Microelectronics System Design Research Group

School of Electrical, Electronic & Computer Engineering

Quality of Service in Power Proportional Computing

Danil Sokolov, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2011-171

July 2011

Contact: danil.sokolov@ncl.ac.uk, alex.yakovlev@ncl.ac.uk

Supported by EPSRC grant EP/G066728/1

NCL-EECE-MSD-TR-2011-171

Microelectronics System Design Research Group
School of Electrical, Electronic & Computer Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

Quality of Service in Power Proportional Computing

Danil Sokolov, Alex Yakovlev

July 2011

Abstract

Power proportionality becomes an important aspect of computation under unpredictable energy supply, e.g. if the energy is being scavenged from the environment and its income varies significantly during the computation process. It is advantageous to build the computational load of energy harvesting electronics in such a way that it continues the operation even in energy deficient conditions, possibly degrading the Quality of Service (QoS). In this work we studied two possibilities for trading the QoS for energy consumption: by changing the computation speed and by adjusting the computation precision.

1 Introduction

This work is set in the context of energy harvesting electronics where a computational load benefits from unlimited energy supply, but has to adapt to unpredictable power fluctuations. We have performed a number of experiments trying to build the computational load in a power proportional way. Power proportionality is the key aspect of energy-modulated computing [1] - a power proportional circuit continues to produce correct results even when the supplied power is below the nominal level, possibly at lower rate or with less precision. This is basically a trade off between the QoS and the energy consumption of the system.

Depending on the application different aspects of the QoS can be sacrificed for power. Usually a circuit speed is the first candidate for power saving through Dynamic Voltage and Frequency Scaling (DVFS) [2]. However, at ultra-low power levels reducing the circuit speed may become a limiting factor for real-time systems [3]. A possible solution to this issue is to degrade the data precision (e.g. bit-width reduction in signal processing) or to skip the computation cycles (e.g. frame skipping in video processing). This would help to save power while maintaining the timing of the circuit at the required standard.

The following sections present the experiments we performed in order to estimate the benefits and the overheads associated with the power proportional computing.

2 Method

We partitioned the experiments into two categories in order to quantify the QoS in terms of speed and in terms of data precision.

In the first group, the source voltage was reduced in small steps from a nominal value until the circuit started to malfunction. As expected, the voltage reduction caused the increase in the computation time, which was interpreted as the QoS in these benchmarks. These experiments required very slow analogue simulation

and therefore we chose to benchmark a set of commonly used small circuits – counters, adders and multipliers of various sizes. All the experiments demonstrated very similar results, therefore the trends are analysed using an 8 bit Booth's multiplier only.

To study the QoS in terms of computation precision we chose a popular domain of signal processing where a Fourier transform plays a key role. The following sections overview the specifics of this transformation for digital circuits and present an implementation which can change the number of sampled points and the fixed-point data representation in run-time - these nobs were utilised to mitigate the energy deficiency in the system.

2.1 Fourier Transform

The core of signal processing is a Fourier transform - a conversion of signals between the timing and frequency domains. The transformation of a time function $f(t)$ into a frequency function $F(\omega)$ is described by the following equation:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot cos(2 \cdot \pi \cdot \omega \cdot t)\, dt + i \cdot \int_{-\infty}^{\infty} f(t) \cdot sin(2 \cdot \pi \cdot \omega \cdot t)\, dt \tag{1}$$

In practice an analogue signal is sampled prior processing and a Discrete Fourier Transform (DFT) is used, where the integration is approximated as follows:

$$F(\omega) = \sum_{n=0}^{N-1} x_n \cdot cos(2 \cdot \pi \cdot \omega \cdot n) + i \cdot \sum_{n=0}^{N-1} x_n \cdot sin(2 \cdot \pi \cdot \omega \cdot n) \tag{2}$$

There are certain restrictions for the minimal rate at which a signal should be sampled in order to catch all its components. For example, the functions $f1(t)$, $f2(t)$ and $f3(t)$ in Figure 1 appear the same if they are sampled at integer times and cannot be correctly recognised by DFT.

