The Freescale Cup
Intelligent Car Racing

Freescale Cup

Project carried out by Poisson Valentin, Venuti Léo at ESIEE PARIS.

Technical

We used the following software:

- Tera Term to observe the values of the shades of gray seen by the camera
- Matlab to make a digital filter (instead of analog)
- 123D Design to make objects in 3D
- Cura to convert STL files to XCODE

- CodeWarrior to program the Cortex in C language
- Processing to display (in the form of a curve) the gradient result in Java language

As regards the components used:

- Cortex M3 which is the microprocessor we used to run the Evalbot.
- Embedded camera to "see" the path
- Servo motors and rear motors (power motors)

I. Technical realization

1. Car construction

We saw in the introduction that we had for idea first to put a body on our car. Finally, after a more precise idea of the project, it seemed more judicious to create a platform by 3D printing thanks to the Witbox machine.

The idea of this platform germinated since we had to connect the motors on the Cortex M3 card and the camera. So we took all the necessary measures (there are two "columns" on the car that we relied on to put the card).
Thanks to the 123D Design software, we have taken care of a three-dimensional printing software. We have therefore deferred on our subject the measures taken previously, to have a rendering such as:

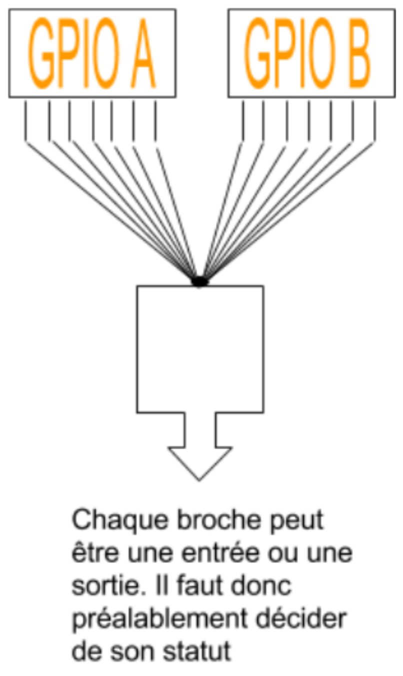
Then, after doing this, we launched the Cura software to convert the .STL to .XCODE, which allows to be read by the Witbox printers, present in room 5209.

After printing the platform, we realized that the platform was filled with holes, that is, the print density was not high enough. So we put a fairly high density so that our object is not easily broken and can be strong enough to support the weight of the Cortex M3 card.

After that, we started to program the camera on the car. But we had to hold it high, a height high enough to see clearly the way. It was quite complicated to hold it and we decided to print a new 3D part to keep it at the desired height.

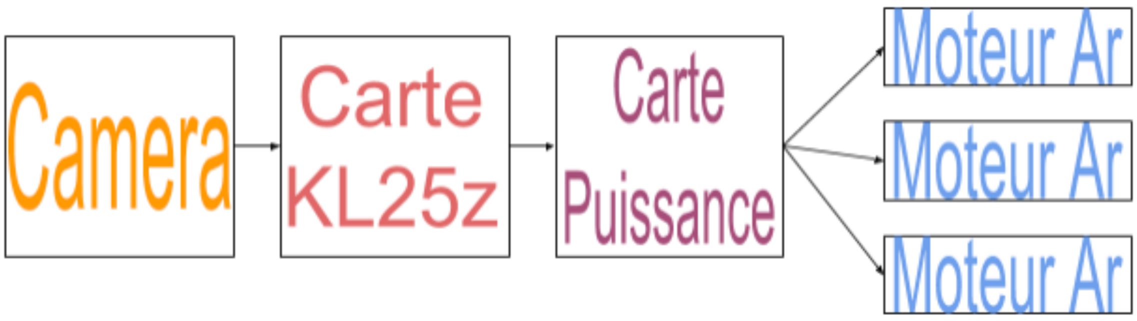
We tried several prototypes but none allowed to manually adjust the height of this camera. And that's what we wanted. But after several searches, we realized that finally we had to calculate the height we wanted and so make sure that the camera sees far enough to perform the anticipation calculations but still close to not miss the l Current location of the car.

2. Base de GPIOS

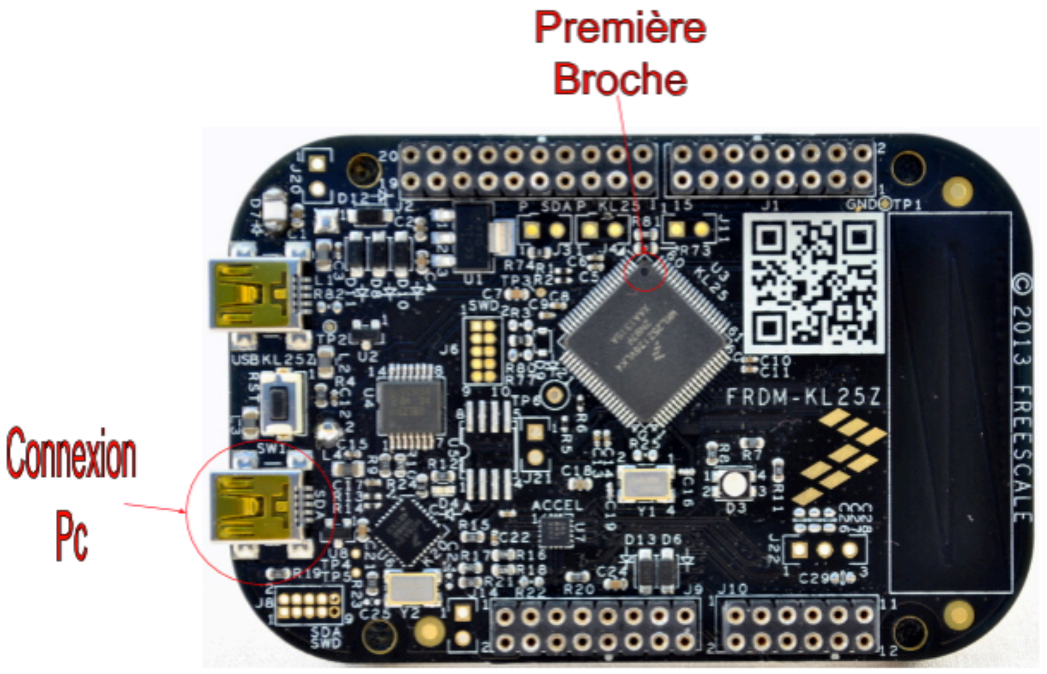

GPIO A
GPIO B
Chaque broche peut être une entrée ou une sortie. Il faut donc préalablement décider de son statut

- We are working on the project called Freescale Cup (ex Motorola).
- This name should be changed shortly in NxP Cup (which is the former Philipps microelectonics) because of buy-back of the company.
  The map that we will be given to program is a processor of the Cortex M3 family, specially dedicated to Freescale.
  The specific name of this processor is FRDM-KL25z.
- The programming environment we are going to use is called CodeWarrior.

When we get to the heart of the matter, a diagram helps us to understand the bases of the links between the different elements that will constitute the final car. This diagram can be found below.

