A metric for the energy-efficiency of dynamically reconfigurable systems

Heiko Hinkelmann, Peter Zipf, Manfred Glesner
Darmstadt University of Technology, Institute of Microelectronic Systems
Karlstr. 15, D-64283 Darmstadt, hinkelmann@mes.tu-darmstadt.de

Abstract: Energy-efficiency has become an important design goal for an increasing number of systems today. If flexibility is required in addition, dynamically reconfigurable hardware becomes a promising design solution for such systems, since it combines the potential for providing both high flexibility and energy-efficiency. However, only few approaches exist so far targeting the design for energy-efficiency in dynamically reconfigurable systems. In this paper we present and discuss the RETR metric for energy-efficiency in such systems, taking into account the special characteristics of dynamic reconfiguration. The RETR metric is primarily intended to evaluate a design and estimate the impact of different design decisions, providing a designer with a better understanding of how to increase a dynamically reconfigurable system’s energy-efficiency. For this purpose, a set of conditions and design techniques for optimizing energy-efficiency is identified. The RETR metric could moreover be used to compare and evaluate existing designs.

1 Introduction

Energy-efficiency has become one of the most important design goals for an increasing number of systems, e.g. in the area of mobile communication, wireless sensor networks or embedded systems. Characteristic for most of such systems is their need for providing just a certain level of performance to meet the desired requirements, data rates or standards. Once this level is obtained, these typically battery-powered devices should be optimized for energy-efficiency instead in order to increase their life-time. Many of these systems can furthermore benefit from flexibility or even require a certain degree of flexibility. Motivations for using a flexible system could be support of multiple standards or functions, reuse of hardware components, lower costs, shorter time-to-market, better support of flexible applications (e.g. the adaptation to changing conditions during run-time), and various others. While ASICs are known to be most energy-efficient, they usually provide only poor flexibility. Microprocessors on the other hand offer a high flexibility but typically show much lower energy-efficiency than ASICs, even if low-power processors, DSPs or ASIPs are used [RAdSJ+00]. Reconfigurable computing systems have been identified in [RAdSJ+00] to be able to close this gap between processors and ASICs and provide a compromise between flexibility and energy-efficiency. Other research work targeting the energy-efficiency of reconfigurable devices is described e.g. in [PEW+03] and [Rab97]. In our work presented in this paper we focus in particular on dynamically reconfigurable
computing systems, since they can offer a degree of flexibility that is still relatively high, but are more difficult to optimize for energy-efficiency.

While traditionally designs were optimized for high performance, the optimization for energy-efficiency requires different strategies and results in a significant change in system design, affecting all levels of the design process. A metric for energy-efficiency can help to evaluate a design and estimate the impact of different design decisions, providing a designer with a better understanding of how to increase the system's energy-efficiency. This can be beneficial in particular during the early phase of the design process where simulation or measurement results might be not available or insufficient, but where design decisions could have a large influence on the final system's energy-efficiency. While a lot of work has been done in the domain of low-power design and also metrics for energy-efficiency (e.g. [BB96], [HIG94]), only few approaches exist so far describing the specific characteristics of dynamically reconfigurable systems with regard to energy or power consumption [BST+05].

In this paper we present a metric for the energy-efficiency of dynamically reconfigurable computing systems. It is called the Reconfigurable architectures Energy Throughput Ratio (RETR) and is based on the ETR metric presented in [BB96]. Though the primary intention of our metric is to provide a design guide for the optimization for energy-efficiency of dynamically reconfigurable architectures, it could also be used to evaluate or compare the efficiency of different architectures. The rest of this paper is organized as follows: In section 2 we present and evaluate equations modeling the throughput and the energy consumed for computing and reconfiguration. Based on this results, we derive the RETR metric for the specific characteristics of dynamically reconfigurable systems in chapter 3. In chapter 4 the metric is analyzed in more detail and several design suggestions are derived from it. The paper concludes with a brief summary and an outlook on current research work utilizing the RETR metric.

