Trabajo Práctico Especial de Sistemas Operativos

FLEDS

Abramowicz, Pablo Federico
Goñi, Juan Ignacio
Pan, Matias Santiago

Instituto Tecnológico de Buenos Aires

Índice

Resumen

Este informe técnico presenta la implementación de un dispositido de leds y su respectivo controlador para el sistema operativo Minix 2,0,0. Se incluye las descripciones técnicas del dispositivo y las modificaciones realizadas a los códigos fuente para la vinculación con el controlador.
Keywords: *fleds, driver, hardware, interrupciones de hardware, IRQ, handler, serial.*

1. Introducción

Este proyecto surgió debido a la consigna de la cátedra de agregarle alguna funcionalidad a Minix 2,0,0. El primer proyecto que se nos ocurrió fue vincular Minix con GPS mediante el puerto serial, pero el hecho de que Minix no contase con un X imposibilitaba visualizar el mapa de forma coherente, agregando que la complejidad del dispositivo de GPS era parsear la telemetría entregaba. Luego de discutir, surgió la idea de armar un *cartel del leds*, de ahora en más **Fleds**, y a continuación se pasa a detallar los detalles del mismo.

2. Implementación

En principio se pensó en realizar la comunicación entre Minix y el **Fleds** compuesta por 2 capas: una donde el usuario pueda comunicarse y mandarle los comandos, y otra donde el driver se comunique con el fleds por medio del puerto serial. La misma sería:

Usuario → driver fleds → driver serial → fleds hardware

Luego de investigar y revisar el controlador provisto por minix del serial (`/usr/src/kernel/RS232c`) se llegó a la conclución de que era prácticamente inutilizable debido a la estrecha vinculación del mismo con la terminal (`/usr/src/kernel/ttyc`), a lo cual se procedió a realizar un único controlador que haga de ambos.

A su vez, se pensó en dejar la lógica del controlador del lado del usuario por un tema de espacio en la ROM del Hardware, pero satisfactoriamente se pudo incluir ciertas lógicas dentro del Hardware.

También se generó una librería de animaciones para una fácil utilización del mismo.

3. Hardware

Fleds consta de 128 leds rgb, con la capacidad de encenderse con los 3 colores primarios luz, rojo, azul y verde, independientemente. Consta de 16 columnas por 8 filas. Se utilizó un decodificador para la selección de la columna, 3 latches para el dibujado de los colores y un microcontrolador para el refresco de la pantalla. En la figura 1 se muestra el diagrama del circuito.

3.1. Principio de funcionamiento

En el microcontrolador tiene cargado el firmware, el cual se encarga de mantener la comunicación con el driver de Minix vía puerto serial. Cuando recibe un byte, se genera una

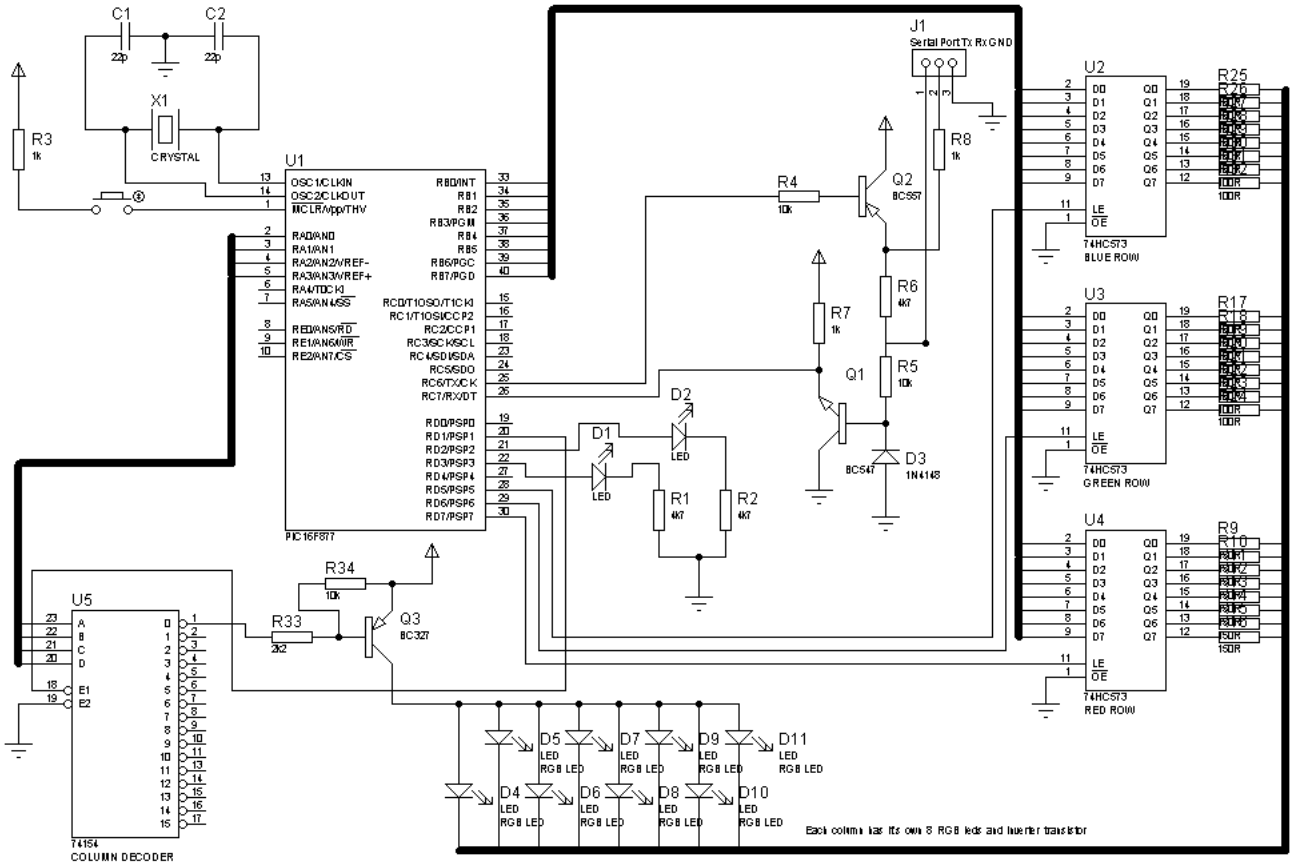

Figura 1: Diseño del circuito eléctrico de FLEDS

interrupción, la cual analiza según el protocolo el comando recibido y lo ejecuta. En estado normal, mantiene el refresco de la pantalla. Primero, le indica al decodificador que debe habilitar la primer columna, luego, se setean en cada latch, la columna del rojo, verde y azul. Se espera un milisegundo, y se proceden a dibujar las siguientes columnas, hasta llegar a la última, donde vuelve a comenzar el ciclo. El refresco es de 60Hz aproximadamente.

3.2. Protocolo

Se enumeran los comandos implementados en el firmware:

0x00: Clear screen. Borra la pantalla

0x40 al 0x4F: Set column. Establece la combinación de colores para la columna del nibble menos significativo (0 a 15). Recibe por parámetros los 3 bytes para el rojo, verde y azul, respectivamente.

0x10 al 0x19: Scroll. Realiza un scroll en base al valor del nibble menos significativo. 0 para ScrollDownCarry, 1 para ScrollDown, 2 para ScrollUpCarry, 3 para ScrollUp, 4

para ScrollLeftCarry, 5 para ScrollLeft, 6 para ScrollRightCarry, 7 para ScrollRight, 8 para StartScroll y 9 para StopScroll.

0x08 al 0x8F: Set UART speed. Establece la velocidad de la UART en base al nibble menos significativo. 0 para 1200, 1 para 2400, 2 para 4800, 3 para 9600, 4 para 19200, 5 para 38400, 6 para 57600 y 7 para 115200 baudios.

0x20 al 0x2F: Set scroll speed. Establece la frecuencia para el scroll.

0x5A: Get screen. Transmite por el serial, el contenido actual de la pantalla, enviando 96 caracteres encerrados entre [y], para su mejor lectura.

