Event-Driven Modeling and Simulation of an Digital PLL

Jingcheng Zhuang
Dept. of Electronics
Carleton University
jzhuang@doe.carleton.ca

Qingjin Du
Dept. of Electronics
Carleton University
qidu@doe.carleton.ca

Tad Kwasniewski
Dept. of Electronics
Carleton University
tak@doe.carleton.ca

ABSTRACT

An event-driven modeling and simulation technique, implemented in Matlab is presented in this paper. It enables rapid and accurate simulation as it only calculates the time instants of interest. This technique is successfully applied to behavioral modeling and simulation of an digital phase-locked loop. The simulation environment retains the flexibility of modeling and mathematical manipulation that characterizes Matlab. For example, it allows time-domain modeling phase noise of each component of a digital PLL.

1. INTRODUCTION

Phase-locked loops (PLLs) have been widely used in frequency synthesis and timing recovery for many years. To date, most PLLs in the literature are in analog form, where analog loop filters (ALFs) and voltage controlled oscillators (VCOs) are employed, and they are difficult to be integrated in a noisy digital environment and are not portable for new processes. Now, deep-submicron CMOS processes allows digitizing many conventional analog circuits, including the analog PLLs, to overcome above problems and enable more desired features that are not achievable in an analog implementation. In fact, digital PLLs (DPLLs) gain increasing interests in both frequency synthesis and the data recovery fields in recent years [1][2][3][4][5].

The block diagram of a DPLL is shown in Figure 1. It consists of a digital phase/frequency detector (DPFD), a digital loop filter (DLF), a digitally controlled oscillator (DCO) and a frequency divider (FDIV). The DPFD compares the clock

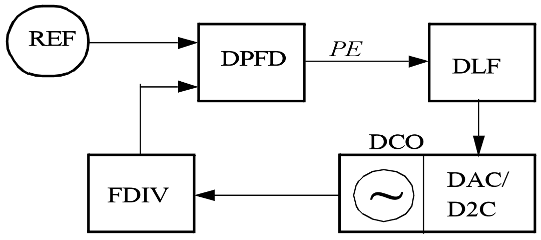

Figure 1. The block diagram of a digital PLL

from the reference source (REF) and the clock from FDIV, and provides digitalized phase error (*PE*) for the DLF. DLF tunes the digital frequency control word of the DCO according to the phase error information. In such a system, the phase error digitizing, finite word length of the DLF, the finite frequency resolution of the DCO and so on, result in truncation error and phase/frequency quantization error. In addition, the DLF might include other features, such as automatic loop bandwidth adjustment. Consequently, the modeling and simulation of those systems at the system level is essential to confirm the feasibility and optimize the system architecture and parameters. To simulate the noise performance of the entire DPLL in time domain, the simulation has to be able to provide precise positions of transition edges. In this case, time-driven type simulators, such as Simulink, Hspice, and so on, are not efficient because the accuracy is limited by the step size, and using a small step size, corresponding to the required accuracy, normally results in excessively long simulation times. For systems such as clock multipliers, clock and data recovery circuits, many time steps calculated by a time-driven simulator are not useful for the end user. Only some special time points of interest, such as, the transition time points of a clock signal, the sampling outputs right after each sampling operation, and so on are needed for loop analysis. In [6], the simulation and modeling of phase noise in an RF oscillators in time-domain, using an event driven VHDL simulator, is studied. However, simulating a DPLL using VHDL simulator requires creating VHDL models for all blocks, and it is not feasible in the system-level design stage, where a simple high-level modeling approach is desired.

This paper presents an event-driven modeling and simulation technique to simulate and analyze the behavior of the whole digital PLL shown in Figure 1. It calculates only the time points of interest, and provides precise edge position information. This modeling and simulation technique can be implemented in various computer languages. In this work, Matlab script is selected to implement this technique because it provides various data processing functions and simplifies modeling of each function block and post-processing of the simulation output data. For a 2 GHz digital PLL as given in the following sections, the simulation speed is approximately 2 μs per second of CPU time on a P4-2G PC. The simulation speed can be further significantly increased by using precompiled computer languages, such as C language. Because this technique allows high-level modeling of function blocks, it is extremly helpful for the system level design and optimization of system parameters.