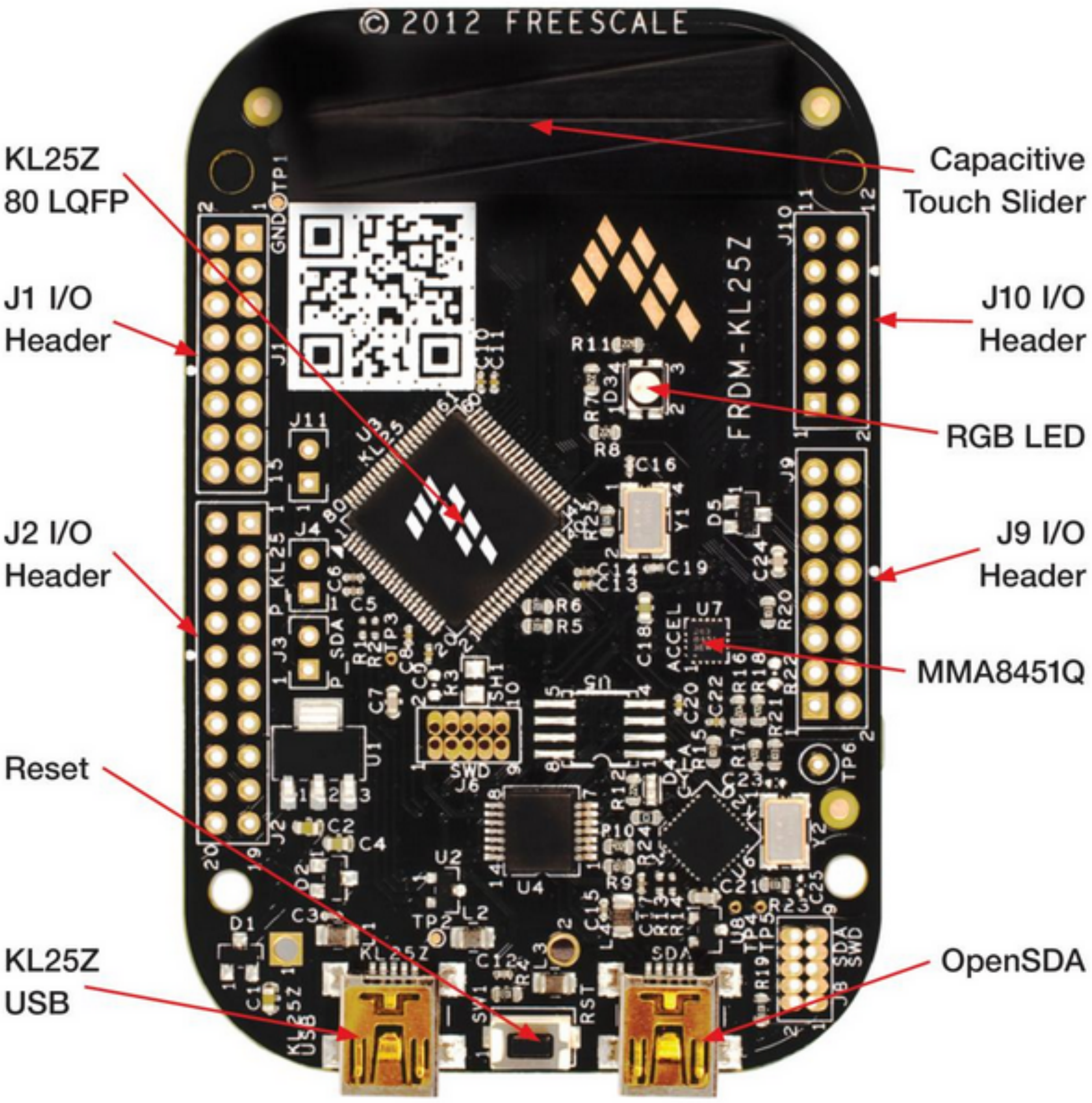


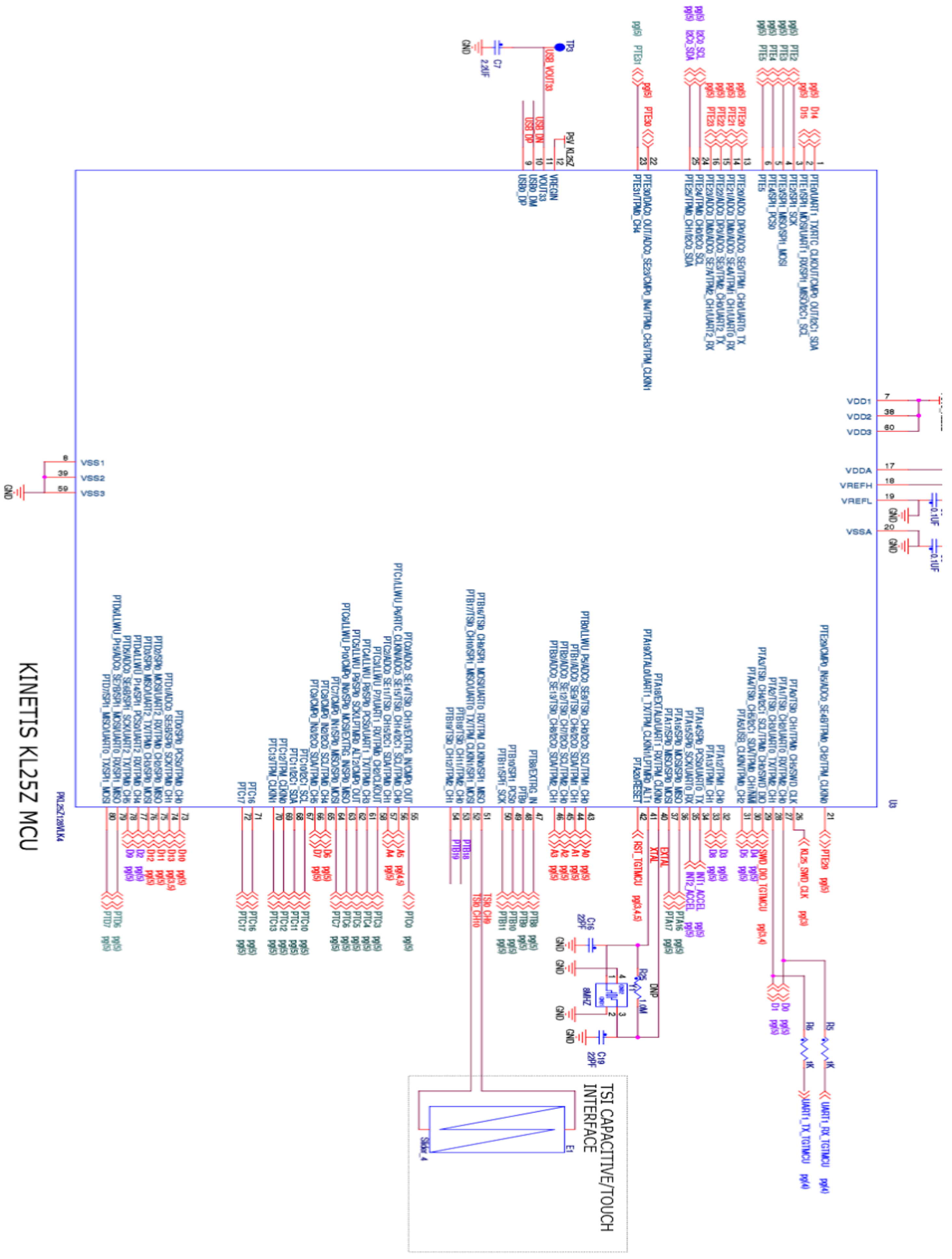
Namely, the term "first spindle" actually corresponds to the first spindle. The other pins are defined around the KL25z 80LQFP and are ranked in ascending order clockwise to have a mark on which to base.
In addition, here is a fairly complete map of the map we need to use for our project. There are all important components:



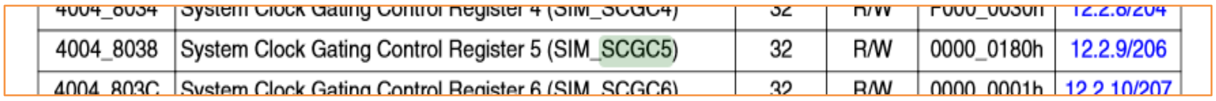
3. Mémoires et schéma de la carte

Two points that we have studied during the first course is the difference of notion between the RAM memory and the FLASH memory:



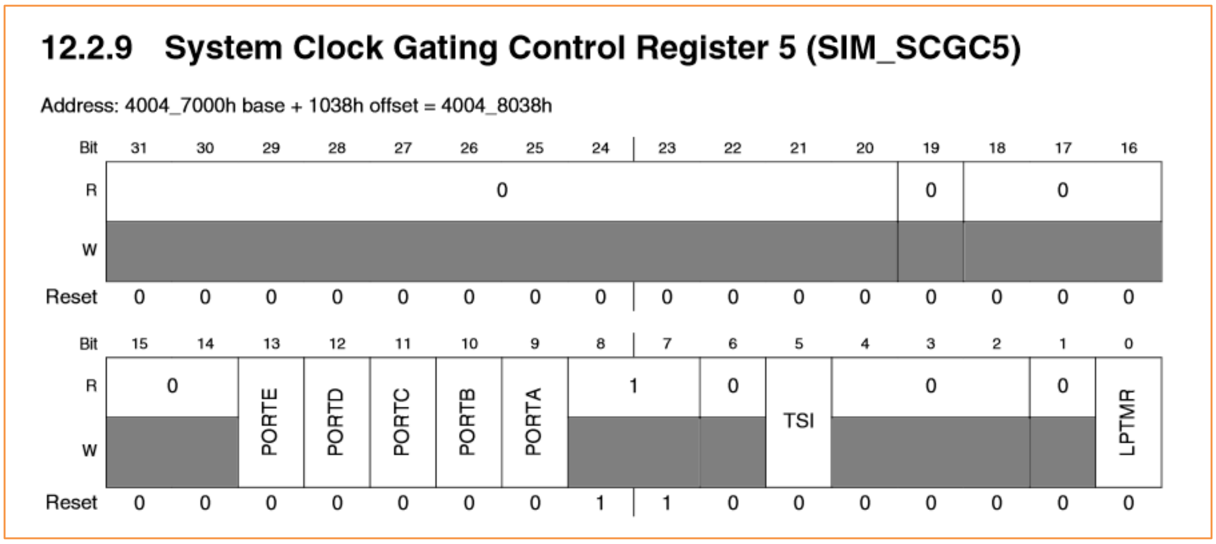

KINETIS KL25Z MCU
PKL25Z128VLK4
U3
TSI CAPACITIVE/TOUCH INTERFACE
VDD1
VDD2
VDD3
VDDA
VREFH
VREFL
VSSA
VSS1
VSS2
VSS3
VREGIN
VOUT33
USB0_DM
USB0_DP
GND
8MHZ
1.0M
22PF
2.2UF
0.1UF

The base address is 4004-7000h for the clock (according to KL25 Sub-Family Reference Manual on page 5). An offset of 1038h must be added, so we arrive at address 4004-8038. This is the SIM_SCGC5, which we have all the necessary information below:



4004_8034	System Clock Gating Control Register 4 (SIM_SCGC4)	32	R/W	F000_0030h	12.2.8/204
4004_8038	System Clock Gating Control Register 5 (SIM_SCGC5)	32	R/W	0000_0180h	12.2.9/206
4004_803C	System Clock Gating Control Register 6 (SIM_SCGC6)	32	R/W	0000_0001h	12.2.10/207

To activate PORTs B and D we have to use SIM_SCGC5 because the address is 4004-8038.
So let's look at the data related to SIM_SCGC5, still in the same document:



