Energy-aware optimisation for run-time reconfiguration

Tobias Becker and Wayne Luk
Department of Computing
Imperial College London, UK

Peter Y. K. Cheung
Department of Electrical and Electronic Engineering
Imperial College London, UK

***Abstract*—Run-time reconfiguration has been shown to produce power and energy efficient designs. However, it is important to take into account the energy overhead of the reconfiguration process itself. This paper presents an analytical model that covers the effects of power consumption and configuration speed of the reconfiguration process. Based on this model, a method is introduced that establishes the optimal degree of parallelism for designs supporting partial run-time reconfiguration. Our energy-aware approach is illustrated by optimising designs for software-defined radio: a reconfigurable FIR filter is shown to be up to 49% more energy efficient and up to 87% more area efficient than a non-reconfigurable design.**

I. Introduction

Energy efficiency is a crucial aspect in battery-based mobile applications to maximise the battery life. Energy and power efficiency is also important in systems that are connected to the grid. Lower power leads to less heat, lower complexity of power supplies and cooling systems, and reduced energy and system costs.

Reconfigurable devices such as FPGAs are a promising technology to implement fast evolving applications such as mobile communication. Rapid development of communication standards with ever increasing bandwidth and complexity requires alternatives to traditional approaches relying on ASICs and DSPs. The reconfigurability of FPGAs can also be exploited to improve power consumption. A circuit specialised to a certain condition can be more efficient than a general-purpose design. Reconfiguration can be used to update the circuit when the condition changes. However, the device will also consume power during reconfiguration which can influence the overall efficiency.

In this paper we study how power consumption during reconfiguration influences the overall efficiency of a reconfigurable application in terms of its energy per computation. Our key contributions are:

- A simple device-independent model based on application, device and implementation parameters that allows us to analyse the energy efficiency of a design.
- An analysis of how reconfiguration and the degree of parallelism in the design influence the energy efficiency.
- A method of design space exploration to quickly identify an implementation with maximum energy efficiency.
- A case study demonstrating our approach.

The rest of the paper is organised as follows. Section II presents the background and related research. Section III presents our model, the analysis of energy efficiency depending on reconfiguration and parallelism in the implementation and our strategy for design space exploration. Section IV explains practical aspects of power estimation and measurements. Section V demonstrates how our optimisation can be performed on an example, showing a reconfigurable FIR filter. Section VI concludes the paper.

II. Background

FPGAs allow fast application development and post-deployment upgradeability, but their flexibility comes at the cost of area and power overheads compared to ASICs. An FPGA can be between 17 and 32 times less area efficient and between 7 and 14 times less power efficient than an ASIC manufactured in the same technology [1]. FPGA vendors continue to address power challenges through process technology and architectural improvements. Devices specifically optimised for low-power operation can show significant reductions in active and standby power [2].

The reconfigurability of FPGAs can be exploited to create specialised designs that improve performance, reduce area and reduce power. It has been shown that run-time reconfiguration can improve the functional density of a design [3]. The reconfigurable design becomes more efficient, the more data are processed between reconfigurations. Reconfigurable hardware can be employed in high-performance computing where compute-intensive tasks are offloaded into reconfigurable modules [4]. The reconfiguration overhead has to be carefully balanced against the hardware speed-up in order to archive good overall performance. The performance of a reconfigurable design can also be improved by exploring the degree of parallelism in the implementation [5].

The reconfigurability of FPGAs makes them a particularly compelling architecture to realise software-defined radio applications [6]. There is extensive research on reconfigurable signal processing components: results include a reconfigurable FIR filter with 40% area reduction including a framework for fast run-time generation of new configurations [7] and a reconfigurable matched filter for UMTS with improved configuration speed [8]. In another approach, the reconfiguration time of an FIR filter is improved by reducing the amount of logic that needs to be reconfigured [9].

A power-optimised Viterbi decoder has been developed that is reconfigured depending on the signal-to-noise ratio [10]. However, the power associated with the reconfiguration process is not specifically addressed. The power during the configuration process has been measured in a Xilinx Virtex-E FPGA [11]. Power tends to ramp up during initial device reconfiguration but remains fairly constant during run-time reconfiguration. It has also been shown that clock-aware placement strategies can lead to power reductions [12].

An important aspect for the overall efficiency of a design is how reconfiguration overheads relate to improvements made in a reconfigurable design. We can judge the overall efficiency of a design by the total amount of energy required for a certain computation including reconfiguration. In the following we develop an energy-aware analysis that shows when a reconfigurable application becomes more energy efficient than a standard one. Our approach, inspired by previous work [5], includes the exploration of parallelism for further energy optimisations in the reconfigurable design. This method is orthogonal to other low-power optimisations based on device architecture or process technology.

III. Energy optimisation

There are many applications that can benefit from reconfiguration and there are different opportunities of how reconfigurability can be exploited. We can classify the following scenarios:

1. Pre-deployment. The reconfigurability of the device is used for fast prototyping and development. It is possible to make changes late into the design process.

