

**PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA**

**DISEÑO E IMPLEMENTACIÓN DE UN DISPOSITIVO DE
LLAMADA, VISUALIZACIÓN Y COMUNICACIÓN PARA
ASCENSORES**

ANEXOS

ANEXO A

Archivos del proyecto para demo desarrollado en Code Composer 6.1.3 de Texas Instruments.

Configuración de las propiedades del proyecto

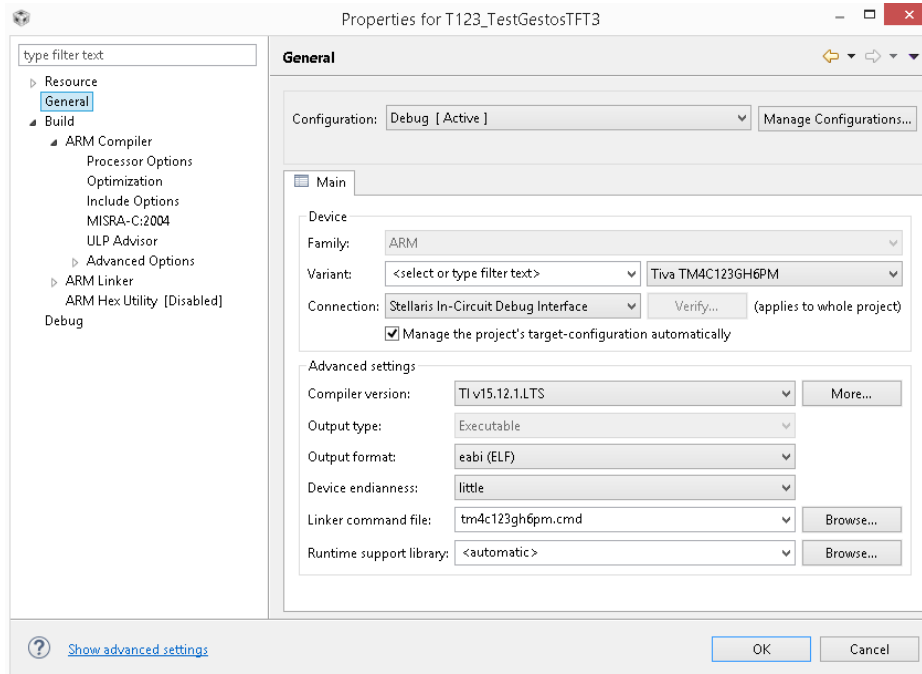


Fig. 1.- Selección de microcontrolador, método de depuración y programación, versión de compilación y tipo de salida del compilador.

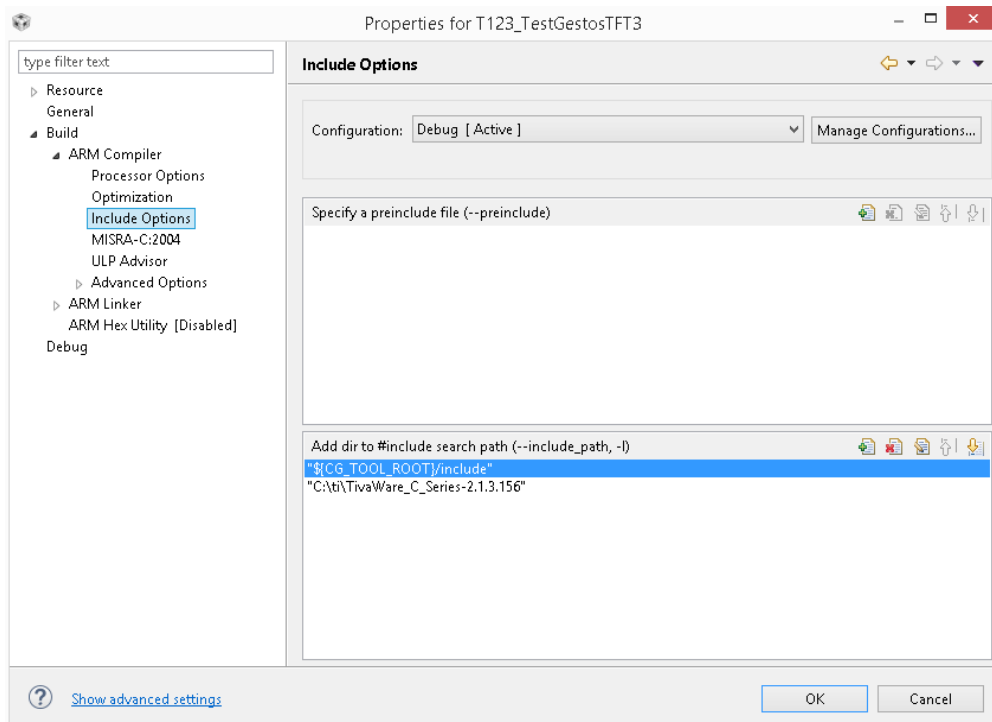


Fig. 2.- Archivo añadido a la ruta del compilador ARM con la librería Tivaware.

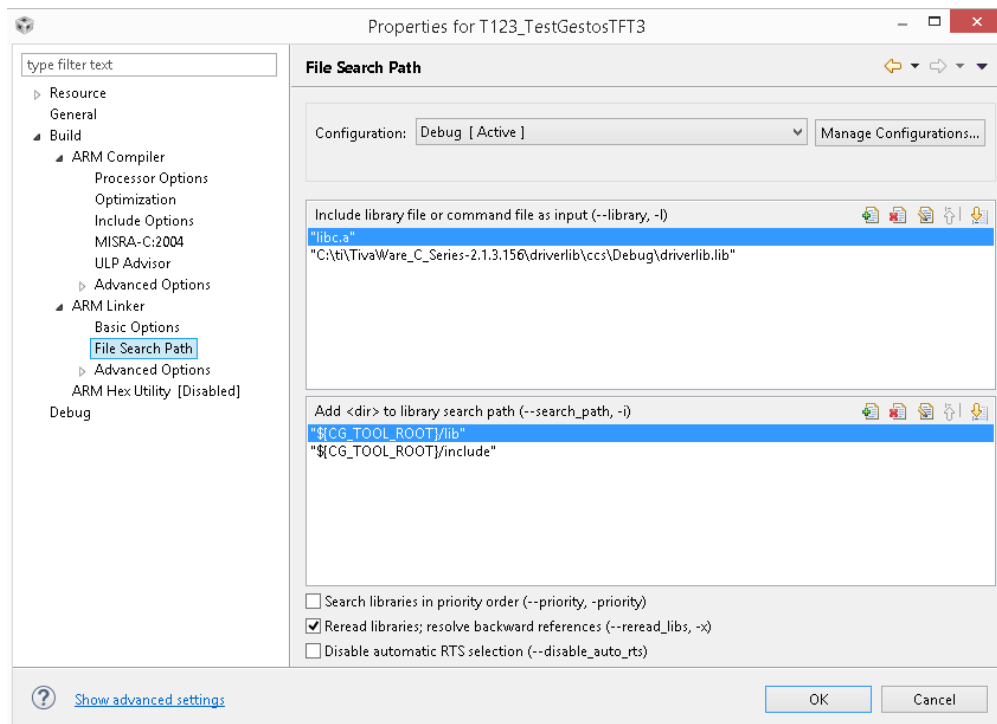


Fig. 3.- Archivo añadido a la ruta del Enlazador ARM al archivo "driverlib.lib" de la librería Tivaware.

Archivo "tm4c123gh6pm_startup_ccs.c"

```

1. //*****
2. //
3. // Startup code for use with TI's Code Composer Studio.
4. //
5. // Copyright (c) 2011-
6. // 2014 Texas Instruments Incorporated. All rights reserved.
7. // Software License Agreement
8. // Software License Agreement
9. //
10. // Texas Instruments (TI) is supplying this software for use solely an
11. // d
12. // exclusively on TI's microcontroller products. The software is owned
13. // by
14. // TI and/or its suppliers, and is protected under applicable copyrigh
15. // t
16. // laws. You may not combine this software with "viral" open-source
17. // software in order to form a larger program.
18. //
19. // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
20. // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BU
21. // T
22. // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS F
23. // OR
24. // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER AN
25. // Y
26. // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
27. // DAMAGES, FOR ANY REASON WHATSOEVER.

```

```

22. //
23. //*****
    *****
24.
25. #include <stdint.h>
26.
27. //*****
    *****
28. //
29. // Forward declaration of the default fault handlers.
30. //
31. //*****
    *****
32. void ResetISR(void);
33. static void NmiSR(void);
34. static void FaultISR(void);
35. static void IntDefaultHandler(void);
36.
37. //*****
    *****
38. //
39. // External declaration for the reset handler that is to be called whe
n the
40. // processor is started
41. //
42. //*****
    *****
43. extern void _c_int00(void);
44. extern void UART2IntHandler(void);
45.
46. //*****
    *****
47. //
48. // Linker variable that marks the top of the stack.
49. //
50. //*****
    *****
51. extern uint32_t __STACK_TOP;
52.
53. //*****
    *****
54. //
55. // External declarations for the interrupt handlers used by the applic
ation.
56. //
57. //*****
    *****
58. // To be added by user
59.
60. //*****
    *****
61. //
62. // The vector table. Note that the proper constructs must be placed o
n this to
63. // ensure that it ends up at physical address 0x0000.0000 or at the st
art of
64. // the program if located at a start address other than 0.
65. //

```

```

66. //*****
        *****
67. #pragma DATA_SECTION(g_pfnVectors, ".intvecs")
68. void (* const g_pfnVectors[])(void) =
69. {
70.     (void (*)(void))((uint32_t)&__STACK_TOP),
71.                                     // The initial stack point
er
72.     ResetISR,                       // The reset handler
73.     NmiISR,                         // The NMI handler
74.     FaultISR,                      // The hard fault handler
75.     IntDefaultHandler,             // The MPU fault handler
76.     IntDefaultHandler,             // The bus fault handler
77.     IntDefaultHandler,             // The usage fault handler
78.     0,                             // Reserved
79.     0,                             // Reserved
80.     0,                             // Reserved
81.     0,                             // Reserved
82.     IntDefaultHandler,             // SVCall handler
83.     IntDefaultHandler,             // Debug monitor handler
84.     0,                             // Reserved
85.     IntDefaultHandler,             // The PendSV handler
86.     IntDefaultHandler,             // The SysTick handler
87.     IntDefaultHandler,             // GPIO Port A
88.     IntDefaultHandler,             // GPIO Port B
89.     IntDefaultHandler,             // GPIO Port C
90.     IntDefaultHandler,             // GPIO Port D
91.     IntDefaultHandler,             // GPIO Port E
92.     IntDefaultHandler,             // UART0 Rx and Tx
93.     IntDefaultHandler,             // UART1 Rx and Tx
94.     IntDefaultHandler,             // SSI0 Rx and Tx
95.     IntDefaultHandler,             // I2C0 Master and Slave
96.     IntDefaultHandler,             // PWM Fault
97.     IntDefaultHandler,             // PWM Generator 0
98.     IntDefaultHandler,             // PWM Generator 1
99.     IntDefaultHandler,             // PWM Generator 2
100.    IntDefaultHandler,              // Quadrature Encode
r 0
101.    IntDefaultHandler,              // ADC Sequence 0
102.    IntDefaultHandler,              // ADC Sequence 1
103.    IntDefaultHandler,              // ADC Sequence 2
104.    IntDefaultHandler,              // ADC Sequence 3
105.    IntDefaultHandler,              // Watchdog timer
106.    IntDefaultHandler,              // Timer 0 subtimer
A
107.    IntDefaultHandler,              // Timer 0 subtimer
B
108.    IntDefaultHandler,              // Timer 1 subtimer
A
109.    IntDefaultHandler,              // Timer 1 subtimer
B
110.    IntDefaultHandler,              // Timer 2 subtimer
A
111.    IntDefaultHandler,              // Timer 2 subtimer
B
112.    IntDefaultHandler,              // Analog Comparator
0

