Contention resolution algorithm for common subexpression elimination in digital filter design

Xu, Fei; Chang, Chip Hong; Jong, Ching Chuen

2005

Xu, F., Chang, C. H., & Jong, C. C. (2005). Contention resolution algorithms for common subexpression elimination in digital filter design. *IEEE Transactions on Circuits and Systems-II: Express Briefs*, 52(10), 695-700.

<https://hdl.handle.net/10356/91438>

<https://doi.org/10.1109/TCSII.2005.851776>

IEEE Transactions on Circuits and Systems-II: Express Briefs © copyright 2005 IEEE.
Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. <http://www.ieee.org/portal/site>.

Contention Resolution Algorithm for Common Subexpression Elimination in Digital Filter Design

Fei Xu, Chip-Hong Chang, *Senior Member, IEEE*, and Ching-Chuen Jong

Abstract—In this paper, a new algorithm, called contention resolution algorithm for weight-two subexpressions (CRA-2), based on an ingenious graph synthesis approach has been developed for the common subexpression elimination of the multiplication block of digital filter structures. CRA-2 provides a leeway to break away from the local minimum and the flexibility of varying optimization options through a new admissibility graph. It manages two-bit common subexpressions and aims at achieving the minimal logic depth as the primary goal. The performances of our proposed algorithm are analyzed and evaluated based on benchmarked finite-impulse-response filters and randomly generated data. It is demonstrated that CRA-2 achieves the shortest logic depth with significant reduction in the number of logic operators compared with other reported algorithms.

Index Terms—Common subexpression elimination (CSE), logic depth, multiple constants multiplication.

I. INTRODUCTION

THE multiple constant multiplication (MCM) has been a core operation and often performance bottleneck in many digital signal processing applications. The efficiency in reducing the cost metric (power, hardware, and logic depth) of the MCM directly influences the performance of ASIC implementation. The notion of primitive operator has been introduced in [1] by applying the graph theoretic algorithm to systematically minimize the implementation cost of the multiplication block of the transposed direct-form finite-impulse-response (FIR) filter. Yet another popular optimization approach to MCM problem is common subexpression elimination (CSE) [3], [4], [6]–[9]. Different CSE algorithms lead to different realizations of the multiplier block. The objective functions and filter coefficient representations play important roles in the quality of solutions.

Due to the complexity of the MCM problem defined in [8], there is no exact algorithm that guarantees optimality of the solutions for all kinds of coefficients. Among the popular representations used for CSE problems, canonical signed digit (CSD) representations provide the minimum number of nonzero digits and unique formation. On average, CSD can reduce nonzero digits by 33% compared with the binary representation. For

coefficients expressed in CSD form, optimal subexpression sharing algorithms based on the integer linear programming (ILP) model have been proposed [3], [9]. Both algorithms have high computational complexity as the optimal selection of common subexpressions is an NP-complete problem. Since the exact optimal solution is intractable, almost all existing algorithms are heuristic in nature, and the solutions generated are highly possibly suboptimal due to local minima. There is no feedback to allow efficient resubstitution of lower cost common subexpressions in case of conflicts of potential subexpression elimination caused by early decision in the execution process. In this paper, we proposed a new contention resolution algorithm for weight-two subexpression (CRA-2) featuring the ability to change the precedence of subexpressions to resolve the local minima problem. CRA-2 takes the CSD coefficients to reduce the implementation cost and maximize the performance (reduced logic depth), but it is also applicable to coefficients represented in other forms. Reducing the logic depth has the added advantage that shallow taps shorten the propagation paths of spurious switching activities through long chains of adders, contributing to lower power dissipation.

The remainder of this paper is organized as follows. Section II introduces the notion of the admissibility graph. Our proposed CRA-2 is described in Section III, followed by the performance analysis and comparison of simulation results with distinguished algorithms in Section IV. Concluding remarks are made in Section V.