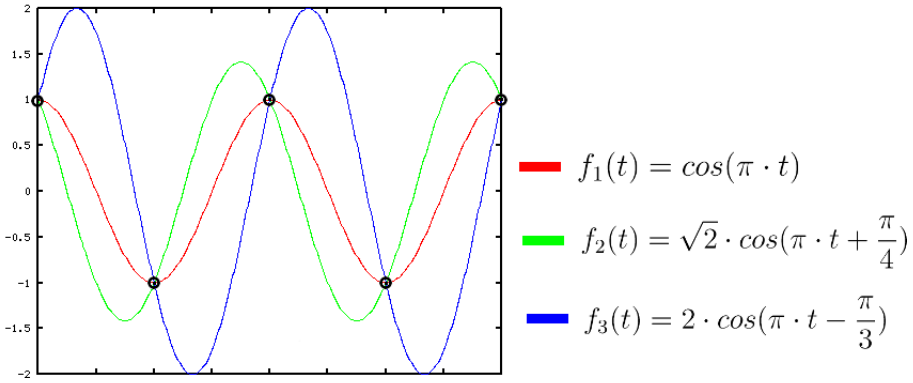

Figure 1: Importance of signal sampling rate

The minimal sampling rate at which all signal components can be distinguished is called a Nyquest rate and must be higher than double frequency of the signal bandwidth. Otherwise the DFT may result in detecting so-called alias frequencies which are not present in the original signal. If a signal cannot be sampled at high enough frequency then in order to avoid incorrect DFT results the high frequency components (those higher than half of the sampling rate) should be filtered out by an analogue low-pass filter.

The main drawback of the DFT algorithm is its $O(N^2)$ complexity with respect to the number of sample points N which makes it impractical for processing large series of samples. A significant improvement is achieved in Fast Fourier Transform (FFT) algorithm which reduces the computation complexity to $O(N \cdot log_2(N))$. The intuition behind the FFT algorithm is as follows. The DFT of the first $N/2$ points can be combined with the DFT of the second $N/2$ points to produce a single$N$-point DFT. Each of these $N/2$-point DFTs can then be calculated using smaller DFTs in the same way. This operation repeats recursively until only two points need to be processed, which is a trivial DFT operation called a *butterfly*:

$$XO = X + Y \cdot W,\ YO = X - Y \cdot W \tag{3}$$

Here X and Y are two points to transform and W is a *twiddle factor* which is usually pre-calculated for each iteration of the FFT algorithm as follows:

$$W[k] = cos\left(\frac{2 \cdot \pi \cdot k}{N}\right) + i \cdot sin\left(\frac{2 \cdot \pi \cdot k}{N}\right) \tag{4}$$

An FFT algorithm for an 8-point series with the required twiddle factors at each stage is illustrated by a Signal Flow Graph (SFG) in Figure 2. At the load stage the input samples are permuted to guarantee a natural order of $F(\omega)$ at the output of the transformation. At stage 1 four 2-point DFTs are calculated, which are subsequently combined to form two 4-point DFTs at stage 2. Finally, at stage 3, a single 8-point DFT is produced in a sorted order and can be unloaded for further calculations. Note that at each of the $log(N)$ stages there are $N/2$ butterfly operations performed which explains the computation complexity of the FFT algorithm.

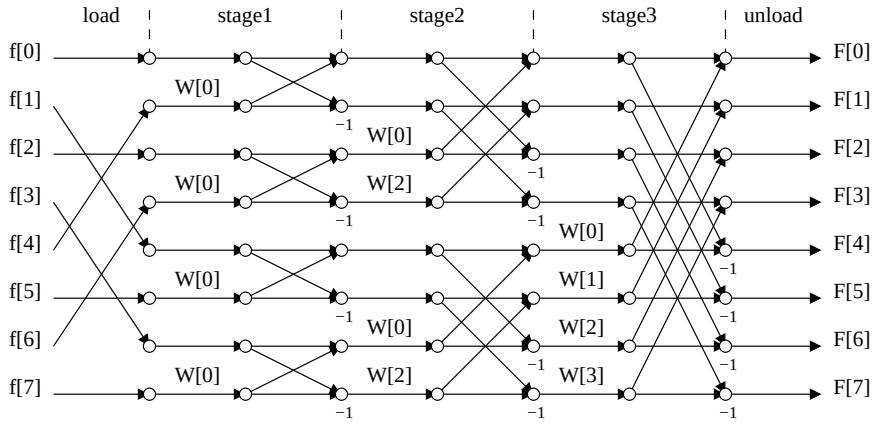

Figure 2: Signal flow graph for 8 point FFT

For more details on FFT a reader is referred to any book on digital signal processing, e.g. [4].