```

```

113.      IntDefaultHandler,          // Analog Comparator
114.      1
115.      IntDefaultHandler,          // Analog Comparator
116.      2
117.      IntDefaultHandler,          // System Control (P
118.      LL, OSC, B0)
119.      IntDefaultHandler,          // FLASH Control
120.      IntDefaultHandler,          // GPIO Port F
121.      IntDefaultHandler,          // GPIO Port G
122.      IntDefaultHandler,          // GPIO Port H
123.      UART2IntHandler,            // UART2 Rx and Tx
124.      IntDefaultHandler,          // SSI1 Rx and Tx
125.      IntDefaultHandler,          // Timer 3 subtimer
126.      A
127.      IntDefaultHandler,          // Timer 3 subtimer
128.      B
129.      IntDefaultHandler,          // I2C1 Master and S
130.     lave
131.      IntDefaultHandler,          // Quadrature Encode
132.      r 1
133.      IntDefaultHandler,          // CAN0
134.      IntDefaultHandler,          // CAN1
135.      0,                          // Reserved
136.      0,                          // Reserved
137.      IntDefaultHandler,          // Hibernate
138.      IntDefaultHandler,          // USB0
139.      IntDefaultHandler,          // PWM Generator 3
140.      IntDefaultHandler,          // uDMA Software Tra
141.      nsfer
142.      IntDefaultHandler,          // uDMA Error
143.      IntDefaultHandler,          // ADC1 Sequence 0
144.      IntDefaultHandler,          // ADC1 Sequence 1
145.      IntDefaultHandler,          // ADC1 Sequence 2
146.      IntDefaultHandler,          // ADC1 Sequence 3
147.      0,                          // Reserved
148.      0,                          // Reserved
149.      IntDefaultHandler,          // GPIO Port J
150.      IntDefaultHandler,          // GPIO Port K
151.      IntDefaultHandler,          // GPIO Port L
152.      IntDefaultHandler,          // SSI2 Rx and Tx
153.      IntDefaultHandler,          // SSI3 Rx and Tx
154.      IntDefaultHandler,          // UART3 Rx and Tx
155.      IntDefaultHandler,          // UART4 Rx and Tx
156.      IntDefaultHandler,          // UART5 Rx and Tx
157.      IntDefaultHandler,          // UART6 Rx and Tx
158.      IntDefaultHandler,          // UART7 Rx and Tx
159.      0,                          // Reserved
160.      0,                          // Reserved
161.      0,                          // Reserved
162.      0,                          // Reserved
163.      IntDefaultHandler,          // I2C2 Master and S
164.     lave
165.      IntDefaultHandler,          // I2C3 Master and S
166.     lave
167.      IntDefaultHandler,          // Timer 4 subtimer
168.      A
169.      IntDefaultHandler,          // Timer 4 subtimer
170.      B
171.      0,                          // Reserved

```

```

160.      0, // Reserved
161.      0, // Reserved
162.      0, // Reserved
163.      0, // Reserved
164.      0, // Reserved
165.      0, // Reserved
166.      0, // Reserved
167.      0, // Reserved
168.      0, // Reserved
169.      0, // Reserved
170.      0, // Reserved
171.      0, // Reserved
172.      0, // Reserved
173.      0, // Reserved
174.      0, // Reserved
175.      0, // Reserved
176.      0, // Reserved
177.      0, // Reserved
178.      0, // Reserved
179.      IntDefaultHandler, // Timer 5 subtimer
    A
180.      IntDefaultHandler, // Timer 5 subtimer
    B
181.      IntDefaultHandler, // Wide Timer 0 subt
    imer A
182.      IntDefaultHandler, // Wide Timer 0 subt
    imer B
183.      IntDefaultHandler, // Wide Timer 1 subt
    imer A
184.      IntDefaultHandler, // Wide Timer 1 subt
    imer B
185.      IntDefaultHandler, // Wide Timer 2 subt
    imer A
186.      IntDefaultHandler, // Wide Timer 2 subt
    imer B
187.      IntDefaultHandler, // Wide Timer 3 subt
    imer A
188.      IntDefaultHandler, // Wide Timer 3 subt
    imer B
189.      IntDefaultHandler, // Wide Timer 4 subt
    imer A
190.      IntDefaultHandler, // Wide Timer 4 subt
    imer B
191.      IntDefaultHandler, // Wide Timer 5 subt
    imer A
192.      IntDefaultHandler, // Wide Timer 5 subt
    imer B
193.      IntDefaultHandler, // FPU
194.      0, // Reserved
195.      0, // Reserved
196.      IntDefaultHandler, // I2C4 Master and S
    lave
197.      IntDefaultHandler, // I2C5 Master and S
    lave
198.      IntDefaultHandler, // GPIO Port M
199.      IntDefaultHandler, // GPIO Port N
200.      IntDefaultHandler, // Quadrature Encode
    r 2
201.      0, // Reserved

```

```

202.         0,                                // Reserved
203.         IntDefaultHandler,                // GPIO Port P (Summ
    ary or P0)
204.         IntDefaultHandler,                // GPIO Port P1
205.         IntDefaultHandler,                // GPIO Port P2
206.         IntDefaultHandler,                // GPIO Port P3
207.         IntDefaultHandler,                // GPIO Port P4
208.         IntDefaultHandler,                // GPIO Port P5
209.         IntDefaultHandler,                // GPIO Port P6
210.         IntDefaultHandler,                // GPIO Port P7
211.         IntDefaultHandler,                // GPIO Port Q (Summ
    ary or Q0)
212.         IntDefaultHandler,                // GPIO Port Q1
213.         IntDefaultHandler,                // GPIO Port Q2
214.         IntDefaultHandler,                // GPIO Port Q3
215.         IntDefaultHandler,                // GPIO Port Q4
216.         IntDefaultHandler,                // GPIO Port Q5
217.         IntDefaultHandler,                // GPIO Port Q6
218.         IntDefaultHandler,                // GPIO Port Q7
219.         IntDefaultHandler,                // GPIO Port R
220.         IntDefaultHandler,                // GPIO Port S
221.         IntDefaultHandler,                // PWM 1 Generator 0

222.         IntDefaultHandler,                // PWM 1 Generator 1
223.         IntDefaultHandler,                // PWM 1 Generator 2
224.         IntDefaultHandler,                // PWM 1 Generator 3
225.         IntDefaultHandler                // PWM 1 Fault
226.     };
227.
228.     //*****
    *****
229.     //
230.     // This is the code that gets called when the processor first st
    arts execution
231.     // following a reset event. Only the absolutely necessary set i
    s performed,
232.     // after which the application supplied entry() routine is calle
    d. Any fancy
233.     // actions (such as making decisions based on the reset cause re
    gister, and
234.     // resetting the bits in that register) are left solely in the h
    ands of the
235.     // application.
236.     //
237.     //*****
    *****
238.     void
239.     ResetISR(void)
240.     {
241.         //
242.         // Jump to the CCS C initialization routine. This will enab
    le the
243.         // floating-
    point unit as well, so that does not need to be done here.
244.         //
245.         __asm("        .global _c_int00\n"

```



```

246.         "    b.w    _c_int00");
247.     }
248.
249.     //*****
250.     //
251.     // This is the code that gets called when the processor receives
252.     // a NMI. This
253.     // simply enters an infinite loop, preserving the system state f
254.     // or examination
255.     // by a debugger.
256.     //
257.     //*****
258.     static void
259.     NmiISR(void)
260.     {
261.         //
262.         // Enter an infinite loop.
263.         //
264.         while(1)
265.         {
266.         }
267.     }
268.     //*****
269.     //
270.     // This is the code that gets called when the processor receives
271.     // a fault
272.     // interrupt. This simply enters an infinite loop, preserving t
273.     // he system state
274.     // for examination by a debugger.
275.     //
276.     //*****
277.     static void
278.     FaultISR(void)
279.     {
280.         //
281.         // Enter an infinite loop.
282.         //
283.         while(1)
284.         {
285.         }
286.     }
287.     //*****
288.     //
289.     // This is the code that gets called when the processor receives
290.     // an unexpected
291.     // interrupt. This simply enters an infinite loop, preserving t
292.     // he system state
293.     // for examination by a debugger.
294.     //
295.     //*****
296.     static void

```

```

293.     IntDefaultHandler(void)
294.     {
295.         //
296.         // Go into an infinite loop.
297.         //
298.         while(1)
299.         {
300.         }
301.     }

```

Código fuente 1.- Archivo con nombre de funciones asociadas a interrupciones.

Archivo "main.c"

```

1.  /*
2.  * main.c
3.  * CAN_LCD_UARTGESTURE SOURCE CODE JOINED
4.  * CAN_TX: SEND UP OR DOWN CALL
5.  * CAN_RX: GET FLOOR, ARROWS AND MAINTENANCE
6.  * UARTGESTURE: RECOGNIZE HOVER_UP or HOVER_DOWN
7.  * LCD: REFLECT CALLS, FLOORS, ARROWS AND MAINTENANCE
8.  * -----
9.  * -----
10. * Author: Luis Gonzales Miranda
11. * Thesis Advisor: Laureano Rodriguez
12. * University: PUCP
13. * Project: Calling and visualization device for elevator systems using CAN co
mmunication
14. * Date: 03/12/2018
15. * Description: Demo with only scrool
16. * -----
17. * -----
18. * Based on:
19. * Tivaware Examples (CAN): https://github.com/yuvadm/tiva-
c/blob/master/boards/dk-tm4c123g/can/can.c
20. * NINJA Blog Tiva (CAN): http://ohm.ninja/tiva-c-series-can-bus-with-
mcp2551/
21. * Arduino Library(Gesture Sensor): https://github.com/sparkfun/SparkFun\_ZX\_Di
stance\_and\_Gesture\_Sensor\_Arduino\_Library
22. * TI E2E Community(LCD): https://e2e.ti.com/support/microcontrollers/tiva\_arm
/f/908/p/471799/1695767
23. *
24. * Clock Tree Reference: http://embedded-lab.com/blog/tiva-c-clock-system/
25. */
26.
27. #include <stdint.h>
28. #include <stdbool.h>
29. #include <string.h>
30.
31. #include "can_func.h"
32. #include "lcd_func.h"
33. #include "uart_func.h"
34. #include "colors_fig.h"
35. #include "watchdog_func.h"
36.
37. #include "inc/hw_memmap.h"
38. #include "inc/hw_types.h"
39. #include "inc/hw_can.h"
40. #include "inc/hw_ints.h"
41.
42. #include "driverlib/rom.h"
43. #include "driverlib/rom_map.h"
44. #include "driverlib/can.h"
45. #include "driverlib/interrupt.h"

```

```

46. #include "driverlib/sysctl.h"
47. #include "driverlib/gpio.h"
48. #include "driverlib/uart.h"
49. #include "driverlib/pin_map.h"
50. #include "driverlib/systick.h"
51.
52. #include "driverlib/debug.h"
53. #include "driverlib/watchdog.h"
54.
55. #include "utils/uartstdio.h"
56.
57. uint16_t piso_actual = 2;
58.
59. void SysTick_IntHandler(void)
60. {
61.     time++;
62. }
63.
64. void main(void) {
65.
66.     //Clock Configuration: 80MHz using PLL: 400MHz from 16MHZ OSC CRYSTAL
67.     MAP_SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_16MHZ);
68.     MAP_SysCtlPioSCCalibrate(SYSCTL_PIOOSC_CAL_FACT);
69.
70.     MAP_SysCtlDelay(MAP_SysCtlClockGet()/60); //F_CPU*3cycleperloop/60 = 50ms
71.
72.     //Peripheral Configuration
73.     conf_uart0(); //Conf Debugger (PA0(U0Rx)-PA1(U0Tx))
74.     conf_uart2(); //Conf Gesture Sensor UART (PD6(U2Rx)-PD7(U2Tx))
75.     conf_can0(); //Conf CAN0 (PE4(CAN0Rx)-PE5(CAN0Tx))
76.     conf_lcd(); //Conf SSI0(MOSI(PA5)-MISO(PA4)-SCLK(PA2)-
        CS(PE3)) and Init_lcd(RESET(PE2)-DC(PE1))
77.     LCD_ScrollingDef(80,170,70);
78.
79.     //habilita_watchdog(2); //Conf Watchdog 2 sec max.
80.
81.     LCD_FillTriangle(67,10,50,100,BRED);
82.     LCD_ShowNum_2(61,85,piso_actual); //XInit:61, YInit: 0, Num: Posicion11
        de tabla newFont, Mode:Non-overlapping
83.
84.     LCD_FillTriangleD(67,250,50,100,BRED);
85.     LCD_ScrollingStart(81);
86.     IntMasterEnable();
87.
88.     SysTickIntRegister(SysTick_IntHandler);
89.     SysTickPeriodSet(80000);
90.     SysTickIntEnable();
91.     SysTickEnable();
92.     //_BKPT(2);
93.     uint16_t cont=0;
94.     uint32_t scrool = 0;
95.     while(true){
96.         //MAP_WatchdogReloadSet(WATCHDOG0_BASE, MAP_SysCtlClockGet() * 2);
97.
98.         processv04_uart2();
99.
100.         if(time-scrool > 10){
101.             LCD_ScrollingStart(81+cont++);
102.             if(cont==170) cont = 0;
103.             scrool = time;
104.         }
105.         //proc_can0();
106.         //SysCtlDelay(SysCtlClockGet()/200);

```

```

107.
108.     }
109.
110.     }

```

Código fuente 2.- Archivo con el programa principal.

