```
/*
 * Title: Code C de pilotage de camera Audine avec Raspberry
 * Author: A. Klotz. Universite de Toulouse
 *
 * To install the GPIO access library BCM2835 for Ubuntu:
 * Some infos: http://www.airspayce.com/mikem/bcm2835/
 * Install procedure: https://raspberry-projects.com/pi/programming-in-c/io-pins/bcm2835-by-mike-mccauley
 * Do not forget to change the rights of the driver:
 * $ sudo chown user:user /dev/gpiomem
 * $ sudo crontab -e   and add the line @reboot chown user:user /dev/gpiomem
 *
 * Code::Blocks configuration to use the GPIO library BCM2835 in C code:
 * Menu Settings -> Compiler...
 * Tab "Linker Settings", "Link libraries" [Add] bcm2835 [OK]
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <bcm2835.h>

#define PHI_V1 RPI_BPLUS_GPIO_J8_11
#define PHI_V2 RPI_BPLUS_GPIO_J8_13
#define PHI_H1 RPI_BPLUS_GPIO_J8_15
#define PHI_R  RPI_BPLUS_GPIO_J8_16
#define PHI_CL RPI_BPLUS_GPIO_J8_18
#define PHI_SC RPI_BPLUS_GPIO_J8_22
#define PHI_SB RPI_BPLUS_GPIO_J8_32
#define PHI_SN RPI_BPLUS_GPIO_J8_36

#define BIT_0  RPI_BPLUS_GPIO_J8_29
#define BIT_1  RPI_BPLUS_GPIO_J8_31
#define BIT_2  RPI_BPLUS_GPIO_J8_33
#define BIT_3  RPI_BPLUS_GPIO_J8_35

#define DELAY  RPI_BPLUS_GPIO_J8_40

#define LEVEL_HIGH 0
#define LEVEL_LOW 1

// -- Structure of camera informations
struct camprop {
   char msg[2048];
   float exptime;
   int binx, biny;
   int x1, y1, x2, y2;
   int w, h;
```

```
  double celldimx;
  double celldimy;
  int overscanindex;
  int nb_deadbeginphotox;
  int nb_deadendphotox;
  int nb_deadbeginphotoy;
  int nb_deadendphotoy;
  int nb_photox;
  int nb_photoy;
  float *p;
};

// -- Declaration of functions
int tp_acquisition_fullframe(struct camprop *cam);
static int tp_fast_vidage(struct camprop *cam);
static int tp_read_frame(struct camprop *cam);
static void tp_sleep(float us);
static int tp_zi_zh(struct camprop *cam);
static int tp_read_pel_fast2(struct camprop *cam);
int tp_fast_line(struct camprop *cam);

/*
 * Wait in micro-seconds.
 */
void tp_sleep(float us)
{
  int i,n;
  n = (int)(us*10);
  for (i=0; i<n ; i++) {
    bcm2835_gpio_write (DELAY, 1) ;
  }
}

int tp_acquisition_fullframe(struct camprop *cam)
{
  int i;
  int nb_vidages = 2;
  /* ============================================== */
  /* === Etape de rincage de la matrice CCD      === */
  /* ============================================== */
  // Vidage de la matrice
  for (i = 0; i < nb_vidages; i++) {
    tp_fast_vidage(cam);
  }
  /* ============================================== */
  /* === Integration de l'image (attente)        === */
  /* ============================================== */
  // Delais du temps de pose (en micro-secondes)
  tp_sleep((int) (1e6 * cam->exptime));
```

```
  /* ============================================== */
  /* === Etape de lecture de la matrice CCD      === */
  /* ============================================== */
  // Parametres de dimensions pour allouer le pointeur image
  cam->w = cam->nb_photox / cam->binx;
  cam->h = cam->nb_photoy / cam->biny;
  // Allocation memoire du pointeur image
  cam->p = (float *) calloc(cam->w*cam->h, sizeof(float));
  // Lecture et numï¿½risation de l'image vers le pointeur p
  tp_read_frame(cam);
  return 0;
}

/*
fast_vidage(struct camprop *cam) --
  Vidage rapide de la matrice. Le decalage des lignes s'effectue
  ici par groupe de 20. C'est le seul parametre a regler ici.
