# VARIABLE DELAY RIPPLE CARRY ADDER WITH CARRY CHAIN INTERRUPT DETECTION

*Andreas P. Burg, Frank K. Gürkaynak, Hubert Kaeslin, and Wolfgang Fichtner*

Integrated Systems Laboratory, ETH-Zurich

## ABSTRACT

Various implementations are known for the efficient implementation of adders. As opposed to traditional optimization techniques a statistical approach using early termination detection is used in this article to obtain efficient implementations for large operands. The completion detection logic is described and the efficiency of the approach is shown and analyzed analytically and through computer simulations. The technique is based on area and routing efficient ripple carry adders, which is are especially desirable properties for very long word length.

## 1. INTRODUCTION

Addition is one of the fundamental and most essential functions used in VLSI design. Finding fast and efficient implementations for this operation is therefore one of the major challenges. A detailed overview on the various adder architectures and their comparison is given in [1]. All presently known adder architectures suffer from a so called "curse of the carry": For all adder architectures that implement an $n$ bit binary addition, there exists a set of inputs $A$ and $B$, where the MSB bit of the output $S_n$ depends on the LSB of the inputs: $a_0$ and $b_0$. In other words, the worst case timing of a binary adder architecture is strongly related to the carry propagation path.

Using standard synchronous design techniques, the adder circuit must be designed with the worst case timing constraints in mind. While there will always be a set of inputs that will stimulate the longest path of a given adder, it can be shown that under a normalized distribution of its inputs, such occurrences are extremely rare. The probability that the longest uninterrupted carry run $l$ is equal or longer than $R$ stages in an adder with $B$ bits is only $P(l \geq R) = B * (1/2)^{R+1}$. As an example for a 16-bit standard ripple carry adder (RCA) architecture only 3% of all inputs will result in a carry propagation chain of length 8 or more. Especially with very wide operand lengths $B > 64$ this results in an on average overly pessimistic timing constraint for the adder operation.

As opposed to synchronous design, self-timed design utilizes functional modules that are capable of signaling the completion of their operation. Completion detection for self timed adder architectures is a well researched topic [2, 5, 4, 3]. These implementations are mainly based on detecting the completion of the carry-propagation with the help of additional circuitry.

In the synchronous domain, speculative execution based methods have been used [6]. Such methods usually employ a two-cycle operation. The addition is started in the first cycle, and the result is assumed to be correct. A parallel carry propagation network checks whether or not the operation has long carry paths. Should this be the case, the system is stalled for the duration of an additional clock cycle where the original addition operation has sufficient time to finish execution.

We present a method to design large operand adders with variable latency, suitable to use in Globally-Asynchronous, Locally-Synchronous (GALS) modules as described in [7] or in any other synchronous designs. We define a general form where an $n$-bit binary adder is divided into $D$ divisions of equal length. A partial carry propagation detection of length $Q$ is performed on the boundary of all but the last of these divisions. The cumulative result of all detections is used to select between $D$ different latency values for the execution. While the method is similar to the one described in [2] for fully asynchronous designs, we use a more general approach applicable to a standard synchronous design methodology. We also analyze the influence of the number of divisions and of the detection circuit depth on the delay and the efficiency ($AT - product$) of the circuit and give an approximate analytical expression for the performance gain.

## 2. EARLY COMPLETION DETECTION

### 2.1. Carry Propagation

A simple full adder is shown in Figure 1. Its worst-case delay is determined by the ripple carry chain and therefore by the time a carry generated at the LSB full-adder needs to propagate to the MSB. The carry propagation in each of the full adders is described as:

$$c_n = a_n b_n + a_n c_{n-1} + b_n c_{n-1} \qquad (1)$$

It is obvious that $c_n$ only depends on $c_{n-1}$when *only one* of the inputs $a_n$or $b_n$ is set. When both are zero the carry propagation will be stopped at the $n$th Full Adder stage, while a carry-out is always generated directly from $a_n$and $b_n$when both are one. Consequently the carry chain is interrupted and propagation in the subsequent stages can start immediately without having to wait for the result of the preceding stages. The operation of the independent partial carry chains is hereby fully parallel. If only a single pair of bits is considered this favorable event has a chance of occurrence of 50%.