Archivo "can_fun.c"

```

1.  /*
2.   * can_func.c
3.   *
4.   * Created on: 4 de dic. de 2016
5.   * Author: Francisco
6.   */
7.
8. #include <stdint.h>
9. #include <stdbool.h>
10. #include <string.h>
11.
12. #include "can_func.h"
13. #include "uart_func.h"
14. #include "lcd_func.h"
15.
16. #include "inc/hw_memmap.h"
17. #include "inc/hw_types.h"
18. #include "inc/hw_can.h"
19. #include "inc/hw_ints.h"
20.
21. #include "driverlib/rom.h"
22. #include "driverlib/rom_map.h"
23. #include "driverlib/can.h"
24. #include "driverlib/interrupt.h"
25. #include "driverlib/sysctl.h"
26. #include "driverlib/gpio.h"
27. #include "driverlib/pin_map.h"
28.
29. void
30. CAN0IntHandler(void) {
31.
32.     uint32_t status = CANIntStatus(CAN0_BASE, CAN_INT_STS_CAUSE);
33.
34.     if(status == CAN_INT_INTID_STATUS) {
35.         status = CANStatusGet(CAN0_BASE, CAN_STS_CONTROL);
36.         errFlag = 1;
37.     } else if (status == NOBJECT_RXCAN0 || status == NOBJECT_RXCAN
38. 0+1 || status == NOBJECT_RXCAN0+2 || status == NOBJECT_RXCAN0+3
39. || status == NOBJECT_RXCAN0+4 || status == NOBJECT_RXCAN
40. 0+5 || status == NOBJECT_RXCAN0+6 || status == NOBJECT_RXCAN0+7
41. || status == NOBJECT_RXCAN0+8 || status == NOBJECT_RXCAN
42. 0+9 || status == NOBJECT_RXCAN0+10 || status == NOBJECT_RXCAN0+11
43. || status == NOBJECT_RXCAN0+12 || status == NOBJECT_RXCAN
44. 0+13 || status == NOBJECT_RXCAN0+14 || status == NOBJECT_RXCAN0+15
45. || status == NOBJECT_RXCAN0+16 || status == NOBJECT_RXCAN
46. 0+17 || status == NOBJECT_RXCAN0+18 || status == NOBJECT_RXCAN0+19
47. || status == NOBJECT_RXCAN0+20 || status == NOBJECT_RXCAN
48. 0+21 || status == NOBJECT_RXCAN0+22 || status == NOBJECT_RXCAN0+23) {

```

```

43.     CANMessageGet(CAN0_BASE, NOBJECT_RXCAN0, &msgCANobjRx, 1);
44.     rxFlag = 1;
45.     errFlag = 0;
46. } else if(status == NOBJECT_TXCAN0){
47.     CANIntClear(CAN0_BASE, NOBJECT_TXCAN0);
48.     CANMessageClear(CAN0_BASE, NOBJECT_TXCAN0);
49.     g_ui32MsgTxCount++;
50.     errFlag = 0;
51. }
52.
53. else {
54.
55. }
56.}
57.
58.
59.//Could be needed activate the flag MSG_OBJ__EXTENDED_ID
60.//Modified expression msgCANobj.ui32Flags = MSG_OBJ_RX_INT_ENABLE | MSG
    G_OBJ_USE_ID_FILTER | MSG_OBJ_FIFO | MSG_OBJ__EXTENDED_ID;
61.//The silent mode before init could be also required:
62.//HWREG(CAN0_BASE+CAN_O_CTL) |= CAN_CTL_TEST;
63.//HWREG(CAN0_BASE+CAN_O_TST) |= CAN_TST_SILENT;
64.//The function for setbittiming is:
65.//tCANBitClkParms CANBitClk;
66.//CANBitClk = {4,4,3,80};
67.//CANBitTimingSet(CAN0_BASE,&CANBitClk);
68.
69.void
70.conf_can0(void){
71.    SysCtlPeripheralEnable(SYSCTL_PERIPH_CAN0);
72.    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
73.    GPIOPinConfigure(GPIO_PE4_CAN0RX);
74.    GPIOPinConfigure(GPIO_PE5_CAN0TX);
75.    GPIOPinTypeCAN(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
76.    CANInit(CAN0_BASE);
77.    CANBitRateSet(CAN0_BASE, SysCtlClockGet(), CAN0_BRATE);
78.    CANIntRegister(CAN0_BASE, CAN0IntHandler);
79.    CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR | CAN_INT_S
    TATUS);
80.    IntEnable(INT_CAN0);
81.    CANEnable(CAN0_BASE);
82.
83.    msgCANobjRx.ui32MsgID = 0;
84.    msgCANobjRx.ui32MsgIDMask = 0;
85.    msgCANobjRx.ui32Flags = MSG_OBJ_RX_INT_ENABLE | MSG_OBJ_USE_ID_FIL
    TER | MSG_OBJ_FIFO;
86.
87.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0, &msgCANobjRx, MSG_OBJ_TYP
    E_RX);
88.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+1, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
89.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+2, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
90.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+3, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
91.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+4, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
92.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+5, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);

```

```

93.     CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+6, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
94.     CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+7, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
95.     CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+8, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
96.     CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+9, &msgCANobjRx, MSG_OBJ_T
    YPE_RX);
97.     CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+10, &msgCANobjRx, MSG_OBJ_
    TYPE_RX);
98.     CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+11, &msgCANobjRx, MSG_OBJ_
    TYPE_RX);
99.     CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+12, &msgCANobjRx, MSG_OBJ_
    TYPE_RX);
100.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+13, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
101.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+14, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
102.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+15, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
103.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+16, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
104.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+17, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
105.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+18, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
106.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+19, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
107.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+20, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
108.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+21, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
109.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+22, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
110.
111.    msgCANobjRx.ui32Flags = MSG_OBJ_RX_INT_ENABLE | MSG_OBJ_USE_
    ID_FILTER;
112.    CANMessageSet(CAN0_BASE, NOBJECT_RXCAN0+23, &msgCANobjRx, MS
    G_OBJ_TYPE_RX);
113.
114.    //Configuring the structure for the object of transmission
115.    msgCANobjTx.ui32MsgID = ID_TXCAN0;
116.    msgCANobjTx.ui32MsgIDMask = 0;
117.    msgCANobjTx.ui32Flags = MSG_OBJ_TX_INT_ENABLE;
118.    msgCANobjTx.ui32MsgLen = LENGHT_TXCAN0;
119.    msgCANobjTx.pui8MsgData = msgDataTx;
120.
121.    }
122.
123.    void
124.    sniff_can0(void){
125.        if(rxFlag) {
126.
127.            msgCANobjRx.pui8MsgData = msgDataRx;
128.            rxFlag = 0;
129.
130.            if(msgCANobjRx.ui32Flags & MSG_OBJ_DATA_LOST) {
131.                UART0Send2("CAN message loss detected.\n\r");
132.            }

```

```

133.         UART0Send2("I: ");
134.         UART0Send4(msgCANObjRx.ui32MsgID);
135.         UART0Send2("\r\n");
136.
137.         UART0Send2("L: ");
138.         UART0Send3(msgCANObjRx.ui32MsgLen);
139.         UART0Send2("\r\n");
140.
141.         UART0Send2("Datos: ");
142.         for(indice_snif=0;indice_snif<msgCANObjRx.ui32MsgLen;ind
ice_snif++){
143.             UART0Send3(msgDataRx[indice_snif]);
144.             UART0Send2(" ");
145.         }
146.         UART0Send2("\r\n");
147.     }
148. }
149.
150. void
151. proc_can0(void){
152.     if(rxFlag) {
153.
154.         msgCANObjRx.pui8MsgData = msgDataRx;
155.         rxFlag = 0;
156.
157.         if(msgCANObjRx.ui32Flags & MSG_OBJ_DATA_LOST) {
158.             UART0Send2("CAN message loss detected.\n\r");
159.         }
160.
161.         proc_canrx(msgDataRx,msgCANObjRx.ui32MsgLen);
162.
163.     }
164. }
165.
166. void
167. proc_canrx(uint8_t *data, uint32_t len){
168.
169.
170.     if(data[0] == KIND_GPIOS_FAM){
171.         uint8_t floor_num      = (data[1]&MASK_BINS)>>SHIFT_BIN
S;
172.         uint8_t arrow_up_flag  = (data[1]&MASK_ARR_UP)>>SHIFT_A
RR_UP;
173.         uint8_t arrow_dw_flag  = (data[1]&MASK_ARR_DW)>>SHIFT_A
RR_DW;
174.         uint8_t maint_flag     = (data[1]&MASK_MAIN)>>SHIFT_MAI
N;
175.
176.         if(maint_flag == 1){
177.             LCD_ShowMaintenance();
178.         }
179.         else{
180.
181.             if(arrow_up_flag == 1){
182.                 LCD_ShowArrowUp();
183.             }
184.             else if(arrow_dw_flag == 1){
185.                 LCD_ShowArrowDw();
186.             }

```

```

187.
188.         LCD_ShowFloor(floor_num);
189.     }
190. }
191.
192.     if(data[0] == KIND_BID){
193.         uint32_t auxbid = (uint32_t)((data[1]<<16)|((data[2]<<8)|
194.         (data[3])));
195.         if(auxbid & CALL_UP_MASK == CALL_UP_MASK) LCD_Confirm_Ca
196.         ll_Up();
197.         if(auxbid & CALL_DW_MASK == CALL_DW_MASK) LCD_Confirm_Ca
198.         ll_Dw();
199.     }
200.
201.
202.     //Funciones para transmision
203.
204.     void asigna_valoresiniciales(id_data * arreglo){
205.         uint8_t i_vi;
206.         for(i_vi=0; i_vi < CANTIDAD_VALORES; i_vi++){
207.             arreglo[i_vi].id = lista_ids[i_vi];
208.             arreglo[i_vi].data = datos[i_vi];
209.         }
210.     }
211.
212.
213.     void asigna_valoreobjetocan(tCANMsgObject *msgfunc, id_data *men
214.     sajес){
215.         (*msgfunc).ui32MsgID = (mensajes)[indice_mensajes].id;
216.         (*msgfunc).pui8MsgData = (mensajes)[indice_mensajes].data;
217.
218.         indice_mensajes++;
219.         if(indice_mensajes == CANTIDAD_VALORES) indice_mensajes = 0;
220.     }
221.
222.     void CANCallUp(void){
223.
224.         uint8_t msgCallUp[4];
225.         CAN_arma_up(FLOOR_NUM, msgCallUp);
226.
227.         msgCANobjTx.ui32MsgID = 0;
228.         msgCANobjTx.ui32MsgLen = 4;
229.         msgCANobjTx.pui8MsgData = msgCallUp;
230.         CANMessageSet(CAN0_BASE, NOBJECT_TXCAN0, &msgCANobjTx, MSG_0
231.         BJ_TYPE_TX);
232.     }
233.
234.     void CANCallDown(void){
235.         uint8_t msgCallDw[4];
236.         CAN_arma_dw(FLOOR_NUM, msgCallDw);
237.
238.         msgCANobjTx.ui32MsgID = 0;
239.         msgCANobjTx.ui32MsgLen = 4;
240.         msgCANobjTx.pui8MsgData = msgCallDw;

