School of Electrical, Electronic & Computer Engineering

A Design Methodology for Transient Peak Power Modulation in Energy Harvesting Circuits

Yu Zhou, Terrence Mak, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2010-156

July 2010

Contact:

yu.zhou@ncl.ac.uk

terrence.mak@ncl.ac.uk

alex.yakovlev@ncl.ac.uk

NCL-EECE-MSD-TR-2010-156

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

`http://async.org.uk/`

A Design Methodology for Transient Peak Power Modulation in Energy Harvesting Circuits

Yu Zhou, Terrence Mak, and Alex Yakovlev

Abstract

An energy harvesting system (EHS) delivers a non-deterministic power density over a range of explicit environmental conditions. The computational architecture is required to be tunable and optimized at run-time in order to adapt the power supply and, simultaneously, deliver optimal performance. In this paper, an important aspect of the supply-consumption relation in EHS is considered, that the transient peak power consumption of the load should be bounded by the energy supply rate, yet the average power utilisation should be maximised. A design flow is proposed in this paper for adjusting the concurrency degree of a system according to the available power, and choosing a run-time schedule for an EHS satisfying the optimisation purpose. In particular, the concept of a scheduling decision graph has been introduced for dynamic scheduling and power adaptation. Algorithms for deriving this graph from a system's data flow relations are proposed. A run-time schedule for the system is then extracted from the decision graph, using a proposed simple and optimal method. Finally, the effects of our design flow is demonstrated on modulating the average/peak power consumption by a FIR filter circuit implemented in FPGA.

1 Introduction

Traditional battery-powered system works under limited energy supply. For applications that require long working duration, energy becomes a critical bottleneck and much effort has been devoted to energy efficient or low-power system design [4, 7]. With advances in microelectromechanical (MEMS) technology, it is possible to implement a self-powered system that harvests ambient energy from the environment [6, 12, 10, 1]. Several different ambient sources have been exploited, including solar [6], electromagnetic [12] and mechanical piezoelectric vibration [8, 2]. Such energy harvesting system provides a promising alternative to battery-powered system and creates an opportunity for architecture and design method innovation for the exploitation of ambient energy source.

The design criteria for systems in using an energy harvesting source are fundamentally different from that in using a battery. The battery-based system benefits from a relatively predictable metric of energy residual, suffices to characterize the energy availability, and is seemingly an unbounded power supply. Traditional low-power system design aims to minimize the average power dissipation in order to increase the power-up duration of the device. For energy harvesting systems, rather than a limited energy supply, it has a limit on the power at which the energy can be used. Also, the harvested energy supply from the ambient sources is stochastic in nature and, thus, a more sophisticated characterization and design metric are required for the energy harvesting circuits.

An energy harvesting system presents a different specification and the system design goal should be aiming at a perennial operation of utilizing the harvested energy at an appropriate rate. Transducer in such a system is employed to maximize the power transfer from the ambient energy sources to the loads, by optimizing the apparent impedance of the load presented to the harvesters. Although it is challenging to achieve

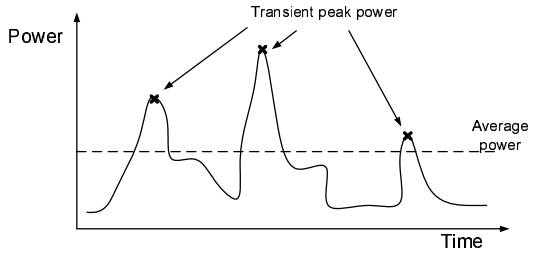

Figure 1: Power profile of an arbitrary computational electronic. The transient peak power corresponds to a large number of simultaneous activities in high degree of concurrency. The average power is a lump statistics of the power usage over a long period of time.

perfect compatibility between the output of the transducer and load electronics, various power electronic techniques, such as rectifier [14] and charge pump [15] design, and impedance matching tuning [9], have been proposed to maximize the power transfer. However, researchers have only been using simple resistive load model, such as simple resistors and capacitors [8], to determine and optimize the transducer design. Computational architectures such as Field Programmable Gate Arrays (FPGAs) have a much more complicated load structure with temporal and dynamic power requirements that presents a challenge to the optimal design specification for energy harvesting circuits. The power supply from energy harvesters is limited and varies with time. Although power regulators aim to stabilise and deliver a constant power supply, there is an upper bound for the transient power delivered to the computational electronics. Transient peak power (see Fig. 1) extracted from the harvester that is over the upper bound of the harvested power would result in a synchronization failure as a consequence of the momentary voltage drop and deceleration of the arithmetic operations. Moreover, the temporal energy buffer or storage in an energy harvester is usually limited. In contrast to low-power circuit design principles, the computational load in an energy harvesting circuit consumes energy at an appropriate rate that is compatible to the harvester, or namely energy-neutral operation [6], in which both the computational performance and energy buffering is optimized.

In this paper, we propose a design methodology based on dynamic scheduling that enables a run-time concurrency adjustment subjected to the power supply upper bound of the harvester. By tuning the degree of concurrency or parallelization in a computational architecture, the transient peak power consumption of the computational electronic can be readily controlled to maintain a reliable computation and to maximize the compatibility of the harvester.

