고속 회로를 위한 비트 단위의 연산 최적화

엄준형 김영태 김태환 여준기 홍성백
한국과학기술원 전산학과
첨단정보기술 연구센터(AITrc)

Optimal Bit-level Arithmetic Optimization for High-Speed Circuits

{jhum, young, tkim, orualy, hongsup}@vlsisyn.kaist.ac.kr
Junhyung Um Youngtae Kim Taewhan Kim Chungi Lyuh Sungpack Hong

Dept. of Computer Science and Advanced Information Technology Research Center(AITrc)
Korea Advanced Institute of Science and Technology

요약

고속 회로 합성에 있어서, Wallace 트리 스타일은 연산을 위한 가장 효율적인 수행 방식의 하나로 인식되어 졌다. 그러나, 이러한 방법은 빠른 곱셈기의 수행이나 여러가지 연산수행에 있어, 입력 시그널을 고려하지 않은 일반적인 구조로 수행되어졌다. 본 논문은 연산기에 있어서 이러한 제한점을 극복하는 문제를 다룬다. 우리는 캐리-세이브 방법을 덧셈, 뺄셈, 곱셈이 혼합되어 있는 일반적인 연산 회로에 적용한다. 그 결과 효율적인 회로를 생성하며, 시그널들의 임의의 도달시간에 대해 회로의 도달시간을 최적화 한다. 또한, 우리는 최적 지연시간의 캐리-세이브 가산 회로를 생성하는 효율적인 알고리즘을 제안하였다. 우리는 이러한 최적화 방법을 여러 고속 디지털 필터에 적용시켜 보았고 이는 기존의 비트 단위가 아닌 캐리-세이브 수행방법보다 5%에서 30%사이의 수행시간 향상을 가져왔다.

1 서론

연산 회로에 있어 덧셈은 상당히 자주 행해지는 연산이기 때문에 효율적인 덧셈 연산의 수행은 매우 중요한 문제이다. 각 피 연산자 X_i가 n_i 비트인 연산식 $F = X_1 + X_2 + \cdots + X_m$ 를 실현하는 고속 회로를 만드는데 있어 가장 효율적이고 흔히 사용되는 방법은 캐리-세이브-가산기(CSA)를[1, 2] 이용하는 것이다. 여기서 주목할 점은 CSA 수행은 단지 덧셈에만 제한되지 않는다는 점이다. 우리는 뺄셈(예를 들어, $x-y = x+\bar{y}+1$)이나 곱셈같은 다른 연산도 덧셈으로 변환할 수 있다[3]. 그림 1은 캐리-세이브 가산기를 수행하는 두 단계의 구조를 보여준다. 첫번째 단계에서는 그림 2에서 보이는 full-adder를 사용한 캐리-세이브 가산을 수행한다. 이때 주목할 점은 full-adder(FA)간에 캐리-지연이 존재하지 않는다는 사실이다. 그리고, 두번째 단계에서는 첫번째 단계에서 생성된 두개의 피연산자를 캐리-지연 연산(carry-propagation addition) 을 이용하여 마지막 결과를 생성한다.

이러한 피연산자의 덧셈 수행은 비트 단위의 addend 행렬로 나타내어진다. 예를 들어, $X = x_3x_2x_1x_0, Y = y_3y_2y_1y_0, Z = z_2z_1z_0$, 그리고 $W = w_2w_1w_0$일때 연산식 $F = X + Y + Z + W$에 대한 행렬은 그림 3에서 볼 수 있다. 최적화 문제는 FA를 할당함으로써 각 열에 많아야 두개의 시그널들만 남도록 addend 행렬을 변환하는 것이다.

