Energy deposit

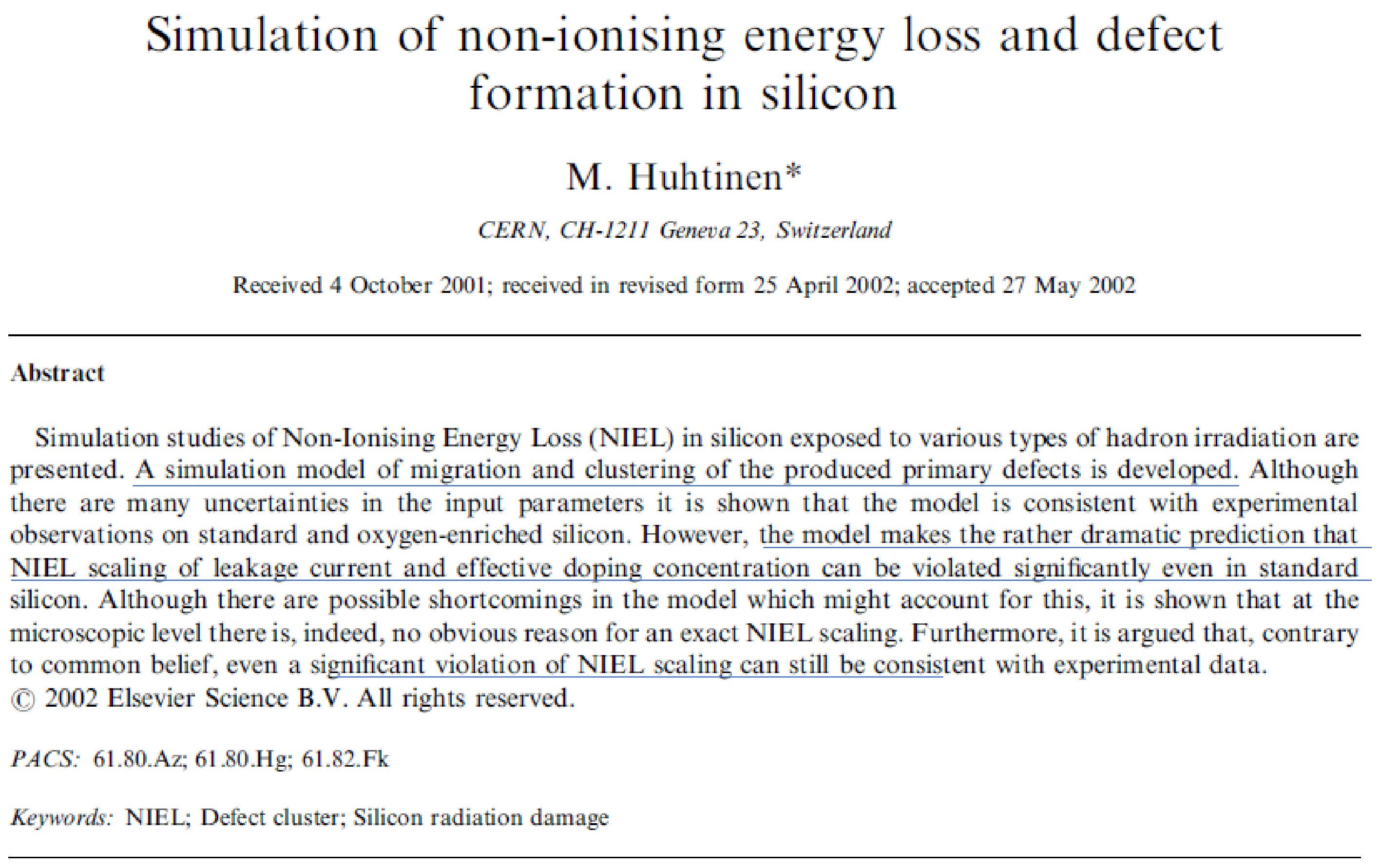
Linear Energy Transfer

- Linear Energy Transfer이란, 하전입자가 물질을 통과할 때 전자와 상호작용하여 남기는 에너지를 의미한다.
- LET는 secondary electron(delta ray)가 만들어지느냐에 따라 restricted LET, unrestricted LET로 나누어 설명한다.
- Restricted LET는 delta ray를 고려하지 않은 LET로 Linear Energy Transfer = $\frac{dE}{dx}$ 로 나타난다. 한편 Unrestricted LET는 delta ray를 고려한 LET로 delta ray에 의해 넓은 범위로의 energy transfer 를 의미한다.
- Unrestricted LET는 electronic stopping power와 동일한 물리량이다.