II. ADMISSIBILITY GRAPH

We use an admissibility graph $G(V, E)$ to describe a set of filter coefficients. The following graph terminologies are useful in the construction of the admissibility graph. An edge $e = (v_i, v_j)$ is a connection of two vertices v_i and v_j , and v_i and v_j are called the endpoints of the edge e . A pair of edges incident with a common vertex is said to be adjacent. The degree of a vertex v denoted by $\deg(v)$ is the number of edges incident with v . The vertex set V is a collection of all the nonzero digits of the coefficients and the edge set E is a collection of all subexpressions of weight two within a coefficient. A subexpression is a signed digit string begins and ends with nonzero digits. Two subexpressions are said to be common if one can be obtained from another without incurring adder costs (i.e., only shifters and inverters are necessary).

An FIR filter can be modeled as a union of admissibility graphs, i.e., $G(V, E) = \bigcup_{i=1}^N G_i(V_i, E_i)$, where N is the total number of coefficients and $G_i(V_i, E_i)$ is an M -vertex admissibility graph for the i th coefficient with M nonzero digits in its CSD representation. Each vertex v is associated with a unique

Manuscript received September 4, 2003; revised January 19, 2005. This work was supported by the Panasonic Singapore Laboratory under ChiPES-PSL Joint R&D M4890001. This paper was recommended by Associate Editor J. McCanny.

The authors are with the Centre for High Performance Embedded Systems and Centre for Integrated Circuits and Systems, School of Electronic and Electrical Engineering, Nanyang Technological University, Singapore 639798 (e-mail: fei.xu@pmail.ntu.edu.sg; echchang@ntu.edu.sg; eccjong@ntu.edu.sg).

Digital Object Identifier 10.1109/TCSII.2005.851776

Fig. 1. Four-vertex admissibility graph.

positional index $\text{index}(v)$ determined by its position. Two types of vertices are defined: a signed vertex corresponds to the digit $\bar{1}(-1)$ and an unsigned vertex corresponds to the digit 1. An edge connecting two identical (opposite) sign vertices v_i and v_j forms an even (odd) subexpression separated by distance $d = |\text{index}(v_i) - \text{index}(v_j)| - 1$ of zeros. All vertices in the admissibility graph are set free initially. A precedence edge $e_p = (v_i, v_j)$ is used to link two free vertices v_i and $v_j \in V_k$ of the k th coefficient. Once a vertex is connected by a precedence edge, it becomes fixed. Two vertices v_i and v_j are linked via a contention edge $e_c = (v_i, v_j)$ if at least one vertex v_i or v_j is fixed. A precedence edge is used to commit a subexpression into the optimal common subexpression list, and a contention edge is used to indicate a subexpression that is denied admission as an optimal common subexpression by another subexpression that has already been committed. In other words, each contention edge implies a conflict (contention) between two common subexpressions. Selection of one into the final common subexpression list will automatically nullify the admissibility of the other. Each edge e is given an order of precedence order(e) according to the number of occurrences of its subexpression, with the lowest value of order(e) signifying the highest number of occurrences. Two edges are said to be equivalent if they have the same order. Equivalent edges are common subexpressions.

As an illustration, Fig. 1 shows a four-vertex admissibility graph for the coefficient $C_2 = 010\bar{0}10\bar{1}0\bar{0}$ of Example 1 in the next section. The orders of the edges are obtained from the statistical analysis of the full set of coefficients, and the state of Fig. 1 occurs at the fourth stage of the contention resolution procedure. The unsigned and signed vertices are drawn as solid and hollow circles, respectively. The precedence and contention edges are drawn with solid and dashed lines, respectively. Each vertex is annotated with its positions within the coefficient, and each precedence edge is annotated with its order of precedence. This admissibility graph features one precedence edge $v_{22}v_{23}$, and two contention edges $v_{21}v_{22}$ and $v_{23}v_{24}$. Edges $v_{22}v_{23}$ represents the odd subexpression $10\bar{1}$ of order 1. Vertices v_{21} and v_{24} are free vertices.