This paper is organized as follows, the basic concept of event-driven method is given in the section 2, and the implementation of the event-driven engine using Matlab is given in Section 3. In Section 4, the modeling of a DPLL is given together with the simulation results. Finally, some conclusions are drawn in Section 5.

2. THE CONCEPT OF EVENT-DRIVEN

Conventional oversampling simulation tools do a time-sweep with a certain time step, and calculate signals at each time point. To achieve a good time resolution, the time step has to be sufficiently small, which means the simulation needs to calculate a large a number of time points. In event-driven simulation, however, only time points of interest are calculated. The concept of event driven has been widely used for many years. An event-driven program basically consists of three parts, the event generators, the event dispatcher and the event handlers as shown in Figure 2. Event generators generates the events. For example, in a computer system, an event can be generated by a mouse click, a key press, a timer and so on. The event dispatcher receives and stores all events and calls corresponding event handlers based on a certain order. This is normally achieved by maintaining an event queue inside the event dispatcher. The program always returns to the event dispatcher after a call to an event handler is completed. In some cases, the event handlers also generate some events, like an event generator, when they are executed and those events will be processed in the same manner by the event dispatcher.

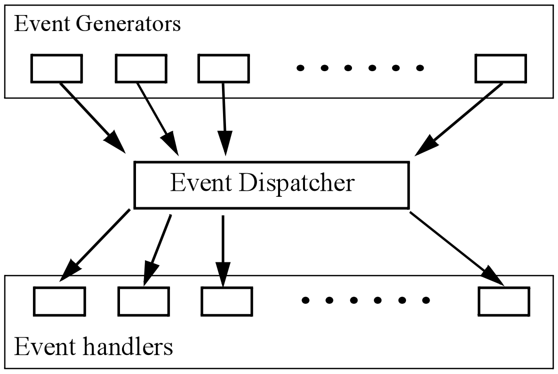

Figure 2. Concept of the event driven program

3. EVENT-DRIVEN ENGINE

The principle of the event-driven modeling is shown in Figure 3, in which each building block (Block A, B ...) of the system to be simulated is represented as an event handler. The event starter generates some initialization events for each building block then the event dispatcher calls each function block according to the event queue, maintained inside the event dispatcher. Those event handlers are responsible for generating all subsequent events to be added to the event queue. Each event includes four fields, event ID, event handler, event time and event parameter. The later two are to be passed to the event handler so that the event handler can process the event correctly. In addition, some top-level signals or variables are shared among building blocks as shown in Figure 3.

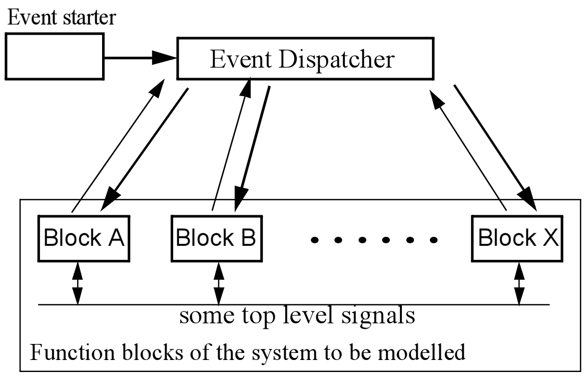

Figure 3. Principle of the event driven simulation

3.1. Event Dispatcher

As shown in Listing 1, the event dispatcher is basically a

Listing 1. Matlab code for the event dispatcher.

```
%start the event-driven engine
h=waitbar(0,'Simulation is running, please wait...');
Proc=0; EventCount=0; t=cputime;
Result=event('RUN');
while (Result<=EndTime)
   EventCount=EventCount+1;
   if floor(Result/EndTime*50)>Proc
        Proc=Proc+1;
        waitbar(Result/EndTime);
   end
   Result=event('RUN');
end
close(h);
disp(sprintf('Simulation is done!\n  Total number of events:
%i',EventCount));
disp(sprintf('  Total CPU time used: %f (s)',cputime-t));
```

loop, which always picks up the earliest event in the event queue and calls the corresponding event handler. The sub-function “event” defined as “Result=event(Task, Func, Time, Para)” is responsible for managing the event queue and, depending on the parameter of *Task*, it performs one of the following tasks:

1. Initialize (INIT) the event queue.
2. Insert (INS) an event to the event queue and return an event ID. This task is called whenever an event is generated by the event starter or an event handler. The event is inserted to the event queue and sorted by event time.
3. Remove (DEL) an event specified by the event ID from the event queue.
4. Run (RUN) the earliest event in the event queue and remove it from the event queue. It returns the event time so that the main program can determine the progress of the simulation. To avoid possible errors, the earliest event has to be temporaily stored and removed from the event queue before the corresponding event handler is called because the event handler may call this event manangement function recursively to add or remove events.