2. In-field upgrade. A configuration providing new features or bug-fixes is loaded into the device. In this case the device is usually rebooted.

3. Infrequent reconfiguration. The design uses run-time reconfiguration occasionally to adapt to a new processing environment. In this case, the reconfiguration overhead is usually insignificant and may be hidden by the application. A reconfigurable Viterbi decoder that adapts to the signal-to-noise ratio is an example of infrequent reconfiguration [10].

4. Frequent reconfiguration. The design uses run-time reconfiguration frequently as part of the application. The influence of the reconfiguration overhead on performance and energy efficiency is significant and needs to be considered. Reconfigurable high-performance computing is an example involving frequent reconfiguration [4].

The last scenario is the most challenging because reconfiguration overheads can have a significant impact. In the following we analyse how the energy efficiency can be improved in such a scenario. We use several application, implementation and device parameters for our energy optimisation. Application parameters are:

- Number of packets or data items n that are processed between reconfigurations
- Number of processing steps s in the algorithm

A reconfigurable implementation is characterised by the following parameters:

- Area requirement of the implementation A
- Amount of parallelism p in the implementation
- Processing time t_p for one packet or datum
- Reconfiguration time t_r
- Power consumed during processing P_p
- Computation power P_c, a component of P_p
- Power overhead P_o, a component of P_p
- Power consumed during reconfiguration P_r

Finally we use the following device parameters:

- Data throughput of the configuration interface ϕ_{config}
- Configuration size per resource or unit of area Θ

We consider two energy optimisation approaches. In the first, we analyse the energy overhead of reconfiguration and identify when a reconfigurable design becomes more energy efficient than a non-reconfigurable one. In the second approach we analyse how the degree of parallelism influences the energy efficiency.

A. Energy efficiency in reconfigurable designs

The energy required to process n data items in a non-reconfigurable implementation is given by equation 2. Note that even a non-reconfigurable design might require a parameter reload between processing data sets. This can have an influence on its energy consumption.

$$E_{total} = E_p + E_{load} \quad (1)$$

$$= P_p \cdot t_p \cdot n + P_{load} \cdot t_{load} \quad (2)$$

The energy normalised per datum is given in equation 3. For practical data set sizes n, it is often possible to neglect the energy influence of the parameter reload.

$$E_{total,n} = P_p \cdot t_p + \frac{P_{load} \cdot t_{load}}{n} \approx P_p \cdot t_p \quad (3)$$

For a reconfigurable design, the energy to process n data items is given as follows:

$$E_{total} = E_p + E_r \quad (4)$$

$$= P_p \cdot t_p \cdot n + P_r \cdot t_r \quad (5)$$

The reconfiguration time t_r can be calculated based on the area requirement A of a design:

$$t_r = \frac{\Theta \cdot A}{\phi_{config}} \quad (6)$$

Θ is a device specific constant and specifies the number of bytes required to configure a particular device resource. ϕ_{config} is the configuration data rate and depends on the configuration interface and the configuration controller. We can normalise equation 5 to n:

$$E_{total,n} = P_p \cdot t_p + P_r \cdot \frac{\Theta \cdot A}{\phi_{config} \cdot n} \quad (7)$$

A reconfigurable design often has the benefit of consuming less processing power P_p and having higher performance, i.e. lower t_p. A reconfigurable design is also smaller than a non-reconfigurable version. In order to be more energy efficient than the non-reconfigurable design, the savings in E_p must be larger than the overhead in reconfiguration energy E_r. To keep E_r low, it is important to reconfigure the smallest possible area A with the highest available configuration speed ϕ_{config}. Configuring between larger datasets n also increases the efficiency.

B. Exploring parallelism

We now consider how parallelism in the implementation of a run-time reconfigurable design can influence its energy efficiency. Many algorithms can be scaled between a small and slow serial implementation and a large and fast parallel implementation. Previous work demonstrates that the degree of parallelism can be used to optimise the performance of a reconfigurable design [5]. We now perform a similar analysis for energy.

The upper three diagrams in figure 1 illustrate three different temporal and spatial mappings of an algorithm with four steps, and the implication on processing time, reconfiguration time and area. One processing element is characterised by $t_{p,e}$, the basic processing time per processing element, and $t_{r,e}$, the basic reconfiguration time per processing element. The processing time t_p for one data item in an algorithm with s steps and an implementation with parallelism p is:

$$t_p = \frac{t_{p,e} \cdot s}{p} \tag{8}$$

Parallelism speeds up processing of data but slows down reconfiguration. This is because a parallel implementation will require more area than a sequential one, and reconfiguration time is directly proportional to area. The reconfiguration time t_r of an implementation with p processing elements is therefore:

$$t_r = t_{r,e} \cdot p \tag{9}$$

We now analyse the power and energy consumption in a reconfigurable design. Instead of static and dynamic power, we consider power components that scale with p and components that remain constant. We consider processing power P_p to be a combination of computation power P_c and power overhead P_o. We assume that each processing element incurs a certain computation power $P_{c,e}$. The computation power P_c is hence directly proportional to p:

$$P_c = P_{c,e} \cdot p \tag{10}$$

The computation energy E_c for processing n data items is:

$$E_c = P_{c,e} \cdot p \cdot t_p \cdot n = P_{c,e} \cdot t_{p,e} \cdot s \cdot n \tag{11}$$

The energy associated with computation is independent of p and is constant for an algorithm with given s and n.