2.2 Reconfigurable FFT

There are two distinctive hardware architectures for FFT: *pipelined* and *iterative*. The former architecture uses a distinctive layer of butterflies for each stage of transformation, which results in high throughput and low latency (due to concurrent computation of the butterfly functions in each stage and the pipeline effect between the stages) at the expense of area and energy consumption. The former architecture utilises a single instance of

the butterfly to repeatedly process the points stored in memory. At each iteration a pair of points is read from the memory, processed by the butterfly and stored back in memory (overwriting the previously stored data). We focus on the iterative architecture only as it better suites low-power operation and is particularly convenient for varying the number of points to transform.

The schematic diagram of an iterative FFT is depicted in Figure 3. Its operation has tree distinctive modes: *load*, *calculate* and *unload*. It starts with $N/2$ cycles of loading the sampled data into memory in a bit-reverse order (*load*=1, *unload*=0) followed by $\frac{log_2(N) \cdot N}{2}$ cycles of transformation (*load*=0, *unload*=0) and finished by $N/2$ cycles of unloading the processed data for further calculations (*load*=0, *unload*=1). Due to the specifics of the FFT algorithm the memory module is implemented as a dual-channel RAM with an interface capable of reading and writing data words (points) in pairs. At each transformation cycle the butterfly module takes a pair of points from the RAM, performs a 2-point DFT using the pre-computed twiddle factors from the ROM and stores the result back in the same RAM addresses. The control module conducts all this activity by calculating the required addresses for *X*, *Y* and *W* which depend on the operation mode and the iteration index, see [5] for details.

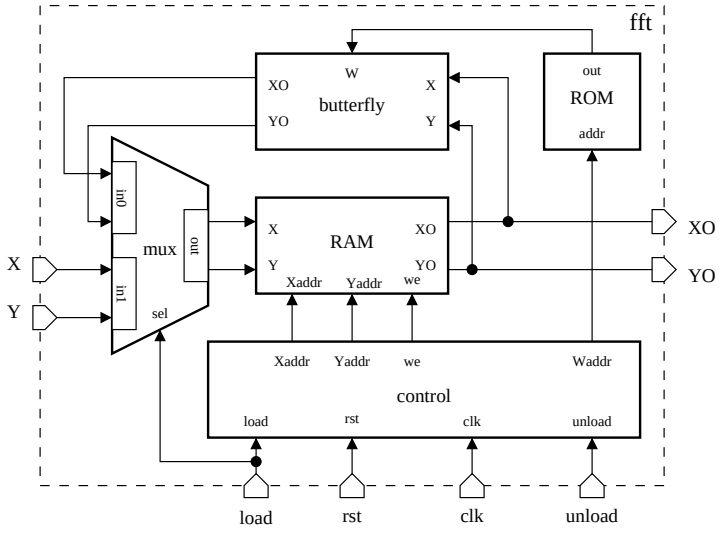

Figure 3: Schematic of an iterative FFT implementation

There are several free RTL implementations of the iterative FFT, e.g. Spiral RTL generator (*http://www.spiral.net/hardware/dftgen.html*) and Open Cores library (*http://opencores.org/*). Both of them support a wide range of design-time parameters, such as transform size (16, 32, 64, 128, 256, 512 and 1024 samples) and data precision (4, 8, 16 and 32 bits). Unfortunately there is no open FFT implementation which allows making these decisions at run-time (rather than at design-time) which is essential for power-proportional computing.