12.2.9 System Clock Gating Control Register 5 (SIM_SCGC5)

Address: 4004_7000h base + 1038h offset = 4004_8038h

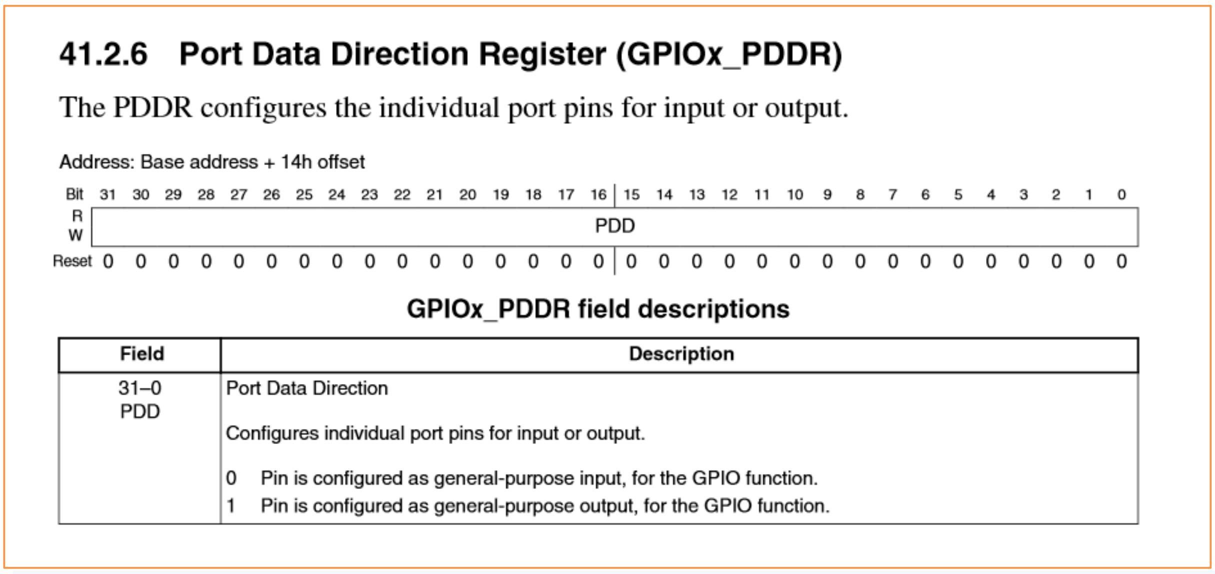
We see that PORTs B and D are situated on bits 11 and 13 respectively (counting 0). So we have to put a 1 in place 11 as well as 13 to make these ports work.
The code in corresponding C:

```
GPIOB_PDDR = (1<<18) | (1<<19);
SIM_SCGC5 = SIM_SCGC5 | (1<<11) //Pour le PORT B
SIM_SCGC5 = SIM_SCGC5 | (1<<13) //Pour le PORT D

⇔

SIM_SCGC5 |= (1<<11) //Pour le PORT B
SIM_SCGC5 |= (1<<12) //Pour le PORT D
```

1. Activer la broche en entrée ou en sortie

The blue LED therefore corresponds to pin 74 and therefore to PTD1. Either at the 1st bit.
The red LED therefore corresponds to pin 53 and therefore to PTB18. Either at the 18th bit.
The green LED therefore corresponds to pin 54 and therefore to PTB19. Either at the 19th bit.
The corresponding bits are given by counting the zero.

In order to define whether a pin is an input or an output, we must use the Port Data Direction Register, whose informatons are given below:



41.2.6 Port Data Direction Register (GPIOx_PDDR)

The PDDR configures the individual port pins for input or output.

Address: Base address + 14h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PDD																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

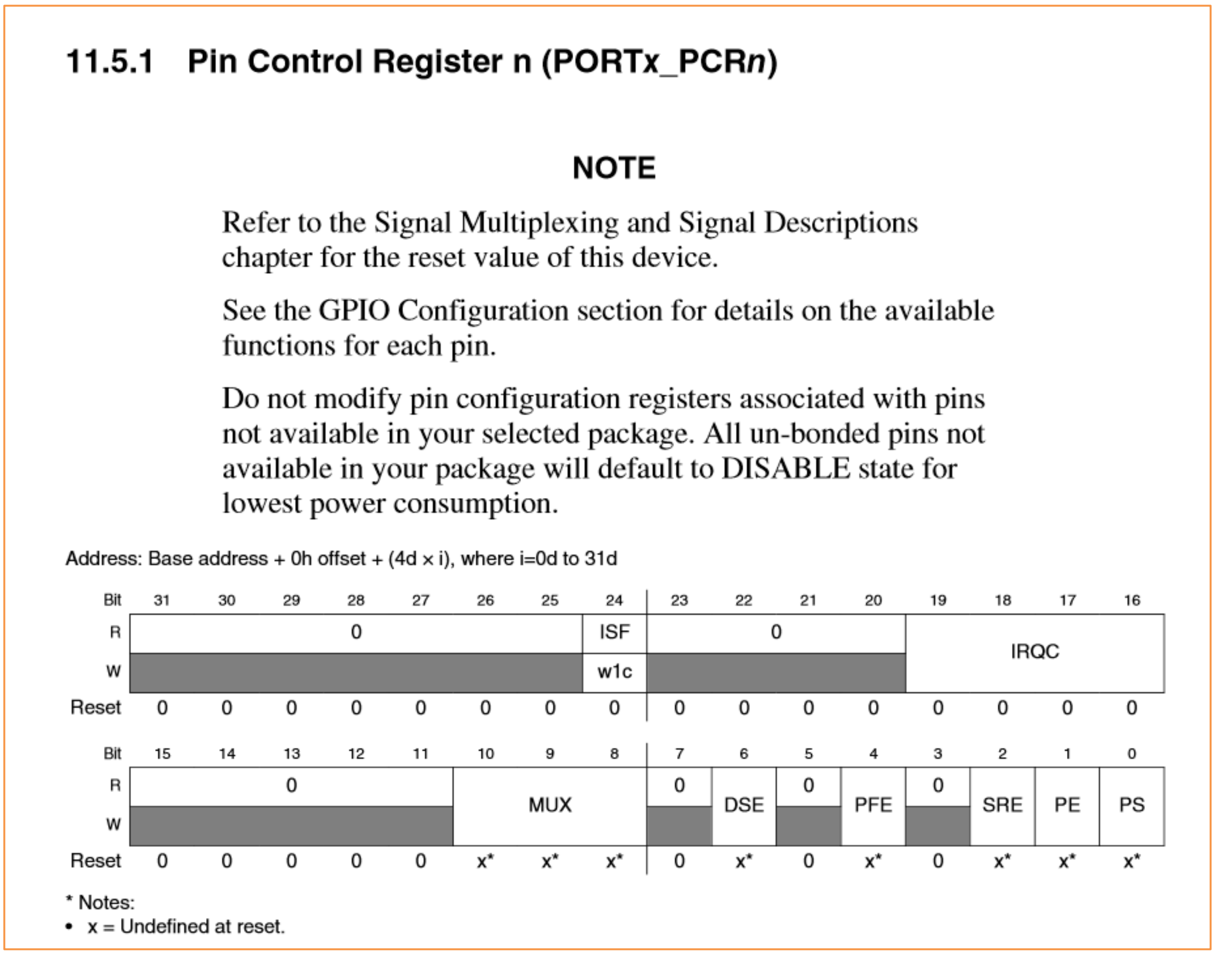
GPIOx_PDDR field descriptions

Field	Description
31–0 PDD	Port Data Direction Configures individual port pins for input or output. 0 Pin is configured as general-purpose input, for the GPIO function. 1 Pin is configured as general-purpose output, for the GPIO function.

```
GPIOB_PDDR = (1<<18) | (1<<19);
//Définir un « 1 » à la place 18 du PORT B (LED rouge en sortie)
//Définir un « 1 » à la place 19 du PORT B (LED verte en sortie)
GPIOD_PDDR = (1<<1);
//Définir un « 1 » à la place 1 du PORT D (LED bleue en sortie)

⇔

//Nous pouvons également utilizer des defines afin de nous simplifier
l'écriture, en
//effet les define permettent de définir une écriture pour une valeur
donnée
```

```
#define LED_R 0x20000 //0x20000 ⌠ (1<<18)
        #define LED_V 0x40000 //0x40000 ⌠ (1<<19)
#define LED_B 0x1 //0x1 ⌠ (1<<1)

GPIOB_PDDR = (LED_R) | (LED_V);
GPIOD_PDDR = (LED_B);
```

2. Configurer la DSE et le MUX



11.5.1 Pin Control Register n (PORTx_PCR*n*)

NOTE

Refer to the Signal Multiplexing and Signal Descriptions chapter for the reset value of this device.

See the GPIO Configuration section for details on the available functions for each pin.

Do not modify pin configuration registers associated with pins not available in your selected package. All un-bonded pins not available in your package will default to DISABLE state for lowest power consumption.

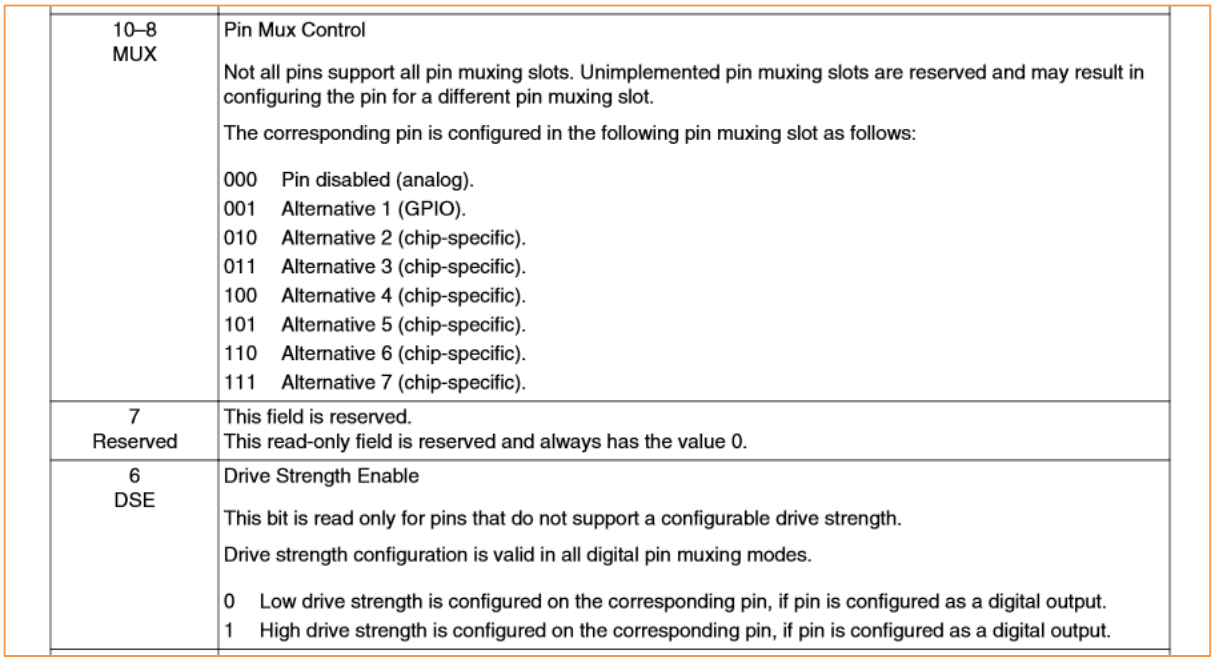
Address: Base address + 0h offset + (4d x i), where i=0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W	▨	▨	▨	▨	▨	▨	▨	w1c	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W	▨	▨	▨	▨	▨				▨		▨		▨			
Reset	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

* Notes:
- x = Undefined at reset.