3.2. Event Starter

In event-driven simulation, almost all events are generated by building blocks of a system to be modelled, except for some events at the beginning of the simulation. Those events, generated by the event starter, are to initialize all building blocks so that more subsequent events can be generated by those blocks. The event starter first requests the initialization of the event queue, and then generate initialization events for each building block. The Matlab code given in Listing 2 is the event starter for the model of the digital phase locked loop as shown in Figure 1. It generates the initialization events for REF, DCO, FDIV, DPFD and DLF.

Listing 2. Matlab code of the event starter.

```
event('INIT');
%Reset all blocks by passing a parameter of '0' at time of 0 to
those blocks
event('INS', @REF, 0, 0);
event('INS', @DCO, 0, 0);
event('INS', @FDIV, 0, 0);
event('INS', @DPFD, 0, 0);
event('INS', @DLF, 0, 0);
```

3.3. Modeling of Building Blocks

The modeling of each building block is actually done by writing a Matlab function for each block. As given in Listing 3, this function processes different events specified by *para*. In the process of an event, some new events might be generated if necessary. The event '0' is assumed to be always the initialization event. Other events are pre-defined for each building block. The models of each building block of a DPLL are presented in the following section.

Listing 3. Matlab code of a build block

```
function Block_Name(time,para)
persistent Var1 Var2 ...;
switch para
   case 0 %Initialize the block
      <Initialization>;
   case 1
      <Process of event type 1>;
   case 2
      <Process of event type 2>;
      <...>;
   case N
      <Process of event type N>;
end
```

4. MODELING OF DPLL

4.1. Reference Clock Source

The PFD compares the edge (rising edges are considered in the paper) of the reference clock and the edges of the divided clock, so only the time instant of each rising edge of the reference clock is of interest. The reference clock source is modeled as follows: In the initialization part of the model, the first rising edge event is generated, and during the process of each rising edge, the event of next rising edge is generated so that the clock signal can be continuously provided to the loop. Obviously, all rising edge events are sent to the PFD so that the PFD can compare them with the divided edges. Consequently, the model of the reference source is written as given in Listing 4, where the reference period is defined as *Tref*, the time of the next rising edge is always calculated as the "*time+Tref*". Here, it is easy to add some additional timing jitter/noise to the reference clock source by replacing "*time+Tref*" with "*time+Tref+Tj*", where *Tj* is the timing jitter to be added.

Listing 4. Matlab code of the model of the reference clock.

```
function REF(time,para)
persistent Tref;
switch para
   case 0 %initialization
      event('INS', @REF, 0 ,1);
      Tref=5e-9; %reference period
   case 1 %process of edges
     %Add next reference edge
     event('INS', @REF, time+Tref,1);
     %Send edges to DPFD
      event('INS', @DPFD, time, 1);
end
```

4.2. Phase and Frequency Detector

In the DPLL, the phase and frequency detector compares the reference clock edges and the divided clock edges to get the phase error, which is quantized to form the output of the DPFD. The structure of the DPFD [3] modelled in this paper, is shown in Figure 4. A conventional tri-state PFD is used to

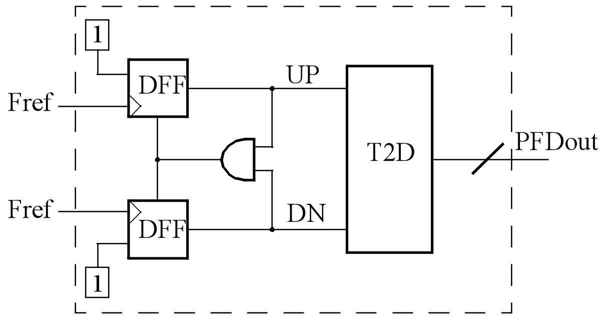

Figure 4. The digital phase and frequency detector

obtain the phase error (the difference in duration of UP/DN pulses). This error is converted to digital domain by time to digital converter (T2D). In this example, a T2D converter with a time resolution of 100ps is modelled. The output of the T2D is carried by a global variable *PFDout*, which is accessible to the digital loop filter.