Non-ionizing energy loss

- Non-ionizing energy loss는 electronic excitation을 포함하지 않는 개념으로, nuclear reaction이 없을 때 nuclear stopping power와 같은 물리량이다.
- Charged particle이 물질을 통과할 때, 원자 핵과 반응하여 vaccancy와 interstitial defect를 만들면서(displacement damage) 에너지를 잃게 되는데 이 때의 energy loss를 non-ionizing energy loss 라고 한다.

관련 논문



Simulation of non-ionising energy loss and defect formation in silicon

M. Huhtinen*

CERN, CH-1211 Geneva 23, Switzerland

Received 4 October 2001; received in revised form 25 April 2002; accepted 27 May 2002

Abstract

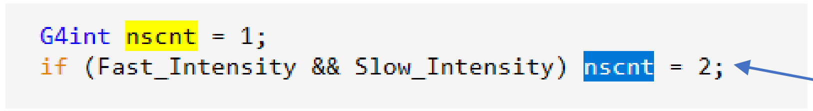
Simulation studies of Non-Ionising Energy Loss (NIEL) in silicon exposed to various types of hadron irradiation are presented. A simulation model of migration and clustering of the produced primary defects is developed. Although there are many uncertainties in the input parameters it is shown that the model is consistent with experimental observations on standard and oxygen-enriched silicon. However, the model makes the rather dramatic prediction that NIEL scaling of leakage current and effective doping concentration can be violated significantly even in standard silicon. Although there are possible shortcomings in the model which might account for this, it is shown that at the microscopic level there is, indeed, no obvious reason for an exact NIEL scaling. Furthermore, it is argued that, contrary to common belief, even a significant violation of NIEL scaling can still be consistent with experimental data.

PACS: 61.80.Az; 61.80.Hg; 61.82.Fk

Keywords: NIEL; Defect cluster; Silicon radiation damage

¼ DCV timing resolution and attenuation length

G4Scintillation



```
G4int nscnt = 1;
if (Fast_Intensity && Slow_Intensity) nscnt = 2;
```

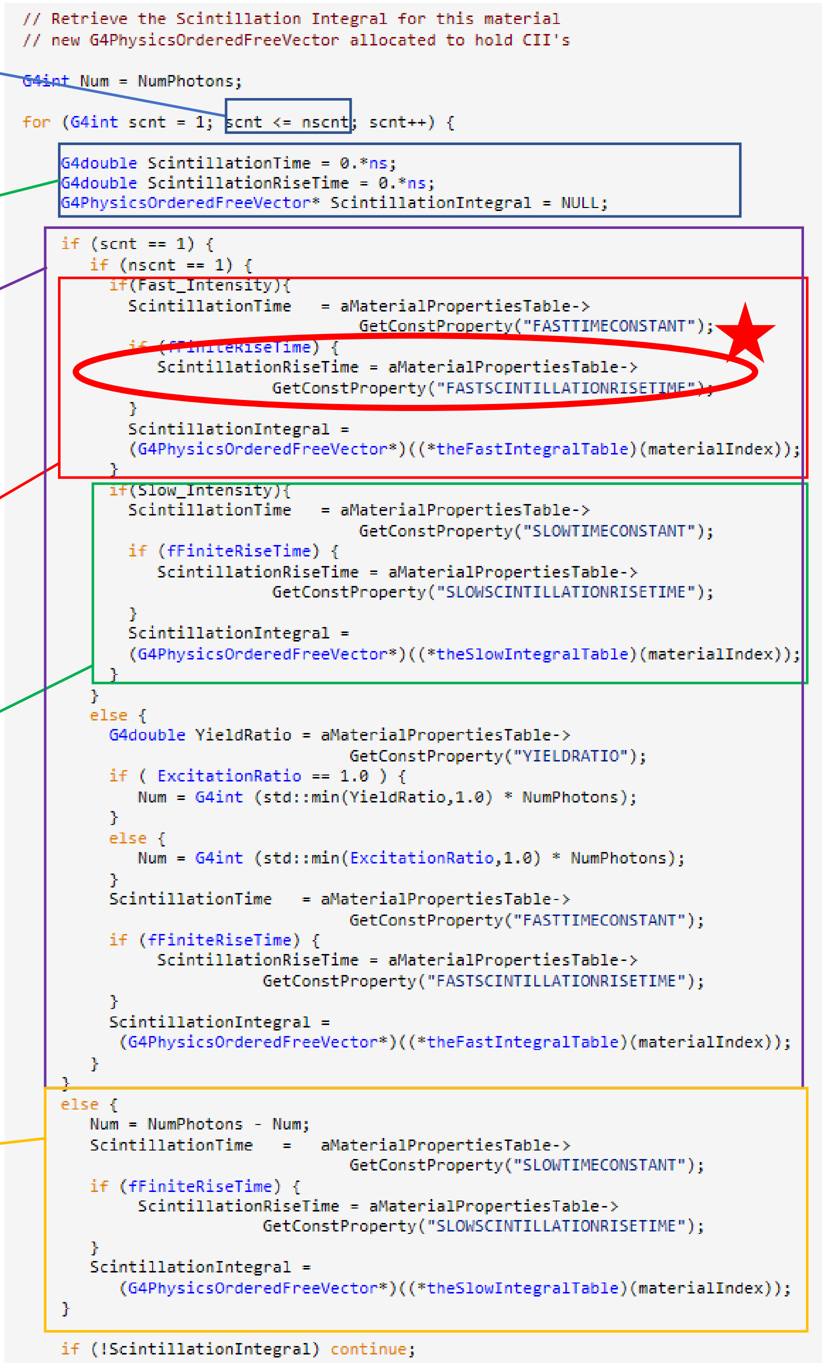
Default value of scintillation time and rise time

