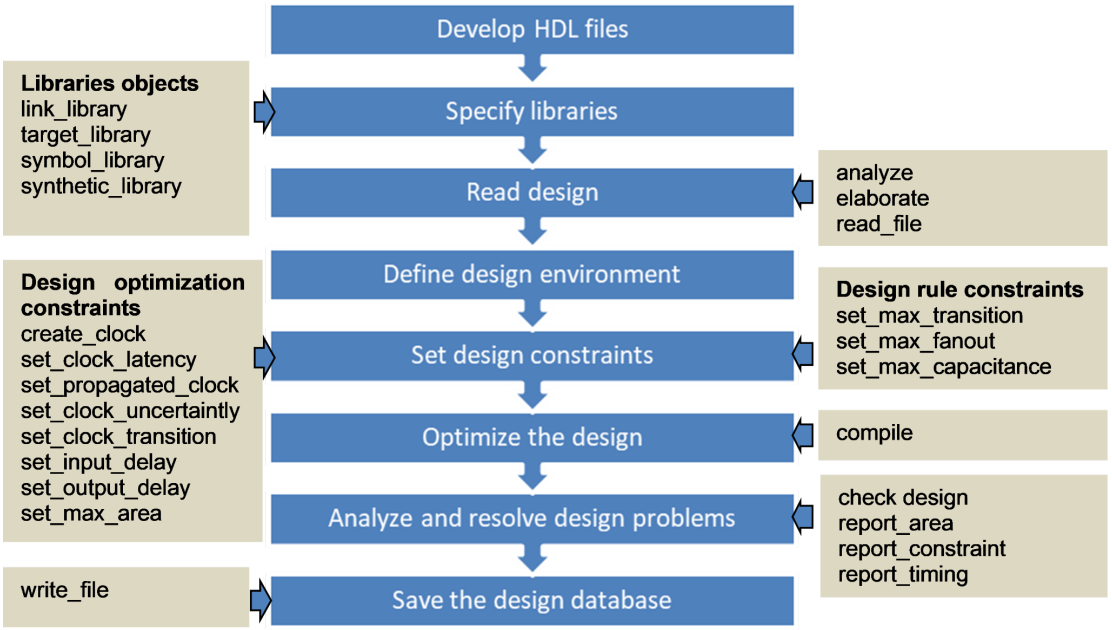
# 4η Εργαστηριακή Άσκηση

*Εξοικείωση με την Σύνθεση Κυκλωμάτων*

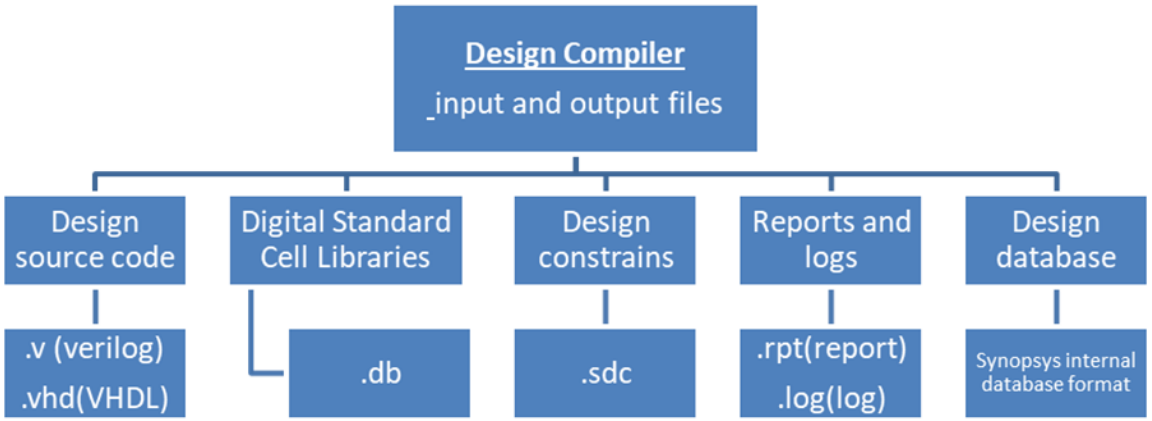
**Εξέταση Άσκησης:** 10/5/2023
**Παράδοση αναφοράς:** 14/5/2023

Στην 4η εργαστηριακή άσκηση να εξοικειωθείτε με την σύνθεση κυκλωμάτων που έχουν περιγραφεί σε HDL (Verilog). Για σύνθεση θα χρησιμοποιήσετε το εργαλείο Design Compiler (DC) της Synopsys. User Guide για το εργαλείο θα βρείτε στα έγγραφα του eclass. Στο user guide μπορείτε να βρείτε αναλυτικές λεπτομέρειες για όλες τις εντολές και όλα τα features του εργαλείου καθώς και διάφορα παραδείγματα.