During processing, we will also find a power overhead P_o that is not directly associated with computation. This overhead may be static power only. However, there can be further elements to this power overhead e.g. the power consumption of auxiliary circuits such as clock managers. This power component is constant and does not scale with p. The energy overhead E_o encountered during the processing of n data items is inversely proportional to p:

$$E_o = P_o \cdot t_p \cdot n = P_o \cdot \frac{t_{p,e} \cdot s \cdot n}{p} \tag{12}$$

Finally, we have to consider the power and energy consumed during the reconfiguration process. We assume that reconfiguration power P_r is constant. For P_r we do not have to consider the power overhead separately. Reconfiguration energy simply scales with reconfiguration time regardless of the distribution of its components. The reconfiguration energy E_r grows with reconfiguration time t_r, and is therefore proportional to p:

$$E_r = P_r \cdot t_r = P_r \cdot t_{r,e} \cdot p \tag{13}$$

The total energy for processing n data items is therefore:

$$E_{total} = E_p + E_r = E_c + E_o + E_r \tag{14}$$

$$= P_{c,e} \cdot t_{p,e} \cdot s \cdot n + P_o \cdot \frac{t_{p,e} \cdot s \cdot n}{p} + P_r \cdot t_{r,e} \cdot p \tag{15}$$

The total energy normalised to n is:

$$E_{total,n} = P_{c,e} \cdot t_{p,e} \cdot s + P_o \cdot \frac{t_{p,e} \cdot s}{p} + P_r \cdot \frac{t_{r,e} \cdot p}{n} \tag{16}$$

Figure 2 shows the normalised energy per datum over p for the parameters $s = 128$, $n = 10,000$, $t_{p,e}/t_{r,e} = 5 \cdot 10^{-4}$ and $P_o/P_r = 0.5$. A variation of either n or $t_{p,e}/t_{r,e}$ with the factor a leads to different optima. Larger datasets or higher processing time to reconfiguration time ratios (faster reconfiguration) push the optimum towards more parallel implementations. Smaller data sets or slower reconfiguration shift the optimum to a more serial solution.

In order to find the optimal degree of parallelism that minimises the energy per datum, we calculate the partial derivative of equation 16 with respect to p:

$$\frac{\partial E_{total,n}}{\partial p} = -\frac{P_o \cdot t_{p,e} \cdot s}{p^2} + \frac{P_r \cdot t_{r,e}}{n} \tag{17}$$

To find the minimum we set equation 17 to 0 and solve for p:

$$p_{opt} = \sqrt{\frac{P_o \cdot t_{p,e} \cdot s \cdot n}{P_r \cdot t_{r,e}}} \tag{18}$$

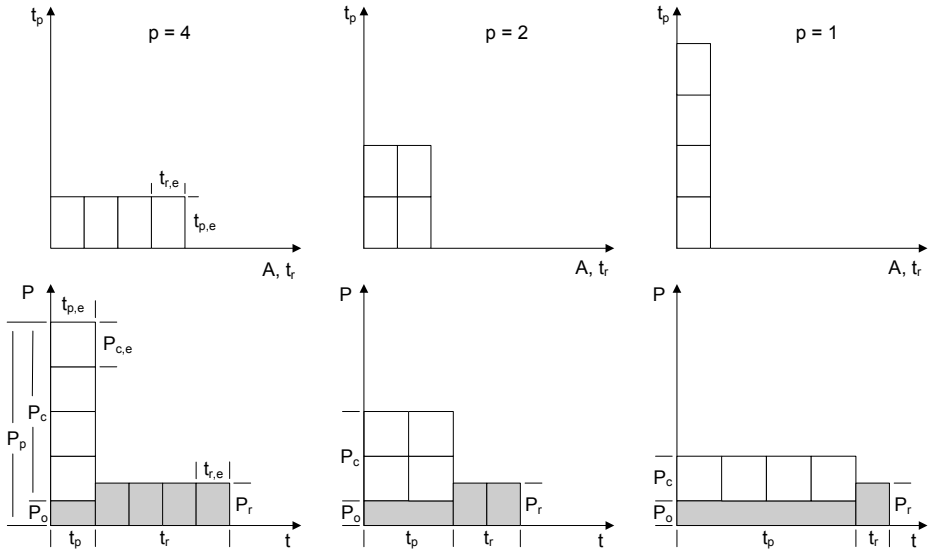

Figure 1. Different temporal and spatial mappings of an algorithm with $s = 4$. The upper diagrams show processing time, reconfiguration time and area. The lower diagrams show the influence of p on power for $n = 1$. We have to minimise the energy represented by the grey rectangles.

