

Performance Modeling of Reconfigurable SoC Architectures and Energy-Efficient Mapping of a Class of Applications*

Jingzhao Ou, Seonil Choi, and Viktor K. Prasanna
EE - Systems, University of Southern California
Los Angeles, California, 90089-2560 USA
{ouj, seonilch, prasanna}@usc.edu

Abstract

Reconfigurable System-on-Chip (RSoC) devices are being used to implement many battery operated systems, where energy efficiency is a major concern. RSoCs incorporate many different components, such as processor core, reconfigurable logic, memory, etc. Various power management techniques can be applied to these components. Tasks within an application can be mapped onto different components for execution. The communication and reconfiguration costs incurred under different mappings significantly impact the overall system energy dissipation. In order to achieve energy-efficient designs on RSoCs, we develop (a) a performance model to abstract a general class of RSoC architectures for application development, (b) a mathematical formulation of the energy-efficient mapping problem for a class of applications, and (c) a dynamic programming algorithm that minimizes the system energy dissipation. We illustrate our approach by mapping two beamforming applications onto Xilinx Virtex-II Pro. For these two applications, our approach leads to an average 52% energy reduction over a greedy algorithm.

1. Introduction

Reconfigurable SoCs (RSoCs), which integrate processor core, reconfigurable logic, memory, etc., are becoming popular. This is because of a strong trend toward programmable design solutions over application specific hardware and a recent trend in integrating configurable logic, e.g., FPGA, and programmable processors, offering the “best of both worlds” on a single chip. One example of these RSoCs is the Triscend A7 CSoC device [14], which integrates a 32-bit ARM7TDMI processor core with pro-

grammable logic, a robust memory subsystem, and a high-performance dedicated internal bus on a single chip. Another example is the Xilinx Virtex-II Pro [16], which integrates low-power IBM PowerPC 405LP processor(s), high density FPGA and on-chip memory.

In recent years, power management has become increasingly important in various computation and communication systems. It is especially critical in battery operated embedded and wireless systems. RSoC architectures offer high efficiency with respect to time and energy performance and have been shown to achieve at least one order of magnitude in power reduction and increase in performance compared to traditional processors in various application areas. Thus, they are being used to implement many of these systems [1]. One important application of RSoCs is software defined radio (SDR) [7]. In SDR, dissimilar and complex wireless standards (e.g. GSM, IS-95) are processed in a single adaptive base station, where a large amount of data from the mobile terminals presents high computational requirement. State-of-the-art RISC processors and DSPs are unable to meet the signal processing requirement of the base stations. Minimizing the power consumption has also become an issue for the base stations due to the high computation requirement that dissipates a lot of energy as well as the inaccessible and distributed locations of the base stations. RSoCs stand out as an attractive option for implementing various functions of SDR due to their high performance, low power dissipation and reconfigurability.

Many control knobs for energy-efficient design are available on RSoCs and are discussed in Section 2. In order to better exploit these control knobs, a performance model of the RSoC architectures and algorithms for mapping applications onto these architectures are required. A RSoC model should allow for a systematic description of the available control knobs and enable system-level optimization. The application can be decomposed into several tasks and the tasks are mapped onto different components of the RSoC device for execution. Different mappings of these tasks can cause significant difference in overall energy dissipa-

*This work is supported by the DARPA Power Aware Computing and Communication Program under contract No. F33615-C-00-1633 monitored by Wright Patterson Air Force Base.

tion. For example, moving data between the processor and the reconfigurable logic results in additional communication time and energy dissipation. Also, the energy required for changing the configuration on reconfigurable logic cannot be ignored compared with that required for computation. A simple greedy algorithm that maps each task onto either the processor or reconfigurable logic depending upon which dissipates less amount of energy, does not guarantee the minimum energy dissipation for executing the application.

This paper makes three contributions to the area of application synthesis on RSoC. First, a performance model to represent a general class of RSoC architectures is developed. The model abstracts various capabilities such as dynamic voltage scaling, reconfiguration, communication, as well as various memory implementations and power management, that can be exploited for energy minimization during application designs. Second, based on the RSoC model, a mapping problem is formulated for a class of applications modeled as a linear task graph. Third, a dynamic programming algorithm is proposed to solve the mapping problem. The algorithm can find a mapping that achieves minimum energy dissipation in polynomial time.

We map two beamforming applications onto Xilinx Virtex-II Pro as examples to demonstrate our RSoC model and the effectiveness of our approach based on dynamic programming. Embedded signal processing applications, such as the two beamforming applications considered in the paper, are typically composed of a linear chain of processing tasks. In these applications, the sampled data are processed on-the-fly by an embedded system on an air-borne or sea-borne vehicle, where energy dissipation is a concern. These beamforming algorithms can also be used at the base stations in SDR to better exploit the limited radio spectrum [11]. Xilinx Virtex-II Pro is used for illustration because it has high computation ability and is a good example for the RSoC architectures. Within Virtex-II Pro, the PowerPC 405 core, the reconfigurable logic, and the on-chip memory are tightly coupled with each other through the FPGA routing resource [19].

The organization of the paper is as follows. Section 2 identifies the knobs for energy-efficient designs on RSoC devices. Section 3 discusses related work. Section 4 describes the proposed RSoC model. Section 5 describes the class of *linear pipeline* applications we are targeting and formulates the energy-efficient mapping problem. Section 6 presents our dynamic programming algorithm. Section 7 illustrates the algorithm using two state-of-the-art beamforming applications. The modeling process and the energy dissipation results of mapping the two applications onto Xilinx Virtex-II Pro are also given in this section. We conclude in Section 8.

2. Knobs for Energy-efficient Designs

Various hardware and system level design knobs are available in RSoC architectures to increase the energy efficiency of designs. For embedded processor cores, dynamic voltage scaling and dynamic frequency scaling can be used to lower the power consumption. The processor cores can be put into idle or sleep mode and further reduce their power dissipation when they are not in execution. These features are available in IBM PowerPC 405 processors and Intel XScale processors. For memory, Triscend A7 CSoC devices can change the state of the memory (SDRAM), to be in active, stand-by, disable, or power-down states. Memory banking is another technique for low-power designs, which splits the memory into banks and activates only the banks currently in use.

