Area Minimization Synthesis for Reconfigurable Single-Electron Transistor Arrays with Fabrication Constraints

Yi-Hang Chen, Jian-Yu Chen and Juinn-Dar Huang
Department of Electronics Engineering and Institute of Electronics
National Chiao Tung University, Hsinchu, Taiwan
{carlok, M100cychen}@adar.ee.nctu.edu.tw, jdhuang@mail.nctu.edu.tw

***Abstract*—As fabrication processes exploit even deeper submicron technology, power dissipation has become a crucial issue for most electronic circuit and system designs nowadays. In particular, leakage power is becoming a dominant source of power consumption. Recently, the reconfigurable single-electron transistor (SET) array has been proposed as an emerging circuit design style for continuing Moore's Law due to its ultra-low power consumption. Several automated synthesis approaches have been developed for the reconfigurable SET array in the past few years. Nevertheless, all of those existing methods consider fabrication constraints, which are mandatory, merely in late synthesis stages. In this paper, we propose a synthesis algorithm, featuring both variable reordering and product term reordering, for area minimization. In addition, our algorithm takes those mandatory fabrication constraints into account in early stages for better outcomes. Experimental results show that our new method can achieve an area reduction of up to 24% as compared to current state-of-the-art techniques.**

Keywords—single-electron transistor; automatic synthesis; reconfigurable; area minimization; binary decision diagram.

I. INTRODUCTION

As manufacturing processes are constantly moving toward very deep submicron (VDSM) technology, the device feature size of CMOS technology is continuously scaling down. However, this trend also makes leakage power play a dominant role in system power dissipation [1]. To tackle the problem of leakage power, various emerging low-power devices have been developed in recent years. Among them, the single-electron transistor (SET) is regarded as one of the most promising devices since it can operate with only few electrons at room temperature [2][3][4].

One of the realizations of SET has been demonstrated as a three-terminal device that looks very similar to a classical metal-oxide-semiconductor field-effect transistor (MOSFET) [5][6]. The wrap-gates are utilized to control the nanowires. Several studies have demonstrated that a SET can be operated in open, Coulomb blockades, and short mode at room temperature [7][8][9]. However, a SET device is suffering from low transconductance and degraded output resistance since only few electrons are involved in a switching operation. Consequently, a SET-based circuit must be designed in conjunction with non-CMOS logic architectures. A binary decision diagram (BDD) [10] based logic structure has been proposed as a feasible approach for realizing logic functions using SETs [11]. Since BDD is an alternate representation of the truth table, any Boolean function can be implemented through a proper mapping onto a hexagonal nanowire network controlled by Schottky wrap-gates [12][13][14].

A BDD-based hexagonal nanowire network can be assembled using a set of node devices, and Fig. 1 illustrates the structure of a node device. Each node device has one entry branch and two exit branches; messenger electrons arrive via the entry branch and then leave through either the left exit branch or the right one depending on the control variable of the wrap-gate. An exit branch is a segment of SET-controlled nanowire, and has four operating modes in terms of its conductivity: *open*, *short*, *active-high*, and *active-low*. In a hexagonal network, each row is controlled by the same logic variable. Hence, with the help of node devices, the BDD representation of a given logic function can be easily mapped onto a hexagonal network according to its inherent structure. The first real BDD-based hexagonal nanowire network has already been demonstrated in [13]. Nevertheless, it is a custom implementation without reconfiguration capability. Besides, it also suffers from low yield due to the high defect rate of nanowire segments.

A reconfigurable SET array architecture using wrap-gate tunable tunnel barriers has been proposed to deal with these variability and reliability issues [15]. The architecture also presents two fabrication constraints, *granularity constraint* and *symmetric fabric constraint*. More technical details about the reconfigurable SET array will be reviewed in Section II. Given a Boolean function, the mapping orders of variables and product terms can significantly affect the final implementation on a SET array. Conventionally, the number of hexagons used during implementation is regarded as the required area. The first automated synthesis method targeting the reconfigurable SET array for area minimization was proposed in [16]. It merely deals with the ordering of product terms. However, experimental results show that the ordering of variables actually has a bigger impact on the optimization outcome than the ordering of product terms. An enhanced version of [16] was later presented in [17], and it thus takes the ordering of variables into account as well. Nevertheless, neither of the above two algorithms is aware of the two mandatory fabrication constraints before proceeding into the actual mapping stage. Therefore, in most cases, the actual mapping solution is far away from what is expected in earlier optimization stages. In addition, another synthesis approach tries to map a given function in a ladder shape to minimize the number of required hexagons [18]. However, consider the two different implementations of an identical design illustrated in Fig. 2, where the two implementations are assumed to consume the same number of hexagons but are in different widths. Note that two solutions must be in the same height since the height is solely determined by the number of variables. Since a mapped circuit is inherently bell-shaped in a SET array, those unused hexagons in the upper portion of one mapped function, in fact, cannot be utilized by other mapped functions either. Consequently, in our opinion,

