SAE 13-IOT ESP32

M LEE

Réalisé par :Irsa AHMAD
.

Etudiante ne BUT1 RT Groupe 2

Table des matières

INTRODUCTION :

Dans ce monde interconnecté d'aujourd'hui, le domaine de l'Internet des Objets (Internet Of Things) est en constante évolution/une révolution technologique. Offrant des possibilités innovantes pour la connectivité, la communication entre les dispositifs électroniques, ce système d'IOT permet de collecter et partager des données par un réseau que ce soit pour commandes des LEDs à distance depuis un smartphone ou encore afin de surveiller les maisons, les entreprises ou les zones publiques à distance. Au cœur de cette révolution se trouve la carte ESP32, un microcontrôleur polyvalent qui joue un rôle essentiel dans le monde de l'électronique.

Notre projet de la SAE13- IOT ESP32, a pour objectif de réaliser un microprocesseur 4_bits composé d'un banc de registre, un ALU4, un registre accumulateur ACC et un multiplexeur 2 vers 1. Ce compte rendu, sera l'occasion de démontrer l'utilisation de la carte ESP32 pour créer un système afin de commander des LEDs à distance depuis un ordinateur connecté via hotspot.

Pour cela, nous allons expliquer notre processus de réalisation des composants fondamentaux dans le domaine informatique, les microprocesseurs à 4-Bit, capables d'effectuer des opérations logiques et arithmétiques sur des données binaires. Pour cela, en premier temps nous allons compléter les codes (la partie programmation) puis réaliserons des instructions ; additions, soustractions, le stockage des nombres, sur notre carte ESP32 depuis l'ordinateur connecté au hotspot avec la carte et nous verrons tout cela grâce à des captures d'écrans de moniteur série, de notre écran et de la carte tout en expliquant le fonctionnement de microprocesseur.

1 INSTALLATION ET PARAMETRAGE

Afin de réaliser le microprocesseur, nous devons tout d'abord installer le logiciel « Arduino IDE » et « ESP32 DEV module ».

1.1 INSTALLATION ET PARAMETRAGE DU LOGICIEL ARDUINO IDE

Nous téléchargeons la dernière version d'Arduino IDE adaptée à mon système d'exploitation Windows, depuis le « Microsoft store ».

Une fois le logiciel lancé nous connectons notre carte Arduino à notre ordinateur à l'aide d'un câble USB.

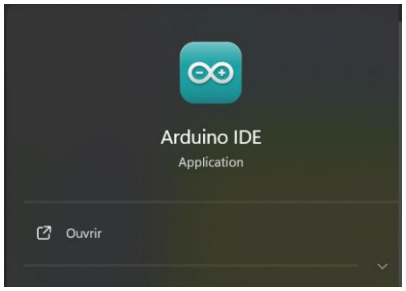


Figure 1 : Le logiciel ARDUINO IDE

1.2 INSTALLATION DE LA CARTE ESP32 et CONNEXION AU HOTSPOT

Nous installons ensuite le type de carte que nous devons utiliser « ESP32 DEV Module ». Voici les étapes d'installation suivis :

- Aller dans Files > Preferences
- Entrer https://dl.espressif.com/dl/package_esp32_index.json dans le "URL de gestionnaire de cartes supplémentaires"