## Synthesis Flow




## Design Compiler input and output files



Αντιγράψτε στο home σας τα απαιτούμενα αρχεία για την 4η εργαστηριακή άσκηση:

```
cp -r /usr/local/vlsi2_2023/lab4 .
cd lab4
```

Στον φάκελο `src` μπορείτε να βρείτε τους πηγαίους κώδικες των designs για τα παραδείγματα αυτής της άσκησης. Στο φάκελο `scripts` υπάρχουν διάφορα script αρχεία που σχετίζονται με την άσκηση.

Το αρχείο `.synopsys_dc.setup` διαβάζεται αυτόματα από τον DC κατα την έναρξη και εκεί μπορούν να οριστούν διάφορες παράμετροι. Για παράδειγμα:

```
$ cat .synopsys_dc.setup
set target_library {/usr/local/eda/synLibs/asap7/7nm/db/asap7.db}
set link_library {* /usr/local/eda/synLibs/asap7/7nm/db/asap7.db}
```

Σε αυτήν την εργαστηριακή άσκηση έχουμε ορίσει στον DC να χρησιμοποιήσει την βιβλιοθήκη asap7 για να κάνει build και map τα κυκλώματά σας.

## **Παράδειγμα 1**

A1. Ξεκινήστε τον Design Compiler

```
dc_shell -gui
```

A2. Κάντε analyze το design σας

```
analyze -format verilog {./src/rca.v ./src/fulladder.v}
```

Η εντολή analyze κάνει τα ακόλουθα:

- Διαβάζει αρχεία HDL
- Ελέγχει για λάθη
- Δημιουργεί HDL library objects σε μια ενδιάμεση αναπαράσταση HDL-independent

A3. Κάντε elaborate το design σας

```
elaborate rca
```

Η εντολή elaborate κάνει τα ακόλουθα

- Μεταφράζει το design από την ανάλυση σε ένα technology-independent design
- Επιτρέπει να αλλάξουν οι τιμές των παραμέτρων
- Αντικαθιστά τους αριθμητικούς τελεστές με κυκλώματα από την βιβλιοθήκη DesignWare
- Κάνει link για να επιλυθούν όλα τα references

A4. Για να ολοκληρωθεί το design σας πρέπει να γίνει link με όλα τα components της βιβλιοθήκης και τα άλλα design που χρησιμοποιεί. Τρέξτε:

```
link
```

A5. Ελέγξτε για τυχόν λάθη ή warnings:

```
check_design
```

A6. Στη συνέχεια πρέπει να θέσετε χρονικούς περιορισμού στο design σας. Για παράδειγμα:

```
set_max_delay 200 -from [all_inputs] -to [all_outputs]
```

Η εντολή αυτή λέει ότι η μέγιστη καθυστέρηση από όλες τις εισόδους σε όλες τις εξόδους πρέπει να είναι το πολύ 200 μονάδες χρόνου. Συνήθως οι χρονικοί περιορισμοί μπορεί να είναι αρκετά πολύπλοκοι να να χρησιμοποιούνται σε διάφορα σημεία της διαδικασίας και για αυτό περιγράφονται σε ξεχωριστό αρχείο π.χ. file.sdc και τους διαβάζετε με την εντολή `read_sdc file.sdc`.

