DE HAAGSE HOGESCHOOL
Technology, Innovation & Society Delft

VOORBLAD SCHRIFTELIJKE TOETSEN

OPLEIDING	:	**ELEKTROTECHNIEK**
TOETSCODE	:	**HM-ES-sc1**
GROEP	:	**E-EMSYS (Minor Embedded Systems)**
TOETSDATUM	:	**5 FEBRUARI 2013**
TIJD	:	**9.00 – 10.30 uur**
AANTAL PAGINA'S (incl. voorblad)	:	**5**
DEZE TOETS BESTAAT UIT	:	**3 open vragen**
GEBRUIK HULPMIDDELEN	:	**JA**
TOEGESTANE HULPMIDDELEN	:	**Tijdens dit tentamen mogen alle boeken, dictaten, aantekeningen enz. worden gebruikt.**
OVERIGE OPMERKINGEN	:	**De toetsopgaven moeten na afloop worden ingeleverd!**
OPSTELLER VAN DEZE TOETS	:	**Harry Broeders**
TWEEDE LEZER VAN DEZE TOETS	:	**John Visser**

BELANGRIJKSTE PUNTEN UIT ARTIKEL 12 VAN DE ONDERWIJS- EN EXAMENREGELING:

- je dient je via Osiris ingeschreven te hebben voor deze toets
- schrijf je naam, je studentnummer, de toetscode en de naam van de docent meteen op het tentamenpapier
- leg je identiteitsbewijs op de hoek van de tafel
- zet alle elektronische communicatiemiddelen (mobiele telefoon, PDA, etc.) uit en stop deze in je tas; deze mogen niet als calculator of klok worden gebruikt
- je mag het lokaal het eerste halfuur niet verlaten
- volg de instructies op het toetsvoorblad
- steek je hand op als je een vraag hebt

HHS TIS Delft

DE HAAGSE HOGESCHOOL

KLAS(SEN)	:	E-EMSYS (Minor Embedded Systems)	BLAD	:	1 van 4 BLADEN
TOETS	:	Hardware/Software Codesign with SystemC	DOCENT	:	Harry Broeders
CODE	:	**HM-ES-sc1**	DATUM	:	5 februari 2013
KWARTAAL	:	2 TYPE : hertentamen	TIJD	:	9.00 – 10:30

Tijdens dit tentamen mogen **alle** boeken, dictaten, aantekeningen enz. worden gebruikt.

Bij elke deelopgave staat tussen haakjes het maximale aantal te behalen punten vermeld.
Eindcijfer = (aantal behaalde punten + 10) / 10.

1. Gegeven is het volgende SystemC programma:

```
#include <systemc>
using namespace sc_core;
using namespace sc_dt;
using namespace std;

SC_MODULE(RaRa) {
    sc_in<bool> in0, in1;
    sc_out<bool> out;
    SC_CTOR(RaRa) {
        SC_METHOD(run);
        sensitive << in0 << in1;
    }
private:
    void run() {
        out.write(in0.read() != in1.read());
    }
};

SC_MODULE(TB) {
    sc_out<bool> out0, out1;
    sc_in<bool> in;
    SC_CTOR(TB) {
        SC_THREAD(run);
    }
private:
    void run() {
        out0.write(false); out1.write(false); wait(10, SC_NS);
        out0.write(true);  out1.write(false); wait(10, SC_NS);
        out0.write(false); out1.write(true);  wait(10, SC_NS);
        out0.write(true);  out1.write(true);  wait(10, SC_NS);
        out0.write(false); out1.write(false);
    }
};

int sc_main(int argc, char* argv[]) {
    RaRa rara("rara");
    TB tb("tb");
    sc_signal<bool> si0, si1, so;
    tb.out0(si0);  tb.out1(si1);  tb.in(so);
    rara.in0(si0); rara.in1(si1); rara.out(so);
    sc_trace_file *tf(sc_create_vcd_trace_file("trace"));
    tf->set_time_unit(1, SC_NS);
    sc_trace(tf, si0, "si0");
    sc_trace(tf, si1, "si1");
    sc_trace(tf, so, "so");
```

Zie volgende blad ⇨

```
    sc_start();
    sc_close_vcd_trace_file(tf);
    return 0;
}
```

(10) Teken het **timingsdiagram** dat door dit programma gegenereerd wordt.

2.