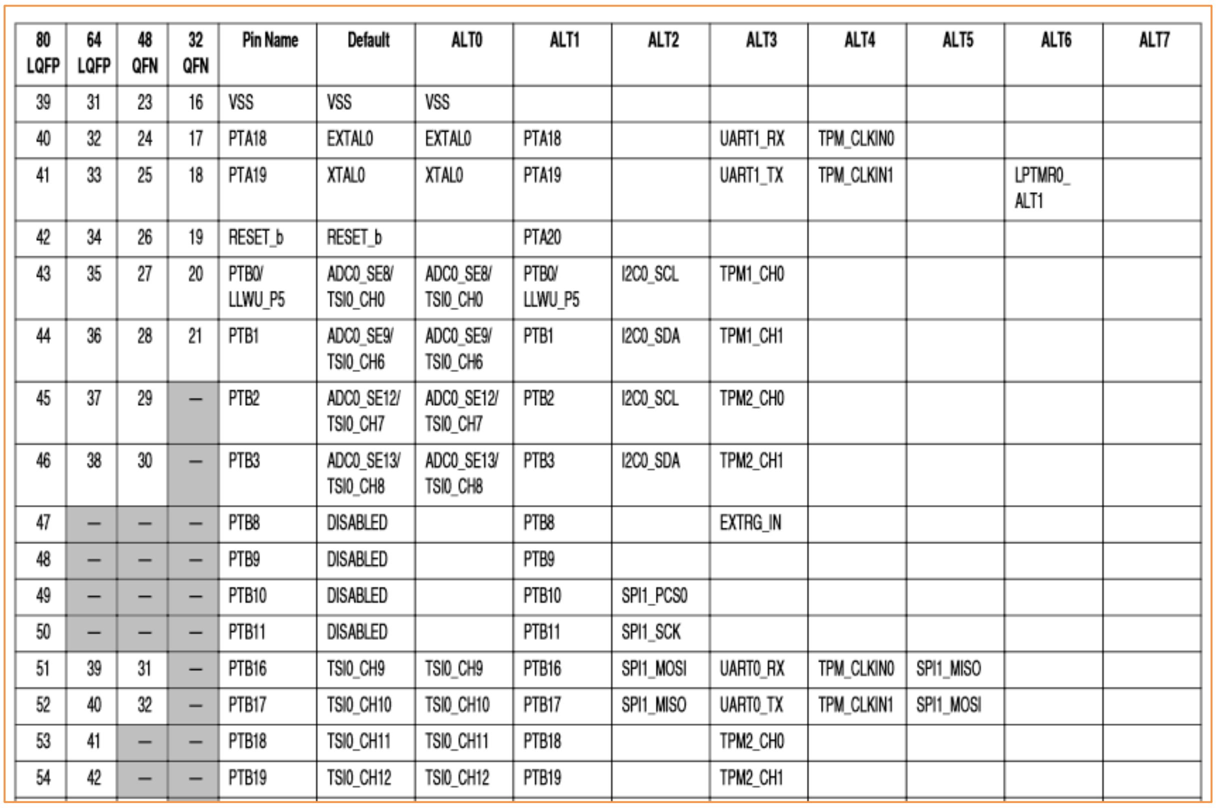
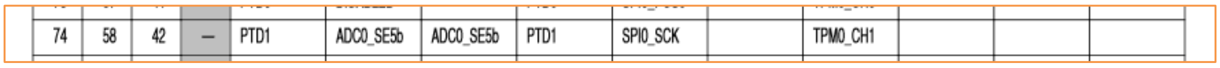
In order to configure the EHR and the MUX, we must first see what we need to use. However, according to KL25 Sub-Family Reference Manual, the DSE and the MUX can be configured using the Pin Control Register.
Here, the DSE is at the 6th bit and the MUX is between the 8th and 10th bits.
What we can easily check with the document below:



10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6 DSE	Drive Strength Enable This bit is read only for pins that do not support a configurable drive strength. Drive strength configuration is valid in all digital pin muxing modes. 0 Low drive strength is configured on the corresponding pin, if pin is configured as a digital output. 1 High drive strength is configured on the corresponding pin, if pin is configured as a digital output.

Then, to find out what value we need to put in the 6th bit, we need to look at the further documentation of the EHR. Similarly for the value to be put in bits 8 to 10 which correspond to the MUX:

We want to force the transition to 1 of the 6th bit because we want the parameter given the value 1 for the DES function, namely "High Drive Strenght". We want alternative 1 for the value of the MUX (because it is the one that gives the names of the pins that we have, see the document below) so we must put the value "001" between the 8th and the 10th bit .

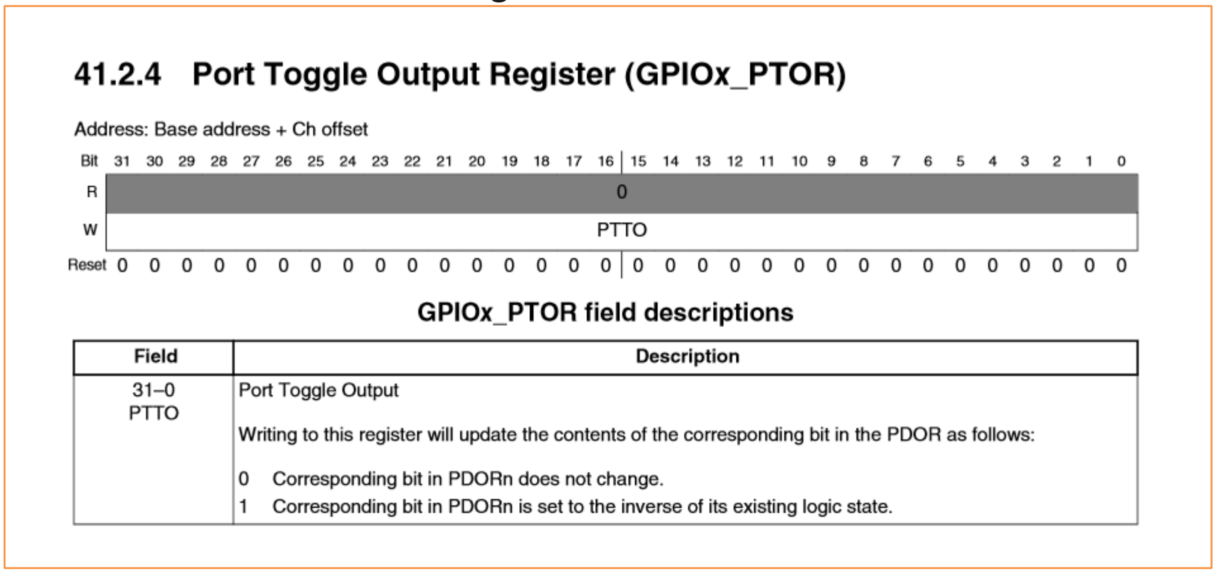



80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
39	31	23	16	VSS	VSS	VSS							
40	32	24	17	PTA18	EXTAL0	EXTAL0	PTA18		UART1_RX	TPM_CLKIN0			
41	33	25	18	PTA19	XTAL0	XTAL0	PTA19		UART1_TX	TPM_CLKIN1		LPTMR0_ ALT1	
42	34	26	19	RESET_b	RESET_b		PTA20						
43	35	27	20	PTB0/ LLWU_P5	ADC0_SE8/ TSI0_CH0	ADC0_SE8/ TSI0_CH0	PTB0/ LLWU_P5	I2C0_SCL	TPM1_CH0				
44	36	28	21	PTB1	ADC0_SE9/ TSI0_CH6	ADC0_SE9/ TSI0_CH6	PTB1	I2C0_SDA	TPM1_CH1				
45	37	29	—	PTB2	ADC0_SE12/ TSI0_CH7	ADC0_SE12/ TSI0_CH7	PTB2	I2C0_SCL	TPM2_CH0				
46	38	30	—	PTB3	ADC0_SE13/ TSI0_CH8	ADC0_SE13/ TSI0_CH8	PTB3	I2C0_SDA	TPM2_CH1				
47	—	—	—	PTB8	DISABLED		PTB8		EXTRG_IN				
48	—	—	—	PTB9	DISABLED		PTB9						
49	—	—	—	PTB10	DISABLED		PTB10	SPI1_PCS0					
50	—	—	—	PTB11	DISABLED		PTB11	SPI1_SCK					
51	39	31	—	PTB16	TSI0_CH9	TSI0_CH9	PTB16	SPI1_MOSI	UART0_RX	TPM_CLKIN0	SPI1_MISO		
52	40	32	—	PTB17	TSI0_CH10	TSI0_CH10	PTB17	SPI1_MISO	UART0_TX	TPM_CLKIN1	SPI1_MOSI		
53	41	—	—	PTB18	TSI0_CH11	TSI0_CH11	PTB18		TPM2_CH0				
54	42	—	—	PTB19	TSI0_CH12	TSI0_CH12	PTB19		TPM2_CH1				
...													
74	58	42	—	PTD1	ADC0_SE5b	ADC0_SE5b	PTD1	SPI0_SCK		TPM0_CH1			

```
PORTB_PCR18 = (1<<6) | (1<<8);
        PORTB_PCR19 = (1<<6) | (1<<8);
        PORTD_PCR1 = (1<<6)| (1<<8);

// Le (1<<6) est le forçage à 1 du DSE et le (1<<8) est le forçage à 001 du
MUX.
//Nous mettons les valeurs que nous avons définies juste au-dessus pour
chacun des
//PORTs et des broches (correspondant donc aux LEDs)
```

There are two ways to make the LEDs blink.
The first is the use of a TOGGLE which will automatically change the value of the previous LED so that it goes out when the TOGGLE is activated and lights up when the TOGGLE is switched off. This allows us to keep our hands on its use.
To do this, we use the Port Toggle Output Register (below), where we want to set the value "1" to the 11th bit to configure

the automatic status change of the blue LED.



