XPP Technologies

Reconfiguration on XPP-III Processors

White Paper

For further information and questions, please contact support@pactxpp.com.

Version 2.0 July 12, 2006

1 Introduction

This White Paper describes the configuration methods employed by PACT's XPP-III architecture. After presenting the basic principles, the typical use of a sequence of configurations in a complete application is explained, with special emphasis on data-stream processing. Finally, some examples illustrate the system design trade-offs between application performance, XPP processor size and frequency, and the number of configurations. The reconfiguration overhead in terms of processing cycles and power consumption is discussed in detail for one example.

For basic information on XPP-III processors, please refer to the White Paper *XPP-III Processor Overview*. Programming methods and tools for XPP systems are described in the White Paper *Programming XPP-III Processors*.

2 Configuration Handling on XPP-III Processors

2.1 Configuration Registers and Configuration Words

The configuration executed on a XPP processor, i. e. the control/dataflow graph mapped to its ALU-/RAM-PAE processing array (also called *XPP dataflow array* in this paper), is stored in *configuration registers* distributed throughout the XPP dataflow array. They control the functionality of the XPP objects, i. e. which opcodes are executed by an ALU, the connections between ALUs, memories and registers, etc.

After a XPP reset and at power-up, all configuration registers are reset to their default value. All related XPP objects are in the inactive "not configured" mode. In this mode, every configuration register can be set by a *configuration word*. It consists of an address (i. e. the PAE row and column number and the number of the register within the PAE) and a value which is written to the configuration register being addressed. This addressing scheme allows to configure PAEs individually and independently.

2.2 Loading Configurations

A single configuration register in the XPP dataflow array is loaded by applying the configuration word to the XPP's *configuration port*. At the next clock edge, the word is stored at the front of the *configuration pipeline*. The pipeline forwards the word to the next row and the next column every cycle, i. e. it distributes the word to all PAEs, cf. Figure 1. When it reaches the PAE being addressed, its value is written to the respective configuration register. Hence the configuration register is set after at most $row + column$ cycles.

To load an entire configuration, all configuration words are sequentially applied to the configuration port, cycle by cycle. Due to the configuration pipeline, one register can be written every cycle after the pipeline is filled. Only the XPP objects which are part

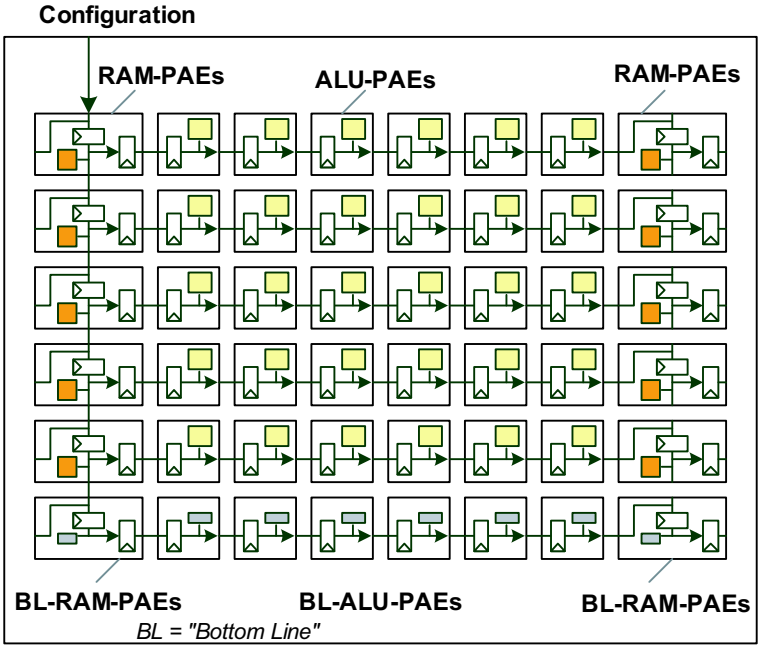

Figure 1: XPP processor with configuration pipeline (without FNC-PAEs).

of the configuration need to be configured (*partial configuration*). Hence larger configurations require more configuration words, and the configuration time is equivalent to the number of configuration words. On the other hand, small configurations are loaded very quickly without any overhead for managing the unused PAEs. Note that several completely independent partial configurations, i. e. configurations not sharing any resources, can be loaded to a XPP dataflow array and execute concurrently. These configurations represent independent tasks operating on their own memories or data streams.