To improve the carry chain interrupt detection (CCID) probability a group of bits $a_{n+m}, b_{n+m}$ with $m = 0..C-1$ can be considered. By computing the partial sum of only the operands in

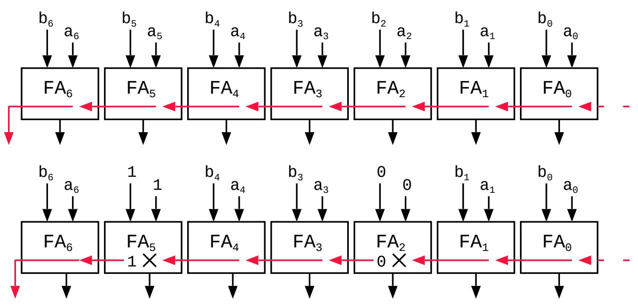
b6 a6 b5 a5 b4 a4 b3 a3 b2 a2 b1 a1 b0 a0
FA6 FA5 FA4 FA3 FA2 FA1 FA0
b6 a6 1 1 b4 a4 b3 a3 0 0 b1 a1 b0 a0
1
0

Figure 1: 2 Operand Ripple Carry Adder

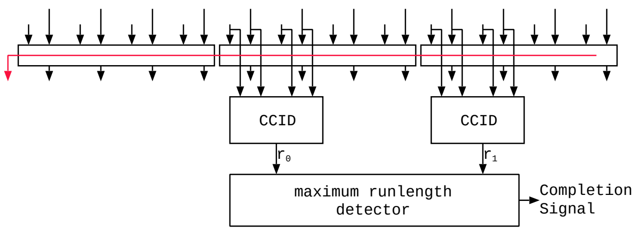

Figure 2: Adder Partitioning

the GOB: $S_n = \{a_n...a_{n+C-1}\} + \{a_n...a_{n+C-1}\}$ a decision can be made whether the carry out in the full adder chain at $FA_{n+C-1}$ will depend on the carry in of $FA_n$ or not:

$$CarryOut_{n+C-1} = \begin{cases} f(CarryIn_n) & ,S_n = 2^C - 1 \\ 0 & ,S_n < 2^C - 1 \\ 1 & ,S_n > 2^C - 1 \end{cases} \tag{2}$$

The probability that the carry at the output of a PA is known a priori has now increased to $1 - \frac{1}{2^C}$.

### 2.2. Partitioning

To use this efficiently the complete $L - Bit$ adder is subdivided into $D$ partial adders (PA) of which each is $Q = \frac{L}{D}$ bits wide. A logic to detect an interruption in the carry propagation chain is placed at the last $C$ bits of all but the last groups as shown in Figure 2. They will reliably detect if the carry-out of the partial adder $d$ is constant ($r_d = 1$) or if it depends on the preceding carry chain ($r_d = 0$).

Herewith an upper bound for the time needed to assure that all outputs of the adder have settled for the given set of inputs can be derived. It is determined by the maximum number of subsequent PAs that will propagate the carry which is given by the longest run of subsequent zeros in $r_{0...C-2}$: $M_{RL}$ and by the delay of a partial adder $T_{PA}$.

## 3. ANALYSIS AND OPTIMIZATION

### 3.1. Model

A mostly technology independent estimate of the efficiency of the proposed approach is obtained through an analysis of the expected

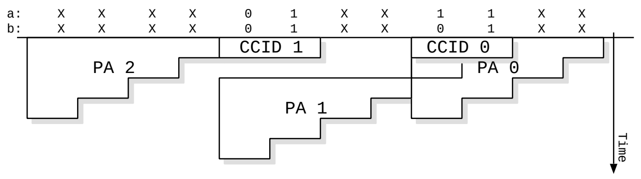

Figure 3: Timing ($M_{RL} = 1$)

average delay gain and the $AT - Product$. The optimization parameters are the number of partial adders $D$ and the number of bits used in the CCID: $C$. While in principle any adder architecture can be chosen for the PAs our analysis focuses on the implementation using ripple-carry architectures as they have the smallest area and excellent regularity/routability and are therefore the most interesting for extremely large operands. Furthermore the following reasonable assumptions are made:

