ΗΥ-225: Οργάνωση Υπολογιστών Τμ. Επ. Υπολογιστών
Άνοιξη 2007 © Πανεπιστήμιο Κρήτης

# Σειρά Ασκήσεων 8:
# Πρώτη Γνωριμία με την Γλώσσα Verilog

Προθεσμία έως Δευτέρα 30 Απριλίου 2007, ώρα 23:59 (βδομάδα 6)
Υπενθύμιση Διαγωνισμού Προόδου: Σάββατο, 12 Μαΐου 2007, ώρα 2:10 μ.μ.

## 8.1 Γλώσσες Περιγραφής Hardware (HDL):

Σήμερα, η σχεδίαση hardware σε βιομηχανικό επίπεδο στηρίζεται πρώτ' απ' όλα στις Γλώσσες Περιγραφής Hardware (Hardware Description Languages - HDL). Χρησιμοποιώντας αυτές, μπορεί κανείς να περιγράψει με ακρίβεια την επιθυμητή λειτουργία ή την υλοποίηση ενός ψηφιακού συστήματος, ή (συνήθως) και τα δύο. Αυτό είναι εξαιρετικά σημαντικό, πρώτον γιά την *προσομοίωση* (simulation) ενός σχεδίου, δηλαδή γιά να προσπαθήσει ένα πρόγραμμα υπολογιστή, ο "προσομοιωτής", να μιμηθεί τον αναμενόμενο τρόπο λειτουργίας του πραγματικού ψηφιακού συστήματος, δίνοντας μας έτσι τη δυνατότητα να βρούμε και να διορθώσουμε τα λάθη σχεδίασης, καθώς επίσης και να μελετήσουμε την επίδοση του υπό σχεδίαση συστήματος. Δεύτερον, όταν η περιγραφή του υπό σχεδιαση ψηφιακού συστήματος γίνει με κατάλληλο τρόπο, υπάρχουν εργαλεία αυτόματης σύνθεσης που μπορούν διαβάζοντας αυτή την περιγραφή να την μετατρέψουν σε άλλη μορφή, χαμηλότερου επιπέδου, κατάλληλη γιά αυτόματη κατασκευή του πραγματικού συστήματος (κάτι σαν compilation προγράμματος, δηλαδή).

Στο μάθημά μας θα χρησιμοποιήσουμε τη γλώσσα περιγραφής hardware *"Verilog"* που είναι η μία από τις δύο δημοφιλέστερες (παγκοσμίως) HDL --η άλλη είναι η "VHDL". Η Verilog είναι μεγάλη και περίπλοκη γλώσσα, με πολλές δυνατότητες και πολλούς τρόπους να κάνει κανείς το κάθε τι. Σε αυτό το μάθημα θα μάθετε και θα χρησιμοποιήσουμε ένα πολύ μικρό, μόνο, υποσύνολο της Verilog. Ο σκοπός των σημερινών ασκήσεων είναι η εξοικείωσή σας με την Verilog και τα συναφή εργαλεία:

- Θα δείτε πώς ένα πρώτο, απλούστατο κύκλωμα περιγράφεται σε Verilog, και πώς περιγράφουμε τις κυματομορφές εισόδου του.
- Θα προσομοιώσετε το κύκλωμα αυτό χρησιμοποιώντας τον interpreter "**Verilog-XL**".
- Θα δείτε γραφικά τις κυματομορφές εξόδου του κυκλώματος με το εργαλείο "**Signalscan**".

## 8.2 Μοντέλο Καθυστερήσεων:

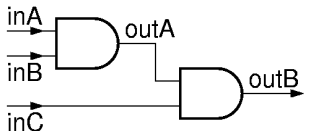
Θα χρησιμοποιήσουμε βιβλιοθήκες δομικών στοιχείων (πυλών, κλπ) που ακολουθούν βασικά το παρακάτω μοντέλο καθυστρήσεων. Κάθε πύλη AND στις σημερινές ασκήσεις θα έχει μέγιστη καθυστέρηση 200 ps και ελάχιστη καθυστέρηση μηδέν. Αυτό σημαίνει ότι όταν αλλάζει μία είσοδος η οποία προκαλεί αλλαγή στην τιμή της εξόδου, η νέα τιμή στην έξοδο εμφανίζεται κάποια στιγμή μεταξύ 0 και 200 ps. Το γεγονός ότι δεν γνωρίζουμε ποιά ακριβώς στιγμή σταθεροποιείται η έξοδος το μοντελοποιούμε στη Verilog θέτοντας την έξοδο στη τιμή x ("άγνωστο") για το διάστημα από 0 μέχρι 200 ps. Η τιμή x στην έξοδο μίας πύλης μπορεί να είναι είσοδος σε μιαν άλλη πύλη.