A7. Επίσης, μπορείτε να θέσετε επιπλέον περιορισμούς όπως η επιφάνεια:

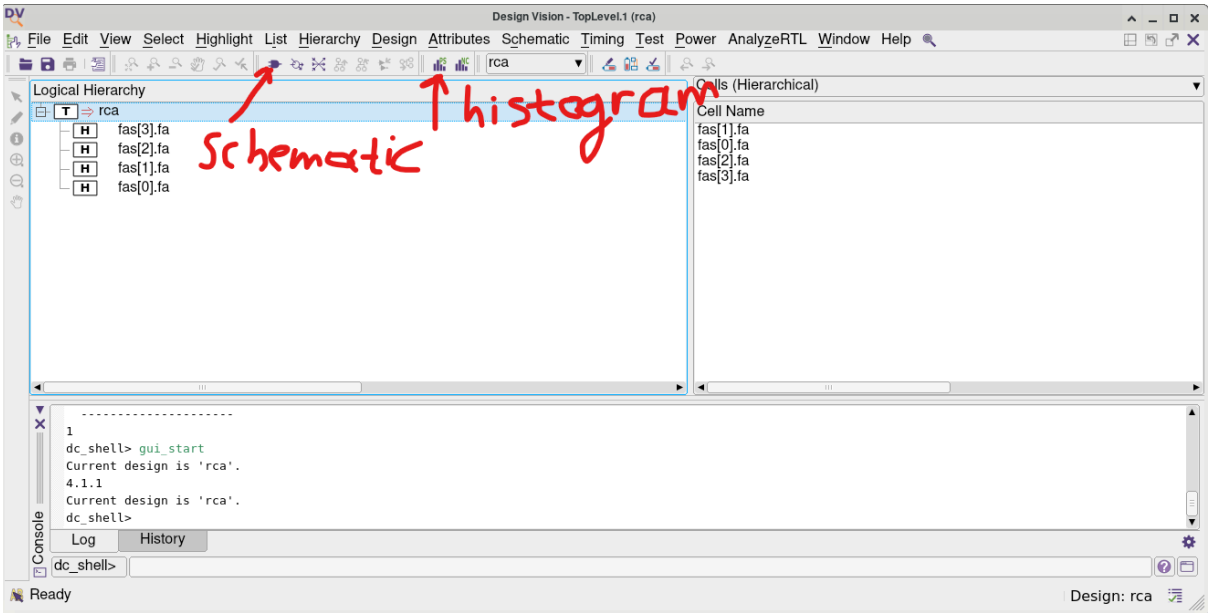
```
set_max_area 0
```

A8. Στη συνέχεια μπορείτε να συνθέσετε το κύκλωμά σας

```
compile
```

A9. Δείτε το schematic του κυκλώματος που συνθέσατε (schematic button) και πλοηγηθείτε στο schematic viewer.

A10. Επίσης, δημιουργήστε και μελετήστε το histogram των καθυστερήσεων των μονοπατιών του κυκλώματός σας.



A11. Ο DC μπορεί να παράξει πολλά report:

```
exec mkdir -p -- ./results_rca
report_timing > ./results_rca/timing_report.compile.200.rpt
report_qor > ./results_rca/qor_report.compile.200.rpt
report_area -hierarchy > ./results_rca/area_report.compile.200.rpt
report_power -hierarchy > ./results_rca/power_report.compile.200.rpt
```

A12. Επίσης μπορεί να σας δώσει και την περιγραφή του κυκλώματος μετά τη σύνθεση (synthesized netlist):

```
write -hierarchy -format verilog \
-output ./results_rca/rca_compiled.compile.v
```

A13. Κλείστε τον DC

```
exit
```

Μελετήστε τα report που αποθηκεύσατε αρχικά καθώς και το netlist που πήρατε.

Επίσης, μπορείτε να γράψετε αυτές τις εντολές σε ένα tcl script και να τις τρέξτε.

B1. Για παράδειγμα εκτελέστε:

```
dc_shell -f scripts/rca_syn.tcl
```

**Ζητούμενα**

**Ζητούμενο 1.1** Στο παράδειγμα 1 πειραματιστείτε με διάφορες τιμές για την καθυστέρηση (εντολή A6). Τι παρατηρείτε;

**Ζητούμενο 1.2** Τι διαφορά παρατηρείτε στις εντολές που εκτελούνται στα A1-A13 και B1. Για την ίδια καθυστέρηση (π.χ. 200 ) συγκρίνετε την επιφάνεια του κυκλώματος που παράγεται καθώς. Στο script `scripts/rca_syn.tcl` αντικαταστήσετε την εντολή quit με την εντολή gui_start και συγκρίνετε το schematic που παίρνετε σε σχέση με αυτό που πήρατε από την διαδικασία A1-A13. Μπορείτε αντίστοιχα να συγκρίνετε τις αντίστοιχες παραγόμενες netlists.