41.2.4 Port Toggle Output Register (GPIOx_PTOR)

Address: Base address + Ch offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0																															
W	PTTO																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIOx_PTOR field descriptions

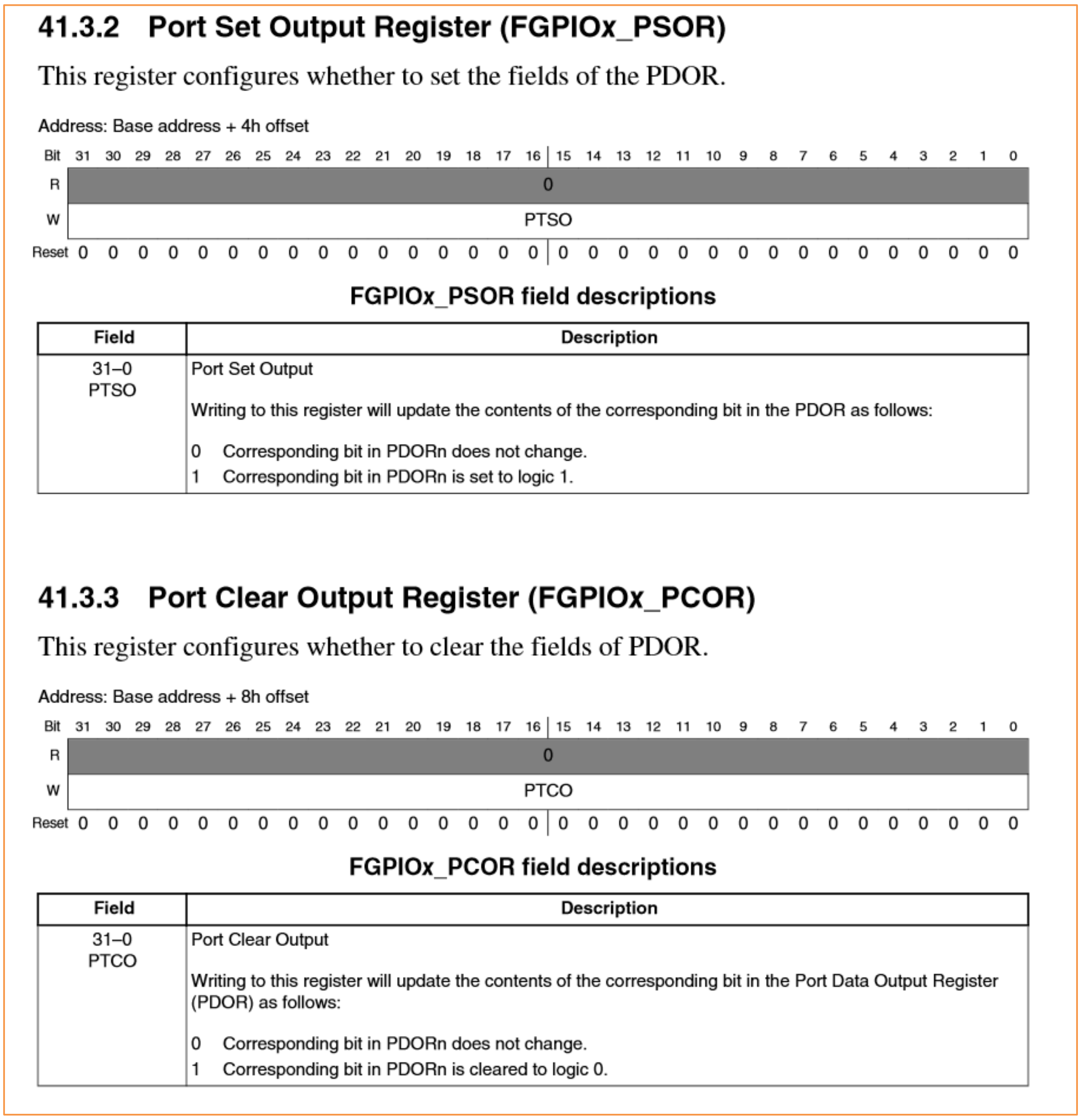
Field	Description
31–0 PTTO	Port Toggle Output Writing to this register will update the contents of the corresponding bit in the PDOR as follows: 0 Corresponding bit in PDORn does not change. 1 Corresponding bit in PDORn is set to the inverse of its existing logic state.

```
GPIO_PTOR = (1<<11) ; //Pour la LED bleue
```

The second way is to (manually) turn on and off the LEDs.
To do this, we use the function that turns the LED on and then the function that turns it off. A delay is necessary between the two in order to see the difference between the ignition and the off. (See the next page for documentation).

The corresponding code is as follows:

```
void main() {

void delay(int a)
//Configurer le délai entre l’allumage et l’éteinte des LEDs
{
             while(a>= 0) {a--;}
}

for(;;) {
delay(1000000);
GPIOB_PCOR = (1<<18);
GPIOD_PCOR = (1<<1);
delay(1000000);
GPIOD_PSOR = (1<<1);
GPIOB_PSOR = (1<<18);
}
}
```



41.3.2 Port Set Output Register (FGPIOx_PSOR)

This register configures whether to set the fields of the PDOR.

Address: Base address + 4h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0																															
W	PTSO																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FGPIOx_PSOR field descriptions

Field	Description
31–0 PTSO	Port Set Output Writing to this register will update the contents of the corresponding bit in the PDOR as follows: 0 Corresponding bit in PDORn does not change. 1 Corresponding bit in PDORn is set to logic 1.

41.3.3 Port Clear Output Register (FGPIOx_PCOR)

This register configures whether to clear the fields of PDOR.

Address: Base address + 8h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0																															
W	PTCO																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

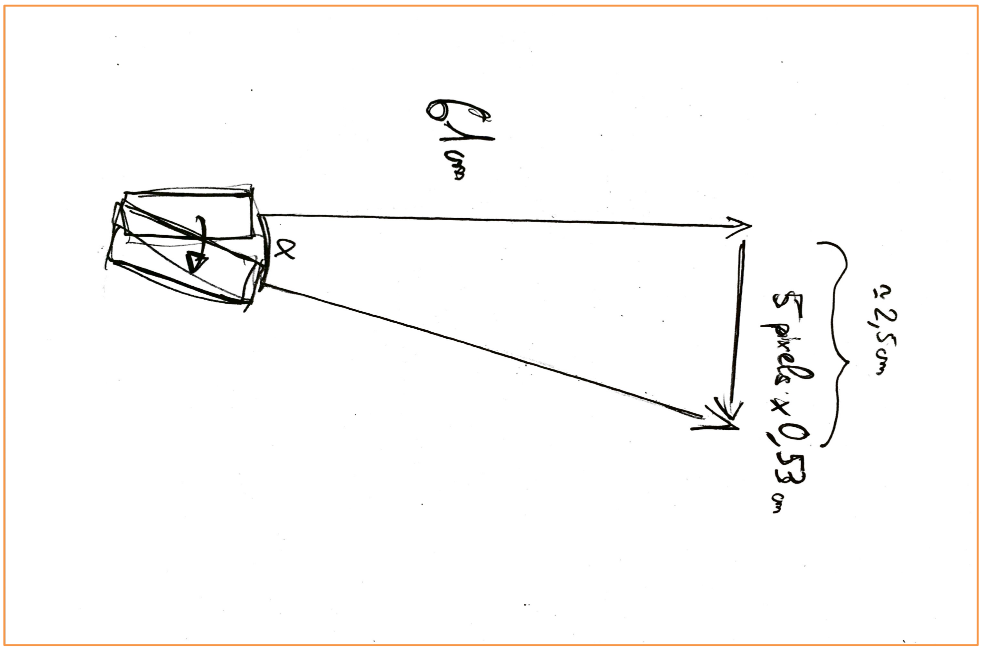
FGPIOx_PCOR field descriptions

Field	Description
31–0 PTCO	Port Clear Output Writing to this register will update the contents of the corresponding bit in the Port Data Output Register (PDOR) as follows: 0 Corresponding bit in PDORn does not change. 1 Corresponding bit in PDORn is cleared to logic 0.