```



```

240.         CANMessageSet(CAN0_BASE, NOBJECT_TXCAN0, &msgCANobjTx, MSG_0
    BJ_TYPE_TX);
241.     }
242.
243.     void CAN_arma_up(uint8_t floor, uint8_t *arreglocalls){
244.         arreglocalls[0] = (uint8_t)(KIND_GPIOS_FAM);
245.         if(floor == FIRST_FLOOR){
246.             arreglocalls[1] = 0b00000001;
247.             arreglocalls[2] = 0b00000000;
248.             arreglocalls[3] = 0b00000000;
249.         }
250.         else if (floor == LAST_FLOOR){
251.             arreglocalls[1] = 0b00000000;
252.             arreglocalls[2] = 0b00000000;
253.             arreglocalls[3] = 0b00000000;
254.         }
255.         else{
256.             arreglocalls[1] = 0b00000000;
257.             arreglocalls[2] = 0b00000000;
258.             arreglocalls[3] = 0b00000000;
259.
260.             uint8_t array_num = (FLOOR_NUM-1)/4;
261.
262.             uint8_t array_pos_aux = (FLOOR_NUM-1)%4;
263.
264.             uint8_t array_pos = array_pos_aux*2;
265.
266.             arreglocalls[array_num] = (1<<array_pos);
267.         }
268.     }
269.
270.
271.     void CAN_arma_dw(uint8_t floor, uint8_t *arreglocalls){
272.         arreglocalls[0] = (uint8_t)(KIND_GPIOS_FAM);
273.         if(floor == FIRST_FLOOR){
274.             arreglocalls[1] = 0b00000000;
275.             arreglocalls[2] = 0b00000000;
276.             arreglocalls[3] = 0b00000000;
277.         }
278.         else if (floor == LAST_FLOOR){
279.             arreglocalls[1] = 0b00000000;
280.             arreglocalls[2] = 0b00000000;
281.             arreglocalls[3] = 0b10000000;
282.         }
283.         else{
284.             arreglocalls[1] = 0b00000000;
285.             arreglocalls[2] = 0b00000000;
286.             arreglocalls[3] = 0b00000000;
287.
288.             uint8_t array_pos_aux = (FLOOR_NUM-1)%4;
289.             uint8_t array_num = (FLOOR_NUM-1)/4;
290.
291.             if(array_pos_aux == 0){
292.                 uint8_t array_pos = 0;
293.                 if(array_num != 0){
294.                     array_num -=1;
295.                     array_pos = 7;
296.                 }
297.             }

```

```

298.             arreglocalls[array_num] = (1<<array_pos);
299.         }
300.
301.         else{
302.             uint8_t array_pos = array_pos_aux*2-1;
303.             arreglocalls[array_num] = (1<<array_pos);
304.         }
305.     }
306. }

```

Código fuente 3.- Archivo con el programa de implementación de funciones relacionadas a la comunicación CAN.

Archivo "can_fun.h"

```

1.  /*
2.  * can_func.h
3.  *
4.  * Created on: 3 de dic. de 2016
5.  * Author: Francisco
6.  */
7.
8. #ifndef CAN_FUNC_H_
9. #define CAN_FUNC_H_
10.
11. #include <stdint.h>
12. #include <stdbool.h>
13. #include <string.h>
14.
15. #include "uart_func.h"
16.
17. #include "inc/hw_memmap.h"
18. #include "inc/hw_types.h"
19. #include "inc/hw_can.h"
20. #include "inc/hw_ints.h"
21.
22. #include "driverlib/rom.h"
23. #include "driverlib/rom_map.h"
24. #include "driverlib/can.h"
25. #include "driverlib/interrupt.h"
26. #include "driverlib/sysctl.h"
27. #include "driverlib/gpio.h"
28. #include "driverlib/pin_map.h"
29.
30.
31. #define FLOOR_NUM 2
32.
33. #define FIRST_FLOOR 1
34. #define LAST_FLOOR 7
35.
36. #define SHIFT_FLOOR 2
37. #define CALL_DW_MASK 0b001000000000000000000000
38. #define CALL_UP_MASK 0b010000000000000000000000
39.
40. #define KIND_GPIOS_FAM 2
41. #define KIND_BID 3
42.
43. #define SHIFT_MAIN 0
44. #define SHIFT_ARR_DW 1
45. #define SHIFT_ARR_UP 2
46. #define SHIFT_BINS 3
47.
48. #define MASK_MAIN 0b00000001

```

```

49. #define MASK_ARR_DW 0b00000010
50. #define MASK_ARR_UP 0b00000100
51. #define MASK_BINS 0b11111000
52.
53. #define CAN0_BRATE 125000
54. #define ID_TXCAN0 0x325
55. #define LENGHT_TXCAN0 8
56.
57. //NUMBER OF OBJECTS FOR RX(ONLY FIRST - 1) & TX(UNIQUE OBJ_TX) CAN MESSAGES
58. #define NOBJECT_RXCAN0 1
59. #define NOBJECT_TXCAN0 31
60.
61. //Flags y contadores usados para la recepcion,
62. //errores y cantidad de mensajes transmitidos
63.
64. static volatile bool rxFlag = 0;
65. static volatile bool errFlag = 0;
66. static volatile uint32_t g_ui32MsgTxCount = 0;
67.
68. //Estructuras usadas para la Rx y Tx de objetos
69. static tCANMsgObject msgCANobjRx;
70. static tCANMsgObject msgCANobjTx;
71.
72. //Espacios de memoria reservado para recibir y transmitir
73. static uint8_t msgDataRx[8];
74. static uint8_t msgDataTx[8];
75.
76. //Indice usada en el sniffer
77. static uint8_t indice_snif=0;
78.
79. //Cantidad de valores usados en la transmision
80. #define CANTIDAD_VALORES 1
81.
82. //Valores y variables usadas en transmision CAN
83. //Valores a enviar: ID y datos en paquete
84. static uint32_t lista_ids[CANTIDAD_VALORES] = {
85.     0x324
86. };
87.
88. static uint8_t datos[CANTIDAD_VALORES][8] = {
89.     {0x33,0x00,0x05,0x00,0x01,0x7F,0x02,0x
90.     01}
91. };
92. //Estructura usada para la transmision
93. typedef struct{
94.     uint32_t id;
95.     uint8_t * data;
96. } id_data;
97.
98. //Indice de mensajes para recorrer el arreglo de estructura de paquetes CAN a
99. //transmitir
100. static uint8_t indice_mensajes = 0;
101.
102.
103.
104. //Funciones basicas de configuracion, interrupcion y sniffer
105. extern void CAN0IntHandler(void);
106. extern void conf_can0(void);
107. extern void snif_can0(void);
108.
109. //Asignacion de valores iniciales y objetos can para la transmision
110. extern void asigna_valoresiniciales(id_data * arreglo);
111. extern void asigna_valoreobjetocan(tCANMsgObject *msgfunc, id_data *men
    sajес);

```

```

112.
113.     //Asignacion de valores iniciales y objetos can para la transmision
114.     extern void CANCallUp(void);
115.     extern void CANCallDown(void);
116.
117.     extern void proc_canrx(uint8_t *data, uint32_t len);
118.     void proc_can0(void);
119.     void CAN_arma_up(uint8_t floor, uint8_t *arreglocalls);
120.     void CAN_arma_dw(uint8_t floor, uint8_t *arreglocalls);
121.     #endif /* CAN_FUNC_H_ */

```

Código fuente 4.- Archivo con el programa de cabeceras de funciones relacionadas a la comunicación CAN.

Archivo "colors_fig.h"

```

1.  /*
2.  *  colors_fig.h
3.  *
4.  *  Created on: 16 de dic. de 2016
5.  *      Author: Francisco
6.  */
7.
8.  #ifndef COLORS_FIG_H_
9.  #define COLORS_FIG_H_
10.
11. #define LCD_W 240
12. #define LCD_H 320
13. #define LCD_W_CENTER 128
14. #define COLOR_SIZE 16
15.
16. //Pen color 16 BIT COLOR PALLETTE    // Color Bits = RRRR RGGG GGGB BBB
    B
17. #define WHITE            0xFFFF
18. #define BLACK            0x0000
19. #define BLUE             0x001F // 0000 0000 0001 1111
20. #define BRED             0XF81F
21. #define GRED             0XFFE0
22. #define GBLUE           0X07FF
23. #define RED              0xF800 // 1111 1000 0000 0000
24. #define MAGENTA          0xF81F
25. #define GREEN            0x07E0 // 0000 0111 1110 0000
26. #define CYAN             0x7FFF
27. #define YELLOW           0xFFE0
28. #define BROWN            0XBC40 //Brown
29. #define BRRED            0XFC07 //Brownish red
30. #define GRAY             0X8430 //Gray
31.
32.
33. #define DARKBLUE         0X01CF //Navy blue
34. #define LIGHTBLUE        0X7D7C //Light Blue
35. #define GRAYBLUE         0X5458 //Gray-blue
36.
37.
38. #define LIGHTGREEN        0X841F //Light green
39. #define LGRAY            0XC618 //Light gray (PANNEL), form the backgr
    ound color
40.

```

```

41. #define LGRAYBLUE          0XA651 //Light gray blue (intermediate layer
    color)
42. #define LBBLUE            0X2B12 //Blue light brown (select entry inver
    se)
43.
44. const static uint8_t font_new[2352*11]={
45.
46. 0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,0x0f,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0xff,
    0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,
    0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,
    0xff,0xff,0xff,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0xf8,0xff,0xff,0xff,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0xfe,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x01,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x07,0x00,
    0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
    0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,
    0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,
    0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,
    0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,
    0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,
    0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x0f,
    0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
    0xff,0x1f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,
    0xff,0xff,0xff,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,
    0x3f,0x00,0x00,0xf0,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0xf8,0xff,
    0xff,0xff,0x03,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,
    0xf8,0xff,0xff,0x7f,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0x01,0x00,
    0x00,0x00,0xfc,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,
    0x01,0x00,0x00,0x00,0xfc,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x80,0xff,
    0xff,0xff,0x03,0x00,0x00,0x00,0xfe,0xff,0xff,0x01,0x00,0x00,0x00,0x00,
    0x00,0xfe,0xff,0xff,0x03,0x00,0x00,0x00,0xfe,0xff,0xff,0x00,0x00,0x00,
    0x00,0x00,0x00,0xfc,0xff,0xff,0x07,0x00,0x00,0x00,0xff,0xff,0x7f,0x00,
    0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x07,0x00,0x00,0x00,0xff,0xff,
    0x3f,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x0f,0x00,0x00,0x80,
    0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x0f,0x00,
    0x00,0x80,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,
    0x1f,0x00,0x00,0x80,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xc0,
    0xff,0xff,0x1f,0x00,0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,
    0x00,0x80,0xff,0xff,0x1f,0x00,0x00,0xc0,0xff,0xff,0x07,0x00,0x00,0x00,
    0x00,0x00,0x00,0xff,0xff,0x3f,0x00,0x00,0xc0,0xff,0xff,0x07,0x00,
    0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x3f,0x00,0x00,0xc0,0xff,0xff,
    0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x3f,0x00,0x00,0xc0,
    0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x3f,0x00,
    0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,
    0x7f,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0xfc,0xff,0x7f,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,
    0x00,0x00,0xfc,0xff,0x7f,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,
    0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,0xc0,0xff,0xff,
    0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,0xc0,
    0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,
    0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
    0x7f,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,
    0xf8,0xff,0x7f,0x00,0x00,0xc0,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x07,0x00,0x00,0x00,0x00,

```


47.