2 Energy Consumption

To execute any operation on a hardware circuit, a certain amount of energy is consumed, which can be expressed as the average power consumption of this circuit multiplied by the time it takes to complete that operation. In CMOS technology, dynamic power consumption can be modeled as the power required to charge and discharge the output capacitances of all gates involved. Since not all gates are switching in every clock cycle, an effective capacitance $C_{eff}$ can be defined that is the sum of all output capacitances weighed by their corresponding switching activity. This yields the well-known CMOS power equation (1). Another component of dynamic power consumption is short circuit power dissipation (reported in [BB96] to be 5-10% of the total power), which is - like (static) leakage - not considered for our analyses within the scope of this paper. In the following we further distinguish between the power consumed for computing and for reconfiguration.

$$P = \sum (C_{out} \cdot V_{DD}^2 \cdot f_{switch}) = C_{eff} \cdot V_{DD}^2 \cdot f_{clk}$$

(1)
2.1 Computing Costs

For modeling computing costs we define the average energy consumption per operation $E_{ex}$ according to [BB96] as the average power consumed for computing divided by the throughput $T$ of an architecture. $T$ can further be expressed as $\Phi \cdot f_{clk}$, where the performance $\Phi$ denotes the number of operations performed in one clock cycle. We thus get:

$$E_{ex} = \frac{P_{ex}}{T} = \frac{C_{eff} \cdot V_{DD}^2 \cdot f_{clk}}{\Phi \cdot f_{clk}} = \frac{C_{eff} \cdot V_{DD}^2}{\Phi}$$  \hspace{1cm} (2)

An increase in throughput will result in a lower energy per operation if the power consumption remains constant, since the required operations can be performed in less time. But in most cases, power consumption and throughput will be correlated, either because a change in the clock frequency affects both values proportionally or because a modification of the architecture will not only change the performance but also $C_{eff}$.

2.2 Reconfiguration Costs

For expressing the reconfiguration costs we first define a set of parameters characterizing a reconfigurable device. The remanence $R$ defined in [BST+03] can be seen as the number of cycles required for the reconfiguration of the complete device, where $N_a$ is the number of operator units available in this device, $f_{rec}$ is the frequency for the reconfiguration process and $N_c$ is the number of operator units that can be configured per configuration cycle. Similar to [BST+05] we define an effective reconfiguration activity $\alpha$, where $(1/\alpha_{op})$ denotes how many operations can be performed on average on one operator until it must be reconfigured, and $\Phi_{op}$ denotes one operator’s performance, giving the reciprocal value of the average number of cycles required for one operation. The utilization factor $u$ is the ratio of the number of operators in use and the total number of operators available, whereas $v$ specifies the average overhead caused if more operators than in use must be reconfigured (due to the granularity of the reconfiguration process). $u \cdot v$ then yields the average number of operators that are reconfigured compared to the total number.

$$R = \frac{N_a}{N_c} \cdot \frac{f_{clk}}{f_{rec}}$$  \hspace{1cm} (3)

$$\alpha = \alpha_{op} \cdot \Phi_{op} \cdot u \cdot v = \alpha_{op} \cdot \Phi_{op} \cdot \frac{N_u}{N_a} \cdot \frac{N_R}{N_u} = \alpha_{op} \cdot \Phi_{op} \cdot \frac{N_R}{N_a}$$  \hspace{1cm} (4)

For a given application, we can distribute the complete reconfiguration costs over the total number of operations performed, thus getting a value comparable to $E_{ex}$. With the effective reconfiguration latency per operation $L_{rec}$ we define the average reconfiguration energy per operation as:

$$L_{rec} = \frac{\alpha \cdot R}{f_{clk}}$$  \hspace{1cm} (5)
\[ E_{rec} = P_{rec} \cdot L_{rec} = (C_{eff} \cdot V_{DD}^2 \cdot f_{clk}) \cdot \frac{\alpha \cdot R}{f_{clk}} = C_{eff} \cdot V_{DD}^2 \cdot \alpha \cdot R \] (6)

2.3 Throughput and Latency

To get to a more detailed model of throughput, we can observe that the operations per cycle depend on the number \( N_a \) of available operators, the average utilization factor \( u \), the number of cycles required for an operation \( (1/\Phi_{op}) \), and a penalty factor \( p \). This penalty factor can be used for modeling delays in the execution process induced by reconfiguration of the device. \( T \) can thus be expressed as:

\[ T = N_a \cdot u \cdot \Phi_{op} \cdot p \cdot f_{clk} = N_u \cdot \Phi_{op} \cdot p \cdot f_{clk} \] (7)

