Universitatea Politehnica din Bucuresti

Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Proiect la disciplina

Inginerie Software

UML Profile for Systems on a Chip (SoC)

Studenti:

Codreanu Radu Adrian

Gheorghe Alin Valentin

Grupa

442A

2015-2016

Cuprins:

1. Introducere(Gheorghe Alin Valentin)

2. Design hardware bazat pe activitati UML(Codreanu Radu)

2.1Un profil UML pentru SystemC

3. Modelarea structurala(Gheorghe Alin Valentin)

4. Diagrame de activitate(Codreanu Radu)

5. Generare cod (Exemplu)(Codreanu Radu)

6. Bibliografie

1.Introducere

Distanta in cadrul productivitatii dintre tehnologia semiconductoarelor, metodologie si suportul tehnic a devenit una dintre cele mai mare provocari in cadrul sistemelor embeded si a design-ului SOC. Pentru a rezolva aceasta problema, specialistii din domeniu au facut cercetari in domeniul ingineriei software, de unde au imprumutat multe idei pentru a micsora aceasta distanta. Majoritatea autorilor sunt de acord cu cel putin cinci principii care ridica nivelul abstractizarii, ierarhizarii, separarii grijilor, refolosire si integrare. Din momentul in care sistemele embedded si dezvolatrea SOC necesita o colaborare intre clienti, atat pentru echipele de software cat si pentru cele de hardware, se prefera un limbaj comun pentru a elimina neintelegerile care pot surveni pe parcursul procesului. Acest limbaj trebuie sa fie capabil sa adune cerintele clientilor si apoi sa dezvolte implementari eficiente atat hardware cat si software, intr-o realizare bine definita, bazata pe cele cinci principii mentionate anterior. Se presupune ca daca este realizat corespunzator, UML-ul, poate fi acest limbaj.

UML-ul este un limbaj standardizat, cu scop general, in cadrul ingineriei software. UML include un set de notatii grafice pentru a creea modele pentru sisteme specifice. Un model UML este format dintr-un numar de elemente de model, asezate in diagrame diferite utilizate pentru cu scop specific. Aceste diagrame include diagrama de clasa, diagrama de componente, diagrama de structura utilizata pentru aspectele structurale, diagrama de activitate, diagrama de stare si diagrama de secventa pentru aspecte comportamentale. Un model UML trebuie sa respecte metamodelul UML care se regaseste intr-un document numit “ UML Stuperstructure”.

In prezent, dezovoltarea software-ului condus de model este in mare parte bazata pe UML. Versiunea sa upgradata la 2.0 a prelungit semnificativ expresivitatea unora dintre notatiile de baza, astfel se deschid noi zone de utilizare. Semantica noii activiatati bazata pe jeton, se potriveste foarte bine cu comportamentul dominant al fluxului de date pentru sistemele hardware si poate fi folosit pentru a descrie un asemenea comportament la un nivel crescut de abstractizare in timp ce aduce imbunatatiri comparabile cu traditionalul HDLs.

In aceasta lucrare, consideram ca activitatile UML sunt potrivite pentru modelarea datelor si a controlului in design-ul hardware si totodata poate fi utilizata ca punct de plecare pentru o incercare de design hardware complet.

2. Design Hardware baza pe activitati UML

Un design bazat pe model pentru SoC trebuie sa cuprinda comportamentul sistemului cat si compozitia sa din blocurile functionale si cateva aspecte ne-functionale, cum ar fi domenii de ceas (clock domains) sau alocarea de resurse fizice. Pentru aceasta , am identificat un subset UML 2.0 pentru specificarea completa SoC. Acest subset este prezentat ca profilul UML.

Un astfel de profil este un subset sintactic al UML cu semantici extinse specifice domeniului.

Baza acestui subset este formata din elemente pentru modelarea comportamentala prin diagrame de activitate. Aceste elemente sunt complementate de elemente de model specializate pentru Clasa, Structura compozita si Diagrame de secventa pentru modelarea structurii si aspectele fizice ale sistemului. Din acest model de sistem, generarea de cod pentru sinteza hardware automata poate fi efectuata.