*/
int tp_fast_vidage(struct camprop *cam)
{
  int i, j;
  int imax, jmax, decaligne;

  // -- Nombre total de photocellules dans le registre horizontal.
  imax = cam->nb_photox + cam->nb_deadbeginphotox + cam->nb_deadendphotox;

  // -- Nombre total de photocellules dans une colonne de la matrice.
  jmax = cam->nb_photoy + cam->nb_deadbeginphotoy + cam->nb_deadendphotoy;

  // -- Nombre de lignes decalees a chaque boucle sur l'axe vertical.
  decaligne = 20;

  // -- Nombre total de boucles sur l'axe vertical.
  jmax = (int) ceil (1. * jmax / decaligne) ;

  // -- Boucle sur les paquets de lignes.
  for (j = 0; j < jmax; j++) {
    // -- Decalage d'un paquet de 'decaligne' lignes.
    for (i = 0; i < decaligne; i++) {
      tp_zi_zh(cam);
    }
    // -- Lecture du registre horizontal sans reset
    for (i = 0; i < imax; i++) {
      tp_read_pel_fast2(cam);
    }
  }
  return 0;
}
```

```
/*
tp_zi_zh(struct camprop *cam) --
  Decalage vertical de toutes les lignes d'un cran vers le bas.
  La premiere ligne du bas est donc transferee dans le registre
  horizontal.
*/
int tp_zi_zh(struct camprop *cam)
{
   float tphiV=2.0;
   float tphiHS=1.0;
   // ---
   bcm2835_gpio_write (PHI_V1, LEVEL_HIGH);
   tp_sleep(tphiV);
   bcm2835_gpio_write (PHI_V1, LEVEL_LOW);
   bcm2835_gpio_write (PHI_V2, LEVEL_HIGH);
   tp_sleep(tphiV);
   bcm2835_gpio_write (PHI_V1, LEVEL_HIGH);
   bcm2835_gpio_write (PHI_V2, LEVEL_LOW);
   tp_sleep(tphiV);
   bcm2835_gpio_write (PHI_V1, LEVEL_LOW);
   tp_sleep(tphiHS);
   return 0;
}

/*
tp_read_pel_fast2(struct camprop *cam) --
  Lecture rapide d'une photocellule : decalage du registre horizontal
  sans Reset, mais sans lecture du CAN,
*/
int tp_read_pel_fast2(struct camprop *cam)
{
   float tphiPIX=0.25;
   // ---
   bcm2835_gpio_write (PHI_H1, LEVEL_LOW);
   tp_sleep(tphiPIX);
   bcm2835_gpio_write (PHI_H1, LEVEL_HIGH);
   tp_sleep(tphiPIX);
   return 0;
}

/*
tp_read_pel_fast(struct camprop *cam) --
  Lecture rapide d'une photocellule : decalage du registre horizontal
  avec Reset, mais sans lecture du CAN,
*/
int tp_read_pel_fast(struct camprop *cam)
{
   float tphiPIX=0.25;
```

```
  float tphiR=0.020;
  // ---
  bcm2835_gpio_write (PHI_R, LEVEL_HIGH);
  tp_sleep(tphiR);
  bcm2835_gpio_write (PHI_R, LEVEL_LOW);
  tp_sleep(tphiR);
  bcm2835_gpio_write (PHI_H1, LEVEL_LOW);
  tp_sleep(tphiPIX);
  bcm2835_gpio_write (PHI_H1, LEVEL_HIGH);
  tp_sleep(tphiPIX);
  return 0;
}

/*
tp_fast_line_() --
 Lecture rapide du registre horizontal, avec la fonction tp_read_pel_fast.
*/
int tp_fast_line(struct camprop *cam)
{
  int i, imax;
  // Nombre total de photocellules dans le registre horizontal.