Gegeven is de volgende SystemC module:

```
template <int N>
SC_MODULE(Sqrt) {
    sc_in<sc_uint<N*2> > n_i;
    sc_out<sc_uint<N> > sqrt_o;
    SC_CTOR(Sqrt) {
        SC_THREAD(run);
        sensitive << n_i;
    }
private:
    void run() {
        while (1) {
            wait();
            wait(10, SC_NS); // time needed for load
            sc_uint<2*N> n = n_i.read();
            sc_uint<N> a = 0;
            sc_uint<N> b = 1 << (N-1);
            wait(10, SC_NS); // time needed for compare
            while (b != 0) {
                wait(10, SC_NS); // time needed for addition
                a = a + b;
                wait(N * 10, SC_NS); // time needed for multiply
                wait(10, SC_NS); // time needed for compare
                if (a * a > n) {
                    wait(10, SC_NS); // time needed for substract
                    a = a - b;
                }
                wait(10, SC_NS); // time needed for shift
                b = b >> 1;
                wait(10, SC_NS); // time needed for compare
            }
            sqrt_o.write(a);
        }
    }
};
```

Deze module berekent sqrt(`n_i`), oftewel √`n_i`. Het model werkt met gehele getallen. Het getal `n_i` is 2*`N` bits breed en het resultaat `sqrt_o` is `N` bits breed (dat is namelijk altijd genoeg). Het model is een zogenoemd *approximately-timed* model.

Zie volgende blad ⇨

De bewerking `1 << (N-1)` is een schuifbewerking. De constante 1 wordt `N-1` bitposities naar links geschoven. Dus als `N` 5 is dan is het resultaat van deze bewerking $10000_{binair} = 16_{decimaal}$.
De bewerking `b = b >> 1` is ook een schuifbewerking. De inhoud van de variabele `b` wordt 1 positie naar rechts geschoven. Dus als `b` $16_{decimaal}$ (= 10000_{binair}) is dan is `b` na deze bewerking gelijk aan $8_{decimaal}$ (= 1000_{binair}).

A. (15) Als de bovenstaande module `Sqrt` wordt geïnstantieerd voor `N`=5 en gebruikt wordt om de wortel van 399 uit te rekenen wat zal dan het antwoord zijn? **Verklaar je antwoord door de waarden van de variabelen** `a` **en** `b` **tijdens het doorlopen van de** `run` **functie te laten zien!**

B. (15) Als de bovenstaande module `Sqrt` wordt geïnstantieerd voor `N`=5 en gebruikt wordt om de wortel van 399 uit te rekenen hoeveel (gesimuleerde) tijd zal dan nodig zijn om het antwoord te berekenen? **Verklaar je antwoord door (gesimuleerde) tijdmarkeringen toe te voegen bij de waarden van de variabelen** `a` **en** `b` **tijdens het doorlopen van de** `run` **functie!**

De volgende SystemC module is ontwikkeld om het antwoord op vraag A en B te vinden:

```
SC_MODULE(W399) {
public:
    sc_in<sc_uint<5> > sqrt_i;
    sc_out<sc_uint<10> > n_o;
    SC_CTOR(W399) {
        SC_THREAD(run);
        sensitive << sqrt_i;
    }
private:
    void run() {
        n_o.write(399);
        wait();
        cout<<"@: "<<sc_time_stamp()<<": sqrt(399) = "
                                                <<sqrt_i.read()<<endl;
    }
};
```

C. (10) Schrijf een SystemC **hoofdprogramma** genaamd `sc_main` waarin:
- de bovenstaande module `Sqrt` wordt geïnstantieerd voor `N`=5;
- de module `W399` wordt geïnstantieerd;
- de benodigde SystemC *channels* worden geïnstantieerd;
- de juiste verbindingen worden gemaakt en
- de simulatie wordt gestart.

Zie volgende blad ⇨

Als we de module `Sqrt` *cycle-accurate* willen maken dan moet een `clk` ingang, een `go` ingang en een `done` uitgang worden toevoegen aan deze module. Ga uit van een clock met een periodetijd van 10 ns. **Let op:** laat de timing van het *cycle-accurate* model zoveel mogelijk gelijk blijven met de timing van het *approximately-timed* model.

D. (20) Geef aan hoe de module `Sqrt` *cycle-accurate* gemaakt kan worden door het toevoegen van de hierboven genoemde signalen en het aanpassen van de `wait` statements. De module `W399` zou nu natuurlijk ook aangepast moeten worden maar dat hoef je bij deze toets **niet** te doen.

E. (15) We willen vervolgens een RTL model van deze module maken bestaande uit een datapath en een controller (zoals op het college behandeld). Geef het **blokschema** van het **datapath** op RTL niveau. Maak gebruik van de volgende RTL bouwstenen: register, multiplexer, multiplier, substractor, adder en comperator. De operatie `b = b >> 1` kun je implementeren door de uitgang van het `b` register verschoven in het `b` register in te klokken. Geef ook duidelijk aan welke data- en statussignalen door het datapad gegenereerd worden en welke data- en controlsignalen door het datapad gebruikt worden. Geef bij alle datasignalen duidelijk aan hoe breed ze zijn. (in bits).

3.

(5) Tijdens het gastcollege werd verteld dat NXP gebruik maakt van SystemC TLM2 modellen. Maakt NXP gebruik van *loosely-timed* of van *approximately-timed* TLM modellen? Verklaar je antwoord! Je antwoord **moet** de volgende vorm hebben: NXP gebruikt ….-timed TLM modellen omdat ….

Succes