UML este un limbaj de modelare grafic orientat pe obiecte, la inceput a fost folosit pentru sisteme software. Folosirea unor astfel de notatii grafice ajuta designer-ul sa inteleaga si sa analizeze cererile clientului in stagiile de inceput ale dezvoltarii. In forma sa de baza, este aplicabil pentru o gama foarte larga de sisteme (open language). Totusi, cateva atribute cheie a UML sunt importante pentru sistemele embedded.

1. UML este abstract, iar designer-ii se pot concentra pe caracteristicile de nivel ridicat ale sistemului, in loc de factorii specifici implementarii.

2. Designer-ii Hardware si Software vor folosi acelasi limbaj.

3. Un set bogat de notatii, modele executabile si semantici potrivite pentru modelarea diferitelor puncte de vedere, simulare si verificare formala.

4. Suport pentru decompozitie structurala bazata pe obiecte.

5. Suport pentru semantici legate de starile masina care pot fi folosite pentru modelare si sinteza.

Dupa diagrama de implementare UML 1.x, StateChart si diagrama de secventa care au fost folosite pentru modelare hardware, pentru sintetizarea controller-elor FSM, si pentru modelarea protocoalelor hardware de comunicare (Handshake) , UML 2.0 include doua noi diagrame mult mai potrivite pentru reprezentarea conceptelor hardware, acestea sunt diagrama de structura (Structure Diagram- SD) si diagrama de timp (Timing Diagram -TD). Diagrama de structura descrie structura sistemului ca o retea de componente (obiecte, obiecte compuse si blocuri) conectate prin canale (link-uri). Este similar cu bine cunoscuta diagrama a blocurilor functionale folosita in domeniul hardware. Putem considera diagrama de timp TD un cronogram care este folosit pentru a reprezenta progresia semnalelor in timp.

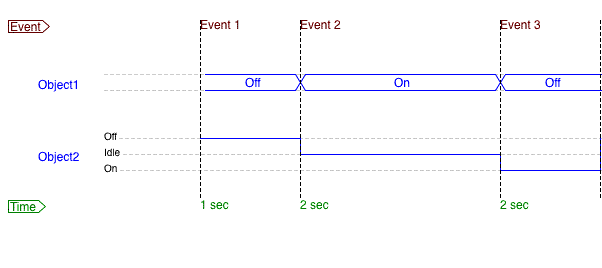


Figura1. Diagrama de timp.

Noile semantici atasate diagramelor de activitate (data flow) sunt de asemenea importante pentru a modela Datapath-ul unitatilor de procesare.

Noi caracteristici incluse in diagrama de secventa UML 2.0 (Control flow, ierarhie, constrangeri de timp) sunt de asemenea foarte importante pentru analiza de performanta a sistemelor on chip (SoC).

Lucrari recente tintesc catre generarea de limbaje de descriere hardware (cum ar fi VHDL- Verilog Hardware Description Language) si SystemC din diagrame UML. Codul generat este folosit fie pentru simulari sau sinteze.

In ciuda efortului in directia design-ului de sisteme bazat pe UML, nu exista nici un design flow consistent pentru sisteme SoC embedded si metodologiile impreuna cu tool-urile asociata inca nu sunt complete si interoperabile. Din acest motiv, multe profile UML 2.0 au fost propuse atat de invatamant cat si de industrie. Potrivit autorilor, UML 2.0 poate fi folosit pentru mai multe domenii de aplicatii prin definirea de profile.

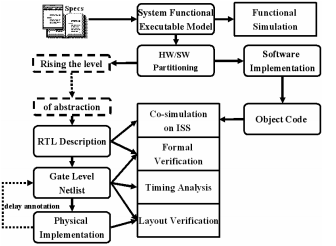


Figura2. SoC Design Flow

Procesul de design incepe prin specificarea cerintelor sistemului- de obicei scrise in limbaj natural-. Un model functional executabil este modelat din cerinte pentru a captura comportamentul sistemului. Acest lucru este de obicei realizat intr-un limbaj de programare software ca C/C++ sau folosind tool-uri de modelare cum ar fi Simulink. Acest nivel functional contine intregul model al sistemului, incluzand partile software si hardware, care apoi sunt verificate impreuna la un nivel inalt de abstractizare verificand corectitudinea lor conceptuala.

