AN ASYNCHRONOUS PIPELINE COMPARISONS WITH APPLICATION TO DCT MATRIX-VECTOR MULTIPLICATION *

Sunan Tugsinavisut, Suwicha Jirayucharoensak

University of Southern California
Los Angeles, CA 90089, USA
{tugsinav,jirayuch}@usc.edu

Peter A. Beerel†

Fulcrum Microsystems
Calabasas Hills, CA 91301
pabeerel@fulcrummicro.com

ABSTRACT

This paper presents comprehensive energy-throughput comparisons of two well-known asynchronous design styles applied to a matrix-vector multiplication core of the discrete cosine transforms (DCT). The first design style, bundled-data pipelines [1], uses a single-rail synchronous datapath with recently proposed true-four-phase controllers integrated with data-dependent delay lines. The design achieves reasonably-high average performance and very low energy but requires significant design effort to verify the two-sided timing constraints (set-up and hold) typical of bundled-data pipelines. The second design style, 2-D QDI pipelines [2], consists of a network of small communicating cells communicating through delay-insensitive 1-of-N encoded channels. Compared to the bundled-data counterpart, transistor-level simulations show that all QDI designs achieve higher throughput at the cost of larger area and energy and in particular have 22% better $E\tau^2$ metric. In addition, the QDI designs require less design effort than the bundled-data counterpart, because they require virtually no timing verification.

1. INTRODUCTION

The DCT is an essential task in several data compression and decompression applications, including both JPEG and MPEG [3]. With the simultaneous increase in demand for faster data rates and longer battery-lifetimes in portable multimedia devices, low-power and high-performance implementations of the DCT are increasingly important.

Asynchronous design is becoming increasingly commercially viable. They often facilitate low-power for a given performance via a combination of enabling automatic power down of unused components, removing the need for power-hungry clock distribution networks, and providing the flexibility to optimize for the common case [4]. Most asynchronous design styles also offer reduction of noise immunity and good design modularity [4, 2].

In this paper, we focus on two well-known asynchronous design styles. The first, bundled-data pipelines or micropipelines [1], uses a single-rail synchronous datapath with asynchronous controllers driving novel speculative delay lines [5], yielding low area, good average performance, and low power. The cost of this design style, however, is the significant increase in effort and risk associated with verifying all the setup and hold constraints typical of bundled-data design. The second design style, quasi-delay-insensitive (QDI) fine-grain 2-D pipelines, has the advantages of robustness and high throughput. In this design style, large functional blocks are decomposed into small communicating cells communicating through asynchronous channels. The cells are implemented using the QDI design style which means they will work correctly regardless of wire delays except for very loose timing assumptions on some internal wire forks [6]. The asynchronous channels use 1-of-N rail signalling providing delay-insensitive communication. The cells are arranged in a so-called 2-D pipeline that facilitates very high-throughput independent of the width of the datapath [2]. The costs associate with this design style compared to other asynchronous styles is generally more area and higher absolute power consumption.

To quantify the energy-throughput tradeoff between these styles, we first implement bundled-data pipeline optimized for average performance and low-power. In addition, four QDI fine-grain pipelines are implemented with a variety of different micro-architectures that explore the impact of both block and bit-skewed datapaths along with the effectiveness of loop-unrolling. In all four cases, a minimal number of extra pipeline buffers (also called slack optimization [2, 7]) were added to minimize pipeline stalls [8].

Transistor-level simulation results indicate that the most efficient QDI design has 3 times higher throughput but consumes 8 times more worst-case energy than the bundled-data pipeline. In particular, the metric $E\tau^2$ indicates that this pipeline yields 22% better $E\tau^2$ than the bundled-data counterpart. This demonstrates that for at least this application, the QDI design style can provide higher performance, lower design effort, and improved energy for a given performance.