**Ζητούμενο 2.1** Σχεδιάστε έναν 4-bit rca του οποίου οι είσοδοι και έξοδοι πάνε σε registers. Συνθέσετε το κυκλωμά σας. Αντι για την εντολή `set_max_delay` να κάνετε `read_sdc scripts/rca_clock.sdc`. Μελετήστε το `scripts/rca_clock.sdc` και αλλάξτε το καταλλήλως. Ποια είναι η μέγιστη συχνότητα που θα μπορούσε να λειτουργήσει το κύκλωμα που συνθέσατε;

**Ζητούμενο 2.2** Τροποποιείστε το 2.1 ώστε ο rca να έχει 2-stage pipeline. Ανατρέξτε στο documentation του DC και διαβάστε για retiming. Αναλύστε το κύκλωμά σας (area-delay) με και χωρίς retime.

**Ζητούμενο 3.1** Συνθέστε τον accumulator της 2ης εργαστηριακής άσκησης. Ποια είναι η ελάχιστη καθυστέρηση και η επιφάνεια των κυκλωμάτων σας;

**Ζητούμενο 3.2** Συνθέστε το κύκλωμα Vending Machine της 3ης εργαστηριακής άσκησης. Ποια είναι η ελάχιστη καθυστέρηση και η επιφάνεια των κυκλωμάτων σας;

**Ζητούμενο 4** Υλοποιήστε ένα κύκλωμα υπολογισμού Μέγιστου Κοινού Διαιρέτη (ΜΚΔ) δύο ακεραίων.

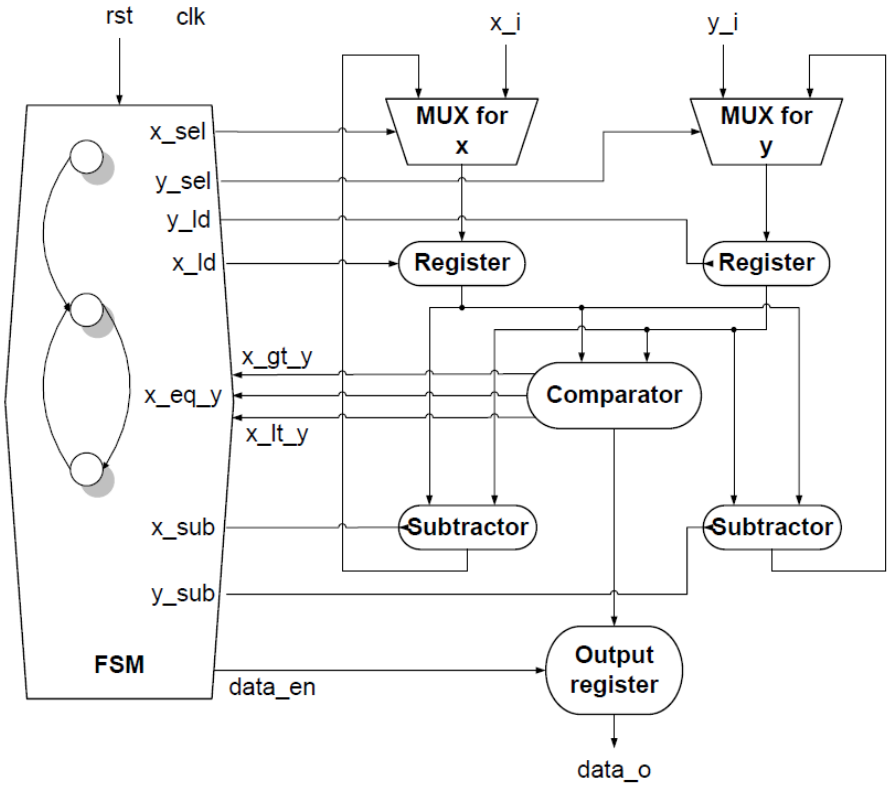
Ας υποθέσουμε ότι έχουμε να υλοποιήσουμε με κύκλωμα τον αλγόριθμο υπολογισμού του