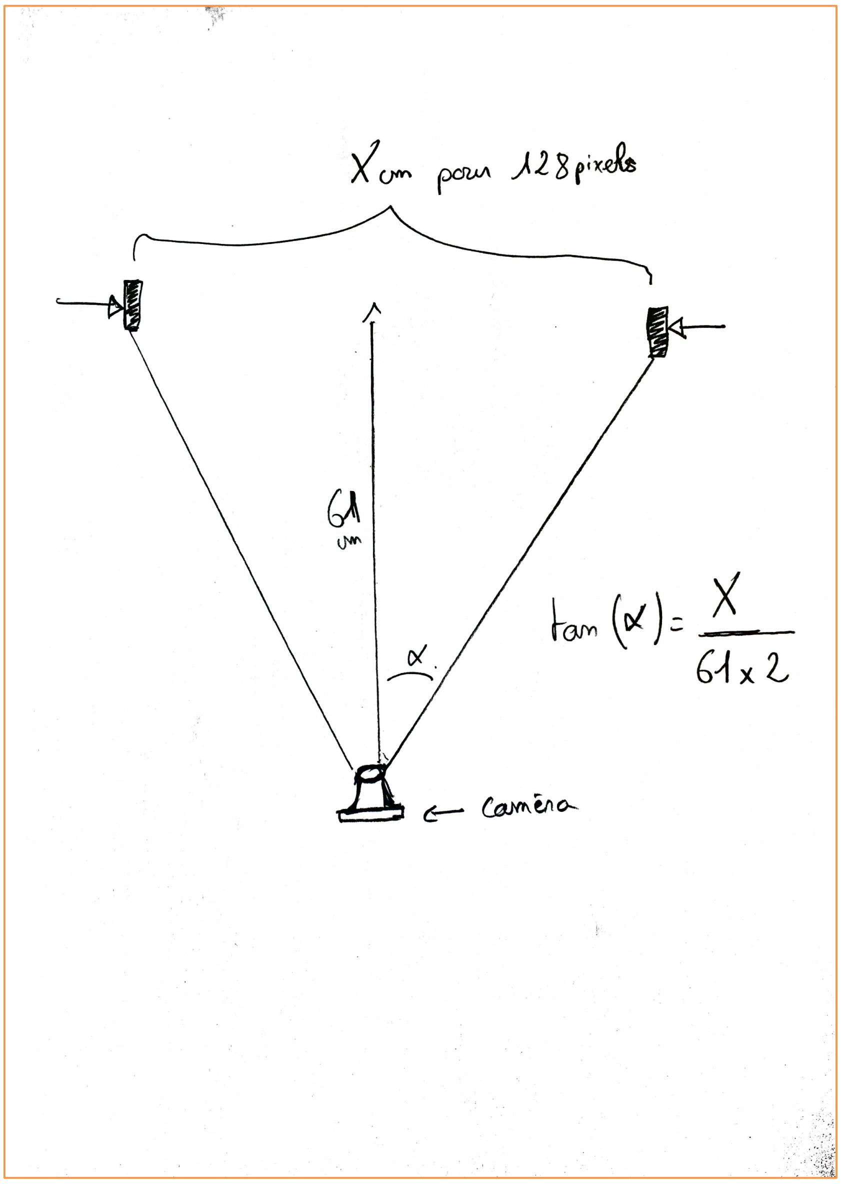
```
#include "derivative.h"
#define LED_B 0x02

void delay(int a) //Délai nécessaire à l'attente
{
    while( a>= 0) {a--;}
}

void main(){

    SIM_SCGC5 |= (1<<12); //Pour Activer le PORT D


    PORTD_PCR1 = (1<<6) | (1<<8); //Pour Activer le DSE et le MUX
    GPIOD_PDDR |= (LED_B);
//Initialiser la broche du PORT D en sortie (forçage du 1 à la place 1)

for(;;){

    delay(15000);
    GPIOD_PCOR = (1<<1); //Allumer la LED bleue
    delay(5000);
    GPIOD_PSOR = (1<<1); //Eteindre la LED bleue
}
}
```

We need to determine several values in order to fill certain parameters in our code. We had to know the inclination of the camera so that it does not see too far since in this case it was complicated to apprehend the road; But not too close either to anticipate the corners to the maximum and not to leave the predefined path.

To determine this inclination value, we had to determine two intermediate values beforehand: the maximum distance between the car and the camera (the location of the path that the camera sees) and the height between the ground and the camera. As a result, we were able to find (thanks to the tangent) the value of the optimum inclination of the camera.



To measure the pixel size needed to know the inclination of the wheels, we had to measure the total range that the camera saw and then divide by 128. So we found after the calculations a value of 0.53 cm for A pixel.

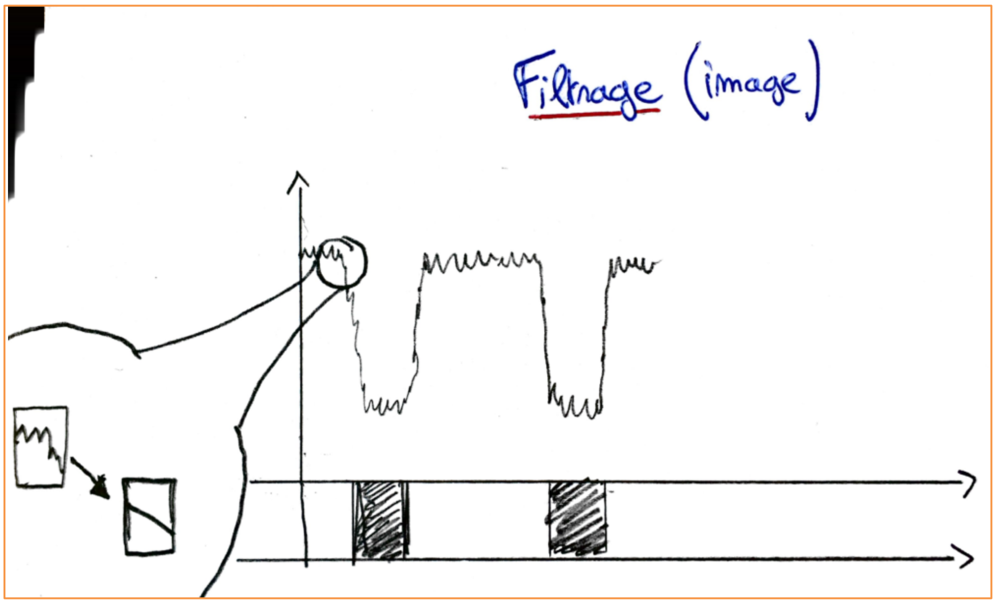


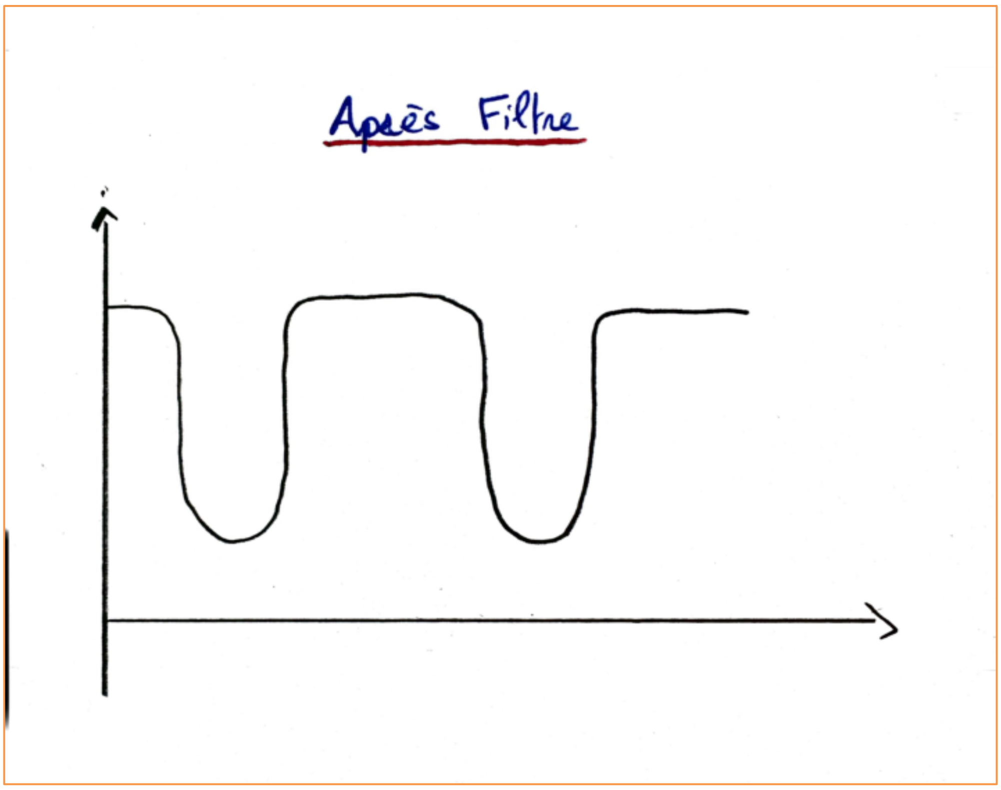
a. Caméra

So the objective of the code was to recover the image of the path given by the camera and to analyze it in order to deduce the position of the car with respect to the black bands.
The first method named void aquisitionImage (struct tab_camera * tab) acquires the image returned by the camera in order to analyze it and to deduce usable results.