  imax = cam->nb_photox + cam->nb_deadbeginphotox + cam->nb_deadendphotox;
  for (i = 0; i < imax; i++) {
    tp_read_pel_fast(cam);
  }
  return 0;
}

int tp_read_frame(struct camprop *cam)
{
  int i, j;
  int k, l;
  int imax, jmax;
  int cx1, cx2, cy1;
  unsigned short buffer[2048];
  unsigned short x;

  // -- Dimensions de l'image a digitaliser
  imax = cam->w;
  jmax = cam->h;
  // -- Nombre de photocellules de debut de ligne a ne pas digitaliser
  cx1 = cam->nb_deadbeginphotox;
  // -- Nombre de photocellules de fin de ligne a ne pas digitaliser
  cx2 = cam->nb_deadendphotox;
  // -- Nombre de lignes de debut a ne pas digitaliser
  cy1 = cam->nb_deadbeginphotoy;

  // -- On supprime les cy1 premieres lignes
  for (i = 0; i < cy1; i++) {
```

```
  tp_zi_zh(cam);
}

// -- On nettoie le registre horizontal
for (i = 0; i < 1; i++) {
  tp_fast_line(cam);
}

// -- Boucle sur le transfert vertical
for (i = 0; i < jmax; i++) {

  // -- Boucle de binning vertical
  for (k = 0; k < cam->biny; k++) {
    tp_zi_zh(cam);
  }

  // -- On retire les cx1 premieres photocellules avec reset
  for (j = 0; j < cx1; j++) {
    tp_read_pel_fast(cam);
  }

  // -- Boucle sur le transfert horizontal
  for (j = 0; j < imax; j++) {
    float tphiR=2.0; //0.020;
    float tRef1=4.0;
    float tRef2=1.0;
    float tRef3=1.0;
    float tphiPIX=1.0; // 0.25
    float tVid1=4.0;
    float tVid2=1.0;
    float tSel=1.0;

    // -- Palier de reset
    bcm2835_gpio_write (PHI_H1, LEVEL_LOW) ;
    bcm2835_gpio_write (PHI_R,  LEVEL_HIGH) ;
    tp_sleep(tphiR);
    bcm2835_gpio_write (PHI_R,  LEVEL_LOW) ;

    // -- Palier de reference
    tp_sleep(tRef1);
    bcm2835_gpio_write (PHI_CL, LEVEL_HIGH) ;
    tp_sleep(tRef2);
    bcm2835_gpio_write (PHI_CL, LEVEL_LOW) ;
    tp_sleep(tRef3);

    // -- Boucle de binning horizontal
    for (l = 0; l < cam->binx; l++) {
      bcm2835_gpio_write (PHI_H1, LEVEL_LOW) ;
      tp_sleep(tphiPIX);
```

```
  bcm2835_gpio_write (PHI_H1, LEVEL_HIGH) ;
  tp_sleep(tphiPIX);
}

// -- Palier video
tp_sleep(tVid1);
bcm2835_gpio_write (PHI_SC, LEVEL_HIGH) ;
tp_sleep(tVid2);

// -- Recupere le nibble _ _ _ _  _ _ _ _  _ _ _ _ 3 2 1 0
// SB = Select Byte      <---- SB=1 ----> <---- SB=0 ---->
// SN = Select Nibble    <SN=1 >  <SN=0 > <SN=1 >  <SN=0 >
bcm2835_gpio_write (PHI_SB, LEVEL_LOW) ;
bcm2835_gpio_write (PHI_SN, LEVEL_LOW) ;
tp_sleep(tSel);
x  = (unsigned short)bcm2835_gpio_lev(BIT_0) ;
x += (unsigned short)bcm2835_gpio_lev(BIT_1) << 1;
x += (unsigned short)bcm2835_gpio_lev(BIT_2) << 2;
x += (unsigned short)bcm2835_gpio_lev(BIT_3) << 3;

// -- Recupere le nibble _ _ _ _  _ _ _ _  3 2 1 0  _ _ _ _
bcm2835_gpio_write (PHI_SB, LEVEL_LOW) ;
bcm2835_gpio_write (PHI_SN, LEVEL_HIGH) ;
tp_sleep(tSel);
x += (unsigned short)bcm2835_gpio_lev(BIT_0) << 4;
x += (unsigned short)bcm2835_gpio_lev(BIT_1) << 5;
x += (unsigned short)bcm2835_gpio_lev(BIT_2) << 6;
x += (unsigned short)bcm2835_gpio_lev(BIT_3) << 7;

// -- Recupere le nibble _ _ _ _  3 2 1 0  _ _ _ _  _ _ _ _
bcm2835_gpio_write (PHI_SB, LEVEL_HIGH) ;
bcm2835_gpio_write (PHI_SN, LEVEL_LOW) ;
tp_sleep(tSel);
x += (unsigned short)bcm2835_gpio_lev(BIT_0) << 8;
x += (unsigned short)bcm2835_gpio_lev(BIT_1) << 9;
x += (unsigned short)bcm2835_gpio_lev(BIT_2) << 10;
x += (unsigned short)bcm2835_gpio_lev(BIT_3) << 11;

// -- Recupere le nibble 3 2 1 0  _ _ _ _  _ _ _ _  _ _ _ _
bcm2835_gpio_write (PHI_SB, LEVEL_HIGH) ;
bcm2835_gpio_write (PHI_SN, LEVEL_HIGH) ;
tp_sleep(tSel);
x += (unsigned short)bcm2835_gpio_lev(BIT_0) << 12;
x += (unsigned short)bcm2835_gpio_lev(BIT_1) << 13;
x += (unsigned short)bcm2835_gpio_lev(BIT_2) << 14;
x += (unsigned short)bcm2835_gpio_lev(BIT_3) << 15;

// -- Remises aux niveaux bas
bcm2835_gpio_write (PHI_SC, LEVEL_LOW) ;
```

```
      bcm2835_gpio_write (PHI_SB, LEVEL_LOW) ;
      bcm2835_gpio_write (PHI_SN, LEVEL_LOW) ;