Accordingly, we define \( L_{ex} = 1/(N_u \cdot \Phi_{op} \cdot f_{clk}) \) as the average execution latency per operation. Multiplying \( L_{ex} \) and \( L_{rec} \) from eq. 5 with the total number of operations \( n \) then yields the overall execution and reconfiguration latencies:

\[ L_{oex} = L_{ex} \cdot n \quad \text{and} \quad L_{orec} = L_{rec} \cdot n \] (8)

The total latency \( L_{total} \) now depends on the grade of parallelism of reconfiguration and execution. It lies in the range of \( L_{oex} + L_{orec} \) for the worst case and \( \max\{L_{orec}, L_{oex}\} \) for the best case. In the worst case, the execution always has to stop for reconfiguration, meaning that both processes never happen in parallel so that their latencies are simply added. When the reconfiguration process can be performed partially or completely in the background, then the total latency would be reduced, down to the value of \( L_{oex} \) in the best case. The reconfiguration latency that can not be hidden, denoted here as \( L_p \), accounts for additional latency and contributes to the penalty factor \( p = (1 - \gamma) \) in our definition for throughput. We can now express \( \gamma \) as the percentage of the total number of cycles where reconfiguration omits execution (eq. 10 denotes only the worst case):

\[ \gamma = \frac{L_p}{L_{total}} \] (9)

\[ \gamma = (1 + \frac{L_{oex}}{L_{orec}})^{-1} \] (10)

3 Energy-Efficiency Metrics

Our RETR metric is based on the energy-efficiency metrics presented by Burd and Brodersen. In [BB96] they defined three metrics for microprocessors: the energy per operation (already reflected in section 2) for applications operating at a fixed throughput, the Energy to Throughput Ratio (ETR) for continuously running applications, and the Microprocessor
ETR (METR) for applications operating at maximum throughput for a fraction of time and being in idle mode otherwise. For the exact definitions we refer to [BB96]. Another metric in use is the energy-delay product [HIG94], but for our purposes the ETR is better suited since it better models the effects of higher performance due to parallelism and pipelining, which is common in reconfigurable architectures. In the following we extend the ETR metric for suitting the special characteristics of reconfigurable architectures, based on the results from section 2.

\[
RETR = \frac{E_{ex} + E_{rec}}{T} = \frac{P_{ex}}{T^2} + \frac{P_{rec} \cdot L_{rec}}{T} \tag{11}
\]

\[
RETR = \frac{P_{ex}}{(N_a \cdot u \cdot \Phi_{op} \cdot p \cdot f_{clk})^2} + \frac{P_{rec} \cdot \alpha \cdot R}{(N_a \cdot u \cdot \Phi_{op} \cdot p \cdot f_{clk}^2)} \tag{12}
\]

\[
RETR = \left[ \frac{C_{ex}}{(N_a \cdot u \cdot \Phi_{op} \cdot p)^2} + \frac{C_{rec} \cdot \alpha \cdot R}{(N_a \cdot u \cdot \Phi_{op} \cdot p)} \right] \cdot \frac{V_{DD}^2}{f_{clk}} \tag{13}
\]

Note that a smaller value for the RETR indicates a better energy-efficiency. If we assume that the computing energy per operation divided by throughput is comparable to the ETR, the RETR metric can also be expressed as:

\[
RETR = ETR \cdot (1 + \frac{E_{rec}}{E_{ex}}) \tag{14}
\]

Under certain conditions, the RETR could be reduced to the ETR. This is obviously true for a non-reconfigurable device with \(P_{rec} = 0\) or if reconfiguration does not occur during runtime, so that \(\alpha \to 0\). But we can also use the ETR metric if there is a fixed relation between the execution and reconfiguration energy per operation, so that the term \(1 + E_{rec}/E_{ex}\) in eq. 14 becomes constant. In the same way as the METR was derived from the ETR, we can derive a more general metric from the RETR that also models idle times and idle energy consumption. For a more detailed view on the consideration of idle times and idle energy consumption we refer to [BB96], since their effects on the metrics are not specific for reconfigurable architectures.

\[
RMETR = \frac{(E_{ex} + E_{rec} + E_{idle})}{T} \tag{15}
\]

