Research Article

Power and Execution Time Optimization through Hardware Software Partitioning Algorithm for Core Based Embedded System

Siwar Ben Haj Hassine,¹ Mehdi Jemai,¹ and Bouraoui Ouni^{1,2}

¹Laboratory of Electronic and Microelectronic, Faculty of Sciences at Monastir, University of Monastir, 5000 Monastir, Tunisia

²Networked Objects Control & Communication Systems Laboratory, National Engineering School of Sousse, BP 264, Sousse Erriadh, 4023 Sousse, Tunisia

Correspondence should be addressed to Siwar Ben Haj Hassine; siwar.haj.hassine@gmail.com

Received 18 August 2016; Revised 8 January 2017; Accepted 24 January 2017; Published 19 February 2017

Academic Editor: Manlio Gaudioso

Copyright © 2017 Siwar Ben Haj Hassine et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Shortening the marketing cycle of the product and accelerating its development efficiency have become a vital concern in the field of embedded system design. Therefore, hardware/software partitioning has become one of the mainstream technologies of embedded system development since it affects the overall system performance. Given today's largest requirement for great efficiency necessarily accompanied by high speed, our new algorithm presents the best version that can meet such unprecedented levels. In fact, we describe in this paper an algorithm that is based on HW/SW partitioning which aims to find the best tradeoff between power and latency of a system taking into consideration the dark silicon problem. Moreover, it has been tested and has shown its efficiency compared to other existing heuristic well-known algorithms which are Simulated Annealing, Tabu search, and Genetic algorithms.

1. Introduction

The exponential rise of embedded systems, all along with the persistent quest for higher levels of performance have resulted in the necessity of creating efficient types of embedded circuits. In fact, the embedded systems have become the worldwide leader technologies since they have penetrated into the human life to a very large extent. Besides, they play a vital role in industries as well as military applications which requires the necessity of having faster and better performing systems. Unfortunately, most of current technologies have only managed to further increase the system's capacity in order to have a faster treatment at the cost of a considerable simultaneous augmentation in their power. However, excessive power consumption may damage the integrated circuits through overheating, limiting the degree of transistors integration on a chip, bringing problem signal integrity, shortening battery durability for portable devices, and requiring expensive cooling and packaging systems. Moreover, the huge dependence

of wastage power consumption on threshold voltage has limited further threshold and provide voltage scaling. Thus, the power consumption is rising with technology scaling, such that it can no longer be cooled down profitably considering the physical limitations forced by cooling technologies and packaging. This gives rise to the dark silicon problem [1–3]. The concept of dark silicon is based on constraint that important fraction of transistors on chip cannot be powered on at a nominal voltage for a specific thermal design power TDP budget and have to be power-gated or simply remain dark. The TDP is the maximum amount of power provided to a chip while maintaining the chip temperature under the thermal safe temperature. In case where the TDP is exceeded, the temperature of the chip will rise beyond the cooling capacity that will throttle the chip. Previous studies [1, 2] have predicted that 50% to 80% of the chip area will be dark for GPU and CPU based systems. To overcome such dilemmas, designers' efforts have been increased to produce less consuming systems. In this context, some research groups have focused

on the creation of new architectures in terms of the material [4] while other groups have focused on extending batteries life cycle [5]. Yet, such solutions require high resources that several research groups do not have. For that, other methods have appeared in order to offer a less power consuming system such as hardware/software partitioning [6, 7].

Traditionally, partitioning was carried out manually which requires a detailed knowledge of circuit operations from designers. Such manual approaches were limited only to small designs with small number of constituent blocks [8, 9]. Since digital systems have become much more sophisticated, automatic HW/SW partitioning has become a necessity. In fact, many research groups have opted for the HW/SW partitioning in order to increase the performance of a system as presented in approaches such as [10, 11]; most of these approaches purposes are to meet performance constraints while keeping the system cost (area) as low as possible. Unfortunately, none of them took the power consumption and the execution time into consideration. Hence, we present in this paper an algorithm that finds a possible HW/SW partitioning of a data flow graph that finds out a tradeoff between power and latency taking into account the dark silicon problem.

The rest of the paper is organized as follows: Section 2 reviews the related literature; the proposed partitioning algorithm is addressed in Section 3 followed by an illustrative example; the numerical experimentation and discussion theorem are presented in Section 5 and finally the article ends up with the conclusion that briefs the present findings and future research on this theme.

2. Related Work

Recently, a new alternative technology that combines logic elements and memory along with an intellectual property processor core has emerged to remedy the excessive need for better performance systems. This technology called System on Programmable Chip SoPC allows and facilitates the SW/HW partitioning.