We addressed the limitation of run-time configurability by building an FFT processor whose iterative architecture is based on work presented in [6], where a reconfigurable transform size was introduced. Our implementation was extended with a flexible choice for the number of bits to represent the fixed-point data - the fewer is the number of bits, the less is the switching activity and hence the lower is the energy consumption. We also performed a basic desynchronisation of the FFT to exploit the timing differences of load, calculation and unload modes. The schematic of the obtained circuit is shown in Figure 4.

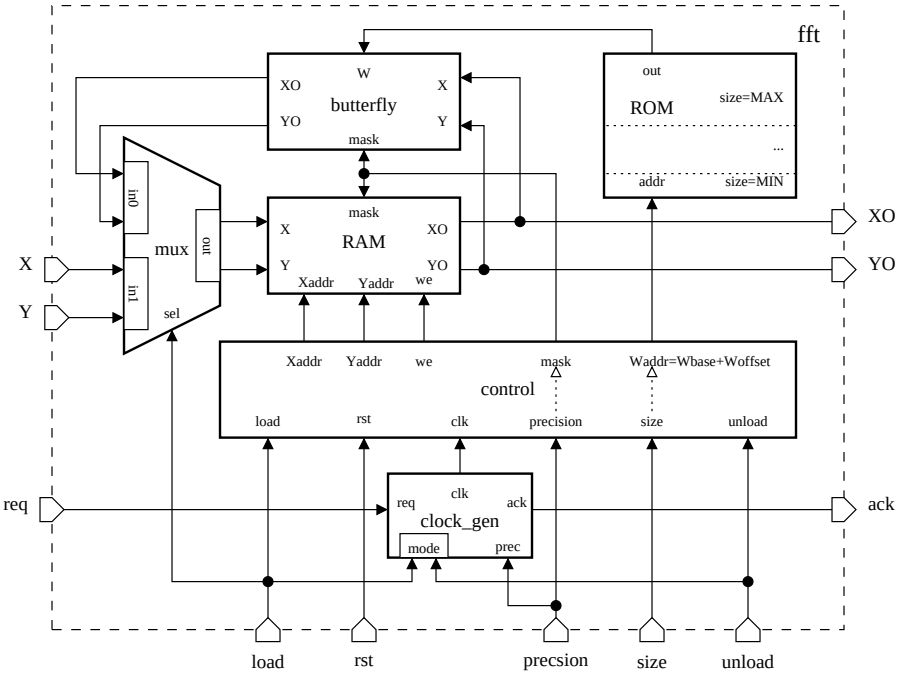

Figure 4: Desynchronised implementation of reconfigurable FFT

The main modifications in this design compared to the standard iterative FFT implementation are as follows. The control module is extended with the *size* input to select the number of points to transform. Besides the number of iterations this input also selects the appropriate bank of twiddle factors from the ROM. Another improvement is the *precision* input which specify the number of bits used for fixed-point data representation. This input determines the number of the least significant bits which are "masked" in the butterfly and in the RAM module (the corresponding write-enable lines are disabled). This basic masking approach allows to reduce the switching activity in the circuit and thus to reduce the dynamic power, but does not help with the leakage - for this a more sophisticated power gating scheme is required.

In the desynchronised design a two-phase request-acknowledgement protocol is employed to generate an "elastic" clock signal, as shown in Figure 5. In order to satisfy the setup condition, a clock pulse is produced as a reaction to the *req* signal after a delay which corresponds to the critical path. Similarly, for the hold condition the *ack* signal is issued after a delay which corresponds to the shortest path.

Note that the load, calculate and unload modes of FFT activate significantly different critical paths. In order to exploit this difference we made use of an adjustable delay line. In the calculate mode the critical path is also dependent on the data precision – this is captured by another level of delay line configurability. Such desynchronised design with adjustable clock speed helps to improve the computation time and thus to reduce the amount of "leaked" energy, which is confirmed by the benchmark results.