[illegible]

0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x0f,0x00,0x00,0x00,0x00,0xf8,
0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,
0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,
0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,
0x0f,0x00,0x00,0x00,0x00,0xf0,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,
0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0x3f,0x00,0x00,0x00,0x00,
0x00,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,0x00,0x00,
0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xe0,
0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0x03,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x80,
0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,
0x01,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x80,
0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x00,0x00,0x00,0x00,
0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0xf8,
0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,0x00,0x00,
0x00,0xfc,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,
0x00,0x00,0x00,0xfe,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,
0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0x7f,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,
0x00,0x00,0xfc,0xff,0xff,0x00,0x00,0x00,0x80,0xff,0xff,0x1f,0x00,0x00,
0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x03,0x00,0x00,0xe0,0xff,0xff,0x0f,
0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x07,0x00,0x00,0xf0,0xff,
0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x1f,0x00,0x00,
0xfc,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x3f,
0x00,0x00,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,
0xff,0xff,0x01,0xc0,0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,
0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,
0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0x7f,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0xff,
0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,
0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,
0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,
0xff,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0x7f,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,0x1f,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,
0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,
0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xfe,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,

49. 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0x03,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,
0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xf0,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xf0,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x03,0x00,0x00,0x00,0x00,0x00,

[illegible]

50. 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x03,0x00
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,
0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xff,0xff,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0xff,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,
0xff,0xff,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,

0x00,0x00,0xf8,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x01,0x00,0x00,
0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x03,
0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0x3f,
0x00,0xc0,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0xff,0x07,0x00,0x00,0xfe,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0xff,0x00,0x00,0x00,0xf0,0xff,0xff,0x3f,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0x7f,0x00,0x00,0x00,0xe0,0xff,0xff,0x3f,0x00,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x1f,0x00,0x00,0x00,0x80,0xff,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0xff,
0xff,0x7f,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0xfe,0xff,0xff,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0xfc,0xff,0xff,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,
0x01,0x00,0x00,0x00,0xf8,0xff,0xff,0x01,0x00,0x00,0x00,0xf8,
0xff,0xff,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x01,0x00,0x00,0x00,
0x00,0xf8,0xff,0x7f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0xfc,0xff,0x7f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,
0x03,0x00,0x00,0x00,0xfc,0xff,0x3f,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0x03,0x00,0x00,0x00,0xf0,0xff,0x1f,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,
0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0xfe,
0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,
0x00,0x00,0x00,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xff,0xff,0x07,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0xff,
0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,
0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0xfe,0x07,0x00,0x00,0x00,0x00,0x00,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x01,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,
0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xfe,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x3f,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x3f,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x1f,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,

[illegible]

```
51. 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,0x00,0x00,  
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,  
    0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,  
    0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
    0xfe,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
    0x00,0x80,0xff,0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,  
    0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xf,0x00,0x00,0x00,0x00,  
    0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,  
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,0xff,0x7f,  
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0xff  
    0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,  
    0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,  
    0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,  
    0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf,0x00,0x00,0x00,0x00,0x00,  
    0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf,0x00,0x00,0x00,  
    0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,  
    0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0xff,0xff,0xff,  
    0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0x3f,0x00,0xf0,  
    0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0x07,  
    0x00,0x80,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,  
    0xff,0x01,0x00,0x00,0xfe,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,  
    0xff,0xff,0xff,0x00,0x00,0x00,0xfc,0xff,0xff,0x03,0x00,0x00,0x00,0x00,
```


[illegible]

0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,0xfe,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0xfe,0xff,0x07,0x00
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0xfe,0xff
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,
0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,
0xff,0x0f,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xff,0xff,0x07,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xff,0xff,0x07,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x03,0xfe,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0xfe,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0xfe,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x00,
0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,
0x7f,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xf8,0xff,0x3f,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xf8,0xff,0x3f,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0xfe,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0xfe,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,
0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0x03,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0x01,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0x01,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,
0x00,0x00,0xe0,0xff,0xff,0x00,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,0x00,0x00,0xfe,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,0x00,0x00,0xfe,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0xfe,0xff,
0x07,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0xfe,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0xfe,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,
0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,
0x00,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0x03,0x00,0x00,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x80,
0xff,0xff,0x03,0x00,0x00,0x00,0xfe,0xff,0x07,0x00,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xfe,0xff,0x07,0x00,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x00,0x00,0xfe,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,0x00,0x00,0x00,0x00,0xfe,0xff

[illegible]

[illegible]

[illegible]

0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,
0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xff,0xff,0x0f,0x00,0x00,
0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xff,0xff,0x0f,
0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,
0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,
0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,
0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,
0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,
0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,
0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfc,0xff,0x0f,
0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfc,
0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,
0x00,0xfc,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,
0x00,0x00,0x00,0xfc,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
0x3f,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,
0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xf8,0xff,0x1f,0x00,0x00,0x00,0x00,
0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xf8,0xff,0x1f,0x00,0x00,
0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0xf8,0xff,0x1f,
0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,0xf8,
0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,
0x00,0xf0,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,
0x00,0x00,0x00,0xf0,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,
0xff,0x1f,0x00,0x00,0x00,0xf0,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,
0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0xe0,0xff,0x7f,0x00,0x00,0x00,
0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0x00,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,0xc0,
0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x03,0x00,0x00,0x00,0xc0,0xff,0xff,
0x03,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0xe0,
0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x0f,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,
0x00,0x00,0x00,0xf0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0xfe,0xff,
0x3f,0x00,0x00,0x00,0xf8,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,
0xfe,0xff,0x3f,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0x00,0x00,0x00,0x00,
0x00,0x00,0xfc,0xff,0xff,0x00,0x00,0x00,0x00,0xfe,0xff,0x7f,0x00,0x00,
0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0x01,0x00,0x00,0x00,0xff,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x03,0x00,0x00,0xc0,0xff,
0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x0f,0x00,0x00,
0xf0,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x3f,
0x00,0x00,0xfc,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,
0xff,0xff,0x01,0x00,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,
0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,
0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x03,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x01,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0xff,0xff,
0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,
0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,
0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,
0xff,0xff,0xff,0xff,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,
0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,
0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

[illegible]

56. 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,
0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,

0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xf8,0xff,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xfe,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0xff,
0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,
0xff,0xff,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,
0x80,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,
0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x01,
0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0x03,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0x00,0x80,
0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x1f,
0x00,0x00,0xfe,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
0xff,0x07,0x00,0x00,0xf8,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0xff,0x03,0x00,0x00,0xe0,0xff,0xff,0x1f,0x00,0x00,0x00,
0x00,0x00,0xfc,0xff,0xff,0x00,0x00,0x00,0xc0,0xff,0xff,0x1f,0x00,0x00,
0x00,0x00,0x00,0x00,0xfe,0xff,0x7f,0x00,0x00,0x00,0x80,0xff,0xff,0x3f,
0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x3f,0x00,0x00,0x00,0xff,
0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,0x00,
0x00,0xfe,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,
0x00,0x00,0x00,0xfc,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x80,0xff,
0x0f,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,0x00,0x00,0x80,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x00,0x00,0x00,
0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x03,0x00,0x00,0x00,0xf0,0xff,
0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,0xe0,
0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,
0x00,0xe0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,
0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,
0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,
0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,
0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,
0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,
0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,
0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,
0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,
0x00,0xe0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x01,0x00,0x00,0x00,0x80,0xff,
0x03,0x00,0x00,0x00,0xe0,0xff,0xff,0x00,0x00,0x00,0x80,
0xff,0xff,0x07,0x00,0x00,0x00,0xf0,0xff,0xff,0x00,0x00,0x00,
0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0xf0,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x0f,0x00,0x00,0x00,0xf8,0xff,
0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,0x00,0x00,0xfc,
0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,0x00,
0x00,0xfc,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x3f,0x00,
0x00,0x00,0x00,0xfe,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,
0x7f,0x00,0x00,0x00,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0xff,0x00,0x00,0x00,0x80,0xff,0xff,0x1f,0x00,0x00,0x00,

0x00,0x00,0xf8,0xff,0xff,0x03,0x00,0x00,0xe0,0xff,0xff,0x0f,0x00,0x00,
0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x07,0x00,0x00,0xf0,0xff,0xff,0x0f,
0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x1f,0x00,0x00,0xfc,0xff,
0xff,0x07,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0x00,0x80,
0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x80,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xfe,0xff,0xff,0xff,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xfc,0xff,0xff,0xff,0xff,0xff,0xff,0x1f,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0x0f,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,
0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,
0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,
0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xfc,0xff,0xff,0xff,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,
0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0x7f,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0xff,0x7f,0x00,0x80,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,
0x00,0x00,0xfe,0xff,0xff,0x0f,0x00,0x00,0xfc,0xff,0xff,0x1f,0x00,0x00,
0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0x03,0x00,0x00,0xf0,0xff,0xff,0x3f,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0x00,0x00,0x00,0xc0,0xff,
0xff,0x7f,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x7f,0x00,0x00,0x00,
0x00,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x1f,0x00,
0x00,0x00,0x00,0xfe,0xff,0xff,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,
0x0f,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0x01,0x00,0x00,0x00,0xe0,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x01,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0xf0,0xff,0xff,0x03,0x00,0x00,0x00,0xe0,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x01,0x00,0x00,0x00,0xc0,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x00,0x00,0x00,0x00,
0x00,0x80,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x0f,0x00,0x00,0x00,0xfc,
0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x1f,0x00,0x00,0x00,
0x00,0xfc,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,
0x00,0x00,0x00,0xfc,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,
0x1f,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,0x00,
0x00,0x00,0xfc,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0xfe,
0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,
0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
0x3f,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,
0xf8,0xff,0x3f,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,
0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,
0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0xfe,
0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,
0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,
0x3f,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,

0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,
0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,
0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,
0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xfe,
0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,
0x00,0xfe,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x3f,0x00,
0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,
0x3f,0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x3f,0x00,0x00,0x00,
0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,0x00,0x00,0xfc,0xff,0x3f,0x00,0x00,
0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,0x00,0x00,0xfc,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0x1f,0x00,0x00,0x00,0x00,0xf8,
0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x0f,0x00,0x00,0x00,
0x00,0xf8,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x0f,0x00,
0x00,0x00,0x00,0xf0,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0x0f,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x03,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x01,0x00,0x00,0x00,
0x00,0x80,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0xff,0xff,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0x01,0x00,0x00,0x80,0xff,
0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0x07,0x00,0x00,
0xe0,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0x1f,
0x00,0x00,0xf8,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,
0xff,0xff,0x00,0x00,0xff,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,
0xf8,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,
0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0x03,
0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,
0xff,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xf8,0xff,0xff,0xff,0xff,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0xff,0x7f,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,
0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,
0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xff,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xf0,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

57. 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,
0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0xff,0xff,0xff,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,
0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,