An admissibility graph is constructed by connecting vertices with precedence or contention edges. A contention occurs when the endpoints of a precedence edge are incident with contention edges that have a free endpoint. For example, in Fig. 1, the pattern graph represented by the path $v_{21}v_{22}v_{23}v_{24}$ requires a contention resolution. The contention resolution decides if the pattern graph should remain unchanged or to promote both contention edges to precedence edges and downgrade the precedence edge to contention edge. Every precedence edge in the

final solution graph is a partial sum, and precedence edges sustaining to the final stage of the solution graph provide all information required for the circuit implementation and logic complexity estimation.

III. CONTENTION RESOLUTION ALGORITHM FOR WEIGHT-TWO SUBEXPRESSIONS

Existing algorithms are based on a heuristic search technique or combined exhaustive search with steepest descent approach [1], [2], [4], [6]–[8] to select the common subexpressions for elimination. These algorithms suffer from a common problem that, once a subexpression is identified as the most profitable common subexpression, the decision cannot be reverted. The gross effect of interdependency of coefficients to the cost of implementation is not considered. In this section, we propose a new algorithm to overcome this problem with minimal logic depth based on the notion of an admissibility graph. We relax the rigidity of the search for common subexpressions by allowing the earlier chosen candidate subexpressions marked by the precedence edges to be substituted by the adjacent contention edges determined by a specific cost metric. The final solution is generated at the last stage when all the major contentions have been resolved. This provides the algorithm with the ability to correct choices made earlier to avoid local minimum.

The proposed CRA-2 searches for optimal common subexpressions of weight two. Let L be the set of unique CSD coefficients that have more than one nonzero digit. A list of common subexpressions of weight two S is generated from L and sorted in descending order of frequency of occurrences. In computing the frequency, we do not differentiate the sign of the subexpressions, and identical overlapping subexpressions are counted only once. When there is a tie, the subexpressions are further sorted in ascending order of distance. A unique order number is assigned to each distinct subexpression and subexpressions of order = 1 have the highest precedence.

Once the order list S is established, we can begin to construct the admissibility graph $G_i(V_i, E_i)$ for each coefficient C_i from L , $i = 1, 2, \dots, |L|$. The pseudocode of the proposed CRA-2 is shown in Fig. 2. E_P and E_C are the lists storing the precedence and contention edges, respectively. The degrees of two endpoints of a precedence edge are used to check if a contention has occurred. Two main types of contentions, as shown in Fig. 3, are considered for resolution.

The first type of contentions that required resolution occurs when both endpoints of a precedence edge are adjacent to contention edges and the other endpoints of the adjacent contention edges have to be free. The second type is similarly defined except that only one endpoint of the precedence edge is adjacent to at least one contention edge. A cost metric NAS is defined to determine if the status of the precedence edge and contention edges need to be swapped. NAS can be interpreted as an anticipated minimal number of adders saved by swapping based on a localized model. For ease of exposition, the term adder is used to describe either an adder or a subtractor. Let $e_p = (v_2, v_3)$ be the precedence edge and $e_{c1} = (v_1, v_2)$ and $e_{c2} = (v_3, v_4)$ be the two contention edges of the first type (see Fig. 3). If e_p is

```

1: for (each  $C_i$  from  $L$ ,  $i = 1 \dots |L|$ )
2:   generate  $G_i$  from the nonzero digits of  $C_i$ ;
3:  $E_P = \emptyset$ ;  $E_C = \emptyset$ ;
4: for ( $i = 1 \dots \max\_order$ ) {
5:   each  $S_j$  from  $S$ , with  $order(S_j) = i$ ;  $v_1, v_2 \in S_j$ ;
6:   if ( $v_1, v_2$  are both free) {
7:     connect ( $v_1, v_2$ ) with precedence edge,  $e_p$ ;
8:      $order(e_p) = order(S_j)$ ;  $E_P = E_P \cup e_p$ ; }
9:   else {
10:    connect ( $v_1, v_2$ ) with contention edge,  $e_c$ ;
11:     $order(e_c) = order(S_j)$ ;  $E_C = E_C \cup e_c$ ; }
12:   for (each  $e_p = (v_1, v_2) \in E_P$ )
13:     if (contention) { -- contention resolution required
14:       compute exchange cost,  $NAS$ ;
15:       if ( $NAS > 0$ ) { -- potential saving by exchange
16:         let  $e_1 \in E_C$  be the minimal-cost edge adjacent to  $v_1$ ;
17:         let  $e_2 \in E_C$  be the minimal-cost edge adjacent to  $v_2$ ;
18:          $E_P = (E_P \cup e_1 \cup e_2) - e_p$ ;  $E_C = (E_C \cup e_p) - e_1 - e_2$ ; } }
19: return  $E_P$ ;