Conventional dynamic methods of modulating power consumption include dynamic voltage frequency scaling (DVFS[5]). In energy harvesting systems, however, voltages are subject to instability in spite of the existence of power/voltage regulators, and scaling of voltages may be restricted in practice. As a result, this paper considers voltages as not adjustable at run-time, and rather, investigates on the effects of dynamic tuning of the concurrency degrees which is directly related to the capacitive loads in a system.

Adjustment of clock periods in a system (frequency scaling, or duty cycling) can also modulate the average power consumption of a system. However, we argue that its ability to modulate the peak power is limited when voltage is not adjustable. This is intuitively because when the average power is reduced by stretching the clock period, the peak power consumption in a clock cycle is still determined by the number of active components (operations) in a system, which is fixed with a particular concurrency degree. In contrast, dynamic scheduling can adjust the number of active components executed in a clock cycle, and thus modulates the peak power consumption more effectively.

The major contributions of this paper are:

1. A design fbw for modulating the average and peak power consumption of an EHS system, through run-time adjusting of the concurrency degree of a system using dynamic scheduling methods is presented. With the fbw, the data fbw graph (DFG) representing a system's high-level behaviour is transformed into a scheduling decision graph (SDG), which provides the different feasible schedules available in a system. Then a run-time schedule can be selected, based on the decision graph and according to the real-time power constraints.

2. Algorithms have been proposed to transform a DFG into its corresponding SDG. Two types of transformation algorithms are considered: a complete one which can produce all the possible schedules of a system, and a truncated one which only generates selected schedules according to certain scheduling policies.

3. An algorithm for run-time scheduling based on a SDG is presented. With the algorithm, an optimal schedule for the system can be selected during run-time which meets the transient peak power requirement and maximises the average power utilisation.

4. The design fbw is exemplified using a simple DSP algorithm.

5. The power dissipation and performance of a FIR filter circuit is evaluated using the dynamic scheduling approach, and comparisons are made with the results using duty cycle adjustment.

2 Preliminary

2.1 Problem formulation

The power adaptive problem can be expressed in the form of a constrained optimization problem. In order to maximize the utilization of the available harvested energy, the optimization objective is to minimize latency of the computation. Also, the transducer with temporal energy storage can provide power supply up to a certain transient maximum. For a computational power demand that is larger than the transient maximum point, the transducer and the computational architecture could be prompted to an error. Therefore, it is important to control the computational load that the power usage at each time stamp is less than the transient maximum power supply.

Suppose that the maximum transient power supply at time t from the transducer can be measured and this value is denoted by $\mathcal{P}(t)$. Also suppose that a computational task involves N arithmetic operations, $\alpha_i, i = 1, 2, ..., N$. For each operation, it can be realized with different degree of concurrency, c_i, that will result in different delay and power dissipation. The corresponding power dissipation for each of the operations with a specific concurrency degree is denoted by $Q(\alpha_i, c_i)$. Thus, the optimization constraint becomes, $Q(\alpha_i, c_i) \leq \mathcal{P}(t), \forall i = 1, 2, ..., N$. Also consider that an energy harvesting system usually has a limited energy buffer and it will be wasteful if the harvested energy is not utilized. It is, therefore, sensible to minimize the overall computational delay given the peak power constraint is satisfied. Given the delay of the i-th operation is denoted by $d(\alpha_i, c_i)$, the transient peak power constrained delay optimization problem can be expressed as

$$\text{minimize } \sum_{i}^{N} d(\alpha_i, c_i)$$

$$\text{subject to } Q(\alpha_i, c_i) \leq \mathcal{P}(t), \forall i = 1, 2, ..., N$$

where c_i is a variable in the optimization problem that specifies the degree of concurrency to realize the arithmetic operation.

There are different approaches to realize the degree of concurrency, c_i, such that the overall delay can be minimized. However, most of the existing approaches are based on design-time optimization schemes. Since the power constraint, $\mathcal{P}(t)$, can only be obtained at run-time, a run-time enabled optimization scheme is required. In this paper, we propose an approach that enables run-time modification of degree of concurrency by modifying the number of active components that are working simultaneously. The transient power has a direct and linear relationship with the number of identical operation working at the same time. By varying the degree of concurrency, the power constraint can be satisfied. Besides, the computational steps with different concurrency levels need to be rescheduled. Dynamic scheduling is, therefore, mandatory to enable such a run-time concurrency tuning scheme. In the following, we present a method to enable run-time delay optimization and satisfy the transient peak power supply constraint.

2.2 The concurrency tuning design flow

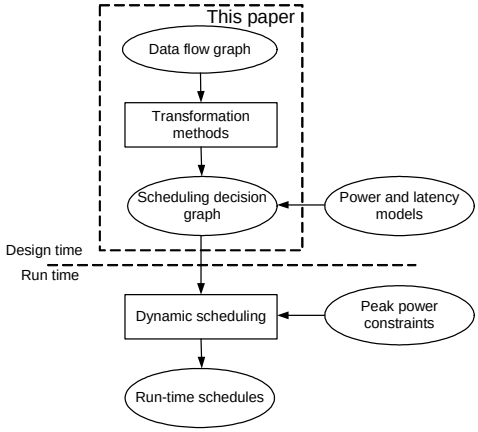

Figure 2: Design flow of an adaptive system comprising of different steps at design-time and run-time.