In addition to initialization, the DPFD processes three events: the rising edge of the reference clock, the rising edge of the divided clock from the frequency divider, and the internal reset event. Listing 5 shows the Matlab code for the PFD model. At the reference signal rising edge event, if *UP* is zero, it changes *UP* to one and stores the current time to *RefTime*. It also checks if *DN* is already one to determine whether it is necessary to generate a reset event. If *DN* is one, a reset event is generated with a delay of 50ps in the reset path. Similar process is done at the event of the divider edge. In the reset event, the time error is calculated and the output of T2D, with a time resolution of 100ps, is obtained and stored in the variable *PFDout*. A new event, with another 20ps delay, is inserted to the event queue so that the DLF can process this phase error.

4.3. Digital Loop Filter

The digital loop filter tunes the control word of the DCO based on the phase error information (*PFDout*) provided by the DPFD. A conventional Type II loop filter, as shown in Figure 5, is modelled in this work. After initialization, DLF processes the events generated by DPFD whenever a time error is obtained. During the process, the DLF adjusts the DCO control word according to the quantized phase error (*PFDout*). Once the new control word is obtained, the DLF generates an event for the DCO to update the DCO frequency. The control word is transferred to DCO using global variable *Dcontrol*. Listing 6 shows the model of the digital loop filter,

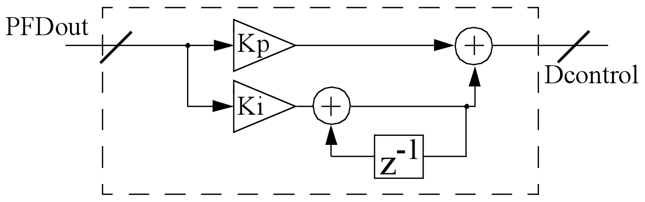

Figure 5. A type II digital loop filter

Listing 5. Matlab code of the model of the PFD

```
function DPFD(time,para)
global PFDout;
persistent UP DN RefTime DivTime;
switch para
  case 0  %Initialization
    UP=0; DN=0; P1=0;
  case 1  %Referece edge
    if UP==0
      RefTime=time;
      UP=1;
      if DN==1
        event('INS', @DPFD, time+50e-12,3);
      end
    end
  case 2  %Divider edge
    if DN==0
      DivTime=time;
      DN=1;
      if UP==1
        event('INS', @DPFD, time+50e-12,3);
      end
    end
  case 3  %PFD reset
    UP=0; DN=0;
    PFDout=2*round((DivTime-RefTime)/100e-12)-1;
    event('INS', @DLF, time+20e-12,1);
end
```

Listing 6. Model of the digital loop filter

```
function DLF(time,para)
global PFDout Dcontrol;
persistent Int;
switch para
    case 0 %Initialization
        Int=400; Dcontrol=400; %Initial value of the DLF output.
        event('INS', @DCO, time, 2); %Update the DCO period
    case 1  %When the PFDout is ready, Calculate the output
        Int=Int+PFDout*0.1;
        Int=min(max(Int,5),2043);
        Dcontrol=round(Int+PFDout*5);
        event('INS', @DCO, time+50e-12,2);
 end
```

in which the gain of integration path is 0.1 and the gain of proportional path is 5. The propagation delay of the DLF is assume to be 50ps.

4.4. Digitally-Controlled Oscillator

The digitally-controlled oscillator can be modelled in the similar way as the reference source, but now the frequency or the period is controlled by its digital control words. Consequently, one more event type DCO should respond to is the event of frequency updating. In the process of frequency updating, the new frequency is calculated based on the control signal and then compared with the previous frequency. If two frequencies are the same, nothing needs to be done. Otherwise, the event of the next VCO edge, previously inserted into the event queue, should be replaced with a new edge event based on the new frequency. The Matlab code for the frequency update is given in Listing 7. Here, an ideal DCO

Listing 7. Frequency update in the DCO model.

```
NewFreq=(1024-(Dcontrol))*50e3+2e9;
NewPeriod=25/NewFreq;
if(NewPeriod~=Tvco)
  event('DEL', @DCO, 0, EventID); %Remove the old next
edge
  NextEdge=(NextEdge-time)/Tvco*NewPeriod+time;
  event('INS', @DCO, NextEdge ,1);%Add the new edge
  Tvco=NewPeriod;
end
```

with 50kHz frequency resolution, a frequency tuning range of 102.4MHz (50kHz*2048) and a center frequency of 2GHz is modelled. Similarly with the model of the reference source, the jitter or phase noise can be easily included in the model to simulate the noise performance of the loop.