```

Fig. 2. CRA for weight-two subexpressions.

Fig. 3. Two main types of contentions.

chosen as the final common subexpression, two adders are required to realize the coefficient for the two free vertices v_1 and v_4 . Otherwise, if e_{c1} and e_{c2} replace e_p to be the final subexpression, there is no free vertex, but an adder is required to sum the two subexpressions. Therefore, the number of adders saved by swapping is one. We set $NAS = 1$ initially for this simplistic case and adjust the cost by considering other edges in E_P and E_C . If there exists equivalent precedent edges of e_{c1} and e_{c2} , then the adders required for the partial sums of e_{c1} and e_{c2} have already been allocated elsewhere. NAS remains unchanged at this stage as no additional adder is incurred. If there is no edge in E_P equivalent to e_{c1} , then one adder is required to realize this subexpression, and $NAS = NAS - 1$. The same evaluation is carried out for e_{c2} as well. If there is no other equivalent precedence edge of e_p , then $NAS = NAS + 1$, because no edge can share with e_p . If the final adjusted value of NAS is greater than zero, it will be profitable to swap the status of the precedence edge with the contention edges, as indicated in lines 16–18 of Fig. 2.

As all edges of the same order are evaluated in the same iteration, more than one contention may be encountered concurrently. Under this complex situation, all pattern graphs involved in the contention resolution have to be considered as a whole. Swappings of some or all of the contentions are performed when their aggregate NAS is greater than the cost of generating the hitherto nonexistent precedence edges.

At the end of the program, the precedence edges in E_P form the partial sums. The cost of implementation in terms of the total number of adders used can be determined from E_P . For a fixed coefficient set, shifters can be hardwired directly, and the complexity and throughput are dictated by the number of adders used. The calculation can be divided into two parts:

- 1) Common subexpression implementation: each unique common subexpression of weight two requires an adder. Let $n(E_P)$ denote the number of distinct common subexpressions in E_P , then the cost of realizing the common subexpressions is $A_1 = n(E_P)$
- 2) Coefficient implementation: nonzero digits that are not included in the final common subexpressions need to be summed with the partial sums to produce the final coefficient. This cost is equal to the number of precedence edges plus the number of free vertices. Let $|E_P|$ denote the number of precedence edges for the whole coefficient set, and $|E_{\text{free}}| = \sum_{v \in V, \deg(v)=0} 1$ is the number of free vertices. The total number of adders required for this part is $A_2 = |E_P| + |E_{\text{free}}| - 1$. In this calculation, the adders in the final linear accumulator are included, while in some papers [1]–[4], [6]–[8] this accumulator cost is not considered because it is not part of the multiplier block.