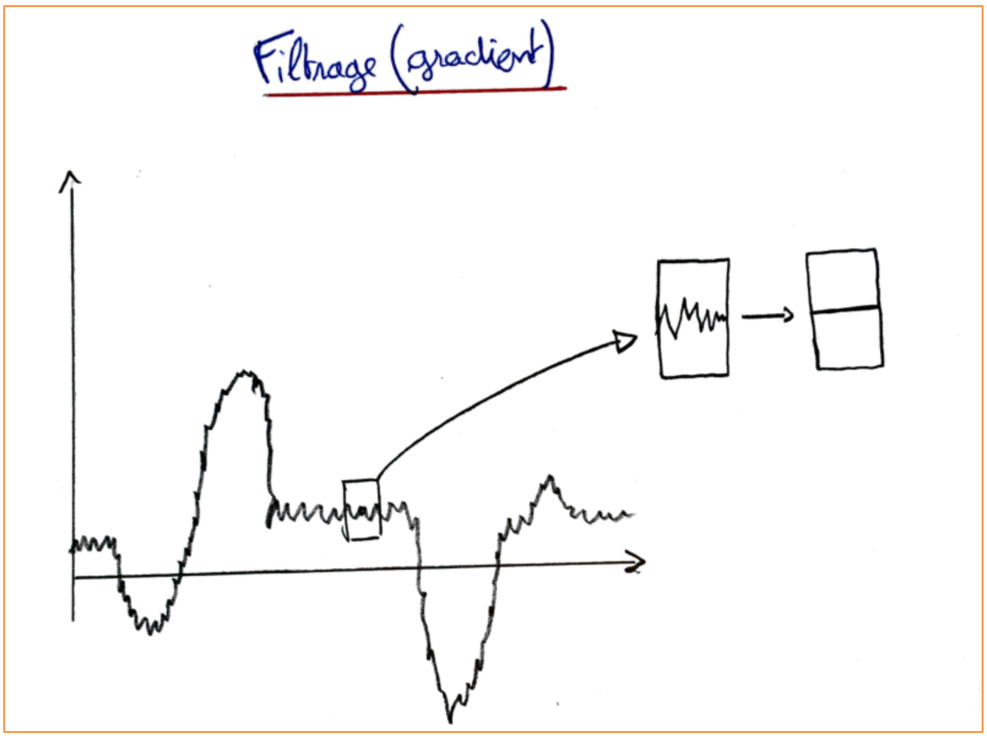
```
void aquisitionImage(struct tab_camera* tab){
// dans cette fonction on prend une image
    uint32_t i=0; // il s'agit d'un unsigned int 32bit.

       TFC_Ticker[0] = 0; // ce compteur sert a ne pas prendre d'image trop
souvent donc on le remet a zÉro dans la boucle.
       LineScanImageReady=0;// On remet lineScanImageReady a 0 car cet
variable repassera a 1 lorsque la camera sera pr□te.
         for(i=0;i<128;i++)    {
             //TERMINAL_PRINTF("%d ",LineScanImage0[i]);
             tab->image[i]=LineScanImage0[i];  // on prend les 128 pixels.
         }

}
```

Then we wrote another method, called void processing_Image (struct tab_camera * tab) and which allows to smooth the image, that is to eliminate the noises and other incoherent parasites (of the image of the Camera) to better control the car.
To illustrate the filtering performed by this code, here is a representative diagram:





The peaks that appear in these diagrams represent the gradient, calculated in order to know the difference between the white band and the black bands delimiting it:



Now we will see the code corresponding to this method and explain it as well as possible:

Now we will see the code corresponding to this method and explain it as well as possible:

```
void traitement_Image(struct tab_camera* tab){ // lisse limage
    int i,j,k;
    uint32_t moyenne_image[128];
    int grad[127];
    int moyenne_gradient[127];

//////////////////////////////////////////////////////
///////// calcul moyenne glissante de l'image/////////
//////////////////////////////////////////////////////

    for (i=NUM_PIXELS_MARGIN;i<128-NUM_PIXELS_MARGIN;i++){
/*cette boucle prend 5 pixels et fait la moyenne entre ces pixels pour
lisser l'image et eviter les pics engendrés par le bruit.
        Explications:
Le pixel dans tab->image va avoir, comme valeur dans moyenne-image, la
moyenne des 5 pixels autour de lui.
    */
      moyenne_image[i]=0;
      k=0;
      for (j=-(NUM_PIXELS_BORDERS-1)/2;j<(NUM_PIXELS_BORDERS-1)/2+1;j++){
          if ((i+j)>0 && (i+j)<128){
            moyenne_image[i]= moyenne_image[i]+tab->image[i+j];
            k++;
          }
      }
      moyenne_image[i]=moyenne_image[i]/k;
    }

//////////////////////////////////////////////////////////////////////
/////////calcul du gradient de l'image avec moyenne glissante/////////
//////////////////////////////////////////////////////////////////////

    for (i=NUM_PIXELS_MARGIN; i<127-NUM_PIXELS_MARGIN; i++) {
        // calcul du gradient une valeur moins celle d'avant (comme une
dÉrivÉe ).
        /* Si le gradient est positif alors la moyenne image est croissante
et donc les pixels passent du noir au blanc. Si le gradient est nÉgatifs
alors la moyenne-image est dÉcroissante et donc les pixels passent du blanc
au noir.
         */
        grad[i]=(moyenne_image[i+1] - moyenne_image[i]);
    }
```

```
///////////////////////////////////////////////////////////////////////
/////////moyenne glissante du gradient de l'image de la caméra ////////
/////////calcul de la moyenne du gradient lissé ///////////////////////
///////////////////////////////////////////////////////////////////////

    // calcul de la moyenne du gradient lissé
    float moyenne_grad=0;
    for (i=NUM_PIXELS_MARGIN;i<127-NUM_PIXELS_MARGIN;i++){
/*Tout comme la moyenne-image mais avec le gradient pour être sur de
n'avoir plus aucun bruit.*/

      moyenne_gradient[i]=0;
      k=0;
      for (j=-(NUM_PIXELS_BORDERS-1)/2;j<(NUM_PIXELS_BORDERS-1)/2+1;j++){
          if ((i+j)>0 && (i+j)<127){
            moyenne_gradient[i]= moyenne_gradient[i]+grad[i+j];
            k++;
          }
      }
      moyenne_gradient[i]=moyenne_gradient[i]/k;
      moyenne_grad=moyenne_grad+moyenne_gradient[i];
    }
    moyenne_grad = moyenne_grad/(127-2*NUM_PIXELS_MARGIN);

////////////////////////////////////////////////////////////////////////
/////////calcul de l'écart type pour faire un seuil intelligent/////////
```

```
/////////qui varie en fonction de la luminosité////////////////////////
//////////////////////////////////////////////////////////////////////////

    float ecart_type_grad=0.0;
    for (i=NUM_PIXELS_MARGIN; i<127-NUM_PIXELS_MARGIN; i++) {
        ecart_type_grad = ecart_type_grad+pow(moyenne_gradient[i]-
moyenne_grad,2);
    }
    ecart_type_grad = sqrt(ecart_type_grad/(127-2*NUM_PIXELS_MARGIN));

    int bande0=-1;
    int bande1=-1;
    int tmp0=-1;
    int tmp1=-1;
    //int bandeCentreTmp1, bandeCentreTmp2;
    int bandeGauche=-1, bandeDroite=129, bandeCentre=64;
```

```
    /*
     Dans la fonction qui suit l'algorithme est:
     bande0 parcourt le tableau de gauche a droite et bande1 de droite a
gauche. ( la camera est a l'envers donc le pixel 0 est a droite de la
voiture et le pixel 125 est ˆ gauche.
     Pour bande0: On cherche un pique inferieur a 0 ( passage du blanc au
noir ) suivit d'un passage du noir au blanc superieur au seuil. On divise
par deux ces deux seuil pour obtenir le centre de la bande de gauche
thÉoriquement.
     Pour bande1: On cherche un pique superieur a 0 ( passage du noir au
blanc ) suivit d'un passage du blanc au noir inferieur au seuil. On divise
par deux ces deux seuil pour obtenir le centre de la bande de droite
théoriquement.
     Ce qu'il faut savoir c'est que si a l'exterieur de la piste il y avais
eu autre chose que du blanc le gradient aurait été moins élevé car le
passage du noir au blanc et le plus important. C'est pour cela que le seuil
du premier petit pique etait ˆ 0.
     */


    for(k=NUM_PIXELS_MARGIN;k<127-NUM_PIXELS_MARGIN;k++){
/*NUM_PIXELS_MARGIN sert a enlever les pixels exterieur car Freescale
disait qu'il fallait les enlever car il etait noir par defaut*/
          if(bande0<0){
                   if (moyenne_gradient[k]<0){ //+ecart_type_grad*N_SIGMA){
                          tmp0=k;
                        } else if
((moyenne_gradient[k]>moyenne_grad+ecart_type_grad*N_SIGMA)&&(tmp0>0)){
                                bande0=k+(NUM_PIXELS_BORDERS-1)/2;
// on cherche une transition blanc noir significative (> moyenne-N SIGMA)
                        }
```

```
           }
         if(bande1<0){
                        if (moyenne_gradient[126-k]>0){
//+ecart_type_grad*N_SIGMA){
                          tmp1=k;
                        } else if ((moyenne_gradient[126-k]<moyenne_grad-
ecart_type_grad*N_SIGMA)&&(tmp1>0)){
                          bande1=126-k+(NUM_PIXELS_BORDERS-1)/2;
//on cherche une transition blanc noir dans l'autre sens
                        }
          }
    } /* A partir de là, nous avons détecter la ou les bandes noires. Il
reste à déterminer si c'est la bande de droite ou de gauche.*/
```

```
    /*Dans l'algorithme ci dessous on cherche à déterminer grâce a
l'historique les bandes de droite et de gauche.*/
    if((bande0!=-1) && (bande1!=-1) ){// la camera voit deux bandes
        bandeDroite = 128-bande0 ;
                bandeGauche = 128-bande1 ;
                if (abs(bandeDroite-bandeGauche)>2*NUM_PIXELS_BORDERS){
// si les bandes sont suffisament espacées
                        if (abs(bandeDroite-bandeGauche)>1.5*tab->curseur){
                                bandeCentre = (int) (((128-bande0) - tab-
>curseur)+((128-bande1) + tab->curseur))/2;
                        } else{
// si les valeurs sont éronnées, on reprend l'ancien centre
                                bandeCentre = tab->TableauBandes[CENTER][tab-
>indexBandes1];
                        }
                }
                else {
                        bandeCentre = (128-bande1)+ tab->curseur;
                        if(abs(bandeCentre-tab->TableauBandes[CENTER][tab-
>indexBandes1])>=1.0*tab->curseur){
                                bandeDroite = 128-bande1 ;
                                bandeCentre = bandeDroite - tab->curseur ;
             }
                }

        }
    else {
        if(bande1!=-1){ // si on ne voit que la bande de droite
                        bandeCentre = (128-bande1)+ tab->curseur;
                        if(abs(bandeCentre-tab->TableauBandes[CENTER][tab-
>indexBandes1])>=1.0*tab->curseur){
```

```
                // on verifie si le centre trouvé n'est pas éronné en le
comparant avec l'ancien centre
                              bandeDroite = 128-bande1 ;
                              bandeCentre = bandeDroite - tab->curseur ;
//on fait ça car tab->curseur est le nombre de pixel qui faut enlever à la
bande de droite pour avoir le centre
                       }
       }
       else if(bande0!=-1){ // si on ne voit que la bande de gauche
                       bandeCentre = (128-bande0)- tab->curseur;
                       if(abs(bandeCentre-tab->TableauBandes[CENTER][tab-
>indexBandes1])>=1.0*tab->curseur){
                // on verifie si le centre trouvé n'est pas éronné en le
comparant avec l'ancien centre
                bandeGauche = 128-bande0 ;
                              bandeCentre = bandeGauche + tab->curseur ;
// on fait ça car tab->curseur est le nombre de pixel qui faut ajouter à la
bande de gauche pour avoir le centre             }
                }

    }
```

```
     tab->indexBandes1++;
     tab->indexBandes1=tab->indexBandes1 % 36;
     tab->TableauBandes[LEFT][tab->indexBandes1]= bandeCentre-tab->curseur;
// la camera est retournée
     tab->TableauBandes[RIGHT][tab->indexBandes1]= bandeCentre+tab-
>curseur;
     tab->TableauBandes[CENTER][tab->indexBandes1]= bandeCentre;
}
```