This paper is organized as follows. Section 2 provides relevant background in asynchronous design and defines the required matrix-vector multiplication for the DCT. Section 3 describes our implementation of bundled-data pipeline design. It is followed by a description of the four QDI pipelines in Section 4. Section 5 discusses the energy-throughput statistics of each design. Finally, our conclusions are presented in Section 6.

2. BACKGROUND

An asynchronous circuit typically consists of a set of functional components that locally communicate using a set of handshaking protocols across channels. A plethora of asynchronous design styles exist which vary the size of functional components, the parallelism in the handshaking protocols, the data encoding across the channels, and the degree of timing assumptions needed to ensure correctness.

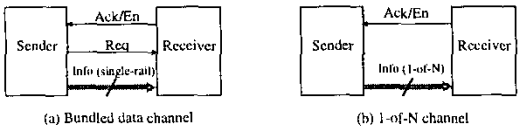
2.1. Channels: bundled-data vs 1-of-N rail

A communication channel is a bundle of wires between a sender and receiver and a protocol for communicating information discretized into tokens (representing data, control, or a mixture) from the sender to the receiver. In a bundled-data channel, as illustrated in Fig. 1(a), token values are encoded using one wire per bit of information to be sent, a request line (Req) is used to tell the receiver when the bit are valid, thus representing a token, and an acknowledge line (Ack) is used to tell the sender when the token has been received. (The data

*This work was supported by a large-scale NSF ITR Award No. CCR-00-86036.

†He is on the leave of absence from the University of Southern California.

0-7803-7761-3/03/$17.00 ©2003 IEEE



(a) Bundled data channel (b) 1-of-N channel

Figure 1: Two types of asynchronous channels.

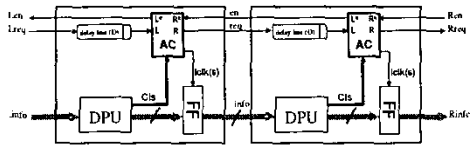


Figure 2: Bundled-data linear pipelines.

is bundled with the request line.) Alternatively, in a 1-of-N (or one-hot) channel, as illustrated in Fig. 1(b), N wires are used to encode $\log_2(N)$ bits and no request line is needed. In particular, a widely used form of 1-of-N encoding is 1-of-2 (also called dual-rail) encoding in which two wires are used to encode one bit of information. In 1-of-N encoding, the validity of the data is encoded in the values of the N wires; all zeros indicate that the bundle of wires is reset and holds no token.

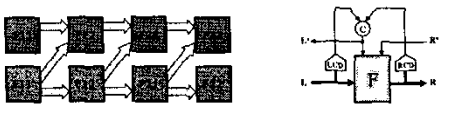
Both two and four-phase handshaking protocols exist across communication channels. In this paper, we restrict ourselves to four-phase protocols in which after the sender sends the data (the first phase), the acknowledgement is asserted by the receiver (the second phase). Next, the sender resets the data (the third phase), followed by the receiver resetting the acknowledgement (the fourth phase). If the acknowledgement is active low, it is often referred to as an enable (En).

2.2. Bundled-data linear pipelines

A bundled-data linear pipeline stage operates on a left and right bundled-data channels, as illustrated in Fig. 2. It consists of a standard synchronous datapath (DPU) in which a combination of a delay line and asynchronous control circuit (AC) controls an output flip-flop (FF). The setup and hold requirements on the flip-flop are often called *bundling constraints*. Additional setup and hold requirements on the conditional inputs (CIs) from the datapath to the asynchronous control may also exist. Each controller is responsible for triggering the associated FFs via the local clock (*lclk*) and generating an output control token to communicate with the next pipeline stage. Note that the use of more efficient latches instead of flip-flops is also possible at the cost of satisfying more stringent bundling constraints.

2.3. 1-of-N QDI fine-grain pipelines

A QDI communicating cell shown in Fig. 3(a) can be implemented with various circuit templates depending on the protocol being selected. One common template is the Pre-Charged Half Buffer (PCHB) shown in Fig. 3(b) in which F is a functional block comprised of controlled dynamic gates followed by inverter drivers. The LCD and RCD are left and right completion sensing circuits respectively. Each



(a) 2-D pipelining (b) PCHB pipeline template

Figure 3: 2-D QDI fine-grain pipelines.