As generally reckoned, embedded systems consist of programmable software part (SW) and application specific hardware part (HW). Software part is much easier to develop and modify, and it consumes less power compared to the hardware part but it requires extra time to give final response. In fact, compared to the software which is less expensive in terms of cost and power consumption, the hardware provides better performance because it offers a faster treatment. For that reason, the purpose of HW/SW partitioning is to design a balanced system that accomplishes all system constraints [12]. Most of formulations of HW/SW partitioning dilemma have proven to be NP-hard [13, 14]. In fact, many exact algorithms have been proposed such as Branch-and-Bound [15], dynamic programming [16], and integer linear programming [17]. However, these exact algorithms tend to be quite slow for bigger inputs. Hence, for bigger partitioning problem heuristic algorithms have been the basis for the majority of researches such as Genetic algorithm (GA) [18], Tabu Search [19, 20], Simulated Annealing [21], Particle Swarm Optimization [22, 23], Ant algorithm [24, 25], shuffled frog leaping

algorithm [26], and greedy algorithm [27]. Other designers have mixed two heuristic algorithms to solve the HW/SW partitioning problems like in [28] where authors have used hybrid algorithm of Genetic algorithm (GA) and Tabu Search one, while others [29] have mixed the Discrete Particle Swarm Optimization (DPSO) and Branch-and-Bound (B&B) algorithms to meet the same aim. Besides, authors in [30] have proposed a new heuristic solution based on HW/SW partitioning that aims to reduce the execution time of the overall circuit. Moreover, authors in [31] have come up with a new IVA-HD which is a programmable, true multistandard, and full HD video coding engine that adopts HW/SW partitioning to achieve the low power and area equipment of the OMAP 4 processor. To attain the same goal of power optimization, [32] has proposed a minimizing approach based on mapping clusters of instructions to a core that yields a high utilization rate of resources and thus minimizes power consumption. Such a method has offered a less consuming system at the cost of an additional hardware overhead. The problems that these previously mentioned works have met are either to optimize one parameter at the cost of another important constraint or to focus on achieving the optimization of only one constraint such as power or execution time. Also, none of them have mentioned the dark silicon problem. In fact, the dark silicon has become a critical issue for designers since it can decrease the reliability in the nanoera [33–35] and leads to soft errors, aging and even process variations [36, 37]. Recent works have explored the dark silicon problem by applying a very low voltage to power on more cores [38] and proposed new accelerators architectures [39, 40]. Almost, the majority of works have handled the dark silicon problem on low level codesign which necessitates a good knowledge of the target circuit and extra time of marketing cycle of the product. Other designers have proposed new architectures by exploiting architectural heterogeneity [41–43]. However, such solutions require high resources that several research groups do not have. In the literature, only few works have combined the HLS and dark silicon problem due to its complexity [44, 45]. It is true that generally the dark silicon problem appears for multiprocessor system-on-chip (MPSoc). But, due to the Soc huge rise the dark silicon problem must be taken into consideration even with one core based embedded system [46]. Motivated by the fact and coming across the shortages of other researches, it has been vital to come up with a new idea of developing a new algorithm that aims to create a less consuming system and a faster one without influencing the system reliability.

3. Problem's Definitions

We consider the applications that can be modeled using data flow graph (DFG). A data flow graph that is used to create a preliminary overview of the system denoted as $G(V, E)$, where $V = \{v_1, v_i, \dots, v_n\}$, is the set of vertices or nodes that are interconnected to each other by edges $E = \{\{v_i, v_j\}, \{v_i, v_{j+1}\}, \dots\}$. Edges of the graph present the dependencies between the components of the system. In general, the node of the graph can represent a basic block [47], a short of

instruction [48], a procedure or a function [49], and so on. In this paper, we use four different types of nodes:

- (i) A start or an end node v_i^{startend} ; $v_i^{\text{startend}} \in V^{\text{startend}}$ and $V^{\text{startend}} \subseteq V$.
- (ii) A node that includes simple code v_i^S ; $v_i^S \in V^S$ and $V^S \subseteq V$.
- (iii) A node that contains the beginning of a control-construct v_i^B ; $v_i^B \in V^B$ and $V^B \subseteq V$.
- (iv) A node that contains the end of a control-construct v_i^E ; $v_i^E \in V^E$ and $V^E \subseteq V$.

3.1. Partitions' Types. The graph partitioning is to cut the graph into possible partitions $P_{\text{all}} = \{P_1, P_2, \dots, P_n\}$ where P_{all} is the set of all possible partitions; P_i is a possible partition; and n is the number of possible partitions.

There exist two kinds of partitions:

- (i) A control-construct partition that includes a whole construct such as **if to end if**, **case to end case**, and so on.
- (ii) A mix partition that could contain either two or more control-constructs or one or more control-construct combined with a simple node (that contains simple construct such as addition operation).

3.2. Node's Links. To facilitate the search of control-construct partitions, we have used the parameter of link. If the node is a beginning of a control-construct or an end of a control construct, the link value equals 1. For the rest of node's types, the link value equals 0. The link definition can be defined as follows:

$$\text{Link}(v_i) = \begin{cases} 1, & \text{if } v_i \in V^B \text{ or } v_i \in V^E \\ 0, & \text{else.} \end{cases} \quad (1)$$

3.3. Related Statements. When a task is realized by hardware or software, its execution time and power consumption show diverse values. We define the following functions $H_L(P_i)$, $S_L(P_i)$, $H_P(P_i)$ and $S_P(P_i)$ to represent the hardware latency, the software latency, the hardware power, and the software power respectively of a given partition P_i . Although obtaining the exact values of the execution time and power consumption is a challenging problem, it is beyond the scope of this article. Rather, we focus on algorithmic issues in partitioning.

