Tema 2 - Hiding secrets

- Responsabili: Ionut P, Ovidiu M (checker)
- Deadline soft (fără penalizări): ~~**23.12.2022**~~ **26.12.2022**, ora **23:59**
- Deadline hard (cu penalizări): ~~**27.12.2022**~~ **30.12.2022**, ora **23:59**
- Data publicării: **09.12.2022**
- Data ultimei actualizări: **23.12.2022, 13:13**
- Istoric modificări:
  - 09.12.2022
    - publicare enunț temă
  - 11.12.2022
    - `[23:0] array [15:0][15:0]` → `[23:0] array [3:0][3:0]`, pentru a reduce riscul de confuzie
  - 13.12.2022
    - adăugare checker offline
    - modificare punctaj task2 (3.5 → 3.6) și task3 (2.0 → 1.9)
  - 17.12.2022
    - **Versiune nouă checker offline/online - UPDATE your offline checker** - rezolvă problema semnalată Aici [https://curs.upb.ro/2022/mod/forum/discuss.php?d=2825] referitoare la parcurgerea șirului de caractere
  - 23.12.2022
    - Prelungire deadline soft și deadline hard

Obiectiv

Tema are ca scop exersarea noțiunilor Verilog folosite pentru implementarea circuitelor secvențiale complexe, prin:

- implementarea unui automat finit complex, pornind de la un algoritm dat;
- interacțiunea cu o memorie exernă, prin operații de citire/scriere;
- interacțiunea cu alte circuite secvențiale, prin folosirea semnalelor de control.

Descriere și cerințe

Implementați în Verilog un circuit secvențial sincron care ascunde un mesaj secret într-o imagine. Imaginile inițiale sunt reprezentate în spațiul de culoare RGB și au dimensiunea de 64×64 de elemente (pixeli), în care fiecare element are 24 de biți (8 biți 'R', 8 biți 'G' și 8 biți 'B'). Mesajul este de tip text, cu caractere ce aparțin setului extASCII (8 biți necesari pentru fiecare caracter). Pentru a putea *cripta* mesajul în imagine, va fi necesară efectuarea următoarelor etape de prelucrare:

1. Conversia imaginii din RGB în grayscale

Imaginea inițială este codată în spațiul RGB, cu valori pe toate cele trei canale. Imaginea rezultată va fi stocată pe 8 biți în canalul 'G'. Valoarea din canalul 'G' va fi calculată drept media dintre maximul și minimul valorilor din cele trei canale. După această operație, canalele 'R' și 'B' vor fi setate pe valoarea '0'. Fiecare pixel se prelucrează individual.

2. Compresia imaginii folosind metoda AMBTC

Algoritmul AMBTC are la bază împărțirea unei imagini *grayscale* în sub-blocuri de dimensiune M suficient de mici (în cazul nostru 4 x 4) și modificarea fiecăruia pe două nivele, L_m și H_m. Pentru fiecare bloc al imaginii se execută următorii pași:

1. Se calculează media aritmetică AVG

$$AVG = \frac{\sum_{i=1}^{M \cdot M} \mu_i}{M \cdot M}$$

2. Se calculează deviația standard var

$$var = \frac{\sum_{i=1}^{M \cdot M} |\mu_i - AVG|}{M \cdot M}$$

3. Se construiește o hartă (bitmap 4×4) *aux* după următoarele două reguli:
    - dacă valoarea pixelului $\mu_i < AVG$ atunci 0
    - altfel 1
4. Se calculează valorile L_m și H_m folosind următoarele formule:

$$L_m = AVG - \frac{M \cdot M \cdot var}{2(M \cdot M - \beta)}$$

$$H_m = AVG + \frac{M \cdot M \cdot var}{2\beta)},$$

   unde β reprezintă numărul biților de 1 din hartă.
5. După obținerea acestor două valori, blocul se va reconstrui, amplasând pe pozițiile marcate cu 0 valoarea L_m, iar pe pozițiile marcate cu 1 valoarea H_m

Metoda este descrisă pe larg în capitolul 2.1 al articolului din Resurse.