For FPGA, there are many energy-efficient design knobs at low level and algorithm level. Low-level knobs refer to knobs at the register-transfer or gate level and algorithm-level knobs refer to knobs that can be used at the algorithm development to reduce energy dissipation.

One of the low-level knobs is clock gating, which is employed to disable clocks to large blocks within a design in order to save power when the output of these blocks is not needed. In Virtex-II Pro, clock gating can be realized by using primitives such as BUFGE to dynamically drive a clock tree only when the corresponding block is used [19]. Choosing energy-efficient *bindings* is another knob. A binding is a mapping of a computation to an FPGA fabric. The ability to choose different bindings is due to the existence of various configurations which dissipate different amounts of energy for the same computation. For example, there are three possible bindings for storage elements in Virtex-II Pro, which are registers, sliced based RAM, and embedded Block RAM (BRAM). As for energy efficiency, while registers and sliced based RAM are better for small storage elements, BRAM is better for large storage elements.

Besides the low-level design knobs, it is known that energy performance can be improved significantly by optimizing a design at the algorithm level [10]. One algorithm-level knob is the architecture selection. It plays a large part in determining the amount of interconnect and logic to be used in the design and thus affects the energy dissipation. For example, matrix multiplication can be implemented using a linear array or a 2-D array. A 2-D array would use more interconnects and would possibly dissipate more energy since 50-70% of the total power is dissipated in the interconnects in Xilinx Virtex-II FPGAs [12]. Another knob is the algorithm selection. A given application is mapped onto FPGAs differently by selecting different algorithms. For example, when implementing FFT, a radix-4 based algorithm would significantly reduce the number of complex number multiplications that would otherwise be needed if a radix-2 algorithm is used. More algorithm-level knobs are parallel

processing and pipelining [5].

3. Related Work

A simple model for specifying and optimizing designs that contain run-time reconfigurable elements is developed in [9]. However, it cannot be used to model and describe RSoC architectures and most importantly, it does not address the energy efficiency of designs.

Several algorithms to solve mapping problems in the context of configurable logic, e.g. FPGAs, have been proposed [2], [6]. In [2], a dynamic programming algorithm to find an optimal mapping of loops onto reconfigurable logic with minimum execution time is proposed. Energy performance is not considered in these algorithms. Also, these algorithms cannot be applied directly to solve the mapping problems for RSoC architectures.

In [18], experiments for re-mapping of critical software loops from a microprocessor to configurable logic are carried out and significant energy savings is achieved for a class of applications. However, a systematic technique that finds the optimal re-mapping is missing in their research and is a focus of this paper.

System level tools are becoming available to map applications onto architectures composed of a processor and configurable logic. Xilinx offers System Generator that integrates hardware and software development for Xilinx Virtex-II Pro [16]. In this design environment, the portion of the application to be mapped onto the embedded PowerPC processor is described using C/C++ and is compiled using GNU gcc. The portion of the application to be executed in the configurable logic is described using VHDL/Verilog and is compiled using Xilinx ISE. In Celoxica DK1 tool [3], the user describes a design using Handel-C (C with additional hardware description) for both hardware and software. Then, the compiler for Handel-C synthesizes the hardware and software. While most system level tools use high level languages to express applications and to map them onto processors and configurable hardware, none of the design tools addresses energy-efficient mapping on RSoCs.

4. Performance Modeling of RSoC architectures

A RSoC model is proposed in this section. A model for Virtex-II Pro is used as an example to illustrate the modeling process.

4.1. RSoC Model

A RSoC system model should capture various capabilities that can be exploited for performance optimization during application mapping onto RSoC architectures. Therefore, as is shown in Figure 1, the RSoC model consists of

four components: a processor, a reconfigurable logic (RL) such as FPGA, a memory, and an interconnect. The interconnect connects the processor, the RL and the memory. Taking Triscend A7 CSoC devices [14] as an example, the interconnect between the ARM7 processor and the SDRAM is a local bus while the interconnect between the SDRAM and the configurable system logic is a dedicated data bus and a dedicated address bus. In Xilinx Virtex-II Pro [19], the interconnect between the PPC405 core and the FPGA is implemented using the FPGA routing resource. We abstract all these buses and connections as an interconnect with different communication time and energy costs between different components. To configure Xilinx FPGA, one can use SelectMAP, serial or boundary-scan programming mode to download the configuration data onto the FPGA, which have significant influence on the configuration time and energy costs. We assume that the memory is shared by the processor and the RL in the model.

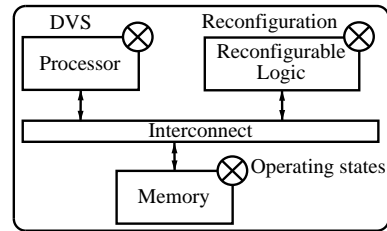


Figure 1. The RSoC model

In the RSoC model, an operating state of the RSoC device, that is, a system state, is determined by the operating states for the processor, the RL, and the memory. Let S denote the set of all possible system states. For system state $s \in S$, $PS(s)$, $RS(s)$ and $MS(s)$ are integers representing the operating states of the processor, the RL and the memory, respectively. An operating state of the processor is defined as the state which the processor is idle or is operating with a specific power consumption. For example, if idle mode and dynamic voltage scaling with $v - 1$ voltage settings are available on the processor, the processor has v operating states, $0 \leq PS(s) \leq v - 1$. $PS(0) = 0$ is the state which the processor is in idle mode; otherwise, $PS(s)$ is the state which the processor is operating in one of the voltage settings. An operating state of the RL is defined as the state which the RL is idle or operating, loaded with a specific configuration. If there are c configurations for the RL, the RL has $2c$ operating states, $0 \leq RS(s) \leq 2c - 1$: (a) for $0 \leq RS(s) \leq c - 1$, $RS(s)$ is the state which the RL is idle, loaded with configuration $RS(s)$; (b) for $c \leq RS(s) \leq 2c - 1$, $RS(s)$ is the state which the RL is operating, loaded with configuration $RS(s) - c$. An operating state of the memory is defined as the state which the memory is operating with a specific power consumption. For example, when memory banking is used to selectively activate the memory banks, each combination of the acti-

vation states of the memory banks represents one operating state of the memory. If the memory has m operating states, $0 \leq MS(s) \leq m - 1$. The operating state of the interconnect is related to the operating states of the other three components. Considering the above, the total number of distinct system states is $2vcm$.