Given a path $P_w: v_i \rightarrow v_j$ and a hardware/software partitioning P_i for all the nodes in P_w , the completion time of P_w under partitioning P_i is the summation of all the latencies occurred on P_w taking into consideration the parallel execution of some tasks. The system completion time is defined to be the completion time of a critical path C_p in DFG.

The hardware latency and the software latency corresponding to a target partition can be written as follows:

$$H_L(P_i) = H_L(C_P(P_i)) = \max_{v_i \in P_i} \left[(X_m(i) H_L(v_i)) + \sum_{v_j \in P_i} ((X_m(j) * \psi_{ij}) H_L(v_j)) \right], \quad (2)$$

where $\psi_{ij} = 1$, if v_j depends to v_i ; 0, else; and $\{X_m(i) = 0$, if v_i is executed by Software; $X_m(i) = 1$, if v_i is executed by Hardware};

$$S_L(P_i) = \sum_{v_i \in P_i} S_L(v_i). \quad (3)$$

For a given $G(V, E)$, we define a vector X_m to indicate either the task is realized by hardware or software. For instance, for a node v_i , $X_m(i)$ equals 0 if the task is executed by the software and equal to 1 if it is realized by the hardware.

The power consumption of the system with respect to a given partitioning can be calculated as the summation of all the task power consumption of each node realized by software or hardware. In fact, it can be written as follows:

$$H_P(P_i) = \sum_{v_i \in P_i} H_P(v_i),$$

$$S_P(P_i) = \sum_{v_i \in P_i} S_P(v_i). \quad (4)$$

So, to recapitalize, we define the hardware/software partitioning problem as follows: given $G(V, E)$ and thermal design power, find a partitioning that offers the best tradeoff between total power and execution time of the system.

4. Proposed Algorithm

Our algorithm is meant to achieve graph partitioning in order to find the best compromise between power and execution time. As generally reckoned, the software consumes less power than the hardware but it requires more time to give response while the hardware which tackles the problem of timing consumes more power. This approach starts with a system totally implemented by software, it will not consume power but it will be too tardy. Whenever a partition of the system migrates to be executed by the hardware, the system will consume more power and become faster. As mentioned previously, our algorithm includes two different kinds of partitions. Its first function is to search for all control-construct partitions (Algorithm 2) and then it builds the mix partitions. After that, it makes all possible combinations between the generated partitions (Algorithm 1).

In case where all nodes are simple, the partitions will simply take all possible combinations of the nodes. Our algorithm (Algorithm 3) is based on three functions. F_1 provides the total latency of the system for each generated partition while F_2 computes the total consumed power of the

```

// generate the mix partitions
For all  $P_i \in P_{\text{all}}$ 
  If  $\forall_{v_i \in V} \text{Link}(\text{pred}(v_{\text{start}p}) = 0)$  then //  $v_{\text{start}p}$  is the start node of the control-construct partition
     $P_k \leq \{\text{pred}(v_{\text{start}p}), P_i\}$ 
    If  $(P_k \neq \emptyset)$ 
       $P_{\text{all}} \leq P_k$ 
       $k = k + 1$ ;
    End if
  else
    If  $(P_{i-1} \in P_{\text{control}})$ 
       $P_k \leq \{P_{i-1}, P_i\}$ ;
      If  $(P_k \neq \emptyset)$ 
         $P_{\text{all}} \leq P_k$ 
         $k = k + 1$ 
      End if
    End if
  End if
  If  $\forall_{v_i \in V} \text{Link}(\text{succ}(v_{\text{end}p}) = 0)$  then //  $v_{\text{end}p}$  is the end node of the control-construct partition
     $P_k \leq \{P_i, \text{succ}(v_{\text{end}p})\}$ 
    If  $(P_k \neq \emptyset)$ 
       $P_{\text{all}} \leq P_k$ 
       $k = k + 1$ ;
    End if
  else
    If  $(P_{i+1} \in P_{\text{control}})$ 
       $P_k \leq \{P_i, P_{i+1}\}$ ;
      If  $(P_k \neq \emptyset)$ 
         $P_{\text{all}} \leq P_k$ 
         $k = k + 1$ 
      End if
    End if
  End if
End for;
// find the combination of the previous generated partitions
 $P_{\text{test}} \leq P_{\text{all}}$ 
For all  $P_i \in P_{\text{test}}$ 
   $z \leq i + 1$ 
  For all  $P_z \in P_{\text{test}}$ 
    if  $P_i \notin P_z$  then
       $P_k \leq \{P_i, P_z\}$ ;
       $P_{\text{all}} \leq P_k$ 
       $z \leq z + 1$ 
       $k \leq k + 1$ 
    else
       $z \leq z + 1$ 
    end if
  end for
if  $P_{\text{control}} = \{\emptyset\}$  then
  while  $(P_i \neq V)$  do
     $P_k \leq \text{make all possible combinations}$ 
     $P_{\text{all}} \leq P_k$ 
     $k = k + 1$ 
  end
end if
End begin