1. The inputs $a$ and $b$ are normally distributed in $\left[0...2^B - 1\right]$
2. Delay and area for an $n$-bit addition using ripple-carry architecture are chosen according to [1]: $T_{Add}(n) \approx 2n$, $A_{Add}(n) \approx 7n$
3. Time and area for adding two $n$ bit numbers and checking them for $2^n - 1$ in the CCID is assumed to be $T_{CCID}(n) \approx 1 + log_2(n)$ and $A_{CCID}(n) \approx 3 * n$
4. The time and area required to find the maximum runlength $M_{RL}$ are negligible

Herewith outputs of a all PAs whos preceding CCID indicates no carry propagation will settle after a carry that has potentially been generated within the range of the preceding CCID has been propagated and after the delay of the partial adder itself:

$$T_{Add}(C) + T_{Add}(Q) \tag{3}$$

The remaining PAs add an additional delay of $M_{RL} * T_{Add}(Q)$ as shown in figure 3 and the overall expected delay becomes

$$T = T_{Add}(C) + T_{Add}\left(\frac{L}{D}\right) * \left(1 + \underbrace{\sum_{t=0}^{D-1} t * P(M_{RL} = t)}_{E[M_{RL}]}\right) \tag{4}$$

As the delay of the CCID is significantly smaller than the propagation delay through each of the subadders it does not contribute to the critical path. This is shown in Figure 3.

When the principle is applied to a synchronous circuit the variable delay of the adder translates into a variable number of cycles between 1 and D, required to complete the operation. The cycle time is lower bounded by Equation 3 and a slightly degraded performance characteristic is obtained:

$$T = \left(T_{Add}(C) + T_{Add}\left(\frac{L}{D}\right)\right) * (E[M_{RL}] + 1) \tag{5}$$

The analytical evaluation of the above equation requires an expression for the expected value for the longest run of carry propagations across PA boundaries $E[M_{RL}]$. This is known as the “longest run of heads” problem. In our case “head” in an unfair coin-tossing experiment with $D-1$ trials represents the case where a possible carry propagation is detected by a CCID. The corresponding probability for this event is has been shown to be $\frac{1}{2^C}$. To our knowledge an exact analytical solution for the problem is not known. However a pessimistic approximation can be given as:

$$E[M_{RL}] = \frac{ln\left((D-1)*\left(1-\frac{1}{2^C}\right)\right)+0.57721}{C*ln(2)} \qquad (6)$$

Figure 4 plots the expected number of cycles versus the normalized cycle time for a 128-bit ripple-carry CCID adder on a logarithmic scale. The former was evaluated using the approximation from Equation 6 and through computer simulations. For comparison the leftmost diagonal represents the delay of an equivalent standard fixed delay ripple-carry implementation. Five different configurations for the CCID-adder were analyzed in which the full length of the operation was divided into 2, 4, 8, 16 and 32 partial adders. An almost 2 fold speedup was achieved with only two stages. Each additional stage reduces the total delay further, resulting in a more than three and five fold gain for four and eight partial adder stages respectively. For each configuration the prediction parameter $C$ of the CCID stage was varied between 2 and the total number of bits per subdivision. From the figure it can be seen that increasing $C$ initially improves the performance significantly as the probability to detect an early termination increases asymptotically to one. However when more than 5 to 6 bits are used the linearly increasing delay $T_{Add}(C)$ prevails and the delay of the overall circuit increases again. This is especially pronounced in a synchronous environment where Equation 5 applies and $T_{Add}(C)$ has a more significant influence on the overall timing than in the asynchronous case. The analytical approximation matches the simulation only qualitatively, however the general trend can still be observed and the model might be used to derive an estimate for an appropriate choice of the parameter $D$ to obtain a desired performance gain. Thereto 5 is substituted into 6. This estimate is an upper bound and therefore the minimum delay with respect to the parameter $C$ is obtained first from the delay equation, considering $L$ and $D$ as constants:

$$T = 2\left(\frac{ln\left((D-1)\left(1-\frac{1}{2^C}\right)\right)+0.57721}{ln(2)} + C + \frac{ln\left((D-1)*\left(1-\frac{1}{2^C}\right)\right)+0.57721}{C*ln(2)}\frac{L}{D} + \frac{L}{D}\right) \qquad (7)$$

After removing constant factors and offsets [1] and after approximation of the logarithmic functions as $ln(D-1)$ the first term also becomes independent of $C$ and does not contribute to the optimization. Deriving the result by $C$ and setting it to zero yields the following expression for the near optimum choice of $C$ :