The objective of the design flow is to enable run-time adjustment of the concurrency degree of a system such that the peak power requirement for the computation is smaller than the maximum tolerable value. Especially, the idea of concurrency adjustment can be applied to different granularities including bit and operator level with different trade-offs in the control efficiency and hardware overhead. As a first attempt, we consider arithmetic operators as the basic unit in concurrency tuning which would provide a reasonable granularity in FPGAs to trade-off the control overhead.

An overview of the design flow for a run-time concurrency tuning system based on dynamic scheduling is illustrated in Fig. 2. The idea is that a computational algorithm in terms of a data flow graph (DFG) is transformed to a scheduling decision graph (SDG), which is a state-based graphic data structure providing different schedules of a system with different degrees of concurrency. A path from the SDG corresponds to a schedule to realize the algorithm subject to certain run-time constraints. During run-time, dynamic scheduling is applied to the SDG to produce a run-time schedule that minimizes delay subject to the power constraints.

The main focus of this paper, shown in the dotted box in Figure 2, is on the scheduling decision graph, and a transformation process to derive a SDG from the corresponding DFG. In particular, algorithms for the transformation process have been proposed. A transformation can be a complete one if all the possible

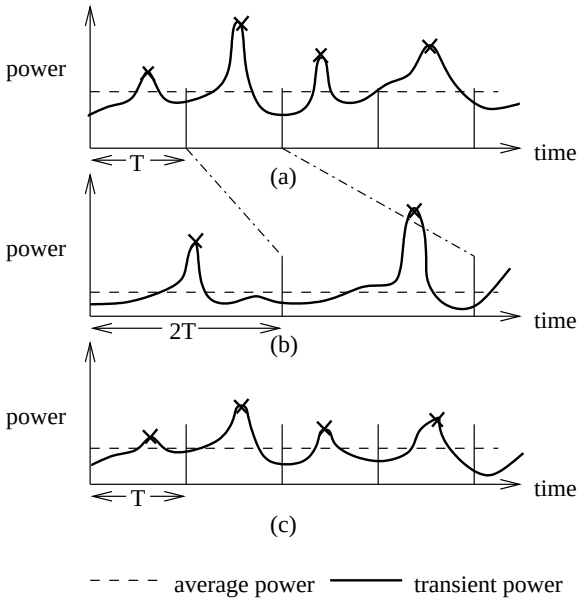

Figure 3: Impact of duty cycling (b) and dynamic scheduling (c) on the average and peak power modulation of the initial circuit (a)

schedules in the system are derived, or it can be a truncated one if only selected schedules are chosen. Truncated transformation based on heuristic scheduling policies reduces the complexity of a complete graph, but at a cost of optimality.

This paper also discusses how to perform dynamic scheduling based on SDGs, and in particular, an algorithm has been proposed for finding the run-time optimal schedule under power constraints. The algorithm has been applied to an intuitive example. The hardware/software implementation of the dynamic scheduling methods for EHS based on the decision graphs is out of the scope of this paper and will be investigated in the future.

2.3 Dynamic scheduling versus duty cycling

Figure 3(a) illustrates the average power (real curve) and transient power (dotted line) consumption of the initial system with the clock period of T, and a particular level of concurrency degree, i.e., a fixed number of operations that are allowed to run in the same cycle. The cross-bars represent peak power values in each clock cycle.

Figure 3(b) shows the effects on power modulation by stretching the clock period to $2 \times T$, but remaining the same concurrency degree as the initial circuit. The average power consumption is reduced compared with the initial system, as the idle time is increased. The peak power value, however, remains relatively the same because stretching of clock periods has little impact on the number of simultaneous operations during a clock cycle.

Figure 3(c) illustrates the influence on power modulation by reducing the concurrency degree of the initial system, i.e., re-scheduling the original schedule to a less concurrent (or equally, more sequential) one such that fewer number of operations is allowed to execute in the same clock cycle (the clock period remains T). As a result of this, both the average and peak power values are reduced compared with the original design.

3 From data flow graphs to scheduling decision graphs

Data flow graphs (DFG) have been vastly used for modelling and synthesising electronic systems as a high-level behavioral model. A DFG describes the data flow relations in a system, whereas for our purpose of dynamically finding the optimal schedules of a system, another type of graph (data structure) is more favorable. Scheduling decision graph (SDG) is proposed to this end, which stores the possible schedules of a system in a compact form that is determined at synthesis time. This section discusses the concept of scheduling decision graph (SDG), and a way of how to construct a SDG from a corresponding Data Flow Graph. Firstly, DFGs and its use in high-level scheduling are briefly reviewed.

A $DFG = (V, E)$ is a directed and acyclic graph. The vertex set $V = \{v_i; i = 0, 1, ..., n\}$ is in one-to-one correspondence with the set of operations in a system and the edge set $E = \{(v_i, v_j); i, j = 0, 1, ..., n\}$ represents data dependencies among the operations. Following the conventions in [3], each graph has a source node v_0 and sink node v_n. The source is the tail of directed edges to all those vertices representing the initial operations and the sink is the head of directed edges from all those representing the final operations. The operation set in a data flow graph has the labellings of D and P. $D = \{d_i; i = 0, 1, ..., n\}$ is the integer delay set where d_i is the delay of executing v_i on an appropriate resource type. $P = \{p_i; i = 0, 1, ..., n\}$ be the set of power consumption values associated with the operations. [1]

Conventional scheduling methods in high-level synthesis, such as as-soon-as-possible (ASAP), list scheduling [11], and force-directed scheduling [13], find one particular schedule subject to resource or timing constraints. A schedule labels the vertex set of a DFG with $T = \{t_i; i = 0, 1, ..., n\}$, which is the start times of the operations, i.e., the clock cycles in which the operations start. The latency of a schedule, L, is the number of cycles to execute the entire tasks, i.e., $L = t_n - t_0$. The average power consumption corresponding to L is $\frac{\sum_i (p_i \times d_i)}{L}$.