Όταν μιά πύλη έχει εισόδους x, δηλαδή εισόδους με άγνωστη τιμή, τότε η τιμή εξόδου της προκύπτει ως εξής. Θεωρούμε όλους τους δυνατούς συνδυασμούς 1 και 0 σε όλες τις εισόδους με τιμή x. Γιά κάθε τέτοιο συνδυασμό εξετάζουμε τι τιμή θα είχε η έξοδος. Αν όλες αυτές οι τιμές είναι 0, τότε η τιμή εξόδου είναι 0. Αν όλες οι παραπάνω τιμές είναι 1, τότε η τιμή εξόδου είναι 1. Αν όμως μερικές από τις παραπάνω τιμές είναι 0 και μερικές είναι 1, τότε η τιμή εξόδου είναι x, που σημαίνει ότι η αβεβαιότητα που υπάρχει γιά τις τιμές των εισόδων που είναι x, επηρεάζει την έξοδο σε αυτή την περίπτωση, και κάνει την τιμή της να είναι επίσης αβέβαιη. Με βάση αυτή τη λογική, η τιμή εξόδου μιάς πύλης AND και μιάς πύλης OR, όταν οι είσοδοί τους είναι 0, 1, ή x, θα είναι:

```
AND: inA-> 0   1   x       OR: inA-> 0   1   x
     inB:                       inB:
      0    0   0   0             0    0   1   x
      1    0   1   x             1    1   1   1
      x    0   x   x             x    x   1   x
```

## Άσκηση 8.3: Περιγραφή Κυκλώματος σε Verilog

Στις σημερινές ασκήσεις θα περιγράψουμε και προσομοιώσουμε το απλούστατο κύκλωμα που φαίνεται στο σχήμα. Ο κώδικας που το περιγράφει σε Verilog δίδεται εδώ:




```
module top;

    wire outA, outB;
    reg  inA, inB, inC;

    // Instantiate two AND gates:
    //
    lib8_and and1(outA, inA,  inB);
    lib8_and and2(outB, outA, inC);

endmodule
```

Στην περιγραφή αυτή ορίζεται ένα "module" με το όνομα "top". Μέσα του, στην αρχή, δηλώνονται τα σύρματα (wire) outA και outB (που είναι έξοδοι πυλών), καθώς και οι είσοδοι πυλών inA, inB, και inC. Παρά το γεγονός ότι και αυτά τα σήματα (inA, inB, inC) είναι τρόπον τινά απλά σύρματα, πρέπει να δηλωθούν "reg", επειδή στο κύκλωμά μας δεν υπάρχει κανείς που να τα οδηγεί, και κατά συνέπεια, ο μόνος τρόπος να κρατάνε ό,τι τιμή τους βάζουμε είναι να δηλωθούν "reg" (κάτι σαν "καταχωρητής", αλλά όχι το ίδιο με τους δικούς μας καταχωρητές).

Στη συνέχεια, μέσα στο module "top", υλοποιούμε (instantiate) δύο αντίτυπα μιάς πύλης AND. Και τα δύο είναι αντίτυπα της ίδιας πύλης ονόματι "`lib8_and`" από μιά κατάλληλη βιβλιοθήκη που θα πούμε παρακάτω. Το πρώτο αντίτυπο (instance) είναι το "and1", και το δεύτερο είναι το "and2". Η πύλη lib8_and έχει οριστεί με τρείς "πόρτες", δηλαδή τρία σήματα επικοινωνίας με τον έξω (της) κόσμο: το πρώτο (σε σειρά) είναι η έξοδός της, και τα επόμενα δύο είναι οι είσοδοί της. Επομένως, βάσει αυτής της σειράς ορισμού, που πρέπει να μας την έχει δώσει αυτός που έφτιαξε τη βιβλιοθήκη, στο αντίτυπο and1 η έξοδος συνδέεται στο (τροφοδοτεί το) σήμα outA, ενώ οι είσοδοι συνδέονται στα (τροφοδοτούνται από τα) σήματα inA και inB. Στο αντίτυπο and2, η έξοδος πάει στο outB, ενώ είσοδοι είναι τα outA και inC.