The application is modeled as a collection of tasks with dependencies (See Section 5.1 for details). Suppose that task i' is scheduled to be executed immediately before task i . Task i' is executed in system state $s_{i'}$ and task i is executed in system state s_i . If $s_{i'}$ and s_i are different system states, a system state transition is required. The transition between different system states incurs certain amount of energy. Our model consists of the following performance parameters:

- $\Delta EV_{PS(s_{i'}), PS(s_i)}$: energy dissipation for the processor from state $PS(s_{i'})$ to state $PS(s_i)$
- $\Delta EC_{RS(s_{i'}), RS(s_i)}$: energy dissipation for the RL from state $RS(s_{i'})$ to state $RS(s_i)$
- $\Delta EM_{MS(s_{i'}), MS(s_i)}$: energy dissipation for changing the memory state from $MS(s_{i'})$ to $MS(s_i)$
- $IP(IR)$: processor (RL) power consumption in the idle state
- $PM_{MS(s_i)}$: memory power consumption in state $MS(s_i)$
- $MP_{MS(s_i)}(MR_{MS(s_i)})$: energy dissipation for transferring unit data between the memory and the processor (RL) when memory is in state $MS(s_i)$

The system state transition costs depend on not only the source and destination system states of the transition but also the requirement of the application. Let $\Delta_{i',i,s_{i'},s_i}$ be the energy dissipation for such system state transition. $\Delta_{i',i,s_{i'},s_i}$ can be calculated as

$$\begin{aligned} \Delta_{i',i,s_{i'},s_i} = & \Delta EV_{PS(s_{i'}), PS(s_i)} + \Delta EC_{RS(s_{i'}), RS(s_i)} \\ & + \Delta EM_{MS(s_{i'}), MS(s_i)} + \Delta A_{i'i} \end{aligned} \quad (1)$$

where, $\Delta A_{i'i}$ is the additional data transfer cost incurred by task i' and task i due to a specific mapping.

4.2. A Model for Virtex-II Pro

There are four components to be modeled in Virtex-II Pro. One is the embedded PowerPC core. Since no voltage scaling is available, the processor has only two operating states: on and off. Thus, $v = 2$. We ignore IP since the PowerPC processor does not draw any power if it is not used in a design. $\Delta EV_{0,1}$ and $\Delta EV_{1,0}$ is also ignored since changing the processor states dissipates negligible amount of energy compared with that when it performs computation. Two partial reconfiguration methods, module based and small bit manipulations, are available on

the RL of Virtex-II Pro [15]. The module based partial reconfiguration has relatively high latency and energy dissipation. Therefore, the small bit manipulation method is used. We estimate the reconfiguration costs by multiplying the number of slices used by the configuration files and the average cost for downloading one slice of data onto the FPGA. According to [16], the complete reconfiguration energy for Xilinx XC2VP20 FPGA is calculated as follows. Since the device supply current $ICC_{Int} = 500 \text{ mA}@1.5\text{V}$ during configuration and $ICC_{Int} = 300 \text{ mA}@1.5\text{V}$ during normal operation, the reconfiguration power is $(500-300) \times 1.5 = 300 \text{ mW}$. The complete reconfiguration time using SelectMAP (50 MHz) is 20.54 ms. Thus, the complete reconfiguration energy is $6162 \mu\text{J}$. There are 9280 slices on the device. Together with the slice usage from the post place-and-route report generated by the Xilinx ISE tool [16], we estimate the energy dissipation of reconfiguration as $\Delta EC_{RS(s_{i'}), RS(s_i)} = 6162 \times (\text{number of slices used by the RL in operating state } RS(s_i)) / 9280 \mu\text{J}$. The quiescent power is the static power dissipated by the RL when it is on. This power cannot be optimized at the system level if we do not power on and off the RL. Thus, we ignore it in this paper. IR represents the quiescent power and is set to zero. We also ignore the energy dissipation for enabling/disabling clocks to the design blocks on the RL in the calculation of $\Delta EC_{RS(s_{i'}), RS(s_i)}$ since it is trivial compared with other energy costs. For memory modeling, we use BRAM. It has only one available operating state, $m = 1$ and $MS(s_i) = 0$. There is no $\Delta EM_{MS(s_{i'}), MS(s_i)}$. The BRAM dissipates negligible amount of energy when it holds data without memory access. We ignore this value so that $PM_0 = 0$. Using the power model from [5] and [17], the energy dissipation MP_0 and MR_0 are estimated as 65.4 nJ/Kbyte and 42.9 nJ/Kbyte, respectively.

5. Problem Formulation

A model for a class of applications with linear constraints is described in the section. Then, a mapping problem is formulated based on both the RSoC model and the application model.

5.1. Application Model

The application consists of a set of tasks, $T_0, T_1, T_2, \dots, T_{n-1}$, with linear precedence constraints. T_i must be executed before initiating T_{i+1} , $i = 0, \dots, n-2$. Due to the precedence constraints, only one task is being executed at any time. The execution can happen in the processor, the RL or both. There is data transfer between the tasks. The transfer can occur between the processor (the RL) and the memory, depending on how the tasks are executed.