우리는 그림 1의 첫번째 단계에서 보여지는 이러한 방법을 *FA 트리 할당*이라 부르고, 두번째 단계의 수행을 *Final-adder 할당*이라 부른다. 여기서 최적화 문제는, 주어진 연산식 F에 대한 addend 행렬과 함수적으로 동치 관계

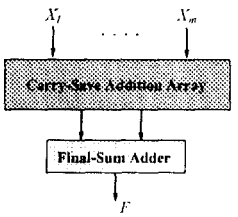

그림 1: 비트 단위의 캐리-세이브 최적화 문제

그림 2: n 비트의 캐리-세이브 가산기

col-3	col-2	col-1	col-0
x_3	x_2	x_1	x_0
y_3	y_2	y_1	y_0
	z_2	z_1	z_0
	w_2	w_1	w_0

그림 3: 초기 addend 행렬의 예

에 있는 가장 빠른 도달시간을 가지는 FA 트리를 만드는 것이다.

우리가 해결하고자 하는 FA 트리 할당 문제는 다음과 같이 나타낼 수 있다:

문제 1: *주어진 연산식*

$$F = X_1 + X_2 + \cdots + X_m \qquad (1)$$

에 대해 (이때, $X_i(= x_{i,n_i-1} \cdots x_{i,1},\ i = 1, 2 \cdots m)$는 n_i-bit이고, 각 비트 $x_{i,j}$의 도달 시간은 $t(x_{i,j})$이다.) 가장 빠른 도달시간($t(F)$)를 갖는 FA 트리 T의 할당을 결정한다.

Final Adder는 서로 다른 지연시간과 회로 면적을 가지는 여러가지의 연산기로 구현될 수 있기 때문에, 우리는 문제 1을 조금 변형시켜 $t(F)$를 최소화 하는 대신에, 모든 입력으로부터 final adder의 입력이 되는 시그널까지의 지연시간을 최적화 한다.

우리는 문제 1에 대한 우리의 해답을 단계적으로 제시한다.(모든 증명은 [5]에 제시되어 있다.) 모든 피연산자가 싱글 비트로 주어지는 연산식 (1)의 간단한 경우를 먼저 생각해 보자. 즉,

$$F = X_1 + X_2 + \cdots + X_m \qquad (2)$$

$bit_width(X_i) = 1, i = 1, 2 \cdots m$인 경우이다. 이에 대응하는 addend 행렬은 m개의 열을 갖는 싱글 비트 행렬이다. 예를 들어, m이 6일때, 이에 대응하는 초기 행렬과 첫번째 단계를 끝낸 행렬은 그림 4의 (a)와 (b)에서 볼 수 있다. 여기서, FA 트리 할당의 목적은 최소의 $max\{t(r_{11}), t(r_{12}), t(r_{21}), t(r_{22})\}$를 갖는 것이다.

col-0
x_{11}
x_{21}
x_{31}
x_{41}
x_{51}
x_{61}

(a) 초기 행렬

col-1	col-0
r_{21}	r_{11}
r_{22}	r_{12}

(b) 2개의 행으로 변환된 행렬

그림 4: $bit\_width(X_i)$=1, $i = 1, 2, \cdots m, m = 6$인 경우 addend 행렬 변환의 예

```
알고리즘 SC_T(M_0):
피연산자가 싱글비트의 입력일때의 지연시간 최적 FA 할당
 • M_0 = {x_{i,0}}, 1 ≤ i ≤ m
 /* 동일열 내부의 피연산자 집합 */
 while (|M_0| ≥ 3) {
   if (|M_0| > 3) {
   • M_0에서 최소 도달시간을 갖는 세 피연산자를 선택한다;
   • 새로운 FA에 선택된 피연산자를 입력으로 할당한다;
   }
   else { /* |M_0| = 3 */
   • M_0에서 최소 도달시간을 갖는 두 피연산자를 선택한다.
   }
   • M_0로부터 사용된 두개/세개의 피연산자를 제거한다;
   • FA/HA로부터의 출력을 M_0에 넣는다;
 }
```

SC_T 알고리즘은 최소 경로를 가지는 이진 트리를 구성한다는 점에서 허프만 [4] 알고리즘과 비슷하다. 같은 비트 열 내부에서는, SC_T 알고리즘은 모든 시그널을 FA를 이용하여 최소 지연시간을 갖는 두개의 시그널로 변환시킨다. 이는, 비트열 내부의 알고리즘은 허프만 알고리즘과 동일하다는 점에서 자명하다. 그러나, 한 열의 carryout 출력이 다음 열의 입력이 되어 다음 열의 지연시간에 영향을 미친다는 점에서 SC_T 알고리즘은 허프만 알고리즘과 큰 차이를 가진다. 다음 Lemma는, 알고리즘 SC_T을 적용했을때 모든 carryout들의 도달시간 또한 최소 지연시간을 갖는다는 강한 성질을 증명한다.