[1]As the absolute delay is not of interest this is a perfectly legal simplifi cation

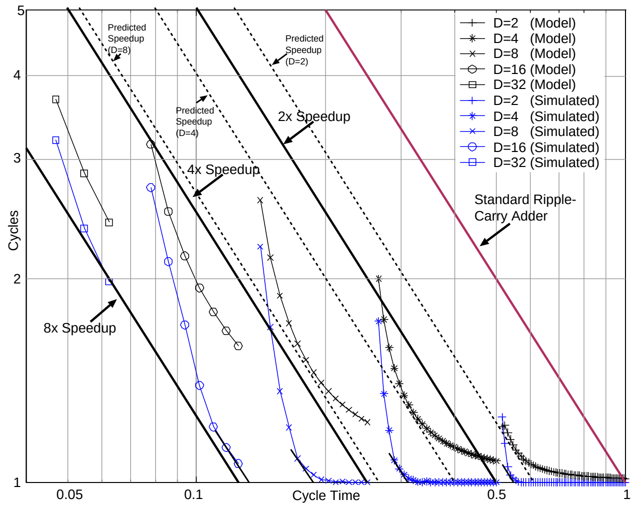

Figure 4: Timing of the CCID-adder

$$C_{opt}(L,D) = \sqrt{\frac{ln(D-1)+0.57721}{ln(2)}\frac{L}{D}} \qquad (8)$$

This result is substituted into the equation for the delay, again with the approximation for the logarithmic terms. After dividing by the delay of the original ripple-carry adder without CCID ($2L$) and after removing the insignificant terms for large $L$ this leads to an equation for a pessimistic estimate of the expected performance gain:

$$\frac{T_{CCID}}{T_{RCA}} = 2\sqrt{\frac{ln(D-1)+0.57721}{ln(2)}}\sqrt{\frac{1}{LD}} + \frac{1}{D} \qquad (9)$$

The complexity of the early termination adder grows slowly over the regular ripple carry adder with the number of PAs and with the number of bits used in the CCID stages. It is described as:

$$A = A_{Add}(L) + (D-1)*A_{CCID}(C) \qquad (10)$$

Figure 5 shows the normalized AT-diagram of the circuit. Because of the strong influence of the number of bits in the CCID-stage ($C$) on the timing when $C$ is small the reduction of the delay initially clearly outweighs the area increase. This is reflected in a significant gain in the AT-product. It is also observed that the point with the shortest average delay is close to the optimum AT-product which is a very desirable property. The original fixed-delay implementation is marked with a circle for comparison.

## 4. EXPERIMENTAL RESULTS

In this section actual implementation results are shown. Various realizations for a 128 and a 256 bit wide adder were implemented

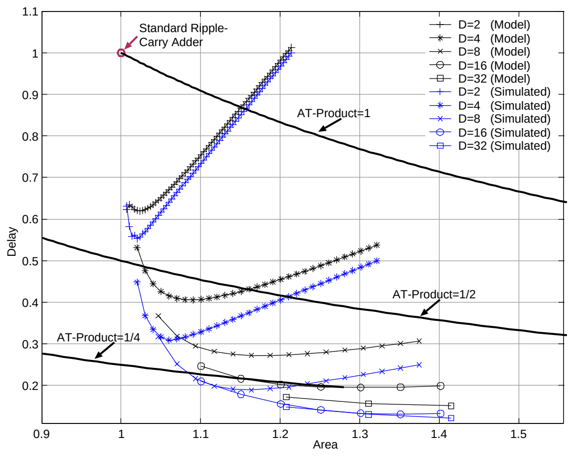

Figure 5: AT-Product of the CCID-adder

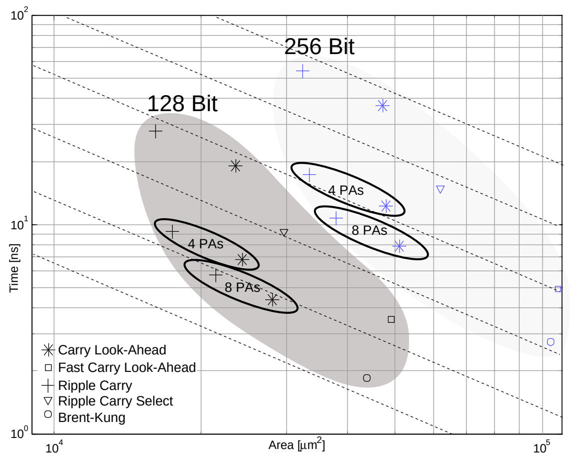

Figure 6: Implementation (AT-product diagram)