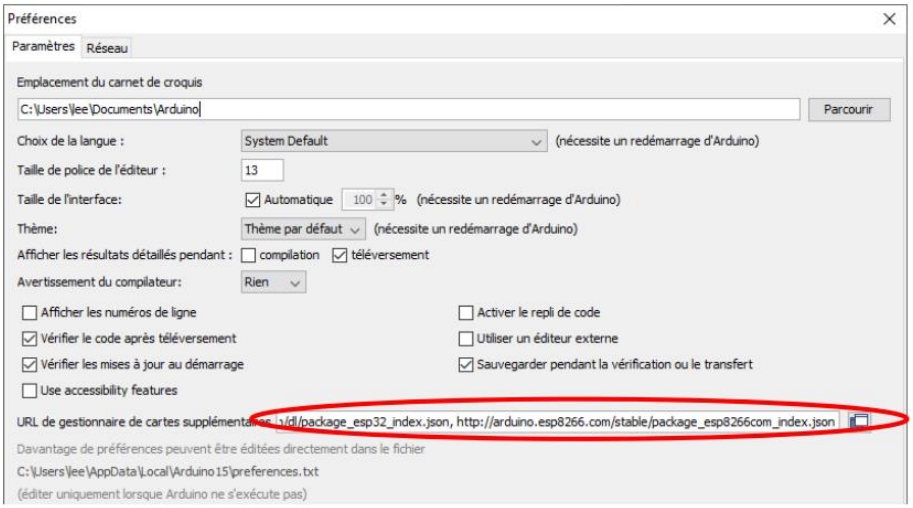


Figure 2 : Le lien dans le logiciel ARDUINO IDE

- Aller dans Outils > Type de cartes > Gestionnaire de cartes
- Taper “ESP32” et installer “ESP32 by Espressif Systems“
- Aller dans Outils > Type de cartes > ESP32 Arduino
- Choisir « ESP32 Dev Module »
- Pour la programmation, aller dans Outils > Programmateur
- Choisir le programmer “AVR ISP” (si le logiciel le propose).

Nous configurons ensuite les paramètres réseau pour la connexion au hotspot.

2 LE FONCTIONNEMENT D’UN MICROPROCESSEUR

Afin de comprendre le fonctionnement d’un microprocesseur, il faut connaitre son architecture. Notre microprocesseur comprend un ensemble de registres une unité arithmétique et logique 4 bits (ALU4), un registres accumulateur (ACC) et un multiplexeur 2 vers 1.

2.1 BANC DE REGISTRE

La composante essentielle d’un microprocesseur réside dans son banc de registres, qui fournit un espace de stockage temporaire pour les données en cours de traitement et a l’adresse (addr) et écriture (WR) comme signaux de contrôle. Ces registres agissent comme des emplacements mémoire temporaires et permettent de transférer des informations au sein du microprocesseur.

Le figure 3 permet de visualiser le programme permettant le fonctionnement du banc de registre.

```
////////////////////////////////////////////  Register bank  //////
struct Output4 Register_bank(bool I[], bool Adr[], bool Clk, bool Wr)
{
    bool Wr_reg[4];
    Output4 DEC24;

    DEC24 = DECOD24(Adr);

    Wr_reg[0] = DEC24.O[0] & Wr;
    Wr_reg[1] = DEC24.O[1] & Wr;
    Wr_reg[2] = DEC24.O[2] & Wr;
    Wr_reg[3] = DEC24.O[3] & Wr;

    Reg0 = Register0(I, Clk, Wr_reg[0]);
    Reg1 = Register1(I, Clk, Wr_reg[1]);
    Reg2 = Register2(I, Clk, Wr_reg[2]);
    Reg3 = Register3(I, Clk, Wr_reg[3]);

    RB = MUX41(Adr, Reg0.O, Reg1.O, Reg2.O, Reg3.O);

    return RB;
}
```

Figure 3 : Le programme de banc de registre

2.2 Unité Arithmétique et Logique - ALU4

L'Unité Arithmétique et Logique 4 bits (ALU4) est le composant opérationnel d'un microprocesseur. Cette unité est responsable de l'exécution des opérations arithmétiques et logiques nécessaires au traitement des données. Cette dernière assure les opérations cruciales sur les bits tel que l'addition (add : op= 0) et la soustraction (*sub : op= 1)* et fournit par ailleurs la capacité de réaliser des calculs complexes.

```
//////////////////////////////////////////// ALU4
struct Output4 ALU_4(bool A[], bool B[], bool op)
{
    bool C, Bo;
    ADD_SUB ADD4, SUB4;
    Output4 ALU4;
    SUB4 = Sub_4(A, B);
    ADD4 = Add_4(A, B);
    ALU4 = MUX21(op, ADD4.S, SUB4.D);
    return ALU4;
}
```

Figure 4 : Le programme de ALU4

Le figure 4 permet de visualiser le programme permettant le fonctionnement d'ALU4.

2.3 L'ACCUMULATEUR – ACC

L'accumulateur se distingue de ALU4 et du banc du registre, étant élément central dans le traitement des données. Ce dernier est un réservoir principal de stockage pour les résultats intermédiaires des opérations effectués par l'ALU4. L'accumulateur agit comme le point central du flux de traitement et maintient les données cruciales tout au long de l'exécution des instructions.

```
//////////////////////// Accumulator DFF0
bool ACC_DFF0 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( ACC.Q[0] & !(!D & Clk));
    ACC.Q[0] = !(Qb & !(D & Clk));
    return ACC.Q[0];
}
```

Figure 5 : Le programme de l'accumulateur DFF0

La coordination entre le banc de registres, l'ALU4 et l'accumulateur est cruciale pour garantir la manipulation appropriée des données et le transfert des résultats vers les étapes ultérieures.