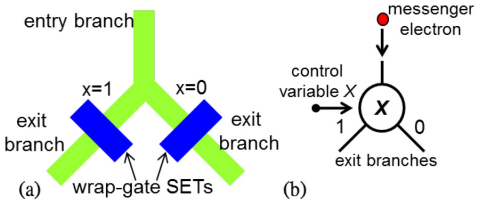

Figure 1. (a) Structural and (b) logical representations of a node device.

978-3-9815370-2-4/DATE14/©2014 EDAA

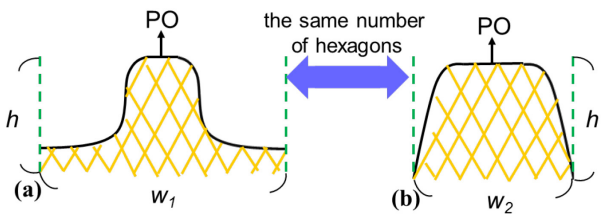

Figure 2. Different outlines of mapped circuits in SET arrays.

minimizing the width of the mapped circuit is more advantageous than minimizing the number of hexagons because it is obvious that the circuit width can better estimate the area than the hexagon count. Though the prior art [18] can effectively reduce the number of hexagons, the reduction in width is not that significant.

In this paper, we propose a synthesis algorithm for area minimization in terms of circuit width. Moreover, the two fabrication constraints are always considered throughout our approach as well. The key idea of the proposed algorithm is to maximize hexagon and path sharing during logic mapping. It consists of two primary stages: variable ordering and product term ordering. An input variable controls the behaviors (on/off) of all SETs at the same row in a SET array. Hence, at first, the proposed method gradually determines a proper variable ordering to maximize path sharing among product terms for width reduction. Next, product terms are mapped into a SET array sequentially, and the outcome is order dependent. The proposed product term ordering method can dynamically pick the best unmapped one that potentially achieves maximal path sharing with those mapped ones. Besides, the two fabrication constraints are constantly considered in both stages so that the mapping result is very close to what is estimated during optimization.

The rest of this paper is organized as follows. In Section II, we briefly introduce the architecture of reconfigurable SET array and give the problem formulation. Section III and Section IV propose our variable ordering and product term ordering methods for area minimization, respectively. Experimental results and analyses are then reported and discussed in Section V. Finally, a few concluding remarks are given in Section VI.

II. Preliminaries and Problem Formulation

The reconfigurable SET array is proposed to tackle the issues of variability as well as reliability [15]. Fig. 3(a) illustrates the structure of the target reconfigurable SET array, which can be further divided into three layers. The bottom layer is the device layer, where the regular hexagonal network is composed of identical node devices. The middle layer is used to configure the operation mode of every SET. Each edge (SET) of a node device can be freely configured to one of three modes: *active*, *short*, or *open*. Meanwhile, input signals are connected to SETs via the top layer, and control whether an active SET is on or off. Each row of SETs in the bottom layer is controlled by the same input.

In a fully-customized design, every SET can be configured to one of three operation modes independently. Nevertheless, to effectively reduce the array size, the target architecture imposes two limitations at fabrication time for a significant area reduction just at the cost of a bit flexibility loss [15]. Those two fabrication constraints are *granularity constraint* and *symmetric fabric constraint.* As mentioned, each SET can be freely configured in a customized design. However, in that case, each SET requires its own configuration wires, and those metal wires physically dominate the array size. To reduce the array size, the granularity constraint enforces that a pair of SETs within the same node device, named a *caret*, share the same set of configuration wires. As for the price, two SETs in a caret must operate in the same mode, which lowers the flexibility. Besides, to implement a BDD-like conditional branch using a caret, it is natural to control those two