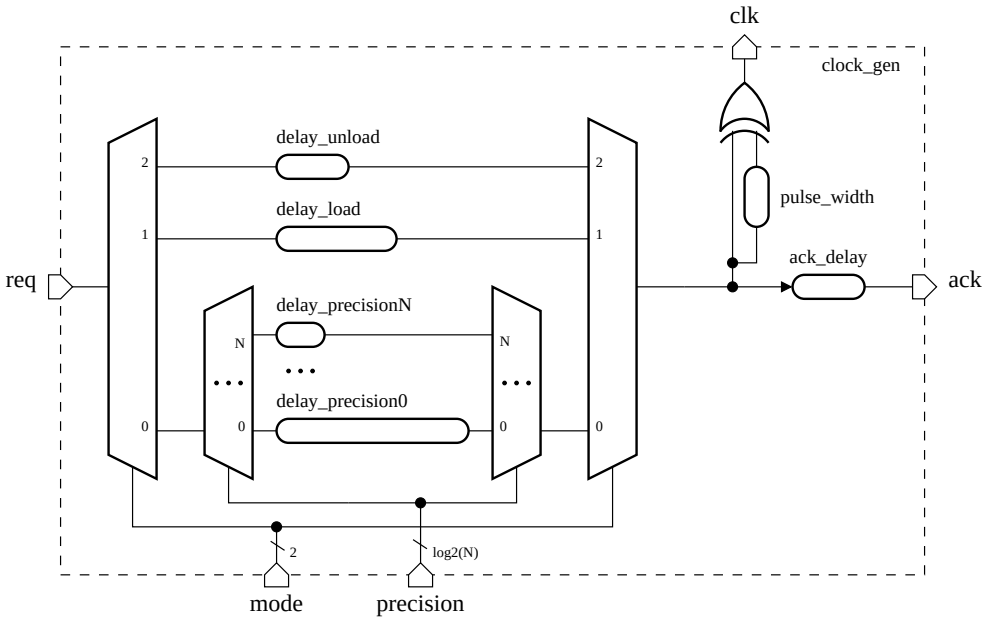

Figure 5: Elastic clock generator

3 Results

In this section the simulation results and their analysis in terms of QoS under restricted energy supply are presented. All the benchmarks used in these experiments were synthesised for Faraday library based on UMC 90nm technology process.

In the first set of experiments the computation speed was interpreted as the QoS which could be traded for the energy consumption. Three implementations of Booth's multiplier were simulated: a) synchronous design synthesised from RTL description, b) asynchronous bundled data design obtained from the synchronous version by applying a desynchronisation technique [7] and c) asynchronous dual-rail design generated from the synchronous version by the VERIMAP tool [8]. The synchronous implementation was simulated in two modes: fixed frequency mode (the clock was set to 1GHz) and frequency scaling mode (the clock frequency was artificially reduced from 1GHz to 500MHz and then to 250MHz when the multiplier could not compute in time).

In order to measure the energy consumption at various "non-standard" supply voltages we employed the SPECTRE analogue simulator. Initially, each Booth's multiplier implementation was simulated at a nominal 1V and its energy consumption and computation time were recorded. Then the supply voltage was reduced to 0.9V, the simulation was re-run. This procedure was repeated several times, with reduction of the supply voltage by 0.1V each iteration until the circuit started malfunctioning (this was detected by observing the signal waveforms).

The simulation results are summarised in Table 1. The synchronous design required the frequency step-down first at 0.7V and then at 0.5V; further frequency reduction could not help at 0.5V. The main feature of asynchronous bundled data implementation is that it "tracks" the variations of operating conditions (e.g. supply voltage) and naturally "adapts" the generated clock frequency to these changes in a wide range. This was

confirmed by the simulation – the bundled data multiplier implementation continued to produce correct results even at 0.3V. There is a moderate area overhead and nearly twofold increase in energy consumption compared to the synchronous implementation. These overheads can be explained by the extra control logic whose fixed area and energy consumption becomes negligible with the increase of the design size.