3. Încapsularea mesajului în imaginea prelucrată

Mesajul ce trebuie secretizat este constituit dintr-un șir de caractere ASCII. Fiecare caracter este codat pe 8 biți. Astfel, în fiecare bloc putem încapsula doar 2 caractere. Inițial șirul de biți se va transforma din baza 2 în baza 3. Fiecare valoare S_j va fi integrată în pixelii blocului, exceptând prima valoare L_m și H_m. Procedura va fi executată după următorul algoritm:

```
  pentru fiecare pixel compresat p[j] din blocul b[i]
    dacă p[j] poate primi un bit secret
      dacă bitul secret s[b] este 0, atunci
        nu se execută nicio operație
      dacă bitul secret s[b] este 1, atunci
        p[j] = p[j] + 1;
      dacă bitul secret s[b] este 2, atunci
        p[j] = p[j] - 1;
```

În Anexă este descris procesul complet de prelucrare a unui bloc de 4x4 pixeli, împreună cu detalii relevante pentru implementarea în Verilog.

După fiecare etapă, imaginea va fi suprascrisă cu rezultatul prelucrării pasului respectiv. Cele 3 cerințe de mai sus trebuie să se comporte ca un proces continuu pe aceeași imagine. Astfel, task-ul 2 va începe automat după task-ul 1, iar task-ul 3 va începe automat după task-ul 2. Este necesar ca task-urile să fie executate în ordine.

Implementare

Pentru implementare este necesară construirea unui automat cu stări finite care să modeleze cele 3 cerințe. Acest automat va comunica cu module adiționale, `image` și `base2_to_base3`. Descrierea detaliată a fiecărui modul se regăsește mai jos.

top

Modul deja implementat, el instanțiază modulul responsabil cu prelucrarea imaginii (`process`), cel ce emulează imaginea încărcată în memorie (`image`) și încarcă șirul ce urmează a fi codat. Totodată, acesta este responsabil pentru realizarea conexiunilor între cele două module.

image

Modul deja implementat, responsabil cu încărcarea imaginii din fisier în memoria internă, acesta reprezintă interfața între fsm-ul implementat în process și imagine. Folosind semnalele de control, modulul `process` poate citi sau scrie elemente individuale (pixeli).

Modulul respectă următoarea interfață:

```
module image (
        input                       clk,
        input [image_size - 1:0] row,
        input [image_size - 1:0] col,
        input                       we,
        input [23:0]                in,
        output[23:0]                out
```

Descrierea semnalelor folosite de acest modul este următoarea:

- `clk` - semnal de ceas;
- `row` - selectează un rând din imagine;
- `col` - selectează o coloană din imagine;
- `we` - write enable - activează scrierea în imagine la rândul și coloana date;
- `in` - valoarea pixelului care va fi scris pe poziția dată;
- `out` - valoarea pixelului care va fi citit de pe pozitia dată.

Funcționalitatea modulului este următoarea:

- Pentru a citi un pixel din imaginea care urmează a fi prelucrată (out), trebuie setate semnalele `row` și `col`. Valoarea va fi disponibilă imediat pe semnalul `out`.
- Pentru a scrie un pixel din imaginea prelucrată (in), trebuie setate semnalele `row` și `col`, precum și semnalul `we`. Valoarea prezentă pe semnalul `in` va fi stocată în memorie la următorul ciclu de ceas.

Urmăriți conexiunile din `top` pentru a vedea corespondența cu modulul `process`

process

Automatul cu stări finite care va modela comportamentul trebuie implementat în modulul process. Aici se vor efectua toate transformările necesare și se va instanția base2_to_base3, util în secvența de transformare a caracterelor în ultima etapă.

Modulul trebuie să respecte următoarea interfață:

```
module process (
        input              clk,
        input  [23:0]      in_pix,
        input  [8*512-1:0] hiding_string,
        output [5:0]       row, col,
        output             out_we,
        output [23:0]      out_pix,
        output             gray_done,
        output             compress_done,
        output             encode_done
```

Descrierea semnalelor folosite de acest modul este următoarea:

- `clk` - semnal de ceas;
- `in_pix` - valoarea pixelului de pe poziția [row, col] din imaginea de intrare (R 23:16; G 15:8; B 7:0);
- `hiding_string` - șirul care trebuie codat;
- `row`, `col` - selectează un pixel din imagine din poziția (row, col), atât pentru citire, cât și pentru scriere;
- `out_we` - activează scrierea pentru imaginea de ieșire (write enable);
- `out_pix` - valoarea pixelului care va fi scrisa in imaginea de iesire pe pozitia [row, col] (R 23:16; G 15:8; B 7:0);
- `gray_done` - semnaleaza terminarea actiunii de transformare în grayscale (activ pe 1);
- `compress_done` - semnalează terminarea acțiunii de compresie (activ pe 1);
- `encode_done` - semnalează terminarea acțiunii de codare (activ pe 1).