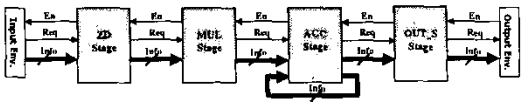


Figure 4: Matrix multiplier using a 4-stage bundled-data pipeline.

cell communicates with left and right environments using PCHB four-phase handshaking protocol [2]. The detailed implementation and other template alternatives can be found in [2].

The functional decomposition into small communicating cells and the 2-D communication among them, shown in Fig. 3(a), is the key to achieving high throughput. This array configuration enables the completion sensing logic in a cell to operate only on a few bits, thereby reducing its contribution to the cycle time. Moreover, because the forward latency path through each cell is domino logic with no related setup and hold timing assumptions, the 2-D array yields very low latency.

2.4. Non-linear pipelines

Complex system design inevitably involves non-linear behaviors such as forks and joins [1, 2]. It is therefore important that any pipeline template can handle multiple inputs and outputs as well as conditional reading and writing of tokens. Such non-linear pipeline templates for the QDI design were presented in [2]. For bundled-data pipeline, alternative control circuit templates were proposed in [5].

2.5. Matrix-Vector multiplication

Our matrix-vector multiplier is iterative. In each iteration, it performs four multiply-accumulate computations on constant coefficients and input vectors X in order to generate one output vector Y. In particular, the calculation is shown below where a, c, and f are constant matrix coefficients.[1]

$$\begin{bmatrix} y0 \\ y1 \\ y2 \\ y3 \end{bmatrix} = \begin{bmatrix} a & a & a & a \\ c & f & -f & -c \\ a & -a & -a & a \\ f & -c & c & -f \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix}$$

$$= \begin{bmatrix} (a*x0)+(a*x1)+(a*x2)+(a*x3) \\ (c*x0)+(f*x1)-(f*x2)-(c*x3) \\ (a*x0)-(a*x1)-(a*x2)+(a*x3) \\ (f*x0)-(c*x1)+(c*x2)-(f*x3) \end{bmatrix}$$

3. BUNDLED-DATA PIPELINE DESIGN

The dominance of zero and small-valued inputs of the DCT [9] motivates the design of a pipeline that exhibits average performance by optimizing the pipeline for small numbers. The key idea is to partition the input into several groups of bit-slices and use speculative delay lines to match the delay of each group. This yields functional units (i.e. multipliers and accumulators) that dynamically operate on different bit-widths. In addition, we adopted the True 4-Phase Full Buffer (T4PFB) templates presented in [5] to implement the control circuits that control the datapath's behavior. This control template coupled with the single-rail datapath and speculative delay line achieves good average performance with low average and worst-case energy.

Our architecture consists of a 4-stage bundled-data pipeline shown in Fig. 4. The first stage simply detects whether the input is zero. The second and third stages make up the multiply-accumulate logic containing three multipliers and four accumulators respectively. These

[1] $a = 2^{-2} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-9} \approx 0.35$, $c = 2^{-1} + 2^{-5} + 2^{-7} + 2^{-10} \approx 0.46$ and $f = 2^{-3} + 2^{-4} + 2^{-8} + 2^{-14} \approx 0.19$

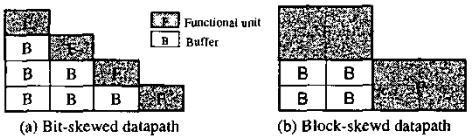


Figure 5: Skewed pipelines.