The result p_{opt} is usually a real number which is not a feasible value to specify parallelism. One should choose a p value which divides s and is close to p_{opt}. If $p \leq 1$, then a fully serial implementation is the most efficient and for $p \geq s$, a fully parallel implementation is the most efficient.

This optimisation is independent of computation power P_c. It balances the energy associated with the power overhead P_o and the energy associated with reconfiguration power P_r. Graphically this corresponds to reducing the combined area of the grey rectangles in figure 1. The area of the white rectangles is constant.

C. Optimisation steps

In the previous section we describe how a variation in the degree of parallelism of the implementation can be used to optimise a design for energy. This effect may be observed in many designs that require several similar processing steps such as multiply-accumulate operations in digital filters, butterfly operations in FFTs or substitution boxes in encryption. When developing several implementations with varying degrees of parallelism, design parameters may not scale exactly as assumed previously. For example, some designs may incur an area overhead for serial implementations while in other cases, parallelising a design may cause an area overhead. It is possible to perform an optimisation by simply building and measuring designs for all possible variations of p. However, this will require a long time.

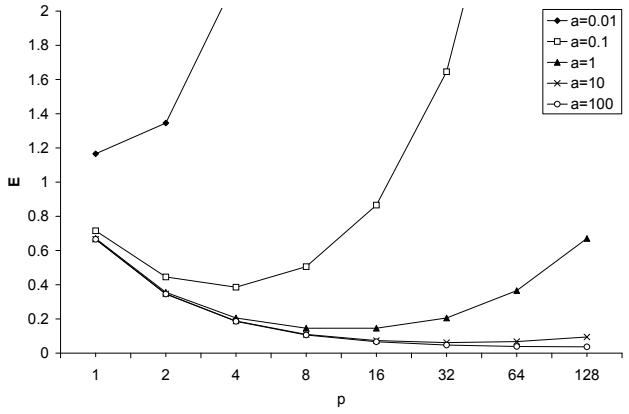

Figure 2. Normalised energy per datum for parameter variations of dataset size n, or the processing to configuration time ratio $t_{p,e}/t_{r,e}$.

Instead, we can use equation 18 to quickly explore the design space based on parameters derived from just one implementation. Even if parameters do scale not exactly as assumed, our exploration will still indicate where the optimum can be found. After determining p_{opt} using equation 18, we implement a new design with a practical p value close to p_{opt}. We then measure its parameters and check the resulting energy efficiency. If the design is less efficient than predicted, p has to be moved closer to the value of the original implementation. If the design is more efficient, then p can be moved further out. Our exploration can be summarised by the following 8 steps:

1) Derive s and n from application.
2) Obtain Φ_{config} and Θ for target device.
3) Implement one design and determine t_p, A, P_r and P_0.
4) Calculate t_r, $t_{p,e}$ and $t_{r,e}$ using equations 6, 8 and 9.
5) Find p_{opt} from equation 18 and determine a practical value for p.
6) Build design for p.
7) Verify if design matches predicted parameters.
8) If necessary refine result by moving p up or down.

If a design scales well, then the above method will find the optimum right away. If parameters vary, a reiteration of steps 6 and 7 may be necessary. This is still considerably less effort than building all possible design variations.

IV. Power measurements

Our optimisations can be performed based on power estimation tools or power measurements. Power estimation tools offer a fast and simple way of determining the power consumption of a design, but may be limited in accuracy. Power estimation tools usually also lack support for reconfiguration power. Power measurements are more accurate but require a target board with power measurement facilities as well as adequate measuring equipment.

Processing power may be calculated with reasonable accuracy using power estimation tools. With power estimation tools it is also straightforward to identify computation power P_c and the power overhead P_o caused by static power and additional non-reconfigurable circuits. For power measurements, P_o can be determined by building a design with all components that are not affected by reconfiguration and measuring its power consumption. This design should include clock managers running at their targeted clock rate. Xilinx power estimation tools currently do not support power estimation for partial reconfiguration. Reconfiguration power P_r has therefore be measured on the board.

FPGA power measurements can be obtained by measuring the current in the device supply rails. However, wiring a multimeter directly into the supply rails may not be a suitable measurement technique. FPGAs are sensitive to core voltage variations. For example, Xilinx Virtex-5 FPGAs have to stay within $50\ mV$ of the specified core supply voltage of $1\ V$. The voltage drop over the internal shunt resistor in a multimeter can easily exceed this limit if higher currents are drawn. Instead it is preferable to insert a precision current sense resistor with a small enough resistance into the supply rail and measure the voltage drop over the resistor.