3.3. Código fuente

Se muestra a continuación el firmware del microcontrolador

main-pic.c

```
1  #define _VERSION_ "0.52"
2
3  #include <16F877.h>
4  #fuses HS,NOWDT,NOPROTECT,NOLVP,NOBROWNOUT,NOPUT
5  #use delay(clock=20000000, restart_wdt)
6  #use rs232(baud=115200, xmit=PIN_C6, rcv=PIN_C7)
7
8  /****** DEFINITIONS ******/
9  typedef enum {B1200 = 0, B2400, B4800, B9600, B19200,
10                               B38400, B57600, B115200} bouds;
11
12 #define SCROLL_NONE               -1
13 #define SCROLL_DOWN_CARRY        0
14 #define SCROLL_DOWN               1
15 #define SCROLL_UP_CARRY                2
16 #define SCROLL_UP                3
17 #define SCROLL_LEFT_CARRY        4
18 #define SCROLL_LEFT               5
19 #define SCROLL_RIGHT_CARRY       6
20 #define SCROLL_RIGHT               7
21 #define START_SCROLL               8
22 #define STOP_SCROLL               9
23
24 #define DISABLED                0
25 #define ENABLED                        (DISABLED + 1)
26
27 /* STATUS*/
28 #define STATUS_ENA               output_high(PIN_D3);
29 #define LINK_ENA                output_high(PIN_D2);
30 #define STATUS_DIS               output_low(PIN_D3);
31 #define LINK_DIS                output_low(PIN_D2);
32
33 /* LEDS */
34 #define R_ENA                        output_high(PIN_D7);
35 #define G_ENA                        output_high(PIN_D6);
36 #define B_ENA                        output_high(PIN_D5);
```

```
37 #define COL_ENA                        output_low(PIN_D1);
38 #define R_DIS                         output_low(PIN_D7);
39 #define G_DIS                         output_low(PIN_D6);
40 #define B_DIS                         output_low(PIN_D5);
41 #define COL_DIS                         output_high(PIN_D1);
42 #define SET_74154_TIME                 1
43 #define SET_74573_TIME                 0
44 #define SCREEN_WIDTH                  16
45 #define COLOR_COUNT                   3
46 #define SCREEN                           (COLOR_COUNT * SCREEN_WIDTH)
47
48 /* TIMER */
49 // 76 && 256 -> 1Hz
50 #define DEFAULT_TIMER_COUNT        76
51 #define DEFAULT_TIMER_DIV         RTCC_DIV_256
52 #define DEFAULT_PING_COUNT         38 // ping every 500ms
53 #define PING_PONG_COUNT                50
54
55 /****** FUNCTIONS ******/
56
57 /* LEDS */
58 void clear(void);
59 void set_red(byte color);
60 void set_green(byte color);
61 void set_blue(byte color);
62 void set_col(byte col);
63 void get_col(void);
64 void scrollUp(void);
65 void scrollUpCarry(void);
66 void scrollDown(void);
67 void scrollDownCarry(void);
68 void scrollLeft(void);
69 void scrollLeftCarry(void);
70 void scrollRight(void);
71 void scrollRightCarry(void);
72 void getScreen(void);
73 void clearScreen(void);
74 void stopScroll(void);
75 void startScroll(void);
76 void setUARTSpeed(byte speed);
77 void setScrollFreq(byte freq);
78 void ping(void);
79 void pong(void);
80 void showSplashScreen(void);
81
82 /* TIMER */
83 byte matrix[SCREEN];
84
85 /****** IMPLEMENTATION ******/
86 // SCROLL CONTROL
87 byte timer_div = DEFAULT_TIMER_DIV;
88 byte timer_count = DEFAULT_TIMER_COUNT;
89 byte int_count = DEFAULT_TIMER_COUNT;
90 byte scroll = DISABLED;
91 byte scroll_type = SCROLL_NONE;
92 // PING CONTROL
93 byte int_ping_count = DEFAULT_PING_COUNT;
94 byte ping_count = DEFAULT_PING_COUNT;
95 byte do_reset = PING_PONG_COUNT;
96 // SERIAL CONTROL
```

```
97  byte actual_col = 0;
98  byte column = 0;
99  byte status = 0;
100
101 /* INTERRUPT */
102 #INT_RDA
103 void recv_rs232(void)
104 {
105         int recv = 0;
106
107         LINK_ENA
108         recv = getc();
109
110         // GET RED BYTE
111         if (status == 0x01)
112         {
113                 status = 0x02;
114                 matrix[column * 3] = recv;
115         }
116         // GET GREEN BYTE
117         else if (status == 0x02)
118         {
119                 status = 0x03;
120                 matrix[column * 3 + 1] = recv;
121         }
122         // GET BLUE BYTE
123         else if (status == 0x03)
124         {
125                 status = 0x00;
126                 matrix[column * 3 + 2] = recv;
127         }
128         // GET COMMAND
129         else if (status == 0)
130         {
131                 // SET COLUMN (0100 ABCD) + R + G + B
132                 if ((recv & 0xF0) == 0x40)
133                 {
134                         column = recv & 0x0F;
135                         status = 0x01;
136                 }
137                 // CLEAR SCREEN (0000 0000)
138                 if (recv == 0x00)
139                         clearScreen();
140                 // GET SCREEN (0101 1010)
141                 if (recv == 0x5A)
142                         getScreen();
143                 // PONG RECV (0111 1110)
144                 if (recv == 0x7E)
145                         pong();
146                 // SET UART SPEED (0000 1ABC)
147                 if ((recv & 0xF8) == 0x08)
148                         setUARTSpeed(recv & 0x07);
149                 // SET SCROLL FREQ (0010 ABCD)
150                 if ((recv & 0xF0) == 0x20)
151                         setScrollFreq(recv & 0x0F);
152                 // SCROLL (0001 ABCD)
153                 if ((recv & 0xF0) == 0x10)
154                 {
155                         switch (recv & 0x0F)
156                         {
```

```
157                                 case SCROLL_DOWN_CARRY:
158                                         scroll_type = SCROLL_DOWN_CARRY;
159                                         startScroll();
160                                         break;
161                                 case SCROLL_DOWN:
162                                         scroll_type = SCROLL_DOWN;
163                                         startScroll();
164                                         break;
165                                 case SCROLL_UP_CARRY:
166                                         scroll_type = SCROLL_UP_CARRY;
167                                         startScroll();
168                                         break;
169                                 case SCROLL_UP:
170                                         scroll_type = SCROLL_UP;
171                                         startScroll();
172                                         break;
173                                 case SCROLL_LEFT_CARRY:
174                                         scroll_type = SCROLL_LEFT_CARRY;
175                                         startScroll();
176                                         break;
177                                 case SCROLL_LEFT:
178                                         scroll_type = SCROLL_LEFT;
179                                         startScroll();
180                                         break;
181                                 case SCROLL_RIGHT_CARRY:
182                                         scroll_type = SCROLL_RIGHT_CARRY;
183                                         startScroll();
184                                         break;
185                                 case SCROLL_RIGHT:
186                                         scroll_type = SCROLL_RIGHT;
187                                         startScroll();
188                                         break;
189                                 case START_SCROLL:
190                                         startScroll();
191                                         break;
192                                 case STOP_SCROLL:
193                                         stopScroll();
194                                         break;
195                                 default:
196                                         stopScroll();
197                                         break;
198                         }
199                 }
200         }
201         return;
202 }
203
204 #INT_RTCC
205 void clock(void)
206 {
207         LINK_DIS
208 /*
209         if(0 == (--int_ping_count))
210         {
211                 ping();
212                 int_ping_count = ping_count;
213                 if (0 == do_reset--)
214                 {
215                         printf("\r\n# Ping timeout - check the drivers #\r\n");
216                         reset_cpu();
```

```
217                 }
218         }
219 */
220
221         if((scroll == ENABLED) && (0 == (--int_count)))
222         {
223                 switch (scroll_type)
224                 {
225                         case SCROLL_RIGHT:
226                                 scrollRight();
227                                 break;
228                         case SCROLL_LEFT:
229                                 scrollLeft();
230                                 break;
231                         case SCROLL_UP:
232                                 scrollUp();
233                                 break;
234                         case SCROLL_DOWN:
235                                 scrollDown();
236                                 break;
237                         case SCROLL_RIGHT_CARRY:
238                                 scrollRightCarry();
239                                 break;
240                         case SCROLL_LEFT_CARRY:
241                                 scrollLeftCarry();
242                                 break;
243                         case SCROLL_UP_CARRY:
244                                 scrollUpCarry();
245                                 break;
246                         case SCROLL_DOWN_CARRY:
247                                 scrollDownCarry();
248                                 break;
249                         default:
250                                 stopScroll();
251                                 break;
252                 }
253                 int_count = timer_count;
254         }
255 }
256
257 void showScreen(void)
258 {
259         int i;
260         actual_col = 0;
261         for (i = 0; i < SCREEN; i+=3)
262         {
263                 COL_DIS
264                 set_red(matrix[i]);
265                 delay_ms(SET_74573_TIME);
266                 set_green(matrix[i+1]);
267                 delay_ms(SET_74573_TIME);
268                 set_blue(matrix[i+2]);
269                 delay_ms(SET_74573_TIME);
270                 set_col(actual_col++);
271         }
272         return;
273 }
274
275 /* MAIN */
276 void main(void)
```

```
277 {
278         int i;
279         unsigned byte alive = 0;
280 
281         set_tris_a(0x00);
282         set_tris_b(0x00);
283         set_tris_d(0x00);
284 
285         output_a(0x00);
286         output_b(0xFF);
287         output_d(0x00);
288 
289         for (i = 0; i < SCREEN; i++)
290                 matrix[i] = 0;
291 
292         status = 0; //means waiting for a command
293         // scroll?
294         scroll = DISABLED;
295         scroll_type = SCROLL_NONE;
296         // watch dog
297         do_reset = PING_PONG_COUNT;
298         ping_count = DEFAULT_PING_COUNT;
299         // reset timer0
300         set_rtcc(0);
301         timer_div = RTCC_DIV_256;
302         setup_counters (RTCC_INTERNAL, timer_div);
303         //timer interrupt
304         enable_interrupts (INT_RTCC);
305         //serial interrupt
306         enable_interrupts(INT_RDA);
307         //global interrupt
308         enable_interrupts(GLOBAL);
309 
310         printf("\r\nInitializing Fleds (ver %s)\r\n", _VERSION_);
311 
312         clear();
313 
314         showSplashScreen();
315 
316         set_col(0);
317         while(1)
318         {
319                 showScreen();
320                 // Red led -> Alive!
321                 if (0 == alive)
322                         STATUS_ENA
323                 if (128 == alive++)
324                         STATUS_DIS
325         }
326 
327          return;
328 }
329 
330 /* LEDS */
331 void set_col(byte col)
332 {
333         if (col < 16)
334         {
335                 output_a(col);
336                 COL_ENA
```

```
337                 delay_ms(SET_74154_TIME);
338                 COL_DIS
339         }
340         return;
341 }
342
343 void get_col(void)
344 {
345         putchar('<');
346         putchar(actual_col);
347         putchar('>');
348
349         return;
350 }
351
352 void clear(void)
353 {
354         R_ENA
355         G_ENA
356         B_ENA
357         output_b(0xFF);
358         delay_ms(SET_74573_TIME);
359         R_DIS
360         G_DIS
361         B_DIS
362         return;
363 }
364
365 void set_red(byte color)
366 {
367         G_DIS
368         B_DIS
369         output_b(0xFF - color);
370         R_ENA
371         delay_ms(SET_74573_TIME);
372         R_DIS
373         return;
374 }
375
376 void set_green(byte color)
377 {
378         G_DIS
379         B_DIS
380         output_b(0xFF - color);
381         G_ENA
382         delay_ms(SET_74573_TIME);
383         G_DIS
384         return;
385 }
386
387 void set_blue(byte color)
388 {
389         G_DIS
390         B_DIS
391         output_b(0xFF - color);
392         B_ENA
393         delay_ms(SET_74573_TIME);
394         B_DIS
395         return;
396 }
```

```
397
398 void scrollUpCarry(void)
399 {
400         byte i = 0;
401         for(i = 0; i < SCREEN; i++)
402                 rotate_right(matrix + i, 1);
403
404         return;
405 }
406
407 void scrollUp(void)
408 {
409         byte i = 0;
410         for(i = 0; i < SCREEN; i++)
411                 shift_right(matrix + i, 1, 0);
412
413         return;
414 }
415
416 void scrollDownCarry(void)
417 {
418         byte i = 0;
419         for(i = 0; i < SCREEN; i++)
420                 rotate_left(matrix + i, 1);
421
422         return;
423 }
424
425 void scrollDown(void)
426 {
427         byte i = 0;
428         for(i = 0; i < SCREEN; i++)
429                 shift_left(matrix + i, 1, 0);
430
431         return;
432 }
433
434 void scrollLeft(void)
435 {
436         byte i = 0;
437         for(i = 0; i < (SCREEN - 3); i++)
438         {
439                 matrix[i] = matrix[i + 3];
440         }
441         // clear last column
442         matrix[i] = 0;
443         matrix[i+1] = 0;
444         matrix[i+2] = 0;
445
446         return;
447 }
448
449 void scrollLeftCarry(void)
450 {
451         byte i = 0, r, g, b;
452
453         // Backup first column
454         r = matrix[0];
455         g = matrix[1];
456         b = matrix[2];
```

```
457
458         for(i = 0; i < (SCREEN - 3); i++)
459         {
460                 matrix[i] = matrix[i + 3];
461         }
462         // set last column
463         matrix[i] = r;
464         matrix[i+1] = g;
465         matrix[i+2] = b;
466
467         return;
468 }
469
470 void scrollRight(void)
471 {
472         byte i = SCREEN-1;
473
474         for(i = SCREEN-1; i >= 3; i--)
475         {
476                 matrix[i] = matrix[i - 3];
477         }
478         // set last column
479         matrix[0] = 0;
480         matrix[1] = 0;
481         matrix[2] = 0;
482
483         return;
484 }
485
486 void scrollRightCarry(void)
487 {
488         byte i = SCREEN-1, r, g, b;
489
490         // Backup first column
491         r = matrix[i-2];
492         g = matrix[i-1];
493         b = matrix[i];
494
495         for(i = SCREEN-1; i >= 3; i--)
496         {
497                 matrix[i] = matrix[i - 3];
498         }
499         // set last column
500         matrix[0] = r;
501         matrix[1] = g;
502         matrix[2] = b;
503
504         return;
505 }
506
507 void clearScreen(void)
508 {
509         byte i = 0;
510
511         for(i = 0; i < SCREEN; i++)
512         {
513                 matrix[i] = 0;
514         }
515         return;
516 }
```

```
517
518 void getScreen(void)
519 {
520         byte i = 0;
521
522         putchar('[');
523         for(i = 0; i < SCREEN; i++)
524         {
525                 putchar(matrix[i]);
526         }
527         putchar(']');
528
529         return;
530 }
531
532 void stopScroll(void)
533 {
534         scroll = DISABLED;
535         return;
536 }
537
538 void startScroll(void)
539 {
540         disable_interrupts(GLOBAL);
541         scroll = ENABLED;
542         set_rtcc(0);
543         setup_counters (RTCC_INTERNAL, timer_div);
544         enable_interrupts(INT_RTCC);
545         enable_interrupts(GLOBAL);
546         return;
547 }
548
549 void setUARTSpeed(byte speed)
550 {
551         switch(speed)
552         {
553                 case B1200:
554                         setup_uart(1200);
555                         break;
556                 case B2400:
557                         setup_uart(2400);
558                         break;
559                 case B4800:
560                         setup_uart(4800);
561                         break;
562                 case B9600:
563                         setup_uart(9600);
564                         break;
565                 case B19200:
566                         setup_uart(19200);
567                         break;
568                 case B38400:
569                         setup_uart(38400);
570                         break;
571                 case B57600:
572                         setup_uart(57600);
573                         break;
574                 case B115200:
575                         setup_uart(115200);
576                         break;
```

```
577                 default:
578                         setup_uart(4800);
579                         break;
580         }
581         return;
582 }
583
584 void setScrollFreq(byte freq)
585 {
586         //global interrupt
587         disable_interrupts(GLOBAL);
588         set_rtcc(0);
589
590         switch(freq)
591         {
592                 case 0:
593                         timer_div = RTCC_DIV_64;
594                         timer_count = 1;
595                         ping_count = 229;
596                         break;
597                 case 1:
598                         timer_div = RTCC_DIV_128;
599                         timer_count = 1;
600                         ping_count = 115;
601                         break;
602                 case 2:
603                         timer_div = RTCC_DIV_64;
604                         timer_count = 3;
605                         ping_count = 230;
606                         break;
607                 case 3:
608                         timer_div = RTCC_DIV_256;
609                         timer_count = 1;
610                         ping_count = 57;
611                         break;
612                 case 4:
613                         timer_div = RTCC_DIV_64;
614                         timer_count = 5;
615                         ping_count = 229;
616                         break;
617                 case 5:
618                         timer_div = RTCC_DIV_128;
619                         timer_count = 3;
620                         ping_count = 115;
621                         break;
622                 case 6:
623                         timer_div = RTCC_DIV_64;
624                         timer_count = 7;
625                         ping_count = 231;
626                         break;
627                 case 7:
628                         timer_div = RTCC_DIV_256;
629                         timer_count = 2;
630                         ping_count = 57;
631                         break;
632                 case 8:
633                         timer_div = RTCC_DIV_64;
634                         timer_count = 9;
635                         ping_count = 230;
636                         break;
```

```
637                 case 9:
638                         timer_div = RTCC_DIV_256;
639                         timer_count = 3;
640                         ping_count = 56;
641                         break;
642                 case 10:
643                         timer_div = RTCC_DIV_64;
644                         timer_count = 17;
645                         ping_count = 230;
646                         break;
647                 case 11:
648                         timer_div = RTCC_DIV_256;
649                         timer_count = 5;
650                         ping_count = 56;
651                         break;
652                 case 12:
653                         timer_div = RTCC_DIV_256;
654                         timer_count = 8;
655                         ping_count = 60;
656                         break;
657                 case 13:
658                         timer_div = RTCC_DIV_256;
659                         timer_count = 9;
660                         ping_count = 54;
661                         break;
662                 case 14:
663                         timer_div = RTCC_DIV_256;
664                         timer_count = 14;
665                         ping_count = 53;
666                         break;
667                 case 15:
668                         timer_div = RTCC_DIV_256;
669                         timer_count = 76;
670                         ping_count = 57;
671                         break;
672                 default:
673                         timer_div = DEFAULT_TIMER_DIV;
674                         timer_count = DEFAULT_TIMER_COUNT;
675                         ping_count = DEFAULT_PING_COUNT;
676                         break;
677         }
678
679         // Reset counters
680         int_count = timer_count;
681         int_ping_count = ping_count;
682
683         //global interrupt
684         enable_interrupts(GLOBAL);
685
686         return;
687 }
688
689 void ping(void)
690 {
691
692         putchar('?');
693
694         return;
695 }
696
```

```
697 void pong(void)
698 {
699         // do not do reset!
700         do_reset = ping_count;
701
702         return;
703 }
704
705 void showSplashScreen(void)
706 {
707         int i = 0;
708         byte splash[SCREEN] = {0x3F, 0x0, 0x0, 0x5, 0x0, 0x0, 0x1, 0x0, 0xF0, 0x0, 0x0, 0x80,
    0x0, 0x3E, 0x0, 0x0, 0x2A, 0x0, 0x0, 0x0, 0x0, 0xF0, 0xF0, 0x0, 0x90, 0x90, 0x0, 0x60, 0x60,
    0x0, 0x0, 0x4, 0x4, 0x0, 0x2A, 0x2A, 0x0, 0x2A, 0x2A, 0x0, 0x10, 0x10, 0x0, 0x0, 0x0, 0xBC,
    0x0, 0xBC};
709
710         for (i = 0; i < SCREEN; i++)
711                 matrix[i] = splash[i];
712
713         return;
714 }
```