.

2.4 MULTIPLEXEUR 2 VERS 1 – MUX21

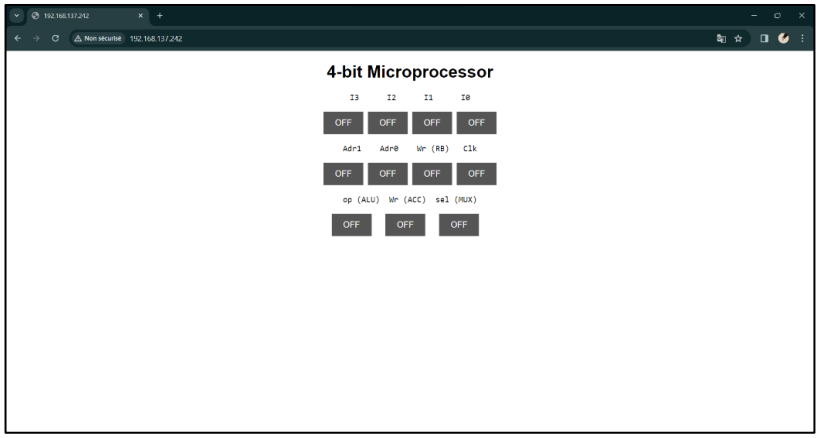
Le multiplexeur 2 vers 1 est installé dans les microprocesseurs afin d'optimiser l'efficacité du microprocesseur et apporte une contribution significative. MUX 21, en fonction des signaux de contrôle, sélectionne entre deux sources d'entrée et dirige les données vers les destinations appropriées ce qui permet d'avoir une architecture fonctionnelle permettant au microprocesseur d'exécuter les instructions avec précision et efficacité.

```
////////////////////////////////////////////// MUX21
struct Output4 MUX21(bool sel, bool i0[], bool i1[])
{
    Output4 MUX_21;
    MUX_21.O[0] = (sel & i1[0]) | (!sel & i0[0]);
    MUX_21.O[1] = (sel & i1[1]) | (!sel & i0[1]);
    MUX_21.O[2] = (sel & i1[2]) | (!sel & i0[2]);
    MUX_21.O[3] = (sel & i1[3]) | (!sel & i0[3]);
    return MUX_21;
}
```

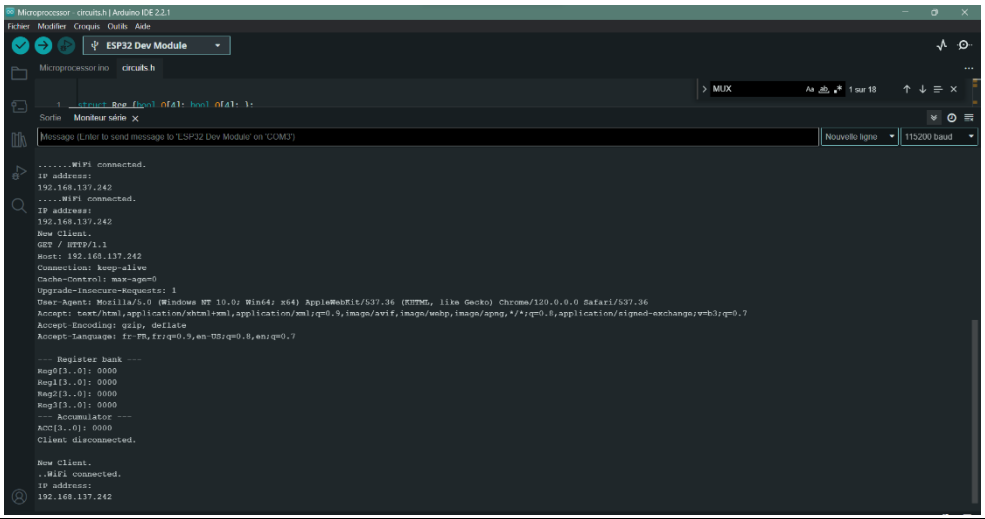
Figure 6 : Le programme du multiplexeur 2 vers 1