      // -- Stockage du pixel dans un buffer de ligne
      buffer[j] = x;

    }

    // -- On retire cx2 photocellules a la fin
    for (j = 0; j < cx2; j++) {
      tp_read_pel_fast(cam);
    }

    // -- On transfere le buffer de ligne vers la matrice image
    if (i != 0) {
      cam->p[(i - 1) * imax] = (float)buffer[0];
    }
    for (j = 1; j < imax; j++) {
      cam->p[(i + 1) * imax - j] = (float)buffer[j];
    }

  }
  return 0;
}

void tp_savefits(struct camprop *cam, char *filename)
{
  FILE *f;
  char line[1024],car;
  short value0;
  char *cars0;
  int k,k0;
  long one= 1;
  int big_endian;
  f=fopen(filename,"wb");
  strcpy(line,"SIMPLE  =                    T / file does conform to FITS standard             ");
  fwrite(line,80,sizeof(char),f);
  strcpy(line,"BITPIX  =                   16 / number of bits per data pixel                  ");
  fwrite(line,80,sizeof(char),f);
  strcpy(line,"NAXIS   =                    2 / number of data axes                           ");
  fwrite(line,80,sizeof(char),f);
  sprintf(line,"NAXIS1  =                    %3d / length of data axis 1                       ",cam->w);
  fwrite(line,80,sizeof(char),f);
  sprintf(line,"NAXIS2  =                    %3d / length of data axis 2                       ",cam->h);
  fwrite(line,80,sizeof(char),f);
  strcpy(line,"BZERO   =                32768 / uint16 32768                                  ");
  fwrite(line,80,sizeof(char),f);
  strcpy(line,"BSCALE  =                    1 / uint16 1                                      ");
  fwrite(line,80,sizeof(char),f);
```

```
  k0=9;
  strcpy(line,"END                                                              ");
  fwrite(line,80,sizeof(char),f);
  strcpy(line,"                                                                ");
  for (k=k0;k<=36;k++) {
    fwrite(line,80,sizeof(char),f);
  }
  // -- Byte order test
  if (!(*((char *)(&one)))) {
    big_endian=1;
  } else {
    big_endian=0;
  }
  // -- Write data
  for (k=0;k<cam->h*cam->w;k++) {
    value0 = cam->p[k]-32768;
    if (big_endian==0) {
      cars0=(char*)&value0;
      car=cars0[0];
      cars0[0]=cars0[1];
      cars0[1]=car;
    }
    fwrite(&value0,1,sizeof(short),f);
  }
  int n= 2880 - (cam->h*cam->w) % 2880;
  value0=(float)0.;
  for (k=0;k<n;k++) {
    if (big_endian==0) {
      cars0=(char*)&value0;
      car=cars0[0];
      cars0[0]=cars0[1];
      cars0[1]=car;
    }
    fwrite(&value0,1,sizeof(short),f);
  }
  fclose(f);
}

// -- This is the start point of the program
int main(int argc, char* argv[])
{

   // -- Decode la ligne de commandes. Exemple : tp_ccd.exe full_frame 0.5 1
   char method[1024];
   char filename[1024];
   float exptime = 0.5;
   int binning = 1;
   strcpy(method,"full_frame");
   if(argc < 4)
```

```
 {
    printf("ERR 1 : Not enough command line parameters\n: method exptime binning filename\n");
    return 1;
 } else {
    strcpy(method, argv[1]);
    exptime = atof(argv[2]);
    binning = atoi(argv[3]);
    strcpy(filename, argv[4]);
 }
 printf("ERR 0 : %s %f %d %s\n", method, exptime, binning, filename);

// -- Initialise la structure des donnees de la camera
struct camprop cam;
strcpy(cam.msg,"");
cam.exptime=exptime; // sec
cam.binx=binning;
cam.biny=binning;
cam.x1=1;
cam.y1=1;
cam.x2=768;
cam.y2=512;
cam.nb_photox=cam.x2-cam.x1+1;
cam.nb_photoy=cam.y2-cam.y1+1;
cam.celldimx=9e-6; // m
cam.celldimy=9e-6; // m
cam.overscanindex=0;
cam.nb_deadbeginphotox=14;
cam.nb_deadendphotox=14;
cam.nb_deadbeginphotoy=4;
cam.nb_deadendphotoy=4;
cam.p=NULL;

// -- Initialise le GPIO
if (!bcm2835_init()) {
   printf("Probleme bcm2835_init");
   return 1;
}

// -- Set the pin to be an output
