_n\}$. Each v_i corresponds to an operation, a fork, or a join of a given algorithm. An edge (a_s, a_t) with $a_s, a_t \in V_d$ describes the data dependencies between the different nodes.*

According to [3], scheduling algorithms can be classified in unconstrained and constrained algorithms. Unconstrained scheduling are to be used:

- when resources are cheap or wiring is the bottleneck
- when every operation uses a different type of resource
- when resource assignment is done before scheduling
- To estimate the best and worst possible execution time (ASAP & ALAP)

As- Soon-As-Possible (ASAP) and As-Late-As-Possible (ALAP) scheduling are examples of unconstrained scheduling. ASAP solves the minimum latency scheduling problem which means scheduling all operations in a way that respects dependencies and finishes as early as possible. ASAP gives the earliest start times possible and if we have a time limit, we can use it to check whether the tasks can finish on time. On the other hand, ALAP schedules each task as late as possible, without missing the overall time limit. It helps understand the latest time each task can start while still meeting the total time goal. Mobility of a task is the difference between its ALAP and ASAP start times. It helps in resource allocation and optimizing performance vs area. When mobility is 0, it means that tasks don't have flexibility meaning they must start at a specific time.

The power scheduler discussed in this paper uses ASAP and ALAP scheduling to calculate some necessary timing information. However, the use of a path-based analysis technique groups the power scheduling algorithm in miscellaneous scheduling algorithms.

III. Power Scheduler

The main goal of the developed low power approach is to partition the design during the scheduling task of the HLS. Each partition allows the integration of dedicated turn-on and turn-off mechanisms into the design. That means, if a partition is not active, it can be turned off to reduce power consumption. The partitioning is described in detail in the following sections. The methodology and experimental results are given in subsection 3.3 and 3.11.

The purpose of this section is to discuss a Power Scheduler developed in [4]. This section is divided into seven subsections. Subsection III-A discusses in brief the work flow of the power scheduler. Subsection III-B discusses the methods used by the power scheduler to control power. Subsection III-C introduces mathematical formation of the cost function used to compare partitioned energy consumption to un-partitioned design. Thereafter in subsection III-D, three different phases of path determination is shown in an example. In subsection III-E, using the paths determined in previous subsection, a table is formed to produce the compatibility which in turn is used to form the clique in subsection III-F. Finally, in subsection **??**

A. Overview

Fig. 1 shows the design flow for HLS. The aim of the work done in [4] is to integrate low power methods within the scheduling process of HLS. The developed system, known as the power scheduler, is embedded into HLS design flow.The Power scheduler, a path-based scheduling technqiue, replaces the existing scheduler of a HLS system. Therefore, it is an alternative scheduling technique to integrate power reduction.

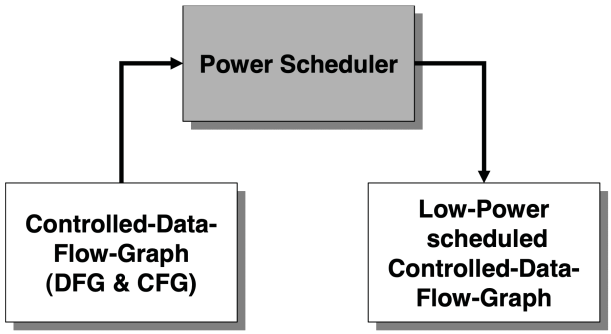

Fig. 4. Power Scheduler [4]

It can be seen in fig. 4 that an unscheduled CDFG as an input produces a scheduled low-power CDFG. The CDFG represents the controller for the DFG and the DFG is the data-path of the given algorithm. Each node in CDFG correspond to operations and each directed edge represent data dependency.

The first goal is to have dedicated turn-on and turn-off mechanisms into the design and the second goal is to minimize the additional control componenets for the activation and deactivation of the partitions. This could be achieved by combining partitions.

Let us assume that the target system consists of n components $\{k_1, \ldots, k_n\}$. At each time step, a component k_i, where $i \in \{1, \ldots, n\}$, is either active or inactive. The goal is to group these components into m partitions $\{p_1, \ldots, p_m\}$, where each partition p_j, with $j \in \{1, \ldots, m\}$, contains a subset of components $\{k_s, \ldots, k_r\}$. The primary objective is to minimize the number of partitions m, as a smaller number of partitions reduces the control overhead required for activation and deactivation.