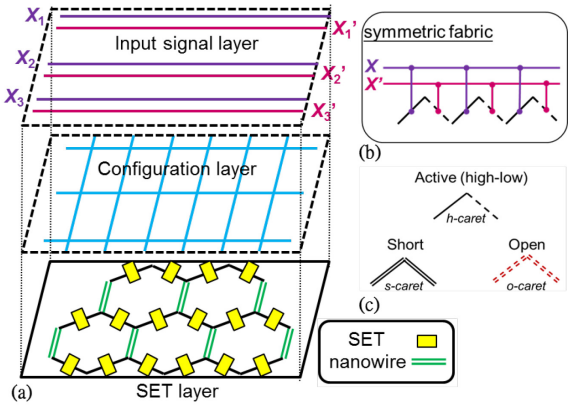

Figure 3. (a) Architecture of reconfigurable SET array, (b) symmetric fabric constraint, and (c) three types of carets.

SETs with a variable and its complement. Though each caret can have its own choice, it is actually an architectural decision since the hardware wiring must be fixed at fabrication time. A study examines several architectural alternatives and concludes that the symmetric fabric, as shown in Fig. 3(b), generally leads to a better synthesis outcome [15]. In fact, there are only three types of carets available under those two constraints, as depicted in Fig. 3(c). It is crucial that a synthesis algorithm should be fully aware of those two constraints, or the area minimization process is very likely to be misguided.

In this paper, we propose a fabrication-constraint-aware area-minimization synthesis algorithm for the reconfigurable SET array. More specifically, given a Boolean function represented by a set of disjoint product terms, the proposed constraint-aware algorithm can determine a proper ordering of variables and product terms such that the width of mapped circuit is minimized.

III. Variable (Column) Ordering

The synthesis process is to allocate a path for every product term one-by-one from a single current detector at the top to current sources at the bottom, and each level is controlled by a different input variable. The orders of variables and product terms do affect the mapping result. Hence, the key objective of the proposed algorithm is to achieve more hexagon and partial path sharing among various product terms through finding good variable and product term orderings.

A set of disjoint product terms can also be represented by a matrix, in which each row specifies a product term and each column represents a variable. Therefore, variable ordering is actually identical to column ordering in the matrix. The proposed fabrication-constraint-aware column ordering method can be further divided into two steps: *first column determination* and *next column determination*.

A. First Column Determination

There is a special concern while choosing a variable as the first column. Due to the granularity constraint, the first caret in a mapped path cannot be an s-caret, which implies the first column in a row cannot be a don't-care (–). If a row begins with a don't-care, it must be split into two rows starting with 1 and 0, respectively. Obviously, this should be avoided since more product terms generally lead to a bigger circuit. In addition, it is preferred that the counts of 1's and 0's in the first column are as unequal as possible because it generally implies most rows heads toward the same branch edge (1 or 0), which increases chances for further partial path sharing. Therefore, for each column c, our method first counts the numbers of 0's, 1's, and –'s, denoted as $n_0(c)$, $n_1(c)$, and $n_-(c)$, respectively. To reduce the count of –'s as well as to increase the unbalance between 0's and 1's, the *skew value* of column c is defined as $sv(c) = max(n_0(c), n_1(c)) - n_-(c)$. The column with the largest skew value is then selected as the first column.

B. Next Column Determination

In this step, the proposed method sequentially determines the next column k+1 one by one based on the previously determined column k. Bit values of two columns k and k+1 in a same row form a bit pair. Since each bit can be 0, 1, or –, there are nine possible combinations for a bit pair. We intend to figure out the relationship between the bit pairs of two consecutive mapped rows and evaluate how good the possibility of partial path sharing is for each bit pair. Due to the symmetric fabric constraint, invalid paths may be accidentally created during mapping. To correct such kind of errors, *expansions* are hence inevitably required. However, those expansions apparently increase the width of mapped circuit. Consequently, the objective of this step is to find a proper variable as the next column so that the need for expansions can be greatly minimized.

