EN224 - Test et vérification

ALBERTY Maxime

9 février 2021

Contents

1 Introduction

Ce TP à pour but de découvrir différentes méthodes de test et vérification dans le développement de composants logiciels et matériels.

2 Software

2.1 Etape 1

L'implémentation de la fonction PGCD fût une simple traduction de l'algorithme présenté en langage C.

```
int PGCD(int A, int B)
{
  while(A != B){
    if (A > B){
      A = A - B;
    } else {
      B = B - A;
    }
  }
  return A;
}
```

Cependant il est important de resté concentré, car même si l'algorithme est trivial, une étourderie est vite arrivé et peut faire perdre du temps inutilement.

Le test de la fonction est effectué au moyen de couples de valeurs d'on aura préalablement calculé le PGCD.

```
printf("PGCD(1024,800) = %d\n",PGCD(1024,800));
printf("PGCD(800,1024) = %d\n",PGCD(800,1024));
printf("PGCD(32767,65535) = %d\n",PGCD(32767,65535));
printf("PGCD(65535,32767) = %d\n",PGCD(65535,32767));
printf("PGCD(512,2048) = %d\n",PGCD(512,2048));
printf("PGCD(2048,512) = %d\n",PGCD(2048,512));
printf("PGCD(458,6272) = %d\n",PGCD(458,6272));
printf("PGCD(6272,458) = %d\n",PGCD(6272,458));
printf("PGCD(783,125) = %d\n",PGCD(783,125));
printf("PGCD(125,783) = %d\n",PGCD(125,783));
```

Il faut ensuite comparé manuellement les résultats aux calculs.

2.2 Etape 2

Afin de pouvoir test plus de valeurs sans avoir à dupliquer les lignes de tests, nous ajoutons une génération aléatoires des valeurs de A et B comprisent entre 0 et 65535.

```
#define MAX_RAND 65535
#define MIN_RAND 0

int RandA(void){
  int A = (rand() % (MAX_RAND + 1 - MIN_RAND)) + MIN_RAND;
    return A;
}

int RandB(void){
  int B = (rand() % (MAX_RAND + 1 - MIN_RAND)) + MIN_RAND;
    return B;
}
```

En ajoutant une boncle FOR dans le main, on peut ainsi test beaucoup plus de valeurs.

```
for(int i = 0 ; i < 200000 ; i++){
  A = RandA();
  B = RandB();
  printf("%d\t%d\t%d\t%d\n", i, A, B,PGCD(A, B));
}
```

Avec l'ajout de cette fonctionnalité, j'ai pu remarqué que la fonction PGCD ne prenait pas en compte les cas où $A = 0$ ou $B = 0$. Conformément à l'annexe du sujet, la fonction PGCD devient alors :

```
int PGCD(int A, int B)
{
  while(A != B){
    if (A==0) return B;
    if (B==0) return A;
    if (A > B){
      A = A - B;
    } else {
      B = B - A;
    }
  }
  return A;
}
```

La vérification des résultats de la fonction PGCD est possible mais beaucoup trop long car il est nécessaire de comparer manuellement les résultats avec les calculs préalablement effectués.

2.3 Etape 3

Afin de tester plus de couple d'entrée, il peut être interressant de comparer ma fonction PGCD avec une autre déjà éprouver. La nouvelle approche est la suivante :

- Assignez à N_1 la valeur de N_2 et à N_2 la valeur du reste de la division de N_1 par N_2;
- Recommencez jusqu'à ce que le reste de la division soit nul;
- A ce moment, N_1 contient alors le PGCD de N_1 et N_2.