bcm2835_gpio_fsel(PHI_V1, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(PHI_V2, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(PHI_H1, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(PHI_R,  BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(PHI_CL, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(PHI_SB, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(PHI_SC, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(PHI_SN, BCM2835_GPIO_FSEL_OUTP);
bcm2835_gpio_fsel(DELAY,  BCM2835_GPIO_FSEL_OUTP); // special delay us
```

```
// -- Set the pin to be an input
bcm2835_gpio_fsel(BIT_0, BCM2835_GPIO_FSEL_INPT);
bcm2835_gpio_fsel(BIT_1, BCM2835_GPIO_FSEL_INPT);
bcm2835_gpio_fsel(BIT_2, BCM2835_GPIO_FSEL_INPT);
bcm2835_gpio_fsel(BIT_3, BCM2835_GPIO_FSEL_INPT);

// -- Set default values for outputs
bcm2835_gpio_write (PHI_V1, LEVEL_LOW) ;
bcm2835_gpio_write (PHI_V2, LEVEL_LOW) ;
bcm2835_gpio_write (PHI_H1, LEVEL_HIGH) ;
bcm2835_gpio_write (PHI_R,  LEVEL_LOW) ;
bcm2835_gpio_write (PHI_CL, LEVEL_LOW) ;
bcm2835_gpio_write (PHI_SB, LEVEL_LOW) ;
bcm2835_gpio_write (PHI_SC, LEVEL_LOW) ;
bcm2835_gpio_write (PHI_SN, LEVEL_LOW) ;

// -- Select the method to apply
if (strcmp(method, "full_frame")==0) {
    tp_acquisition_fullframe(&cam);
    tp_savefits(&cam, filename);
} else if (strcmp(method, "zi_zh")==0) {
   for (int i = 0; i <1; i++) {
     tp_zi_zh(&cam);
   }
} else if (strcmp(method, "fast_line")==0) {
   for (int i = 0; i <1; i++) {
     tp_fast_line(&cam);
   }
} else if (strcmp(method, "read_pel_fast")==0) {
   for (int i = 0; i <1; i++) {
     tp_read_pel_fast(&cam);
   }
} else if (strcmp(method, "read_pel_fast2")==0) {
   for (int i = 0; i <1; i++) {
     tp_read_pel_fast2(&cam);
   }
} else if (strcmp(method, "fast_vidage")==0) {
   for (int i = 0; i <1; i++) {
     tp_fast_vidage(&cam);
   }
} else if (strcmp(method, "square_signal")==0) {
    for (int i = 0; i < 100000; i++) {
        float tphiV=2.0;
        float tphiHS=1.0;
        // ---
        bcm2835_gpio_write (PHI_V1, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_V2, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_H1, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_R, LEVEL_HIGH) ;
```

```
        bcm2835_gpio_write (PHI_CL, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_SB, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_SC, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_SN, LEVEL_HIGH) ;
        tp_sleep(tphiV);
        bcm2835_gpio_write (PHI_V1, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_V2, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_H1, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_R, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_CL, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_SB, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_SC, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_SN, LEVEL_LOW) ;
        tp_sleep(tphiHS);
      }
  } else if (strcmp(method, "set_255")==0) {
        bcm2835_gpio_write (PHI_V1, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_V2, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_H1, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_R, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_CL, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_SB, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_SC, LEVEL_HIGH) ;
        bcm2835_gpio_write (PHI_SN, LEVEL_HIGH) ;
        tp_sleep(5e6);
  } else if (strcmp(method, "set_0")==0) {
        bcm2835_gpio_write (PHI_V1, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_V2, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_H1, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_R, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_CL, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_SB, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_SC, LEVEL_LOW) ;
        bcm2835_gpio_write (PHI_SN, LEVEL_LOW) ;
  }
  printf("=== Termine\n");
  return 0;
}
```