The number of defined component is one

For fast decay component

For slow component

Two components are well-defined



```
// Retrieve the Scintillation Integral for this material
// new G4PhysicsOrderedFreeVector allocated to hold CII's

G4int Num = NumPhotons;

for (G4int scnt = 1; scnt <= nscnt; scnt++) {

    G4double ScintillationTime = 0.*ns;
    G4double ScintillationRiseTime = 0.*ns;
    G4PhysicsOrderedFreeVector* ScintillationIntegral = NULL;

    if (scnt == 1) {
       if (nscnt == 1) {
         if(Fast_Intensity){
           ScintillationTime   = aMaterialPropertiesTable->
                                   GetConstProperty("FASTTIMECONSTANT");
           if (fFiniteRiseTime) {
              ScintillationRiseTime = aMaterialPropertiesTable->
                                   GetConstProperty("FASTSCINTILLATIONRISETIME");
           }
           ScintillationIntegral =
           (G4PhysicsOrderedFreeVector*)((*theFastIntegralTable)(materialIndex));
         }
         if(Slow_Intensity){
           ScintillationTime   = aMaterialPropertiesTable->
                                   GetConstProperty("SLOWTIMECONSTANT");
           if (fFiniteRiseTime) {
              ScintillationRiseTime = aMaterialPropertiesTable->
                                   GetConstProperty("SLOWSCINTILLATIONRISETIME");
           }
           ScintillationIntegral =
           (G4PhysicsOrderedFreeVector*)((*theSlowIntegralTable)(materialIndex));
         }
       }
       else {
         G4double YieldRatio = aMaterialPropertiesTable->
                                   GetConstProperty("YIELDRATIO");
         if ( ExcitationRatio == 1.0 ) {
            Num = G4int (std::min(YieldRatio,1.0) * NumPhotons);
         }
         else {
            Num = G4int (std::min(ExcitationRatio,1.0) * NumPhotons);
         }
         ScintillationTime   = aMaterialPropertiesTable->
                                   GetConstProperty("FASTTIMECONSTANT");
         if (fFiniteRiseTime) {
              ScintillationRiseTime = aMaterialPropertiesTable->
                                   GetConstProperty("FASTSCINTILLATIONRISETIME");
         }
         ScintillationIntegral =
          (G4PhysicsOrderedFreeVector*)((*theFastIntegralTable)(materialIndex));
       }
    }
    else {
       Num = NumPhotons - Num;
       ScintillationTime   =   aMaterialPropertiesTable->
                                   GetConstProperty("SLOWTIMECONSTANT");
       if (fFiniteRiseTime) {
            ScintillationRiseTime = aMaterialPropertiesTable->
                                   GetConstProperty("SLOWSCINTILLATIONRISETIME");
       }
       ScintillationIntegral =
          (G4PhysicsOrderedFreeVector*)((*theSlowIntegralTable)(materialIndex));
    }

    if (!ScintillationIntegral) continue;
```

Things to do

- Geant4의 G4Scintillation에서 rise time 관련된 package 사용법 확인 후 넣기
- G4OpWLS package에서 timing을 줄 수 있는 방법 생각해보기.
-