In contrast to the conventional scheduling methods which provide only one schedule, multiple schedules are available from a scheduling decision graph. The multiplicity comes from the choices of *concurrency degree* in executing the operations in a system, i.e., the number of operators that are scheduled in a clock cycle, and/or their different combinations.

3.1 Scheduling Decision Graphs

A Scheduling Decision Graph $SDG = (V, E, F)$ is a bi-polar directed acyclic graph (DAG), where V is the vertex set. Each $v \in V$ corresponds to a *schedule state* labelled by an integer set of timing stamps $t(v)$. A schedule state consists a set of *sub-operations*, which are defined as follows.

A *sub-operation* v_i^k is a multi-cycle operation $v_i \in DFG.V$ whose first k cycles have been executed at a particular time, where $0 \leq k \leq d_i$. Trivially, $v_i = v_i^0$.

A *schedule state* is a set of sub-operations that can be scheduled beginning from a clock cycle $l \in t(v)$. This indicates that any sub-operation v_i^k belonging to a schedule state $v1 \in SDG.V$ must have its data-dependency relation satisfied, i.e., $\forall v_j : (v_j, v_i) \in DFG.E$ and $t1 \in t(v1)$, the execution of v_j finishes by $t1$, so does the first-k cycles of v_i.

The source vertex of a SDG is the initial schedule state with $t(v) = \{0\}$, and the destination of a SDG contains no operations and is referred to *NULL*.

E is the edge set of a SDG where $e \in E$ represents a *schedule step*. A schedule step is a 3-tuple $< s(e), \lambda(e), p(e) >$, where $s(e)$ is the sub-operations arranged in a the step for execution, $\lambda(e)$ is the step length in terms of clock cycles, and $p(e)$ is the corresponding power consumption. A schedule step is a

[1] In general, D and P can have data-dependent values. In this paper, they are assumed not dependent on data for the sake of simplicity, and, in particular, source and sink nodes have zero delay and power consumption.

decision at a particular state to schedule $s(e)$ for a period of $\lambda(e)$, with power consumption of $p(e)$. The concurrency degree of a schedule step is the number of operations (or operations of a particular resource) in $s(e)$.

Flow of a SDG is expressed by $F \subseteq \{(V \times E) \cup (E \times V)\}$. In particular, $(v1, e) \cup (e, v2) \subseteq F$, where e is an output arc from state $v1$ ($e \in o(v1)$) and input arc to $v2$ ($e \in i(v2)$), indicates the following conditions (1)-(5).

(1) $s(e) \subseteq v1$;

(2) $\lambda(e) \leq d_i^k, \forall v_i^k \in s(e)$. d_i^k is the execution delay of v_i^k, and $d_i^k = d_i - k$;

(3) If $\lambda(e) = d_i^k$, then $v_i^k \notin v2$;

(4) If $\lambda(e) < d_i^k$, then $v_i^{k+\lambda(e)} \in v2$;

(5) $\forall t1 \in t(v1), \exists t2 \in t(v2) : t2 = t1 + \lambda(e)$.

Condition (1) and (2) ensure that $s(e)$ is a sub-set of the sub-operations in a schedule state $v1$, and that the schedule step is bound from above by the sub-operation with the shortest delay, respectively. A schedule step $e \in o(v1)$ satisfying (1) and (2) is called a *legal step* at state $v1$. Further, conditions (3)-(5) provide properties that must be satisfied when determining the output state of $v2$ from $v1$ via e.

SDG provides a graphical means for finding the possible schedules of a system. A *schedule* s is defined as chain of schedule steps of any path from the initial state s_0 to $NULL$ in a SDG, i.e., $s =< e_0, e_1, ..., e_n >$ where $e_0 \in o(s_0)$ and $e_n \in i(NULL)$, and $\exists v \in SDG.V : e_i \in i(v) \wedge e_{i+1} \in o(v), \forall i = (0, 1, ..., n-1)$. The latency of s is $\sum_{i=0}^{n} \lambda(e_i)$, and the average power consumption corresponding to s is $\frac{\sum_{i=0}^{n} p(e_i) \times \lambda(e_i)}{\sum_{i=0}^{n} \lambda(e_i)}$.

The total number of schedules in a SDG is determined by the number of different paths from s_0 to $NULL$, and the number of schedules with distinct latency is determined by $|t(NULL)|$.

3.2 Transformation from DFGs to SDGs

The transformation from a DFG to the corresponding SDG is *complete*, if all the legal schedule steps are formed at each schedule state when constructing the SDG. Algorithm 1 describes a complete transformation method. The method recursively generates the schedule states in a *depth-first* manner, beginning from the initial state s_0, which consists of the operations in a DFG without data dependency on others. The SDG is constructed when $explore(s_0)$ returns successfully. In the algorithms, $v.scheduled\_ops$ denotes the set of sub-operations that have been scheduled at a state v.