The term $b(i, j)$ is defined as the bit value of row i and column j in the matrix representation of a logic function. Similarly, $bp(i, j, k)$ represents the bit pair of column j and k in row i. Given two bit pairs bp_1 and bp_2, the possible situations between them during the mapping process can be classified into three categories: a) expansion is always required, b) expansion may or may not be required, and c) expansion is never required. A function $score(bp_1, bp_2)$ is therefore defined to assign appropriate scores (0~2) based on the above classification. It represents the *safety score* between two associated bit pairs. A score of 2 guarantees that the bit pair bp_2 never causes an expansion with respect to bp_1; a score of 1 implies that bp_2 occasionally causes an expansion with respect to bp_1; similarly, a score of 0 indicates that bp_2 always causes an expansion with respect to bp_1. In order to evaluate how good the chance for partial path sharing between two columns p and q, the *neighborhood gain* between them is further defined as:

$$ng(p,q) = \sum_{1 \le x < y \le |R|} score(bp(x,p,q), bp(y,p,q)), \qquad (1)$$

where $|R|$ is the total number of rows (product terms). A high value of $ng(p, q)$ suggests column q is generally a very good neighbor for column p in terms of area minimization. The reason for summing up scores for all possible row pairs is that the row ordering is not determined yet at this point. With the help of $ng(p, q)$, the proposed method can gradually complete the column ordering by selecting only one most appropriate variable at a time. Assume variable p has just been selected as column k, $k \ge 1$, and Q is the set containing those variables q that have not been selected yet. Our method examines all the neighborhood gains between p and every candidate variable q and then picks the one with the highest gain as column k+1. This process is not terminated until the complete column ordering is finalized.

IV. Product Term (Row) Ordering

In this section, we present a dynamic approach to determine the row ordering. Since the synthesis process maps those product terms to a SET array in downward order starting from the top one, the order of rows in a matrix is actually the mapping order of product terms during implementation. Most of previous works use static heuristic approaches to determine the row ordering. In contrast, our approach adopts a dynamic iterative approach instead. At each iteration, our method dynamically just selects the next mapping row that has the most potential for circuit width reduction based on the current intermediate mapping result. Hence, our on-the-fly ordering strategy can generally achieve a better mapping outcome.

Due to the fabrication constraints, two branch edges in a caret must be configured simultaneously in the mapping process. However, only one of them is occasionally required to form the path for the target product term. Hence, the key idea of the proposed method is to utilize those configured-but-not-in-use edges, created in previous iterations, for circuit width minimization. Similarly, the proposed dynamic row ordering algorithm can be further divided into two steps: *first row determination* and *next row determination*.

A. First Row Determination

An interesting observation from the mapping outcome is that don't-cares in different positions of a product term make different degrees of impact on the mapping area; in general, the more forward the don't-care, the larger the area. To summarize the overall impact for a product term p, the *don't-care weight* of p, $dw(p)$, is defined as:

$$dw(p) = \sum_{bit\ j\ is\ a\ don't-care} bpw(j). \qquad (2)$$

$bpw(j)$ represents the *bit position weight* of the j^{th} bit and is defined as $bpw(j) = |C| - j + 1$, where $|C|$ denotes the number of columns (variables). After calculating don't-care weights for all rows, the row with the smallest weight is thus selected to be the first for mapping.

B. Next Row Determination

As aforementioned, two branch edges of a caret are configured simultaneously during mapping, but only one is required occasionally. Hence, there are many unterminated paths, named *dangling paths*, in the middle of mapping process. For the sake of area minimization, it would be a big plus if the upcoming path for the next product term can utilize one of those dangling paths (i.e., path sharing). Consequently, the proposed method keeps track of all dangling paths throughout the mapping process and records those path prefixes into a *prefix table*. At each iteration, our method examines if there are matches between unmapped rows and known path prefixes in decreasing order of path length. Once a match is found, our method further checks whether the matched row can be successfully mapped without introducing extra expansions. If it is the case, that matched row would be formally accepted for mapping. Unfortunately, if all unmapped rows fail to pass the above qualification, the one with the smallest don't-care weight would be selected for mapping instead. At the end of the iteration, the prefix table is updated according to the current mapped circuit. This process is not terminated until all rows are mapped.

At the end of this section, Fig. 4 illustrates the overall flow. It takes a set of disjoint sum-of-product terms as input at the beginning. Next, it gradually completes the variable ordering for circuit width minimization under the two mandatory fabrication constraints. Then, it adopts an on-the-fly strategy to maximize path sharing while dynamically mapping product terms into a SET array one by one.