Table 1: Simulation results for Booth's multiplier

Implementation of Booth's multiplier	**Area (cells)**	**Energy consumption (pJ) - top / *Computation time (ns) - bottom***								
		1.0V	**0.9V**	**0.8V**	**0.7V**	**0.6V**	**0.5V**	**0.4V**	**0.3V**	**0.2V**
Synchronous, fixed frequency @1GHz	2495	22.5 *18.0*	17.9 *18.0*	13.8 *18.0*	-	-	-	-	-	-
Synchronous, freq. scaling @1GHz, 500MHz, 250MHz	2495	22.5 *18.0*	17.9 *18.0*	13.8 *18.0*	10.4 *36.0*	7.5 *36.0*	5.2 *72.0*	-	-	-
Asynchronous, bundled data	2931	39.6 *17.2*	30.8 *19.2*	23.3 *22.2*	17.2 *26.9*	12.3 *35.2*	8.4 *52.1*	5.4 *96.1*	3.3 *263.9*	-
Asynchronous, dual-rail	7683	279.0 *38.1*	217.3 *41.6*	166.1 *49.5*	123.8 *61.5*	87.1 *106.1*	60.8 *142.6*	39.1 *283.6*	24.3 *831.4*	19.1 *4140.0*

Further reduction of the supply voltage down to 0.2V was survived by dual-rail implementation only. However, this achievement was at the expense of a threefold area increase compared to the synchronous design. This is due to the combination of the following overheads associated with dual-rail design: duplication of combinational logic, extra completion detection circuitry; use of complicated dual-rail flip-flops and additional circuits to provide the interface to a single-rail environment. Also the switching activity of the dual-rail circuits are usually much higher than for the single-rail ones because each dual-rail gate goes through a spacer and code-word state every computation cycle.

An interesting interpretation of the above results was found in terms of speed as the QoS for a given energy rate, which is illustrated in Figure 6 (for low power levels only). This interpretation correlates with the case of energy being harvested from the environment at an unpredictable rate and a power-proportional computational load adapting its computation speed to this energy rate. One can observe that the synchronous implementation jumps between the QoS levels in huge steps which means that most of the time the circuit operates inefficiently (not power proportionally). Contrary, the asynchronous bundled data implementation exhibits good power proportionality and utilises the energy more efficiently than synchronous design almost at all power levels. The use of dual-rail design is somewhat questionable as it only beats the bundled data implementation at the very low (below 0.01pW) power levels at the expense of extremely slow computation. The degraded QoS of dual-rail circuits in combination with the threefold area overhead may become too much of a price to pay for a relatively small benefit of operating at a very low power.

In the second set of experiments the computation precision was considered as the QoS which can be traded for energy. The supply voltage was fixed at the nominal 1V and only the data precision was changed.

We benchmarked several implementations of iterative FFT: a) non-reconfigurable 1024-point FFT; b) reconfigurable FFT with up to 1024-points; c) FFT with variable data precision; d) fully reconfigurable FFT (a combination of the previous two); e) asynchronous bundled data implementation with adjustable delay (as described in the previous section).

The use of nominal voltage allowed us to employ a fast digital SYNOPSYS VCS simulator (as opposed to slow analogue simulation). Each FFT implementation was simulated by the same testbench (with a minor mod-