In Algorithm 1, the recursive subroutine $explore(v)$ determines all the legal schedule steps from a state v. $explore(v)$ works as follows. The power-set of the sub-operations in v, 2^v is calculated. Each schedule step e from v corresponds to an $ops \in 2^v$, [2] and is formed by setting $s(e) = ops$, $\lambda(e) = min(d_i^k : v_i^k \in ops)$, and $p(e) = \sum_i (p_i : v_i^k \in ops)$. $explore(v)$ returns either when all the legal steps of v are visited and their output states explored, or when $NULL$ is reached.

Once a schedule step e is determined, the output state from v via e, v', is computed by function $next\_state(v, e)$, according to Algorithm 2. The sub-operations in v' include those belonging to v but not scheduled in e (line 2 in Algorithm 2), the updated sub-operations in e with longer execution delays than the step length (line 7-10), and the operations whose data dependencies have been newly satisfied with e (line 13). $next\_state(v, e)$ also updates $v'.schedule\_ops$ (line 4-5), and $t(v')$ (line 14-17).

A SDG generated by a complete transformation produces all possible schedules of a system considering all different concurrency degrees, as well as the different combinations of the operations in a certain degree. This is reflected by the decision steps formed by the power-set of the operations at a schedule state. This exponential increase of arc and state numbers will however cause *state explosion* and make the transformation methods and SDGs intractable for even modest graph size. To control the complexity, *policies* can be

[2] Without loss of generality, an $ops = \emptyset$ corresponds to a self-loop step at v with 0 delay and power.

Algorithm 1 Complete transformation from a DFG to its SDG

1: **Inputs**: A $DFG = (V, E)$
2: **Outputs**: A $SDG = (V, E, F)$
3: **Main**:
4: $s_0 \leftarrow \{v_i \in DFG.V : \nexists v_j : (v_j, v_i) \in DFG.E\}$
5: $SDG.V \leftarrow SDG.V \cup s_0$
6: $SDG.E \leftarrow \emptyset$, $SDG.F \leftarrow \emptyset$,
7: $t(s_0) = \{0\}$
8: $s_0.scheduled\_ops \leftarrow \emptyset$
9: explore (s_0)
10: **End Main**
11: **Subroutine** explore (v) where $v \in SDG.V$
12: **if** $v = NULL$ **then**
13: return
14: **else**
15: **for all** $ops \in$ power-set(v) **do**
16: **form** the legal schedule step e based on ops
17: $SDG.E \leftarrow SDG.E \cup e$
18: $v' = next\_state(v, e)$
19: $SDG.F \leftarrow SDG.F \cup (v, e) \cup (e, v')$
20: **if** $v' \notin SDG.V$ **then**
21: $SDG.V \leftarrow SDG.V \cup v'$
22: explore(v')
23: **end if**
24: **end for**
25: return
26: **end if**

introduced during a *truncated* transformation methods where only certain concurrency degrees or operation combinations are allowed for the schedule steps.

For truncated transformation methods, only a subset of 2^v are selected at a schedule state v for constructing possible schedule steps, according to scheduling policies. An algorithm for truncated transformation replaces line 15 in algorithm 1 with

Select $ops \in 2^v$ according to the scheduling policies.

In particular, for each type of computation resource in a DFG/SDG, the following two constraints are considered for the scheduling policies in this paper.

(1) **constraints on concurrency degrees.** The number of operations belonging to a particular resource type that can be scheduled in a step equals the specified concurrency degree for that type.

(2) **constraints on combinations.** For a concurrency degree, only the specified combinations of operations are allowed to be scheduled in a step.

Example 1 *Truncated transformation of an intuitive example*

Figure 4 depicts a DFG of a simple algorithm for DSP, where $C1 = A1 * B1 + A2 * B2$*, and* $C2 = A3 * B1 + A4 * B2$*. In the DFG, multiplication (*1-4*) is assumed to take* 2 *units of delay and* 10 *units of power, and addition (*5-6*) takes* 1 *unit of delay and power, when implemented on hardware resources. Since the power consumption of addition is trivial compared with that of multiplication, we decide to define the concurrency degree purely based on the number of multiplications that can be scheduled in the same step.*

The following three policies are adopted when transforming the DFG into a truncated SDG.

(1) The output steps from a schedule state v *should reflect all possible concurrency degrees allowed at* v*. If* v_{max} *is the maximum number of multiplications in* v*, then for each* $1 \leq i \leq v_{max}$*, there should exist*

Algorithm 2 $v' = next\_state(v, e)$

1: $v'.scheduled\_ops \leftarrow v.scheduled\_ops$
2: $v' \leftarrow v - s(e)$
3: **for all** $v_i^k \in s(e)$ **do**
4: **if** $d_i^k = \lambda(e)$ **then**
5: $v'.scheduled\_ops \leftarrow v'.scheduled\_ops \cup v_i$
6: **else**
7: **if** $d_i^k > \lambda(e)$ **then**
8: $v' \leftarrow v' \cup v_i^{k+\lambda(e)}$
9: $d_i^{k+\lambda(e)} \leftarrow d_i^k - \lambda(e)$
10: **end if**
11: **end if**
12: **end for**
13: $v' \leftarrow v' \cup \{v_i \in DFG.V : \forall (v_j, v_i) \in DFG.E : v_j \in v'.scheduled\_ops \wedge v_i \notin v'.scheduled\_ops\}$
14: **if** $v' \notin SDG.V$ **then**
15: $t(v') \leftarrow \emptyset$
16: **end if**
17: $t(v') \leftarrow t(v') \cup (t + \lambda(e)), \forall t \in t(v)$

