|  |  |  |
| --- | --- | --- |
|  |  | Projet Microcontroleur  Le Simon |

21 oct. 16

# Presentation

Nous avons réalisé avec le microcontrôleur STM32 un jeu « le Simon » en utilisant le carte de contrôle en utilisant les boutons poussoirs, l’affichage LED, le moteur ainsi que le biper.



**Projet réalisé par**

Berthier Jérémy

Caudron Vincent

Table des matières

[Presentation 1](#_Toc464807682)

[Origine du jeu 3](#_Toc464807683)

[But du jeu 3](#_Toc464807684)

[Matériel à disposition 3](#_Toc464807685)

[Description du code 4](#_Toc464807686)

[Présentation des fonctions utilisés : 4](#_Toc464807687)

[Présentation des variables : 4](#_Toc464807688)

[Présentation du main : 5](#_Toc464807689)

[Code complet 6](#_Toc464807690)

[Code principal 6](#_Toc464807691)

[Code d’interruption 14](#_Toc464807692)

# Origine du jeu

Le jeu du Simon trouve son origine dans le jeu pour enfant Jacques a dit, d'où il tire également son nom, puisque dans les pays anglophones ce n'est pas Jacques mais Simon (Simon Says…) qui donne les ordres.

Le jeu, électronique, éclaire une des quatre couleurs et produit un son toujours associé à cette couleur. Le joueur doit alors appuyer sur la touche de la couleur qui vient de s'allumer dans un délai assez court.

Le jeu répète la même couleur et le même son, puis ajoute au hasard une nouvelle couleur. Le joueur doit reproduire cette nouvelle séquence. Chaque fois que le joueur reproduit correctement la séquence, le jeu ajoute une nouvelle couleur.

# But du jeu

Le but du jeu étant de reproduire la plus longue des suites grâce au bouton poussoir.

# Matériel à disposition

Pour réaliser le jeu, nous avions à disposition :

* Une carte microcontrôleur STM32
* Une carte de contrôle avec :
  + 4 boutons poussoirs
  + 8 LEDs
  + Un afficher LED
  + Un biper
  + Un moteur

# Description du code

## Présentation des fonctions utilisés :

|  |
| --- |
| void enable\_interrupt\_ext(void); // Initialisation des interruptions  void init\_TIM3(void); // Initialisation du TIM3  void init\_PWM(void); // Initialisation du PWM  void init\_MOTEUR\_EXT(void); // Initialisation du Moteur  void init\_led(void); // Initialisation des LED  void init\_BP(void); // Initialisation des boutons poussoirs  void wait(uint32\_t tmp); // Fonction Wait  void affiche(uint32\_t num); // Fonction qui affiche (principale)  void affiche\_sequence(uint32\_t num); // Sous fonction qui affiche la séquence  void introduction(void); // Fonction affichage d’introduction  void suivant(void); // Fonction affiche suivant (entre chaque action)  void erreur(void); // Fonction affiche erreur  void init\_BIP(void); // Fonction Initialisation du Biper  void init\_BIP\_TIM3(void); // Fonction du Biper avec le TIM3 (Differente fréquence)    void init\_SPI(void); // Initialisation de liaison avec afficheur LED  void SPI\_Write(uint8\_t add,uint8\_t data); // Fonction écriture LED  void SendData\_SPI(uint8\_t add); // Fonction d’envoi LED  void Set\_CS (void); // Sous Fonction de Send DATA  void Reset\_CS (void); // Sous Fonction de Send DAT  void Afficheur(void); // Fonction qui affiche le niveau dans le jeu |

## Présentation des variables :

|  |
| --- |
| uint32\_t compt; // Variable qui compte le niveau dans le jeu  uint32\_t rep; // Variable qui enregistre le touche sélectionné  char sequence[40]; // Tableau de la séquence  uint32\_t wait\_tmp; // Variable temps d’attente  uint8\_t randNumber; // Variable de nombre aléatoire |

## Présentation du main :

|  |
| --- |
| int main()  {  // Initialisation des fonctions  init\_led();  init\_BP();  enable\_interrupt\_ext();  init\_TIM3();  init\_PWM();  init\_MOTEUR\_EXT();  init\_BIP();  init\_BIP\_TIM3();    init\_SPI();    //Commandes d'initialisation pour MAX  SPI\_Write(0x0C,0x00); //Eteindre MAX  SPI\_Write(0x0C,0x01); //Mode Normal  SPI\_Write(0x0A,0x0A); //Luminosité  SPI\_Write(0x09,0x0F); //Decode Mode car 4 digits  SPI\_Write(0x0B,0x03); // Scan Limit  SPI\_Write(0x0F,0x00); //Display test    //Initialisation des digits à 0    SPI\_Write(0x01,0x00);  SPI\_Write(0x02,0x00);  SPI\_Write(0x03,0x00);  SPI\_Write(0x04,0x00);    //Variable valeur par défaut  wait\_tmp = 1000000;  compt = 0;  rep = 0;    // Debut du jeu  introduction();  affiche\_sequence(1);    while(1)  {  **// vide (entièrement en interruption)**  } |