Cette approche correspond à cette fonction C :

```
int PGCD2(int A, int B){
  int reste;

  while(B != 0){
    reste = A % B;
    A = B;
    B = reste;
  }
  return A;
}
```

En modifiant la boucle de l'étape 2, il est possible de tester plus de valeur en comparent le résultat des fonctions PGCD1 et PGCD2.

```
for(i = 0 ; i < 65536 ; i++){
  A = RandA();
  B = RandB();
  pgcd_1 = PGCD1(A,B);
  pgcd_2 = PGCD2(A,B);
  test = (pgcd_1==pgcd_2)?true:false;
  printf("%d\t%d\t%d\t%d\t%d\n", i, A, B,PGCD1(A, B), test);
}
```

Cette méthode nous permet de comparer les résultats de deux fonction effectuant la même opération mais avec des approches différentes. Seulement si les deux fonctions ont des résultats identiquement faux pour des couples de valeurs, le test sera positif au lieu d'indiquer une erreur de calcul.

2.4 Etape 4

La vérification des résultats peut commencer en assurant une cohérence entre les valeurs d'entrée et de sortie de la fonction PGCD. Cela est réalisé par des assertions.

```
int PGCD(int A, int B)
{
    //Pre-condition
  assert(A>=0);
  assert(B>=0);
  assert(A<=65535);
  assert(B<=65535);

  while(A != B){
    if (A > B){
      A = A - B;
    } else {
      B = B - A;
    }
  }
  return A;
}
```

Ces pré-condition vont permettre d'assurer que les valeurs de A et B font partie de la plage des valeurs pour laquelle la fonction est conçue.

Ajouter limitation des pré-conditions

Il faut également noté que les assertions ne servent que pour le développement. Lors de la compilation de la version finale du programme, il faut désactiver les assertions.

```
    gcc mon_prog.c -o mon_prog          //Compilation avec les assertions
    gcc mon_prog.c -o mon_prog -NDEBUG  //Compilation sans les assertions
```

2.5 Etape 5

En plus de vérifier si les données fournis à la fonction sont conforme, on va vérifier que le résultat est cohérent.

- La valeur de sortie ne peut pas être plus grande qu'une des valeurs d'entrée,
- Si $A \neq 0$ et $B \neq 0$, la valeur à la sortie de la boucle WHILE ne peut pas être également à 0.

```
int PGCD(int A, int B)
{
  int firstA = A;
    //pre-condition
  assert(A>=0);
  assert(B>=0);
  assert(A<=65535);
  assert(B<=65535);
  while(A != B){
    if(A == 0) return B;
    if(B == 0) return A;
    if (A > B){
      A = A - B;
    } else {
      B = B - A;
    }
  }
    //Post-condition
  assert(A > 0);
  assert(A <= firstA);
  return A;
}
```

Ajouter limitation des post-conditions

2.6 Etape 6

Dans cette partie, nous séparons la fonction PGCD dans un fichier spécifique. Le nouveau fichier PGCD.c contient donc le code suivant :

```c
#include "assert.h"
#include "pgcd.h"

int PGCD(int A, int B)
{
    assert((A>=0) & (B>=0));
    while(A != B){
        if(A == 0) return B;
        if(B == 0) return A;
        if (A > B){
            A = A - B;
        } else {
            B = B - A;
        }
    }
    assert(A > 0);
    return A;
}
```

Afin d'utiliser notre fonction dans le fichier main.c, il faut créer un fichier PGCD.h.

```c
#ifndef _PGCD_H
#define _PGCD_H

int PGCD(int A, int B);

#endif
```

Le fichier main.c est alors comme suit:

```c
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "assert.h"

#include "pgcd.h"

int main (int argc, char * argv []){
    printf("(II) Starting PGCD program\n");

    assert(PGCD(1024,800)==32);

    assert(PGCD(32767,65535)==1);

    assert(PGCD(512,2048)==512);

    assert(PGCD(458,6272)==2);

    printf("(II) End of PGCD program\n");
    return 0;
}
```

Les tests unitaires permettent de vérifié le fonctionnement d'une fonctionnalité pendant sont développement. Ainsi on vérifie si les modifications que l'on apporte nous pas créer de bug. On parlera alors de test de non-régression.