Lemma 1 *식 (2)의 주어진 싱글 비트의 addend 행렬에 대해, s_1과 s_2를 SC_T에 의해 생성된 두개의 sum 시그널이라 하고 $(t(s_1) \le t(s_2))$, $c_1, \cdots c_l$ $(t(c_1) \le \cdots \le t(c_l))$를 SC_T에 의해 생성된 두개의 carryout 시그널이라 하자. 또한, s_1'과s_2'를 $(t(s_1') \le t(s_2'))$ 임의의 알고리즘에 의해 생성된 두개의 sum 시그널이라 하고 $c_1', \cdots c_l'$ $(t(c_1') \le \cdots \le t(c_l'))$ 을 그에 대응하는 carryout 시그널이라 하자. 그러면, $t(s_1) \le t(s_1')$, $t(s_2) \le t(s_2')$, $t(c_1) \le t(c_1')$, $\cdots$이고, $t(c_l) \le t(c_l')$이다.*

Lemma 1에 기초하여, (2)의 수식 F에 대한 최소 지연시간을 갖는 FA 트리를 생성하기 위해, 우리는 알고리즘 SC_T를 생성되는 모든 열에 반복적으로 적용한다. SC_T는 초기의 싱글 비트 addend 행렬에 적용되어 FA를 이용하여 첫번째 열에 두개의 피연산자를 남기게 되며, 동시에, FA 할당으로 인해 다음 열에 carryout으로 인한 피연산자를 생성하게 된다. carryout으로 생성된 다음 열에 대해 SC_T를 반복적으로 적용하여 역시 두개의 피연산자만 남게 한다. 이러한 과정을 반복적으로 적용함으로써 우리는 (2)의 수식 F에 대한 최소 지연시간을 갖는 FA 트리를 생성할 수 있다.

Lemma 2 *수식 (2)에 대응하는 addend 행렬에 대해, n을 전체 열의 개수라 했을때, r_{i1}와 r_{i2} $(t(r_{i1}) \le t(r_{i2}))$, $i = 1, 2, \cdots n$ 를 SC_T의 반복적인 적용으로 생성된 i번째 열의 두개의 시그널이라 하자. 또한, r_{i1}'와 r_{i2}' $(t(r_{i1}') \le t(r_{i2}'))$, $i = 1, 2, \cdots n$ 를 임의의 알고리즘에 의해 최종적으로 생성된 i번째 열의 두개의 시그널이라 하자. 그러면, $t(r_{i1}) \le t(r_{i1}')$이고 $t(r_{i2}) \le t(r_{i2}')$, $i = 1, 2, \cdots n$이다.*

Lemma 2에서, FA의 carryout으로 인해 생성되는 모든 열에 SC_T 알고리즘을 반복적으로 적용했을때, 생성된 최종적인 각 열의 (많아야) 두개의 시그널이 최적의 지연시간을 갖는다는것을 보였다. 이러한 결과에서, 다음과 같은 Observation을 유도할 수 있다.

Observation: Lemma 2는, (2)의 수식 F에 대해, 우리의 FA 할당 알고리즘으로 생성된 시그널의 도달시간보다 더 적은 도달시간을 갖는 FA 트리 할당 알고리즘이 존재하지 않음을 증명하였다. 결과적으로, 우리의 알고리즘으로 생성된 시그널을 final adder의 입력으로 이용했을때, 최적의 지연시간을 갖는 출력을 생성할 수 있다. (즉, 수식 (2)의 문제 1에 대한 최적의 $t(F)$값을 생성한다.)