# Code complet

Le code est découpé en deux parties :

* Le code principal avec le main et les fonctions
* Le code des fonctions d’interruption

## Code principal

|  |
| --- |
| /\* programme PARTIEL CAUDRON Vincent \*/  #include "stm32l1xx\_nucleo.h"  void enable\_interrupt\_ext(void);  void init\_TIM3(void);  void init\_PWM(void);  void init\_MOTEUR\_EXT(void);  void init\_led(void);  void init\_BP(void);  void wait(uint32\_t tmp);  void affiche(uint32\_t num);  void affiche\_sequence(uint32\_t num);  void introduction(void);  void suivant(void);  void erreur(void);  void init\_BIP(void);  void init\_BIP\_TIM3(void);    void init\_SPI(void);  void SPI\_Write(uint8\_t add,uint8\_t data);  void SendData\_SPI(uint8\_t add);  void Set\_CS (void);  void Reset\_CS (void);  void Afficheur(void);  uint32\_t compt;  uint32\_t rep;  char sequence[40];  uint32\_t wait\_tmp;  uint8\_t randNumber;  int main()  {  init\_led();  init\_BP();  enable\_interrupt\_ext();  init\_TIM3();  init\_PWM();  init\_MOTEUR\_EXT();  init\_BIP();  init\_BIP\_TIM3();    init\_SPI();    //Commandes d'initialisation pour MAX  SPI\_Write(0x0C,0x00); //Eteindre MAX  SPI\_Write(0x0C,0x01); //Mode Normal  SPI\_Write(0x0A,0x0A); //LuminositÈ  SPI\_Write(0x09,0x0F); //Decode Mode car 4 digits  SPI\_Write(0x0B,0x03); // Scan Limit  SPI\_Write(0x0F,0x00); //Display test    //Initialisation des digits ‡ 0  SPI\_Write(0x01,0x00);  SPI\_Write(0x02,0x00);  SPI\_Write(0x03,0x00);  SPI\_Write(0x04,0x00);      wait\_tmp = 1000000;  compt = 0;  rep = 0;    introduction();  affiche\_sequence(1);    while(1)  {  //Boucle  /\*randNumber = (uint8\_t) rand();  randNumber = (randNumber % 4) + 1 ;  if(randNumber == 1)  {  TIM3->PSC = 200;  TIM3->CCER |= (1<<4);  wait(wait\_tmp);  TIM3->CCER &= ~(1<<4);  }  if(randNumber == 2)  {  TIM3->PSC = 400;  TIM3->CCER |= (1<<4);  wait(wait\_tmp);  TIM3->CCER &= ~(1<<4);  }  if(randNumber == 3)  {  TIM3->PSC = 800;  TIM3->CCER |= (1<<4);  wait(wait\_tmp);  TIM3->CCER &= ~(1<<4);  }  if(randNumber == 4)  {  TIM3->PSC = 1000;  TIM3->CCER |= (1<<4);  wait(wait\_tmp);  TIM3->CCER &= ~(1<<4);  }\*/  }  }  void enable\_interrupt\_ext(void)  {  NVIC->ISER[1] |= (1<<8); //vecteur d'interuption  NVIC->ISER[0] |= (1<<23);    RCC->APB2ENR |= (1<<0); //Reset clock    EXTI->IMR |= (1<<11); // activer interuption  EXTI->IMR |= (1<<12);  EXTI->IMR |= (1<<6);  EXTI->IMR |= (1<<5);    EXTI->FTSR |= (1<<11); // Fond descendant  EXTI->FTSR |= (1<<12);  EXTI->FTSR |= (1<<6);  EXTI->FTSR |= (1<<5);    SYSCFG->EXTICR[1] &= ~((1<<8)|(1<<10)|(1<<11)); // PC6  SYSCFG->EXTICR[1] |= (1<<9);    SYSCFG->EXTICR[1] &= ~((1<<4)|(1<<6)|(1<<7)); // PC5  SYSCFG->EXTICR[1] |= (1<<5);    SYSCFG->EXTICR[2] &= ~((1<<12)|(1<<13)|(1<<14)|(1<<15));// PA11    SYSCFG->EXTICR[3] &= ~((1<<0)|(1<<1)|(1<<2)|(1<<3));// PA12  }  void init\_BP(void)  {  /\* Activation GPIOA \*/  RCC->AHBENR |= (1<<0);  /\* Activation GPIOC \*/  RCC->AHBENR |= (1<<2);    /\* Activation MODER BP1 & 2 \*/  GPIOA->MODER &= ~((1<<22)|(1<<23)|(1<<24)|(1<<25));    /\* Activation MODER BP2 & 3 \*/  GPIOC->MODER &= ~((1<<10)|(1<<11)|(1<<12)|(1<<13));    /\* Activation PUPDR BP1 & 2 \*/  GPIOA->PUPDR &= ~((1<<22)|(1<<23)|(1<<24)|(1<<25));  /\* Activation PUPDR BP3 & 4 \*/  GPIOC->PUPDR &= ~((1<<10)|(1<<11)|(1<<12)|(1<<13));  }  void init\_led(void)  {  /\* Activation GPIOB \*/  RCC->AHBENR |= (1<<1);    /\* Activation MODER LED \*/  GPIOB->MODER |= ((1<<2)|(1<<4)|(1<<20)|(1<<22)|(1<<24)|(1<<26)|(1<<28)|(1<<30));  GPIOB->MODER &= ~((1<<3)|(1<<5)|(1<<21)|(1<<23)|(1<<25)|(1<<27)|(1<<29)|(1<<31));  GPIOB->OTYPER &= ~((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  }  void wait(uint32\_t tmp)  {  uint32\_t i = 0;  for(i = 0; i<tmp; i++);  }    void affiche(uint32\_t num)  {  if(num == 1)  {  TIM3->PSC = 200;  TIM3->CCER |= (1<<4);  GPIOB->ODR |= ((1<<11)|(1<<12));  wait(wait\_tmp);  GPIOB->ODR &= ~((1<<11)|(1<<12));  TIM3->CCER &= ~(1<<4);  wait(wait\_tmp);  }    if(num == 2)  {  TIM3->PSC = 400;  TIM3->CCER |= (1<<4);  GPIOB->ODR |= ((1<<11)|(1<<12)|(1<<10)|(1<<13));  wait(wait\_tmp);  GPIOB->ODR &= ~((1<<11)|(1<<12)|(1<<10)|(1<<13));  TIM3->CCER &= ~(1<<4);  wait(wait\_tmp);  }    if(num == 3)  {  TIM3->PSC = 600;  TIM3->CCER |= (1<<4);  GPIOB->ODR |= ((1<<11)|(1<<12)|(1<<10)|(1<<13)|(1<<2)|(1<<14));  wait(wait\_tmp);  GPIOB->ODR &= ~((1<<11)|(1<<12)|(1<<10)|(1<<13)|(1<<2)|(1<<14));  TIM3->CCER &= ~(1<<4);  wait(wait\_tmp);  }    if(num == 4)  {  TIM3->PSC = 800;  TIM3->CCER |= (1<<4);  GPIOB->ODR |= ((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(wait\_tmp);  GPIOB->ODR &= ~((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  TIM3->CCER &= ~(1<<4);  wait(wait\_tmp);  }  }    void affiche\_sequence(uint32\_t num)  {  Afficheur();    uint32\_t i;  for(i=0;i<100;i++)  {  randNumber = (uint8\_t) rand();  }  randNumber = (randNumber % 4) + 1 ;  sequence[compt] = randNumber;  for(i=0;i<num;i++)  {  affiche(sequence[i]);  }  }  void introduction(void)  {  uint32\_t tmp = 1000000;    GPIOB->ODR &= ~((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(tmp);  GPIOB->ODR |= ((1<<1)|(1<<15));  wait(tmp);  GPIOB->ODR |= ((1<<2)|(1<<14));  wait(tmp);  GPIOB->ODR |= ((1<<10)|(1<<13));  wait(tmp);  GPIOB->ODR |= ((1<<11)|(1<<12));  wait(tmp);  GPIOB->ODR &= ~((1<<11)|(1<<12));  wait(tmp);  GPIOB->ODR &= ~((1<<10)|(1<<13));  wait(tmp);  GPIOB->ODR &= ~((1<<2)|(1<<14));  wait(tmp);  GPIOB->ODR &= ~((1<<1)|(1<<15));  wait(tmp);  }  void suivant(void)  {  uint32\_t tmp = 100000;    GPIOB->ODR &= ~((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(tmp);  GPIOB->ODR |= ((1<<1));  wait(tmp);  GPIOB->ODR &= ~((1<<1));  GPIOB->ODR |= ((1<<2));  wait(tmp);  GPIOB->ODR &= ~((1<<2));  GPIOB->ODR |= ((1<<10));  wait(tmp);  GPIOB->ODR &= ~((1<<10));  GPIOB->ODR |= ((1<<11));  wait(tmp);  GPIOB->ODR &= ~((1<<11));  GPIOB->ODR |= ((1<<12));  wait(tmp);  GPIOB->ODR &= ~((1<<12));  GPIOB->ODR |= ((1<<13));  wait(tmp);  GPIOB->ODR &= ~((1<<13));  GPIOB->ODR |= ((1<<14));  wait(tmp);  GPIOB->ODR &= ~((1<<14));  GPIOB->ODR |= ((1<<15));  wait(tmp);  GPIOB->ODR &= ~((1<<15));  }  void erreur(void)  {  uint32\_t tmp = 100000;    TIM3->PSC = 1599;  TIM3->CCER |= (1<<0);  GPIOB->ODR |= ((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(tmp);  GPIOB->ODR &= ~((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(tmp);  GPIOB->ODR |= ((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(tmp);  GPIOB->ODR &= ~((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(tmp);  GPIOB->ODR |= ((1<<1)|(1<<2)|(1<<10)|(1<<11)|(1<<12)|(1<<13)|(1<<14)|(1<<15));  wait(tmp);  TIM3->CCER &= ~(1<<0);  suivant();  affiche\_sequence(1);  }  void init\_MOTEUR\_EXT(void)  {  RCC->AHBENR |= (1<<1);    GPIOB->MODER |= (1<<9);  GPIOB->MODER &= ~(1<<8);    GPIOB->OTYPER &= ~(1<<4);    GPIOB->AFR[0] |= (1<<17);  }  void init\_BIP(void)  {  RCC->AHBENR |= (1<<2); // PC    GPIOC->MODER |= (1<<15); //PC7  GPIOC->MODER &= ~(1<<14); //PC7    GPIOC->OTYPER &=~(1<<7); // pushpull    GPIOC->AFR[0] |= (1<<29); // TIM3  }  void init\_PWM(void)  {  TIM3->CCMR1 &= ~((1<<0)|(1<<1));  TIM3->CCMR1 &= ~(1<<4);  TIM3->CCMR1 |= ((1<<5)|(1<<6));    TIM3->CCR1 = 20;    //TIM3->CCER |= (1<<0);  }  void init\_BIP\_TIM3(void)  {  TIM3->CCMR1 &= ~((1<<8)|(1<<9));  TIM3->CCMR1 &= ~(1<<12);  TIM3->CCMR1 |= ((1<<13)|(1<<14));    TIM3->CCR2 = 38;    //TIM3->CCER |= (1<<4);    }  void init\_TIM3(void)  {  RCC->APB1ENR |= (1<<1);    TIM3->PSC = 1599; /\* PSC+1-1599+1-1600 soit F=10kHz donc T=100us \*/  TIM3->ARR = 39; /\* 40\*100us = 4ms donc ARR = 40-1 = 39 \*/    //TIM3->PSC = 15;  //TIM3->ARR = 49;    TIM3->CR1 |= (1<<7);  TIM3->CR1 &= ~(1<<6);  TIM3->CR1 &= ~(1<<5);  TIM3->CR1 &= ~(1<<4);    TIM3->CR1 |= (1<<0);  }  void init\_SPI(void)  {  RCC->AHBENR |= (1<<0); //On passe le bit un a 1 pour AHB pour les ports A  RCC->APB2ENR |= (1<<12); //Horloge sur le MAX    //ParamÈtrages des trois pattes du MAX (PA5 PA8 et PA7)  GPIOA->MODER &= ~(1<<10); //PA5 en Alternate fonction  GPIOA->MODER |= (1<<11);  GPIOA->MODER &= ~(1<<14); //PA7 en Alternate fonction  GPIOA->MODER |= (1<<15);  GPIOA->MODER |= (1<<16); //PA8 en sortie  GPIOA->MODER &= ~(1<<17);  GPIOA->OTYPER &= ~(1<<5); //PA5  GPIOA->OTYPER &= ~(1<<7); //PA7  GPIOA->OTYPER &= ~(1<<8); //PA8    //Multiplexage des pattes sur le AF5 du multiplexeur  GPIOA->AFR[0] |= (1<<20); // Registre de selection de la fonction multiplexÈe  GPIOA->AFR[0] |= (1<<22);  GPIOA->AFR[0] |= (1<<28);  GPIOA->AFR[0] |= (1<<30);    SPI1->CR1 &= ~(1<<3); //BR  SPI1->CR1 |= (1<<4);  SPI1->CR1 &= ~(1<<5);    SPI1->CR1 &= ~(1<<0); //CPOL sur FMontant  SPI1->CR1 &= ~(1<<1); //CPHA sur Fmontant    SPI1->CR1 |= (1<<2); //Bit MSTR    SPI1->CR1 &= ~(1<<11); //DFF    SPI1->CR1 &= ~(1<<7); //LSBFIRST    SPI1->CR1 |= (1<<14); // Mode only transmit  SPI1->CR1 |= (1<<15); //    SPI1->CR2 |= (1<<2); //SSOE    SPI1->CR1 |= (1<<6); //Activation    }    void SPI\_Write( uint8\_t add,uint8\_t data)  {  Set\_CS();  Reset\_CS();  SendData\_SPI(add);  SendData\_SPI(data);  while ((SPI1->SR & SPI\_SR\_BSY) == SPI\_SR\_BSY);  Set\_CS();  }  void SendData\_SPI (uint8\_t add)  {  while ((SPI1->SR & SPI\_SR\_TXE)!= SPI\_SR\_TXE);  SPI1->DR = add;  }  void Set\_CS (void)  {  GPIOA->ODR |= (1<<8);  }  void Reset\_CS (void)  {  GPIOA->ODR &= ~(1<<8);  }  void Afficheur(void)  {  uint32\_t compt\_unit;    compt\_unit = compt;    if(compt == 0)  {  //Initialisation des digits ‡ 0  SPI\_Write(0x03,0x00);  SPI\_Write(0x04,0x00);  }  if(compt >= 10)  {  compt\_unit = compt - 10;  SPI\_Write(0x03,0x01);  }  if(compt >= 20)  {  compt\_unit = compt - 20;  SPI\_Write(0x03,0x02);  }  if(compt >= 30)  {  compt\_unit = compt - 30;  SPI\_Write(0x03,0x03);  }  if(compt >= 40)  {  compt\_unit = compt - 40;  SPI\_Write(0x03,0x02);  }  SPI\_Write(0x04,compt\_unit);  } |