4. Controlador

La implementación del driver del fleds involucró la modificación de códigos fuentes del kernel de Minix y el agregado de nuevos archivos. Se detalla a continuación las modificaciones correspondientes con sus respectivas explicaciones. *Aclaración: Los paths que encabezan las siguientes subsecciones son relativos a* usr*., y representan la locación del archivo en el file-system.*

4.1. include/minix/com.h

Este archivo contiene los números de tareas que involucra el kernel, a lo cuál se procedió a cambiar el número de tarea de la **TTY** como último y agregarle la tarea del **Fleds** para que el kernel la pueda manejar.

com-min.h

```
1 ...
2 #define TTY                (DL_ETH - 2)
3 /* FLEDS: agrego el pid del fleds */
4 #define FLEDS               (DL_ETH - 1)
5 ...                                /* terminal I/O class */
6
```

4.2. include/minix/config.h

En este archivo se definen configuraciones que se van a utilizar. Como se está agregando un nuevo dispositivo, se agrega la definición del **ENABLE** del mismo.

config-min1.h

```
1 ...
2 /* FLEDS: incluyo el enabale del fleds */
```

```
#define ENABLE_FLEDS           1
/* Include or exclude device drivers.  Set to 1 to include, 0 to exclude. */
#define ENABLE_NETWORKING  0        /* enable TCP/IP code */
#define ENABLE_AT_WINI     1        /* enable AT winchester driver */
#define ENABLE_BIOS_WINI   1        /* enable BIOS winchester driver */
#define ENABLE_ESDI_WINI   1        /* enable ESDI winchester driver */
#define ENABLE_XT_WINI     0        /* enable XT winchester driver */
#define ENABLE_ADAPTEC_SCSI 1        /* enable ADAPTEC SCSI driver */
#define ENABLE_MITSUMI_CDROM 0        /* enable Mitsumi CD-ROM driver */
#define ENABLE_SB_AUDIO    0        /* enable Soundblaster audio driver */
...
```

También se procedió a deshabilitar el serial utilizado por la TTY poniendo `NR_RS_LINES` en 0.

config-min2.h

```
...
/* NR_CONS, NR_RS_LINES, and NR_PTYS determine the number of terminals the
 * system can handle.
 */
#define NR_CONS            2        /* # system consoles (1 to 8) */
#define        NR_RS_LINES           0        /* # rs232 terminals (0, 1, or 2) */
#define        NR_PTYS                   0        /* # pseudo terminals (0 to 64) */
...
```

4.3. include/minix/const.h

El `NR_TASK` contiene la cantidad de tareas que corre inicialmeten el kernel, a lo cual, como se está agregando un controlador y se desea que esté activado todo el tiempo, agregamos la tarea

const-min.h

```
...
/* Number of tasks. */
#define NR_TASKS        (9 + ENABLE_WINI + ENABLE_SCSI + ENABLE_CDROM \
                        + ENABLE_NETWORKING + 2 * ENABLE_AUDIO + ENABLE_FLEDS)
...
```

4.4. src/kernel/table.c

En principio se tiene que agregar el tamaño de stack correspondiente al controlador. En `FLEDS_STACK` se define el mismo como un `SMALL_STACK`

table-min.c

```
...
#define SMALL_STACK        (128 * sizeof(char *))

#define        TTY_STACK        (3 * SMALL_STACK)
/* FLEDS: Agrego el stack del fleds */
#define FLEDS_STACK        (3 * SMALL_STACK)
...
```

También se tiene que agregar el mismo al tamaño total de stack definido en `TOT_STACK_SPACE`.

table-min2.c

```
...
/* FLEDS: Incluyo el stack al total */
#define        TOT_STACK_SPACE                (TTY_STACK + DP8390_STACK + SCSI_STACK + \
        SYN_ALRM_STACK + IDLE_STACK + HARDWARE_STACK + PRINTER_STACK + \
        WINCH_STACK + FLOP_STACK + MEM_STACK + CLOCK_STACK + SYS_STACK + \
        CDROM_STACK + AUDIO_STACK + MIXER_STACK + FLEDS_STACK)
...
```

El **struct tasktab tasktab**[] contiene el punto de entrada de las tareas que se ejecutan, a lo cual se debe agregar la tarea del controlador del **Fleds**. Para agregar una entrada se debe especificar primero la función encargada de atender los pedidos al controlador, en nuestro caso `fleds_task`, luego se define el stack correspondiente de la tarea, en nuestro caso `FLEDS_STACK`, y por último se define el nombre de la tarea a mostrar.

table-min3.c

```
...
PUBLIC struct tasktab tasktab[] = {
        { tty_task,                TTY_STACK,        "TTY"               },
/* FLEDS: Agrego la tarea al scheduler */
#if ENABLE_FLEDS
        { fleds_task,                FLEDS_STACK,        "FLEDS"               },
#endif
#if ENABLE_NETWORKING
        { dp8390_task,                DP8390_STACK,        "DP8390"        },
#endif
#if ENABLE_CDROM
        { cdrom_task,                CDROM_STACK,        "CDROM"               },
#endif
#if ENABLE_AUDIO
        { audio_task,                AUDIO_STACK,        "AUDIO"               },
        { mixer_task,                MIXER_STACK,        "MIXER"               },
#endif
#if ENABLE_SCSI
        { scsi_task,                SCSI_STACK,        "SCSI"               },
#endif
#if ENABLE_WINI
        { winchester_task,        WINCH_STACK,        "WINCH"               },
#endif
        { syn_alrm_task,        SYN_ALRM_STACK, "SYN_AL"        },
        { idle_task,                IDLE_STACK,        "IDLE"               },
        { printer_task,                PRINTER_STACK,        "PRINTER"        },
        { floppy_task,                FLOP_STACK,        "FLOPPY"        },
        { mem_task,                MEM_STACK,        "MEMORY"        },
        { clock_task,                CLOCK_STACK,        "CLOCK"               },
        { sys_task,                SYS_STACK,        "SYS"               },
        { 0,                        HARDWARE_STACK,        "HARDWAR"        },
        { 0,                        0,                "MM"                },
        { 0,                        0,                "FS"                },
#if ENABLE_NETWORKING
        { 0,                        0,                "INET"                },
#endif
        { 0,                        0,                "INIT"                },
};
...
```

Es importante que se agregue en conconrdancia al número de tarea incluido en `/include/minix/com`h. Entonces, como se incluyó en segundo lugar, aquí se hace de la misma manera.