2.5 La COMPILATION et le TELEVERSEMENT

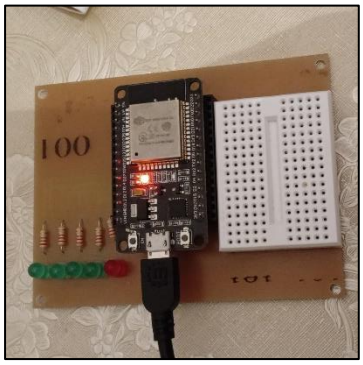
Voici l'écran de l'ordinateur connecté avec la carte, du moniteur série et des LEDs après la compilation et le téléversement du programme :



Voici l'aperçu du moniteur série après la compilation et le téléversement du programme :



Voici l'aperçu des LEDs après la compilation et le téléversement du programme :



3 APPLICATION DANS LE CONTEXTE DE L'IOT

Focalisons maintenant vers l'application pratique de notre microprocesseur 4 bits, conçue avec la carte ESP32. Nous allons commander à distance des LEDs de notre carte depuis notre ordinateur connecté via hotspot, offrant une perspective tangible sur la pertinence et la puissance de notre microprocesseur dans des scénarios réels d'interconnectivité, contribuant à l'évolution constante du domaine de l'Internet des Objets.

3.1 REALISATION DE L'INSTRUCTION 7+3-6 SUR LE MICROPROCESSEUR

Le fonctionnement du microprocesseur est séquentiel, (comme vu dans le module R106), et régi par des signaux de contrôle qui guident son comportement en fonction des instructions données. Il faut donc procéder par étapes afin de réaliser ces instructions.

- Nous initialisons tout d'abord al valeur 7 dans l'accumulateur et *pour cela mettons I0, I1 et I2 à 1 (1+2+4=7) et Wr (ACC) à 1 afin de copier 7 à l'adresse 0 :*

```
--- Register bank ---
Reg0[3..0]: 0000
Reg1[3..0]: 0000
Reg2[3..0]: 0000
Reg3[3..0]: 0000
--- Accumulator ---
ACC[3..0]: 0111
```

Figure 7 : Initialisation à 7

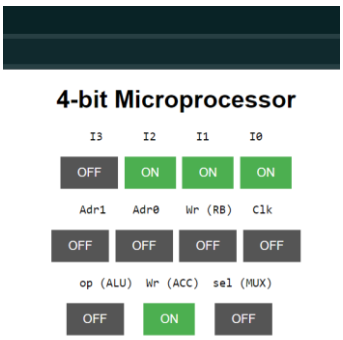
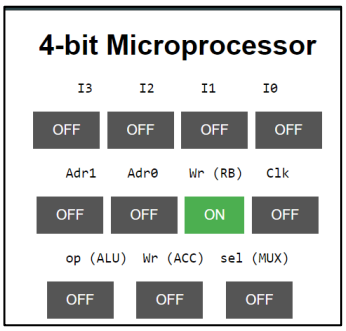
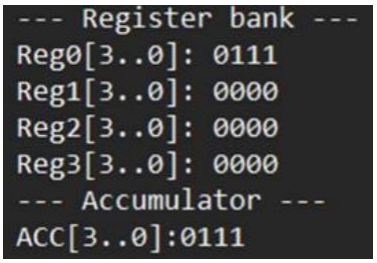


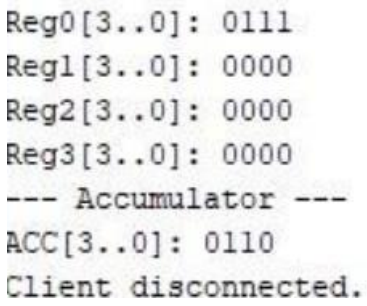
Figure 8 : Ecriture sur ACC et initialisation à 7

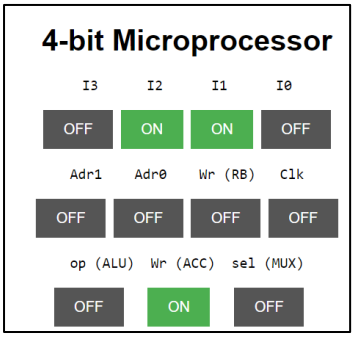
- Nous mettons ensuite WR(ACC) à 0 pour une lecture et Wr (RB) à 1 pour une écriture et les bits d'adresse à 0 afin de stocker la valeur 7 à l'adresse 0 :