All our experiments are implemented on the Xilinx ML505 board and power results are based on FPGA core current measurements. The ML505 board does not provide core current measurement facilities as a standard feature. The board was therefore modified by inserting a precision current sense resistor between the voltage regulator and the FPGA core supply rails. We measure the voltage drop across the resistor with a digital storage oscilloscope.

V. Case study

In software-defined radio we can identify many components that can benefit from reconfiguration. Examples are FIR filters, FFTs, correlators, the CORDIC algorithm and error correction such as Viterbi or Turbo codes. These basic blocks are used in different communication standards with varying parameters [13] and may require reconfiguration. These components also offer the opportunity to explore the degree of parallelism in their implementation.

We demonstrate our optimisations on the example of a reconfigurable FIR filter. FIR filters are used in many stages of a radio receiver or transmitter and may require some form of modification. Examples of filter modifications are coefficient reloading to change the filtered frequency band, or template reloading in a matched filter. A flexible filter can be implemented as a non-reconfigurable filter that provides additional circuitry to reload parameters into the design. As an alternative, the filter can be reconfigurable where each instance is specific and optimised to one set of parameters. A filter modification is then carried out through run-time reconfiguration. In the following, we first analyse the energy efficiency of a reconfigurable filter and compare with a non-reconfigurable counterpart. In the second part of our case study, we explore how the degree of parallelism can be used to further improve energy efficiency. We perform our analysis on the example of an 80-tap FIR filter that has to process 10,000 16-bit samples between parameter updates.

Our experiments are conducted on the Xilinx ML505 board which contains a Virtex-5 XC5VLX50T FPGA. Virtex-5 FPGAs contain an internal configuration access port (ICAP) which can provide a maximum configuration throughput of $\phi_{config} = 400MB/s$. ICAP can be used in the HWICAP core [14] in combination with a MicroBlaze softcore processor to control reconfiguration. When using this core, we measure a configuration throughput of only $5MB/s$. However, Claus et.al. recently presented an improved ICAP controller that provides a throughput of $300MB/s$ [15]. In Virtex-5 FPGAs, the configuration size per unit of area is $\Theta_{CLB} = 295.2bytes/CLB$ or $\Theta_{LUT} = 36.9bytes/LUT$ [16].

A. Comparing reconfigurable and non-reconfigurable designs

We implement our 80-tap FIR filter in a reconfigurable and a non-reconfigurable version. The filters are designed in VHDL and synthesised with Xilinx XST 11 synthesis tools using IEEE arithmetic libraries. The non-reconfigurable filter provides a port to reload filter coefficients. Coefficients are loaded into a register chain and a reload requires 80 clock cycles. The reconfigurable filter contains hard-coded coefficients. The synthesis tools use this to create optimised fixed-coefficient multipliers. We implement one reconfigurable version of the design with an embedded configuration controller based on a MicroBlaze processor and the HWICAP core. This system creates an overhead in terms of logic utilisation and power. We also create a second version of the reconfigurable design based on an external configuration controller. No logic and power overhead is added to the design. Even though an external controller will add some general overhead to the system, it could still be a lot more efficient than a MircoBlaze-based processor system. This scenario can be considered the best case in energy efficiency.

	LUTs	f_{max}	$t_p[ns]$	$P_p[W]$	$E_p[\mu J]$	$P_r[W]$	$t_r[\mu s]$	$E_r[\mu J]$	$E_{total}[\mu J]$
reconfigurable, MB, $\phi_{config} = 5MB/s$	12867 [1]	182 [2]	5.49	1.36	74.7	0.44	69755	30692	30767
reconfigurable, MB, $\phi_{config} = 300MB/s$							1162	512	586
reconfigurable, ext, $\phi_{config} = 5MB/s$	9452	182	5.49	1.23	67.9	0.19	69755	12695	12763
reconfigurable, ext, $\phi_{config} = 300MB/s$							1162	211	279
non-reconfigurable, 80 cycle coeff reload	26134	100	10	3.43	343	3.8 [3]	0.8 [3]	3 [3]	346

note 1: total size of design including MicroBlaze, only 9452 LUTs are reconfigured
note 2: f_{max} applies to FIR, not MicroBlaze
note 3: coefficient reload instead of reconfiguration

Table I
COMPARISON OF A NON-RECONFIGURABLE DESIGN WITH RECONFIGURABLE DESIGNS USING AN INTERNAL MICROBLAZE CONFIGURATION CONTROLLER (MB) AND AN EXTERNAL CONFIGURATION CONTROLLER (EXT). ALSO SHOWN ARE FAST AND SLOW CONFIGURATION SPEEDS.