Το module `top`, έτσι "σκέτο" όπως το ορίσαμε παραπάνω, αν περάσει από τον προσομοιωτή "Verilog-XL" της Verilog, δεν θα βγάλει κανένα αποτέλεσμα, αφού κανένα ενδιαφέρον συμβάν (γεγονός - event) δεν γεννιέται από καμία πηγή: δεν υπάρχουν συγκεκριμένα σήματα εισόδου για να δώσουν εξόδους. Επομένως πρέπει να επεκτείνουμε το module έτσι ώστε να εφαρμόσουμε ακολουθίες εισόδου (test vectors) και να παρατηρήσουμε τις τιμές σε εξωτερικούς και εσωτερικούς κόμβους του κυκλώματος. Οι επεκτάσεις αυτές δεν αποτελούν τμήμα του κυκλώματος, αλλά το περιβάλλον ελέγχου ("test bench"). Για λόγους απλότητας, σε αυτή την άσκηση, θα τοποθετήσουμε την περιγραφή του κυκλώματος στο ίδιο αρχείο με το περιβάλλον ελέγχου:

```
module top;
    wire outA, outB;
    reg  inA, inB, inC;

    // Instantiate two AND gates:
    //
    lib8_and and1(outA, inA,  inB);
    lib8_and and2(outB, outA, inC);

    // Test Bench
    //
    initial
    begin
        // Start Tracing (signalscan)
        //
        $shm_open("Test.shm");
        $shm_probe(top, "AS");

        // print the values of "outA" and "outB" at stdout
        // each time one of them changes
        //
        $monitor($time, ": outA=%b, outB=%b\n", outA, outB);
```

```
        // Test vectors
        //
        #100    inA = 1; inB = 1; inC = 1;
        #900    inA = 0;
        #1000   inA = 1;
        #1000   inB = 0; inC = 0;
        #1000   inA = 0;
        #1000   inA = 1;
        #1000   inB = 1; #100 inA = 0;
        #1000

        // Stop  Tracing
        //
        $shm_close();
    end
endmodule
```

- Τα τμήματα των γραμμών μετά το "**//**" είναι σχόλια.
- Το περιβάλλον ελέγχου (test bench) βρίσκεται μέσα σε μιάν εντολή "**initial**" (που το block της καθορίζεται από το ζευγάρι begin-end). Η εντολή "initial" σημαίνει "κάνε αυτά που σου λέω μία φορά, αρχίζοντας από την αρχή του χρόνου (από το χρόνο 0)".
- Οι γραμμές "**$shm_open**" και "**$shm_probe**" είναι οδηγίες στον προσομοιωτή να κρατήσει πληροφορίες γιά όσα συμβαίνουν στο module top, μέσα στο directory "Test.shm", μέχρι να συναντήσουμε την οδηγία "**$shm_close**".
- Η οδηγία "**$monitor**" λέει στον προσομοιωτή να μας τυπώνει τον παρόντα χρόνο ακολουθούμενο από τις τιμές των σημάτων outA και outB, όποτε αλλάζει κάποιο από αυτά. Η σύνταξη της οδηγίας αυτής θυμίζει τη σύνταξη της printf στη C ("%b" σημαίνει "binary").
- Η εντολή "***#αριθμός***", στην Verilog, σημαίνει "περίμενε να περάσει τόση ώρα όση λέει ο αριθμός, και μετά προχώρα σε ό,τι λέει η επόμενη εντολή". Έτσι, στο κομάτι με τις κυματομορφές εισόδου (test vectors), η πρώτη γραμμή (#100) λέει "περίμενε 100 ps" (μετά την αρχή του initial, άρα μετά το χρόνο 0), δηλαδή περίμενε να φτάσει ο χρόνος το 0 + 100 = 100 ps, και τότε κάνε το σήμα inA να γίνει 1, το σήμα inB να γίνει επίσης 1, και το ίδιο και το σήμα inC. (Ερώτηση: φροντίστε να διαπιστώσετε τι τιμή δίνει η Verilog-XL σε αυτά τα σήματα όταν αρχίζει να δουλεύει ο προσομοιωτής, δηλαδή μεταξύ χρόνου 0 και 100 ps).
- Η επόμενη γραμμή λέει "περίμενε (άλλα) 900 ps", δηλαδή περίμενε να φτάσει ο χρόνος στα 100 + 900 = 1000 ps, και τότε κάνε το σήμα inA να γίνει 0 (εν τω μεταξύ, το σήμα αυτό κρατούσε την τιμή 1 που του είχαμε δώσει τελευταία, επειδή έχει δηλωθεί τύπου "reg"). Η επόμενη γραμμή λέει "περίμενε (άλλα) 1000 ps", δηλαδή περίμενε να φτάσει ο χρόνος στα 1000 + 1000 = 2000 ps, και τότε ξανακάνε το σήμα inA = 1, κ.ο.κ. Στο τέλος, περιμένουμε 1000 επιπλέον ps μετά την τελευταία αλλαγή εισόδου, προκειμένου να δούμε τις συνέπειες που είχε αυτή η τελευταία αλλαγή σε μετέπειτα χρονικές στιγμές, και μετά τερματίζουμε την προσομοίωση.