```

ALGORITHM 1: Generation of partitions.

```

Begin
For all  $v_i \in V$ 
  Compute  $\text{Link}(v_i)$ 
End for
 $k = 1; k \in \mathbb{N}$ 
// generation of all control construct partitions
For all  $v_i \in V^B$ :
  If  $\forall_{v_i, v_j \in V} (\text{Link}(v_i) = 1 \text{ and } v_j \notin V^B)$  then
     $P_k \leq v_i$ 
    do
       $P_k \leq v_j$ ;
    While  $(\text{Link}(v_j) = 1)$ 
     $P_{\text{all}} \leq P_k$ 
     $P_{\text{control}} \leq P_k$ 
     $k = k + 1$ 
  End if
End for

```

ALGORITHM 2: Generation of control-construct partitions.

system under a given partition. When the algorithm becomes so close to the best solution, it will offer us an interval that includes some suggestions. F_1 and F_2 are written as follows:

$$F_1(P_i) = [S_L(C_P) - S_L(C_P(P_i)) + H_L(C_P(P_i))], \quad (5)$$

where $S_L(C_P) = \sum_{v_i \in C_P} S_L(v_i)$, $S_L(C_P(P_i))$ and $H_L(C_P(P_i))$ are the software and the hardware latency of the critical path of the partition P_i , respectively.

$$F_2(P_i) = [S_P(G) - S_P(P_i) + H_P(P_i)], \quad (6)$$

where $S_P(G) = \sum_{v_i \in G} S_P(v_i)$.

To avoid the dark silicon problem, we have introduced a new constraint called thermal design power TDP. This constraint refers to the maximum amount of power that can be provided to a chip while maintaining the chip temperature under the thermal safe temperature. If the system power under a specific partition exceeds the TDP, then that partition will not be among the suggested ones. Thus, ensure the good performance and the reliability of the system.

F_3 is introduced to facilitate taking the decision of which the suggested partition is the best one. F_3 is introduced as follows:

$$F_3(P_i) = \frac{F_2(P_i)}{F_1(P_i)}. \quad (7)$$

The best solution equals the closest value to

$$\frac{1}{m} \times \sum_{i=1}^m F_3(P_i), \quad (8)$$

where m is the number of suggested solutions.

5. Illustrative Example

To further clarify our algorithm, it has been applied on the graph shown in Figure 1. The node's constraints $V = \{v_1, v_i, \dots, v_{15}\}$ are presented in Table 1.

FIGURE 1: The data flow graph.

The first step consists in finding all possible paths and calculating the software latency of all paths in order to get the critical path. In our case C_P : $S_L(C_P) = 45$. The second step is to compute the link of each node as Table 2 presents. Then, all possible partitions will be generated.

In our case, there exist eleven partitions $P_{\text{all}} = \{P_1, P_2, \dots, P_{11}\}$ as Table 3 describes.

For each generated partition the latency as well as the consumed power will be calculated using the functions F_1 and F_2 respectively (Table 4). For instance, P_1 does not belong to C_P so the latency of the system will stay the same “45.” However, with partitions whose nodes belong to C_P the latency of the graph can change such as in the case of P_2 and in such circumstances we have to be aware that the critical path changes as well. The final decision is taken using F_3 . The pace of function F_1 and F_2 and the best partition are shown in Figure 2.

TABLE 1: Node's parameters.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
H_L	3	2	1	2	1	2	3	1	1	3	3	1	5	4	2
S_L	6	4	2	4	2	4	7	2	2	7	6	2	10	9	5
H_p	2	1	2	1	1	2	2	1	1	3	3	1	4	4	5
S_p	0.5	0.25	0.5	0.25	0.25	0.5	0.5	0.25	0.25	1	1	0.25	1	1.5	2

TABLE 2: The links of nodes.

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
Link	0	1	0	0	1	0	0	1	1	0	0	1	0	1	0

```

Begin
  Find all paths  $P_w$ 
  Calculate  $S_L(\text{all}) \leq \{S_L(P_{w1}), S_L(P_{w2}), \dots, S_L(P_{wm})\}$ 
  Find  $C_p$ 
  Compute  $S_L(C_p)$ 
  Generate all partitions
  For all  $P_i \in P_{\text{all}}$  do
    Find  $C_p(P_i)$ 
    if  $(v_i \in P_i) \notin C_p$  then
       $F_1 = S_L(C_p)$ 
    else
       $S_L(\text{new}) \leq S_L(\text{all})$  //  $S_L(\text{new})$  is the set of the new latency of each path
      For all  $P_w$  do
        Compute  $S_L(P_w)$ ; // the new latency of the path  $S_L(P_w) = F_1(P_w)$ ;
      end for
      update  $S_L(\text{new})$ 
       $F_1(P_i) \leq \max[S_L(\text{new})]$ 
    end if
    Compute function  $F_2(P_i)$ 
  End for
  // choose the best partitions//
   $m \leq 1$ 
  TDP  $\leq$  the dark power
  For all  $P_i \in P_{\text{all}}$  do
    For all  $v_h \in P_i$  do
      if  $(v_h \in C_p \text{ and } v_{h+1} \notin C_p)$  then
        if  $F_2(P_i) < \text{TDP}$  then
           $P_{\text{suggested}} \leq P_i$ 
        end if
      end for
    For all  $P_i \in P_{\text{suggested}}$  do
      Compute function  $F_3(P_i)$ 
    End for
     $m \leq \text{number of partition}$ 
    For  $i$  from 1 to  $m$  do
       $P_{\text{best}} \leq \frac{1}{m} \times \sum_{i=1}^m F_3(P_i)$ 
    End for
  End.

