Parallel Circuit and Interconnect Simulation Using Multi-core PC

Chun-Jung Chen
Department of Computer Science
Chinese Culture University
Taipei, Taiwan
teacherchen@faculty.pccu.edu.tw

Tien-Hao Shih
Department of Mechanical Engineering
Master's Program of Digital Mechatronics, Chinese Culture University
Taipei, Taiwan

***Abstract*—This paper presents methods that utilize multi-core PC to perform MOSFET circuit simulation and transmission line calculation. A very coarse-grained parallel computing strategy is proposed for circuit simulation. A parallel transmission line calculation method based on Method of Characteristic is also described. All proposed methods have been implemented and tested. Experimental results justify pleasing effects of proposed methods.**

Keywords-circuit simulation; parallel computing; transmission line; relaxation-based; simulation-on-demand

I. INTRODUCTION

Undertaking circuit simulation is very important in IC design community. Accurate approaches to solve circuit simulation problem include the "standard" simulators (such as SPICE) and relaxation-based simulators (such as RELAX [1], and SPLICE [2]). There exist faster simulation approaches using simpler simulation models (such as piecewise-linear model and switch-level model). But they only provide coarse waveforms. The trade-off between the solution accuracy and simulation speed firmly exists. Also, it is well known that delays of interconnects play a very important role today. There exist many researches, e.g. [3, 4], discussing calculations of transmission lines, but few of them consider the situation of large-scale simulation, in which the flexibility and efficiency of transmission line calculations are very important.

In this paper, we propose PC-based parallel-computing strategies to raise the calculation efficiency directly, and then break the trade-off mentioned above. Since multi-core PCs are very popular today, this paper has practical values.

We propose parallel-computing strategies for MOSFET circuit simulation and transmission line calculation respectively. In the first subject, we ask various processors to simulate different portions of the simulated circuit, and then combined waveforms derived by various processors. We call this strategy the *Combining Simulation Method* (*CSM*). *Backward-traversing Waveform Relaxation* (*BTWR*) [5] is a specialized algorithm that simulates subcircuits by traversing subcircuits from the rear end to front end backwardly. So, it has the function of "simulation-on-demand" (SOD), i.e. only simulate subcircuits contributing to wanted outputs. Implementing CSM, we find that there are many methods to divide the simulated circuits into portions. We will use SOD of BTWR to divide.

In the second subject, we exploit [3]. We call this method *FMOC* (Fast method based on the Method of Characteristic) in this paper. FMOC is flexible enough to cooperate with Fast SPICE. It transforms transmission lines into time-domain equivalent circuit elements and calculate their parameter values at (FMOC's) "inner" time points. We will show that calculations on inner time points can be computed in parallel.

The solving strategies for the two designated subjects of this paper all have been implemented. Experiments are made to justify their effects. The outline of this paper is as follows. In Section 2, the methods for solving first subject are explained, including BTWR algorithm and the Combining Simulation Method. In Section 3, we illustrate FMOC and the related parallel computing. Section 4 then shows experimental results to illustrate the effeteness of proposed methods. Finally, conclusions are made in Section 5.

II. BTWR AND COMBINING SIMULATION METHOD

A. BTWR Algorithm

The fundamental circuit simulation algorithm of our work is BTWR, which is relaxation-based [1, 2]. The two famous algorithms of this class of algorithms are WR (Waveform Relaxation) and ITA (Iterated Timing Analysis) [1, 2]. BTWR is a more complicated algorithm. In fact, it collects advantages of WR and ITA and performs circuit simulations efficiently and stably [5]. We describe mathematic equations now. The simulated circuit can be described as following time-varying differential equation:

$$F(Y(t), \dot{Y}(t), t) = 0 \tag{1}$$

Where Y is the vector of circuit variables, t is the time, F is a continuous function and "." means differentiation with respect to time. The simulated circuit is partitioned into subcircuits, and the ith subcircuit is:

$$F_i(Y_i(t), \dot{Y}_i(t), D_i(t), \dot{D}_i(t), t) = 0 \tag{2}$$

This equation, a, can be expressed as the abbreviate form:

$$f(y(t), \dot{y}(t), w(t), \dot{w}(t), t) = 0 \tag{3}$$

Where y (Y_i, a sub-vector of Y) is the vector of circuit variables in a, w (D_i, the *decoupling* vector) is the vector of circuit variables not in a, and f is a continuous function. In this paper, a *subcircuit calculation* (used as performance index) means the computation efforts to solve (3) for $y(t_{n+1})$ (t_{n+1} is current time point), which include applying integral formula (such as Trapezoidal method) to (3), and solving the derived nonlinear algebraic equations by Newton's iteration.