Modulul process va interacționa cu modulul `image` pentru operațiile de citire și scriere a imaginii și cu modulul `base2_to_base3` pentru transformarea caracterelor într-un șir compatibil operației de codare. Ambele module sunt deja implementate în scheletul temei.

Modificarea declarării ieșirilor în `output reg` este permisă **exclusiv** pentru acest modul.

base2_to_base3

Modul deja implementat, responsabil cu executarea algoritmului de transformare din baza 2 în baza 3.

Modulul are următoarea interfață:

```
module base2_to_base3 (
    output   [31 : 0]  base3_no,
    output             done,
    input    [15 : 0]  base2_no,
    input              en,
    input              clk);
```

Descrierea semnalelor folosite de acest modul este următoarea:

- base3_no - valoarea numărului exprimată în baza 3 - fiecare cifra este codată pe 2 biți; urmărește exemplul din Anexă pentru mai multe detalii.
- done - semnal ce marchează sfârșitul conversiei; acesta trebuie asertat în momentul în care pe portul de ieșire *base3_no* este prezentă valoarea finală
- base2_no - numărul în baza 2 ce trebuie transformat în baza 3; acesta are sens să fie citit doar în momentul în care *en* are valoarea 1;
- en - semnal ce marchează faptul că numărul prezent pe portul *base2_no* este valid și poate fi citit
- clk - semnal de ceas

Pentru a-l putea folosi, intrarea base2_no va fi setată și pe același ciclu de ceas semnalul en va fi asertat. Ulterior, într-o altă stare, se va aștepta semnalul done, care va indica faptul că output-ul este valid și poate fi citit. Pentru o descriere detaliată, revedeți Tema 1.

Pentru transformarea numerelor în baza 3 este necesară folosirea modulului dat.

Observații

- Operația de citire este asincronă, operația de scriere este sincronă; nu se poate executa mai mult de o operație într-un ciclu de ceas.
- Nu este permisă cache-uirea întregii imagini (citirea și salvarea acesteia în cadrul modulului pentru procesare ulterioară). Pentru procesare sunt permise **maxim** 3 blocuri de tip `[23:0] array [3:0][3:0]`, dacă considerați necesar; motivați folosirea lor.
- Semnalele gray_done, compress_done și encode_done trebuie să mențină valoarea HIGH timp de **un ciclu de ceas** pentru a putea fi luate în considerare de tester. În acest ciclu de ceas nu veți face alte procesări și nu veți începe rezolvarea următorului task.

Precizări

- În resursele temei există scheletul ce conține toate modulele necesare;
- Arhiva temei (de tip **zip**) trebuie să cuprindă în rădăcina sa (*fără alte directoare*) **doar**:
  - fișierul sursă **process.v** (prezent în schelet);
  - alte fișiere sursă adiționale (*.v) definite de utilizator;
  - fișierul README.
- Arhiva **nu** trebuie să conțină fișierele sursă implmenentate (*.v) din schelet (ele sunt deja integrate în tester), fișiere de test, fișiere specifice proiectelor etc.
- Fișierului README va conține minim:
  - numele și grupa;
  - prezentarea generală a soluției alese (ex: descrierea de nivel înalt a algoritmului folosit);
  - explicarea porțiunilor complexe ale implementării (poate fi făcută și în comentarii);
  - alte detalii relevante.
- Vmchecker ne permite să revenim la orice soluție încărcată de voi; cereți revenirea la cea mai convenabilă soluție trimisă (punctaj teste automate + depunctare întârziere) printr-un mail titularului de laborator.
- Tema trebuie realizată individual; folosirea de porțiuni de cod de la alți colegi sau de pe Internet (cu excepția site-ului de curs și a resurselor puse la dispoziție în conținutul temei) poate fi considerată **copiere** și va fi penalizată conform regulamentului.