When an object has been configured completely, it is set to “configured” mode. It immediately starts computing, provided its input values are available and the output values are consumed. Note that an object can only accept a new configuration after it has been reset to “not configured”.

However, the XPP dataflow array also supports special configuration words which can address XPP objects which are already in “configured” mode. One of these configuration words sets input registers in XPP objects. It can be used to change constants in a configuration. This *differential configuration* allows e. g. to quickly adapt filter coefficients without the need to reconfigure the entire filter (i. e. ALUs and connections). Another use of this feature allows to insert packets, e. g. an event packet to restart a counter. It is thus possible to restart a computation using the configuration port. Finally, a special configuration word is used to remove an object (and its configuration) from the array, as described in the next section.

2.3 Removing Configurations

To remove a configuration from the XPP dataflow array, all used XPP objects must be reset to the "not configured" mode by resetting all used configuration registers. Since all objects of a configuration need to be removed together, the XPP dataflow array contains special signals which propagate the remove signal from one object to all other objects connected by event or data connections. Hence it is only necessary to explicitly remove one object. All connected objects will then be removed in the following cycles. Only completely independent configurations (i. e. control/dataflow graphs) remain configured. The removal can be triggered by the following two methods:

- External remove: As mentioned in the previous section, a special configuration word can be applied to the configuration port. It removes the object being addressed.
- Internal remove: If an event packet is received at the `RECONF` input of an ALU, the ALU removes itself immediately.

2.4 Power Consumption during Computation and Configuration

This section discusses the power consumption of a PAE (16-bit word length) in 90 nm CMOS technology, both during computation and configuration.

$P_{PAEcomp}$, the simulated average power consumption of a PAE during computation, is about 30 μW/MHz. While the PAE is being configured, its consumption P_{PAEcfg} is in the same range of about 30 μW/MHz. The additional consumption P_{BUScfg} of the configuration bus is about 7 μW/MHz *for each PAE* of the XPP processor.

Hence the power consumption of an entire XPP dataflow array can be computed as follows:

$P_{Computation} = P_{PAEcomp} \times N_{configuredPAEs}$

$P_{Configuration} = P_{PAEcfg} + P_{BUScfg} \times N_{PAEs}$

Here, $N_{configuredPAEs}$ denotes the number of PAEs configured for an application, and N_{PAEs} denotes the overall number of PAEs in the XPP processor.

3 Processing and Reconfiguration

This section describes how XPP configurations are used on a XPP-III processor with memory. In such a system, the XPP dataflow array is typically configured by a Function-PAE (FNC-PAE) processor kernel using a DMA channel. The configurations are stored in on-chip or off-chip RAM.

3.1 Mapping Applications to Configurations

An application must be partitioned between the FNC-PAEs and the XPP dataflow array. Application parts which can be parallelized and which are suitable for execution on the XPP dataflow array are mapped to configurations. The mapping also depends on the size of the available XPP processor. To amortize the reconfiguration time, each configuration should execute a larger "phase" of the application, typically several thousand cycles. After the computation of a configuration has terminated, the next configuration is loaded for the next phase, etc. The FNC-PAEs and the dataflow array exectue concurrently.

Memories, i. e. XPP IRAMs or external RAMs, play an important role in the system design since they hold intermediate results during reconfiguration, and may communicate values between FNC-PAEs and the dataflow array. Note that long XPP phases result in a better amortization of the reconfiguration time, but normally require larger buffers for intermediate results. Therefore a suitable trade-off between the conflicting design goals, long XPP phases and small buffers, must be found.

3.2 Stream Processing Requirements

Data-stream processing systems are a special type of real-time systems which need to process input data streams at a given sample rate and with a maximum output latency. They typically repeat the sequence of configurations in an infinite loop. The timing and buffering requirements are critical for these systems: An input buffer is required which is large enough to store all data arriving during a repetition of the entire configuration sequence. Since the reconfigurations take time and the samples are normally accessed repeatedly, the sample rate must be lower than the XPP frequency. Only if there is just one XPP configuration which processes one sample every cycle, the sample rate can be equal to the XPP frequency. For a given sample rate, the system designer must select an appropriate trade-off between XPP frequency, buffer size, output latency and XPP processor size. The latter is important since larger XPP processors reduce the number of required reconfigurations in many cases. The next section analyzes these trade-offs in more detail.