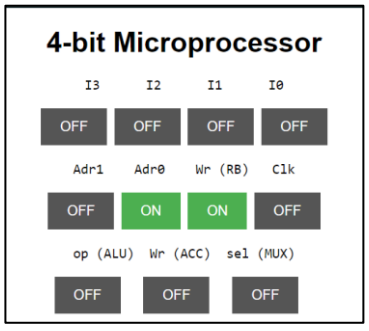


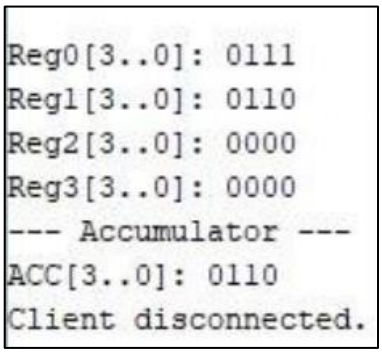
- Nous stockant maintenant la valeur 6 (0110) dans l'accumulateur et *pour cela mettons I1 et I2 à 1 (2+4=6) et Wr (ACC) à 1 afin de copier 6 à l'adresse 1*



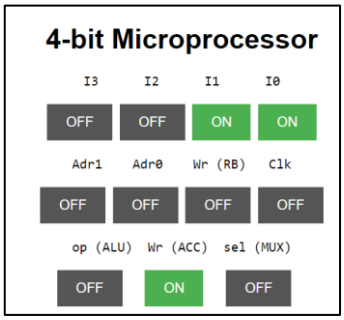


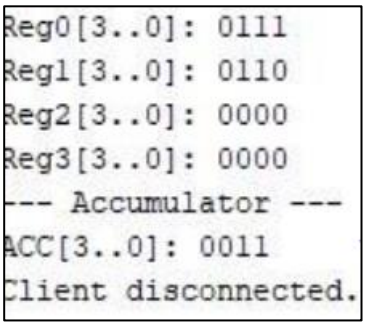
- Vérification en mettant le addr0 et le WR(RB) :



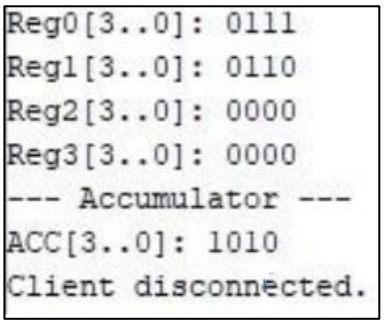


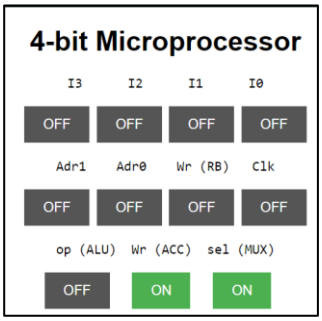
- Initialisons l'accumulateur à 3 (1+2, I0 et I1 = 1) et Wr acc à 1, afin de réaliser l'addition par la suite :



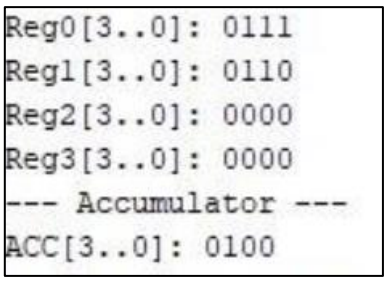


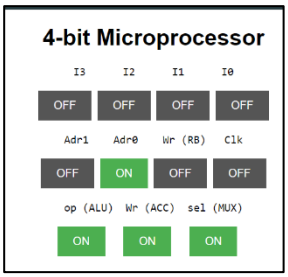
- Réalisons maintenant l'instruction 7+3, et pour cela on met op à 0 pour une addition, sel à 1 pour effectuer une opération et nous obtenons bien 10 (1010) :





- Réalisons maintenant notre opération 7+3-6 pour lequel nous mettons op à 1 pour indiquer une soustraction, sel à 1 pour effectuer une opération. Le résultat obtenu est 4 (0100) comme attendu :





CONCLUSION :

Ce compte rendu nous a permit d'illustrer de manière détaillée le processus de conception et de mise en œuvre d'un microprocesseur 4 bits sur la carte ESP32 en illustrant les différentes l'installation et paramétrages nécessaires puis en mettant en avant différents programmes réalisés ainsi que l'architecture d'un microprocesseur. Ces étapes ont été également illustrés grâce à des captures d'écrans offrant une compréhension approfondie du fonctionnement du microprocesseur. Les implications étendent au-delà des simples réalisations techniques mais cette expérience a enrichi nos compétences pratiques, et nous a permis de voir l'importance des cartes ESP32 et cette technologie dans le monde d'aujourd'hui.