2.7 Etape 7

Il existe de framework permettant de simplifer la rédaction des procédure de test et l'analyse des résultats. Dans cette exercice, le framework Catch2 va permettre d'exprimer des séquences de tests.

```
#include "pgcd.hpp"
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

TEST_CASE ( "Fonctionnement normal", "[PGCD]"){
    SECTION("A > B"){
        REQUIRE( PGCD(1250,570) == 10 );
        REQUIRE( PGCD(5615,1248) == 1 );
        REQUIRE( PGCD(247,570) == 19 );
        REQUIRE( PGCD(14796,570) == 6 );
    }
    SECTION("A < B"){
        REQUIRE( PGCD(6580,9896) == 4 );
        REQUIRE( PGCD(1250,3245) == 5 );
        REQUIRE( PGCD(1250,2000) == 250 );
        REQUIRE( PGCD(1250,1251) == 1 );
    }
        SECTION("A = B"){
        REQUIRE( PGCD(25,25) == 25 );
        REQUIRE( PGCD(1250,1250) == 1250 );
        REQUIRE( PGCD(100,100) == 100 );
        REQUIRE( PGCD(8000,8000) == 8000 );
    }
}

TEST_CASE ( "Fonctionnement autre", "[PGCD]"){
    SECTION("A > B"){
        REQUIRE( PGCD(65535,570) == 15 );
        REQUIRE( PGCD(1248,0) == 1248 );
        REQUIRE( PGCD(570,0) == 570 );
        REQUIRE( PGCD(42,0) == 42 );
    }
    SECTION("A < B"){
        REQUIRE( PGCD(42,65535) == 3 );
        REQUIRE( PGCD(1,65535) == 1 );
        REQUIRE( PGCD(0,480) == 480 );
        REQUIRE( PGCD(0,42) == 42 );
    }
    SECTION("A = B"){
        REQUIRE( PGCD(65535,65535) == 65535 );
        REQUIRE( PGCD(0,0) == 0 );
    }
}
```

Dans le cas où des erreurs sont détéctées, l'affichage est le suivant :

Dans le cas contraire, on retrouvera l'affichage suivant:

Cependant, si une erreur ce produit dans une section, cette dernière est abandonée pour passé à la suivante. Si d'autre erreurs sont présentes sur d'autre cas de la section, elles ne seront pas visible. Nous n'obtenons pas un rapport total des tests.