```

ALGORITHM 3: Our proposed algorithm.

TABLE 3: The generated partitions.

Partitions	Nodes														
P_1	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_2	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_3	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_4	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_5	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_6	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_7	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_8	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_9	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_{10}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
P_{11}	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}

TABLE 4: The calculation results.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}
F_1	45	41	45	41	45	39	35	39	39	34	34
F_2	14.5	15.5	15.25	18.5	16.75	20	23	20.75	22.25	23.75	25.25
F_3	0.32	0.38	0.39	0.45	0.37	0.51	0.66	0.53	0.57	0.7	0.74
$P_{\text{suggested}}$	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}
$(1/m) \times \sum_{i=1}^m F_3(P_i)$						0.62					
Best partition						P_7

FIGURE 2: The pace of functions F_1 and F_2 .

We have to mention that the used TDP value in the previous example is 26. In case where TDP value equals only 22, then the suggested partitions will only be the set of $\{P_6, P_8\}$ and the best partition will be P_6 instead of P_7 .

6. Experiment Results

To prove the efficiency of our algorithm, we have implemented a comparative study that is meant to illustrate the amount of power consumed by a given application and its execution time by the use of our algorithm in comparison to three existing heuristic algorithms which are the Simulated Annealing (SA), Tabu Search (TS), and Genetic algorithms (GA). To put that into practice we have applied our approach on 8-point Discrete Cosine Transform (DCT) [50] (Figure 6), 16-point DCT [51] (Figure 7), and H.264 [52]. The DCT is the most intensive part of the CLD algorithm. The H.264 is a video coding format that is one of the most used formats for compressing, recording, and distributing of video content. The characterizations of the three previous applications are presented in Table 5.

The hardware power values are based on the results provided in [53] and [54] for the DCT and the H.264, respectively. The software power is almost negligible compared to the hardware power. The latency of hardware (FPGA) equals one-third to one-fifth of the latency of software (processor) [55]. The TDP value equals 7W for the rest of the comparison.

Table 6 illustrates the design results provided by our approach for the 8-point DCT, 16-point DCT, and H.264 applications.

6.1. Comparison with Other Algorithms and Discussion. Tables 7, 8 and 9 summarize the design results provided by Simulated Annealing, Tabu Search, and Genetic algorithms, respectively.

TABLE 5: The characterization of each application.

		Number of nodes	
8-point DCT		42	
	16 multiplication operations		26 addition operations
16-point DCT		224	
	128 multiplication operations		96 addition operations
H.264		28	

TABLE 6: Design result provided by our algorithm.

8-point DCT	Number of partitions		458	
	Best partition	Hardware	12 operations of ($\times$)	19 operations of (+)
		Software	4 operations of ($\times$)	7 operations of (+)
	Total power consumed (mW)		82.35	
	The system's latency (ns)		453	
16-point DCT	Number of partitions		12512	
	Best partition	Hardware	68 operations of ($\times$)	91 operations of (+)
		Software	60 operations of ($\times$)	5 operations of (+)
	Total power consumed (mW)		525.876	
	The system's latency (ns)		2780	
H.264	Number of partitions		24	
	Total power consumed (W)		6.850	
	The system's latency (ns)		1530	

TABLE 7: Design result provided by Simulated Annealing algorithm.

8-point DCT	Best partition	Hardware	14 operations of ($\times$)	4 operations of (+)
		Software	2 operations of ($\times$)	22 operations of (+)
	Total power consumed (mW)		91.55	
	The system's latency (ns)		586	
16-point DCT	Best partition	Hardware	60 operations of ($\times$)	48 operations of (+)
		Software	68 operations of ($\times$)	48 operations of (+)
	Total power consumed (mW)		487.78	
	The system's latency (ns)		3024	
H.264	Total power consumed (W)		6.250	
	The system's latency (ns)		1740	

TABLE 8: Design result provided by Tabu Search algorithm.

8-point DCT	Best partition	Hardware	13 operations of ($\times$)	0 operations of (+)
		Software	3 operations of ($\times$)	26 operations of (+)
	Total power consumed (mW)		86.8	
	The system's latency (ns)		493	
16-point DCT	Best partition	Hardware	40 operations of ($\times$)	44 operations of (+)
		Software	88 operations of ($\times$)	52 operations of (+)
	Total power consumed (mW)		393.712	
	The system's latency (ns)		3776	
H.264	Total power consumed (W)		5.276	
	The system's latency (ns)		1990	

TABLE 9: Design result provided by Genetic algorithm.