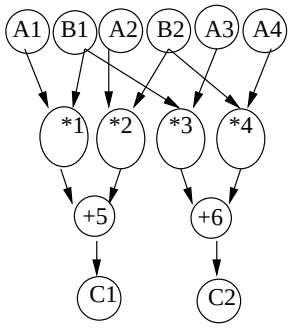

Figure 4: DFG of an intuitive DSP algorithm

exactly one output edge e from v with i multiplication scheduled, i.e., $|s(e)| = i$. If $v_{max} = 0$, then $s(e)$ includes all the sub-operations in v.

(2) The choice of the combination of i multiplications from v_{max} ones in a schedule step e , where $|s(e) = i|$ and $i = 1, 2, ..., v_{max}$, is according to the numerically increasing (lexicographical) order of the operations.

(3) $s(e)$ of an output edge e from v includes all the addition in v.

Following the transformation algorithms, the SDG of the example is derived and illustrated in Figure 5. In the SDG, each node is a schedule state labelled by the a set of sub-operations. In particular, operations $\{1, 2, 3, 4\}$ do not depend on other operations, and therefore form the initial state s_0 with the start-time stamp of $t = 1$. The four output edges from s_0 represent the 4 different concurrency degrees (1-4) at that state. For example, the right most output arc from s_0, representing concurrency degree 2, indicates the decision of scheduling $s(e) = \{1, 2\}$ for a length of $\lambda(e) = 2$, with a peak power consumption of $p(e) = 20$. Note that according to policy (2), multiplication 1 and 2 are selected rather than other pairs. Following this decision, a schedule state of $\{3, 4, 5\}$ is reached with $t = 3$, including both the unscheduled operations from previous state and the operation (5) whose data dependency is newly satisfied with that decision step.

A path from s_0 to NULL forms a schedule of the system. For example, $< (1, 2, 3, 4)/2/40, (5, 6)/1/2 >$ is one possible schedule, where $\{1, 2, 3, 4\}$ are scheduled for the first 2 cycles and $\{5, 6\}$ scheduled for the last one. This schedule, with latency of 3 and average power of 27.3, happens to be the shortest schedule

since no other paths in the SDG have smaller latency.

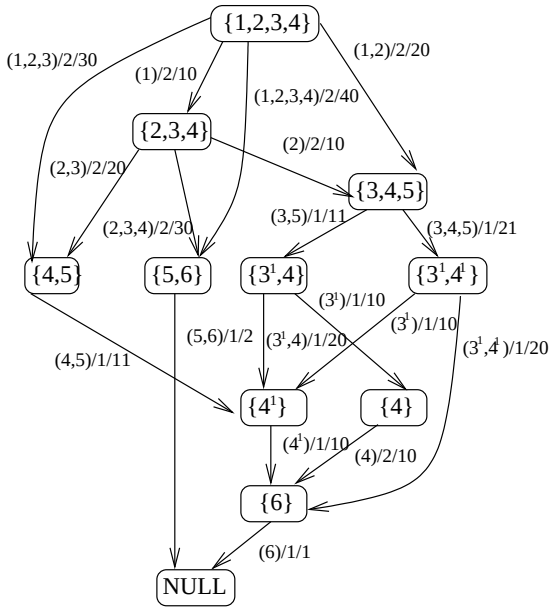

Figure 5: A SDG of the intuitive example (DFG in Figure. 4)

4 Dynamic scheduling based on SDGs

In an energy harvesting system, the available peak power $P(t)$ is a function of time, depending on the rate of harvested energy from the environment, as well as the energy buffered in a battery or capacitor, if there is one. In this paper, dynamic scheduling is considered on time slots t_s, where $P(t_s)$ is assumed to be constant. In addition, we assume that the interval of t_s is a multiple of the path latencies in the SDG of a system.

Given the SDG of a system, and the power constraints $P(t)$, the task of dynamic scheduling for the system can be described as the following time-slot oriented run-time optimisation problem:

Find a schedule corresponding to a path s of the SDG during a time slot t_s, such that the power consumption of each step in the path, $p(e)$ where $e \in s$, is bounded from above by $P(t_s)$, and the latency of s is minimal.

4.1 An optimal dynamic scheduling method based on truncated SDG

A dynamic scheduling method for deriving an optimal schedule of a time slot is described in Algorithm 3. This method first prunes all the edges in a SDG with $p(e) > P(t_s)$, and then find the path in the pruned SDG with the shortest latency, which is the well established shortest-path problem. The schedule corresponding to the shortest path is then adopted as the run-time schedule for the time slot. This method has a complexity of $O(|E| + |E'| + |V'|)$, where E' and V' are the edge and vertex sets of the pruned SDG, respectively.