This difference will be considered when comparing the results of different algorithms. The reason of its consideration here is because common subexpressions may be eliminated across multiple multiplier blocks to form a more efficient solution, but no redundancy can be explored from this accumulator cost. Thus, the total cost of adders is given by

$$N_a = A_1 + A_2 = n(E_P) + |E_P| + |E_{\text{free}}| - 1. \quad (1)$$

The logic depth is contributed by the coefficients with the greater number of common subexpressions and free vertices. Since the logic depth of a common subexpression of weight two is equal to one adder delay, the logic depth D is given by

$$D = \max \left(\lceil \log_2 |V_i| \rceil, \max_{i=2}^{|L|} \lceil \log_2 |V_i| \rceil + 1 \right) \quad (2)$$

where $|V_i|$ is the number of vertices of the subgraph G_i and $\lceil a \rceil$ is the smallest integer larger than a . Note that the logic depth calculation has included the delay of the accumulator.

Example 1: Consider the following set of filter coefficients: $L = [10\bar{1}010\bar{1}000\bar{1}, 010010\bar{1}0\bar{1}00, 100100\bar{1}0\bar{1}00]$. The common subexpressions $10\bar{1}$ and $100000\bar{1}$ occur three times. 101 , 1001 , 10001 , and $10000\bar{1}$ occurs twice, and they are assigned $order = 1, 2, \dots, 6$ successively. Fig. 4 shows the labeling of all order-1 subexpressions $10\bar{1}(\bar{1}01)$ on the admissibility graph. An arc is used to cover all vertices of a subgraph of a single coefficient. $E_P = \{v_{12}v_{13}, v_{14}v_{15}, v_{22}v_{23}\}$, and $E_C = \{v_{13}v_{14}\}$. The contention edge $v_{13}v_{14}$ bridging the two precedence edges does not fulfill the contention resolution criterion since both its endpoints are fixed.

We proceed with the labeling of higher order subexpressions progressively. There is a contention between $v_{12}v_{13}$ and $v_{11}v_{13}$ for order-2 subexpressions, but no change is made since $NAS = 0$. Similar contention patterns with no resolution made

Fig. 4. Labeling of order-1 subexpressions.

Fig. 5. Labeling of order-4 subexpressions.

Fig. 6. Results of the contention resolution.

are also observed in the coefficients C_2 and C_3 for order-3 subexpressions.

Two interesting contentions occurred in order-4 subexpressions are shown in Fig. 5. One occurs between the precedence edge $v_{22}v_{23}$ and contention edges $v_{21}v_{22}$ and $v_{23}v_{24}$ in C_2 and another between the precedence edge $v_{32}v_{34}$ and contention edges $v_{31}v_{32}$ and $v_{33}v_{34}$ in C_3 . To these ends, $E_P = \{v_{12}v_{13}, v_{14}v_{15}, v_{22}v_{23}, v_{32}v_{34}\}$ and $E_C = \{v_{11}v_{13}, v_{12}v_{15}, v_{13}v_{14}, v_{21}v_{22}, v_{23}v_{24}, v_{31}v_{32}, v_{33}v_{34}\}$. We set $NAS(v_{22}v_{23}) = NAS(v_{32}v_{34}) = 1$. Since there is no equivalent edge in E_P with orders 3 and 4, $NAS(v_{22}v_{23})$ and $NAS(v_{32}v_{34})$ remain as 1. As there exists equivalent precedence edge of order 1 in E_P , $NAS(v_{22}v_{23})$ remains unchanged. Since there is no equivalent edge of order 2 in E_P to $v_{32}v_{34}$, an adder is needed to realize $v_{32}v_{34}$ and $NAS(v_{32}v_{34}) = 1 + 1 = 2$. The cost of constructing the nonexistent precedence edges of orders 3 and 4 is two. Therefore, if the contentions are considered independently, $NAS(v_{22}v_{23}) < 2$ and $NAS(v_{32}v_{34}) = 2$, no swapping will be carried out. However, as the aggregate $NAS = NAS(v_{22}v_{23}) + NAS(v_{32}v_{34}) = 3 > 2$, one adder can be saved by simultaneously exchanging the precedence edges $v_{22}v_{23}$ and $v_{32}v_{34}$ with their respective adjacent contention edges. The result is $E_P = \{v_{12}v_{13}, v_{14}v_{15}, v_{21}v_{22}, v_{23}v_{24}, v_{31}v_{32}, v_{33}v_{34}\}$ and $E_C = \{v_{11}v_{13}, v_{12}v_{15}, v_{13}v_{14}, v_{22}v_{23}, v_{32}v_{34}\}$, as shown in Fig. 6.