## Code d’interruption

|  |
| --- |
| …  /\* Includes ------------------------------------------------------------------\*/  #include "stm32l1xx\_it.h"  extern uint32\_t compt;  extern uint32\_t rep;  extern char sequence[12];    /\*\* @addtogroup Template\_Project  \* @{  \*/  /\* Private typedef -----------------------------------------------------------\*/  /\* Private define ------------------------------------------------------------\*/  /\* Private macro -------------------------------------------------------------\*/  /\* Private variables ---------------------------------------------------------\*/  /\* Private function prototypes -----------------------------------------------\*/  /\* Private functions ---------------------------------------------------------\*/  void EXTI15\_10\_IRQHandler()  {  if(EXTI->PR & (1<<11))  {  affiche(1);  if(sequence[rep] == 1)  {  if(rep == compt)  {  suivant();  compt ++;  rep = 0;  affiche\_sequence(compt+1);  }  else rep ++;  }  else  {  erreur();  compt = 0;  rep = 0;  }  EXTI->PR |= (1<<11);  }    if(EXTI->PR & (1<<12))  {  affiche(2);  if(sequence[rep] == 2)  {  if(rep == compt)  {  suivant();  compt ++;  rep = 0;  affiche\_sequence(compt+1);  }  else rep ++;  }  else  {  erreur();  compt = 0;  rep = 0;  }  EXTI->PR |= (1<<12);  }  }    void EXTI9\_5\_IRQHandler()  {  if(EXTI->PR & (1<<6))  {  affiche(3);  if(sequence[rep] == 3)  {  if(rep == compt)  {  suivant();  compt ++;  rep = 0;  affiche\_sequence(compt+1);  }  else rep ++;  }  else  {  erreur();  compt = 0;  rep = 0;  }  EXTI->PR |= (1<<6);  }  if(EXTI->PR & (1<<5))  {  affiche(4);  if(sequence[rep] == 4)  {  if(rep == compt)  {  suivant();  compt ++;  rep = 0;  affiche\_sequence(compt+1);  }  else rep ++;  }  else  {  erreur();  compt = 0;  rep = 0;  }  EXTI->PR |= (1<<5);  }  }      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Cortex-M3 Processor Exceptions Handlers \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  … |