V. Experimental Results

The proposed algorithm has been implemented in C++/Linux. To evaluate the proposed method we compare it against two prior synthesis techniques [17][18]. A set of 21 test cases are selected from the MCNC benchmark suite [19] for the experiments and the results are presented in Table 1. The first column lists the names of test cases; the second and the third give the numbers of primary inputs and primary outputs, respectively. The forth column shows the number of disjoint product terms of a test case, which is obtained from its ROBDD [20]. We have re-implemented the method proposed in [17] because no width data are reported in [17]. Meanwhile, the results of the approach presented in [18] are directly quoted since the width data are already available. As mentioned previously, circuit width should be a better indicator for the required area than the hexagon count. Hence, the circuit width would be the primary comparison factor here though the number of hexagons is also reported for reference. Note that both of the methods in [17] and [18] focus on hexagon minimization rather than circuit width.

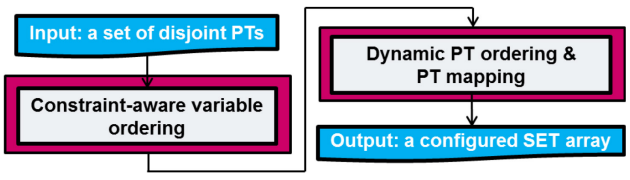

Figure 4. The overall flow of the proposed algorithm.

The overall experimental results show that our approach can achieve a reduction of circuit width by 19% and 24% on average as compared to [17] and [18], respectively. Furthermore, our method performs better than [17] in both circuit width and hexagon count. The comparisons between our method and [18] are a bit interesting. The work [18] tries to minimize the number of required hexagons by mapping the circuit in the fashion illustrated in Fig. 2(b). Though that strategy can effectively decrease the hexagon count, it also leads to an increase of the circuit width. Both of the comparisons between our method and [18] as well as between [17] and [18] draw the same conclusion. Therefore, it is no doubt that our method should be considered better in terms of area (width) minimization. Regarding the runtime efficiency, our method can finish the synthesis task in one second for each of test cases. It confirms that the better synthesis outcome does not come from the increase of runtime. Consequently, it is conclusive that the proposed algorithm can efficiently offer a better area-minimized synthesis solution than the prior arts.

VI. CONCLUSION

In this paper, we address the issue of synthesis for the reconfigurable SET array. We first point out that the circuit width can better estimate the actual area required by a mapped circuit than the conventionally-used hexagon count. We also propose a synthesis algorithm for area minimization. The algorithm is fully aware of the two mandatory fabrication constraints, granularity constraint and symmetric fabric constraint, throughout the entire synthesis process. It consists of two major phases, the constraint-aware column ordering for width reduction with the help of skew value and neighborhood gain, as well as the dynamic row ordering for path sharing by means of don't-care weight and prefix table. The experimental results demonstrate that the proposed method achieves 19% and 24% reduction of circuit width as compared to [17] and [18], respectively. Consequently, it is convincing that our approach should be a better solution for area minimization synthesis targeting the reconfigurable SET array.

REFERENCES

[1] ITRS, Semiconductor Industry Association, 2006.

[2] H. W. Ch. Postma *et al.*, "Carbon nanotube single-electron transistors at room temperature," *Science*, vol. 293, pp. 76–79, Jul. 2001.

[3] Y. T. Tan *et al.*, "Room temperature nanocrystalline silicon single-electron transistors," *JAP*, vol. 94, pp. 633–637, Jul. 2003.

[4] L. Zhuang *et al.*, "Silicon single-electron quantum-dot transistor switch operating at room temperature," *APL*, vol. 72, pp. 1205–1207, Mar. 1998.

[5] P. S. K. Karre *et al.*, "Room temperature operational single electron transistor fabricated by focused ion beam deposition," *JAP*, vol. 102, pp. 024316–024316-4, 2007.

[6] S. Kasai *et al.*, "GaAs Schottky wrap-gate binary-decision-diagram devices for realization of novel single electron logic architecture," *in Proc. IEDM*, 2000, pp. 585–588.

[7] L. Liu *et al.*, "Device circuit co-design using classical and non-classical IIIV multi-gate quantum-well FETs (MuQFETs)," *in Proc. IEDM*, 2011, pp. 4.5.1–4.5.4.

[8] K. Yano *et al.*, "Room-temperature single-electron memory," *IEEE TED*, vol. 41, pp. 1628–1638, Sep. 1994.

[9] K. Uchida *et al.*, "Programmable single-electron transistor logic for low-power intelligent Si LSI," *in Proc. ISSCC*, 2002, pp. 206.