A part concerning the camera is its calibration according to the luminosity of the room in order to overcome the problems of too much sunshine and therefore not enough shades of gray to see correctly the black bands; Or totally the opposite, ie too little sunlight, not allowing us to distinguish between black and white bands.
This part is therefore done through two methods:
`void Calibration_Exposition_Time(struct tab_camera* tab)` et `void traitement_Image_Blanc(struct tab_camera* tab)`

```
void Calibration_Exposition_Time(struct tab_camera* tab){
/*
     Cette fonction sert a calibrer la camera en fonction de la luminositÉ
de la pi□ces.
```

```
     Explication:
     La camera se calibre d'environ 10.000 jusqu'^ 100.000 en augmentant de
20.000. Lors de toute ces calibrations elle calcul la difference entre le
max et le min. Plus la valeur ( Max-Min ) est importante plus la caibration
est efficace. Et la fonction a la fin prend la calibration pour laquelle
Max-Min est le plus grand.
     */

        uint32_t i,j=1;
        uint32_t Delta_Min_Max[45];
        uint32_t Min,Max;

        TFC_SetLineScanExposureTime(10000);

        while(j<45){ // cette boucle fait varier la calibration
                 if(TFC_Ticker[0]>100 && LineScanImageReady==TRUE) {
                           TFC_Ticker[0] = 0;
                           LineScanImageReady=FALSE;
                           Min=LineScanImage0[NUM_PIXELS_MARGIN];
                           Max=LineScanImage0[NUM_PIXELS_MARGIN];
                           for(i=NUM_PIXELS_MARGIN;i<128-
NUM_PIXELS_MARGIN;i++){ // on cherche le min et le max dans les pixels
                                if(Min>LineScanImage0[i]){
                                        Min = LineScanImage0[i];
                                }
                                if(Max<LineScanImage0[i]){
                                        Max = LineScanImage0[i];
                                }
                           }
                           Delta_Min_Max[j]=Max-Min; // on calcul la
difference et on cherche si il est plus grand on pas.
                           j=j+1;
                          TFC_SetLineScanExposureTime(10000+j*2000);

                 }
                 if (j%2==0){
// ces LEDs servaient a debeuguer le programme.
        TFC_BAT_LED3_ON;
                }
                else{
                        TFC_BAT_LED3_OFF;
                }
        }

        Max=0;
        for(j=1;j<45;j++){
// cette boucle cherche le delta le plus grand pour trouver la bonne
calibration
                if(Max<Delta_Min_Max[j]){
                        Max = Delta_Min_Max[j];
                        tab->Exposition = 10000+j*2000;
                }
        }
}
```

```
void traitement_Image_Blanc(struct tab_camera* tab){
/* Cette fonction est plus simple que l'autre traitement mais quasi
identique il est juste utilisé pour la calibration.
     */
```

```
void PID(struct tab_camera* tab){
    // il faut changer les valeurs donné de maniere empirique
    int variation_erreur = 0 ;
    int index =0;

    tab->mesure = tab->out_BandeCentre;
    tab->erreur = tab->mesure - 64 ;
    //tab->somme_erreur = tab->somme_erreur + tab->erreur;
    variation_erreur = tab->erreur - tab->erreur2;
    index = abs(tab->erreur);
    tab->CommandeDirection = (index/tab->erreur)*(tab-
>TableauOrientationPID[index])+ 0.01*(variation_erreur);
    //commande = Kp * erreur + Ki * somme_erreurs + Kd * variation_erreur;
    tab->CommandeVitesse = tab->TableauVitessePID[index]; // la vitesse
doit diminuer lorsque l'erreur augmente

    tab->erreur2 = tab->erreur;
    }
```

```
void init_Tab (struct tab_camera* tab){
    int j=0;
    tab->indexBandes1=0;

    tab->stopbandes[0]=-1;
    tab->stopbandes[1]=-1;
    tab->erreur=0;
    tab->erreur2=0;
    tab->somme_erreur=0;
    tab->out_BandeCentre=0;
    tab->mesure=0;
    tab->CommandeDirection=0;
    tab->CommandeVitesse=0;
    tab->curseur=DistanceCentreBande;
    tab->Exposition=20000;

    for(j=0;j<36;j++){
        tab->TableauBandes[LEFT][j]= -1;
        tab->TableauBandes[RIGHT][j]= 129;
        tab->TableauBandes[CENTER][j]= 64;
    }

    //////// PID /////////
    //Les valeurs des tableau ont ÉtÉ mesurÉ puis enregistrer dans le
tableau pour eviter au processeur de faire tous les calculs.

    for(j=0;j<2;j++){
        tab->TableauOrientationPID[j]= 0.0;
    }
    for(j=2;j<4;j++){
        tab->TableauOrientationPID[j]=0.0 ;
    }
    for(j=4;j<6;j++){
        tab->TableauOrientationPID[j]= 0.05 ;
    }
    for(j=6;j<8;j++){
        tab->TableauOrientationPID[j]= 0.1;
    }
    for(j=8;j<10;j++){
        tab->TableauOrientationPID[j]= 0.15 ;
    }
    for(j=10;j<12;j++){
        tab->TableauOrientationPID[j]= 0.2 ;
    }
    for(j=12;j<14;j++){
        tab->TableauOrientationPID[j]= 0.25;
    }
    for(j=14;j<16;j++){
        tab->TableauOrientationPID[j]= 0.3 ;
    }
    for(j=16;j<18;j++){
        tab->TableauOrientationPID[j]= 0.35 ;
    }
    for(j=18;j<20;j++){
        tab->TableauOrientationPID[j]= 0.4;
    }
    for(j=20;j<22;j++){
        tab->TableauOrientationPID[j]= 0.45 ;
    }
    for(j=22;j<24;j++){
        tab->TableauOrientationPID[j]= 0.5;
    }
    for(j=24;j<26;j++){
        tab->TableauOrientationPID[j]= 0.55 ;
    }
    for(j=26;j<28;j++){
        tab->TableauOrientationPID[j]= 0.6;
    }
    for(j=28;j<30;j++){
        tab->TableauOrientationPID[j]=0.65 ;
    }
    for(j=30;j<32;j++){
        tab->TableauOrientationPID[j]= 0.7 ;
    }
    for(j=32;j<34;j++){
      tab->TableauOrientationPID[j]= 0.75;
    }
    for(j=34;j<36;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=36;j<38;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=38;j<40;j++){
        tab->TableauOrientationPID[j]= 0.8;
    }
    for(j=40;j<42;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=42;j<44;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=44;j<46;j++){
        tab->TableauOrientationPID[j]= 0.8;
    }
    for(j=46;j<48;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=48;j<50;j++){
        tab->TableauOrientationPID[j]= 0.8;
    }
    for(j=50;j<52;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=52;j<54;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=54;j<56;j++){
        tab->TableauOrientationPID[j]= 0.8;
    }
    for(j=56;j<58;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }
    for(j=58;j<60;j++){
            tab->TableauOrientationPID[j]= 0.8;
    }
    for(j=60;j<127;j++){
        tab->TableauOrientationPID[j]= 0.8 ;
    }

    for(j=0;j<5;j++){
        tab->TableauVitessePID[j]= 0.5+vitesse;
    }
    for(j=5;j<7;j++){
        tab->TableauVitessePID[j]= 0.4+vitesse;
    }
    for(j=7;j<10;j++){
        tab->TableauVitessePID[j]= 0.3+vitesse;
    }
    for(j=10;j<20;j++){
        tab->TableauVitessePID[j]=0.3+vitesse ;
    }
    for(j=20;j<30;j++){
        tab->TableauVitessePID[j]= 0.2+vitesse ;
    }
    for(j=30;j<40;j++){
        tab->TableauVitessePID[j]= 0.2+vitesse;
    }
    for(j=40;j<50;j++){
        tab->TableauVitessePID[j]= 0.2+vitesse ;
    }
    for(j=50;j<127;j++){
        tab->TableauVitessePID[j]= 0.2+vitesse ;
    }

}
```