4.5. src/fs/table.c

La tabla `dmap` determina la vinculación entre el número principal y la tarea que recibe el mensaje. En la misma se agregó en una posición libre, la 9, la tarea de Fleds.

fs-table-min.c

```
...
PUBLIC struct dmap dmap[] = {
/*  ?   Open       Read/Write   Close       Task #      Device  File
    -   ----       ----------   -----       -------     ------  ----       */
  DT(1, no_dev,    no_dev,      no_dev,     0)           /* 0 = not used   */
  DT(1, dev_opcl,  call_task,   dev_opcl,   MEM)         /* 1 = /dev/mem   */
  DT(1, dev_opcl,  call_task,   dev_opcl,   FLOPPY)      /* 2 = /dev/fd0   */
  DT(ENABLE_WINI,
        dev_opcl,  call_task,   dev_opcl,   WINCHESTER)  /* 3 = /dev/hd0   */
  DT(1, tty_open,  call_task,   dev_opcl,   TTY)         /* 4 = /dev/tty00 */
  DT(1, ctty_open, call_ctty,   ctty_close, TTY)         /* 5 = /dev/tty   */
  DT(1, dev_opcl,  call_task,   dev_opcl,    PRINTER)     /* 6 = /dev/lp    */

#if (MACHINE == IBM_PC)
  DT(ENABLE_NETWORKING,
        net_open,  call_task,   dev_opcl,   INET_PROC_NR)/* 7 = /dev/ip    */
  DT(ENABLE_CDROM,
        dev_opcl,  call_task,   dev_opcl,   CDROM)       /* 8 = /dev/cd0   */
/* FLEDS: agrego el dispositivo del fleds al mapeo */
  DT(1, dev_opcl,  call_task,   dev_opcl,   FLEDS)           /* 9 = /dev/fleds   */
  DT(ENABLE_SCSI,
        dev_opcl,  call_task,   dev_opcl,   SCSI)        /*10 = /dev/sd0   */
  DT(0, 0,         0,           0,          0)           /*11 = not used   */
  DT(0, 0,         0,           0,          0)           /*12 = not used   */
  DT(ENABLE_AUDIO,
        dev_opcl,  call_task,   dev_opcl,   AUDIO)       /*13 = /dev/audio */
  DT(ENABLE_AUDIO,
        dev_opcl,  call_task,   dev_opcl,   MIXER)       /*14 = /dev/mixer */
#endif /* IBM_PC */

#if (MACHINE == ATARI)
  DT(ENABLE_SCSI,
        dev_opcl,  call_task,   dev_opcl,   SCSI)        /* 7 = /dev/hdscsi0 */
#endif
};
...
```

4.6. src/kernel/fleds.c

Este archivo contiene realmente el controlador del dispositivo. Contiene una única función pública `fleds_task()` que es el proceso controlador y la comunicación con el mismo se hace mediante las primitivas `open`, `close`, `read` y `write`. Inicialmente, cuando arranca el controlador inicializa el serial debido que ya no lo hace más la tty. En este proceso se involucra los seteos de la velocidad, paridad, interrupción, buffers internos, etc. Luego la tarea queda esperando a recibir algún mensaje. Una de las complejidades surgidas fue el tema de los

direccionamientos que utiliza minix en modo kernel y en modo usuario y el requerimento de copiar los datos entre los distintos segmentos (en el caso de `write` y `read`).