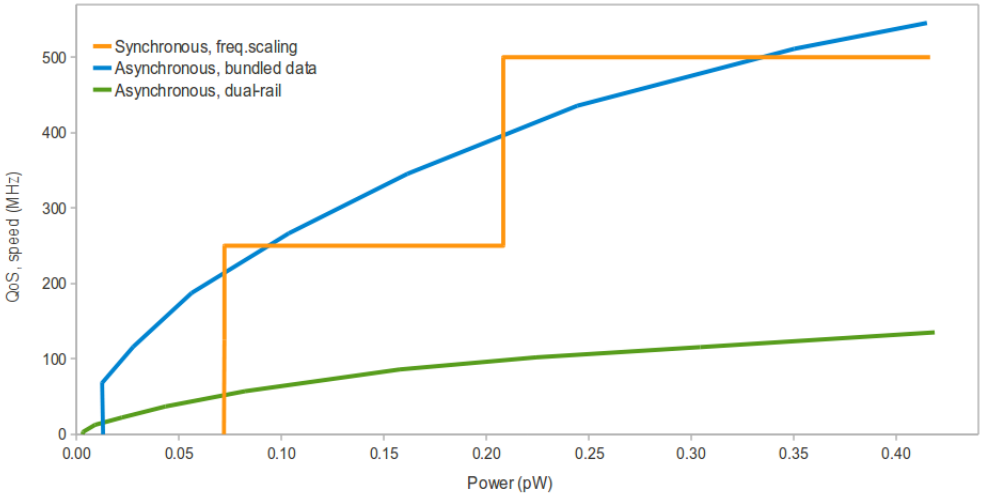

Figure 6: Speed as the QoS in Booth's multiplier benchmark

ification for request-acknowledgement interface of the asynchronous implementation) which exercised three transform sizes (1024, 512 and 256-points) in all combinations with three data representations (16, 12 and 8-bit). The switching activity of each run was dumped into a VCD file and subsequently analysed SYNOPSYS in PRIMETIME (with PX extension) to estimate the power consumption of the circuits. The obtained results are presented in Table 2 and visualised in Figure 7.

Table 2: Simulation results for iterative FFT designs

Implementation of FFT	**Area (cells)**	**Energy consumption (μJ)**								
		1024 samples			**512 samples**			**256 samples**		
		16bit	**12bit**	**8bit**	**16bit**	**12bit**	**8bit**	**16bit**	**12bit**	**8bit**
Non-reconfigurable	914K	5.51	-	-	-	-	-	-	-	-
Reconf. transform size	920K	6.59	-	-	3.01	-	-	1.36	-	-
Reconf. data precision	925K	5.83	4.71	3.40	-	-	-	-	-	-
Reconf. size and precision	936K	6.66	5.39	3.90	2.98	2.41	1.76	1.32	1.08	0.79
Asynchronous, bundled data	942K	6.09	4.92	3.55	2.73	2.20	1.60	1.21	0.98	0.72

One can notice that the asynchronous bundled data implementation exhibits the best proportionality of its QoS (computation precision) to the amount of energy consumed. The only overhead compared to the fixed-size synchronous FFT is the increase of circuit size and energy consumption for the 1024-point transformation. Both these overheads are due to the additional control logic and the clock generator module. While the area overhead is only 3%, the extended control module has the highest switching activity and contributes to the 10% increase in energy consumption. Interestingly, the bundled data implementation shows better energy consumption than the fully reconfigurable synchronous design. This is because the adjustable clock speed saves more energy by reducing the leakage (the circuit is doing "nothing" for shorter period of time) than is consumed due to extra switching activity of the clock generator module.