The application model consists of the following performance parameters:



Figure 2. A linear pipeline of tasks

- $D_{in}^i (D_{out}^i)$: amount of data input (output) to (from) task T_i from (to) memory.
- $EP_{i,s_i} (TP_{i,s_i})$: the processor energy (time) cost for executing task T_i on system state s_i . $EP_{i,s_i} = TP_{i,s_i} = \infty$ if task T_i cannot be executed in system state s_i .
- $ER_{i,s_i} (TR_{i,s_i})$: the RL energy (time) cost for executing task T_i on system state s_i . $ER_{i,s_i} = TR_{i,s_i} = \infty$ if task T_i cannot be executed in system state s_i .

5.2. Problem Definition

We now formulate the problem based on the parameters of the RSoC model and the application model. In this paper, the energy efficiency of a mapping is defined as the energy dissipation for executing an application under this mapping. Thus, a mapping that minimizes the overall energy dissipation for executing an application is also the one that achieves maximum energy efficiency.

For any possible system state s , the processor and the RL cannot be in idle state at the same time. Thus, the total number of possible system states is $|S| = (2v - 1)cm$. Let the system states be numbered from 0 to $(2v - 1)cm - 1$. Then, depending on the sources of energy dissipation, we divide the system states into three categories: (a) for $0 \leq s \leq (v - 1)cm - 1$, $s = (PS(s) - 1)cm + RS(s)m + MS(s)$ denotes the system state that the processor is in state $PS(s)$ ($1 \leq PS(s) \leq v - 1$), the RL is in the idle state loaded with configuration $RS(s)$ and the memory is in state $MS(s)$; (b) for $(v - 1)cm \leq s \leq vcm - 1$, $s = (RS(s) - c)m + MS(s) + (v - 1)cm$ denotes the system state that the processor is in the idle state ($PS(s) = 0$), the RL is operating with configuration $RS(s) - c$ ($c \leq RS(s) \leq 2c - 1$) and the memory is in state $MS(s)$; (c) for $vcm \leq s \leq (2v - 1)cm - 1$, $s = (PS(s) - 1)cm + (RS(s) - c)m + MS(s) + vcm$ denotes the system state that the processor is in state $PS(s)$ ($1 \leq PS(s) \leq v - 1$), the RL is operating in state $RS(s) - c$ ($c \leq RS(s) \leq 2c - 1$) and the memory is in state $MS(s)$.

Note that s_i is the system state in which task T_i is executed and E_{i,s_i} is the energy dissipation for executing T_i in state s_i . E_{i,s_i} is the sum of (1) the energy dissipated by the processor or/and the RL that is/are executing T_i ; (2) if the processor or the RL is in the idle state, the idle energy dissipation of the component; (3) the energy dissipated by the memory during the execution of T_i . These three sources of energy dissipation are calculated as in Table 1.

We calculate the system state transition costs using Equation (1). Since a linear pipeline of tasks is con-

sidered, $i' = i - 1$. The energy dissipation for state transitions between the execution of two consecutive tasks T_{i-1} and T_i , namely, $\Delta_{i-1,i,s_{i-1},s_i}$ is calculated as $\Delta EV_{PS(s_{i-1}),PS(s_i)} + \Delta EC_{RS(s_{i-1}),RS(s_i)} + \Delta EM_{MS(s_{i-1}),MS(s_i)} + D_{out}^i \cdot MP_{MS(s_{i-1})} + D_{in}^i \cdot MP_{MS(s_i)}$.

Let s_0 denote the initial system state. The overall system energy dissipation is given by

$$E_{total} = E_{0,s_0} + \sum_{i=1}^{n-1} (E_{i,s_i} + \Delta_{i-1,i,s_{i-1},s_i}) \quad (2)$$

Now, the problem can be stated as: *find a mapping of tasks to system states, that is, a sequence of s_0, s_1, \dots, s_{n-1} , such that the overall system energy dissipation given by Equation (2) is minimized.*

6. Algorithm for Energy Minimization

We create a trellis according to the RSoC model and the application model. Based on the trellis, the dynamic programming algorithm is presented.

6.1. Creation of the Trellis

The trellis is illustrated in Figure 3 and consists of $n + 2$ steps, ranging from step -1 to step n . Step -1 and step n consist of only one node N_0 each, which represents the initial state and the final state of the system. Step i , $0 \leq i \leq n - 1$, consists of $|S|$ nodes, $N_0, N_1, \dots, N_{|S|-1}$, each of which represents the system state for executing task T_i . The weight of node N_{s_i} in step i is the energy cost E_{i,s_i} for executing task T_i in system state s_i . If task T_i cannot be executed in system state s_i , set $E_{i,s_i} = \infty$. Since node N_0 in step -1 and step n do not contain any tasks, set $E_{-1,0} = E_{n,0} = 0$. There are directed edges (1) from node N_{-1} in step -1 to node N_j in step 0; (2) from node N_j in step $i - 1$ to node N_k in step i for $i = 1, \dots, n - 1$; and (3) from node N_j in step $n - 1$ to node N_0 in step n . The weight of the edge from node $N_{s_{i-1}}$ in step $i - 1$ to node N_{s_i} in step i is the system state transition energy cost $\Delta_{i-1,i,s_{i-1},s_i}$.