Table I shows the parameters of our three designs. The MicroBlaze-based reconfigurable design requires less than half of the logic resources than the non-reconfigurable version. The MicroBlaze system also requires some BRAM resources which are not shown in this comparison. The reconfigurable design with external reconfiguration requires 64% less logic resources than the non-reconfigurable design. The maximum clock frequency also increases from 100 MHz to 182 MHz. The processing power is reduced by 60% for the MicroBlaze-based design and by 64% for the design with external reconfiguration. Area and power savings and the performance increase can be explained given the fact that more efficient fixed-coefficient multipliers are used in the reconfigurable design. These improvements lead to significant reductions in the processing energy. However, reconfiguration can incur an energy penalty that may offset the improvements in processing energy. Table I shows the total energy to process 10,000 samples in our reconfigurable designs for two different reconfiguration speeds. The slow speed of $5MB/s$ corresponds to what can be achieved with an unoptimised HWICAP core, and the fast speed of $300MB/s$ corresponds to what has been demonstrated in [15]. The results show that slow configuration leads to significant energy penalties that make the reconfigurable designs less efficient than the non-reconfigurable ones. Even the MircoBlaze design with fast reconfiguration is less efficient than the design without reconfiguration. However, the design with external reconfiguration is 19% more energy efficient.

Figure 3 shows the normalised energy efficiency of our designs over a range of data set sizes. For large data sets, all reconfigurable designs approach an energy of $6.8nJ/sample$. This is 5 times more efficient than the non-reconfigurable design with $34.3nJ/sample$. However, slow reconfiguration speeds as well as energy overheads caused by the reconfiguration controller lead to fast deterioration of energy efficiency for smaller data sets. To obtain improvements in energy efficiency through reconfiguration, it is important to develop reconfiguration mechanisms with high speed and low power overhead.

B. Exploring parallelism

We now demonstrate how our design exploration method can be used to further optimise the design. As step 1, we determine the application parameters as $s = 80$ and $n = 10,000$. For Viretx-5 with fast reconfiguration, we obtain $\phi_{config} = 300MB/s$ and $\Theta_{LUT} = 36.9bytes/LUT$ (step 2). We use the reconfigurable design with external configuration as initial implementation. This design is fully parallel, i.e. $p = 80$. The power overhead $P_o = 350mW$ is measured on a configured device with clock managers running but without any FIR logic, and t_p, A and P_r are obtained from table I (step 3). As step 4, we calculate the processing time per element as $t_{p,e} = 5.49ns$ using equation 8, and the reconfiguration time per element as $t_{r,e} = 14.5\mu s$ using equation 6 and 9. Using equation 18 we can calculate $p_{opt} = 24.1$ (step 5). We choose $p = 20$ as a practical value. According to equation 16, we predict the energy for this design to be $17.8nJ/sample$. We now build a design for $p = 20$ and obtain its parameters (step 6). Table II shows circuit size, maximum clock rate, time and energy parameters for designs with p ranging from 80 down to 5. This is for illustration purposes; not all

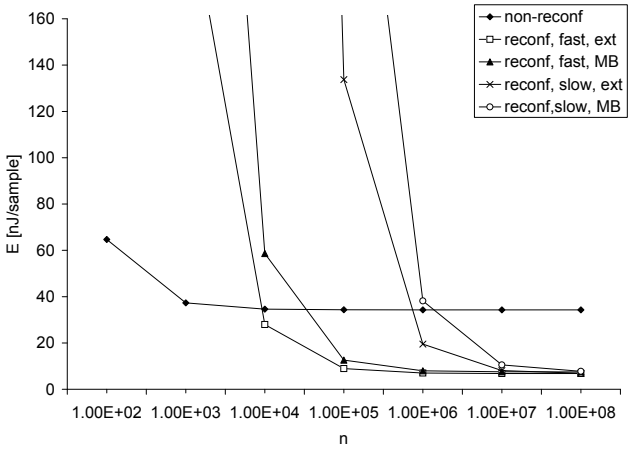

Figure 3. Energy efficiency over dataset size n for MicroBlaze (MB) and external (ext) reconfiguration and with various configuration speeds.

	VHDL design								CoreGen design							
p	LUTs	f_{max}	P_p $[mW]$	$t_{p,e}$ $[ns]$	$t_{r,e}$ $[\mu s]$	$E_{p,n}$ $[nJ]$	$E_{r,n}$ $[nJ]$	$E_{total,n}$ $[nJ]$	LUTs	f_{max}	P_p $[mW]$	$t_{p,e}$ $[ns]$	$t_{r,e}$ $[\mu s]$	$E_{p,n}$ $[nJ]$	$E_{r,n}$ $[nJ]$	$E_{total,n}$ $[nJ]$
80	9452	182	1236	5.49	14.5	6.8	21.1	27.9	12039	300	1928	3.33	18.5	6.4	27.0	33.4
40	6278	145	908	6.89	19.3	12.5	14.0	26.5	6652	300	1276	3.33	20.4	8.5	14.8	23.3
20	5058	120	778	8.33	31.1	25.9	11.3	37.2	3321	300	774	3.33	20.4	10.3	7.4	17.7
10	3129	120	630	8.33	38.4	42.0	7.0	49.0	1674	300	550	3.33	20.4	14.7	3.7	18.4
5	1774	110	398	9.09	43.6	57.9	3.9	61.8	841	300	452	3.33	20.5	24.1	1.9	26.0

Table II
VHDL AND COERGEN IMPLEMENTATIONS OF THE RECONFIGURABLE FIR FILTER FOR VARIOUS p AND A DATASET SIZE OF $n = 10,000$.