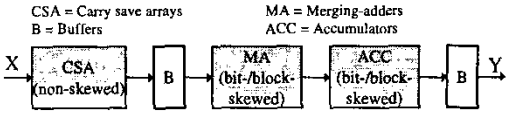


Figure 6: Basic QDI pipeline design.

two stages dynamically operate on different input bit-widths controlled by bit-slice activation control signals referred to as *mask* signals. In the multiplier stage, an input is multiplied with three constant matrix coefficients a, c, and f. The accumulator stage then adds/subtracts the results from multipliers with the previous accumulated results. After the fourth computation, the accumulator's result is latched in the output pipeline stage.

Experimental results demonstrate that this architecture yields significantly better $E\tau^2$ metric than the comparable synchronous design, particularly because it significantly reduces average cycle time τ with negligible energy overhead [5].

4. QDI PIPELINE DESIGN

4.1. Micro-architecture alternatives

Before describing the four QDI micro-architectures quantified, this section reviews several design dimensions upon which a QDI micro-architecture may vary.

4.1.1. Delay-insensitive encoding selection

The most common encodings for QDI designs are 1-of-2 (*dual-rail*) and 1-of-4 (*quad-rail*) encodings. To transmit 2 bits of data, quad-rail encoding uses five wires and 4 transitions while two dual-rail encoded channels would require 8 transitions with six wires. Quad-rail encoding is thus more energy and area efficient when transmitting multiple bits at a time. Higher order encodings are even more energy efficient than quad-rail encoding but incur an exponential hit in area efficiency.

4.1.2. Cell size choice

Asynchronous cells can operate on individual bits or groups of bits independent of the encoding. For example, a cell operating on 4 bits, often called a *nibble*, may have data inputs transmitted as four dual-rail channels or two quad-rail channels. In both cases, the cells must handle completion sensing of multiple 4-bit quantities which will likely limit its minimum cycle time. In this paper, we focused on micro-architectures that use cells to operate on one-bit input token, enabling higher throughput than otherwise possible at the cost of more area.

4.1.3. Skewed datapath choice

A skewed asynchronous datapath is necessary in 2-D pipelines when a wide datapath is decomposed into small functional units (i.e. a one-bit adder) and a non-skewed structure would require vertical communication. For example, in a ripple carry adder, a non-skewed structure would require the carry bit to ripple downwards. This downwards ripple would stall higher-order bits increasing the overall cycle time. One solution, as shown in Fig. 5(a), is a skewed pipeline, in

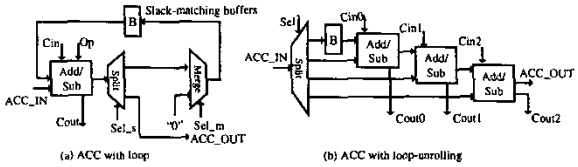


Figure 7: Loop and complete loop-unrolled designs of the ACC block.

which the vertical communication is converted into diagonal communication. Extra buffers are used to store higher bits until they are needed thereby removing such stall conditions and maximizing system throughput in a process called *pipeline optimization or slack matching* [2, 7].

These buffers are not necessary between functional blocks that are both skewed but are necessary at the boundary between a non-skewed and skewed blocks or when the most significant bit of one block is needed in the computation of the least significant bit of the subsequent block (e.g., in the case that the comparison result is used to determine a subsequent token routing).

As shown in Fig. 5, we consider two types of skewed datapaths, *bit-skewed* datapath and *block-skewed* datapath. Block-skewed datapaths align logic cells within each block vertically but skew the blocks. The block size can vary from several bits to the width of the entire datapath (creating at one extreme a *non-skewed* datapath). Block-skewed datapaths require fewer slack-matching buffers than their bit-skewed alternatives but are only possible when vertical communication within the blocks is not necessary or can be avoided by appropriate choice of functional unit architectures. For example, to facilitate block-skewed datapaths of our accumulators, we adopted a Brent-Kung-based tree adder architecture for each block that does not require vertical communication within the block.

At first glance, it may appear that the number of slack-matching buffers required is equal to the number of skewed stages. However, since buffer cells have a cycle time faster than other logic cells, fewer slack-matching buffers are actually necessary to achieve maximum system throughput [8]. In our designs, buffer cells have a cycle time of 10 gate delays and logic cells have a cycle time of 14 gate delays, the number of slack-matching buffers required is half the number of skewed stages.