One big advantage of these metrics is that they can approximate the effects of voltage scaling, which is one of the most powerful techniques to reduce power consumption. But scaling down the supply voltage also slows down switching and hence decreases clock frequency and throughput. Since this decrease is non-linear, a plot of energy per operation vs. throughput was suggested in [BB96] for highest accuracy. However, like the ETR the RETR can be seen as a linear approximation of this ratio. Since voltage scaling has the same effect on the ETR as on both of the terms related to execution energy (=execution efficiency) and reconfiguration energy (=reconfiguration efficiency) in eqs. 11-13, the corresponding conclusions drawn in [BB96] also apply for the RETR.
4 Design for Energy-Efficiency

4.1 Throughput optimizations

As long as voltage scaling is applicable, higher throughput than required can be traded off for lower energy consumption. Consequently, optimizing a device’s throughput and performance respectively will lead to a better energy-efficiency. However, many design modifications aiming at increasing the throughput (i.e. changing the parameters from eq. 7) could also negatively affect the energy per operation, thus possibly compensating the expected benefits. The effects on $E_{ex}$ and $E_{rec}$ will furthermore be different in many cases. Regarding the RMETR metric, every increase in throughput would also improve the efficiency related to idle energy consumption (see eq. 15).

An increase of the number of available operators $N_a$ could possibly increase $C_{ex}$ and $C_{rec}$, depending on how the operators will be connected and controlled. If these additional operators can be utilized, e.g. by parallel or pipelined processing, the throughput and consequently the execution efficiency will be increased, in the best case quadratically if $C_{ex}$ can be kept constant. For the reconfiguration process, a higher $N_a$ or $N_u$ would probably result in an equivalent increase of the term $\alpha \cdot R$ if the additional operators are reconfigured in the same way, what is usually the case for simple scaling. Then, the reconfiguration efficiency would remain constant or could even decrease if $C_{rec}$ is negatively affected. Hence, more sophisticated solutions for improving the reconfiguration efficiency will be required here (e.g. by reducing $\alpha$ or $R$). The potential for parallel and pipelined processing in an application is usually limited, so that there is an upper bound for $N_u$.

A second way to increase the throughput is to provide operators with a higher performance $\Phi_{op}$, meaning that an operation would require less time. Unfortunately this would also lead to a proportional increase in the effective reconfiguration activity $\alpha$ if $\alpha_{op}$ cannot be reduced at the same time (see eq. 4). Therefore, an improvement in the reconfiguration efficiency can only be achieved if the operators are reconfigured less often per operation.

Another technique to increase throughput is to avoid delays in the execution process due to reconfiguration, bringing the penalty factor $p$ as close to 1 as possible. If the reconfiguration latency is much smaller than the execution latency, i.e. $L_{orec} << L_{oex}$, then $p$ will always be neglectable since $\gamma$ will be close to 0 even in the worst case (see eq. 10). E.g., this could be the case if the reconfiguration process is in general very fast or if a large amount of processing can be done between reconfigurations. Otherwise, an architecture should be able to parallelize reconfiguration and execution as much as possible to be energy-efficient. If this parallelism cannot be achieved and the reconfiguration process prevents execution for a non-neglectable fraction of time, the efficiency is significantly reduced. In this case it should be tried to reduce the overall reconfiguration latency $L_{orec}$ to lower the negative effects at least. From eqs. 5 and 8 follows that this can only be achieved by reducing $\alpha$ or $R$ since $n$ is constant. The importance of optimizing $p$ comes from the fact that it scales down the reconfiguration efficiency linearly and the execution efficiency even quadratically, while there is no direct correlation with another parameter of the metric that would compensate the efficiency reduction.
4.2 Optimization of reconfiguration efficiency

As could be seen so far, an increase in throughput will probably improve the execution efficiency, but would not necessarily affect the reconfiguration efficiency positively. As a consequence, the reconfiguration costs will account for a higher percentage of the total energy consumption and might even become dominant. Therefore, it is important to find additional methods to increase the efficiency of the reconfiguration process. From eqs. 5 and 6 follows that this can be achieved by reducing $L_{rec} \propto \alpha \cdot R$, for which some techniques specific to reconfigurable architectures will be discussed in the following. It should be noted that reducing $L_{rec}$ will not only increase the reconfiguration efficiency alone, but would also reduce the penalty factor $p$ if it is not neglectable, thus resulting in an even bigger efficiency gain.