3.3 Application Trade-Offs

This section presents an analysis of the design trade-offs for an "ideal" streaming application. The example applications presented in Section 4 will apply this analysis to real-world applications.[1]

The ideal streaming application meets the following assupmtions:

- The application can be executed in one configuration on a large XPP processor, or can be split arbitrarily into smaller configurations on smaller XPP processors, cf. Figure 2.

[1] Note that the FIR filter application presented in Section 4.1 meets the assumptions of this section.

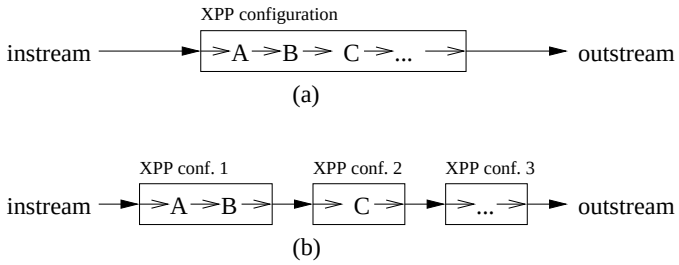

Figure 2: Ideal streaming application: (a) Execution in one configuration. (b) Any split between stages A, B, C etc. into separate configurations is possible.

- Every stage of the application (A, B, C etc.) is mapped to one ALU-PAE on the XPP dataflow array.
- Per configuration, every sample must only be read once and written once. The first configuration reads the input samples, and the other configurations only read the outputs of the previous configuration.
- The throughput of all configurations is one sample per cycle.
- After the initial configuration, only differential configurations are required. They contain one configuration word per ALU-PAE.

If the application fits in one configuration, no buffer memory is required. The input and output streams can be directly connected to the XPP processor as shown in Figure 2(a). However, if we need to switch between several configurations, a double buffering scheme is required, cf. Figure 3. It uses two RAM banks storing N samples each. They are capable of one read and one write operation in one XPP cycle. While N input samples (from *instream*) are written to RAM1, the previous N samples (stored in RAM2) are sequenatially processed by all configuration on the XPP processor. Next, the roles of RAM1 and RAM2 are swapped as indicated by the dotted arrows connecting the RAMs and the XPP processor in Figure 3.

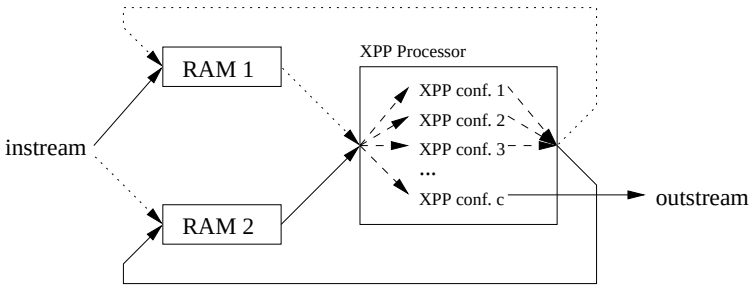

Figure 3: Dual-buffer system for implementation using several configurations.

Param.	Parameter Meaning	Definition
F_S	Frequency of input samples	
k	Number of stages of the applications = ALU-PAEs required if executed in one conf.	
P	Number of ALU-PAEs available in XPP processor	
c	Number of configurations	$c = ceil(k/P)$
M	Words of RAM memory, combined for both buffers	
N	Number of samples processed in one conf. sequence	$N = M/2$
pipefill	Number of cycles required for pipeline filling	*pipefill* $\leq P$
diffconf	Number of configuration words (and cycles) required for differential configuration	*diffconf* $\approx P$

Table 1: System parameters.

In the XPP dataflow array, XPP configuration 1 reads its input data from the selected RAM and writes its results back to the same RAM. Then, differential reconfiguration is used to load XPP configuration 2. It operates in the same way on the same RAM. This is repeated for all configurations. Only the last configuration (XPP configuration c) directly streams its output to *outstream*.[2] Table 1 defines the parameters of this system. The *latency* L of the system, i. e. the time between the input of a sample and the output of its processed result, is nearly twice the computation time of an entire sequence of configurations, i. e. $2 \times N/F_S = M/F_S$ in the worst case.