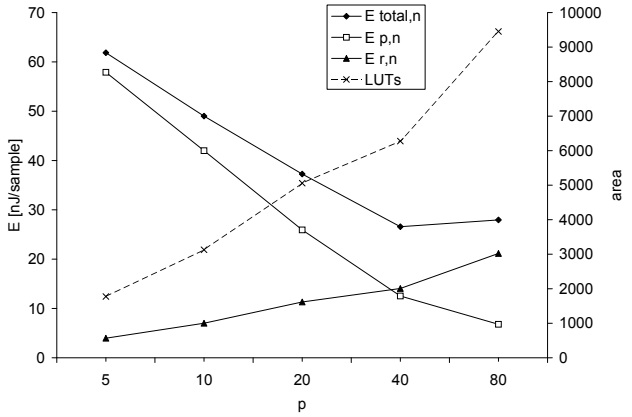

Figure 4. Energy efficiency and area for different levels of parallelism p in VHDL implementation.

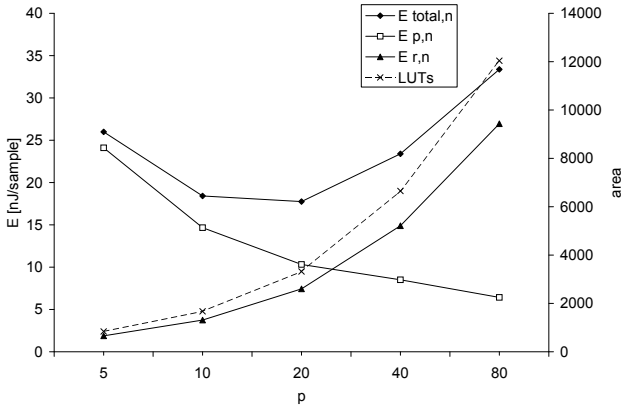

Figure 5. Energy efficiency and area for different levels of parallelism p in CoreGen implementation.

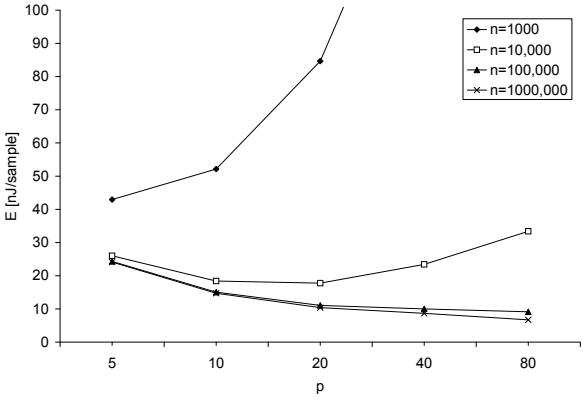

Figure 6. Energy efficiency for different levels of parallelism p and various data set sizes n in CoreGen implementation.

of these designs would be built when using our method. A breakdown of the energy components and circuit size is also illustrated in figure 4. For $p = 20$, we find that $E_{total,n} = 37.2nJ/sample$ (step 7). This is worse than expected and can be explained with the fact that the clock frequency drops and that circuit size does not scale as well as expected. As step 8, we move p back closer to the original value and try $p = 40$. For this value we do find an optimum. The design uses $26.5nJ/sample$ which is 5% less than the fully parallel design and 23% less than the original non-reconfigurable design. The design with $p = 40$ requires 6278 LUTs which is 34% less than the fully parallel design and 76% less than the original non-reconfigurable design.

We now reimplement the same reconfigurable filter with Xilinx CoreGen. Filters are implemented using distributed arithmetic and fixed coefficients. Parallelism cannot be directly specified in CoreGen, but it can be implied by setting clock rates and data rates accordingly. We first build a fully parallel version of the design and perform our optimisation again from step 3. Table II shows that the fully parallel version requires more area and power and is less energy efficient than the comparable VHDL implementation.

The power overhead for this design is measured as $P_o = 356mW$ and we calculate $t_{p,e} = 3.33ns$ and $t_{r,e} = 18.5\mu s$. All other parameters are identical to the previous case. Using equation 18, we calculate $p_{opt} = 16.8$. A practical value for p is 16. However, CoreGen can only efficiently generate designs where parallelism is reduced by powers of 2. We therefore choose the closest practical value $p = 20$. Using equation 16 we calculate the energy for this design to be $16.7nJ/sample$. The real energy according to table II is $17.7nJ/sample$ which is close to what we expect. The design with $p = 20$ is indeed the optimal design and requires 47% less energy than the fully parallel CoreGen design and

49% less energy than the original non-reconfigurable design. The design with $p = 20$ has a size of 3321 LUTs which is 72% less than the fully parallel CoreGen version, and 87% less than the original design.