ΜΚΔ δύο ακεραίων. Αν x και y είναι οι ακέραιοι, τότε ένας απλός αλγόριθμος σε γλώσσα C, που βασίζεται στον αλγόριθμο του Ευκλείδη και περιλαμβάνει μόνο αφαιρέσεις, είναι ο ακόλουθος:

```
int gcd(int x, int y) {
 while (x!=y) {
   if (x>y) x=x-y;
   else y=y-x;
 }
 return x;
}
```

Για την καλύτερη κατανόηση του αλγορίθμου δίνονται τα ακόλουθα παραδείγματα:
*Παράδειγμα 1: ΜΚΔ των αριθμών 117 και 39*
Βήμα 1: X=117, Y=39, X>Y  X=X-Y=117-39=78
Βήμα 2: X=78, Y=39, X>Y  X=X-Y=78-39=39
Βήμα 3: X=39, Y=39, X=Y  ο ΜΚΔ των 117 και 39 είναι ο 39
*Παράδειγμα 2: ΜΚΔ των αριθμών 95 και 25*
Βήμα 1: X=95, Y=25, X>Y  X=X-Y=95-25=70
Βήμα 2: X=70, Y=25, X>Y  X=X-Y=70-25=45
Βήμα 3: X=45, Y=25, X>Y  X=X-Y=45-25=20
Βήμα 4: X=20, Y=25, X<Y  Y=Y-X=25-20=5
Βήμα 5: X=20, Y=5, X>Y  X=X-Y=20-5=15
Βήμα 6: X=15, Y=5, X>Y  X=X-Y=15-5=10
Βήμα 7: X=10, Y=5, X>Y  X=X-Y=10-5=5
Βήμα 8: X=5, Y=5, X=Y  ο ΜΚΔ των 95 και 25 είναι ο 5
*Παράδειγμα 3: ΜΚΔ των αριθμών 17 και 10*
Βήμα 1: X=17, Y=10, X>Y  X=X-Y=17-10=7
Βήμα 2: X=7, Y=10, X<Y  Y=Y-X=10-7=3
Βήμα 3: X=7, Y=3, X>Y  X=X-Y=7-3=4
Βήμα 4: X=4, Y=3, X>Y  X=X-Y=4-3=1
Βήμα 5: X=1, Y=3, X<Y  Y=Y-X=3-1=2
Βήμα 5: X=1, Y=2, X<Y  Y=Y-X=2-1=1
Βήμα 6: X=1, Y=1, X=Y ο ΜΚΔ των 17 και 10 είναι ο 1 ή δεν υπάρχει ΜΚΔ

Η υλοποίηση ενός τέτοιου κυκλώματος μπορεί να γίνει με πολλούς τρόπους. Για λόγους απλότητας, ας υποθέσουμε ότι η αρχιτεκτονική υλοποίησης είναι δεδομένη και παρουσιάζεται στο ακόλουθο σχήμα. Στη δεξιά πλευρά του σχήματος, εικονίζονται οι δομικές μονάδες που αποτελούν τον τμήμα της αρχιτεκτονικής που είναι υπεύθυνο για την εκτέλεση αριθμητικών πράξεων (μονοπάτι δεδομένων), ενώ στο αριστερό εικονίζεται ένα FSM, που είναι υπεύθυνο για το συγχρονισμό των διαφόρων μονάδων και την κυκλωματική υλοποίηση του βρόχου while (μονοπάτι ελέγχου).




Είναι οι τιμές των συνθηκών μετάβασης και των εξόδων πλήρεις έτσι ώστε να λειτουργεί το κύκλωμα σωστά; Αν όχι, κάντε τις απαιτούμενες προσθήκες. Αν κρίνετε σκόπιμο, μπορείτε να κάνετε αλλαγές και στις δομικές μονάδες που έχουν δοθεί (πολυπλέκτη, καταχωρητή, συγκριτή, αφαιρέτη). Υλοποιήστε το κύκλωμα του ΜΚΔ όπου το FSM και το dataflow να είναι 2 ξεχωριστά modules. Υλοποιήστε ένα testbench και ελέγξτε την ορθή λειτουργία του κυκλώματός σας. Αφού απασφαλματώσετε την υλοποίησή σας και είστε σίγουροι για την ορθότητά της συνθέσετε το κύκλωμά σας.