8-point DCT	Best partition	Hardware	12 operations of ($\times$)	22 operations of (+)
		Software	4 operations of ($\times$)	4 operations of (+)
	Total power consumed (mW)			84.4
	The system's latency (ns)			630
16-point DCT	Best partition	Hardware	60 operations of ($\times$)	90 operations of (+)
		Software	68 operations of ($\times$)	6 operations of (+)
	Total power consumed (mW)			501.652
	The system's latency (ns)			2896
H.264	Total power consumed (W)			6.050
	The system's latency (ns)			1650

TABLE 10: Design results.

	W_{worst}	Simulated Annealing	Tabu Search	Genetic algorithm	Our algorithm	W_{best}
8-point DCT	73.482	53.648	54.648	40.206	37.304	8.379
16-point DCT	3335	1475.04	1486.65	1558	1461.9	531.831
H.264	27462	10875	10499.24	10560	10480.5	3432.75

To compare our algorithm to the previously mentioned algorithms, we have introduced a comparison metric called Ω . It is obvious that the worst case W_{worst} presents when the system consumes the highest value of power (purely hardware) and takes too much time (purely software). We assume that $W_{\text{worst}} = H_{p\text{max}} \times S_{L\text{max}}$. The best case W_{best} takes place when the system consumes less power and responds faster: $W_{\text{best}} = H_{p\text{min}} \times S_{L\text{min}}$.

$$\Omega = F_1 \times F_2 = H_{p\text{system}} \times S_{L\text{system}}, \quad (9)$$

where $W_{\text{worst}} < \Omega < W_{\text{best}}$.

According to (9), when Ω is close to W_{best} , the solution is better. Table 10 presents the different values of Ω .

Based on Table 10, we deduce that our algorithm offers less and closest rates to the best case, whereas using Simulated Annealing, Tabu Search, and Genetic algorithms the values of Ω are too high and far from the best state. Thus, we admit that our algorithm offers the best tradeoff between the two critical parameters: power consumption and system execution time.

7. Conclusion

Given today's requirements for less consuming systems accompanied by high speed, the necessity of creating more efficient types of embedded systems has been persisted. One of the most elegant solutions that provides system's optimizations is the HW/SW partitioning. For that, we have developed a new algorithm based on HW/SW partitioning in order to obtain the best tradeoff between power and latency taking into account the dark silicon problem. Our algorithm has been applied and tested to Simulated Annealing, Tabu Search, and Genetic algorithms and as the research has illustrated, we admit that our algorithm is best suited for the urgent achievement of the desired combination of high speed and less power in core based embedded systems.

Competing Interests

The authors declare that they have no competing interests.

References

- [1] Z. Hadi Esmaeil, E. Blem, R. A. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*, pp. 365–376, ACM, San Jose, Calif, USA, June 2011.
- [2] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. 4, pp. 6–15, 2011.
- [3] M. B. Taylor, "Is dark silicon useful?: harnessing the four horse-men of the coming dark silicon apocalypse," in *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*, pp. 1131–1136, ACM, San Francisco, Calif, USA, June 2012.
- [4] Silicon Laboratories, <http://www.silabs.com/Pages/default.aspx>.
- [5] Lawrence Berkeley National Laboratory (Berkeley Lab), <http://www.lbl.gov/>.
- [6] T. Yen and W. Wolf, *Hardware-Software Co-Synthesis of Distributed Embedded Systems*, Springer US, Boston, Mass, USA, 1996.
- [7] J. Staunstrup and W. Wo, *Hardware/Software Co-Design: Principles and Practice*, October 1997, https://books.google.tn/books?hl=fr&lr=&id=yKXzBwAAQBAJ&oi=fnd&pg=PR16&dq=Hardware/Software+Co-Design:+Principles+and+Practice&ots=FFzq9utmwi&sig=TiLf_EKndx8SU_Ffg6Hx32teqBw&redir_esc=y#v=onepage&q=Hardware%2FSoftware%20Co-Design%3A%20Principles%20and%20Practice&f=false.
- [8] F. Cloute, J.-N. Contensou, D. Esteve, P. Pampagnin, P. Pons, and Y. Favard, "Hardware/software co-design of an avionics communication protocol interface system: an industrial case study," in *Proceedings of the 7th International Conference on Hardware/Software Codesign (CODES '99)*, pp. 48–52, Rome, Italy, May 1999.