Whenever a larger amount of operators must be reconfigured, it can be tried to exploit redundancies in the configuration bitstream, e.g. by feeding the same configuration to multiple operators or by configuration compression [LH01]. As a consequence, the remanence $R$ would decrease. As long as this is not compensated by a larger $C_{eff}$ (e.g. due to additional control circuitry), the reconfiguration efficiency would be increased. Another approach with a similar goal is to separate configuration bits that change frequently from those that remain constant for a longer time period. This would also reduce the average size and delay of a reconfiguration by affecting $\alpha$. An example for the application of this technique can be found in [PSS+04].

In some architectures, the granularity of a unit in the reconfiguration process is too high, meaning that more bits than necessary have to be reconfigured. This has been modeled with the parameter $v$ in eq. 4, so that the reconfiguration efficiency scales proportionally to $v$. In the worst case, a device can only be reconfigured as a whole. For energy-efficiency, a designer should try to allow partial reconfiguration at a low granularity. The smaller the units that can be reconfigured separately, the higher is the chance to avoid unnecessary reconfigurations. On the other hand, this should not be done too excessively, since the benefits gained will become lower with smaller granularities and come at the price of an increasing complexity in the reconfiguration control resulting in a higher $C_{rec}$. A suitable choice could e.g. be to match the granularity with the operator’s configuration size or the number of bits that can be reconfigured per clock cycle.

Configuration caching [LCH00] is another powerful technique to lower the average configuration latency. Though a cache in general causes an additional increase in $C_{eff}$, it could already pay off by reducing the number of accesses to the main reconfiguration memory if they come at a much higher price regarding $C_{eff}$ (e.g. because the memory is off-chip) [BB96]. In addition, the average reconfiguration latency can be reduced significantly. In an architecture that allows to cache multiple contexts, the configuration of the complete device could possibly be changed within one clock cycle. If the configuration bits could be stored locally, i.e. close to their destination operator, the resulting overhead for $C_{eff}$ would be limited. Such a configuration caching or multi-context architecture would furthermore allow configuration prefetching. This provides the possibility to load a configuration to the cache before it is needed (provided that the application permits this). The greatest benefit of this approach is its potential to reconfigure the device without stopping execution.
As mentioned before, this becomes very important if the reconfiguration latency is not neglectable compared to the total latency. Clearly, by applying configuration prefetching for a multi-context device, it should be possible to reduce \( p \) significantly or even hide all reconfiguration latencies. Apart from the increased \( C_{eff} \) for the cache or context memory, the prefetching comes at no costs itself.

The required amount of reconfigurations highly depends on the application and on what is mapped to the hardware. Most applications contain kernels, i.e. loops executing the same kind of processing several times. If a reconfiguration occurs within such a kernel (e.g. due to reuse of limited hardware resources), it would be performed at each iteration and the reconfiguration activity \( \alpha \) would consequently be high. If on the other hand a kernel could be mapped to the hardware completely, then \( \alpha \) could be reduced significantly: Assuming a loop of \( m \) iterations (maybe in the range of 10 to 1000) and equal configuration sizes for both cases, \( \alpha \) would also be reduced by a factor \( m \), directly affecting energy-efficiency with the same factor! This small example shows that careful sizing of the reconfigurable hardware can lead to immense energy savings for the reconfiguration, or conversely to a highly degraded efficiency if the size is not appropriate for the application. If the processing in the kernel can be parallelized or pipelined (i.e. high utilization \( u \)), we furthermore have the combined benefits of increased throughput and reduced \( \alpha \) if we increase the number of operators \( N_a \) until the kernel fits completely to the hardware.

Various other techniques exist that increase energy-efficiency by reducing \( C_{eff} \) and should be applied in addition to the techniques discussed above for gaining high energy-efficiency. Common examples are clock gating, operand gating or simply the appropriate sizing of system components and data path widths. Usually such low-power techniques are not specific to dynamically reconfigurable architectures and are hence not discussed here further.