Table II shows that the CoreGen design scales better than the previous VHDL implementation. The clock rate remains constant and the circuit size scales proportionally to p. This causes the CoreGen design to become more efficient than the VHDL version as p is reduced. A breakdown of the energy efficiency is illustrated in figure 5.

The previous analysis considers a fixed dataset size of $n = 10,000$. Figure 6 shows the efficiency of the CoreGen design when the dataset size varies. For a dataset size of 1000, a fully serial implementation is the best choice for a reconfigurable design. However, in our case this is 15% less energy efficient than the non-reconfigurable design. Larger dataset sizes push the optimum to fully parallel implementations which are all more energy efficient than the non-reconfigurable design.

VI. Conclusion

We analyse the energy efficiency of reconfigurable and non-reconfigurable designs including the overhead caused by reconfiguration. Our analysis is based on a simple analytical model that allows us to evaluate the energy efficiency based on a set of application, device and implementation parameters. Using our model we can perform a fast design-space exploration to identify the most efficient implementation without building and measuring all design variations. We conduct a case study of a reconfigurable FIR filter and show that fast reconfiguration speeds and low-power reconfiguration are essential to making reconfigurable designs beneficial. We also evaluate how changing the degree of parallelism in the implementation can influence the overall energy efficiency. A CoreGen implementation scales exactly as assumed in our model, and our model is also useful for a VDHL implementation that scales less well. A refinement of the results leads us to the optimal solution. Current and future work includes studying the effect of this optimisation on a wide range of applications, as well as automating our energy-aware approach.

Acknowledgements

The support of Xilinx, UK EPSRC and FP7 HiPEAC is gratefully acknowledged.

References

[1] I. Kuon and J. J Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[2] T. Tuan, A. Rahman, S. Das, S. Trimberger, and S. Kao, "A 90-nm low-power FPGA for battery-powered applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 296–300, Feb. 2007.

[3] M. Wirthlin and B. Hutchings, "Improving functional density using run-time circuit reconfiguration," *IEEE Transactions on VLSI Systems*, vol. 6, no. 2, pp. 247–256, 1998.

[4] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Exploiting partial runtime reconfiguration for high-performance reconfigurable computing," *ACM Transactions Reconfigurable Technology and Systems*, vol. 1, no. 4, pp. 1–23, 2009.

[5] T. Becker, W. Luk, and P. Y. K. Cheung, "Parametric design for reconfigurable software-defined radio," in *ARC '09: Proceedings of the 5th international workshop on Reconfigurable Computing*. Springer, 2009, pp. 15–26.

[6] M. Cummings and S. Haruyama, "FPGA in the software radio," *Communications Magazine*, vol. 37, no. 2, pp. 108–112, Feb 1999.

[7] K. Bruneel, F. Abouelella, and D. Stroobandt, "Automatically mapping applications to a self-reconfiguring platform," in *Design, Automation and Test in Europe*. IEEE, 2009, pp. 964–969.

[8] I. Kennedy, "A dynamically reconfigured UMTS multi-channel complex code matched filter," in *Field-Programmable Technology*. IEEE, 2005, pp. 199–206.

[9] A. H. Gholamipour, H. Eslami, A. Eltawil, and F. Kurdahi, "Size-reconfiguration delay tradeoffs for a class of DSP blocks in multi-mode communication systems," in *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*. IEEE Computer Society, 2009, pp. 71–78.

[10] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, "A reconfigurable, power-efficient adaptive Viterbi decoder," *IEEE Transactions on VLSI Systems*, vol. 13, no. 4, pp. 484–488, 2005.

[11] J. Becker, M. Hübner, and M. Ullmann, "Power estimation and power measurement of Xilinx Virtex FPGAs: trade-offs and limitations," in *16th Symposium on Integrated Circuits and Systems Design, SBCCI 2003*. IEEE Computer Society, 2003, pp. 283–288.

[12] J. Lamoureux and S. J. E. Wilton, "On the trade-off between power and flexibility of FPGA clock networks," *ACM Transactions Reconfigurable Technology and Systems*, vol. 1, no. 3, pp. 1–33, 2008.

[13] W. Tuttlebee, *Software Defined Radio: Enabling Technologies*. Wiley, 2002.

[14] *Xilinx Logic Core: OPB HWICAP*, Xilinx Inc., July 2006.

[15] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner, and J. Becker, "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput," in *Field Programmable Logic and Applications*. IEEE, 2008, pp. 535–538.

[16] *Virtex-5 Configuration Guide v2.1*, Xilinx Inc., October 2006.