4.5. Frequency Divider

The frequency divider accepts the rising edge events generated by the DCO and generates a divider edge event for the DPFD once every *N* DCO periods. The Matlab code for the frequency divider model is given in Listing 8. The time of the DPFD event is calculated as “time+100e-12” so that a 100ps time delay of the frequency divider is modelled. Again, some additional noise can still be added here if necessary.

Listing 8. Model of the frequency divider.

```
function FDIV(time,para)
global Div_Edges
persistent COUNT N;
switch para
  case 0
    COUNT=0; N=25; %Initialize the counter
  case 1  %DCO edge event
    COUNT=COUNT+1;
    if (COUNT==N)
      COUNT=0;
      event('INS', @DPFD, time+100e-12, 2);
    end
end
```

4.6. Simulation Results

With the models and the event-driven engine created above, various types of analysis can be performed to examine the performance of a digital PLL. In this section, the locking behavior and the jitter transfer characteristic (from the reference source to the DCO output) of the loop are analyzed.

Locking behavior can be observed by plotting the DCO’s control word magnitude and the magnitude of the PFD output. The DPLL modelled above is simulated for 10μs within 5 seconds CPU time, and the results are shown in Figure 6.

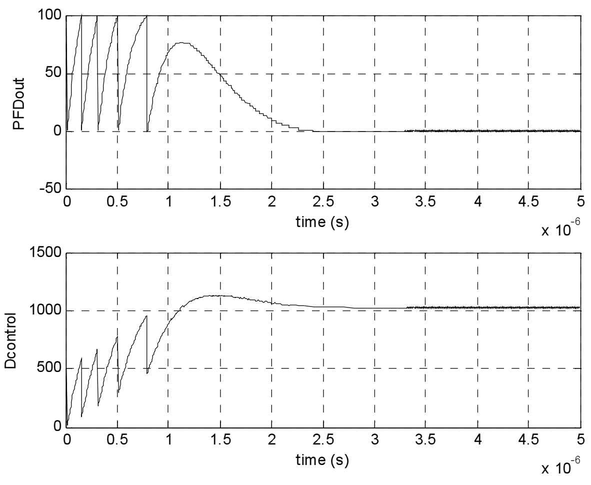

Figure 6. Locking behavior simulation result

The Figure shows that the loop acquires locking within 3.5μs, and 4 cycle slips occur during the locking process. Due to the finite frequency resolution of the DPFD/DLF/DCO, after the loop is locked. the control word typically is not constant, but jumps among several discrete frequency levels. The average frequency equals to the desired frequency.

Digital PLL shows significant non-linear characteristics for small phase noise amplitude originating in the PFD/DLF/DCO. The noise transfer characteristics, for different noise amplitude levels, are examined using the event-driven environment. As an example, a given amplitude/frequency single-tone phase variation is added in the reference source and the edge timing jitter of the DCO output is examined. To add some timing jitter, the model of the REF is modified as given in Listing 9. The reference edge locations stored in the array *REF_Edges* include the additional phase noise or edge jitter of a given noise frequency and noise amplitude.

Listing 9. Model of the reference with edge timing jitter

```
function REF(time,para)
global REF_Edges;
persistent Tref n;
switch para
  case 0 %initialization
    event('INS', @REF, 0 ,1);
    Tref=5e-9; %reference period
    n=0;
  case 1 %process of edges
    %Add next reference edge
    n=n+1;
    event('INS', @REF, REF_Edges(n),1);
    %Send edges to DPFD
    event('INS', @DPFD, time, 1);
end
```

A simulation with two-dimensional parameter sweep for the noise amplitude and noise frequency, is done and the RMS jitter at the output of the DCO is calculated. The ratios between the output noise power and input noise power for different input noise amplitudes (An) are plotted as shown in Figure 7.