4.1.4. Complete loop-unrolling

An algorithmic loop within the architecture can sometimes limit the system throughput by creating a pipeline stall. Also, small loops within the architecture require additional pipeline buffers to prevent pipeline stalls associated with the delay of bubbles moving backwards through the pipeline [8]. Both of these issues can be avoided for algorithmic loops with a deterministic number of iterations through a technique we call *complete loop-unrolling*. Complete loop-unrolling involves implementing the iterative algorithm in a linear structure at the cost of replicating the functional units multiple times.

4.2. Basic QDI pipeline architectures

Our basic QDI architecture is decomposed into three major functional blocks, as depicted in Fig. 6. The first block contains three hard-wired carry save arrays (CSA) multipliers. The second block is the merging-adders (MA) adding the results from the CSA. The third block has four accumulators (ACC).

We assume that inputs X and outputs Y arrive in a non-skewed fashion. The CSA multipliers are designed using non-skewed datapath because they require the fewest slack-matching buffers. The merging adders and accumulators are skewed, either bit-skewed or

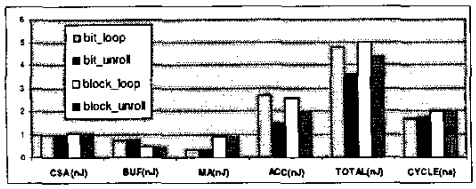


Figure 8: Energy and cycle time statistics of the 4 QDI designs

block-skewed with 8-bit chunks. Slack-matching buffers (B) are also inserted between the output of CSA (non-skewed) and the input of MA (skewed) and between the output of ACC (skewed) and the input of the right environment (assuming it is non-skewed).

4.3. QDI pipeline micro-architectural alternatives

Among many combinations of micro-architecture choices discussed early, we implemented four different QDI designs.

The first two designs, referred to as QDI-bit-loop and QDI-bit-unroll, are bit-skewed designs in which the ACC block is implemented with and without complete loop unrolling, as depicted in Figure 7. Note that the complete loop-unrolled version consists of three adder/subtractor blocks that implement the 3 different add/subtract operations. In both cases the channels are implemented with dual-rail encoding and the cells operate on bit-wide operands.

The third and fourth designs, referred to as QDI-block-loop and QDI-block-unroll, both use 8-bit block-skewed datapaths to reduce the number of extra buffers and use quad-rail encoding to reduce energy consumption and vary in whether or not the ACC block is unrolled. The block-skewed datapath are applied to the MA and ACC blocks shown in Fig. 6. The quad-rail 8-bit block-skewed adders were implemented using a Brent-Kung carry-look-ahead structure. This adder has 4 quad-rail input channel (8 data bits) and has 4 quad-rail sum output channels and one dual-rail carry output channel. We construct our 22-bit adders by connecting three such adder blocks together in a ripple carry fashion.

5. SIMULATION RESULTS

We implemented all five designs at the transistor-level using manual transistor sizing in TSMC's CMOS 0.25μ process. We simulated each design using Nanosim at a nominal 25°C environment and at the 2.5 V nominal supply voltage. We setup four simulations for the QDI designs and calculate average cycle time and energy per computation for 20 random input vectors. The detailed results of each decomposed blocks are shown in Fig. 8. The results suggest that required pipeline buffers in the block-skewed designs accounts for 40% less energy than those in the bit-skewed designs. However, since these extra buffers are a small portion of the whole design and the block-based designs are more complex, the overall energy consumption of the bit-skewed designs are still lower. Moreover, the data shows that bit-skewed pipelines operates faster than the block-skewed pipeline primarily because the logic cells in bit-skewed pipeline are smaller. Finally, the loop-unrolling micro-architecture reduces the total energy by 25% for the bit-skewed design and 12% for the block-skewed design by eliminating the need for many slack-matching buffers in the ACC block.