It all starts with a High-Level specification which is transformed into behavioral VHDL source code. These descriptions are transformed into internal formats known as the CDFG. This CDFG, which describes the design, is the input to the Power Scheduler. This Power Scheduler reads the CDFG, consisting of CFG and DFG, and store the graphs into internal data format. After that the power scheduler uses ASAP and ALAP on the DFG to calculate the mobility of each operation within the DFG. The next step is path calculation. In this step, the system is trying to identify which parts of the design (paths) are active or inactive at different times during operation. Instead of just turning off unused parts at each step, all the possible paths of activity ahead of time are analyzed. This is done in three phases:

- Disjoint paths: paths that never work at the same time. Some of these paths are not active during the entire runtime of the system.
- Fork and join nodes: where the circuit splits and joins meaning paths that do not run at the same time. 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.
- Control paths: paths that happen based on decisions. 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 are 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. Each path corresponds to a node in a so called compatibility graph. The edges in the compatibility graph shows if the nodes are compatible with each other. That means, compatible nodes can be combined to a partition that can be turned on and off. Thereafter, a clique search algorithm is implemented that is used for finding cliques in the compatibility graph. Finally, a clique builds a partition than can be activated or deactivated during the run-time to save energy. During this step the CDFG will also be scheduled.

Turning something on or off in hardware uses extra power in both the circuit and the control logic. So, instead of controlling each tiny path separately, the scheduler groups compatible paths together into partitions. These partitions can be activated/deactivated efficiently. Nevertheless, it could be possible that after the clique approach a partition contains only one path.

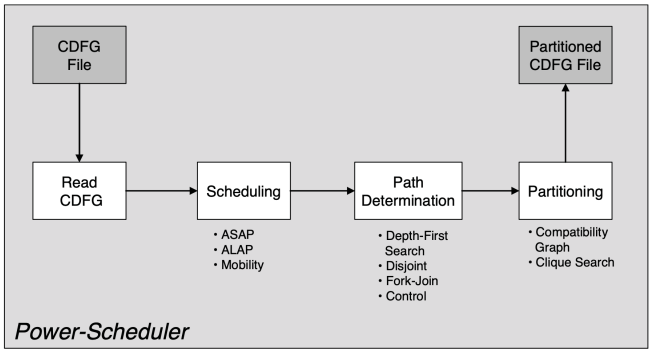

Fig. 5. Step of the power scheduler [4]

B. Power Reduction Methods

In this section we describe the low power reduction methods that are used within the Power Scheduler. It is possible to combine all methods or to apply only one method for a design.

- Gated Clock: To reduce dynamic power gated clocks can be applied. The clock signal goes to an AND gate. If the clock enable signal is set to true the clock is directed to the storage elements of the logic block. Hence, only storage elements are driven with a clock signal and no arithmetic logic. An ALU (arithmetic logical unit) needs a clock for an internal carry register. Nevertheless, if the clock is not directed to the storage elements the switching activity is set to zero and that reduces the dynamic but not the static power consumption.
- A guard in this case corresponds to a storage element (like registers) with write enable signal, see figure 3.5. If the write enable signal is set to false the registers don't write the incoming data to the output. Therefore, there is now switching activity in the logic block behind the register and this reduces the dynamic power consumption similar to the gated clock technique.
- With power down is it possible to reduce the dynamic and the static power consumption. Power pins can be separately controlled. That means, if the power goes down for one area there is no switching activity. This reduces the dynamic power consumption, but besides this the static power is also reduced, because everything is switched off.

C. Cost Function

Equation 2 shows the formula for dynamic power dissipation. The power of a gate is given by equation 3

$$P_{\text{dynamic}} = \frac{1}{2} \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} \tag{3}$$

where C_L is the capacity of the node and f_{clk} is the clock frequency. Therefore it can be said that dynamic power of the code can be calculated by simply multiplying the switching activity, α_{node} of the node. During HLS process, the switching activity is not easy to predict, therefore α_{node} is assumed as 1.

Summing up the dynamic power of all nodes gives the total power dissipation of a design as can be seen in equation 4:

$$P_{\text{dyn,tot}} = \sum_{i=1}^{n} P_{\text{dyn,i}} \tag{4}$$

where n is the total number of nodes in the design.

Equation 3 can be used to calculate the dynamic power dissipation for a partition which itself consists of a number of nodes. Hence, the dynamic power dissipation for a partition p is given by equation 5:

$$P_{\text{dyn,p}} = \sum_{j=1}^{m} P_{\text{dyn,j}} \tag{5}$$

where m are the number of nodes inside the partition.