The basic idea of BTWR is to consider the cause-and-consequence concept. The left part of Fig. 1 is a signal flow graph for partitioned subcircuits, in which the transient solution of subcircuit a is obviously the consequence of transient solutions of b and c. Therefore, b and c need to be solved before a to raise the computation efficiency. To trace these cause-and-consequence relations, we use the backward graph traversal technique. Introducing more clearly, we define some variables inside each subcircuit: t_c is the time point for which the subcircuit has converged so far, t_{now} is the current time point to be solved, and t_a (time arrived) is the time boundary at which the subcircuit is asked to be solved. In Fig. 1, the traversal starts from subcircuit a (tries to solve for y at $a.t_{now}$). The traversal visits a's fan-in subcircuit b at first and ask it to be solved at time point later than $b.t_a$ (which is also $a.t_{now}$) in order to provide waveform references for subcircuit a. The traversal then continually visits c and asks it to be solved at time point later than $c.t_a$ (which is also $b.t_{now}$) for the same sake. Subcircuit c needs to forward two time points to move its t_{now} to over $c.t_a$. In Fig. 1, the actual subcircuit calculation sequence would be: c be solved at its two t_{now} time points, b be solved at its t_{now} time points, and then a be solved at its t_{now} time points. This process might repeat several times until $a.t_{now}$ is converged. The job of main program of BTWR is simple. It just picks subcircuit with smallest t_{now}, activates the "starting" backward traversal (called Mode-0 traversal) from it, and repeats the same process until no subcircuit left.

There are software schemes [5] to handle the feedback subcircuits (including adjacent coupling and global feedback loops) to strengthen the robustness of BTWR. BTWR exhibits several advantages. First, the multi-rate behaviors of circuits can be exploited. Second, the windowing technique [1] is automatically applied. Third, the function of selective-tracing scheme of ITA [2] (to calculate connected subcircuits) is retained. Finally and most important for this paper, it's easy to implement high quality SOD on BTWR. BTWR is represented in the pseudo codes in Algorithm 1. Note that SOD function is built in lines labeled by "Sod1" and "Sod2."

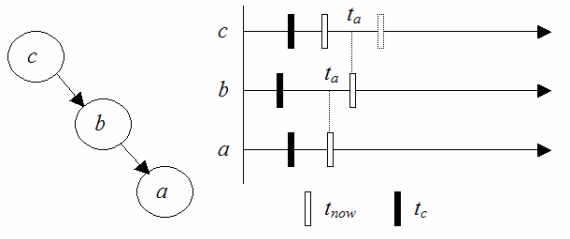

Figure 1. A traversal starting from subcircuit a. Subcircuit b and c are asked to be calculated to time point later then t_a.