0x00,0xfc,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x3f,0x00,
0x00,0x00,0x00,0xfc,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0x3f,0x00,0x00,0x00,0x00,0xfc,0xff,0x7f,0x00,0x00,0x00,0x00,0x80,
0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x00,0x00,0x00,0x00,
0x00,0xc0,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x00,0x00,
0x00,0x00,0x00,0xc0,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,
0x01,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xf0,
0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xf0,0xff,0xff,0x3f,0x00,0x00,0x00,
0x00,0xf0,0xff,0xff,0x07,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0x3f,0x00,
0x00,0x00,0x00,0xe0,0xff,0xff,0x0f,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,
0x3f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0xfe,
0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x3f,0x00,0x00,0x00,
0x00,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x7f,0x00,
0x00,0x00,0x80,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0xff,0x01,0x00,0x00,0xc0,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x80,
0xff,0xff,0xff,0x03,0x00,0x00,0xf0,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,
0x00,0x00,0xff,0xff,0xff,0x1f,0x00,0x00,0xfc,0xff,0xff,0xff,0x3f,0x00,
0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0x00,0x80,0xff,0xff,0xff,0xff,
0xf7,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xf3,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xf3,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xf0,0xff,
0xff,0xff,0xff,0xff,0xff,0xf1,0xff,0x3f,0x00,0x00,0x00,0x00,0xf0,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xf0,0xff,0x3f,0x00,0x00,0x00,
0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,0xf7,0xf8,0xff,0x3f,0x00,
0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0xff,0x3f,0xf8,0xff,
0x3f,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0xff,0xff,0xff,0xff,0x1f,
0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0xff,0xff,
0xff,0x0f,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0xff,
0xff,0xff,0xff,0x03,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xf8,
0xff,0xff,0xff,0xff,0xff,0x01,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0xff,0xff,0x7f,0x00,0xf8,0xff,0x3f,0x00,0x00,
0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0x3f,0x00,0xf8,0xff,0x1f,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,0x0f,0x00,0xf8,0xff,
0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0xf8,0xff,0xff,0xff,0x03,0x00,
0xf8,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x7f,
0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,
0xff,0x0f,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,0x1f,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,
0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0xfe,0xff,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x07,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xff,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x80,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,
0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x03,0x00,0x00,0x00,0xf8,
0xff,0x3f,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x03,0x00,0x00,0x00,
0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0x01,0x00,
0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,
0x01,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,0x00,0x00,0xe0,

```

0xff,0xff,0x01,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,0x00,0x00,
0x00,0xe0,0xff,0xff,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,
0x00,0x00,0x00,0xf0,0xff,0xff,0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,
0x00,0x00,0x00,0x00,0xf0,0xff,0x7f,0x00,0x00,0x00,0x00,0xf0,
0xff,0xff,0x00,0x00,0x00,0x00,0xf8,0xff,0x7f,0x00,0x00,0x00,
0x00,0xf0,0xff,0xff,0x00,0x00,0x00,0x00,0xf8,0xff,0x3f,0x00,0x00,
0x00,0x00,0x00,0xf0,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0xfc,0xff,0x3f,
0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x01,0x00,0x00,0x00,0xfe,
0xff,0x1f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x03,0x00,0x00,0x00,
0x00,0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0x03,0x00,
0x00,0x00,0x00,0xff,0xff,0xf,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,
0x07,0x00,0x00,0x00,0x80,0xff,0xff,0xf,0x00,0x00,0x00,0x00,0xc0,
0xff,0xff,0xf,0x00,0x00,0x00,0xc0,0xff,0xff,0x07,0x00,0x00,0x00,
0x00,0x80,0xff,0xff,0x1f,0x00,0x00,0x00,0xe0,0xff,0xff,0x07,0x00,0x00,
0x00,0x00,0x00,0x80,0xff,0xff,0x3f,0x00,0x00,0x00,0xf8,0xff,0xff,0x03,
0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x7f,0x00,0x00,0x00,0xfc,0xff,
0xff,0x01,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0x01,0x00,0x00,
0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0xfe,0xff,0xff,0x03,
0x00,0xc0,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,
0xff,0x1f,0x00,0xf8,0xff,0xff,0x7f,0x00,0x00,0x00,0x00,0x00,
0xfc,0xff,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,
0x00,0xf8,0xff,0xff,0xff,0xff,0xff,0xff,0x1f,0x00,0x00,0x00,
0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xf,0x00,
0x00,0x00,0x00,0x00,0xe0,0xff,0xff,0xff,0xff,0xff,0xff,
0x07,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,
0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
0xff,0xff,0xff,0xff,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,
0xff,0xff,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0xfe,0xff,0xff,0xff,0xff,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xf0,0xff,0xff,0xff,0xff,0xff,0x07,0x00,0x00,
0x00,0x00,0x00,0x00,0xc0,0xff,0xff,0xff,0xff,0xff,0x01,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0xff,
0x7f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0xff,
0xff,0xff,0x1f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0xe0,0xff,0xff,0xff,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0xfe,0xff,0x3f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,

```

```

58.
59.
60.
61. };
62.
63. #endif /* COLORS_FIG_H_ */

```

Código fuente 5.- Archivo con el programa de cabecera para colores y figuras.

Archivo "ges_sensor.c"

```

1. /*
2.  * ges_sensor.c
3.  *
4.  * Created on: 2 de dic. de 2016
5.  * Author: Francisco
6.  */
7.
8. #include "ges_sensor.h"
9. #include "uart_func.h"

```

```

10. #include "lcd_func.h"
11. #include "can_func.h"
12.
13. #define TARGET_IS_BLIZZARD_RB1
14. #include <stdint.h>
15. #include <stdbool.h>
16. #include <string.h>
17.
18. #include "colors_fig.h"
19.
20. #include "inc/hw_ints.h"
21. #include "inc/hw_memmap.h"
22. #include "inc/hw_types.h"
23. #include "inc/hw_gpio.h"
24.
25. #include "driverlib/systick.h"
26. #include "driverlib/debug.h"
27. #include "driverlib/fpu.h"
28. #include "driverlib/gpio.h"
29. #include "driverlib/interrupt.h"
30. #include "driverlib/pin_map.h"
31. #include "driverlib/rom.h"
32. #include "driverlib/sysctl.h"
33. #include "driverlib/uart.h"
34. #include "driverlib/rom_map.h"
35.
36. uint8_t gest_det=0;
37.
38.
39. void gest_hoverleft(void){
40.     UART0Send2("HLEFT\r\n");
41.     LCD_FillTriangle(67,10,50,100,GREEN);
42.     if(piso_actual < 11) piso_actual++;
43.
44.     LCD_ShowNum_2(61,85,piso_actual); //Confirm call up
45.     CANCallUp();
46.     gest_det = 1;
47.     cron = time;
48. }
49.
50. void gest_hoverright(void){
51.     UART0Send2("HRIGHT\r\n");
52.     LCD_FillTriangleD(67,250,50,100,GREEN);
53.     if(piso_actual > 1) piso_actual--;
54.
55.     LCD_ShowNum_2(61,85,piso_actual); //Confirm call down
56.     CANCallDown();
57.     gest_det = 1;
58.     cron = time;
59. }
60.
61. void gest_hovercenter(void){
62.     UART0Send2("HCENT\r\n");
63.     LCD_FillTriangle(67,10,50,100,YELLOW);
64.     LCD_FillTriangleD(67,250,50,100,YELLOW);
65.     gest_det = 1;
66.     cron = time;
67.     //LCD_CallClean();
68.     //LCD_ShowNum_2(61,85,3); //Confirm cancelation

```

```

69. }
70.
71. void gest_swleft(void){
72.     UART0Send2("SLEFT\r\n");
73. }
74.
75. void gest_swright(void){
76.     UART0Send2("SRIGHT\r\n");
77. }
78.
79. void gest_swcenter(void){
80.     UART0Send2("SCENT\r\n");
81. }

```

Código fuente 6.- Archivo con el programa de implementación de funciones relacionadas al procesamiento o acciones a gestos.

Archivo "ges_sensor.h"

```

1. #ifndef DRIVERS_GES_SENSOR_H_
2. #define DRIVERS_GES_SENSOR_H_
3.
4. #define TARGET_IS_BLIZZARD_RB1
5. #include <stdint.h>
6. #include <stdbool.h>
7. #include <string.h>
8.
9. #include "inc/hw_ints.h"
10. #include "inc/hw_memmap.h"
11. #include "inc/hw_types.h"
12. #include "inc/hw_gpio.h"
13.
14. #include "driverlib/systick.h"
15. #include "driverlib/debug.h"
16. #include "driverlib/fpu.h"
17. #include "driverlib/gpio.h"
18. #include "driverlib/interrupt.h"
19. #include "driverlib/pin_map.h"
20. #include "driverlib/rom.h"
21. #include "driverlib/sysctl.h"
22. #include "driverlib/uart.h"
23. #include "driverlib/rom_map.h"
24.
25. //CMD Type
26. #define IRCMD_LOWLIMIT 0xF0 //Minimum value for a command sent by the GestureSensor
27. #define IRCMD_ID 0xF1 //Value for the ID message
28. #define IRCMD_GESTURE 0xFC //Value for the Gesture message
29. #define IRCMD_ZCOOR 0xFB //Value for the ZCoordinate message
30. #define IRCMD_XCOOR 0xFA //Value for the XCoordinate message
31. #define IRCMD_RANGES 0xFE //Value for the Ranges message
32. #define IRCMD_PENUP 0xFF //Value for the PenUp message
33.
34. //Gesture 2nd Byte
35. #define GES_SWRG 0x01 //Value for the swipe right gesture
36. #define GES_SWLF 0x02 //Value for the swipe left gesture
37. #define GES_SWDW 0x03 //Value for the swipe down gesture
38. #define GES_HVMV 0x15 //Value for the hover and move gesture

```

```

39.
40. //Hover Options
41. #define HVMV_center    0x00  //Value for the center hover and move
42. #define HVMV_right     0x01  //Value for the right hover and move
43. #define HVMV_left      0x02  //Value for the left hover and move
44.
45. //Sizes definition
46. #define LIM_RAN_SZ      0x03
47. #define LIM_GES_HOVMOV  0x03
48. #define LIM_GES_SWMOV   0x03
49. #define LIM_GES_ZCOORD  0x02
50.
51. //Functions per hover
52. extern void gest_hoverleft(void);
53. extern void gest_hoverright(void);
54. extern void gest_hovercenter(void);
55.
56. //Functions per swipe
57. extern void gest_swleft(void);
58. extern void gest_swright(void);
59. extern void gest_swcenter(void);
60.
61. //Counter of hover
62. static uint8_t hv_count=0;
63. static uint8_t hv_right=0;
64. static uint8_t hv_left=0;
65. static uint8_t hv_center=0;
66.
67. static uint8_t hv_right_ctr=0;
68. static uint8_t hv_left_ctr=0;
69.
70. //Gesture detected
71. extern uint8_t gest_det;
72.
73. //Gesture detected
74. extern uint16_t piso_actual;
75.
76. //Z-COORD first and current
77. static uint8_t zcoor_first = 0;
78. static uint8_t zcoor_cur   = 0;
79.
80. //Limit of counter for hover
81. #define LIM_HOVER 3
82. #define LIM_RIGHT_CTR 1
83. #define LIM_LEFT_CTR 1
84. #define LIM_HOVCENT 0x10
85. //Flag of hover
86.
87. //Kind of CAN messages
88. #define KIND_CALL 1
89. #define KIND_GPIOS_FAM 2
90. #define KIND_BID 3
91.
92.
93. //*****
94. //
95. // Mark the end of the C bindings section for C++ compilers.
96. //

```

```

97. //*****
    *****
98. #ifdef __cplusplus
99. }
100.     #endif
101.
102.     #endif

```

Código fuente 7.- Archivo con el programa de cabeceras de funciones relacionadas al procesamiento de gestos.

Archivo "lcd_func.c"

```

1.  /*
2.   * lcd_func.c
3.   *
4.   * Created on: 10 de dic. de 2016
5.   * Author: Francisco
6.   */
7.
8. #include <stdint.h>
9. #include <stdbool.h>
10.
11. #include "lcd_func.h"
12. #include "colors_fig.h"
13.
14. #include "inc/hw_memmap.h"
15. #include "inc/hw_types.h"
16. #include "inc/hw_gpio.h"
17.
18. #include "driverlib/rom.h"
19. #include "driverlib/rom_map.h"
20. #include "driverlib/sysctl.h"
21. #include "driverlib/gpio.h"
22. #include "driverlib/ssi.h"
23. #include "driverlib/pin_map.h"
24.
25. void conf_lcd(void){
26.     conf_pineslcd();
27.     ini_lcd();
28. }
29.
30.
31. void conf_pineslcd(void){
32.
33.     #ifndef PERIPHERAL_GPIOA_ACTIVE
34.     #define PERIPHERAL_GPIOA_ACTIVE
35.         MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // SSI pins:
36.         Conf SSI0(MOSI(PA5)-MISO(PA4)-SCLK(PA2)-CS(PD2))
37.     #endif
38.     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); // D/C' pin: PE1
39.
40.     MAP_SysCtlDelay(MAP_SysCtlClockGet()/30); // 100mS delay
41.
42.     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
43.