Each partition has control unit responsible for activation or deactivation of the entire partition. This control unit is always active instead of the partitions that are being controlled. Therefore, it is necessary to add the costs of this partition control unit. The power cost of a partition control unit, $P_{gc,p}$, can be calculated by equation 4. In equation 4, nodes were used to calculate total power dissipation. Replacing the number of nodes n by the number of partitions p in the entire design leads to the calculation of the total dynamic power dissipation

of the design by taking additional partitioning costs into account. This is shown in equation 6:

$$P_{\text{design}} = \sum_{j=1}^{p} P_{\text{dyn,p}} + P_{gc,p} \tag{6}$$

The run-time of the system is not taken into account in 6. By taking the run-time of the system into account, the delay d of entire design and of each partition is calculated using the power-delay product. Equation 7 shows the power-delay product for a partition k:

$$PD_k = (P_{\text{dyn},k} \cdot d_k) + (P_{\text{gc},k} \cdot d_l) \tag{7}$$

whereas d_k is the delay of the partition and d_i is the delay of the part of the circuit where partition k is embedded.

The power-delay product of the entire design can now be calculated by summing up the power-delay product of all partition using the equation 8:

$$TPD = \sum_{x=1}^{l} PD_x, \tag{8}$$

whereas l is the number of design partitions.

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, this equation 8 is used as a cost function by [4].

D. Path Analysis

The path analysis is the third step of the power scheduler as descirbed in II-C. Path and path-time can be defined with the following definitions:

Definition 3 (PATH)**.** *A path p, with $i \in \mathbb{N}$, is a connection from a source node v_s to a destination node v_e to transport a data-word within the DFG $G = (V_d, E_d)$, where 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: $p_{v_s,v_e,i} = \{v_j, \ldots, v_n\}$ with $j, n \in \mathbb{N}$ and index $i \in \mathbb{N}$.*

Definition 4 (PATH-TIME)**.** *Let $time(p_{v_s,v_e,i})$ represent 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 (DPS), which traverses or searches graph data structures starting at the root node and explores as far as possible along each branch before backtracking. Extra memory, usually a stack, is needed to keep track of the nodes discovered so far along a specified branch which helps in backtracking of the graph.

1) Disjoin Paths: To find all disjoint paths, the DFG is slightly modified 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.

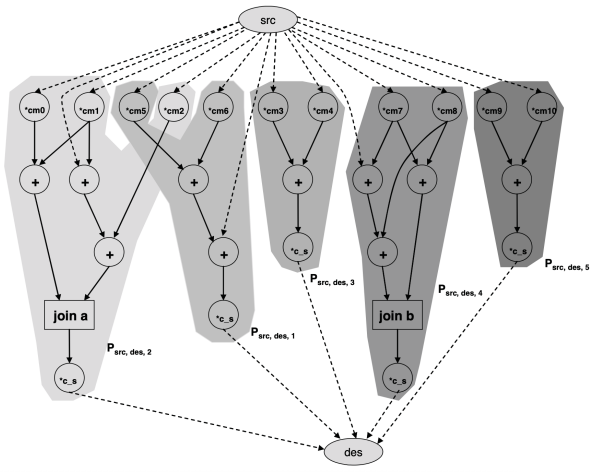

Fig. 6. Disjoint path [4]

From fig. 6, the following 5 disjoint paths have been identified:

$$\text{psrc,des,1} = \{*cm5, *cm6, +, +, *c\,s\}$$
$$\text{psrc,des,2} = \{*cm0, *cm1, *cm2, +, +, +, \text{join a}, *c\,s\}$$
$$\text{psrc,des,3} = \{*cm3, *cm4, +, *c\,s\}$$
$$\text{psrc,des,4} = \{*cm7, *cm8, +, +, +, \text{join b}, *c\,s\}$$
$$\text{psrc,des,5} = \{*cm9, *cm10, +, *c\,s\}$$

2) Fork and Join Nodes: For paths between fork and join nodes, the analysis starts from the fork towards the join node and all visited nodes to the path are added. If nodes with additional inputs or outputs are found on the path, they are followed to their primary inputs or outputs after which all visited nodes to the path are added. Already visited nodes are ignored. Assuming that operation * needs two timesteps and operation + one, the paths times are calculated using fig. 7 as an example, where $P_{fork,join,i}$ (with i = 1 ...3):

- $P_{fork,join,1}$ = {*cm2, + , +}, time($p_{fork,join,1}$) = 4
- $P_{fork,join,2}$ = {*cmo}, time($p_{fork,join,2}$) = 2
- $P_{fork,join,3}$ = {*cm1,+,*c_s}, time($p_{fork,join,3}$) = 5

3) Control Nodes: Control nodes are examined by scheduling ASAP algorithm to allow the identification of alternative paths. Once more, those paths are the basis of the partitioning and they are combined to partitions. Fig. 8 shows the same diagram as in fig. 6 however to evaluate paths for fork and join nodes. All paths from the join nodes to the primary inputs or constants are examined. The figure shows two paths of node join a. As *cm1 is connected to both paths, it is not a member of the paths.