Algorithm 1 (BTWR-based Circuit Simulation):

```
// ckt is the simulated circuit partitioned into N subcircuits.
// Simulation duration is T_begin ~T_end
BTWR(ckt, T_begin, T_end) {
  Set t_c, t_now of all subcircuits to their initial values;
  while(there is any subcircuit whose t_c is not equal to T_end) {
    Pick the subcircuit x with smallest t_now;
Sod1:
    if(! contribute[x]) continue;  // the SOD function
    BTWRtrace(0, x.t_now, x);  // begin to call the traversal
  }
}

BTWRtrace(mode, t_a, sub) {
    // sub.in_stack array records whether sub has been traversed
    if(mode is 0) Clear all subcircuits' in_stack flag, t_ever = 0;
    else if(mode == 1) sub.in_stack = 1;
    if(mode is 0) Clear GFL; // the set containing subcircuits of GFLs
    dt_now = sub.t_now; // the old t_now
    do {
      for(all sub's fan-in subcircuit x) {
        // backwardly traverse all predecessors
        if(x is strongly coupled with sub) synchronize x and sub;
Rtr:    if(! sub.in_stack) BTWRtrace(1, sub.t_now, x);
Loop:   else { // has encountered a back edge
          Add subcircuits from sub to x (in the recursion queue)
            into GFL;
        }
      }
S1:   if(sub is not in GFL) { // simulate sub or all subcircuits in GFL
        Solve (3) of sub at sub.t_now;
        if(Newton iteration diverges or solution quality is bad)
          reduce t_now;  // shrink the time step
        else if(results have been converged) {
          t_ever = MAX(t_ever , sub.t_c);
          sub.t_c = sub.t_now;
          Estimate new sub.t_now;
        }
      }
      // simulate GFL
S2:   if(sub is the first subcircuit of GFL) {
        Simulate GFL by using WR algorithm;
        break;
      }
Sod2: contribute[sub] = true;  // the SOD function
Stop1: if(mode is 0 && sub.t_c >= dt_now) break;
Stop2: else if(mode is 1 && t_ever >= t_a) break;
    } while (true);
}
```

B. *The Combining Simulation Method*

In using relaxation-based algorithm, there exist many strategies to utilize the parallelism. These strategies can be classified into space, temporal, and iteration respects [6]. The Combining Simulation Method is in the space respect. The basic idea is to use "client" processors to simulate different portions of the simulated circuit, and then combine the obtained "client-waveforms," which is illustrated in Fig. 2. In this figure, there is a master-simulation that analyzes the simulated circuit, divides the circuit, sends the divided portions to client-simulations (which is simulated by one single processor), waits for the end of client-simulations, and then combines client-waveforms. The client-simulation just simulates portions of the analyzed circuits and generates portions of the wanted waveforms.

For the success of CSM, dividing the simulated circuit is a critical step. The divided portions of the simulated circuit

should be independent or the client-waveforms would be inaccurate. It is possible to undertake waveform relaxation between client-simulations to achieve the convergence of client-waveforms [6], but we don't consider this complex process in this paper. Moreover, we don't really divide the simulated circuit. We exploit the SOD ability of BTWR. Our dividing method is to divide the list of wanted outputs and send them to client-simulations, while each client-simulation simulates the same circuit. This strategy is simple and trustable, since each client-simulation simulates the entire circuit by using SOD, in which the obtained client-waveforms are accurate and no waveform relaxation processes are needed.

To derive better efficiency of CSM, client-simulations need to exhaust roughly the same amount of CPU time and simulate as few overlapping portions of the simulated circuit as possible. These necessities can be taken cared by considering the "quality" of dividing of the list of wanted outputs. The criterion for dividing is to put outputs of the same independent portion of the simulated circuit together such that they are computed by the same client-simulation. Since SOD is used, the client-simulation will only simulate the related independent portion of the simulated circuit, and hence save the simulation time. To accomplish this dividing criterion, we need to analyze the simulated circuit. Because the relaxation-based algorithm (BTWR) is used, the simulated circuit has been partitioned into subcircuits. We can utilize the signal flow graph of subcircuits to do such analysis, e.g. traversing the signal flow graph from the wanted outputs backwardly to see the "contributing" subcircuits. We have designed an automatic dividing subroutine to divide the list of wanted outputs.

We note that CSM is a very coarse-grained strategy for parallel circuit simulation, in which several circuit simulators execute at the same time. Therefore, there is no necessity to rewrite any code of circuit simulators. Moreover, various simulators can be used to purchase better simulation results, e.g. use SPICE to simulate analog portions, and use Fast SPICE to simulate digital portions. CSM is also a high level algorithm that omits many details. So, it can be used in single computer that has many cores or many computers (having one or several cores) on networks. In the latter case, CSM constructs the distributed circuit simulation. In this paper, we just implement CSM in the multiple-core PC and only use our simulator (MOSTIME). Experiments will be described later.