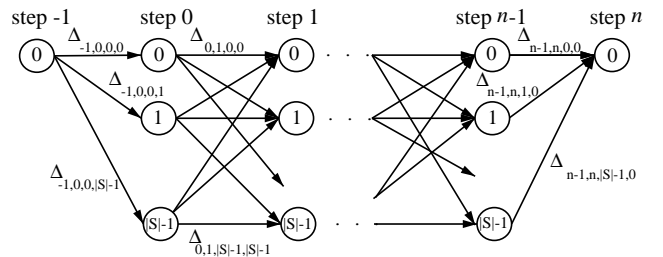


Figure 3. The trellis

Table 1. Energy dissipation E_{i,s_i} for executing task T_i in state s_i

	$0 \leq s_i \leq (v-1)cm - 1$	$(v-1)cm \leq s_i \leq vcm - 1$	$vcm \leq s_i \leq (2v-1)cm - 1$
1	EP_{i,s_i}	ER_{i,s_i}	$EP_{i,s_i} + ER_{i,s_i}$
2	$TP_{i,s_i} \cdot IR$	$TR_{i,s_i} \cdot IP$	0
3	$TP_{i,s_i} \cdot PM_{MS(s_i)}$	$TR_{i,s_i} \cdot PM_{MS(s_i)}$	$\max(TP_{i,s_i}, TR_{i,s_i}) \cdot PM_{MS(s_i)}$

6.2. Dynamic Programming Algorithm

Based on the trellis, our dynamic programming algorithm is described as below. We associate each node with a path cost P_{i,s_i} . Define P_{i,s_i} as the minimum energy cost for executing T_0, T_1, \dots, T_i with T_i executed in node N_{s_i} in step i . Initially, $P_{-1,0} = 0$ for node N_0 in step -1. Then, for each successive step i , $0 \leq i \leq n$, we calculate the path cost for all the nodes in this step. The path cost P_{i,s_i} for node N_{s_i} in step i is calculated as

$$P_{i,s_i} = \min_{0 \leq s_j \leq |S|-1} \{P_{i-1,s_j} + \Delta_{i-1,i,s_j,s_i} + E_{i,s_i}\} \quad (3)$$

Only one path cost is associated with node N_0 in step n . A path that achieves this path cost is defined as a *surviving path*. Using this path, we identify a sequence of s_0, s_1, \dots, s_{n-1} , which specifies how each task is mapped onto the RSoC device. From the above discussion, we have,

Proposition 1 *The task mapping identified by a surviving path achieves minimum energy dissipation over all mappings.*

Since we need to consider $O((2v-1)cm)$ possible paths for each node and there are $O((2v-1)cm \cdot n)$ nodes in the trellis, the time complexity of the algorithm is $O(v^2c^2m^2n)$. However, due to the resource limitation on the RL, such as the number of slices, the number of tasks that a configuration can be used to execute is limited. Thus, the trellis in Figure 3 is usually sparsely connected. The following pre-processing can be applied to further reduce the practical time complexity of the algorithm: (1) nodes with ∞ weight and the edges incident on these nodes are deleted from trellis; (2) the remaining nodes within each step are renumbered. After the two-step pre-processing, we form a reduced trellis and the dynamic programming algorithm is run on the reduced trellis.

7. Illustrative Examples

To demonstrate the energy efficiency of our mapping algorithm, we implement a broadband delay-and-sum beamforming application and an MVDR (minimum-variance distortionless response) beamforming application on Virtex-II Pro. These applications are widely used in embedded signal processing systems and SDR,

7.1. Delay-and-Sum Beamforming

Using the model for Virtex-II Pro discussed in Section 4.2, implementing the delay-and-sum beamforming application is formulated as a mapping problem. This problem is then solved using the proposed dynamic programming algorithm. Note that even though energy and power are closely related to each other, they are different. Energy is the product of average power consumption and latency.

7.1.1 Problem Formulation

The task graph of the broadband delay-and-sum beamforming application [8] is illustrated in Figure 4. A cluster of seven sensors samples data. Each set of the sensor data is processed by an FFT unit and then all the data are fed into the beamforming application. The application calculates twelve beams and is composed of three tasks with linear dependences: calculation of the relative delay for different beams according the positions of the sensors (T_0), computation of the frequency responses (T_1), and calculation of the amplitudes for each output frequencies (T_2). The *data in* and *data out* are performed via the input and output pads on Virtex-II Pro. The input data contains different number of FFT points depending on different frequency resolution requirements. The number of output frequency points is determined by the spectrum of interest. The three tasks can be executed either on the PowerPC processor or on the RL. The amount of data input (D_{in}^i) and output (D_{out}^i) varies with the tasks. For example, when both the numbers of FFT points and the output frequency points are 1024, D_{in}^1 and D_{out}^1 for task T_1 are 14 bytes and 84 Kbytes, respectively.

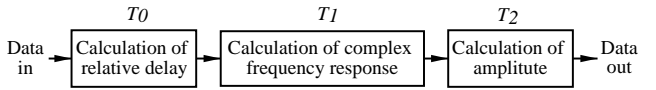


Figure 4. Task graph of the delay-and-sum beamforming

We employ the algorithm-level control knobs discussed in Section 2 to develop various designs on the FPGA. There are many possible designs. For the sake of illustration, we implement two of them for each task. One main difference in the designs is the level of parallelism, which causes different usage of I/O ports and *sine/cosine* look-up tables. For example, one configuration of task T_0 handles two input data per clock cycle and requires more I/O ports than the

other configuration that handles only one input per clock cycle. While the first configuration would dissipate more power and more reconfiguration energy than the second one, it reduces the latency to finish the computation. Similarly, one configuration for task T_2 uses two *sine/cosine* tables and thus can get the result in one clock cycle while the other configuration uses only one *sine/cosine* table and thus requires two clock cycles in order to obtain the output.

Each task is mapped on the RL to obtain TR_{i,s_i} and ER_{i,s_i} values. The designs for the RL are coded using VHDL and are synthesized using XST on Xilinx ISE 4.2i [16]. The VHDL codes for each task are parameterized according to the application requirements such as the number of FFT points, and the architectural control knobs such as precision and binding. The utilization of resource is obtained from the place-and-route report file (.par file) for Virtex-II Pro XC2VP20. Since the XPower from the latest ISE for the time being does not support power estimation of Virtex-II Pro devices, we first estimate the power dissipation for XC2V1500 and then all the power values for Virtex-II are multiplied by 85% except for the SelectI/O power to get the power dissipation for Virtex-II Pro XC2VP20, as is suggested by Xilinx [17]. XC2V1500 is chosen because it is the FPGA in the Virtex-II family whose gate count is closest to that of XC2VP20. To get the power consumption for XC2V1500, our designs are synthesized using XST for XC2V1500 and the place-and-route file (.ncd file) is obtained. Mentor Graphics ModelSim 5.6b is used to simulate the design and generate simulation results (.vcd file). These two files are then provided to XPower to estimate the energy dissipation for XC2V1500. TR_{i,s_i} is calculated based on our designs running at 50 MHz and 16-bit precision. ER_{i,s_i} is calculated based on both TR_{i,s_i} and the power measurement from XPower when task T_i is executed on the FPGA in operating state $RS(s_i)$.