The following path and respective time flow is found from fig. 8:

- $P_{joina,src,1}$ = {*cm0 , +}, time($P_{joina,src,1}$) = 2

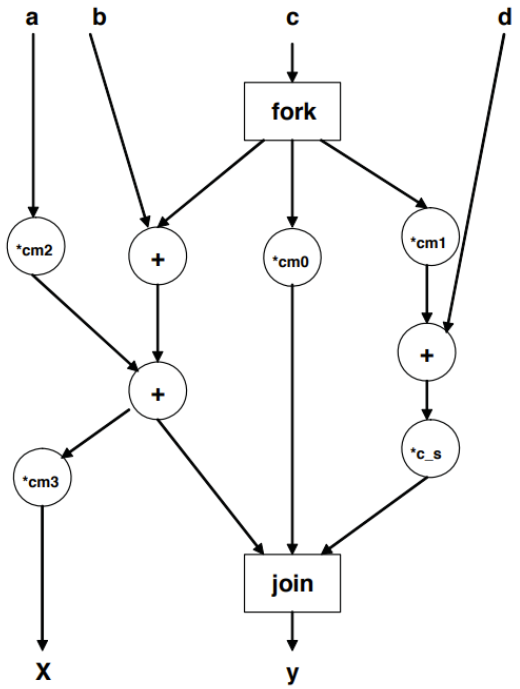

Fig. 7. Example for paths between fork and join node [5]

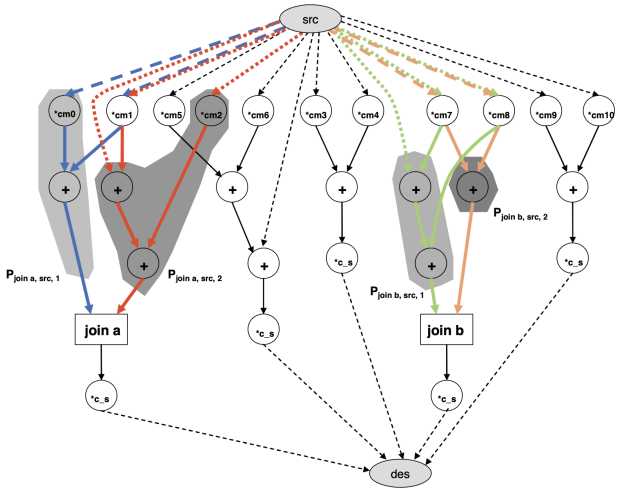

Fig. 8. Paths of join a and join b [4]

- $P_{joina,src,2}$ = {*cm2 ,+, +}, time($P_{joina,src,2}$) = 2
- $P_{joinb,src,1}$ = {+ , +}, time($P_{joinb,src,1}$) = 2
- $P_{joinb,src,2}$ = {+}, time($P_{joinb,src,2}$) = 1

E. Compatibility Graph

Definition 5 (Compatibility Graph (CO)). *Let $G = (V_k, E_k)$ be an undirected graph, where the set of nodes $V_k = \{v_1, v_2, \ldots, v_n\}$ corresponds to the number of paths in the CDFG (Control and Data Flow Graph). An edge (v_i, v_j) with $v_i, v_j \in V_k$ exists in E_k if there is no conflict between the nodes v_i and v_j.*

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 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 is used to find cliques in the compatibility graph.

$$PC_{p_{x,y,i},p_{a,b,j}} = \begin{cases} \text{true} & \text{when } |time(p_{x,y,i}) - time(p_{a,b,j})| \leq \beta \\ & \quad \wedge \left|st_{p_{x,y,i}} - st_{p_{a,b,j}}\right| \leq \varepsilon \\ & \quad \wedge \quad p_{x,y,i} \text{ and } p_{a,b,j} \text{ are timing and} \\ & \qquad\qquad \text{dataflow compatible} \\ \text{false} & \text{else} \end{cases}$$

That means, if $PC_{p_{x,y,i},p_{a,b,j}}$ is true , then there is an edge between node $p_{x,y,i}$ and $p_{a,b,j}$ in the compatibility graph.