fleds.c

```
1  #include "kernel.h"
2  #include <termios.h>
3  #include <sys/ioctl.h>
4  #include <minix/callnr.h>
5  #include <minix/com.h>
6  #include <sys/types.h>
7  #include "proc.h"
8  #include "fleds.h"
9
10 FORWARD _PROTOTYPE( void send_serial, (int data)               );
11 FORWARD _PROTOTYPE( void do_open, (void) );
12 FORWARD _PROTOTYPE( void do_close, (void) );
13 FORWARD _PROTOTYPE( void do_write, (message *m_ptr) );
14 FORWARD _PROTOTYPE( void do_read, (message *m_ptr) );
15 FORWARD _PROTOTYPE( void fleds_init, (void) );
16 FORWARD _PROTOTYPE( void reply, (int code, int replyee, int process,
17                 int status) );
18
19 FORWARD _PROTOTYPE( int setParity, (int parity, int odd) );
20 FORWARD _PROTOTYPE( int setInterrupt, (int on) );
21 FORWARD _PROTOTYPE( int setMCR, (void) );
22 FORWARD _PROTOTYPE( int setbps, (serialSpeedT speed) );
23 FORWARD _PROTOTYPE( int setStop, (int stopBit)  );
24 FORWARD _PROTOTYPE( int setBitCount, (int bitCount) );
25 FORWARD _PROTOTYPE( int doReset, (void) );
26 FORWARD _PROTOTYPE( int fleds_handler, (int irq) );
27 FORWARD _PROTOTYPE( int add_buffer,(int data) );
28 FORWARD _PROTOTYPE( char get_buffer,(void) );
29 FORWARD _PROTOTYPE( int bufferNotFull, (void) );
30 FORWARD _PROTOTYPE( int bufferNotEmpty, (void) );
31
32 typedef struct {
33         unsigned char LCR;
34         unsigned char DLL;
35         unsigned char DLM;
36         unsigned char MCR;
37         unsigned char IER;
38 } serialStateT;
39
40 PRIVATE serialStateT serialState;
41
42 PRIVATE int initialized = 0;
43
44 PRIVATE int commBase = COMM1_BASE;
45
46 typedef struct
47 {
48         int buffer[FLEDS_BUFFER_SIZE];
49         int write;
50         int read;
51         int qty;
52 }fledsBufferT;
53
54 fledsBufferT k_buffer;
55
56 PRIVATE int bufferNotEmpty () {
```

```
57          return (k_buffer.qty != 0);
58  }
59
60  PRIVATE int bufferNotFull () {
61          return (k_buffer.qty < FLEDS_BUFFER_SIZE);
62  }
63
64
65  PUBLIC void fleds_task()
66  {
67          message mess;
68          int status;
69
70          /* inicializo el fleds */
71          fleds_init();
72
73          while(1)
74          {
75                  receive(ANY, &mess);
76                  switch(mess.m_type)
77                  {
78                          case DEV_OPEN:
79                                  do_open();
80                                  reply(TASK_REPLY, mess.m_source, mess.PROC_NR, OK);
81                                  break;
82                          case DEV_WRITE:
83                                  do_write(&mess);
84                                  break;
85                          case DEV_READ:
86                                  do_read(&mess);
87                                  break;
88                          case DEV_CLOSE:
89                                  do_close();
90                                  reply(TASK_REPLY, mess.m_source, mess.PROC_NR, OK);
91                                  break;
92                          case DEV_IOCTL:
93                                  reply(TASK_REPLY, mess.m_source, mess.PROC_NR, OK);
94                                  break;
95                          case HARD_INT:                                  break;
96                  }
97
98          }
99
100 }
101
102 /*===========================================================================*
103  *                                do_write                                   *
104  *===========================================================================*/
105
106 PRIVATE void do_write(m_ptr)
107 register message *m_ptr;        /* pointer to the newly arrived message */
108 {
109
110         register int r;
111         int i, count=0;
112         char obuf[FLEDS_BUFFER_SIZE];
113         phys_bytes user_phys;
114
115         /* Reject command if last write is not finished, count not positive, or
116         * user address bad.
```

```
117         */
118
119         if (m_ptr->COUNT <= 0)
120         {
121                 r = EINVAL;
122         }
123         else
124                 if (numap(m_ptr->PROC_NR, (vir_bytes) m_ptr->ADDRESS, m_ptr->COUNT) == 0)
125                 {
126                         r = EFAULT;
127                 }
128                 else
129                 {
130                         #if FLEDS_PING_ENABLE
131                         disable_irq(RS232_IRQ);
132                         #endif
133                         /* cantidad de bytes a copiar */
134                         count = m_ptr->COUNT;
135                         /* obtengo  la direccion fisica del puntero enviado */
136                         user_phys = proc_vir2phys(proc_addr(m_ptr->PROC_NR), (vir_bytes)
    m_ptr->ADDRESS);
137                         /* copio los datos a una posicion local */
138                         phys_copy(user_phys, vir2phys(obuf), (phys_bytes) count);
139                         /* envio al serial */
140                         for (i=0; i < count; i++)
141                                 send_serial(obuf[i]);
142                         #if FLEDS_PING_ENABLE
143                         enable_irq(RS232_IRQ);
144                         #endif
145                         r = OK;
146                 }
147
148         reply(TASK_REPLY, m_ptr->m_source, m_ptr->PROC_NR,r);
149 }
150
151
152 /*===========================================================================*
153  *                                do_read                                    *
154  *===========================================================================*/
155
156 PRIVATE void do_read(m_ptr)
157 register message *m_ptr;        /* pointer to the newly arrived message */
158 {
159
160         register int r;
161         int i, count=0;
162         char orBuf[FLEDS_BUFFER_SIZE];
163         phys_bytes user_phys;
164
165         /* Reject command if last write is not finished, count not positive, or
166         * user address bad.
167         */
168
169         if (m_ptr->COUNT <= 0)
170         {
171                 r = EINVAL;
172         }
173         else
174                 if (numap(m_ptr->PROC_NR, (vir_bytes) m_ptr->ADDRESS, m_ptr->COUNT) == 0)
175                 {
```

```
176                         r = EFAULT;
177                 }
178                 else
179                 {
180                         #if FLEDS_PING_ENABLE
181                         disable_irq(RS232_IRQ);
182                         #endif
183                         /* cantidad de bytes a copiar */
184                         count = m_ptr->COUNT;
185                         /* obtengo  la direccion fisica del puntero enviado */
186                         user_phys = proc_vir2phys(proc_addr(m_ptr->PROC_NR), (vir_bytes)
    m_ptr->ADDRESS);
187                         /* copio  los datos del buffer a una variable local (por que el
    buffer es ciclico */
188                         for (i=0; i < count; i++)
189                                 orBuf[i]=get_buffer();
190                         /* copio los datos a la posicion fisica del usuario */
191                         phys_copy(vir2phys(orBuf), user_phys, (phys_bytes) count);
192                         #if FLEDS_PING_ENABLE
193                         enable_irq(RS232_IRQ);
194                         #endif
195                         r = OK;
196                 }
197
198         reply(TASK_REPLY, m_ptr->m_source, m_ptr->PROC_NR,r);
199 }
200
201
202 /*===========================================================================*
203  *                                setInterrupt                                 *
204  *===========================================================================*/
205
206 PRIVATE int setInterrupt(on)
207 int on;
208 {
209         unsigned char ie = 0;
210
211         if (on)
212                 ie = 0x01;
213
214         out_byte(commBase + 3, serialState.LCR & 0x7F);
215         out_byte(commBase + 1, ie);
216         out_byte(commBase + 3, serialState.LCR);
217
218         return 1;
219 }
220 /*===========================================================================*
221  *                                setParity                                  *
222  *===========================================================================*/
223
224 PRIVATE int setParity(parity, odd)
225 int parity;
226 int odd;
227 {
228         unsigned char p = 0;
229
230         if (parity)
231         {
232                 p |= 0x08;
233                 if (!odd)
```

```
234                         p |= 0x10;
235         }
236
237         serialState.LCR = (serialState.LCR & 0xE7) | p;
238
239         out_byte(commBase + 3, serialState.LCR);
240
241         return 1;
242 }
243
244 /*=========================================================================*
245  *                                setMCR                                   *
246  *=========================================================================*/
247
248 PRIVATE int setMCR()
249 {
250         out_byte(commBase + 4, 0x08);
251         return 1;
252 }
253
254 /*=========================================================================*
255  *                                setbps                                   *
256  *=========================================================================*/
257
258 PRIVATE int setbps(speed)
259 serialSpeedT speed;
260 {
261         char speedReg[][2] = { {0x00, 0x09}, {0x80, 0x01}, {0x60, 0x00}, {0x30, 0x00},
262         {0x18, 0x00}, {0x0C, 0x00}, {0x06, 0x00}, {0x03, 0x00}, {0x02, 0x00}, {0x01, 0x00}};
263
264         out_byte(commBase + 3, serialState.LCR | 0x80);
265         out_byte(commBase + 0, serialState.DLL = speedReg[speed][0]);
266         out_byte(commBase + 1, serialState.DLM = speedReg[speed][1]);
267         out_byte(commBase + 3, serialState.LCR);
268
269         return 1;
270 }
271
272 /*=========================================================================*
273  *                                setStop                                  *
274  *=========================================================================*/
275
276 PRIVATE int setStop(stopBit)
277 int stopBit;
278 {
279         unsigned char s = 0;
280
281         if (stopBit)
282                 s = 0x04;
283
284         serialState.LCR = (serialState.LCR & 0xFB) | s;
285
286         out_byte(commBase + 3, serialState.LCR);
287
288         return 1;
289 }
290
291 /*=========================================================================*
292  *                                setBitCount                              *
293  *=========================================================================*/
```

```
294
295 PRIVATE int setBitCount(bitCount)
296 int bitCount;
297 {
298         if (bitCount >= 5 && bitCount <= 8)
299                 bitCount -= 5;
300
301         serialState.LCR = (serialState.LCR & 0xFC) | bitCount;
302
303         out_byte(commBase + 3, serialState.LCR);
304
305         return 1;
306 }
307
308 PRIVATE int doReset()
309 {
310         out_byte(commBase + 3, serialState.LCR & 0x7F);
311         out_byte(commBase + 2, 0x06);
312         out_byte(commBase + 3, serialState.LCR);
313
314         return 1;
315 }
316
317
318
319 /*===========================================================================*
320  *                                fleds_init                                 *
321  *===========================================================================*/
322
323 PRIVATE void fleds_init()
324 {
325         doReset();
326
327         /*  inicializo las variables a usar */
328         serialState.LCR = 0;
329         serialState.DLL = 0;
330         serialState.DLM = 0;
331         serialState.MCR = 0;
332         serialState.IER = 0;
333         /* Inicializo las variables del buffer */
334         k_buffer.write = 0;
335         k_buffer.read = 0;
336         k_buffer.qty = 0;
337
338
339         setInterrupt(SERIAL_INT_E_OFF);
340         setParity(SERIAL_PARITY_OFF,SERIAL_PARITY_ODD);
341         setMCR();
342         setbps(SERIAL_S_115200);
343         setStop(SERIAL_STOP_1);
344         setBitCount(8);
345
346         /* si esta habilitado el ping lo uso */
347         #if FLEDS_PING_ENABLE
348         put_irq_handler(RS232_IRQ, fleds_handler);
349         enable_irq(RS232_IRQ);
350         #endif
351
352         doReset();
353
```

```
354         setInterrupt(SERIAL_INT_E_ON);
355
356 }
357
358
359
360 /*=========================================================================*
361  *                              add_buffer                                 *
362  *=========================================================================*/
363
364 PRIVATE int add_buffer(data)
365 int data;
366 {
367         if( bufferNotFull() ) {
368                 k_buffer.buffer[k_buffer.write] = data;
369                 k_buffer.qty++;
370                 if ( ++(k_buffer.write) == FLEDS_BUFFER_SIZE)
371                         k_buffer.write = 0;
372                 return 1;
373         }
374         return 0;
375 }
376
377 /*=========================================================================*
378  *                              get_buffer                                 *
379  *=========================================================================*/
380
381 PRIVATE char get_buffer()
382 {
383         char data;
384
385         if ( bufferNotEmpty() ) {
386                 k_buffer.qty--;
387                 data = k_buffer.buffer[k_buffer.read];
388                 if ( ++(k_buffer.read) == FLEDS_BUFFER_SIZE)
389                                         k_buffer.read = 0;
390                 return data;
391         }
392         return 0;
393 }
394
395
396 /*=========================================================================*
397  *                              fleds_handler                                  *
398  *=========================================================================*/
399
400 PRIVATE int fleds_handler(irq)
401 int irq;
402 {
403         char data;
404
405         switch (data = in_byte(commBase))
406         {
407                 #if FLEDS_PING_ENABLE
408                 case FLEDS_PING:
409                         send_serial(FLEDS_PONG);
410                         break;
411                 #endif
412                 default:
413                         add_buffer(data);
```

```
414         }
415         return(1);        /* reenable serial interrupt */
416 }
417
418
419 /*==========================================================================*
420  *                                do_open                                   *
421  *==========================================================================*/
422
423 PRIVATE void do_open()
424 {
425         return;
426 }
427
428 /*==========================================================================*
429  *                                do_close                                  *
430  *==========================================================================*/
431
432 PRIVATE void do_close()
433 {
434         return;
435 }
436
437
438 /*==========================================================================*
439  *                                reply                                     *
440  *==========================================================================*/
441
442 PRIVATE void reply(code, replyee, process, status)
443 int code;                         /* TASK_REPLY or REVIVE */
444 int replyee;                        /* destination for message (normally FS) */
445 int process;                        /* which user requested the printing */
446 int status;                        /* number of  chars printed or error code */
447 {
448 /* Send a reply telling FS that printing has started or stopped. */
449
450   message pr_mess;
451
452   pr_mess.m_type = code;        /* TASK_REPLY or REVIVE */
453   pr_mess.REP_STATUS = status;        /* count or EIO */
454   pr_mess.REP_PROC_NR = process;        /* which user does this pertain to */
455   if ((status = send(replyee, &pr_mess)) != OK)
456         panic("reply failed, status\n", status);
457 }
458
459
460 /*==========================================================================*
461  *                                send_serial                               *
462  *==========================================================================*/
463
464 PRIVATE void send_serial(data)
465 int data;
466 {
467         out_byte(commBase, data);
468 }
469
```

4.7. src/kernel/fleds.h

Este archivo contiene la configuración del Fleds.

fleds.h

```
1  #define COMM1_BASE                0x3F8
2  #define COMM2_BASE                0x2F8
3
4  #define SERIAL_INT_E_ON                0x01
5  #define SERIAL_INT_E_OFF        0x00
6
7  #define SERIAL_PARITY_ON        0x01
8  #define SERIAL_PARITY_OFF        0x00
9
10 #define SERIAL_PARITY_ODD        0x00
11 #define SERIAL_PARITY_EVEN        0x01
12
13 #define SERIAL_STOP_1                0x00
14 #define SERIAL_STOP_2                0x01
15
16 #define FLEDS_PING_ENABLE        0
17 #define FLEDS_PING                '?'
18 #define FLEDS_PONG                '~'
19
20 #define FLEDS_BUFFER_SIZE        128
21
22 typedef enum{SERIAL_S_50, SERIAL_S_300,SERIAL_S_1200, SERIAL_S_2400, SERIAL_S_4800,
   SERIAL_S_9600, SERIAL_S_19200, SERIAL_S_38400, SERIAL_S_57600, SERIAL_S_115200} serialSpeedT;
23
24 PUBLIC _PROTOTYPE( void fleds_task, (void) );
```

5. Aplicación

A modo de utilización del Fleds se implementó una librería libFleds.c destinada a facilitarle el uso del mismo al usuario, la misma incluye las siguientes funciones:

5.1. fledsADT newFleds (int height, int width, char * device)

Crea una nueva instacia del Fleds.

5.2. int freeFleds (fledsADT fleds)

Libera la instancia del Fleds.

5.3. int loadText (fledsADT fleds, char * text, int x, int y)

Carga un texto dado en el Fleds dadas las coordenadas x, y.

5.4. int loadPic (fledsADT fleds, pic_t pic, int x, int y)

Carga una imagen dado un pic_t en las coordenadas dadas.

5.5. int loadMovie (fledsADT fleds, pic_t * pic)

Carga una película en un vector de pic_t.

5.6. int clear (fledsADT fleds)

Limpia la pantalla del Fleds.

5.7. int show (fledsADT fleds)

Activa la imagen que tiene cargado el Fleds. Prende los leds.

5.8. int hide (fledsADT fleds)

Oculta lo que está mostrando el Fleds. Apaga los leds.

5.9. int animate (fledsADT fleds, animation_t animation, int iterations, int speed)

Anima los datos cargados en el Fleds con un tipo de animación. Existen diferentes tipos de animación, los cuales son:

5.9.1. SCROLL_NONE

No hace ningún tipo de scroll.

5.9.2. SCROLL_RIGHT

Scroll hacia la derecha y borra el contenido.

5.9.3. SCROLL_LEFT

Scroll hacia la izquierda y borra el contenido.

5.9.4. SCROLL_UP

Scroll hacia arriba y borra el contenido.