The final list of partial sums is given by the precedence edges of E_P . The contention edges are removed from the graph. The number of adders N_a required to implement the coefficient set L can be calculated from (1). There are three distinct common subexpressions of orders 1, 3, and 4, so $n(E_P) = 3$. $|E_P| +$

Fig. 7. Final AG graph of Example 1.

Fig. 8. Circuit implementation.

TABLE I
SIMULATION RESULTS FOR FIR FILTERS

Filter	N	L	CSD		BHM		Hartley		RAG-n		Pasko		NRSCSE		CRA-2	
			LO	LD	LO	LD	LO	LD	LO	LD	LO	LD	LO	LD	LO	LD
FIR1	3	11	12	3	9	6	10	3	9	5	10	3	10	3	9	3
FIR2[7]	4	8	14	3	9	7	10	3	11	6	10	4	9	3	9	3
FIR3[8]	4	11	17	4	11	7	13	4	10	6	11	5	13	4	13	4
FIR4[6]	25	9	35	3	31	6	33	3	31	3	30	3	30	3	30	3
FIR5[6]	59	14	115	3	86	21	99	3	86	5	90	3	90	3	90	3
FIR6[4]	121	17	265	4	173	45	204	4	172	7	179	5	188	4	182	4

Fig. 9. Comparison of LO between CRA-2 and NRSCSE.

$|E_{\text{free}}| = 7$. Thus, $N_a = 3 + 7 - 1 = 9$. From (2), the logic depth $D = 3$.

It is intuitive to map the admissibility graph solution to the hardware architecture. The steps involved are as follows.

- Step 1) Generate the weight two common subexpressions corresponding to the distinct precedence edges. One adder is needed to generate the partial sum for each order.
- Step 2) Free vertices are summed with the generated partial sums to complete the multiplier block. The adder

Fig. 10. Comparison of average adder cost and computation time of CSD, NRSCSE, Pasko, and CRA-2 algorithms.

tree of partial sums generated in Step 1) and the free vertices are balanced to obtain the minimum depth tree for speed.

Step 3) Accumulate the products of input and coefficients to the final output.

To balance the adder tree in Step 2), free vertices of each G_i are arbitrarily paired and added first at level 0. As the partial sums reach adder level 1, they are then added with the partial sums of precedence edges. This way of allocating the sparse vertices and edges according to the arrival time of partial sum outputs will minimize the logic depth without increasing the number of operators. Fig. 8 shows the result of mapping Fig. 7 to an adder tree.

IV. EXPERIMENTAL RESULTS

Two methods are used to evaluate the performances of CRA-2. First, several benchmark filters commonly used for performance evaluation are simulated. The qualities of the generated solutions, in terms of the number of logic operators (LOs) and logic depth (LD), are compared with the reported results in the literature. Second, a more rigorous experimentation is also performed using randomly generated coefficient sets composed of different numbers of taps N and coefficient lengths l . The direct CSD implementation, Pasko [6], and nonrecursive signed CSE (NRSCSE) [7] algorithms are prototyped and run on the same machine to show the efficiency of our CRA-2 algorithm.

The simulation results of the benchmark filters for the comparison of various algorithms are shown in Table I. FIR4–FIR6 are symmetrical filters and the values of N refer to the total number of taps. BHM and Hartley refer to the Modified Bull and Horrock [1] and Hartley [4] algorithms, respectively. For those algorithms that are not implemented, their results are taken from [7] directly. For consistency, the number of LOs presented in Table I has included the accumulator associated with each filter tap.