The Path Mobility is the minimal mobility of all nodes from the path. Hence, that we don't take into account the influence of other paths in this definition. Therefore, a path itself as a union can only be moved by node with the minimal mobility. Path copies are also generated for similar paths with different start times. For $p_{x,y,i}$, the following paths can be generated $p'_{x,y,i}$ and $p''_{x,y,i}$ with a path mobility of 2.

From fig. 6 and fig. 8, the following table can be summarised:

Path	Path Mobility	Start Times	Path Copies
$p_{src,des,1}$	1	0, 1	$p'_{src,des,1}$
$p_{src,des,2}$	0	0	
$p_{src,des,3}$	2	0, 1, 2	$p'_{src,des,3}$, $p''_{src,des,3}$
$p_{src,des,4}$	0	0	
$p_{src,des,5}$	2	0, 1, 2	$p'_{src,des,5}$, $p''_{src,des,5}$
$p_{join\ a,src,1}$	1	0, 1	$p'_{join\ a,src,1}$
$p_{join\ a,src,2}$	1	0, 1	$p'_{join\ a,src,2}$
$p_{join\ b,src,1}$	0	1	
$p_{join\ b,src,2}$	1	1, 2	$p'_{join\ b,src,2}$

TABLE I
PATH PROPERTIES AND CORRESPONDING COPIES [4]

Therefore, the following table can be generated showing path compatibility:

As already discussed that each path in the compatibility graph is a node and edge between two nodes exist if they have no conflict, the following the compatibility graph can be generated:

F. Clique Search

A clique is a group of nodes where every node is connected to every other node. The maximum clique is the largest such group that can be found in the graph.

The compatibility graph (CG) is built from compatible paths in the CDFG. Each clique in the CG becomes a control partition, which can be independently powered down when inactive — minimizing overall energy consumption.

Path	Compatible Paths
$p_{src,des,1}$	$p_{src,des,2},\ p_{src,des,4}$
$p'_{src,des,1}$	$p_{src,des,2},\ p_{src,des,4}$
$p_{src,des,2}$	$p_{src,des,1},\ p'_{src,des,1},\ p_{src,des,4}$
$p_{src,des,3}$	$p_{src,des,5}$
$p'_{src,des,3}$	$p'_{src,des,5}$
$p''_{src,des,3}$	$p''_{src,des,5}$
$p_{src,des,4}$	$p_{src,des,1},\ p_{src,des,2}$
$p_{src,des,5}$	$p_{src,des,3}$
$p'_{src,des,5}$	$p'_{src,des,3}$
$p''_{src,des,5}$	$p''_{src,des,3}$
$p_{join\ a,src,1}$	$p_{join\ b,src,1},\ p_{join\ b,src,2},\ p'_{join\ b,src,2}$
$p'_{join\ a,src,1}$	$p'_{join\ a,src,1}$
$p_{join\ a,src,2}$	$p'_{join\ a,src,1}$
$p'_{join\ a,src,2}$	$p'_{join\ a,src,1}$
$p_{join\ b,src,1}$	$p'_{join\ a,src,1}$
$p_{join\ b,src,2}$	$p'_{join\ a,src,1}$
$p'_{join\ b,src,2}$	$p'_{join\ a,src,1}$

TABLE II
PATHS AND THEIR COMPATIBILITIES [4].

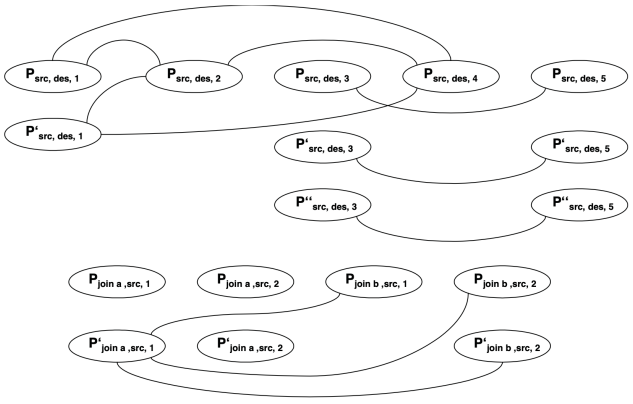

Fig. 9. Compatibility Graph [4]

Two strategies are implemented in [4] to ensure no path appears in more than one partition and paths with timing or dataflow conflicts are removed after placing a clique. Example: If a node $p'_{\text{src,des,3}}$ is selected in a clique, then $p_{\text{src,des,3}}$, $p''_{\text{src,des,3}}$ (its copies), and all conflicting paths are also deleted. The strategies are : 1) finding all cliques, 2) finding the maximum clique, delete its nodes and all conflicting nodes, and repeat until no more cliques are found.