To determine the XPP frequency requirements, let us first compute the number C_N of cycles required to execute all XPP configurations on N samples as follows.

For $c = 1$ (one configuration, after configuration and initialization):

$C_N = N$

For $c > 1$ (several configurations, after initial configuration):

$C_N = c \times (N +$ *pipefill* $+$ *diffconf*$) \approx c \times (N + 2 \times P) = c \times (\frac{M}{2} + 2 \times P)$

As mentioned above in Section 3.2, for $c = 1$ the sample frequency F_S can be as high as the XPP frequency F_X. However, for $c > 1$, F_S must be smaller than F_X since the XPP processor must execute at least C_N cyles while N new input samples arrive. The *computation/configuration ratio* $(N +$ *pipefill* $)/$*diffconf* grows with N. I. e., for large N the number of cycles spent on reconfiguration becomes insignificant compared to the computation cycles. For the processing time of N samples, the following condition must be met:

$$C_N/F_X \leq N/F_S$$

The minimum ratio $R = F_X/F_S$ represents the required XPP performance for a given sample rate. The minimum requirement $C_N/F_X = N/F_S$ implies $R = C_N/N$.

The following formula rewrites the definition of R as a function of the system design

[2]For simplicity, we allow that *outstream* generates bursts of N samples at XPP frequency.

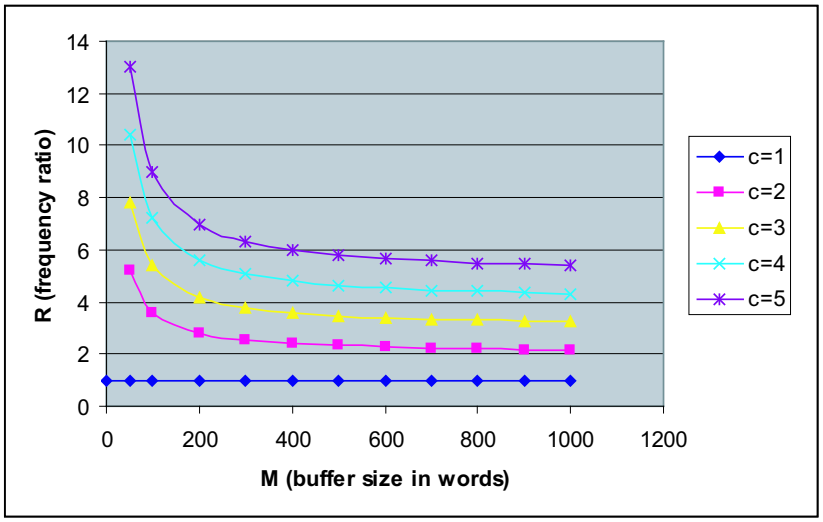

Figure 4: Frequency ratio $R = F_X/F_S$ as a function of M and c for $P = 20$.

parameters c, P and M:

$$R = \frac{C_N}{N} = \frac{c \times (\frac{M}{2} + 2 \times P)}{\frac{M}{2}} = c + \frac{4 \times c \times P}{M}$$

The graph in Figure 4 plots R versus M for the frequently used XPP processor size $P = 20$ and for the numbers of configurations $c = 1..5$. It is assumed that the application uses all ALU-PAEs of the XPP processor in all configurations, i. e. the number of application stages k equals $c \times P$. Obviously, $R > c$, i. e. F_X must be at least c times F_S since the samples are processed c times. The additional reconfiguration overhead (represented by the term $\frac{4 \times c \times P}{M}$) grows with the size P of the XPP processor and decreases with the size M of the buffers in the system. This term leads to the hyperbolic curves in Figure 4. They show that, on the one hand, sufficient buffer memory is required to avoid extreme frequency requirements. On the other hand, the advantage of adding more memory gets very small for large buffer sizes. More buffer memory also increases the latency L of the system. Depending on c, a buffer size between 400 and 800 words is a good choice for this ideal application.