Γράψτε το παραπάνω module στο αρχείο σας "**ask8.v**". Επίσης, αντιγράψτε στην περιοχή σας το αρχείο:

**~hy225/verilog/lib/lib8.v**

που περιέχει τα βοηθητικά modules για την πύλη AND και τους ορισμούς των καθυστερήσεων.

Ο προσομοιωτής της Verilog και το εργαλείο ελέγχου κυματομορφών που θα χρησιμοποιήσετε --της εταιρείας Cadence και τα δύο-- είναι εργαλεία μεγάλης εμπορικής αξίας, τα οποία το Πανεπιστήμιο Κρήτης έχει αγοράσει γιά εσωτερική χρήση και γιά εκπαιδευτικούς και ερευνητικούς σκοπούς **μόνο**, με τη βοήθεια του οργανισμού "Europractice" της Ευρωπαϊκής Ενωσης, και τρέχουν μόνο κάτω από αυστηρούς περιορισμούς χρήσης (license). Τα εργαλεία αυτά είναι διαθέσιμα μόνο για μηχανήματα αρχιτεκτονικής Sparc/Solaris. Για τους σκοπούς του μαθήματος θα σας δοθεί πρόσβαση στα παρακάτω μηχανήματα, που έχουν τέτοια κατάλληλη αρχιτεκτονική και στα οποία τρέχουν τα παραπάνω εργαλεία. Για να τρέξετε τον προσομοιωτή:

```
% xhost +
% rlogin  [katallilo Mixanima]
% setenv  DISPLAY [arxiko Mixanima]:0.0
% source  ~hy225/verilog/scripts/cds_ldv.sh
% verilog lib8.v ask8.v
```

όπου *[arxikoMixanima]* είναι το όνομα της μηχανής από την οποία κάνετε rlogin, και *[katallilo Mixanima]* είναι ένα από τα:

**apraktias, kirkios, typhon, apiliotis, levantes, pounentes, garbis**

Το script "**cds_ldv.sh**" προσθέτει τα paths και environment variables που χρειάζεστε γιά να τρέξετε

τα εργαλεία (εάν το shell σας παραπονεθεί ότι "setenv: too many arguments", θέσετε τις μεταβλητές του "PATH" και "LD_LIBRARY_PATH" στα απολύτως απαραίτητα και μόνο, πρίν κάνετε source το cds_ldv.sh --επίσης, μην κάνετε source το cds_ldv.sh πάνω από μία φορά σε κάθε παράθυρο). Η τελευταία από τις παραπάνω εντολές τρέχει την Verilog-XL. Τα ονόματα των δύο αρχείων με κατάληξη ".v" πρέπει να δωθουν ως παράμετροι με τη σειρά που φαίνονται παραπάνω, διότι πρώτα πρέπει να οριστούν η πύλη lib8_and κλπ. στο αρχείο "lib8.v", και μετά να την χρησιμοποιήστε εσείς στο αρχείο "ask8.v". Όταν θα τρέξει ο προσομοιωτής, ελέγξτε ότι οι έξοδοι έχουν την αναμενόμενη τιμή, με βάση τις εισόδους που δώσαμε στο test bench.