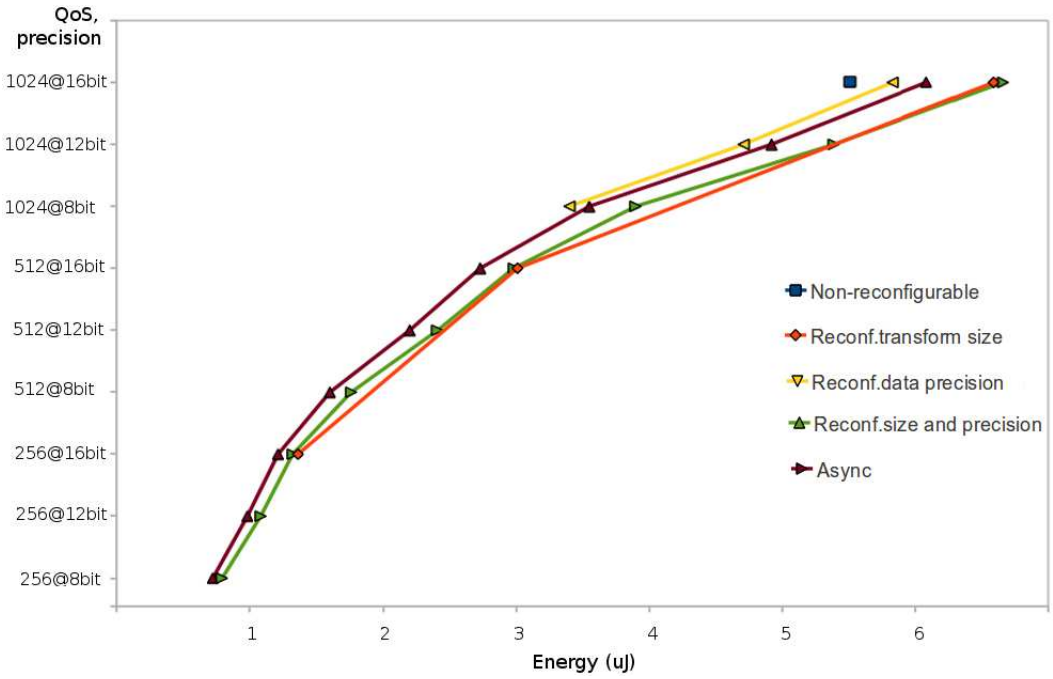

Figure 7: Data precision as the QoS in iterative FFT benchmark

4 Conclusions and Future Work

The power-adaptive systems are essential for truly energy-modulated computing as they exhibit better utilisation of energy at low power levels and in general are more resilient to unpredictable power fluctuations in a broad range of power levels. Our experiments demonstrate that asynchronous design techniques may play a significant role for building the power-adaptive systems because self-timed systems exhibit adaptability to varying environmental conditions and offer better energy proportionality than synchronous counterparts.

Good characterisation of circuit components is of paramount importance for high-quality adaptation of a design to different power levels. In our experiments we simulated the designs before place-and-route, which may introduce some error. Also the memory module in FFT design was built out of flip-flops as a scratch-pad rather than a real RAM. It would be interesting to see how the use of proper memory and the presence of layout information influences the simulation results.

There may be a potential improvement for reconfigurable FFT in the following scenario. The circuit starts the transformation at the lowest sampling rate and obtains a low precision result. Then, if the energy is sufficient, another iteration is made at a higher sampling rate and the results are updated with a higher precision data. This procedure repeats until either of the following conditions is met: a) all the energy is consumed; b) the highest sampling rate is achieved; c) the transformation results are urgently required by the consumer of the service (e.g. in a real-time system). The main challenge in this approach is to re-use the results of the previous iterations when computing the higher precision results.

Currently we are missing a unified theory, abstract models and generic algorithms for handling the power-adaptation at run-time. The energy token model [9] is the first step in this direction and is also a subject for further research.

References

[1] Alex Yakovlev. “Energy-modulated computing”, In Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE), pp.1-6, 2011

[2] Maurice Meijer, Jose Pineda de Gyvez, Ralph Otten. “On-chip digital power supply control for system-on-chip applications”, In Proc. International Symposium on Low Power Electronics and Design (ISLPED), 2005

[3] Kees Goossens, Dongrui She, Aleksandar Milutinovic, Anca Molnos. "Composable dynamic voltage and frequency scaling and power management for dataflow applications", In Proc. Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD), 2010

[4] Steven Smith. “The scientist and engineer’s guide to digital signal processing”, California Technical Publishing, 1997

[5] Danny Cohen. “Simplified control of FFT hardware”, IEEE Trans. Acoustics, Speech and Signal Processing, pp.577 578, 1976

[6] Yutian Zhao, Ahmet Erdogan, Tughrul Arslan. “A low-power and domain-specific reconfigurable FFT fabric for system-on-chip applications”, Proc. International Parallel and Distributed Processing Symposium (IPDPS), 2005

[7] Jordi Cortadella, Alex Kondratyev, Luciano Lavagno, Christos Sotiriou. “Desynchronization: Synthesis of Asynchronous Circuits From Synchronous Specifications”. IEEE Trans. on CAD, vol.25(10), pp.1904 1921, 2006

[8] Danil Sokolov, Julian Murphy, Alex Bystrov, Alex Yakovlev: ’Design and analysis of dual-rail circuits for security applications’, IEEE Trans. on Computers, 54(4), pp.449 460, 2005

[9] Danil Sokolov, Alex Yakovlev. “Task scheduling based on energy token model”, Workshop on Micro Power Management for Macro Systems on Chip (uPM2SoC), 2011