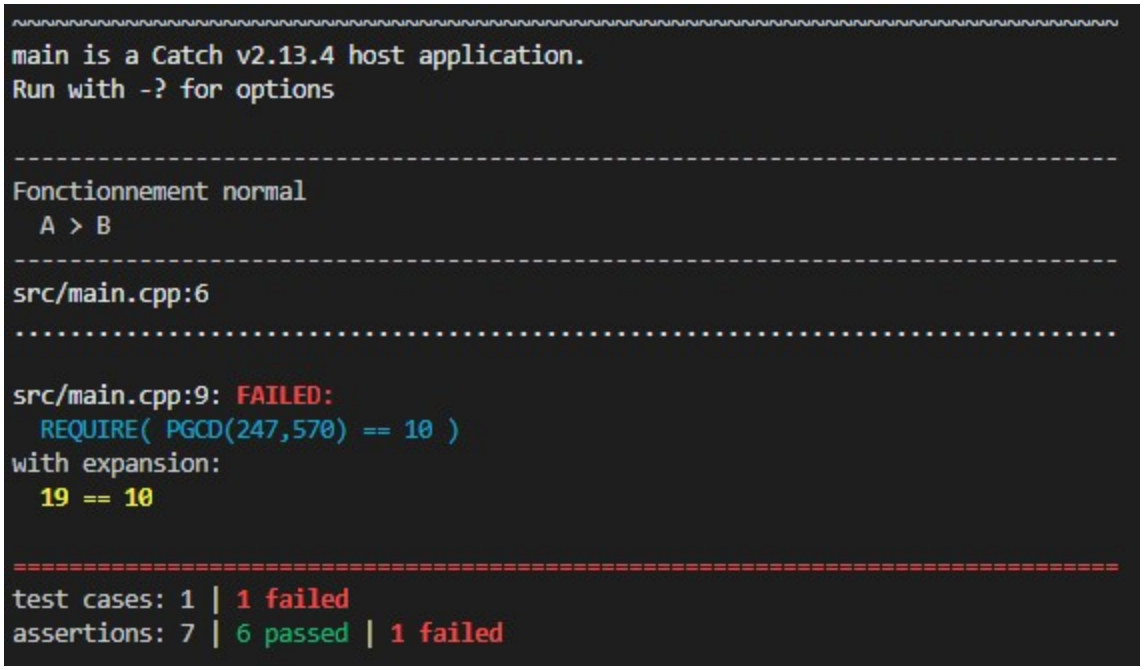

Figure 1: Exemple d'affichage en cas de test échoué

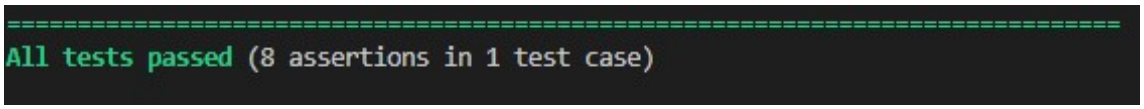

Figure 2: Exemple d'affichage en cas de test réussi

3 Hardware

3.1 Etape 1

De même que pour l'étape 1 de la partie sotfware, il faut décrire un module VHDL permetant d'implanter le calcul du PGCD de deux nombres. Le module aura le prototype suivant :

```
ENTITY PGCD IS
PORT (
    CLK       : in  STD_LOGIC;
    RESET     : in  STD_LOGIC;

    idata_a   : in  STD_LOGIC_VECTOR (31 downto 0);
    idata_b   : in  STD_LOGIC_VECTOR (31 downto 0);
    idata_en  : in  STD_LOGIC;

    odata     : out STD_LOGIC_VECTOR (31 downto 0);
    odata_en  : out STD_LOGIC
);
END PGCD;
```

Le module VHDL fonctionnement grâce à une machine à état.

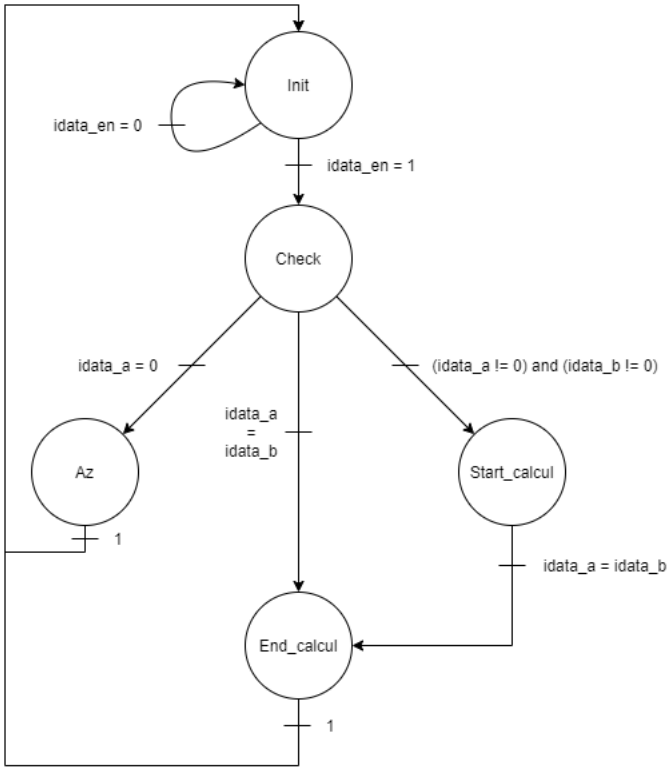

Figure 3: Machine etat module PGCD

La description de cette machine nécéssite 4 process:

- Mise à jour synchrone de l'état
- Calcul de nouvelle état en fonction des entrées
- Calcul des sorties en fonction de l'état en cours
- Cacule du PGCD

3.2 Etape 2

3.3 Etape 3

4 Conclusion

Todo list