ANNEXE :

MUX 21 :

```
////////////////////  MUX21  ////////////////////////////
struct Output4 MUX21(bool sel, bool i0[], bool i1[])
{
    Output4 MUX_21;
    MUX_21.O[0] = (sel & i1[0]) | (!sel & i0[0]);
    MUX_21.O[1] = (sel & i1[1]) | (!sel & i0[1]);
    MUX_21.O[2] = (sel & i1[2]) | (!sel & i0[2]);
    MUX_21.O[3] = (sel & i1[3]) | (!sel & i0[3]);
    return MUX_21;
}
```

MUX 41 :

```
//////////////////////  MUX41  ////////////////////////////////////////////////////////////
struct Output4 MUX41(bool s[], bool i0[], bool i1[], bool i2[], bool i3[])
{
    Output4 MUX_41;
    MUX_41.O[0] = (i0[0] & !s[1] & !s[0]) | (i1[0] & !s[1] & s[0]) | (i2[0] & s[1] & !s[0]) | (i3[0] & s[1] & s[0]);
    MUX_41.O[1] = (i0[1] & !s[1] & !s[0]) | (i1[1] & !s[1] & s[0]) | (i2[1] & s[1] & !s[0]) | (i3[1] & s[1] & s[0]);
    MUX_41.O[2] = (i0[2] & !s[1] & !s[0]) | (i1[2] & !s[1] & s[0]) | (i2[2] & s[1] & !s[0]) | (i3[2] & s[1] & s[0]);
    MUX_41.O[3] = (i0[3] & !s[1] & !s[0]) | (i1[3] & !s[1] & s[0]) | (i2[3] & s[1] & !s[0]) | (i3[3] & s[1] & s[0]);

    return MUX_41;

}
```

Full adder :

```
//////////////  Full Adder  ////////////////////////
bool Full_Adder(bool A, bool B, bool &Carry)
{
    bool Sum = A ^ B ^ Carry;
    Carry = (A & B) | (Carry & (A | B));
    return Sum;
}
```

Full Substractor :

```
//////////  Full Subtractor  ////////////////////////
bool Full_Subtractor(bool A, bool B, bool &Bo)
{
    bool Diff = A ^ B ^ Bo;
    Bo = !A&Bo | (!A&B) | B&Bo;
    return Diff;
}
```

ADD4 :

```
//////////////////////////  ADD4  ////////////////
struct ADD_SUB Add_4 (bool A[], bool B[])
{
    bool Carry = 0;
    ADD_SUB ADD4;
    ADD4.S[0] = Full_Adder(A[0], B[0], Carry);
    ADD4.S[1] = Full_Adder(A[1], B[1], Carry);
    ADD4.S[2] = Full_Adder(A[2], B[2], Carry);
    ADD4.S[3] = Full_Adder(A[3], B[3], Carry);
    ADD4.C = Carry;
    return ADD4;
}
```

SUB4

```
////////  SUB4  ////////////////////////////////////////
struct ADD_SUB Sub_4 (bool A[], bool B[])
{
    bool Borrow = 0;
    ADD_SUB SUB4;
    SUB4.D[0] = Full_Subtractor(A[0], B[0], Borrow);
    SUB4.D[1] = Full_Subtractor(A[1], B[1], Borrow);
    SUB4.D[2] = Full_Subtractor(A[2], B[2], Borrow);
    SUB4.D[3] = Full_Subtractor(A[3], B[3], Borrow);
    SUB4.Bo = Borrow;
    return SUB4;
}
```

ALU4 :

```
//////////  ALU4  ////////////////////////////////////
struct Output4 ALU_4(bool A[], bool B[], bool op)
{
    bool C, Bo;
    ADD_SUB ADD4, SUB4;
    Output4 ALU4;
    SUB4 = Sub_4(A, B);
    ADD4 = Add_4(A, B);
    ALU4 = MUX21(op, ADD4.S, SUB4.D);
    return ALU4;
}
```