```



```

44.     MAP_GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, DC | CS | RESET); //D/Cb
    ar: pin1 E1, CS: pin3 E2, RESET: pin2 E2
45.
46.     MAP_SSIDisable(SSIO_BASE);
47.     MAP_GPIOPinConfigure(GPIO_PA2_SSIOCLK);
48.     MAP_GPIOPinConfigure(GPIO_PA4_SSIO_RX);
49.     MAP_GPIOPinConfigure(GPIO_PA5_SSIO_TX);
50.
51.     MAP_GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_2| GPIO_PIN_4| GPIO_P
    IN_5);
52.
53.     MAP_SSI_ClockSourceSet(SSIO_BASE, MAP_SysCtlClockGet()/60); //50ms
    delay
54.
55.     MAP_SSIConfigSetExpClk(SSIO_BASE, MAP_SysCtlClockGet(), SSI_FRF_MO
    TO_MODE_2, SSI_MODE_MASTER, 4000000, 8); // defines base, System clk, M
    ode 0 = SPH = SPO = 0, Master, 400 KHz, no. of bits = 8 = 1 byte transf
    er
56.
57.     MAP_SSIEnable(SSIO_BASE); // enables SSI
58.
59. }
60.
61. void ini_lcd(void){
62.     uint32_t clockSys = MAP_SysCtlClockGet();
63.
64.     //TFT_CS =1;
65.     GPIOPinWrite(GPIO_PORTE_BASE, CS, CS);
66.
67.     //TFT_RST=1;
68.     GPIOPinWrite(GPIO_PORTE_BASE, RESET, RESET);
69.     SysCtlDelay(clockSys/600); // 5 ms delay
70.
71.     //TFT_RST=0;
72.     GPIOPinWrite(GPIO_PORTE_BASE, RESET, 0);
73.     SysCtlDelay(clockSys/60); // 15 ms delay
74.
75.     //TFT_RST=1;
76.     GPIOPinWrite(GPIO_PORTE_BASE, RESET, RESET);
77.     SysCtlDelay(clockSys/60); // 15 ms delay
78.
79.     //TFT_CS =1;
80.     GPIOPinWrite(GPIO_PORTE_BASE, CS, CS);
81.     SysCtlDelay(clockSys/100); // 15 ms delay
82.
83.     //TFT_CS =0;
84.     GPIOPinWrite(GPIO_PORTE_BASE, CS, 0);
85.
86.     LCD_WR_REG_SSI(0xCB);
87.     LCD_WR_DATA8_SSI(0x39);
88.     LCD_WR_DATA8_SSI(0x2C);
89.     LCD_WR_DATA8_SSI(0x00);
90.     LCD_WR_DATA8_SSI(0x34);
91.     LCD_WR_DATA8_SSI(0x02);
92.
93.     LCD_WR_REG_SSI(0xCF);
94.     LCD_WR_DATA8_SSI(0x00);
95.     LCD_WR_DATA8_SSI(0xC1);
96.     LCD_WR_DATA8_SSI(0x30);

```

```

97.
98.     LCD_WR_REG_SSI(0xE8);
99.     LCD_WR_DATA8_SSI(0x85);
100.         LCD_WR_DATA8_SSI(0x00);
101.         LCD_WR_DATA8_SSI(0x78);
102.
103.         LCD_WR_REG_SSI(0xEA);
104.         LCD_WR_DATA8_SSI(0x00);
105.         LCD_WR_DATA8_SSI(0x00);
106.
107.         LCD_WR_REG_SSI(0xED);
108.         LCD_WR_DATA8_SSI(0x64);
109.         LCD_WR_DATA8_SSI(0x03);
110.         LCD_WR_DATA8_SSI(0X12);
111.         LCD_WR_DATA8_SSI(0X81);
112.
113.         LCD_WR_REG_SSI(0xF7);
114.         LCD_WR_DATA8_SSI(0x20);
115.
116.         LCD_WR_REG_SSI(0xC0);           //Power control
117.         LCD_WR_DATA8_SSI(0x23);       //VRH[5:0]
118.
119.         LCD_WR_REG_SSI(0xC1);           //Power control
120.         LCD_WR_DATA8_SSI(0x10);       //SAP[2:0];BT[3:0]
121.
122.         LCD_WR_REG_SSI(0xC5);           //VCM control
123.         LCD_WR_DATA8_SSI(0x3e);
124.         LCD_WR_DATA8_SSI(0x28);
125.
126.         LCD_WR_REG_SSI(0xC7);           //VCM control2
127.         LCD_WR_DATA8_SSI(0x86);       //--
128.
129.         LCD_WR_REG_SSI(0x36);           // Memory Access Control
130.         LCD_WR_DATA8_SSI(0x48);
131.
132.         LCD_WR_REG_SSI(0x3A);           //Pixel format set
133.         LCD_WR_DATA8_SSI(0x55);
134.
135.         LCD_WR_REG_SSI(0xB1);           //Frame rate control in full col
or
136.         LCD_WR_DATA8_SSI(0x00);
137.         LCD_WR_DATA8_SSI(0x18);
138.
139.         LCD_WR_REG_SSI(0xB6);           // Display Function Control
140.         LCD_WR_DATA8_SSI(0x08);
141.         LCD_WR_DATA8_SSI(0x82);
142.         LCD_WR_DATA8_SSI(0x27);
143.
144.         LCD_WR_REG_SSI(0xF2);           // 3Gamma Function Disable
145.         LCD_WR_DATA8_SSI(0x00);
146.
147.         LCD_WR_REG_SSI(0x26);           //Gamma curve selected
148.         LCD_WR_DATA8_SSI(0x01);
149.
150.         LCD_WR_REG_SSI(0xE0);           //Set Gamma
151.         LCD_WR_DATA8_SSI(0x0F);
152.         LCD_WR_DATA8_SSI(0x31);
153.         LCD_WR_DATA8_SSI(0x2B);
154.         LCD_WR_DATA8_SSI(0x0C);

```

```

155.     LCD_WR_DATA8_SSI(0x0E);
156.     LCD_WR_DATA8_SSI(0x08);
157.     LCD_WR_DATA8_SSI(0x4E);
158.     LCD_WR_DATA8_SSI(0xF1);
159.     LCD_WR_DATA8_SSI(0x37);
160.     LCD_WR_DATA8_SSI(0x07);
161.     LCD_WR_DATA8_SSI(0x10);
162.     LCD_WR_DATA8_SSI(0x03);
163.     LCD_WR_DATA8_SSI(0x0E);
164.     LCD_WR_DATA8_SSI(0x09);
165.     LCD_WR_DATA8_SSI(0x00);
166.
167.     LCD_WR_REG_SSI(0XE1);           //Set Gamma
168.     LCD_WR_DATA8_SSI(0x00);
169.     LCD_WR_DATA8_SSI(0x0E);
170.     LCD_WR_DATA8_SSI(0x14);
171.     LCD_WR_DATA8_SSI(0x03);
172.     LCD_WR_DATA8_SSI(0x11);
173.     LCD_WR_DATA8_SSI(0x07);
174.     LCD_WR_DATA8_SSI(0x31);
175.     LCD_WR_DATA8_SSI(0xC1);
176.     LCD_WR_DATA8_SSI(0x48);
177.     LCD_WR_DATA8_SSI(0x08);
178.     LCD_WR_DATA8_SSI(0x0F);
179.     LCD_WR_DATA8_SSI(0x0C);
180.     LCD_WR_DATA8_SSI(0x31);
181.     LCD_WR_DATA8_SSI(0x36);
182.     LCD_WR_DATA8_SSI(0x0F);
183.
184.     LCD_WR_REG_SSI(0x11);           //Exit Sleep
185.
186.     SysCtlDelay(clockSys/15);       // 200 ms delay
187.
188.     LCD_WR_REG_SSI(0x29);           //Display on
189.     LCD_WR_REG_SSI(0x2c);           //Memory write
190.
191.     SysCtlDelay(clockSys/30);       // 100ms delay
192.
193.     LCD_Clear(BLACK);
194.     BACK_COLOR=BLUE;
195.     POINT_COLOR=WHITE;
196. }
197.
198.     ////Clear screen function
199.     ////Color:To clear the screen filled with color
200. void LCD_Clear(uint16_t Color)
201. {
202.     uint8_t VH,VL;
203.     uint16_t i,j;
204.     VH=Color>>8;
205.     VL=Color;
206.     Address_set(0,0,LCD_W,LCD_H);
207.     for(i=0;i<LCD_W;i++)
208.     {
209.         for (j=0;j<LCD_H;j++)
210.         {
211.             LCD_WR_DATA8_SSI(VH);
212.             LCD_WR_DATA8_SSI(VL);
213.         }

```

```

214.         }
215.     }
216.
217.     ///Clear screen function
218.     ///Color:To clear the screen filled with color
219.     void LCD_CallUp(uint16_t Color)
220.     {
221.         uint8_t VH,VL;
222.         uint16_t i,j;
223.         Address_set(0,0,LCD_W,LCD_H/2);
224.         for(i=0;i<LCD_H/2;i++){
225.             if(i< LCD_H/4){
226.                 VH=Color>>8;
227.                 VL=Color;
228.             }
229.
230.             else{
231.                 VH=0x00;
232.                 VL=0x00;
233.             }
234.
235.             for (j=0;j<LCD_W;j++){
236.                 LCD_WR_DATA8_SSI(VH);
237.                 LCD_WR_DATA8_SSI(VL);
238.             }
239.         }
240.     }
241. }
242.
243.
244. void LCD_CallDown(uint16_t Color)
245. {
246.     uint8_t VH,VL;
247.     uint16_t i,j;
248.
249.     Address_set(0,0,LCD_W,LCD_H/2);
250.
251.     for(i=0;i<LCD_H/2;i++){
252.         {
253.             if(i> LCD_H/4){
254.                 VH=Color>>8;
255.                 VL=Color;
256.             }
257.             else{
258.                 VH=0x00;
259.                 VL=0x00;
260.             }
261.
262.             for (j=0;j<LCD_W;j++){
263.                 LCD_WR_DATA8_SSI(VH);
264.                 LCD_WR_DATA8_SSI(VL);
265.             }
266.         }
267.     }
268. }
269.
270. void LCD_CallClean(void){
271.     uint16_t i,j;
272.     Address_set(0,0,LCD_W,LCD_H/2);

```

```

273.
274.     for(i=0;i<LCD_H/2;i++)
275.     {
276.         for (j=0;j<LCD_W;j++)
277.         {
278.             LCD_WR_DATA8_SSI(0x00);
279.             LCD_WR_DATA8_SSI(0x00);
280.         }
281.     }
282.
283. }
284.
285. void Address_set(uint16_t x1,uint16_t y1,uint16_t x2,uint16_t y2
    )
286. {
287.     LCD_WR_REG_SSI(0x2a);
288.     LCD_WR_DATA8_SSI(x1>>8);
289.     LCD_WR_DATA8_SSI(x1);
290.     LCD_WR_DATA8_SSI(x2>>8);
291.     LCD_WR_DATA8_SSI(x2);
292.
293.     LCD_WR_REG_SSI(0x2b);
294.     LCD_WR_DATA8_SSI(y1>>8);
295.     LCD_WR_DATA8_SSI(y1);
296.     LCD_WR_DATA8_SSI(y2>>8);
297.     LCD_WR_DATA8_SSI(y2);
298.
299.     LCD_WR_REG_SSI(0x2c);
300. }
301.
302. void LCD_WR_DATA8_SSI(uint8_t data) //Send -
    8 bit parameter data
303. {
304.     //DC=1;
305.     GPIOPinWrite(GPIO_PORTE_BASE, DC,DC); //Pulses up the dc lin
        e
306.     LCD_Writ_Bus(data);
307. }
308.
309. void LCD_WR_DATA16_SSI(uint16_t data16)
310. {
311.     //DC=1;
312.     GPIOPinWrite(GPIO_PORTE_BASE, DC,DC); //Pulses up the dc lin
        e
313.     LCD_Writ_Bus(data16>>8);
314.     LCD_Writ_Bus(data16);
315. }
316.
317. void LCD_WR_REG_SSI(uint8_t data)
318. {
319.     //DC=0;
320.     GPIOPinWrite(GPIO_PORTE_BASE, DC,0); //Pulses down the dc li
        ne
321.     LCD_Writ_Bus(data);
322. }
323.
324. void LCD_Writ_Bus(uint8_t data)
325. {
326.     SSIDataPut(SSIO_BASE,data);