5.9.5. SCROLL_DOWN

Scroll hacia abajo y borra el contenido.

5.9.6. SCROLL_RIGHT_CARRY

Scroll hacia la derecha y hace cíclico el muestreo.

5.9.7. SCROLL_LEFT_CARRY

Scroll hacia la izquierda y hace cíclico el muestreo.

5.9.8. SCROLL_UP_CARRY

Scroll hacia arriba y hace cíclico el muestreo.

5.9.9. SCROLL_DOWN_CARRY

Scroll hacia abajo y hace cíclico el muestreo.

5.9.10. SCROLL_ROW_LEFT

Scrolea fila por fila hacia la izquierda limpiando la pantalla.

5.9.11. SCROLL_ROW_RIGHT

Scrolea fila por fila hacia la derecha limpiando la pantalla.

5.9.12. SCROLL_COLUMN_DOWN

Scrolea columna por columna hacia abajo limpiando la pantalla.

5.9.13. WAVE

Hace *la ola* con la imagen puesta en el fleds.

5.9.14. TWINKLE

Parpadea el contenido de la pantalla.

5.9.15. LSD

Tira animaciones re-locas.

5.9.16. MOVIE

Anima la pelicula.

5.9.17. CAMEL

Genera el efecto camel que tanto se comenta entre los jóvenes.

5.10. libFleds.h

libFleds.h

```
#define        PIC_WIDTH        16
#define FLEDS_DEVICE        "/dev/fleds"
#define USLEEP_TIME        100

typedef enum {SCROLL_NONE = 0, SCROLL_RIGHT, SCROLL_LEFT, SCROLL_UP, SCROLL_DOWN,
        SCROLL_RIGHT_CARRY,  SCROLL_LEFT_CARRY, SCROLL_UP_CARRY,  SCROLL_DOWN_CARRY,
         SCROLL_ROW_LEFT, SCROLL_COLUMN_DOWN, SCROLL_ROW_RIGHT, SCROLL_COLUMN_UP,
         WAVE, TWINKLE, LIFE, LSD, CAMEL, MOVIE} animation_t;

typedef struct column
{
        char id;
        char red;
        char green;
        char blue;
} column_t;

typedef struct pic
{
        column_t cols[PIC_WIDTH];
} pic_t;

typedef struct fledsCDT * fledsADT;

fledsADT newFleds (int height, int width, char * device);

int freeFleds (fledsADT fleds);

int loadText (fledsADT fleds, char * text, int x, int y);

int loadPic (fledsADT fleds, pic_t pic, int x, int y);

int loadMovie (fledsADT fleds, pic_t * pic);

int clear (fledsADT fleds);

int show (fledsADT fleds);

int hide (fledsADT fleds);

int animate (fledsADT fleds, animation_t animation, int iterations, int speed);