2 :4 Decoder :

```
/////  2:4 Decoder ////////////////////////
struct Output4 DECOD24(bool Adr[])
{
    Output4 dec;
    dec.O[0] = (!Adr[1] & !Adr[0]);
    dec.O[1] = (!Adr[1] &  Adr[0]);
    dec.O[2] = ( Adr[1] & !Adr[0]);
    dec.O[3] = ( Adr[1] &  Adr[0]);
    return dec;
}
```

Register0 DFF :

```
//////// Register0·DFF ////////////////////////////////
bool Reg0_DFF0 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg0.Q[0] & !(!D & Clk));
    Reg0.Q[0] = !(Qb & !(D & Clk));
    return Reg0.Q[0];
}
bool Reg0_DFF1 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg0.Q[1] & !(!D & Clk));
    Reg0.Q[1] = !(Qb & !(D & Clk));
    return Reg0.Q[1];
}
```

```
bool Reg0_DFF2 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg0.Q[2] & !(!D & Clk));
    Reg0.Q[2] = !(Qb & !(D & Clk));
    return Reg0.Q[2];
}
bool Reg0_DFF3 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg0.Q[3] & !(!D & Clk));
    Reg0.Q[3] = !(Qb & !(D & Clk));
    return Reg0.Q[3];
}
```

Register 1 DFF :

```
//////// Register1 DFF ////////////////////////////////
bool Reg1_DFF0 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg1.Q[0] & !(!D & Clk));
    Reg1.Q[0] = !(Qb & !(D & Clk));
    return Reg1.Q[0];
}
bool Reg1_DFF1 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg1.Q[1] & !(!D & Clk));
    Reg1.Q[1] = !(Qb & !(D & Clk));
    return Reg1.Q[1];
}
```

```
bool Reg1_DFF2 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg1.Q[2] & !(!D & Clk));
    Reg1.Q[2] = !(Qb & !(D & Clk));
    return Reg1.Q[2];
}
bool Reg1_DFF3 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg1.Q[3] & !(!D & Clk));
    Reg1.Q[3] = !(Qb & !(D & Clk));
    return Reg1.Q[3];
}
```

Register 2 DFF :

```
bool Reg2_DFF0 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg2.Q[0] & !(!D & Clk));
    Reg2.Q[0] = !(Qb & !(D & Clk));
    return Reg2.Q[0];
}
bool Reg2_DFF1 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg2.Q[1] & !(!D & Clk));
    Reg2.Q[1] = !(Qb & !(D & Clk));
    return Reg2.Q[1];
}
```

```
bool Reg2_DFF2 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg2.Q[2] & !(!D & Clk));
    Reg2.Q[2] = !(Qb & !(D & Clk));
    return Reg2.Q[2];
}
bool Reg2_DFF3 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg2.Q[3] & !(!D & Clk));
    Reg2.Q[3] = !(Qb & !(D & Clk));
    return Reg2.Q[3];
}
```

Register 3 DFF :

```
bool Reg3_DFF0 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg3.Q[0] & !(!D & Clk));
    Reg3.Q[0] = !(Qb & !(D & Clk));
    return Reg3.Q[0];
}
bool Reg3_DFF1 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg3.Q[1] & !(!D & Clk));
    Reg3.Q[1] = !(Qb & !(D & Clk));
    return Reg3.Q[1];
}
```

```
bool Reg3_DFF2 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg3.Q[2] & !(!D & Clk));
    Reg3.Q[2] = !(Qb & !(D & Clk));
    return Reg3.Q[2];
}
bool Reg3_DFF3 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( Reg3.Q[3] & !(!D & Clk));
    Reg3.Q[3] = !(Qb & !(D & Clk));
    return Reg3.Q[3];
}
```

Register 0 et 1 :

```
//////////////////////////////////////////// Register0 ////////////////////////////////////////////////////////////
struct Reg Register0 (bool I[], bool Clk, bool Wr)
{
    Reg0.O[0] = Reg0_DFF0(I[0], Clk, Wr);
    Reg0.O[1] = Reg0_DFF1(I[1], Clk, Wr);
    Reg0.O[2] = Reg0_DFF2(I[2], Clk, Wr);
    Reg0.O[3] = Reg0_DFF3(I[3], Clk, Wr);
    return Reg0;
}
//////////////////////////////////////////// Register1 ////////////////////////////////////////////////////////////
struct Reg Register1 (bool I[], bool Clk, bool Wr)
{
    Reg1.O[0] = Reg1_DFF0(I[0], Clk, Wr);
    Reg1.O[1] = Reg1_DFF1(I[1], Clk, Wr);
    Reg1.O[2] = Reg1_DFF2(I[2], Clk, Wr);
    Reg1.O[3] = Reg1_DFF3(I[3], Clk, Wr);
    return Reg1;
}
```