Example 2 *For the intuitive example 1, application of algorithm 3 to a time-slot with* $P(t_s) = 22$ *units will have three edges pruned. The pruned edges are* $(1, 2, 3)/2/30$*,* $(2, 3, 4)/2/30$*, and* $(1, 2, 3, 4)/2/40$*. As a result of the pruning, no more than* 3 *multiplications can be scheduled in the same step, and the maximum concurrency degree is restricted to* 2*. The path with the shortest latency in the pruned graph*

Algorithm 3 run-time heuristic scheduling with SDG

1: **Inputs**: A $SDG = (V, E, F)$ and $P(t_s)$
2: **Outputs**: An optimal schedule for t_s
3: **prune** the edges e in SDG with $p(e) > P(t_s)$
4: **find** the path s in the pruned SDG with the shortest latency, i.e., *min*($\sum_{i=0}^{n} \lambda(e_i) : e_i \in s$)
5: **form** the schedule according to s

Table 1: Optimal schedules of the intuitive example under different $p(t_s)$

$p(t_s)$	D	optimal schedules	L	P_{ave}
≥ 40	4	$< (1,2,3,4),(5,6) >$	3	27.3
$[30, 40)$	3	$< (1,2,3),(4,5),(4^1),(6) >$, *or* $< (1,2),(3,4,5),(3^1,4^1),(6) >$	5	16.4
$[21, 30)$	2	$< (1,2),(3,4,5),(3^1,4^1),(6) >$	5	16.4
20	2	$< (1,2),(3,5),(3^1,4),(4^1),(6) >$	6	13.7
$[10, 20)$	1	$< (1),(2),(3,5),(3^1),(4),(6) >$	9	9.1
$[0, 10)$	0	N/A	N/A	N/A

is then $< (1,2)/2/20, (3,4,5)/1/21, (3^1,4^1)/1/20, (6)/1/1 >$. *When this path is chosen as the optimal run-time schedule, it has a latency of* 5 *and average power consumption of* 16.4.

Table 1 summarises the optimal schedules including their latency (L) and average power(P_{ave}) for the example, considering different ranges of $P(t_s)$. Also listed in the table is the maximum concurrency degree (D) in terms of the number of multiplications that can be scheduled in the same clock cycle, given the power budget. Schedules listed in the table only contain the operations ID in each schedule step, for the sake of simplicity.

Note that first from this table, two different levels of $P(t_s)$, e.g., within power ranges of $[30, 40)$ and $[21, 30)$, can lead to optimal schedules with the same latency, though the former one allows one more candidate schedule. Whether this is true in general depends on the power of the schedule steps in a SDG under consideration. Second, the same maximum concurrency degree (e.g., 2) can incur schedules with different latency (e.g., for power ranges of $[21, 30)$ and 20). This is the result of defining the concurrency degree purely based on one type of computation resource (i.e., multiplication) and ignoring the power contribution of the other (i.e., addition).

5 Experimental results

To examine the effects of dynamic scheduling on modulating the average and peak power consumption of a circuit corresponding to power constraints from harvesters, a 4-tap 64-bit FIR filter (basic operations include multiplication and addition) has been implemented and mapped to a Xilinx Virtex-6 FPGA (xc6vsx315t-3ff1156) for timing and power analysis, using Xilinx tools SystemGenerator and Xpower, respectively.

Different schedules for the Filter example have been tested, varying from the most concurrent case of running to the most sequential one. The clock rate for all schedules is 76.9 Mhz. In the most concurrent case, all the 20 multiplications are executed in the same schedule step, and a minimum latency of 7 cycles is required to finish the functional loop. In the most sequential case, only one multiplication can be scheduled in a step, and the latency is up to 100 cycles (with a further 2 idle cycles padded between multiplication operations). Other schedules are selected by trading off the concurrency degree in terms of multiplications scheduled in the same step and the total latency.

Figure 6 plots the average and peak power consumption corresponding to the different schedules of the

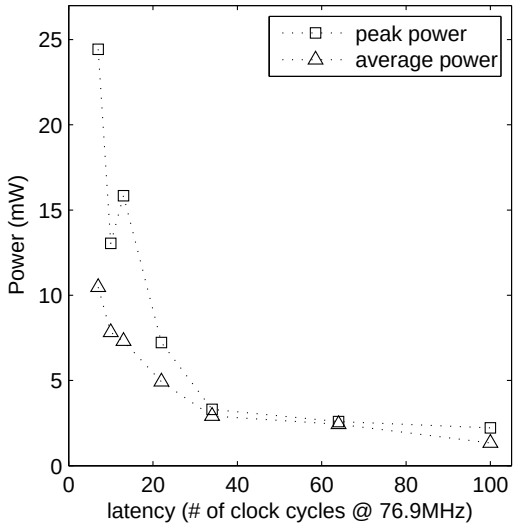

Figure 6: Power-delay curve of the FIR filter using dynamic scheduling

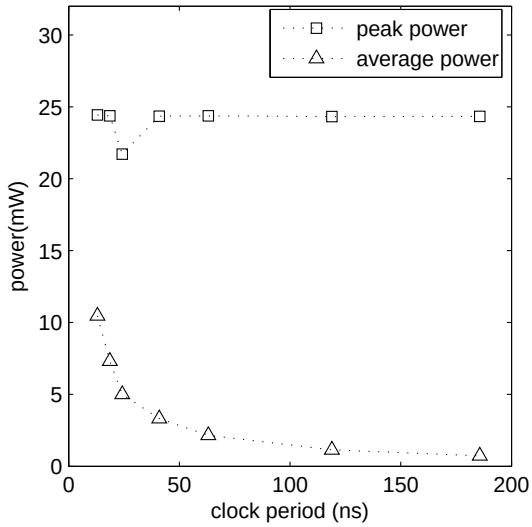

Figure 7: Power-delay curve of the FIR filter using duty cycling

FIR filter. For the estimation of peak power consumption, we first amortise the circuit energy consumption into the multipliers to calculate the average power per multiplier, using formula (1).