4 Examples

4.1 Differential Reonfiguration for Large FIR Filter

This section compares two implementations of a 48-tap integer FIR filter with a sample frequency of 50 MHz. The first implementation uses one large configuration on an XPP

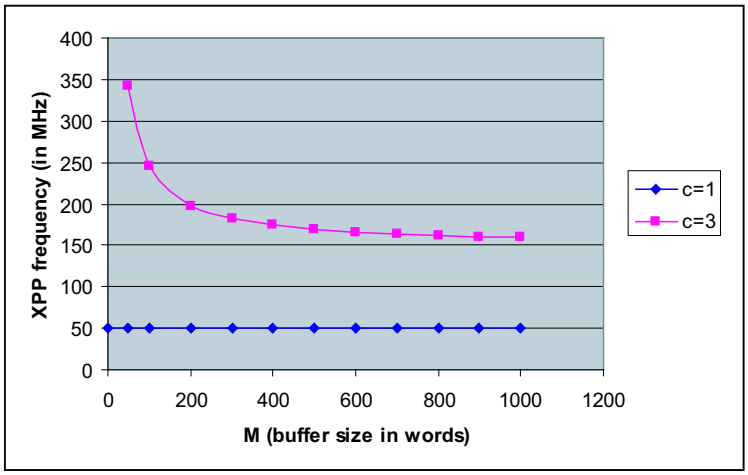

Figure 5: XPP frequency F_X for FIR filter sample frequency $F_S = 50$ MHz as a function of M for the combinations $c = 1/P = 48$ and $c = 3/P = 16$.

processor with 48 ALU-PAEs (system as in Figure 2(a)), and the second implementation employs three differential configurations on a XPP processor with 16ALU-PAEs (system as in Figure 3). Note that a FIR filter meets all requirements of an "ideal" streaming application as defined in Section 3.3. The system parameters are as follows: $F_S = 50$ MHz, $k = 48$, $P = 48$ or $P = 16$, respectively, and $c = 1$ or $c = 3$, respectively. The XPP frequency requirements are as follows:

For $c = 1$: $F_X = F_S = 50$ MHz
For $c = 3$: $F_X = R \times F_S = (c + \frac{4 \times c \times P}{M}) \times F_S = (3 + \frac{192}{M}) \times 50$ MHz

The graph in Figure 5 plots the buffer size/frequency trade-off. It shows that a moderately sized buffer reduces the required XPP frequency substantially.

4.2 Complete Reconfiguration for IQ and IDCT in MPEG4

This section finally discusses an MPEG4 decoder which uses the XPP dataflow array to accelerate the Inverse Quantization (IQ) and Inverse Discrete Cosine Transform (IDCT) kernels.[3] All other computations are performed on the FNC-PAEs. Figure 6 shows the kernels and the data streams. Three XPP processor sizes are considered. They contain 30, 20, or 10 ALU-PAEs, respectively. An IQ implementation (about 10 ALU-PAEs) and a fast IDCT implementation (about 20 ALU-PAEs) both fit on the XPP30 processor. For this system, the assumptions of Section 3.3 hold since the output of the IQ kernel can be directly fed to the IDCT kernel, and both kernels can process

[3]Note that an optimized software library for video processing — comprising IQ, IDCT and all other relevant kernels — is available from PACT.

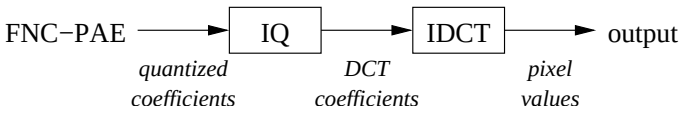

Figure 6: MPEG4 decoder kernels and data streams..

one sample per cycle. In other words, the XPP processor has only to operate at the frequency with which the FNC-PAEs produce and consume samples.

For the smaller XPP processors, IQ is mapped to one configuration and IDCT to another one. Figure 7 shows the related phases and the RAM usage: In phase (a), the quantized coefficients are read from RAM 1, processed by the IQ configuration on the XPP dataflow array, and stored in RAM 2. Both memories have the same size since they both need to store N coefficients. The combined RAM size M depends on the value chosen for N. Then, the dataflow array is reconfigured. In phase (b), the DCT coefficients in RAM 2 are read, processed by the IDCT configuration, and output. Next, the dataflow array is reconfigured, and phase (a) is executed again. Note that the roles of RAM 1 and 2 need not be swapped as in the dual-buffer system in Figure 3. With correct synchronization, the old quantized coefficients are streamed to the XPP processor in phase (a) before they are overwritten by new coefficients. This is the case since the data stream from the FNC-PAE (active in both phases) is slower than the stream from RAM 1 to the dataflow array in phase (a), and since the coefficients are written and read in the same order.