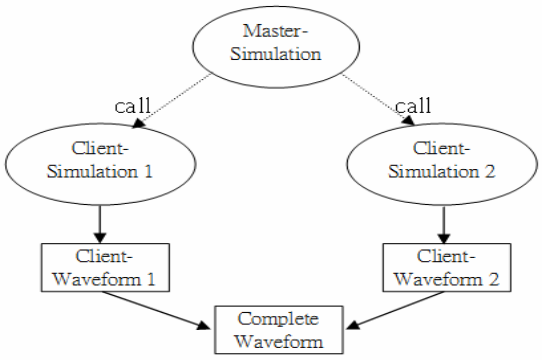

Figure 2. An example of Combining Simulation Method, in which only two processors (and hence two client simulations) are shown.

III. FMOC AND ITS PARALLELISM

A. The FMOC Method

The method to transform transmission lines is based on the Method of Characteristic [3]. Consider a coupled uniform transmission line whose resistance, inductance, capacitance, and conductance per unit length are R, L, C, and G respectively. The electrical behaviors (voltages v, and currents i) are described by following Telegraph's equation:

$$\frac{\partial}{\partial x}v(x,t) = -L\frac{\partial}{\partial t}i(x,t) - Ri(x,t) \tag{4}$$

$$\frac{\partial}{\partial x}i(x,t) = -C\frac{\partial}{\partial t}v(x,t) - Gv(x,t) \tag{5}$$

Where x denotes distance, t denotes time. These equations are partially decoupled (on the L and C sense only) by (6) to derive (7) and (8):

$$\begin{aligned} v(x,t) &= Xu(x,t), \\ i(x,t) &= (X^T)^{-1} j(x,t) \end{aligned} \tag{6}$$

$$\frac{\partial}{\partial x}u(x,t) = -\hat{L}\frac{\partial}{\partial t}j(x,t) - \hat{R}j(x,t) \tag{7}$$

$$\frac{\partial}{\partial x}j(x,t) = -\hat{C}\frac{\partial}{\partial t}u(x,t) - \hat{G}u(x,t) \tag{8}$$

Where X is the eigenvector matrix of LC, $\hat{L} = X^{-1}L(X^T)^{-1}$, $\hat{C} = X^T CX$, $\hat{R} = X^{-1}R(X^T)^{-1}$, and $\hat{G} = X^T GX$. Note that both $\hat{L}$ and $\hat{C}$ are diagonal matrices. We consider the kth equation of both (7) and (8):

$$\frac{\partial u_k(x,t)}{\partial x} + \hat{L}_k\frac{\partial j_k(x,t)}{\partial t} = -\sum_{l=1}^{N}[\hat{R}_{k,l}j_l(x,t)] \tag{9}$$

$$\frac{\partial j_k(x,t)}{\partial x} + \hat{C}_k\frac{\partial u_k(x,t)}{\partial t} = -\sum_{l=1}^{N}[\hat{G}_{k,l}u_l(x,t)] \tag{10}$$

Where $1 \le k \le N$, and N is the number of lines of this transmission line. Now we apply the Method of Characteristic. We use the equations $\frac{dx}{dt} = \gamma_k$ and $\frac{dx}{dt} = -\gamma_k$ (where $\gamma_k = (\hat{L}_k\hat{C}_k)^{-0.5}$) to define the characteristic lines in the x-t plane, which are called the kth characteristic α and β lines respectively. Consider the differentiating along the kth characteristic α line:

$$\begin{aligned} &\frac{d^{\alpha}}{dt}[u_k(x,t) + Z_k j_k(x,t)] = \\ &-\sum_{t=1}^{N}[\gamma_k\hat{R}_{k,l}j_l(x,t)] - \sum_{l=1}^{N}\frac{\hat{G}_{k,l}u_l(x,t)}{\hat{C}_k} \end{aligned} \tag{11}$$

, in which $Z_k = (\hat{L}_k / \hat{C}_k)^{0.5}$ is the characteristic impedance of the kth transmission line. Similarly, along the kth characteristic β line, we have:

$$\frac{d^{\beta}}{dt}[u_k(x,t) - Z_k j_k(x,t)] = \\ + \sum_{l=1}^{N}[\gamma_k \hat{R}_{k,l} j_l(x,t)] - \sum_{l=1}^{N} \frac{\hat{G}_{k,l} u_l(x,t)}{\hat{C}_k} \tag{12}$$