Notare

- +10 pct: implementarea corectă
  - 4.5 pct: implementarea transformării grayscale
  - 3.6 pct: implementarea AMBTC
  - 1.9 pct: implementarea codării mesajului
- +1 pct: fiecare bug găsit în implementarea de referință - cea din tester - (se acordă primei persoane care-l semnalează);
- **-1.5pct: folosirea intregii imagini cache-uite pentru fiecare subpunct. (max -4pct dacă imaginea este cache-uită pentru toate subpunctele);**
- -10 pct: folosirea construcțiilor nesintetizabile;
- -10 pct: folosirea construcțiilor cu număr variabil de iterații (ex: while x != 0);
- -1 pct: lipsa fișierului README;
- -1 pct: indentare haotică (incluzând **spațiere inutilă**);

- -0.5 pct: pentru fiecare zi de întârziere; tema poate fi trimisă cu maxim 7 zile întârziere față de termenul specificat în enunț (față de deadline-ul soft);
- -0.5 pct: folosirea incorectă a atribuirilor continue (assign), blocante (=) și non-blocante (⇐);
- -0.2 pct: diverse alte probleme constatate în implementare (per problemă)
- -0.1 pct: comentarii inutile

Punctajul inițial al checker-ului nu reprezintă punctajul final al temei. Acesta va fi acordat de către asistent, în urma analizei individuale a fiecărei implementări.

Dacă tema primește 0 puncte pe platforma vmchecker, se pot acorda maxim 2 pct pentru ideea de implementare, la latitudinea asistentului. Ideea și motivele pentru care nu funcționează trebuie **documentate** temeinic în README și/sau comentarii. Temele care au **erori** de compilare vor fi notate cu 0 puncte.

Resurse

- **Schelet** - skel
- **Tester** - Tester offline - updated 17.12.2022
- **Articol** - An AMBTC compression based data hiding scheme using pixel value adjusting strategy

- Verilog 2000 Standard [https://sutherland-hdl.com/papers/2001-SNUG-presentation_Verilog-2000_standard.pdf]
- Ghidul studentului la AC [https://ocw.cs.pub.ro/courses/ac-is/studentguide]
- Utilizarea vmchecker [https://ocw.cs.pub.ro/courses/ac-is/tutoriale/6-vmchecker-utilizare]
- Debugging folosind Xilinx ISE [https://ocw.cs.pub.ro/courses/ac-is/tutoriale/3-ise-debug]

Anexă

În anexă va fi prezentat un exemplu numeric pentru un bloc generic B, de dimensiune $4x4$, precum și detalii relevante pentru implementarea în Verilog.

Considerăm imaginea noastră de dimensiune $64x64$ pixeli, împărțită în blocuri de dimensiune $4x4$, pentru a putea efectua algoritmul de încapsulare a mesajului secret.

B82D**52**	439A6A	5DD1FF	E421F3
415F26	7222CF	A13F77	45A9DC
F068ED	4D3BEF	34757A	4A55D4
99B662	AF2561	F59DF4	765FEA

Pentru exemplificare considerăm primul bloc, format din intersecția primelor 4 rânduri cu primele 4 coloane. Valoarea minimă și valoarea maximă sunt marcate cu **bold** în figura de mai jos. Asupra lor se efectuează media aritmetică.

B82D52	439A6A	5DD1FF	E421F3
415F26	7222CF	A13F77	45A9DC
F068ED	4D3BEF	34757A	4A55D4
99B662	AF2561	F59DF4	765FEA

grayscale

007200	006E00	00AE00	008A00
004200	007800	007000	009000
00AC00	009500	005700	008F00
008C00	006A00	00C900	00A400

Ulterior transformării imaginii din RGB în grayscale, se trece la compresie. Dacă pentru operația anterioară împărțirea pe blocuri nu e necesară, pentru compresie este obligatorie considerarea blocurilor individuale. După citirea elementelor (se poate citi un singur pixel / ciclu de ceas) se calculează media AVG, variația var, se construiește matricea cu noile valori L_m și H_m. Bitmap-ul (matricea ce marchează tipul noului pixel) este o etapă intermediară care poate să fie inclusă în implementare, pentru o lizibilitate mai bună a algoritmului, dar nu este obligatorie.