[10] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE TC*, vol. 35, pp. 677–691, Aug. 1986.

[11] N. Asahi *et al.*, "Single-electron logic device based on the binary decision diagram," *IEEE TED*, vol. 44, pp. 1109–1116, Jul. 1997.

[12] H. Hasegawa *et al.*, "Hexagonal binary decision diagram quantum logic circuits using Schottky in-plane and wrap gate control of GaAs and InGaAs nanowires," *Physica E*, vol. 11, pp. 149–154, Oct. 2001.

[13] S. Kasai *et al.*, "Fabrication of GaAs-based integrated 2-bit half and full adders by novel hexagonal BDD quantum circuit approach," *in Proc. ISDRS*, 2001, pp. 622–625.

[14] S. Kasai *et al.*, "A single electron binary-decision-diagram quantum logic circuit based on Schottky wrap gate control of a GaAs nanowire hexagon," *EDL*, vol. 23, pp. 446–448, Aug. 2002.

[15] S. Eachempati *et al.*, "Reconfigurable BDD-based Quantum Circuits," *in Proc. NANOARCH*, 2008, pp. 61–67.

[16] Y. C. Chen *et al.*, "Automated mapping for reconfigurable single-electron transistor arrays," *in Proc. DAC*, 2011, pp. 878–883.

[17] Y. C. Chen *et al.*, "A synthesis algorithm for reconfigurable single-electron transistor arrays," *ACM JETC*, vol. 9, no. 1, article 5, Feb. 2013.

[18] C. E. Chiang *et al.*, "On reconfigurable single-electron transistor arrays synthesis using reordering techniques," *in Proc. DATE*, 2013, pp. 147–152.

[19] S. Yang, "Logic synthesis and optimization benchmarks, Version 3.0," MCNC, Research Triangle Park, NC, Tech. Rep., 1991.

[20] F. Somenzi, CUDD: CU decision diagram package - release 2.4.2, 2009. http://vlsi.colorado.edu/~fabio/CUDD

Table 1. Comparisons among the proposed method, [17], and [18].

Benchmark				Proposed Method			[17] (re-implemented)			[18] (quoted)		
Name	PI	PO	PT	Width	N_{hex}	Runtime (s)	Width	N_{hex}	Runtime (s)	Width	N_{hex}	Runtime (s)
c17	5	2	8	22	45	0.00	29	83	0.00	31	54	0.09
cm138a	6	8	48	129	332	0.01	310	1029	0.00	199	460	0.10
x2	10	7	40	122	498	0.02	170	850	0.00	134	397	0.10
cm151a	12	2	17	75	372	0.00	112	668	0.00	109	521	0.09
cm162a	14	5	41	152	849	0.01	156	1045	0.00	156	578	0.09
cu	14	11	23	95	285	0.01	96	512	0.00	103	415	0.09
cmb	16	4	26	55	80	0.00	111	695	0.00	122	376	0.09
cm163a	16	5	31	114	647	0.02	129	939	0.00	113	391	0.09
pm1	16	13	49	136	591	0.02	164	1085	0.00	156	586	0.10
pcle	19	9	45	116	581	0.02	164	1323	0.00	183	751	0.10
sct	19	15	153	439	3521	0.05	608	5544	0.02	620	3168	0.11
cc	21	20	53	193	1180	0.01	206	1793	0.00	191	1040	0.09
i1	25	16	38	97	322	0.01	168	1617	0.01	159	1190	0.09
lal	26	19	171	543	5455	0.06	591	6883	0.04	613	3312	0.11
pcler8	27	17	67	234	1756	0.04	256	2713	0.01	315	1920	0.10
c8	28	18	85	249	2208	0.05	298	3465	0.02	427	2026	0.11
count	35	16	200	444	3844	0.07	754	11329	0.05	755	4590	0.15
unreg	36	16	48	194	2424	0.03	213	3061	0.01	257	1515	0.11
b9	41	21	376	1190	20674	0.26	1375	26003	15.20	1520	9112	0.24
cht	47	36	81	340	3528	0.06	345	6206	0.02	349	3556	0.13
example2	85	66	447	1144	23964	0.49	1280	42332	0.66	1517	14402	0.50
Total	**-**	**-**	**-**	**6083**	**73156**	**1.24**	**7535**	**119175**	**16.04**	**8029**	**50360**	**2.68**