For the IBM PowerPC core on Virtex-II Pro, we implement the application in C, compile it using gcc compiler for PowerPC and generate the bitstream using the tools from Xilinx Virtex-II Pro developers kit. We perform a SWIFT model simulation of the PowerPC, which permits the execution of actual PPC405 code. The data to be computed is stored in the BRAM of Virtex-II Pro. The latencies for executing the C code come directly from the simulation based on ModelSim 5.6b. The energy dissipation is obtained assuming a clock frequency of 300MHz and the analytical expression for processor power dissipation provided by Xilinx [19] as: $0.9 \text{ mW/MHz} \times 300 \text{ MHz} = 270 \text{ mW}$. Then, we estimate the TP_{i,s_i} and EP_{i,s_i} values. Note that the quiescent power is ignored in our experiments as is discussed in Section 4.2.

Therefore, considering both the PowerPC and the FPGA, we have three system states for each of the three tasks on the reduced trellis after the pre-processing discussed in Sec-

tion 6.2. Thus, $0 \leq s_i \leq 2$. Table 2 shows the E_{i,s_i} values for the three tasks when both the numbers of input FFT points and the output frequency points are 1024.

Table 2. Energy dissipation of the tasks in the delay-and-sum beamforming (μJ)

$E_{0,0}$	9.17	$E_{1,0}$	38.47	$E_{2,0}$	2.31
$E_{0,1}$	4.65	$E_{1,1}$	26.31	$E_{2,1}$	2.29
$E_{0,2}$	48.31	$E_{1,2}$	3039.52	$E_{2,2}$	123.24

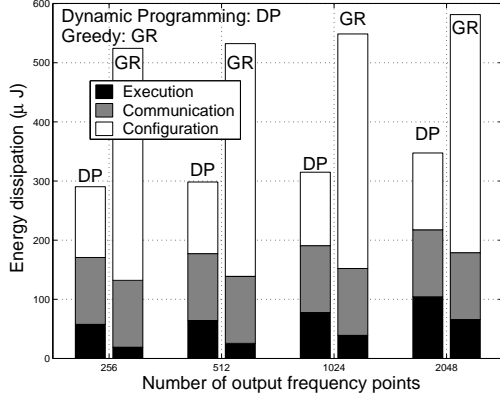
For simple designs, the parameter values can be obtained through low-level simulations. However, for complex designs, it is too time-consuming to obtain these values in this way. This is especially the case for designs on FPGAs. However, using the domain-specific modeling technique proposed in [4], it is possible to have rapid and fairly accurate system-wide energy estimation of data paths configured using FPGAs. The end user can first obtain estimates of the parameter values for various designs and use the proposed RSoC model and dynamic programming algorithm to obtain a few candidate designs. Then, low-level simulation is used to refine the parameter values of these designs. The dynamic programming algorithm is used again to identify the final design.

7.1.2 Energy Minimization

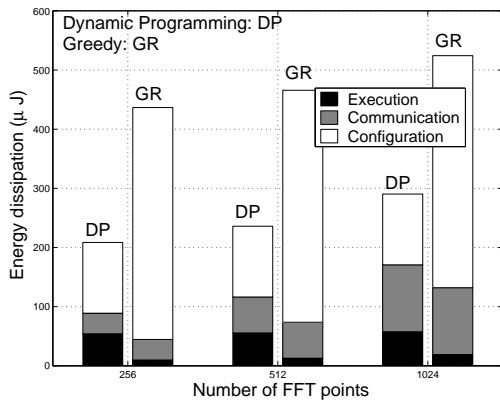
We create a trellis with five steps to represent this beamforming application. After the pre-processing discussed in Section 6.2, step -1 and step 3 contain one node each while step 0, 1 and 2 contain three nodes each on the reduced trellis. By using the values described above, we obtain the weights of all the nodes and the edges on the trellis. Based on this, our approach based on dynamic programming algorithm is used to find the mapping that minimizes the overall energy dissipation.

For the purpose of comparison, we consider a greedy algorithm that always maps each task to the system state in which executing the task dissipates the least amount of energy. The results are shown in Figure 5. For all the considered problem sizes, energy reduction from 41% to 54% can be achieved by our dynamic programming algorithm over the greedy algorithm.

Considering the case where both the number of FFT points of the input data and the number of output frequency points are 2048, the greedy algorithm maps task T_0 on the RL. However, the dynamic programming algorithm maps this task on the processor and a 54% reduction of overall energy dissipation is achieved by doing so. The reason for the energy reduction is analyzed as follows. Task T_0 is executed efficiently on the RL for both the configuration files employed (4.15 to 9.17 μJ). But the configuration costs for these two files are high (272.49 to 343.03 μJ) since task



(a) Energy dissipation when the inputs are after 2048-point FFT processing



(b) Energy dissipation with 256 output frequency points

Figure 5. Simulation results for the broadband delay-and-sum beamforming application

T_0 needs *sine/cosine* functions. The Xilinx FPGA provides the CORE Generator lookup table [19] to implement the *sine/cosine* functions. For 16-bit input and 16-bit output *sine/cosine* lookup tables, the single output design (*sine* or *cosine*) needs 50 slices and the double output (both *sine* and *cosine*) design needs 99 slices. Two and three *sine/cosine* look-up tables are used in the two designs employed for T_0 and this increases the reconfiguration costs. The amount of computation performed by task T_0 is relatively small in this case and the configuration energy cost can raise the overall energy dissipation significantly. Therefore, executing the task on the processor dissipates less amount of energy than on the RL.