Register 2 et 3 :

```
//////////////////////////////////////////// Register2 ////////////////////////////////////////////////////////////
struct Reg Register2 (bool I[], bool Clk, bool Wr)
{
    Reg2.O[0] = Reg2_DFF0(I[0], Clk, Wr);
    Reg2.O[1] = Reg2_DFF1(I[1], Clk, Wr);
    Reg2.O[2] = Reg2_DFF2(I[2], Clk, Wr);
    Reg2.O[3] = Reg2_DFF3(I[3], Clk, Wr);
    return Reg2;
}
//////////////////////////////////////////// Register3 ////////////////////////////////////////////////////////////
struct Reg Register3 (bool I[], bool Clk, bool Wr)
{
    Reg3.O[0] = Reg3_DFF0(I[0], Clk, Wr);
    Reg3.O[1] = Reg3_DFF1(I[1], Clk, Wr);
    Reg3.O[2] = Reg3_DFF2(I[2], Clk, Wr);
    Reg3.O[3] = Reg3_DFF3(I[3], Clk, Wr);
    return Reg3;
}
//////////////////////////////////////////// Register bank ////////////////////////////////////////////////////////
```

Register bank :

```
//////////////////////////////////////////// Register bank ////////////
struct Output4 Register_bank(bool I[], bool Adr[], bool Clk, bool Wr)
{
    bool Wr_reg[4];
    Output4 DEC24;

    DEC24 = DECOD24(Adr);

    Wr_reg[0] = DEC24.O[0] & Wr;
    Wr_reg[1] = DEC24.O[1] & Wr;
    Wr_reg[2] = DEC24.O[2] & Wr;
    Wr_reg[3] = DEC24.O[3] & Wr;

    Reg0 = Register0(I, Clk, Wr_reg[0]);
    Reg1 = Register1(I, Clk, Wr_reg[1]);
    Reg2 = Register2(I, Clk, Wr_reg[2]);
    Reg3 = Register3(I, Clk, Wr_reg[3]);

    RB = MUX41(Adr, Reg0.O, Reg1.O, Reg2.O, Reg3.O);

    return RB;
}
```

Accumulator DFF 0 et 1/

```
//////////////////////// Accumulator DFF0 ////////////////////////////
bool ACC_DFF0 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( ACC.Q[0] & !(!D & Clk));
    ACC.Q[0] = !(Qb & !(D & Clk));
    return ACC.Q[0];
}

//////////////////////// //////////////////////// Accumulator DFF1 /////////
bool ACC_DFF1 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( ACC.Q[1] & !(!D & Clk));
    ACC.Q[1] = !(Qb & !(D & Clk));
    return ACC.Q[1];
}
```

Accumulator DFF 2 et 3 :

```
//////////////////////  Accumulator DFF2  ////////////////////////////////////////////////////////////////////////////////////////////
bool ACC_DFF2 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( ACC.Q[2] & !(!D & Clk));
    ACC.Q[2] = !(Qb & !(D & Clk));
    return ACC.Q[2];
}

//////////////////////  Accumulator DFF3  ////////////////////////////////////////////////////////////////////////////////////////////
bool ACC_DFF3 (bool D, bool Clk, bool Wr)
{
    bool Qb;
    Clk = Clk ^ (Clk & !Wr);
    Qb = !( ACC.Q[3] & !(!D & Clk));
    ACC.Q[3] = !(Qb & !(D & Clk));
    return ACC.Q[3];
}
```

Accumulateur :

```
//////////////////////  Accumulator ////////////////////////////////////////////////////////////////////////////////////////////
struct Reg Accumulator (bool I[], bool Clk, bool Wr)
{
    ACC.O[0] = ACC_DFF0(I[0], Clk, Wr);
    ACC.O[1] = ACC_DFF1(I[1], Clk, Wr);
    ACC.O[2] = ACC_DFF2(I[2], Clk, Wr);
    ACC.O[3] = ACC_DFF3(I[3], Clk, Wr);
    return ACC;
}
```