O alegere fundamentala este partitionarea intre hardware si software, care decide destinatia finala a mai multor parti al design-ului. Doua fluxuri de design separate incep pentru partea de software , respectiv hardware. Partile software sunt compilate pentru elementele de procesare dorite. In fluxul de rafinament hardware, in schimb, este o distanta mare intre nivelul de specificare si nivelul de implementare.

Partea hardware este initial reprezentata intr-un limbaj de descriere hardware ( in schema la blocul RTL -Description) , este apoi rafinat la o reprezentare structurala folosind sinteza logica care produce un netlist care poate fi mapat pe o librarie de celule standard sau blocuri hard-macro pentru implementarea finala.

Aceasta reprezentare structurala este folosita pentru a produce layout-ul fizic final .

2.1 Un profil UML pentru SystemC

Un profil UML este un grup de extensii - stereotipuri, constrangeri, si valori - care adauga informatie specifica domeniului elementelor de modelare UML (sau unui subset de al lor). Un profil UML poate fi vazut ca un nou dialect al UML pentru o platforma specifica, sau pentru un domeniu de aplicatie particular cum ar fi banking , aplicatii real-time, testare, telecomunicatii, etc.

Introducerea unui profil UML pentru limbajul SystemC bazat pe specificatiile UML 2.0 si pe specificatiile SystemC 2.0

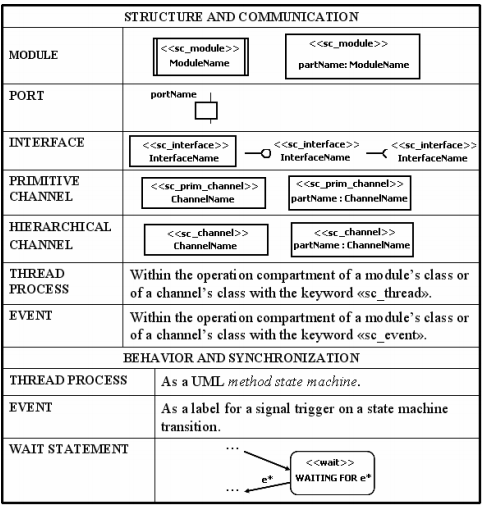


Figura3. Notatii UML pentru concepte SystemC

In figura de mai sus sunt prezentate cele mai semnificative elemente stereotip care pot fi folosite in multe diagrame structurale UML ( cum ar fi diagrame de clasa) pentru reprezentarea blocurilor structurale ale SystemC. A doua parte defineste stereotipurile care pot fi folosite in diverse diagrame de comportament (cum ar fi UML method state machine) pentru modelarea functionalitatii date de procese si canale intr-o specificare SystemC data.

Infromatiile prezentate mai sus arata cum UML si profilele UML pot fi utilizate cu succes in applicatii de tipul design SoC. Profilele UML aduc o reprezentare vizuala standardizata , usor de invatat si suportata de un numar mare de tool-uri de design, implementare, etc.

3. Modelarea structurala

Modelarea structurii interne a unui sistem-on-chip se bazeaza pe diagramele de clasa pentru definirea tipurilor din componenta blocurilor si structura compozitionala pentru definirea componentei unui SoC complet din blocurile respective. Mapara catre resursele fizice, precum domeniile de ceas este obtinuta prin aplicarea implementarii diagramelor.

Modelarea structurala captureaza caracteristicile statice ale sistemului. Aceastea sunt :

-diagramele de clasa

-diagramele de obiect

-diagramele de implementare

-diagramele de pachete

-diagramele structurale

-diagramele de componente

Modelul structural reprezinta cadrul pentru sistem si acest cadru este locul unde existat toate componentele sistemului. Astfel diagrama de clasa, diagrama de componenete si diagrama de implementare fac parte din modelarea structurala. Toate reprezinta elementele si mecanismul care le asambleaza.