We obtain average performance of our bundled-data design by applying two sets of 20 random input vectors, one set for a lower bound and another set for an upper bound of average performance. Both input sets imitate input statistics from [9]. Since our QDI designs do not attempt to take advantage of energy saving from input statistics [10], to obtain a more reasonable comparison, we measured the worst-case energy of the bundled-data design in which all bit-sliced are active. Table 1 shows average cycle time and energy per computation. We then combined this data in the last column where all designs are compared using the $E\tau^2$ metric. Compared to the bundled-data design, the results indicate that the best QDI design is QDI-bit-unroll yielding 22% better $E\tau^2$, followed by the QDI-bit-loop which yields a 4% better $E\tau^2$. The other two QDI designs had inferior $E\tau^2$ metrics.

Designs	Area (Ktrs)	Throughput (Mhz)	E/cyc (nJ)	τ (ns)	$E\tau^2$ ($nJ * ns^2$)
Bundled-data	26	184	0.48	5.43	14.15
QDI-bit-loop	125	595	4.83	1.68	13.63
QDI-bit-unroll	137	571	3.62	1.75	11.08
QDI-block-loop	123	510	4.92	1.96	18.90
QDI-block-unroll	200	510	4.31	1.96	16.55

Table 1: Area, cycle time, energy per cycle and $E\tau^2$ statistics

6. CONCLUSIONS

This paper presents energy-throughput comparisons of two well-known asynchronous design styles (bundled-data pipeline vs 2-D QDI fine-grain pipeline) applied to a matrix-vector multiplication core of the DCT. We implemented and compared bundled-data design and four QDI designs of different micro-architecture choices considering bit/block skewed datapath and the impact of loop-unrolled. The simulation results suggest that the best QDI design is the bit-skewed QDI design with loop-unrolling. It yields significantly better worst-case performance than the average-case performance of the bundled-data design and has a 22% better $E\tau^2$. This demonstrates that for at least this application, the QDI design style can provide higher performance, lower design effort, and improved energy for a given performance.

All our QDI designs, however, do not take advantage of DCT input statistics. Future work should include comparison with a data-dependent architecture called width adaptive datapaths (WAD) [10] that dynamically generates and consumes tokens based on input data, potentially achieving lower average energy for this application.

7. REFERENCES

[1] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[2] A. Lines, "Pipelined asynchronous circuits," Technical Report CS-TR-95-21, California Institute of Technology, June, 1998.

[3] K. Rao and P. Yip, *Discrete Cosine Transform, Algorithm, Advantages, Applications*, Academic Press, 1990.

[4] S. M. Nowick, M. B. Josephs, and C. H. K. Berkel, "Scanning the issue: Special issue on asynchronous circuits and systems," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 219–222, Feb. 1999.

[5] S. Tugsinavisut, Y. Hong, D. Kim, K. Kim, and P.A. Beerel, "Efficient asynchronous bundled-data pipelines for dct matrix-vector multiplication," *submitted to Tran. on VLSI*, Sept. 2002.

[6] A. J. Martin, "The limitations to delay-insensitivity in asynchronous circuits," in *ARVLSI*, 1990, pp. 263–278.

[7] S. Kim and P.A. Beerel, "Pipeline optimization for asynchronous circuits: Complexity analysis and an efficient optimal algorithm," in *ICCAD*, Nov. 2000.

[8] J. Teifel, D. Fang, D. Biermann, C. Kelly, and R. Manohar, "Energy-efficient pipelines," in *ASYNC*, Apr. 2002, pp. 23–33.

[9] K. Kim, P. A. Beerel, and Y. Hong, "An asynchronous matrix-vector multiplier for discrete cosine transform," in *ISLPED*, July 2000, pp. 256–261.

[10] R. Manohar, "Width-adaptive data word architectures," in *ARVLSI*, Mar. 2001, pp. 112–129.