From Table I, our proposed CRA-2 always achieves the same lowest logic depth as NRSCSE and Hartley, but CRA-2 outperforms NRSCSE and Hartley in the number of logic operators. When the number of filter taps and the word lengths increase, the number of logic operators saved by CRA-2 over

NRSCSE becomes significant. All other algorithms have higher logic depth. The number of logic operators of CRA-2 is also comparable to that of the Pasko algorithm, which is an algorithm that seeks for optimal common subexpressions of all weights. Being a heuristic algorithm, CRA-2 generates results which are asymptotically close to the minimal achievable logic complexity modeled and solved by ILP algorithms [9] for coefficients represented in CSD form. For the three examples given in [3], CRA-2 gives exactly the same number of logic operators as [9] as both optimization algorithms use only weight-2 common subexpressions. In short, CRA-2 promises good overall performance in view of minimal logic depth for solutions with commensurate optimality of logic complexity.

For more general comparisons, the performances of CRA-2, NRSCSE, and Pasko algorithms are tested on randomly generated data. Every data used in this test are taken from 50 sets of randomly generated coefficients. First, CRA-2 and NRSCSE are compared under different word lengths and different numbers of taps, as shown in Fig. 9. NRSCSE is one of the latest algorithms based on CSD representation and produces excellent solutions with special emphasis on minimizing the logic depth. It is evident from the results of Fig. 9 that CRA-2 performs well in terms of logic complexity. In general, CRA-2 saves 1%–3% more logic operators than NRSCSE does. As the size of the coefficient set increases, the predominance of CRA-2 over NRSCSE becomes discernible.

The average adder costs for filters with different number of taps at a fixed word length are simulated and average adder cost and computation time are charted in Fig. 10. The MATLAB programs are run on a Pentium IV 1.9-GHz personal computer with 256 MB of system memory. The word lengths of the coefficients are fixed at 11. It is obvious that CRA-2 work well at the common word length. The least logic depth CRA-2 has adder cost comparable to Pasko algorithm while it consumes the least CPU time in all the test cases.

V. CONCLUSION

A new contention resolution algorithm has been proposed to circumvent the local minimum problem encountered by the existing CSE algorithms. A novel admissibility graph is employed as an efficient data structure for the proposed CRA-2.

The benefit of the algorithm is derived from their ability to appraise and substitute the chosen subexpressions when better alternatives emerge. Reduction in the number of logic operators is used directly as the criterion of the appraisal. Logic depth is also considered in devising the algorithms. The results are promising. The proposed algorithm CRA-2 yields the shortest logic depth architecture with a substantially minimized number of logic operators.

REFERENCES

- [1] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *Proc. Inst. Elect. Eng.*, pt. G, vol. 138, no. 3, pp. 401–412, Jun. 1991.
- [2] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 42, no. 9, pp. 569–577, Sep. 1995.
- [3] O. Gustafsson and L. Wanhammar, "ILP modeling of the common subexpression sharing problem," in *Proc. IEEE Int. Conf. Electron. Circuits and Syst.*, vol. 3, Croatia, Sep. 2002, pp. 1171–1174.
- [4] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 43, no. 10, pp. 677–688, Oct. 1996.
- [5] I. C. Park and H. J. Kang, "Digital filter synthesis based on minimal signed digit representation," in *Proc. IEEE Design Automation Conf.*, Las Vegas, NV, Jun. 2001, pp. 468–473.
- [6] R. Paško, P. Schaumont, V. Derudder, S. Vernalde, and D. Đuračková, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 1, pp. 58–68, Jan. 1999.
- [7] M. M. Peiró, E. I. Boemo, and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 49, no. 3, pp. 196–203, Mar. 2002.
- [8] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 2, pp. 151–165, Feb. 1996.
- [9] A. P. Yurdakul and G. Dundar, "Multiplierless realization of linear DSP transforms by using common two-term expressions," *J. VLSI Signal Process.*, vol. 22, pp. 163–172, Sep. 1999.