이제, 우리는 일반적인 입력에 대한, (1)의 수식 F에 대한 문제를 해결해야 한다. 앞으로 제시될 FA 할당 알고리즘은 Observation 1과 수식 (2)에 관련된 addend 행렬을 변환하기 위해 사용한 과정에 기초하고 있다. 새로운 알고리즘(FA_AOT)는 다음과 같은 방법으로 진행된다:

여러개의 열을 갖는 초기 addend 행렬에서 2^0의 weight를 갖는 열에 대해 알고리즘 SC_T를 적용한다(즉, 가장 오른쪽의 열). 다음 반복에, 알고리즘 SC_T를 weight 2^1를 갖는 다음 열에 적용한다. 이러한 과정을 가장 왼쪽의 열에 대해서까지 계속 적용한다.

Theorem 1 *알고리즘 FA_AOT 는 수식 (1)의 연산식 F에 대해 최소 지연시간을 갖는 FA 트리를 할당한다.*

Theorem 1은 우리의 알고리즘이 최적 지연시간을 갖는 FA 트리를 생성함을 증명한다. 이제, 우리는 다음에서 전체적인 알고리즘의 흐름을 요약한다:

알고리즘 FA_AOT(F):
수식(1)의 F를 위한 최적 지연시간의 FA 트리 할당 :
- $n = max\{\, n_k \mid 1 \le k \le m \,\}$ /* n_k: *bit-width*(X_k) */
- $\mathcal{M}_j = \{x_{i,j}\},\ 0 \le j \le n-1$

/* addend 행렬 $\mathcal{M}$의 j번째 열의 시그널들의 집합 */
- $j = 0$; /* 가장 오른쪽의 열(최소 weight을 갖는 열) */

repeat {
- *SC_T*($\mathcal{M}_j$)를 부른다;
- FA(만약 HA가 만들어졌다면 HA)에 의해 생성된 모든 시그널을 $\mathcal{M}_{j+1}$에 넣는다:

(만약 M_{j+1}가 존재하지 않는다면, M_{j+1}를 새로 만들고, 생성된 시그널을 넣는다.)
- $j = j + 1$;

} **until** ($|\mathcal{M}_s| \le 2$, $s = 1, 2, \cdots$)

/* 여기서, $\mathcal{M}$ 는 두개의 행만을 갖는다 */
- 두 행의 비트 width와 동일한 width를 갖는 final adder를 만든다;
- 두개의 피연산자를 생성된 final adder의 입력으로 할당한다;

2 실험

우리는, 우리의 알고리즘 FA_AOT를 흔히 사용되는 많은 수의 연산에 적용하였다. 우리의 프로그램은 연산식을 입력으로 받아들여 (입력들의 특성, 즉, 시그널들의 width와 도달시간등과 함께) 최적의 지연시간을 갖는, 함수적으로 동치인 Verilog HDL로 이루어진 netlist를 생성한다. 다음으로, Netlist는 Synopsys *Design Compiler*[8] 을 이용하여 logic 최적화 되었다. 알고리즘의 효율을 비교하기 위해 word 단위의 CSA 최적화 알고리즘인 *CSA_OPT*[6] 알고리즘을 수행하고, 이를 우리의 결과와 비교하였다. 실험에서, 우리는 *lcbg10pv* ($.35\mu$) technology[7]를 사용하였다.

표 1은 *CSA_OPT*, 그리고 *FA_AOT*에 의해 생성된 회로를 비교한 결과를 보여준다. 첫번째 열에서, 각 디자인의 비트 width와 도달시간을 명시하였다. IIR은 2nd-order iir필터의 연산 부분이며 Kalman는 kalman filter의 state vector 계산 부분이다. Complex는 허수 계산의 연산 부분이며, Serial-Adapter는 많은 ladder 디지털 필터 구조에서 사용되는 3-port serial adapter이다. *FA_AOT*는 항상 *CSA_OPT*보다 지연시간에 있어 같거나 더 빠른 최적 지연시간을 갖는다는것이 본문에서 증명되었기 때문에, 이러한 비교는, 단지 우리의 알고리즘이 얼마나 효율적인지를 보이는것이다. 실제로, 우리의 알고리즘이 polynomial 연산식 뿐아니라 필터 디자인들에 대해서도 좋은 결과를 생성함이 표에서 보여진다. 전체적으로, *FA_AOT*는 일반적인 알고리즘에 대해 더 적은 회로 면적을 차지하며 평균 18% 의 지연시간의 감소를 갖는 회로를 생성하였다.