- [9] B. Mei, P. Schaumont, and S. Vernalde, "A hardware/software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems," in *Proceedings of the 11th IEEE Program for Research on Integrated Systems and Circuits*, Veldhoven, The Netherlands, 2000.
- [10] S. Dimassi, M. Jemai, B. Ouni, and A. Mtibaa, "Hardware-software partitioning algorithm based on binary search trees and genetic algorithm to optimize logic area for SOPC," *Journal of Theoretical & Applied Information Technology*, vol. 66, no. 3, pp. 788–794, 2014.
- [11] M. Jemai, S. Dimassi, B. Ouni, and A. Mtibaa, "Optimization of logic area for System on Programmable Chip based on hardware-software partitioning," in *Proceedings of the International Conference on Embedded Systems and Applications (ICESA '14)*, Hammamet, Tunisia, March 2014.
- [12] J. Teich, "Hardware/software codesign: the past, the present, and predicting the future," *Proceedings of the IEEE*, vol. 100, pp. 1411–1430, 2012.
- [13] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 136–156, 2005.
- [14] W. Jigang, T. Srikanthan, and G. Chen, "Algorithmic aspects of hardware/software partitioning: ID search algorithms," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 59, no. 4, pp. 532–544, 2009.
- [15] K. S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, pp. 193–208, 2002.
- [16] J. Wu and T. Srikanthan, "Low-complex dynamic programming algorithm for hardware/software partitioning," *Information Processing Letters*, vol. 98, no. 2, pp. 41–46, 2006.
- [17] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, "Integrating physical constraints in HW-SW partitioning for architectures with partial dynamic reconfiguration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 11, pp. 1189–1202, 2006.
- [18] K. Anil and C. H. Shampa, "Design optimization using genetic algorithm and cuckoo search," in *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT '11)*, pp. 1–5, IEEE, 2011.
- [19] J. Wu, P. Wang, S.-K. Lam, and T. Srikanthan, "Efficient heuristic and tabu search for hardware/software partitioning," *The Journal of Supercomputing*, vol. 66, no. 1, pp. 118–134, 2013.
- [20] J. Wu, T. Srikanthan, and T. Lei, "Efficient heuristic algorithms for path-based hardware/software partitioning," *Mathematical and Computer Modelling*, vol. 51, no. 7–8, pp. 974–984, 2010.
- [21] Y. Jing, J. Kuang, J. Du, and B. Hu, "Application of improved simulated annealing optimization algorithms in hardware/software partitioning of the reconfigurable system-on-chip," in *Parallel Computational Fluid Dynamics: 25th International Conference, ParCFD 2013, Changsha, China, May 20–24, 2013. Revised Selected Papers*, vol. 405 of *Communications in Computer and Information Science*, pp. 532–540, Springer, Berlin, Germany, 2014.
- [22] S.-A. Li, C.-C. Hsu, C.-C. Wong, and C.-J. Yu, "Hardware/software co-design for particle swarm optimization algorithm," *Information Sciences*, vol. 181, no. 20, pp. 4582–4596, 2011.
- [23] J. Wu, T. Srikanthan, and G. Chen, "Algorithmic aspects of hardware/software partitioning: ID search algorithms," *IEEE Transactions on Computers*, vol. 59, no. 4, pp. 532–544, 2010.
- [24] T. He and Y. Guo, "Power consumption optimization and delay based on ant colony algorithm in network-on-chip," *Engineering Review*, vol. 33, no. 3, pp. 219–225, 2013.
- [25] Y.-D. Zhang, L.-N. Wu, G. Wei, H.-Q. Wu, and Y.-L. Guo, "Hardware/software partition using adaptive ant colony algorithm," *Control and Decision*, vol. 24, no. 9, pp. 1385–1389, 2009.
- [26] T. Zhang, X. Zhao, Y.-K. Yu et al., "Research on hardware/software partitioning method of improved shuffled frog leaping algorithm," *Journal of Signal Processing*, vol. 9, article 003, 2015.
- [27] G. Lin, "An iterative greedy algorithm for hardware/software partitioning," in *Proceedings of the 9th International Conference on Natural Computation (ICNC '13)*, pp. 777–781, IEEE, Shenyang, China, July 2013.
- [28] G. Li, J. Feng, C. Wang, and J. Wang, "Hardware/software partitioning algorithm based on the combination of genetic algorithm and Tabu search," *Engineering Review*, vol. 34, no. 2, pp. 151–160, 2014.
- [29] T. Eimuri and S. Salehi, "Using DPSO and B&B algorithms for Hardware/Software partitioning in co-design," in *Proceedings of the 2nd International Conference on Computer Research and Development (ICCRD '10)*, pp. 416–420, May 2010.
- [30] H. Han, W. Liu, W. Jigang, and G. Jiang, "Efficient algorithm for hardware/software partitioning and scheduling on MPSoC," *Journal of Computers (Finland)*, vol. 8, no. 1, pp. 61–68, 2013.
- [31] M. Mehendale, S. Das, M. Sharma et al., "A true multistandard, programmable, low-power, full HD video-codec engine for smartphone SoC," in *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC '12)*, pp. 226–228, San Francisco, Calif, USA, February 2012.
- [32] J. Henkel, "Low power hardware/software partitioning approach for core-based embedded systems," in *Proceedings of the 36th Annual Design Automation Conference (DAC '99)*, pp. 122–127, New Orleans, La, USA, June 1999.
- [33] F. Kriebel, S. Rehman, D. Sun, M. Shafique, and J. Henkel, "ASER: adaptive soft error resilience for reliability-heterogeneous processors in the dark silicon era," in *Proceedings of the 51st Annual Design Automation Conference (DAC '14)*, ACM, San Francisco, Calif, USA, June 2014.
- [34] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors," in *Proceedings of the 16th Design, Automation and Test in Europe Conference and Exhibition (DATE '13)*, pp. 39–44, EDA Consortium, Grenoble, France, March 2013.
- [35] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon era," in *Proceedings of the 51st Annual Design Automation Conference (DAC '14)*, San Francisco, Calif, USA, June 2014.
- [36] J. Henkel, L. Bauer, N. Dutt et al., "Reliable on-chip systems in the nano-era: lessons learnt and future trends," in *Proceedings of the 50th Annual Design Automation Conference (DAC '13)*, ACM, Austin, Tex, USA, June 2013.
- [37] J. Henkel, L. Bauer, H. Zhang, S. Rehman, and M. Shafique, "Multi-layer dependability: from microarchitecture to application level," in *Proceedings of the 51st Annual Design Automation Conference (DAC '14)*, San Francisco, Calif, USA, June 2014.
- [38] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "Near-threshold voltage (NTV) design: opportunities and challenges," in *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*, pp. 1153–1158, San Francisco, Calif, USA, June 2012.