### 4.3 Architecture Comparison

The RETR metric could further be used to compare different architectures. Of special interest is the comparison with a non-reconfigurable device, getting a condition for when the reconfiguration costs pay off. With \( E_{ex,r}, E_{rec,r}, T_r \) for a reconfigurable device and \( E_{ex,nr}, T_{nr} \) for a non-reconfigurable device this condition can be expressed as:

\[
\frac{E_{ex,r} + E_{rec,r}}{T_r} \leq \frac{E_{ex,nr}}{T_{nr}} \quad \text{or} \quad \frac{E_{ex,r}}{T_r} \leq \frac{E_{ex,nr}}{T_{nr}} - \frac{E_{rec,r}}{T_r} \quad (16)
\]

In section 4.1 we found that the reconfiguration efficiency \( E_{rec,r}/T_r \) remains unaffected for an increase in throughput in many cases. If we assume that it is constant and that our non-reconfigurable reference is also constant, their difference gives an upper bound for the execution efficiency of the reconfigurable device. Being beyond that limit could be achieved by reducing the execution energy or increasing the throughput, whereas the improvement of the reconfiguration efficiency could raise that bound. It becomes also clear that if \( E_{rec,r}/T_r \) already exceeds \( E_{ex,nr}/T_{nr} \), it is never possible to compensate the reconfiguration overhead by a better execution efficiency. If we however apply more
sophisticated reconfiguration techniques so that the reconfiguration efficiency can scale with throughput, we can reorder eq. 16 as:

\[
\frac{E_{r,e} + E_{r,x}}{E_{nr}} \leq \frac{T_r}{T_{nr}}
\]

(17)

Now we could easily compare the throughput ratio and the energy per operation ratio of both devices. A throughput gain for the reconfigurable architecture would give room for an equivalent overhead in the energy consumption per operation, whereas any lower ratio of energy per operation would indicate a better energy-efficiency for the reconfigurable device. More details of both architectures, e.g. from simulations or measurements, would allow a more accurate comparison. This could be used to tune the reconfigurable device if there is still room for optimizations of the architecture or its utilization. However, it should be remembered that the RETR is still a linear approximation like the ETR and consequently includes a certain error, in particular for a large scaling of $V_{DD}$ [BB96].

Of special interest is the comparison of a dynamically reconfigurable device with a microprocessor, since the latter would always provide a higher flexibility and is easier to program (good tool support is still a problem for many dynamically reconfigurable architectures). On the other hand, dynamically reconfigurable architectures can provide a much higher throughput. In addition, operators and data paths could be better adapted to the specific application requirements, thus sacrificing flexibility for lower energy per operation. As follows from eq. 17, these features could give plenty of room for providing much better energy-efficiency despite of the reconfiguration overhead (which could moreover be optimized as outlined in section 4.2). How big the differences in throughput and execution energy per operation really are (defining the maximum potential for increased energy-efficiency) finally depends on the target application and the reconfigurable hardware used. A processor could still be more energy-efficient. For the general case, we can just state that dynamically reconfigurable hardware can be a competitive alternative to processors if high energy-efficiency at a certain level of flexibility is the crucial requirement.

5 Conclusion

Dynamically reconfigurable systems have the potential to meet the requirements of applications that demand both high energy-efficiency and flexibility. As presented here, their optimization for energy-efficiency is however not trivial. The RETR metric presented in this paper addresses this problem and provides a better insight for the energy-efficiency of dynamically reconfigurable systems. Based on this metric, a set of design techniques and conditions was identified for optimizing the energy-efficiency. It was shown that execution efficiency could be improved by increased throughput, whereas the reconfiguration efficiency was not affected in most cases. Instead it could be improved best by applying advanced reconfiguration techniques in order to reduce the average reconfiguration latency per operation. It has furthermore been seen that mapping kernels completely to the reconfigurable architecture could also result in a high gain in the reconfiguration efficiency, meaning that size and utilization of an architecture should be matched with the target ap-
plication’s characteristics. If reconfiguration time is not neglectable compared to execution time, a negative effect that can significantly degrade the energy-efficiency was identified for the case that the reconfiguration process delays execution. Here, the parallelization of reconfiguration and execution would be much more energy-efficient.

In our actual research work, a dynamically reconfigurable computing platform for a wireless sensor node is currently under development. The RETR metric was applied for finding possible design optimizations and for evaluating design decisions concerning their expected effect on the architecture’s energy-efficiency. This work provides the practical background for the RETR metric and is one example for its useful application.

Literatur