$$P_{mult} = \frac{P_{ave} \times L}{M \times D_{mult}} \tag{1}$$

where P_{ave} is the power consumption of the filter circuit corresponding to a schedule with latency L. M and D_{mult} are the number of multiplications in the filtering task, and the latency of a multiplication, respectively.

The peak power consumption of the circuit is estimated using formula (2),

$$P_{peak} = P_{mult} \times D, \tag{2}$$

where D is the maximum concurrency degree of a schedule. Note that the peak power presented here is an estimation based on the equations 1 and 2. These values are average peak power rather than the peak power. We assume that the multiplication operation produce the peak power consumption in the type of resource-dominated circuit such as an FIR filter.

From Figure 6, the average power consumption of the circuit decreases monotonically with the reduction of the concurrency degrees, i.e., with the increase of the latencies of the schedules. The trend of the peak power reduction is similar, capping the average power at each schedule decision point.

To compare with the dynamic scheduling approach, duty cycle adjustment of the FIR circuit is also implemented based on the most concurrent case of running. Figure 7 shows the peak and average power for duty cycling, the points are chosen to have clock period adjusted to meet the same latencies of the schedules in Figure 6, in a one-to-one correspondence following the order on x-axis. From the results of the filter example, duty cycle adjustment can reduce more average power consumption compared with dynamic scheduling when the latency is same, but has little impact on the peak power consumption, which remains constant as the concurrency degree remains the same regardless of the clock rates.

6 Conclusion and Future work

This paper proposes a design methodology and flow for energy harvesting circuit. With this flow, the peak power consumption of a system can be modulated by adjusting its concurrency degree using dynamic scheduling, so that the power constraints from the harvesters can be satisfied. This paper discusses the concept of scheduling decision graph, which is novel to the best of the authors' knowledge. Algorithms have been proposed to derive a decision graph from a system's data flow behaviour. In using the decision graph for dynamic scheduling purpose, an algorithm has been reported which can find the run-time optimal schedule of a system according to the peak power constraints. Finally, the effects of dynamic scheduling on the power-delay characteristics have been examined using a FIR filter circuit, and compared with those by duty cycling. In future research, the authors plan to implement the software/hardware framework using this design flow for energy harvesting systems, and explore the different trade-offs in generating the decision graphs and choosing the design granularities for self-tuning. These explorations will enable us to find the optimal design points when considering the overheads of the controlling schemes themselves.

References

[1] Rajeevan Amirtharajah, Jamie Collier, Jeff Siebert, Bicki Zhou, and Anantha Chandrakasan. Dsps for energy harvesting sensors: Applications and architectures. *IEEE Pervasive Computing*, 4:72–79, 2005.

[2] S. P. Beeby, M. J. Tudor, and N. M. White. Energy harvesting vibration sources for microsystems applications. 17(12):175–195, 2006.

[3] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw Hill, 1994.

[4] Stephan Henzler. *Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies (Springer Series in Advanced Microelectronics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] J.M.Rabaey. *Digital Integrated Circuits*. Prentice Hall, 1996.

[6] Aman Kansal, Jason Hsu, Sadaf Zahedi, and Mani B. Srivastava. Power management in energy harvesting sensor networks. *ACM Trans. Embed. Comput. Syst.*, 6(4):32, 2007.

[7] Michael Keating, David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.

[8] PD Mitcheson, EM Yeatman, GK Rao, AS Holmes, and TC Green. Energy harvesting from human and machine motion for wireless electronic devices. 96(9):1457–1486, 2008.

[9] A Erturk N Kong, D S Ha and D J Inman. Resistive impedance matching circuit for piezoelectric energy harvesting. Jan 2010.

[10] Ani Nahapetian, Paolo Lombardo, Andrea Acquaviva, Luca Benini, and Majid Sarrafzadeh. Dynamic reconfiguration in sensor networks with regenerative energy sources. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 1054–1059, San Jose, CA, USA, 2007. EDA Consortium.

[11] D. Nestor and D.E. Thomas. Behavioural synthesis with interfaces. In *Proceedings of the International Conference on Computer Aided Design*, pages 112–115, 1986.

[12] Joseph A. Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4:18–27, 2005.

[13] P. G. Paulin and J. P. Knight. Force-directed scheduling in automatic data path synthesis. In *DAC '87: Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 195–202, New York, NY, USA, 1987. ACM.

[14] Y K. Ramadass and A P. Chandrakasan. An efficient piezoelectric energy-harvesting interface circuit using a bias-flip rectifier and shared inductor. pages 296–299.

[15] Hui Shao, Chi-Ying Tsui, and Wing-Hung Ki. An inductor-less mppt design for light energy harvesting systems. In *ASP-DAC '09: Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 101–102, Piscataway, NJ, USA, 2009. IEEE Press.