On the XPP20 processor, 10 of the 20 ALU-PAEs are idle during IQ processing. Only the IDCT kernel fully uses the XPP resources. Another difference to the "ideal" application is that differential configuration cannot be used. The kernels have to be con-

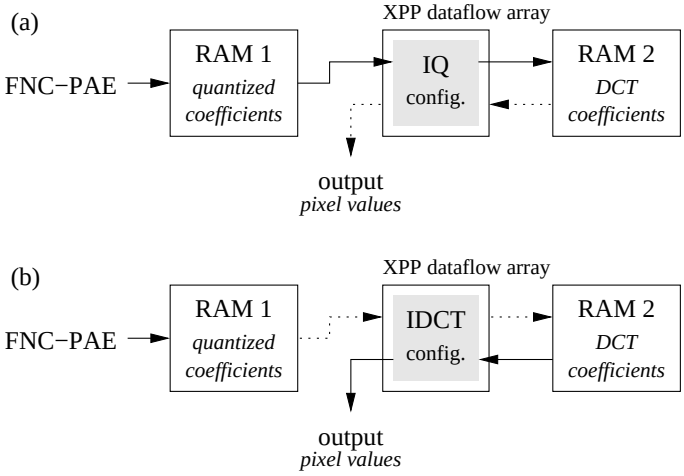

Figure 7: MPEG4 decoding phases: XPP and RAM usage.

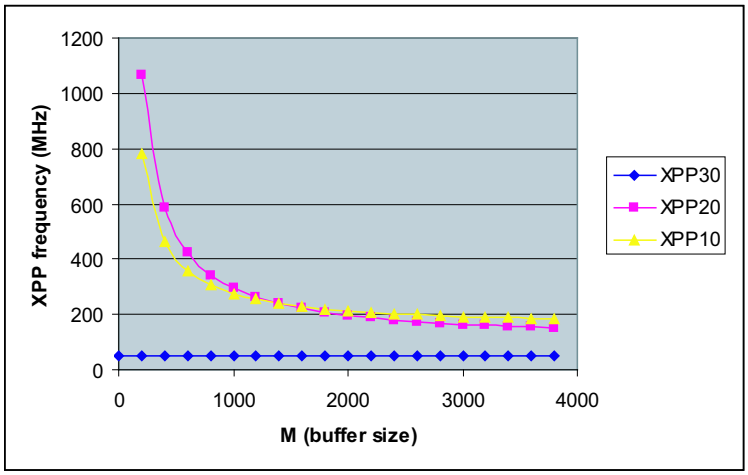

Figure 8: XPP frequency F_X for MPEG4 coefficient sample frequency $F_S = 50$ MHz as a function of M for XPP30 ($c = 1$), XPP20 ($c = 2$) and XPP10 ($c = 2$).

figured completely. The IQ implementation on the 20-ALU XPP dataflow array has 273 configuration words, and the IDCT kernel 1581 words, many of which are used to preload FIFOs and IRAMs. For comparison, the combined configuration has 1903 configuration words. The additional words are due to the connection between the two modules and the longer connections to external ports in the larger XPP dataflow array. As in the "ideal" application, both kernels compute one output sample every cycle. After pipeline filling, the 64 samples of an 8×8 block are computed in 64 cyles.

For this system, the sum of the values of both configurations for *pipefill* (80) and *diffconf* (1854) must be used instead of the term $2 \times c \times P$ to compute F_X for this application. For a sample frequency of 50 MHz (from the FNC-PAE), the following equation holds:

$$F_X = R \times F_S = (2 + \tfrac{3868}{M}) \times 50 \text{ MHz}$$

On the smallest XPP10 processor, a smaller but slower IDCT implementation must be used which requires only 10 ALU-PAEs, but takes 128 cycles to process an 8×8 block. This increases the XPP frequency requirements for this system. The small IDCT configuration has 985 configuration words while the size of the IQ configuration does not change for the smaller XPP processor. Hence the following equations hold:

$$C_N = \Sigma\, \textit{pipefill} + \Sigma\, \textit{diffconf} + 3 \times N = 80 + (985 + 273) + 3 \times N$$

$$F_X = \tfrac{C_N}{N} \times S_N = (3 + \tfrac{2516}{M}) \times 50 \text{ MHz}$$