Now, solve (12) by Forward Euler (FE) integral method:

$$u_k(t_n) - Z_k j_k(t_n) = u_k(t_{n-1}) - Z_k j_k(t_{n-1}) \\ + \Delta t_i \sum_{l=1}^{N}[\gamma_k \hat{R}_{k,l} j_l(t_{n-1})] - \Delta t_i \sum_{l=1}^{N} \frac{\hat{G}_{k,l} u_l(t_{n-1})}{\hat{C}_k} = V_{0,k} \tag{13}$$

Note that both *x*-axis and *t*-axis have been divided into segments (Δt_i is the segment length of *t*-axis) in order to use (13). The *x*-*t* planes of all lines are divided in the same Δt and Δx. FMOC uses $1/\max\{\gamma_k | \ k=1\sim N\}$ as the slope in the *x*-*t* plane, shown in Fig. 3. The denser and sparser grids are drawn in Fig. 2 together, in which the *u*/*j* values of point *a* are derived by referring point a_p and a_f, and those of point *b* are derived by referring point b_p and b_f. Note the horizontal segment numbers are reversely proportional to the size of Δt_i.

Equation (13) is expended into matrix form to represent *N* lines (now *i*, *j*, *v*, and *u* are represented in upper case):

$$U(t_n) - ZJ(t_n) = V_0 \tag{14}$$

Then, by substituting (6) into (14) we get:

$$I(t_n) = G_0 V(t_n) + I_0 \tag{15}$$

, where $G_0=(X^T)^{-1}Z^{-1}X^{-1}$, and $I_0=-(X^T)^{-1}Z^{-1}V_0$. Equation (15) represents the $x = 0$ end terminal equivalent circuit of this transmission line. The $x = D$ (*D* is the length of this transmission line) end terminal equivalent circuit can be obtained by processing (11) similarly. The result is:

$$u_k(t_n) + Z_k j_k(t_n) = u_k(t_{n-1}) + Z_k j_k(t_{n-1}) \\ - \Delta t_i \sum_{l=1}^{N}[\gamma_k \hat{R}_{k,l} j_l(t_{n-1})] - \Delta t_i \sum_{l=1}^{N} \frac{\hat{G}_{k,l} u_l(t_{n-1})}{\hat{C}_k} = V_{D,k} \tag{16}$$

$$-I(t_n) = G_D V(t_n) + I_D \tag{17}$$

, in which $G_D = G_0$, $I_D = -(X^T)^{-1}Z^{-1}V_D$. Equations (15) and (17) are the time-domain equivalent circuits of a transmission line to be used in circuit simulation.

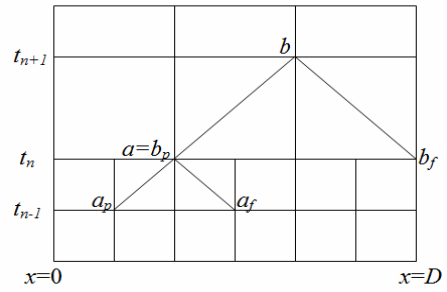

Figure 3. Space (horizontal) and time (vertical) grids for simulating a transmission line. Sparser and denser grids are shown together.

B. Parallel Computing in FMOC

Multi-core machines are popular recently. To exploit parallelism of FMOC is a direct way to speedup the simulation. Equations (13) and (16) have to be solved for *M* times, one of which represents one point on the *x*-axis of Fig. 3, with respect to an inner time point. Each solving process for both equations is independent to those of other inner time points. So, parallel computing can be exploited here. Algorithm 2 represents the pseudo code, in which OpenMP API is utilized to realize parallel computations. The parallelism is small-grained. Therefore, our circuit simulator, MOSTIME [5], needs to be carefully rewritten. The major concern is that codes computing (13) and (16) should use shared memories carefully.