00**72**00	00**6E**00	00**AE**00	00**8A**00
00**42**00	00**78**00	00**70**00	00**90**00
00**AC**00	00**95**00	00**57**00	00**8F**00
00**8C**00	00**6A**00	00**C9**00	00**A4**00

AMBTC

avg = 85

0	0	1	1
0	0	0	1
1	1	0	1
1	0	1	1

L_m = 67
H_m = 9D

RECONSTRUCT

00**67**00	00**67**00	00**9D**00	00**9D**00
00**67**00	00**67**00	00**67**00	00**9D**00
00**9D**00	00**9D**00	00**67**00	00**9D**00
00**9D**00	00**67**00	00**9D**00	00**9D**00

M = 4;

AVG = 0x85;
var = 0x1b;

$\beta = 0x09;$

$L_m = AVG - \frac{M \cdot M \cdot var}{2(M \cdot M - \beta)} = 0x85 - \frac{4 \cdot 4 \cdot 0x1b}{2(4 \cdot 4 - 0x09)} = 0x67;$

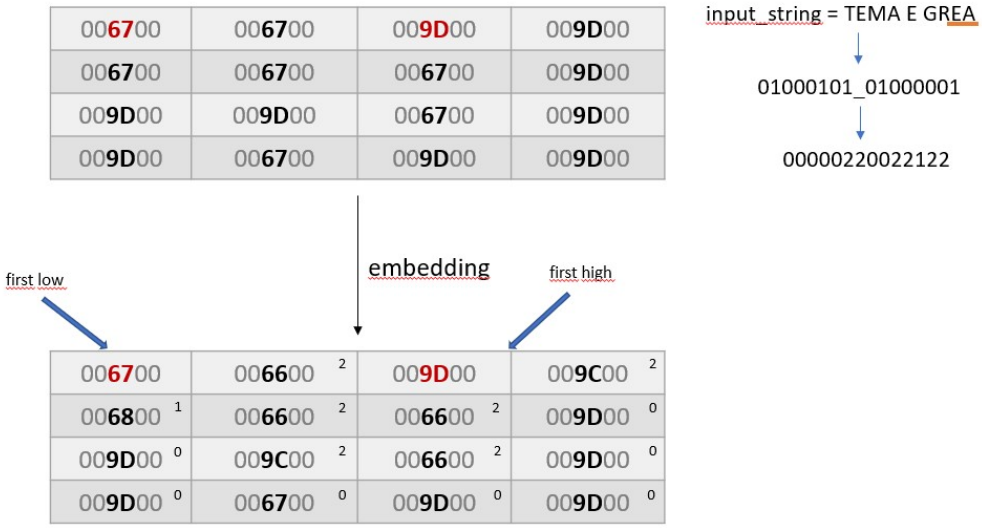
$H_m = AVG + \frac{M \cdot M \cdot var}{2\beta} = 0x85 + \frac{4 \cdot 4 \cdot 0x1b}{2 \cdot 0x09} = 0x9D;$

Atenție la exprimarea numerelor în diversele baze de numerație!

Este recomandat ca stările automatului să aibă funcționalități bine determinate, cu o lungime a execuției potrivită (i.e. deși toată etapa de prelucrare a imaginii se poate face, din punct de vedere al simulării, într-un singur ciclu de ceas, frecvența de operare în practică va fi foarte mică datorită lungimii circuitului combinațional)

Ultima etapă presupune codarea mesajului în interiorul imaginii. Ordinea codării se consideră începând cu primul caracter de la **final** (LSB→MSB).
În fiecare bloc sunt codate 2 caractere (16 biți), întrucât folosirea a 3 caractere conduce la depășirea spațiului alocat. Așadar, în primul bloc vor fi secretizate caracterele cu indicii [15:8] și [7:0], **EA**. Caracterele se vor converti împreună

din baza 2 în baza 3, folosind modulul `base2_to_base3`, și vom folosi primele 14 cifre (fiecare cifră e codată pe 2 biți) pentru a le încapsula în imagine. Ordinea este LSB → MSB, așa cum este prezentat și în imagine. Pixelii marcați cu roșu se omit, deoarece ei vor reprezenta baza pentru decodarea mesajului.



Aceste operații se efectuează pentru toate caracterele: 512 caractere, câte 2 în fiecare bloc, întrucât lungimea șirului nu este definită.

Pentru selecția unei secțiuni dinamice de 8 biți a unei variabile folosită într-o buclă iterativă, construcția aux[i+7:i] nu este permisă. Este necesară folosirea operatorului +: sau -:, astfel: aux[i+7-:8] sau aux[i+:8]. În acest fel sunt selectați biții din intervalul i, i+7. Mai multe detalii în Standardul Verilog 2000, Indexed Vector Part Selects, din resurse