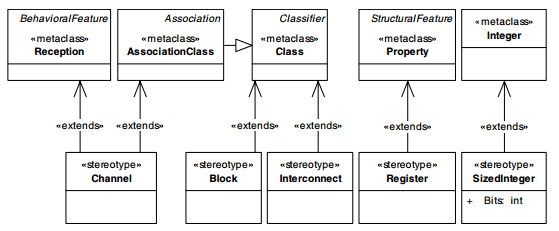


Figura4.Blocurile de modelare hardware

Un sistem-on-chip este format din blocuri sincronizate logic. Tipurile de blocuri sunt definite prin clase active specializate (Figura 4).Comportamentul unui block este definit in cadrul unei activitati private. Activitatea poate apela subactivitati in acelasi exemplu la fel de bine ca exemple detinute prin compozitie. Activitatile care aparti unei clase pot fi considerate ca fiind metodele actuale pentru operatiile unde paramentrii sunt figurati in nodurile parametrului de activitate. Cu toate aceste, datorita sincronizarii operatiilor, aceasta se potriveste doar pentru parametrii fara curent. Astfel, un block poate inovatii in receptia specializata pentru a obtine semnale care sunt privite ca token-uri in cadrul Activitatii. Atributele unui bloc sunt registrele. Tipul lor este limitat la intreg(int) si vectori (arrays). Cu toate acestea, spre deosebire de sistemele software, dimensiunea intregilor pentru sistemele hardware trebuie reparata, pentru ca acestia determina marimea circuitului rezultat.

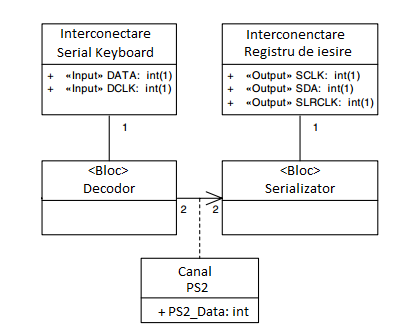


Figura 5. Diagrama de clasa a decodorului de tastatura PS2

Diferitele blocuri pot fi conectate prin intermediul interfetelor electrice. Acest lucru este reprezentat de stereotipul interconectarii. Cu toate acestea, implementarea unui astfel de interconector constituie un numar mic de linii. Este ineficient sa se realizeze un model al acestor linii. Cu toate acestea, sunt tratate drept o interfata si o clasa care o implementeaza. La nivelul de instantiere, o instanta a unei clase este presupusa a porni legaturile simetrice intre instantele participante.

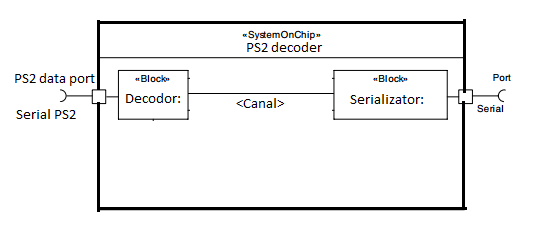


Figura 6. Diagrama de structura de decodorului PS2

Pentru sinteza, trebuie determinata o platforma aditionala specifica parametrilor fizici. O serie de parametrii fizici, precum iesirile fizice din cip, sunt calculati prin echipamente de sinteza, care nu necesita specificatii explicite. Alte proprietati precum asignarea pinilor si definirea domeniului de ceas, sunt rezultatul deciziilor de design. Asemenea proprietati trebuie reprezentate in modelul de design. Sistemele-on-cip trebuie explicit trimise pe baza domeniului de ceas. Modelul rezultat trebuie sa corespunda CSD-ului respectiv.

Din punct de vedere tehnic, domeniul de ceas, este un bloc logic sincron plasat pe cip. Din punct de vedere logic, acesta poate fi format dintr-o serie de instante(blocuri) care functioneaza sincron pe acelasi ceas. Intregul sistem-on-cip este deasemenea un domeniu de ceas care reflecta faptul ca cipul este periodicizat extern la o anumita rata. Ceasul poate fi un ceas intern sau furnizat din exterior in cazul domeniului de ceas extern. Ceasurile interne pot fi derivate de la un ceas deja existent printr-un simplu driver, sau, in cazul domeniul de ceas derivat, o expresie compelxa. Mai mult domeniile de ceas absolute, deblocheaza specificatiile frecventei absolute de ceas, care va fi implementata pe baza sistemele de ceas disponibile. Cu toate acestea, combinatia platformelor alese ( FPGA sau ASIC) si adancimea logica a circuitului real reprezinta factorii limita pentru rata de ceas a circuitului final.