Table 1: Area and Timing

| | CCID | MaxRL | PA | CCID | MaxRL | PA |
|---|---|---|---|---|---|---|
| | Area [$\mu m^2$] | | | Delay [$ns$] | | |
| RCA:128/4 | 261 | 150 | 4055 | 0.24 | 0.27 | 6.88 |
| RCA:128/8 | 261 | 2890 | 2027 | 0.24 | 1.31 | 3.46 |
| CLA:128/4 | 261 | 150 | 5750 | 0.24 | 0.27 | 3.95 |
| CLA:128/8 | 261 | 2890 | 2898 | 0.24 | 1.31 | 2.04 |
| RCA:256/4 | 261 | 150 | 8110 | 0.24 | 0.27 | 13.7 |
| RCA:256/8 | 261 | 2890 | 4055 | 0.24 | 1.31 | 6.88 |
| CLA:256/4 | 261 | 150 | 14723 | 0.24 | 0.27 | 5.45 |
| CLA:256/8 | 261 | 2890 | 5750 | 0.24 | 1.31 | 3.95 |

in VHDL and mapped to a 0.25$\mu m$ technology using a standard synchronous design methodology. Figure 6 and Table 1 summarize the results after synthesis. For the CCID-adder results are shown for 4 and 8 subdivisions, each using 4 bits for early termination detection. A lookup table is used for the realization of the maximum run-length detection. Straight forward logic optimization in the synthesis tool leads to sufficiently good implementations with an area overhead of about 2800$\mu m^2$ and a delay below 1.5ns for 7 CCIDs. The initial implementation of the partial adders is based on a ripple-carry structure. A significant average speedup is obtained by using the CCID technique while the low area requirement of the ripple-carry adder is maintained. This is reflected in an almost fourfold gain in the $AT - product$ as opposed to the traditional implementation. To further increase the throughput the number of partial adders can be increased or other adder architectures can be used. The results when a carry-look-ahead adder is used instead are also shown in the figure.

When even faster architectures, such as fast carry-look-ahead or Brent-Kung tree adders are being used the performance gain saturates. This is because the delay of the CCID stage and of the maximum run-length detection circuit become larger than the delay of a single PA stage.

## 5. CONCLUSION

In this paper the design and application of a variable delay adder in a fully synchrnous design or as a synchronous island in a globally asynchronous, locally synchronous system was described. An analytical analysis of the achievable performance gain was presented and it was shown that a significant speedup can be achieved with only little area overhead. The technique is especially suitable for extremely large operands where very low area implementations are most desirable.

## 6. REFERENCES

[1] R. Zimmermann, *Binary Adder Architectures for Cell-Based VLSI and their Synthesis*, Konstanz: Hartung-Gorre Verlag, Series in Microelectronics Volume 73, 1998.

[2] S. M. Nowick, K. Y. Yun, A. E. Dooply, and P. A. Beerel, “Speculative Completion for the Design of High-Performance Asynchronous Dynamic Adders”, *In Proc. 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC ’97)*, Eindhoven, The Netherlands, April 1997, pp 210-223.

[3] V. A. Bartlett, and E. Grass, “Completion-detection technique for dynamic logic”, *Electronics Letters*, October 1997, vol 33, no 22, pp 1850-1852.

[4] A. De Gloria, and M. Olivieri, “Statistical Carry Lookahead Adders”, *in IEEE Tran. on Computers*, March 1996, vol 45, no 3, pp 340-347.

[5] A. De Gloria, and M. Olivieri,”Completion-detecting carry select addition”, *IEE Proc.-Comput. Digit. Tech.*, March 2000, vol. 147, no 2, pp 93-100.

[6] Y. Kondo et al., “An Early-Completion-Detecting ALU for a 1GHz 64b Datapath,” *in IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, 1997. pp 418-419, 497.

[7] J. Muttersbach, T. Villiger, and W. Ficthner, “Practical Design of Globally-Asynchronous, Locally-Synchronous Sytems”, *In Proc. 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC )*, Eilat, Israel, April 2000, pp 52-59.