```

5.11. libFleds.c

libFleds.c

```
#include "libFleds.h"
#include <sys/types.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <time.h>


typedef struct fledsCDT {
        int height;
        int width;
        char * device;
        int fd;
        column_t * pic;
        pic_t * movie;
} fledsCDT;

char letter[] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,0x0, 0x0, 0x0, 0x4F,
0x0, 0x0, 0x0, 0x0,0x0, 0x0, 0x07, 0x0, 0x07, 0x0, 0x0, 0x0,0x0, 0x14, 0x7F,
0x14, 0x7F, 0x14, 0x0, 0x0,0x0, 0x24, 0x0, 0x7F, 0x0, 0x12, 0x0, 0x0,0x0,
0x23, 0x13, 0x08, 0x64, 0x62, 0x0, 0x0,0x0, 0x36, 0x49, 0x55, 0x22, 0x50, 0x0,
0x0,0x0, 0x0, 0x05, 0x03, 0x0, 0x0, 0x0, 0x0,0x0, 0x0, 0x1C, 0x22, 0x41, 0x0,
0x0, 0x0,0x0, 0x0, 0x41, 0x22, 0x1C, 0x0, 0x0, 0x0,0x0, 0x14, 0x08, 0x3E, 0x08,
0x14, 0x0, 0x0,0x0, 0x08, 0x08, 0x3E, 0x08, 0x08, 0x0, 0x0,0x0, 0x0, 0x50,
0x30, 0x0, 0x0, 0x0, 0x0,0x0, 0x08, 0x08, 0x08, 0x08, 0x08, 0x0, 0x0,0x0, 0x0,
0x60, 0x60, 0x0, 0x0, 0x0, 0x0,0x0, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x0,0x0,
0x3E, 0x51, 0x49, 0x45, 0x3E, 0x0, 0x0,0x0, 0x0, 0x42, 0x7F, 0x40, 0x0, 0x0, 0x0,
0x0, 0x42, 0x61, 0x51, 0x49, 0x46, 0x0, 0x0,0x0, 0x21, 0x41, 0x45, 0x4B, 0x31, 0x0,
0x0,0x0, 0x18, 0x14, 0x12, 0x7F, 0x10, 0x0, 0x0,0x0, 0x27, 0x45, 0x45, 0x45, 0x39,
0x0, 0x0,0x0, 0x3C, 0x0, 0x49, 0x49, 0x30, 0x0, 0x0,0x0, 0x03, 0x01, 0x71, 0x09,
0x07, 0x0, 0x0,0x0, 0x36, 0x49, 0x49, 0x49, 0x36, 0x0, 0x0,0x0, 0x06, 0x49, 0x49,
0x29, 0x1E, 0x0, 0x0,0x0, 0x0, 0x36, 0x36, 0x0, 0x0, 0x0, 0x0,0x0, 0x0, 0x56, 0x36,
0x0, 0x0, 0x0, 0x0,0x0, 0x08, 0x14, 0x22, 0x41, 0x0, 0x0, 0x0,0x0, 0x14, 0x14,
0x14, 0x14, 0x14, 0x0, 0x0,0x0, 0x0, 0x41, 0x22, 0x14, 0x08, 0x0, 0x0,0x0, 0x02,
0x01, 0x51, 0x09, 0x06, 0x0, 0x0,0x0, 0x32, 0x49, 0x79, 0x41, 0x3E, 0x0, 0x0,0x0,
0x7E, 0x11, 0x11, 0x11, 0x7E, 0x0, 0x0,0x0, 0x7F, 0x49, 0x49, 0x49, 0x36, 0x0, 0x0,
0x0, 0x3E, 0x41, 0x41, 0x41, 0x22, 0x0, 0x0,0x0, 0x7F, 0x41, 0x41, 0x22, 0x1C, 0x0,
0x0,0x0, 0x7F, 0x49, 0x49, 0x49, 0x41, 0x0, 0x0,0x0, 0x7F, 0x09, 0x09, 0x09, 0x01,
0x0, 0x0,0x0, 0x3E, 0x41, 0x49, 0x49, 0x0, 0x0, 0x0,0x0, 0x7F, 0x08, 0x08, 0x08,
0x7F, 0x0, 0x0,0x0, 0x0, 0x41, 0x7F, 0x41, 0x0, 0x0, 0x0,0x0, 0x20, 0x40, 0x41,
0x3F, 0x01, 0x0, 0x0,0x0, 0x7F, 0x08, 0x14, 0x22, 0x41, 0x0, 0x0,0x0, 0x7F, 0x40,
0x40, 0x40, 0x40, 0x0, 0x0,0x0, 0x7F, 0x02, 0x1C, 0x02, 0x7F, 0x0, 0x0,0x0, 0x7F,
0x04, 0x08, 0x10, 0x7F, 0x0, 0x0,0x0, 0x3E, 0x41, 0x41, 0x41, 0x3E, 0x0, 0x0,0x0,
0x7F, 0x09, 0x09, 0x09, 0x06, 0x0, 0x0,0x0, 0x3E, 0x41, 0x51, 0x21, 0x5E, 0x0, 0x0,
0x0, 0x7E, 0x09, 0x19, 0x29, 0x46, 0x0, 0x0,0x0, 0x46, 0x49, 0x49, 0x49, 0x31, 0x0,
0x00, 0x0, 0x01, 0x01, 0x7F, 0x01, 0x01, 0x0, 0x0,0x0, 0x3F, 0x40, 0x40, 0x40, 0x3F,
0x0, 0x0,0x0, 0x1F, 0x20, 0x40, 0x20, 0x1F, 0x0, 0x0,0x0, 0x3F, 0x40, 0x38, 0x40,
0x3F, 0x0, 0x0,0x0, 0x63, 0x14, 0x08, 0x14, 0x63, 0x0, 0x0,0x0, 0x07, 0x08, 0x70,
0x08, 0x07, 0x0, 0x0,0x0, 0x61, 0x51, 0x49, 0x45, 0x43, 0x0, 0x0,0x0, 0x7F, 0x41,
0x41, 0x0, 0x0, 0x0, 0x0,0x0, 0x02, 0x04, 0x08, 0x10, 0x20, 0x0, 0x0,0x0, 0x0, 0x0,
0x41, 0x41, 0x7F, 0x0, 0x0,0x0, 0x04, 0x02, 0x01, 0x02, 0x04, 0x0, 0x0,0x0, 0x40,
0x40, 0x40, 0x40, 0x40, 0x0, 0x0,0x0, 0x0, 0x0, 0x01, 0x02, 0x04, 0x0, 0x0,0x0, 0x20,
0x54, 0x54, 0x54, 0x78, 0x0, 0x0,0x0, 0x7F, 0x48, 0x44, 0x44, 0x38, 0x0, 0x0,0x0,
0x38, 0x44, 0x44, 0x44, 0x20, 0x0, 0x0,0x0, 0x38, 0x44, 0x44, 0x48, 0x3F, 0x0, 0x0,
```

```
0x0, 0x38, 0x54, 0x54, 0x54, 0x18, 0x0, 0x0,0x0, 0x10, 0x7E, 0x11, 0x01, 0x02, 0x0,
0x0,0x0, 0x08, 0x54, 0x54, 0x54, 0x3C, 0x0, 0x0,0x0, 0x7F, 0x08, 0x04, 0x04, 0x78,
0x0, 0x0,0x0, 0x0, 0x48, 0x7D, 0x40, 0x0, 0x0, 0x0,0x0, 0x20, 0x40, 0x44, 0x3D, 0x0,
0x0, 0x0,0x0, 0x7F, 0x10, 0x28, 0x44, 0x0, 0x0, 0x0,0x0, 0x0, 0x41, 0x7F, 0x40, 0x0,
0x0, 0x0,0x0, 0x7C, 0x04, 0x78, 0x04, 0x78, 0x0, 0x0,0x0, 0x7C, 0x08, 0x04, 0x04,
0x78, 0x0, 0x0,0x0, 0x38, 0x44, 0x44, 0x44, 0x38, 0x0, 0x0,0x0, 0x7C, 0x14, 0x14,
0x14, 0x08, 0x0, 0x0,0x0, 0x08, 0x14, 0x14, 0x18, 0x7C, 0x0, 0x0,0x0, 0x7C, 0x08,
0x04, 0x04, 0x08, 0x0, 0x0,0x0, 0x48, 0x54, 0x54, 0x54, 0x20, 0x0, 0x0,0x0, 0x04,
0x3F, 0x44, 0x40, 0x20, 0x0, 0x0,0x0, 0x3C, 0x40, 0x40, 0x20, 0x7C, 0x0, 0x0,0x0,
0x1C, 0x20, 0x40, 0x20, 0x1C, 0x0, 0x0,0x0, 0x3C, 0x40, 0x30, 0x40, 0x3C, 0x0, 0x0,
0x0, 0x44, 0x28, 0x10, 0x28, 0x44, 0x0, 0x0,0x0, 0x0C, 0x50, 0x50, 0x50, 0x3C, 0x0,
0x0,0x0, 0x0, 0x08, 0x36, 0x41, 0x0, 0x0, 0x0,0x0, 0x44, 0x64, 0x54, 0x4C, 0x44, 0x0,
0x0,0x0, 0x0, 0x41, 0x36, 0x08, 0x0, 0x0, 0x0,0x0, 0x0, 0x0, 0x7F, 0x0, 0x0, 0x0, 0x0,
0x0, 0x10, 0x08, 0x08, 0x10, 0x08, 0x0, 0x00};

int isalpha(char c) {
        return(((c) >= 'a' && (c) <= 'z' ) || ((c) >= 'A' && (c)<='Z'));
}

int islower(char c) {
        return((c) >= 'a' && (c) <= 'z' );
}

int toupper(char c){
        return(islower(c)?(c)-'a'+'A':(c));
        }
void usleep(int time)
{
        int i;
        i=time*USLEEP_TIME;
        while(i--);

        return;
}

char * strdup(char * str)
{
        char * aux;
        int i;

        if ((aux=malloc(strlen(str)+1))==NULL)
                return NULL;

        for(i=0;i<=strlen(str);i++)
                aux[i]=str[i];

        return aux;
}

static int _initFleds(char * device) {
        return open(FLEDS_DEVICE, O_RDWR);
}

fledsADT newFleds (int height, int width, char * device) {
        fledsCDT * ret;
        ret = malloc(sizeof(struct fledsCDT));

        if (ret != NULL ) {
                ret->height = height;
                ret->width = width;
```

```
118                 ret->device = strdup(device);
119                 ret->pic = malloc(sizeof(column_t) * width /* agregar algo para la altura*/);
120
121                 if ((ret->fd=_initFleds(device)) < 0) {
122                         free(ret->device);
123                         free(ret->pic);
124                         free(ret);
125                         return NULL;
126                 }
127         }
128         printf("leds inicializados\n");
129         return ret;
130 }
131
132 int freeFleds (fledsADT fleds) {
133         close(fleds->fd);
134         free(fleds->device);
135         free(fleds->pic);
136         free(fleds);
137         return 0;
138 }
139
140 int loadText (fledsADT fleds, char * text, int x, int y) {
141         int i = 0, j = 0;
142         char c;
143         if(text == NULL || strlen(text) > ((fleds->width)/8))
144                 return 1;
145
146         while(*text) {
147                 if (*text >= ' ' && *text< (' '+95)) {
148                         fleds->pic[i++].red = letter[(*text-' ')* 8 + j++];
149                         fleds->pic[i++].red = letter[(*text-' ')* 8 + j++];
150                         fleds->pic[i++].red = letter[(*text-' ')* 8 + j++];
151                         fleds->pic[i++].red = letter[(*text-' ')* 8 + j++];
152                         fleds->pic[i++].red = letter[(*text-' ')* 8 + j++];
153                         fleds->pic[i++].red = letter[(*text-' ')* 8 + j++];
154                         j = 0;
155                 }
156                 text++;
157         }
158         return 0;
159 }
160
161 int loadPic (fledsADT fleds, pic_t pic, int x, int y) {
162         int i = 0;
163
164         while(i + y < fleds->width) {
165                 if(pic.cols[i].red != 0) {
166                         fleds->pic[i + y].red = pic.cols[i].red;
167                 }
168                 if(pic.cols[i].blue != 0) {
169                         fleds->pic[i + y].blue = pic.cols[i].blue;
170                 }
171                 if(pic.cols[i].green != 0) {
172                         fleds->pic[i + y].green = pic.cols[i].green;
173                 }
174
175                 i++;
176
177         }
```

```
178         return 0;
179 }
180
181 int loadMovie (fledsADT fleds, pic_t * pic) {
182         fleds->movie = pic;
183         return 0;
184 }
185
186 int clear (fledsADT fleds) {
187         int i = 0;
188
189         while(i < fleds->width) {
190                 fleds->pic[i].red = 0;
191                 fleds->pic[i].green = 0;
192                 fleds->pic[i].blue = 0;
193                 fleds->pic[i].id = i;
194                 i++;
195         }
196
197         fleds->movie = NULL;
198
199         return 0;
200 }
201
202 int show (fledsADT fleds) {
203         int i = 0;
204         int index;
205
206         while(i < 16) {
207                 index = i + '@';
208                 write(fleds->fd,&index,1);
209                 write(fleds->fd,&(fleds->pic[i].red),1);
210                 write(fleds->fd,&(fleds->pic[i].green),1);
211                 write(fleds->fd,&(fleds->pic[i].blue),1);
212                 i++;
213         }
214         /*escribir al serial*/
215         return 0;
216 }
217
218 int hide (fledsADT fleds) {
219         char c=0;
220         write(fleds->fd,&c,1);
221
222         return 0;
223 }
224
225 static int _rotlChar(unsigned short int value) {
226         return (value* 2)%128 + (value *2)/128;
227 }
228
229 static int _rotrChar(unsigned short int value) {
230         if (value%2) {
231                 value /=2;
232                 value +=128;
233         } else {
234                 value /=2;
235         }
236         value %= 255;
237         return (value & 0x0ff);
```

```
238 }
239
240 static int scroll( fledsADT fleds, animation_t animation, int iterations, int speed) {
241         int i = 0;
242         column_t actual, proximo;
243
244         show(fleds);
245
246         if(iterations == 0)
247                 return 0;
248         switch(animation) {
249                 case SCROLL_LEFT:
250                 case SCROLL_LEFT_CARRY:
251                         actual = fleds->pic[i];
252                         while(i < fleds->width) {
253                                 proximo = fleds->pic[(i + 1)%fleds->width];
254                                 fleds->pic[(i + 1)%fleds->width] = actual;
255                                 actual = proximo;
256                                 i++;
257                         }
258                         break;
259                 case SCROLL_RIGHT:
260                 case SCROLL_RIGHT_CARRY:
261                         actual = fleds->pic[i];
262                         i = fleds->width;
263                         while(i > 0) {
264                                 proximo = fleds->pic[(i - 1)%fleds->width];
265                                 fleds->pic[(i - 1)%fleds->width] = actual;
266                                 actual = proximo;
267                                 i--;
268                         }
269                         break;
270                 case SCROLL_UP:
271                 case SCROLL_UP_CARRY:
272                         while(i < fleds->width) {
273                                 fleds->pic[i].red  = _rotlChar(fleds->pic[i].red);
274                                 fleds->pic[i].blue = _rotlChar(fleds->pic[i].blue) ;
275                                 fleds->pic[i].green = _rotlChar(fleds->pic[i].green);
276                                 i++;
277                         }
278                 case SCROLL_DOWN:
279                 case SCROLL_DOWN_CARRY:
280                         while(i < fleds->width) {
281                                 fleds->pic[i].red  = _rotrChar(fleds->pic[i].red);
282                                 fleds->pic[i].blue = _rotrChar(fleds->pic[i].blue) ;
283                                 fleds->pic[i].green = _rotrChar(fleds->pic[i].green);
284                                 i++;
285                         }
286         }
287         usleep(speed);
288         return scroll(fleds, animation, iterations - 1, speed);
289 }
290
291 static int scrollRow( fledsADT fleds, animation_t animation, int iterations, int speed) {
292         int i= 0, j=0;
293         int k = 0, aux = 0, x = 1;
294
295         column_t actual, proximo;
296
297         switch(animation) {
```

```
298                 case SCROLL_COLUMN_DOWN:
299                         while(i < iterations && i<fleds->width) {
300                                 while(j < 8) {
301                                         fleds->pic[i].red /=2;
302                                         fleds->pic[i].green /=2;
303                                         fleds->pic[i].blue /=2;
304                                         j++;
305                                         show(fleds);
306                                         usleep(speed);
307                                 }
308                                 j=0;
309                                 i++;
310                         }
311                         break;
312                 case SCROLL_COLUMN_UP:
313                         while(i < iterations && i<fleds->width) {
314                                 while(j < 8) {
315                                         fleds->pic[i].red *=2;
316                                         fleds->pic[i].red %= 256;
317
318                                         fleds->pic[i].green *=2;
319                                         fleds->pic[i].green %= 256;
320
321                                         fleds->pic[i].blue *=2;