G. Result

Table 3 shows the five partitions and the dynamic power consumption by each partition. Using 8 as the cost function, PowerDelay for unpartitioned design is $1\,180\,000\,\mu$J. For the partitioned design, the PowerDelay is $1\,008\,000\,\mu$J. Therefore an energy reduction of 14.6% is achieved.

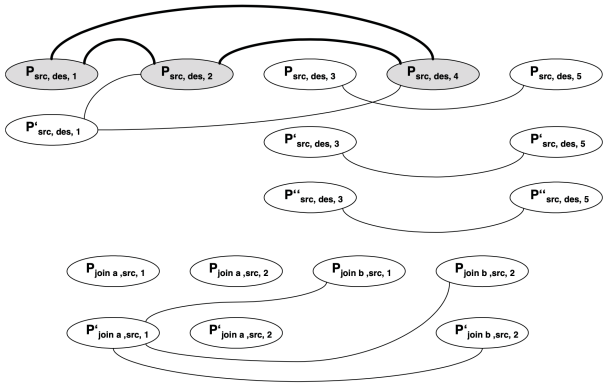

Fig. 10. Clique containing $p''_{\text{src,des,1}}$, $p''_{\text{src,des,2}}$, $p''_{\text{src,des,4}}$ [4]

Name	Partition	Power consumption $P_{dynamic,partition}$
A_1	$\{p_{src,des,1},\ p_{src,des,2},\ p_{src,des,4}\}$	$780\,\mu W/MHz$
A_2	$\{p_{src,des,3},\ p_{src,des,5}\}$	$400\,\mu W/MHz$
A_3	$\{p_{join\ b,src,1},\ p'_{join\ a,src,1}\}$	$120\,\mu W/MHz$
A_4	$\{p_{join\ a,src,2}\}$	$100\,\mu W/MHz$
A_5	$\{p_{join\ b,src,2}\}$	$20\,\mu W/MHz$

TABLE III
PARTITIONS AND POWER CONSUMPTION [4].

IV. CONCLUSION

In this paper,an approach for low power driven synthesis is disussed. The Power Scheduler, introduced and developed, by [4] is the primary focus of the paper. This scheduler reads a CDFG and writes a scheduled and partitioned CDFG. Besides the standard scheduling approaches, a path determination is applied which is the basis for the design partitioning. Each partition allows the integration of dedicated turn-on and turn-off mechanisms into the design. That means, if a partition is not active, it can be turned off to reduce power consumption and therefore energy. With the proposed power methods both the dynamic and the static power consumption is reduced by using power down.

ACKNOWLEDGMENT

Most, if not all, contents of this paper are taken from the Phd. paper of Prof. Dr. Achim Rettberg cited as [4]. His guidance in class has allowed for clear and concise understanding of the importance of design at system level to to be able to reduce power using High Level Synthesis.

REFERENCES

[1] Allen C-H Wu Daniel D. Gajski Nikil D. Dutt and Steve Y-L Lin. *High Level Synthesis*. Kluwer Academics, 1992.

[2] M.C. McFarland, A.C. Parker, and R. Camposano. "The high-level synthesis of digital systems". In: *Proceedings of the IEEE* 78.2 (1990), pp. 301–318. DOI: 10.1109/5.52214.

[3] Saraju P. Mohanty. “High level synthesis techniques for low power design’ a tuto- rial review”. In: *IPSJ Transactions on System and LSI Design Methodology* (2001).

[4] Achim Rettberg, Bernd Kleinjohann, and Franz Rammig. “Low Power Driven High-Level Synthesis for Dedicated Architectures”. In: (Jan. 2006).

[5] Achim Rettberg and Franz Rammig. “Integration of Energy Reduction into High-Level Synthesis by Partitioning”. In: *From Model-Driven Design to Resource Management for Distributed Embedded Systems*. Ed. by Bernd Kleinjohann et al. Boston, MA: Springer US, 2006, pp. 225–234. ISBN: 978-0-387-39362-9.

[6] H.J.M. Veendrick. “Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits”. In: *IEEE Journal of Solid-State Circuits* 19.4 (1984), pp. 468–473. DOI: 10.1109/JSSC.1984.1052168.