Design	CSA (timing,area)	Proposed (timing,area)	Impr. wrt CSA
X^2 (X: 32-bit)	4.69 ns 37981 units	3.52 ns 36797 units	24.9% faster 3.1% smaller
$X \cdot Y+1$ (X,Y: 8-bit, 2.0 ns)	4.07 ns 2871 units	3.61 ns 2704 units	11.3% faster 5.8% smaller
$X^2 + X + Y$ (X,Y: 8-bit, X: 0.7 ns)	3.84 ns 3789 units	3.18 ns 3111 units	17.2% faster 17.8% smaller
$(X+Y)^2 + 2(X+Y) + 1$ (X,Y: 8-bit, 1.0 ns)	4.63 ns 8134 units	4.01 ns 6458 units	13.4% faster 20.6% smaller
IIR (16-bit output)	4.75 ns 11202 units	3.68 ns 8349 units	22.5% faster 25.5% smaller
Kalman (32-bit output)	4.50 ns 25713 units	3.69 ns 21542 units	18.0% faster 16.2% smaller
IDCT (32-bit output)	6.38 ns 77052 units	4.45 ns 60307 units	30.2% faster 21.7% smaller
Complx (32-bit output)	4.51 ns 50083 units	3.70 ns 38343 units	17.9% faster 23.4% smaller
Serial-Adapter (16-bit output)	6.00 ns 5608 units	5.72 ns 5631 units	4.7% faster 0.4% larger

표 1: 지연시간 최적화된 회로 비교

3 결론

본 논문에서, 우리는 비트 단위의 캐리-세이브 가산에 기초한 새로운 연산 회로 합성 알고리즘을 제시하였다. 곱셈 연산기에 대한 기존의 비트 연산 변환의 적용과는 다르게 우리는 덧셈, 뺄셈, 곱셈이 혼합된 일반적인 연산식에 이러한 방법을 적용하였으며, 각 입력 시그널들에 대한 동일한 도달시간을 가정한 기존의 방법과는 다르게 일반적인 시그널의 도달시간에 대해 최적 지연시간을 생성하는 비트 압축 알고리즘을 제시하였다. 실험에서, 우리는 우리의 알고리즘이 비트 단계의 캐리 세이브 가산을 전반적인 가산 회로에 효율적으로 적용함을 보였으며, 이는 회로 지연시간과 회로 면적에 상당한 향상을 가지고 오는것을 보였다.

감사의 글

본 논문은 첨단정보기술 연구센터(AITrc)를 통하여 과학재단의 지원을 받았음.

참조 서적

[1] D. D. Gajski, "Parallel Compressors," *IEEE Transactions on Computers*, Vol. C-29, No.5, pp. 393-398, May 1980.
[2] N. Weste, K. eshraghian, *Principles of CMOS VLSI Design - A Systems Perspective*, Addition-Wesley Publishers, 1985.
[3] T.kim, W.Jao, and S. Tjiang, "Circuit optimization using carry-save-adder Cells", *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, October 1998, 17, No.10, pp. 974-984.
[4] D. Huffman, "A method for the construction of minimum redundancy codes", *Proc. of the IRE*, Vol.40, pp.1098-1101, 1952.
[5] J.Um and T.Kim, "A study on fine-grained arithmetic optimization for high -performance/low-power data path", *KAIST/CS Technical Report, No. CS-TR-99-141*, 1999.
[6] J. Um, T. Kim and C. L. Liu, "Optimal Allocation of Carry-Save-Adders in Arithmetic Optimization", *Proc. of International Conference on Computer-Aided Design*, November 1999.
[7] LSI Logic Inc., *G10-p Cell-Based ASIC Products Databook*, 1996.
[8] Synopsys Inc., *Design Compiler Reference Manual*, 1998.