322                                         fleds->pic[i].blue %= 256;
323
324                                         j++;
325                                         show(fleds);
326                                         usleep(speed);
327                                 }
328                                 j=0;
329                                 i++;
330                         }
331                         break;
332                 case SCROLL_ROW_RIGHT:
333                                 x = 1;
334                                 for (j = 0; j < 8 && j< iterations; j++)
335                                 {
336
337                                         for( k = 0; k < fleds->width; k++)
338                                         {
339                                                 for( i = fleds->width; i > 0; i--)
340                                                 {
341                                                         fleds->pic[i].red =
(fleds->pic[i].red & (~x)) | (fleds->pic[i - 1].red & x);
342                                                         fleds->pic[i].green =
(fleds->pic[i].green & (~x)) | (fleds->pic[i - 1].green & x);
343                                                         fleds->pic[i].blue =
(fleds->pic[i].blue & (~x)) | (fleds->pic[i - 1].blue & x);
344                                                 }
345                                                 fleds->pic[0].red &= (~x);
346                                                 fleds->pic[0].green &= (~x);
347                                                 fleds->pic[0].blue &= (~x);
348
349                                                 show(fleds);
350                                                 usleep(speed);
351                                         }
352                                         x = x * 2;
353
354                                 }
```

```
355                         break;
356                 default:break;
357         }
358         return 0;
359 }
360
361 static int wave(fledsADT fleds, int iterations, int speed) {
362         int i = 2;
363
364         if (iterations == 0) {
365                 return 0;
366         }
367         show(fleds);
368         usleep(speed);
369
370         fleds->pic[0].red  = _rotlChar(fleds->pic[0].red);
371         fleds->pic[0].blue = _rotlChar(fleds->pic[0].blue) ;
372         fleds->pic[0].green = _rotlChar(fleds->pic[0].green);
373
374         show(fleds);
375         usleep(speed);
376
377         fleds->pic[0].red  = _rotlChar(fleds->pic[0].red);
378         fleds->pic[0].blue = _rotlChar(fleds->pic[0].blue) ;
379         fleds->pic[0].green = _rotlChar(fleds->pic[0].green);
380
381         fleds->pic[1].red  = _rotlChar(fleds->pic[1].red);
382         fleds->pic[1].blue = _rotlChar(fleds->pic[1].blue) ;
383         fleds->pic[1].green = _rotlChar(fleds->pic[1].green);
384
385         show(fleds);
386         usleep(speed);
387
388         fleds->pic[0].red  = _rotrChar(fleds->pic[0].red);
389         fleds->pic[0].blue = _rotrChar(fleds->pic[0].blue) ;
390         fleds->pic[0].green = _rotrChar(fleds->pic[0].green);
391
392         fleds->pic[1].red  = _rotlChar(fleds->pic[1].red);
393         fleds->pic[1].blue = _rotlChar(fleds->pic[1].blue) ;
394         fleds->pic[1].green = _rotlChar(fleds->pic[1].green);
395
396         fleds->pic[2].red  = _rotlChar(fleds->pic[2].red);
397         fleds->pic[2].blue = _rotlChar(fleds->pic[2].blue) ;
398         fleds->pic[2].green = _rotlChar(fleds->pic[2].green);
399
400         show(fleds);
401         usleep(speed);
402
403
404         while(i < fleds->width - 1) {
405                 fleds->pic[i-2].red  = _rotrChar(fleds->pic[i-2].red);
406                 fleds->pic[i-2].blue = _rotrChar(fleds->pic[i-2].blue) ;
407                 fleds->pic[i-2].green = _rotrChar(fleds->pic[i-2].green);
408
409                 fleds->pic[i-1].red  = _rotrChar(fleds->pic[i-1].red);
410                 fleds->pic[i-1].blue = _rotrChar(fleds->pic[i-1].blue) ;
411                 fleds->pic[i-1].green = _rotrChar(fleds->pic[i-1].green);
412
413                 fleds->pic[i].red  = _rotlChar(fleds->pic[i].red);
414                 fleds->pic[i].blue = _rotlChar(fleds->pic[i].blue) ;
```

```
415                 fleds->pic[i].green = _rotlChar(fleds->pic[i].green);
416
417                 fleds->pic[i+1].red  = _rotlChar(fleds->pic[i+1].red);
418                 fleds->pic[i+1].blue = _rotlChar(fleds->pic[i+1].blue) ;
419                 fleds->pic[i+1].green = _rotlChar(fleds->pic[i+1].green);
420
421                 show(fleds);
422                 usleep(speed);
423                 i++;
424         }
425
426         fleds->pic[fleds->width-3].red  = _rotrChar(fleds->pic[fleds->width-3].red);
427         fleds->pic[fleds->width-3].blue = _rotrChar(fleds->pic[fleds->width-3].blue) ;
428         fleds->pic[fleds->width-3].green = _rotrChar(fleds->pic[fleds->width-3].green);
429
430         fleds->pic[fleds->width-2].red  = _rotrChar(fleds->pic[fleds->width-2].red);
431         fleds->pic[fleds->width-2].blue = _rotrChar(fleds->pic[fleds->width-2].blue) ;
432         fleds->pic[fleds->width-2].green = _rotrChar(fleds->pic[fleds->width-2].green);
433
434         fleds->pic[fleds->width-1].red  = _rotlChar(fleds->pic[fleds->width-1].red);
435         fleds->pic[fleds->width-1].blue = _rotlChar(fleds->pic[fleds->width-1].blue) ;
436         fleds->pic[fleds->width-1].green = _rotlChar(fleds->pic[fleds->width-1].green);
437
438         show(fleds);
439         usleep(speed);
440
441         fleds->pic[fleds->width-2].red  = _rotrChar(fleds->pic[fleds->width-2].red);
442         fleds->pic[fleds->width-2].blue = _rotrChar(fleds->pic[fleds->width-2].blue) ;
443         fleds->pic[fleds->width-2].green = _rotrChar(fleds->pic[fleds->width-2].green);
444
445         fleds->pic[fleds->width-1].red  = _rotrChar(fleds->pic[fleds->width-1].red);
446         fleds->pic[fleds->width-1].blue = _rotrChar(fleds->pic[fleds->width-1].blue) ;
447         fleds->pic[fleds->width-1].green = _rotrChar(fleds->pic[fleds->width-1].green);
448
449         show(fleds);
450         usleep(speed);
451
452         fleds->pic[fleds->width-1].red  = _rotrChar(fleds->pic[fleds->width-1].red);
453         fleds->pic[fleds->width-1].blue = _rotrChar(fleds->pic[fleds->width-1].blue) ;
454         fleds->pic[fleds->width-1].green = _rotrChar(fleds->pic[fleds->width-1].green);
455
456         show(fleds);
457         usleep(speed);
458
459         return wave(fleds, iterations-1, speed);
460
461 }
462
463 static int twinkle(fledsADT fleds, int iterations, int speed) {
464
465         show(fleds);
466         usleep(speed);
467         if(iterations == 0) {
468                 return 0;
469         }
470         hide(fleds);
471         usleep(speed);
472         return twinkle(fleds, iterations-1, speed);
473 }
474
```

```
475 static int life(fledsADT fleds, int iterations, int speed) {
476         return 0;
477 }
478
479 static int lsd(fledsADT fleds, int iterations, int speed) {
480         int i = 0;
481
482         show(fleds);
483         if(iterations == 0) {
484                 return 0;
485         }
486
487         while(i < fleds->width) {
488                 fleds->pic[i].red = (~(fleds->pic[i].green)+time(NULL))%256;
489                 fleds->pic[i].green = ~(fleds->pic[i].blue);
490                 fleds->pic[i].blue = (~((fleds->pic[i].red + fleds->pic[i].blue +
    fleds->pic[i].green )%256) + time(NULL))%256;
491                 i++;
492         }
493
494         usleep(speed);
495         return lsd(fleds, iterations - 1, speed);
496 }
497
498 static int showCamel(fledsADT fleds) {
499         int i = 0;
500         int index;
501
502         index = 4 + '@';
503
504         write(fleds->fd,&index,1);
505         write(fleds->fd,&(fleds->pic[i].red),1);
506         write(fleds->fd,&(fleds->pic[i].green),1);
507         write(fleds->fd,&(fleds->pic[i].blue),1);
508
509         return 0;
510 }
511
512 static int camel(fledsADT fleds, int iterations, int speed) {
513         column_t actual, proximo;
514         int i= 0;
515         showCamel(fleds);
516
517         if(iterations == 0) {
518                 return 0;
519         }
520
521         actual = fleds->pic[i];
522         while(i < fleds->width) {
523                 proximo = fleds->pic[(i + 1)%fleds->width];
524                 fleds->pic[(i + 1)%fleds->width] = actual;
525                 actual = proximo;
526                 i++;
527         }
528
529         return camel(fleds,iterations-1, speed);
530 }
531
532 static int showFrame(pic_t pic, int width, int fd) {
533         int i = 0;
```

```
534
535         while(i < 16) {
536
537                 int index;
538
539                 index = i + '@';
540
541                 write(fd,&index,1);
542                 write(fd,&(pic.cols[i].red),1);
543                 write(fd,&(pic.cols[i].green),1);
544                 write(fd,&(pic.cols[i].blue),1);
545
546                 i++;
547         }
548         /*escribir al serial*/
549         return 0;
550 }
551
552 static int play(fledsADT fleds, int iterations, int speed) {
553         int i;
554
555         while(iterations--) {
556                 while( ((fleds->movie)[i]).cols[0].id != -1 ) {
557                         showFrame((fleds->movie)[i], fleds->width, fleds->fd);
558                         i++;
559                         usleep(speed);
560                 }
561                 i = 0;
562                 usleep(speed);
563         }
564
565         return 0;
566 }
567
568 int animate (fledsADT fleds, animation_t animation, int iterations, int speed) {
569         switch(animation) {
570
571                 case SCROLL_NONE:
572                         break;
573
574                 case SCROLL_RIGHT_CARRY:
575                 case SCROLL_RIGHT:
576                 case SCROLL_LEFT_CARRY:
577                 case SCROLL_LEFT:
578                 case SCROLL_UP_CARRY:
579                 case SCROLL_UP:
580                 case SCROLL_DOWN_CARRY:
581                 case SCROLL_DOWN: scroll(fleds, animation, iterations, speed);break;
582
583                 case SCROLL_ROW_LEFT:
584                 case SCROLL_COLUMN_DOWN:
585                 case SCROLL_ROW_RIGHT:
586                 case SCROLL_COLUMN_UP:scrollRow(fleds, animation, iterations, speed);break;
587
588                 case WAVE: wave(fleds, iterations, speed);break;
589
590                 case TWINKLE: twinkle(fleds, iterations, speed);break;
591
592                 case LIFE:life(fleds, iterations, speed);break;
593
```

```
594                 case LSD:lsd(fleds, iterations, speed);break;
595
596                 case CAMEL:camel(fleds, iterations, speed);break;
597
598                 case MOVIE:play(fleds, iterations, speed);break;
599
600                 default: break;
601         }
602         return 0;
603 }
604
605 int prueba(fledsADT fleds, int iterations)
606 {
607         int k = 0, i = 0, j = 0, aux = 0, x = 1;
608
609         for (j = 0; j < 8; j++)
610         {
611                 x = 1;
612                 for( k = 0; k < fleds->width; k--)
613                 {
614                         for( i = fleds->width; i > 0; i--)
615                         {
616                                 fleds->pic[i].red = (fleds->pic[i].red & (~x)) |
    (fleds->pic[i - 1].red & x);
617                                 fleds->pic[i].green = (fleds->pic[i].green & (~x)) |
    (fleds->pic[i - 1].green & x);
618                                 fleds->pic[i].blue = (fleds->pic[i].blue & (~x)) |
    (fleds->pic[i - 1].blue & x);
619                         }
620                         fleds->pic[0].red &= (~x);
621                         fleds->pic[0].green &= (~x);
622                         fleds->pic[0].blue &= (~x);
623                         x = x * 2;
624                 }
625         }
626
627         return 0;
628 }
```

6. Conclusiones

Creemos haber cumplido el objetivo que nos planteamos al comienzo del tpe, desarrollar un dispositivo físico de leds y su respectivo driver para el sistema operativo Minix.
Se comprendió la interacción de Minix internamente, el armado de un controlador nuevo y la vinculación entre sí.
Finalmente, estamos satisfechos con el resultado obtenido y consideramos ésta una experiencia muy positiva.

7. Posibles extensiones

Se podría agregar dentro de lo que es el controlador la posibilidad de cambiarle parámetros de conexión con el dispositivo utilizando ioctl. Actualmente está definido pero no se está utilizando. Respecto al dispositivo, agregar más módulos para mostrar imágenes con

una mayor resolución o textos más largos.

Referencias

[1] "*Sistemas Operativos. Diseño e implementación*" - Andrew S. Tanenbaum, Albert S. Woodhull

[2] "*Intro_minix.pdf*" - Cátedra de Sistemas Operativos

[3] "*Diseño de Sistemas Operativos*" - http://gsd.unex.es/ jdiaz/asig/dso/lab/p8/p8_v2.pdf

[4] "*Librerías fuentes de Minix*" - /usr/ dentro de minix.