## Άσκηση 8.4: Ελεγχος Κυματομορφών με το Signalscan

Οταν τελειώσει η προσομοίωση, παρατηρήστε ότι στο directory που τρέξατε το πρόγραμμα υπάρχει και ένα νέο subdirectory ονόματι "**`Test.shm`**". Αυτό δημιουργείται από τις οδηγίες στο `ask8.v` που είπαμε, και περιέχει όλες τις τιμές για όλα τα σήματα καθ' όλη τη διάρκεια της προσομοίωσης. Χρησιμοποιήστε το εργαλείο "**Signalscan**", πάντα στα ίδια κατάλληλα μηχανήματα όπως παραπάνω, γιά να δείτε μιά γραφική αναπαράσταση των κυματομορφών του κυκλώματός σας:

```
% cd Test.shm
% signalscan
```

Αφού τρέξει το γραφικό περιβάλλον του προγράμματος, ακολουθήστε τα εξής βήματα:

- Ανοίξτε το αρχείο της προσομοίωσης με "File->Open Simulation File", και διαλέγοντας το αρχείο "**`Test.trn`**".
- Επιλέξτε "Windows->Design Browser".
- Από το υποπαράθυρο "Instances in Current Context" βλέπετε τα ονόματα των modules που έχουν οριστεί (με ιεραρχικό τρόπο). Επιλέξτε το top.
- Γιά να μετακινηθούμε σε υψηλότερο επίπεδο της ιεραρχίας, κάνουμε δεξί-κλικ μέσα στο υποπαράθυρο "Instances in Current Context".
- Απο το πιο κάτω υποπαράθυρο (Nodes/Variables in Current Context), επιλέξτε όλα τα σήματα του top (inA, inB, inC, outA, outB).
- Πατήστε το "Add, Close" κουμπί (πάνω δεξιά). Έτσι, τα επιλεγμένα σήματα μεταφέρονται στο κεντρικό παράθυρο και ο Design Browser κλείνει.
- Ολα τα σήματα που μας ενδιαφέρουν φαίνονται πλέον. Ελέγξτε και πάλι τις τιμές τους.

Το signalscan αναπαριστά τη τιμή x με κόκκινη γραμμή τόσο στο επίπεδο 0 όσο και στο επίπεδο 1. Παρατηρήστε τη συμπεριφορά των εξόδων σε κάθε μία από τις χρονικές στιγμές 2ns, 3ns, 4ns, 6ns. Πόση ώρα μένει η έξοδος στην τιμή x σε κάθε περίπτωση και γιατί;

**ΠΡΟΣΟΧΗ:** σε ώρες πυκνής χρήσης, αποφύγετε να μένετε πολλή ώρα μέσα στο εργαλείο Signalscan: έχουμε περιορισμένο αριθμό αδειών χρήσης (licenses), κι έτσι άλλοι συνάδελφοί σας μπορεί να μην μπορούν να μπούν σε αυτό. (Το ίδιο ισχύει και γιά την Verilog XL, αλλά αυτό το εργαλείο ούτως ή άλλως τρέχει πολύ γρήγορα γιά τα (πολύ μικρά!) δικά σας κυκλώματα...).

Παραδώστε, όπως και στις προηγούμενες ασκήσεις, ένα χαρακτηριστικό στιγμιότυπο από το Signalscan της άσκησης 8.4, "**`ask8.4.jpg`**", και τα σχόλια σας για τις τιμές στις εξόδους, "**`ask8.4.txt`**", πακεταρισμένα στο αρχείο "**`ask8.tar`**", μέσω:

```
tar -cvf ask8.tar  ask8.4.jpg ask8.4.txt
~hy225/bin/submit 8
```

---




Last updated: 23 Apr. 2007, by M. Katevenis.