The graph in Figure 8 plots the buffer size/frequency trade-off for the three XPP processors. It shows that a considerably larger buffer memory is required in this application since the overhead for complete reconfigurations is larger than in the previous example. The area advantage of the smaller XPP processor must be compensated by

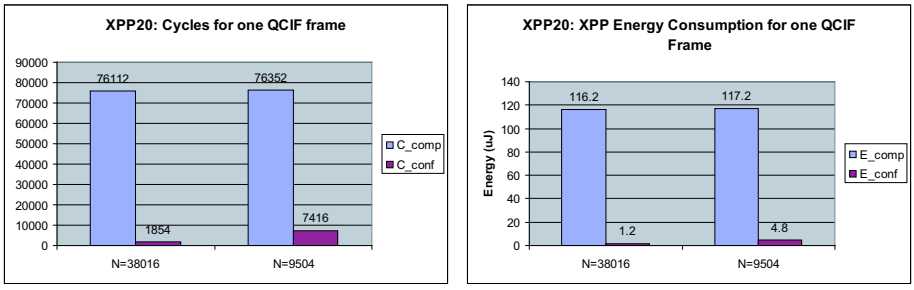

Figure 9: Computation/configuration analysis for XPP20 implemenation.

higher frequencies. Note that the smaller XPP10 system is more efficient than the XPP20 for small buffer sizes ($M < 1352$) with frequent reconfiguration, since the higher IDCT throughput cannot compensate the increased reconfiguration penalty for the larger IDCT implementation.

Finally, the configuration overhead – in terms of execution cycles and energy consumption – is analyzed for the XPP20 implementation. Since processing an entire VGA frame (640×480 pixels) in one configuration requires large buffers and leads to an almost negligible configuration overhead, QCIF frames are used for this analysis instead. Note that processing a QCIF frame is equivalent to processing about one twelfth of a VGA frame.

Two scenarios for the processing of one QCIF frame (176×144 pixels) are considered. With the YUV 4:2:0 format used in MPEG4, a frame has 38,016 samples. In the first scenario (N = 38,016), both configurations process an entire frame between reconfigurations. The second scenario (N = 9,504) processes only a quarter of a frame at a time. Therefore it needs fewer buffer memory, but the reconfiguration overhead is higher. Figure 9 displays the cycles required (derived from the formulas presented in Section 3.3) in the left diagram and the energy consumed (in μJ, derived from the equations in Section 2.4) in the right diagram. The numbers for processing an entire frame in both scenarios are given, separately for compuation and configuration.[4] While both configuration overheads are four times higher in the second scenario, the cycle overhead is still below 10% and the energy overhead below 5%. The increase of the cycle number and energy consumption for the computation phases is only due to repeated pipeline filling and negligible.

Decoding Multiple MPEG Streams

There are two alternatives to decode two independent MPEG streams simultaneously: To decode both streams with full throughput, the hardware (PAEs, RAM) has to be duplicated. Then both streams are decoded independently on their own hardware without

[4]For the computations, the values N_{PAEs}=28, $N_{configuredPAEs}$=24 for IDCT, and $N_{configuredPAEs}$=10 for IQ are used.

interference. On the other hand, if saving hardware is more important than high performance, the two MPEG streams can be multiplexed onto the same hardware. First, a part of the first stream is processed on the given XPP and RAM resources. Then, a part of the second stream is processed on the same resources, etc. Since the XPP dataflow array is reconfigured between the processing phases anyway, there is no additional overhead involved. Similar alternatives exist if more than two MPEG streams are decoded.

5 Summary

This White Paper presented the XPP-III configuration methods as well as system design principles and trade-offs between XPP processor size and frequency, buffer size, reconfiguration frequency, and output latency. Run-time reconfiguration allows the system designer to choose from many implementation options, ranging from extreme high-performance solutions requiring large die sizes to smaller, flexible, and nevertheless powerfull systems.

The design flexibility is only limited by the minimum XPP processor size required for some application kernels and by the reconfiguration time of large configurations. The latter can be amortized by long processing phases between reconfigurations which, however, in turn require larger buffer memories.