Algorithm 2 (Parallel Computing in FMOC):

```
#pragma omp parallel for
for(s = 1; s < M; s++) {
    Solve (13) and (16) for the sth point on the x-axis;
}
```

In [4], Waveform Relaxation methods for the longitudinal partitioning of transmission lines are investigated, in which parallel computing is applied on the WR iterations. However, our method is more robust, and its performance will be shown in next section. Our method shows very good parallel-computing efficiency (approximates to single-cored machine).

IV. EXPERIMENTAL RESULTS

We have implemented all proposed methods in our simulator MOSTIME, and run it on multi-core PCs. CSM is implemented in a single PC, in which several MOSTIME will run at the same time. The master-simulation activates client-simulations and then combines client-waveforms, in which the passing of simulated circuits (which is described in the "deck" file) and retrieving of client-waveforms all use the file system of Windows. Note that when parallel FMOC runs, several processors would be activated in one MOSTIME. So, to avoid "processor competition," CSM and parallel FMOC are tested separately. The experimental results are machine-dependent. Our practical environment includes: Visual Studio 2008, OpenMP API, Intel Core 2 Duo CPU, and Intel Core i7 CPU.

At first, we check the effect of CSM. Several circuits have been simulated, and results are listed in Table 1. The two types of circuits are inverter chain and ALU (whose schematics are given in Fig. 4) chain. Waveforms of a tested circuit generated by BTWR and CSM are compared in Fig. 5, which shows that our implementation is correct. There are several independent portions in these circuits, e.g. "inv100x2" has two independent inverter chains. The number of cores is specified manually, depending on the number of independent portions. Numbers of outputs are important in BTWR-based CSM, since SOD is used. The list of wanted outputs is partitioned automatically in each CSM simulation. Three algorithms are tested for each circuit, which are BTWR, BTWR plus SOD and BTWR-based CSM. The used CPU times of simulations are listed. The column labeled with "IO" is the time for combining client-waveforms, which can be referred to know the amount of overheads for processing client-waveforms. In the two right-most columns are speedup (compared with BTWR) and efficiency of CSM. Note that efficiency is defined as follows:

$$\eta = \frac{T(BTWR)}{T(CSM) * Core\#} \quad (18)$$

In which $T(x)$ is the used CPU time of algorithm x. We can observe obvious performance enhancements. Note that last two circuits have not simulated well by BTWR due to the lack of memory.

The efficiencies of parallel computing are not good in some circuits, e.g. in the last circuit, only 46% of efficiency is recorded. We find that one client-simulation, which needs to compete for "global resources" of the same PC (such as the right to access disc and global main memory, and so on) with other client-simulations, spends more simulation time than normal simulation processing the same circuit. Therefore, to use network of PCs might be an improvement method.

Next, we check the effect of parallel FMOC. We run some circuits with many transmission lines and record the used CPU time in Table 2. The more popular ITA algorithm is used. The transmission line's parameters are: 10cm long, R=75 Ω/m, L=500nH/m, C=63pF/m, G=50m Ω^{-1}/m, and M=500. Transmission lines have been added to each output of inverter (or RC), and each circuit runs for three clocks. The used CPU has two cores. Speedups and efficiencies are good. Using parallel computing in FMOC is quite successful.

V. CONCLUSION

In this paper, we have presented two techniques to utilize the popular and powerful multi-core PC. The first technique is CSM, and the second one is parallel FMOC. Real implementations on multi-core PC have been tested. Experimental results justify that proposed techniques provide good parallel-computing efficiencies. The more complicated and better partitioning method for CSM and using computer networks might be our future works.

TABLE I. CPU TIME COMPARISON FOR PARALLEL CIRCUIT SIMULATIONS

Ckt.	Used CPU Time*				Output #	Core #	Speedup	η $
	BTWR	+SOD	CSM	IO				
inv100x2	9.438	9.613	6.475	0.187	2	2	1.4	0.72
inv100x4	19.23	18.68	8.783	0.359	3	4	2.1	0.54
alu4x2	10.26	9.454	7.099	0.53	8	2	1.4	0.72
alu2x4	9.064	7.566	4.04	0.577	8	4	2.2	0.56
alu32x2	N. A.	14.384	10.93	1.029	8	2	1.3	0.65
alu16x4	N. A.	14.025	7.472	1.264	8	4	1.8	0.46