Este important de mentionat ca aceeasi instanta bloc poate de asemenea sa reprezinte o parte din domeniile multiple de ceas in diferite cazuri contextuale. Astfel, diagramele de implementare respective, trebuie sa clarifice contextul prin includerea tuturor asocierilor relevante. Daca un bloc este plasat doar intr-un singur domeniu de ceas, aceste lucru nu este necesar.

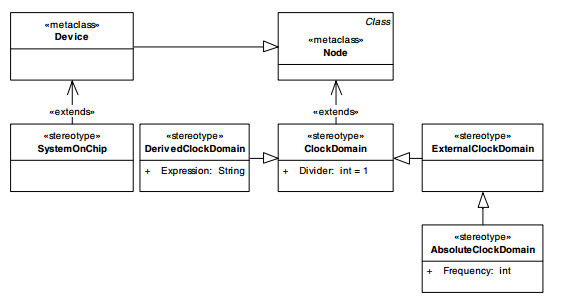


Figura7. Domeniile de ceas prin Implementare.

4. Diagrame de activitate

In timp ce modelele structurale aduc informatii despre conexiunile externe al unei instante bloc, specificarea comportamentului arata cum functioneaza in interior. Specificarea comportamentului este bazat pe activitatile reprezentate in Diagramele de Activitate. Aceste activitati reprezinta fluxul de date si procesarea intr-o instanta bloc prin elemente model pentru activitati. Aceste elemente includ actiuni interconectate de fluxuri de control sau obiecte. Nodurile de decizie, imbinare, jonctiune sunt folosite pentru controlul fluxului in activitate.

Activitatile, in acest caz pot contine trei tipuri de actiuni. Trimitere\_Semnal\_Actiune si Acceptare\_Eveniment\_Actiune sunt folosite pentru a transmite si primi date folosind un canal (Channel). Apelare\_comportament invoca sub-activitati pentru a incapsula sintaxe C in Activitati ( pentru asignari). Sintaxele C sunt de asemenea folosite pentru expresii.

Semanticile pentru jonctiuni folosite in fluxul de obiecte trebuie de asemenea sa fie definite. In acest caz consideram fluxul de obiect ca o conexiune directe intre logica si actiune. Astfel nu avem buffering. Daca am fi avut buffering, implementarea necesita o coada de tip FIFO.

Hardware-ul nu accepta re-intrari. Acest lucru se aplica si actiunilor si activitatilor din moment ce mapeaza direct o parte de circuit Hardware. Ca o consecinta, activitatile nu pot avea apeluri recursive.

Pentru exemplul nostru, specificarea comportamentului clasei Decoderului PS2 este prezentat in figura de mai jos. Aceasta activitate controleaza procesul de decodare care are mai multi pasi si se efectueaza pe un singur canal.

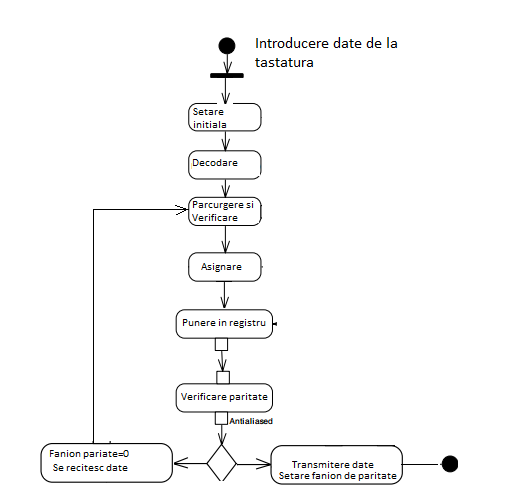


Figura8. Diagrama de activitate Decodor PS2

5. Generarea codului

Modelele SoC bazate pe exemplele anterioare pot determina generarea automata de descriptii hardware HDL sintetizabile, care pot fi transformate automat intr-un cod ASIC sau FPGA prin diferite aplicatii si tool-uri deja existente. Transformarea de la modelul SoC la un HDL particular difera, depinzand de nivelul de abstractizare adus de HDL (vezi Capitolul 2 - SoC Design Flow).