- [39] J. Cong, M. A. Ghodrati, M. Gill, B. Grigorian, and G. Reinman, "Architecture support for accelerator-rich CMPs," in *Proceedings of the 49th Annual Design Automation Conference (DAC '12)*, pp. 843–849, ACM, San Francisco, Calif, USA, June 2012.
- [40] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, "The accelerator store: a shared memory framework for accelerator-based systems," *Transactions on Architecture and Code Optimization*, vol. 8, no. 4, article no. 48, 2012.
- [41] J. Allred, S. Roy, and K. Chakraborty, "Designing for dark silicon: a methodological perspective on energy efficient systems," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '12)*, pp. 255–260, Redondo Beach, Calif, USA, August 2012.
- [42] H. Esmailzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA '11)*, pp. 365–376, San Jose, Calif, USA, June 2011.
- [43] Y. Turakhia, B. Raghunathan, S. Garg, and D. Marculescu, "HaDeS: architectural synthesis for heterogeneous dark silicon chip multi-processors," in *Proceedings of the 50th Annual Design Automation Conference (DAC '13)*, Austin, Tex, USA, June 2013.
- [44] M. Shafique, S. Garg, T. Mitra, S. Parameswaran, and J. Henkel, "Dark silicon as a challenge for hardware/software co-design," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES '14)*, ACM, New Delhi, India, October 2014.
- [45] D. Diamantopoulos, S. Xydis, K. Siozios, and D. Soudris, "Mitigating memory-induced dark silicon in many-accelerator architectures," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 136–139, 2015.
- [46] P. Mantovani, E. G. Cota, K. Tien et al., "An FPGA-based infrastructure for fine-grained DVFS analysis in high-performance embedded systems," in *Proceedings of the 53rd Annual ACM IEEE Design Automation Conference (DAC '16)*, Austin, Tex, USA, June 2016.
- [47] P. V. Knudsen and J. Madsen, "PACE: a dynamic programming algorithm for hardware/software partitioning," in *Proceedings of the 4th International Workshop on Hardware/Software Co-Design (Codes/CASHE '96)*, pp. 85–92, IEEE, Pittsburgh, PA, USA, March 1996.
- [48] G. Stitt, F. Vahid, G. McGregor, and B. Einloth, "Hardware/software partitioning of software binaries: a case study of H. 264 decoder," in *Proceedings of the IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, pp. 285–290, New York, NY, USA, 2005.
- [49] J. R. Armstrong and P. J. M. B. Adhipathi Jr., "Model and synthesis directed task assignment for systems on a chip," in *Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems*, pp. 472–475, Cambridge, Mass, USA, 2002.
- [50] K. K. Parhi and T. Nishitani, *Digital Signal Processing for Multimedia Systems*, CRC Press, Boca Raton, Fla, USA, 1999.
- [51] R. Ayadi, B. Ouni, and A. Mtibaa, "A partitioning methodology that optimizes the communication cost for reconfigurable computing systems," *International Journal of Automation and Computing*, vol. 9, no. 3, pp. 280–287, 2012.
- [52] M. Jemai, S. Dimassi, B. Ouni et al., "Combined partitioning hardware-software algorithms," *International Journal of Computer Applications*, vol. 119, no. 4, pp. 11–15, 2015.
- [53] B. C. Sahoo, *Design and power estimation of booth multiplier using different adder architectures [Ph.D. thesis]*, National Institute of Technology, Rourkela, India, 2013.
- [54] B. A. B. Sarif, M. Pourazad, P. Nasiopoulos, and V. C. M. Leung, "A study on the power consumption of H.264/AVC-based video sensor network," *International Journal of Distributed Sensor Networks*, vol. 2015, Article ID 304787, 10 pages, 2015.
- [55] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware HW-SW partitioning for reconfigurable architectures with partial dynamic reconfiguration," in *Proceedings of the 42nd Design Automation Conference (DAC '05)*, pp. 335–340, ACM, Anaheim, Calif, USA, June 2005.

Hindawi

Submit your manuscripts at
<https://www.hindawi.com>