7.2. MVDR Beamforming

Using a similar approach as in Section 7.1, we formulate the implementation of the MVDR beamforming application as a mapping problem and solve this problem using the dynamic programming algorithm.

7.2.1 Problem Formulation

We implement the fast algorithm in [8] to compute the MVDR spectrum. This fast algorithm eliminates much computation that is required by the direct calculation of the spectrum. The task graph is illustrated in Figure 6, which can also be decomposed into three tasks with linear constraints: Levinson Durbin recursion to calculate the coefficients of a prediction-error filter (T_0), correlation of the predictor coefficients (T_1), and the MVDR spectrum computation using FFT algorithm (T_2).

We consider both the low-level and algorithm-level control knobs discussed in Section 2 and develop various designs for the tasks which are listed in Table 3. Different levels of parallelism are employed in designs for task T_0 and T_1 . Task T_2 uses FFT to calculate the MVDR spectrum. We employ the various FFT designs discussed in [5] which are based on the radix-4 algorithm as well as the design from Xilinx CORE Generator. Clock gating and different memory bindings are used in these FFT designs to improve the energy efficiency. Two different bindings are used, with one using sliced based RAM and one using BRAM to store the intermediate values. Using the approach described in Section 7.1, we develop parameterized VHDL code. All the designs for the RL and the processor core are mapped on the corresponding components. Parameter values for the RSoC model and the application model are obtained through low-level simulation. For the RL, the designs run at 50 MHz and the data precision is 10 bits. Table 4 shows the E_{i,s_i} values when $M = 8$ and the number of FFT points is 16 after the pre-processing discussed in Section 6.2.

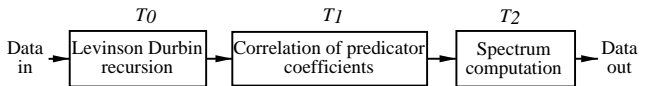


Figure 6. Task graph of MVDR beamforming

Let M denote the number of antenna elements. For task T_0 and task T_1 , we need to calculate complex multiply-and-accumulate (MAC) for problem sizes from 1 to M . Typically, an MVDR algorithm with $M = 8$ is used in software defined radio while MVDR algorithms with $M = 32$ and $M = 64$ are used in embedded sonar systems. As far as energy-efficient design is concerned, there are trade-offs in selecting the number of inputs to the complex MAC structure. Architectures of the complex MAC with 2, 4, and 8 inputs are shown in Figure 7. For a fixed M , using a complex MAC architecture that can handle more inputs at the same time would reduce the execution latency. However, it uses more FPGA interconnects and slices and thus dissipate more power. Also note that due to the limitations on the available FPGA resource, it is not possible to have a complex MAC architecture that can handle all the input data at the same time. For example, there are not enough slices on

XC2VP20 to implement the complex MAC structure with 64 inputs. Thus, we need to use the complex MAC structure repeatedly to perform the calculation. The energy dissipation when using the MAC with different input sizes for task T_1 is analyzed in Figure 8. While the MAC with input size of 4 is most energy-efficient for task T_1 when $M = 8$, the MAC with input size of 2 is most energy-efficient when $M = 64$. Also, the numbers of slices required for a complex MAC that can handle 2, 4, and 8 inputs are 100, 164, and 378, respectively, which have great impact on the configuration costs.

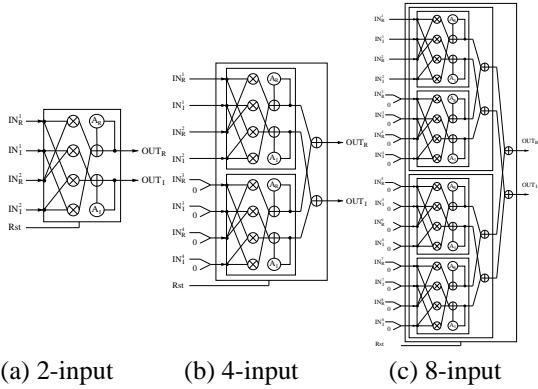


Figure 7. MAC architectures with different input sizes

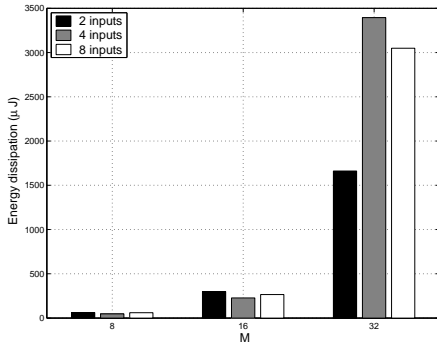


Figure 8. Energy dissipation for task T_1 using different MAC architectures

7.2.2 Energy Minimization

A trellis with five steps is created to represent the MVDR application. After applying the pre-processing technique discussed in Section 6.2, step -1 and step 3 contain one node each, step 0 and 1 contain four nodes each, and step 2 contains seven nodes on the reduced trellis. Using the values described in the previous section, we obtain the weights of all the nodes and the edges on the reduced trellis. Our proposed dynamic programming algorithm is then used to find the mapping that minimizes the overall energy dissipation.