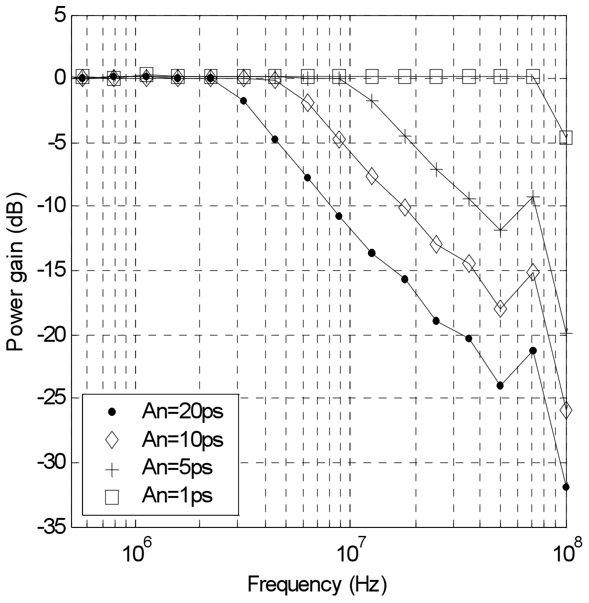

Figure 7. Reference noise transfer characteristic

It shows that the DPLL exhibits different loop bandwidth for different input noise amplitudes, which is due to the non-linear/quantization operation of the loop. In this figure, the output noise includes not only the noise caused by the reference noise but the quantization noise of the DPLL. At higher frequencies, the noise caused by the reference clock decreases and finally the output noise is dominated by the quantization noise. For reference jitter with low amplitudes, 1ps or less, the output noise of the DCO is almost dominated by the quantization error for the entire frequency band. The simulation yielding results of Figure 7, required 64 sub-simulations and some data processing. It took less than 10 minutes of CPU time. Similarly, the capability of the DPLL to suppress DCO noise can be obtained by adding extra noise component in the model of the DCO and examining the output jitter.

5. CONCLUSIONS

An event-driven modelling and simulation technique using Matlab, is applied to a digital PLL. The proposed technique achieves a rapid and accurate time-domain simulation, and allows various noise sources to be easily included and analyzed. With this technique, a simulation of a system is invoked by a function call in Matlab script, so mulitple variable sweeps, parameter optimization and post-processing of the output data can be easily achieved. An an example, the locking behavior and the noise transfer characteristic of the loop are obtained. The modeling of each block is essentially writing the functional description of that block so that one can focus on the functionality design of each component.

REFERENCES

[1] T. Olsson, P. Nilsson, "A digitally controlled PLL for digital SOCs", Proceedings of the International Symposium on Circuits and Systems 2003, Pages: 437 - 440 vol.5.

[2] Liming Xiu, Wen Li, J. Meiners, R. Padakanti, "A novel all-digital PLL with software adaptive filter", IEEE Journal of Solid-State Circuits, Vol. 39, Issue 3, March 2004, Pages: 476 - 483.

[3] J. Lin, B. Haroun, T. Foo, Jin-Sheng Wang, B. Helmick, S. Randall, T. Mayhugh, C. Barr, J. Kirkpatric, "A PVT tolerant 0.18MHz to 600MHz self-calibrated digital PLL in 90nm CMOS process", Digest of Technical Papers. IEEE International Solid-State Circuits Conference 2004, Pages: 488 - 541 Vol.1.

[4] R. B. Staszewski, J. Wallberg, S. Rezeq, Chih-Ming Hung, O. Eliezer, S. Vemulapalli, C. Fernando, K. Maggio, R. Staszewski, N. Barton, Meng-Chang Lee, P. Cruise, M. Entezari, K. Muhammad, D. Leipold, "All-digital PLL and GSM/EDGE transmitter in 90nm CMOS", Digest of Technical Papers. IEEE International Solid-State Circuits Conference 2005, Pages: 316 - 600 Vol. 1.

[5] A. M. Fahim, "A compact, low-power low-jitter digital PLL", Proceedings of the 29th European Solid-State Circuits Conference, Pages: 101 - 104.

[6] R. B. Staszewski, C. Fernando, P. T. Balsara, "Event-driven Simulation and modeling of phase noise of an RF oscillator", IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, Vol. 52, Issue 4, April 2005, Pages: 723 - 733.