Am realizat pentru exemplul mai sus prezentat un fragment de cod VHDL ce ar putea descrie o interfata PS2 (decodare- deserializare).

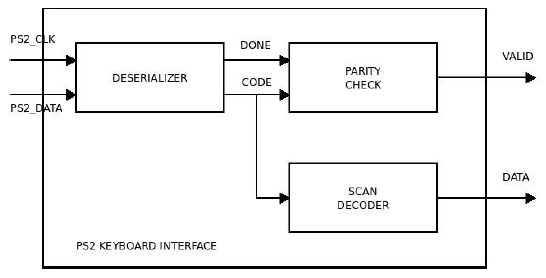


Figura9. Schema Bloc a interfetei.

Datele sunt primite de la tastatura, sunt deserializate, se verifica paritatea, apoi sunt decodate si trimise mai departe.

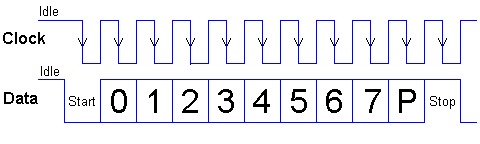


Figura10. Diagrama de timp a semnalelor Clock si Data.

Exemplu de cod posibil a fi generat de o aplicatie ce se bazeaza pe diagramele UML prezentate mai sus:

module PS2( input ps2\_clk,

input ps2\_data,

output reg [5:0] data,

output reg valid

);

reg [3:0] contor;

reg [7:0] code;

reg [7:0] code1;

reg [7:0] tasta;

reg flag;

reg parity;

reg valid;

intial begin

contor<=4'h1;

code<=8'hf0;

code1<=8'hf0;

tasta<=8'hf0;

flag<=1'b0;

end

// punem in registrul code, datele transmise de la tastatura

//la fiecare front negativ al ceasului

//apoi verificam daca este apasata o tasta, daca da o sa punem data transmisa

//in registrul tasta.

always @ (negedge clk) begin

case (contor)

//primul bit (contor=1) este bitul de start

2:code[0]<=ps2\_data;//al doilea bit este primul bit de date

3:code[1]<=ps2\_data;

4:code[2]<=ps2\_data;

5:code[3]<=ps2\_data;

6:code[4]<=ps2\_data;

7:code[5]<=ps2\_data;

8:code[6]<=ps2\_data;

9:code[7]<=ps2\_data;

10:parity<=ps2\_data;

flag<=1'b1;

11:flag<=1'b0;

endcase

if(contor<=10)begin

contor=contor+1;

end

else if (contor==11)begin

contor<=4'h1;

end

always @(posedge flag) begin

if(code==8'hf0)

tasta<=code1;

else

code1<=code;

end

end

//Citesc codul ( tasta ) verifica daca este apasata o tasta de interes

//daca da, asigneaza fiecarui bit o anumita semnificatie

always @ (negedge clk) begin

vaild<=1'b0;

if(tasta==4'h1D)begin

valid<=1;

data[5:0]<=6'b000001;//sus

end

if(tasta==4'h1B)begin

valid=1;

data[5:0]=6'b000010;//jos

end

if(tasta==4'h1A)begin

valid=1;

data[5:0]=6'b000100;//stanga

end

if(tasta==4'h23)begin

valid=1;

data[5:0]=6'b001000;//dreapta

end

if(tasta==4'h2D)begin

valid=1;

data[5:0]=6'b010000;//reset

end

if(tasta==4'h29)begin

valid=1;

data[5:0]=6'b100000;

end

end

//Verificare paritate

always@(negedge clk) begin

if ((code[0]+code[1]+code[2]+code[3]+code[4]+code[5]+code[6]+code[7])%2==0)begin

parity<=0;

else

parity<=1;

end

end

Bibliografie:

<http://www.jot.fm/issues/issue_2009_01/article1.pdf>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.4885&rep=rep1&type=pdf>

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4020201>

<http://www.lifl.fr/west/courses/SoCdesign/umlprofiles.pdf>