Table 3. Different designs for tasks in the MVDR beamforming application

T_0 and T_1		T_2			
Design	No. of inputs to the MAC	Design	V_p	H_p	Binding
		1	1	2	SRAM
1	2	2	1	3	SRAM
2	4	3	1	3	BRAM
3	8	4	1	4	BRAM
		5	1	5	BRAM
		6	Xilinx design		

Table 4. Energy dissipation for the tasks in the MVDR beamforming application (μJ)

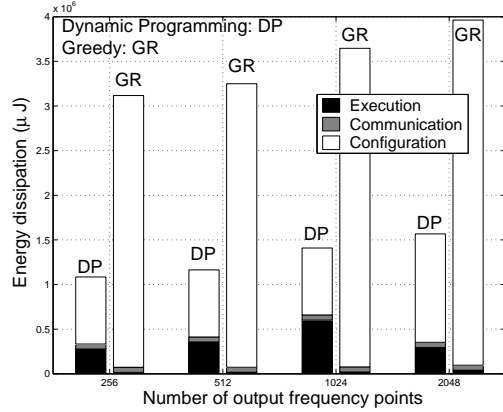
$E_{0,0}$	145.2	$E_{1,0}$	61.0	$E_{2,0}$	67.2
$E_{0,1}$	161.9	$E_{1,1}$	47.3	$E_{2,1}$	432.3
$E_{0,2}$	274.4	$E_{1,2}$	59.8	$E_{2,2}$	363.2
$E_{0,3}$	24590.2	$E_{1,3}$	5397.8	$E_{2,3}$	2223.1
				$E_{2,4}$	12687.4
				$E_{2,5}$	167.7
				$E_{2,6}$	16038.0

The results are shown in Figure 9. For all the considered problem sizes, energy reduction from 52% to 78% are achieved by our dynamic programming algorithm over the greedy algorithm discussed in Section 7.1.2.

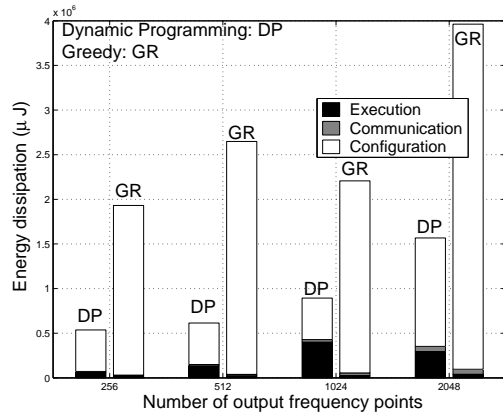
Task T_0 and task T_1 are always mapped to the RL. However, the dynamic programming algorithm maps it to the design using 2-input complex MAC while the greedy algorithm maps them to the designs based on 4-input complex MAC in cases such as when $M = 16$. Designs based on the 2-input complex MAC is not the ones that always dissipate the least amount of execution energy. However, the designs based on the MAC with 2 inputs occupy less amount of area than those based on the MAC with 4 and 8 inputs. Configuration energy reduction ranging from 34% to 66% is observed during our simulation. For task T_2 , the dynamic programming algorithm always maps it on the PowerPC processor core while the greedy algorithm always maps it on the FPGA. Even though the parameterized FFT designs in [5] minimize the execution energy dissipation through the employment of parallelism, radix and choices of storage types, such energy minimization is achieved by using more area on the FPGA, which increases the configuration costs.

8. Conclusion

We have proposed a model for general RSoC architectures and a formulation of the energy-efficient mapping problem based on the model. We have also proposed a dynamic programming algorithm to solve the mapping problem for a class of applications with linear precedences. The



(a) Energy dissipation when $M = 64$



(b) Energy dissipation when the number of inputs to FFT is set at 256

Figure 9. Simulation results for the MVDR application

RSoC model is used to enhance the resource model and the dynamic programming algorithm is used to generate performance bounds for the design space exploration in [13].

While our work focuses on energy minimization, with minor modifications, the performance model does capture throughput and area, and the mapping algorithm can be used to minimize the latency for mapping the class of applications onto RSoCs. We are implementing the designs using a Virtex-II Pro development board to verify the results obtained in this paper.

Acknowledgments

The authors wish to thank Sumit Mohanty, Yang Yu, and Gokul Govindu for helpful discussions.

References

[1] J. Becker, "Configurable Systems-on-Chip: Challenges and Perspectives for Industry and Universities," *ERSA*, 2002.

[2] K. Bondalapati and V. K. Prasanna, "Mapping Loops onto Reconfigurable Architectures," *Field Programmable Logic and Applications*, 1998.

[3] Celoxica, Inc., www.celoxica.com.

[4] S. Choi, J.-W. Jang, S. Mohanty, V. K. Prasanna, "Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures," *ERSA*, 2002.

[5] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, "Energy Efficient Signal Processing Using FPGAs," *FPGA*, 2003.

[6] D. Deshpande, A. K. Somani, and A. Tyagi, "Configuration Scheduling Schemes for Striped FPGA," *FPGA*, 1999.

[7] C. Dick, "The Platform FPGA: Enabling the Software Radio," *Software Defined Radio Technical Conference and Product Exposition (SDR)*, November 2002.

[8] S. Haykin, "Adaptive Filter Theory," *Prentice Hall*, 3rd Edition, 1991.

[9] W. Luk, N. Shirazi, and P. Y. K. Cheung, "Modelling and Optimising Run-Time Reconfigurable Systems," *FCCM* 1996.

[10] A. Raghunathan, N. K. Jha, and S. Dey, "High-level Power Analysis and Optimization," *Kluwer Academic Publishers*, 1998.

[11] J. Razavilar, F. Rashid-Farrokhi, and K. J. Ray Liu, "Software Radio Architecture with Smart Antennas: A Tutorial on Algorithms and Complexity," *IEEE Journal on Selected Area in Commu. (JSAC)*, Vol 17, No. 4, April 1999.

[12] L. Shang, A. Kaviani, and K. Bathala, "Dynamic Power Consumption in Virtex-II FPGA family," *FPGA*, 2002.

[13] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation," *LCTES* 2002.

[14] Triscend, Inc., <http://www.triscend.com>.

[15] "Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations," available online at <http://www.xilinx.com>.

[16] Xilinx Corporation, Inc., www.xilinx.com.

[17] Xilinx Spreadsheet Power Tools, available online at http://www.xilinx.com/ise/power_tools/spreadsheet_pt.htm.

[18] J. Villarreal, D. Suresh, G. Stitt, F. Vahid, and W. Najjar, "Improving Software Performance with Configurable Logic," *Kluwer Journal on Design Automation of Embedded Systems*, 2002.

[19] "Virtex-II Pro Platform FPGA User Guide," <http://www.xilinx.com/publications/products/v2pro/handbook/>.