*: The used CPU is Intel Core i7 (1.73 GHZ) that has eight cores

$: The efficiency of parallel computing

TABLE II. CPU TIME* COMPARISON FOR PARALLEL FMOC

Ckt.	Tr. Line#	ITA	Parallel FMOC	Speedup	η
RC	1	7.00	3.76	1.86	0.93
1-stage Inverter	1	11.13	7.34	1.51	0.75
4-stage Inverter	4	54.30	27.40	1.98	0.99
10-stage Inverter	10	104.0	68.67	1.51	0.75

*: CPU is Intel Core 2 Duo (2.53 GHZ) that has two cores

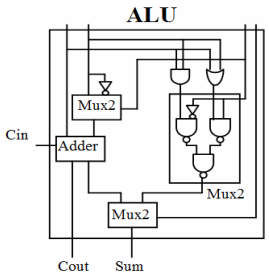

Figure 4. The schematic of ALU.

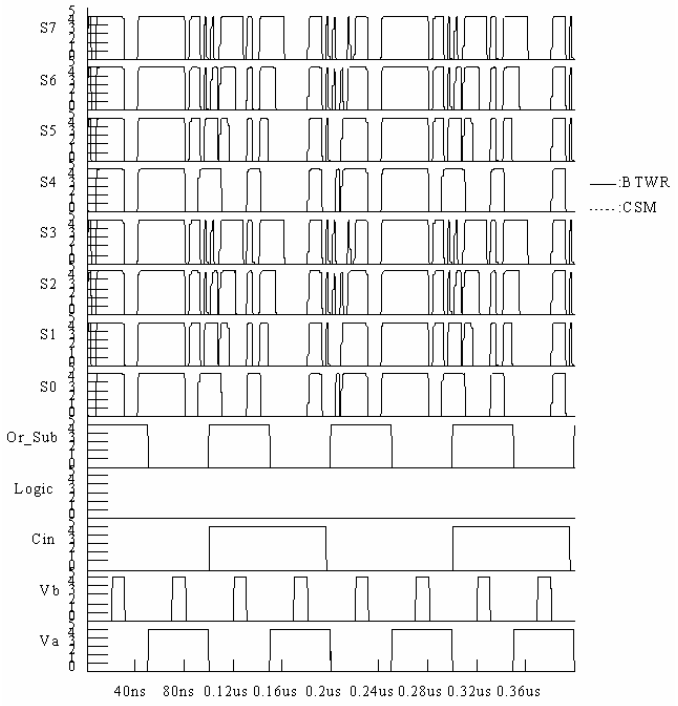

Figure 5. Waveform comparison for circuit alu4-2, which has two 4-bit ALU.

REFERENCES

[1] A. R. Newton and A. L. Sangiovanni-Vincentelli, "Relaxation-based electrical simulation," IEEE Trans on CAD, Vol. CAD-3, pp. 308-311, Oct. 1984.

[2] R. A. Saleh and A. R. Newton, "The exploitation of latency and multirate behavior using nonlinear relaxation for circuit simulation," IEEE Trans., Computer-aided Design, vol. 8, pp. 1286-1298, December 1989.

[3] J. F. Mao, E. S. Kuh, "Fast simulation and sensitivity analysis of lossy transmission lines by the method of characteristics," IEEE Tran. on Computer-aided Design, Vol. 44, No. 5, pp. 391-401, May 1997.

[4] Martin J. Gander, Mohammad Al-Khaleel and Albert E. Ruehli, "Optimized Waveform Relaxation Methods for the Longitudinal Partitioning of Transmission Lines," IEEE Trans. Circuits and System, pp. 1732~1743, 2009.

[5] C. J. Chen, T. N. Yang, and J. D. Sun, "The Backward-traversing Relaxation Algorithm for Circuit Simulation," IEEE Custom Integrated Circuit Conference, San Jose, California, pp. 353-356, September 10-13, 2006.

[6] Carlos Pon Soto, Resve Saleh, and Tad Kwasniewski, "Time warping-waveform relaxation in a distributed circuit simulation environment," pp. 338-341, Proceedings of the 38th Midwest Symposium on Circuit and System, 1995.