```

```

327.         while(SSIBusy(SSIO_BASE));
328.     }
329.
330.     void LCD_ScrollingDef(uint16_t topfix, uint16_t heightscroll, ui
nt16_t botfix){
331.         LCD_WR_REG_SSI(0x33);    //Set Scrolling Def
332.         LCD_WR_DATA16_SSI(topfix);
333.         LCD_WR_DATA16_SSI(heightscroll);
334.         LCD_WR_DATA16_SSI(botfix);
335.     }
336.
337.     void LCD_ScrollingStart(uint16_t startline){
338.         LCD_WR_REG_SSI(0x37);    //Set Scrolling Def
339.         LCD_WR_DATA16_SSI(startline);
340.     }
341.
342.
343.
344.     void LCD_ShowNum_2(uint16_t x,uint16_t y,uint16_t num)
345.     {
346.         uint8_t t=0;
347.         POINT_COLOR = WHITE;
348.         LCD_ShowChar_2(x+LCD_W_CENTER*t,y,num,0);
349.
350.     }
351.
352.     void LCD_ShowChar_2(uint16_t x,uint16_t y,uint8_t num,uint8_t mo
de) // CONVERTED TO CCS
353.     {
354.         uint8_t temp;
355.         uint16_t pos,t;
356.         uint16_t x0=x;
357.         uint16_t colortemp=POINT_COLOR;
358.
359.         if(x>LCD_W-128||y>LCD_H-128)
360.         {
361.             return;
362.         }
363.
364.         //Settings window
365.         num=num-1;//Obtained after the offset value
366.         Address_set(x,y,x+128-1,y+147-
1); //Setting the cursor position
367.         if(!mode) //Non-overlapping mode
368.         {
369.             for(pos=0;pos<147*COLOR_SIZE;pos++)
370.             {
371.                 temp=font_new[(uint16_t)num*147*COLOR_SIZE+pos];
372.                 //Call new font converted
373.                 for(t=0;t<8;t++) // Pooling bit by bit in the byte
374.                 {
375.                     if(temp&0x01)POINT_COLOR=colortemp;
376.                     else POINT_COLOR=BLACK;
377.                     LCD_WR_DATA16_SSI(POINT_COLOR);
378.                     temp>>=1;
379.                     x++;
380.                 }
381.                 x=x0;
382.                 y++;

```

```

382.     }
383. }else//Superimposition
384. {
385.     for(pos=0;pos<147;pos++)
386.     {
387.         temp=font_new[(uint16_t)num*147+pos];
388.         for(t=0;t<8;t++)
389.         {
390.             if(temp&0x01)LCD_DrawPoint(x+t,y+pos);//Draw a p
oint
391.             temp>>=1;
392.         }
393.     }
394. }
395. POINT_COLOR=colortemp;
396. }
397.
398. void LCD_ShowNum_3(uint16_t x,uint16_t y,uint16_t num)
399. {
400.     uint8_t t=0;
401.     LCD_ShowChar_3(x+LCD_W_CENTER*t,y,num);
402. }
403.
404. void LCD_ShowChar_3(uint16_t x,uint16_t y,uint8_t num) // CONVER
TED TO CCS
405. {
406.     uint8_t temp;
407.     uint16_t pos,t;
408.     uint16_t x0=x;
409.     uint16_t colortemp=POINT_COLOR;
410.     LCD_CallUp(GREEN);
411.
412.     if(x>LCD_W-128||y>LCD_H-128)
413.     {
414.         return;
415.     }
416.
417.     //Settings window
418.     num=num-1;//Obtained after the offset value
419.     Address_set(x,y,x+128-1,y+147-
1); //Setting the cursor position
420.
421.     for(pos=0;pos<147*COLOR_SIZE;pos++)
422.     {
423.         temp=font_new[(uint16_t)num*147*COLOR_SIZE+pos];
424.         //Call new font converted
425.         for(t=0;t<8;t++) // Pooling bit by bit in the byte
426.         {
427.             if(temp&0x01)POINT_COLOR=colortemp;
428.             else{
429.                 if(pos<(80*16)) POINT_COLOR=GREEN;
430.                 else POINT_COLOR = BLACK;
431.             }
432.             LCD_WR_DATA16_SSI(POINT_COLOR);
433.             temp>>=1;
434.             x++;
435.         }
436.         x=x0;

```

```

437.         y++;
438.     }
439.     POINT_COLOR=colortemp;
440. }
441.
442. void LCD_ShowNum_4(uint16_t x,uint16_t y,uint16_t num)
443. {
444.     uint8_t t=0;
445.     LCD_ShowChar_4(x+LCD_W_CENTER*t,y,num);
446. }
447.
448. void LCD_ShowChar_4(uint16_t x,uint16_t y,uint8_t num) // CONVERTED TO CCS
449. {
450.     uint8_t temp;
451.     uint16_t pos,t;
452.     uint16_t x0=x;
453.     uint16_t colortemp=POINT_COLOR;
454.
455.     LCD_CallDown(GREEN);
456.
457.     if(x>LCD_W-128||y>LCD_H-128)
458.     {
459.         return;
460.     }
461.
462.     //Settings window
463.     num=num-1;//Obtained after the offset value
464.     Address_set(x,y,x+128-1,y+147-
465. 1); //Setting the cursor position
466.     for(pos=0;pos<147*COLOR_SIZE;pos++)
467.     {
468.         temp=font_new[(uint16_t)num*147*COLOR_SIZE+pos];
469.         //Call new font converted
470.         for(t=0;t<8;t++) // Pooling bit by bit in the byte
471.         {
472.             if(temp&0x01)POINT_COLOR=colortemp;
473.             else{
474.                 if(pos > 80*16) POINT_COLOR=GREEN;
475.                 else POINT_COLOR = BLACK;
476.             }
477.             LCD_WR_DATA16_SSI(POINT_COLOR);
478.             temp>>=1;
479.             x++;
480.         }
481.         x=x0;
482.         y++;
483.     }
484.     POINT_COLOR=colortemp;
485.
486.     //Function for draw filled triangles
487.     void LCD_FillTriangle(uint16_t offset_x,uint16_t offset_y,uint16_t height,uint16_t base,uint16_t color)
488.     {
489.         uint16_t i,j;
490.         uint16_t base_half = base/2;
491.         uint16_t color_fill;

```



```

492.
493.     Address_set(offset_x,offset_y,offset_x+base,offset_y+height)
494.     ;
495.     for(i = 0; i<= height; i++){
496.         for(j = 0; j <= base ;j++){
497.             if((j+1+i>=(base_half)) && (j+1<=(base_half+i))) co
lor_fill = color;
498.             else color_fill = BLACK;
499.             LCD_WR_DATA16_SSI(color_fill);
500.         }
501.     }
502.
503. }
504.
505. //Function for draw filled triangles
506. void LCD_FillTriangleD(uint16_t offset_x,uint16_t offset_y,uint1
6_t height,uint16_t base,uint16_t color)
507. {
508.     uint16_t i,j;
509.     uint16_t double_height = height*2;
510.     uint16_t color_fill;
511.
512.     Address_set(offset_x,offset_y,offset_x+base,offset_y+height)
513.     ;
514.     for(i = 0; i<= height; i++){
515.         for(j = 0; j <= base ;j++){
516.             if((j<=i+1) || (i>=double_height-
j)) color_fill = BLACK;
517.             else color_fill = color;
518.             LCD_WR_DATA16_SSI(color_fill);
519.         }
520.     }
521. }
522.
523. void LCD_DrawPoint(uint16_t x,uint16_t y)
524. {
525.     Address_set(x,y,x,y); //Setting the cursor position
526.     LCD_WR_DATA16_SSI(POINT_COLOR);
527. }
528.
529. void LCD_ShowMaintenance(void){
530.     LCD_ShowNum_3(61,0,11); //Confirm call up
531. }
532. void LCD_ShowArrowUp(void){
533.     LCD_FillTriangle(20,191,100,200,YELLOW);
534. }
535. void LCD_ShowArrowDw(void){
536.     LCD_FillTriangle(20,191,100,200,YELLOW);
537. }
538. void LCD_ShowFloor(uint8_t piso){
539.     GV_PISO = piso;
540. }
541.
542. void LCD_Confirm_Call_Up(void){
543.     LCD_ShowNum_3(61,0,GV_PISO); //Confirm call up
544. }
545.

```

```

546.     void LCD_Confirm_Call_Dw(void){
547.         LCD_ShowNum_4(61,0,GV_PISO); //Confirm call down
548.     }
549.
550.
551.
552.     //A circle the size of the appointed position draw
553.     //(x,y):The center
554.     //r :Radius
555.     void Draw_Circle(uint16_t x0,uint16_t y0,uint8_t r) // CONVERTE
D TO CCS
556.     {
557.
558.         int a,b;
559.         int di;
560.         a=0;b=r;
561.         di=3-
(r<<1); //Judgment flag next point position
562.         while(a<=b)
563.         {
564.             LCD_DrawPoint(x0-b,y0-a); //3
565.             LCD_DrawPoint(x0+b,y0-a); //0
566.             LCD_DrawPoint(x0-a,y0+b); //1
567.             LCD_DrawPoint(x0-b,y0-a); //7
568.             LCD_DrawPoint(x0-a,y0-b); //2
569.             LCD_DrawPoint(x0+b,y0+a); //4
570.             LCD_DrawPoint(x0+a,y0-b); //5
571.             LCD_DrawPoint(x0+a,y0+b); //6
572.             LCD_DrawPoint(x0-b,y0+a);
573.             a++;
574.             //Using the Bresenham algorithm Circle
575.             if(di<0)di +=4*a+6;
576.             else
577.             {
578.                 di+=10+4*(a-b);
579.                 b--;
580.             }
581.             LCD_DrawPoint(x0+a,y0+b);
582.         }
583.     }
584.
585.     void Draw_CircleFill(uint16_t x0,uint16_t y0,uint8_t r,uint8_t o
p) // CONVERTED TO CCS
586.     {
587.
588.         int rlocal=0;
589.
590.         if(op==1) POINT_COLOR = GREEN;
591.         if(op==2) POINT_COLOR = BLACK;
592.
593.         while(rlocal<=r)
594.         {
595.             Draw_Circle(x0,y0,rlocal++);
596.         }
597.     }

```

Código fuente 8.- Archivo con el programa de implementación de funciones relacionadas al manejo de pantalla tft lcf ili9341.

Archivo "lcd_func.h"

```
1. /*
2.  * lcd_func.h
3.  *
4.  * Created on: 10 de dic. de 2016
5.  * Author: Francisco
6.  */
7.
8. #ifndef LCD_FUNC_H_
9. #define LCD_FUNC_H_
10.
11.
12. #define SCK GPIO_PIN_2 //PORTA
13. #define MISO GPIO_PIN_4 //PORTA
14. #define MOSI GPIO_PIN_5 //PORTA
15.
16. #define CS GPIO_PIN_3 //PORTE
17. #define RESET GPIO_PIN_2 //PORTE
18. #define DC GPIO_PIN_1 //PORTE
19.
20. static uint16_t BACK_COLOR, POINT_COLOR, GV_PISO;
21.
22. extern void conf_lcd(void);
23.
24. extern void conf_pineslcd(void);
25. extern void ini_lcd(void);
26.
27. extern void LCD_Clear(uint16_t Color);
28. extern void Address_set(uint16_t x1,uint16_t y1,uint16_t x2,uint16_t y
29. 2);
30. extern void LCD_WR_DATA8_SSI(uint8_t data);
31. extern void LCD_WR_DATA16_SSI(uint16_t data16);
32. extern void LCD_WR_REG_SSI(uint8_t data);
33. extern void LCD_Writ_Bus(uint8_t data);
34.
35. extern void LCD_DrawPoint(uint16_t x,uint16_t y);
36. extern void LCD_FillTriangle(uint16_t xsta,uint16_t ysta,uint16_t xend
37. ,uint16_t yend,uint16_t color);
38. extern void LCD_FillTriangleD(uint16_t offset_x,uint16_t offset_y,uint
39. 16_t height,uint16_t base,uint16_t color);
40. extern void LCD_ScrollingDef(uint16_t topfix, uint16_t heightscroll, u
41. int16_t botfix);
42. extern void LCD_ScrollingStart(uint16_t startline);
43.
44. extern void LCD_ShowChar_2(uint16_t x,uint16_t y,uint8_t num,uint8_t m
45. ode);
46. extern void LCD_ShowNum_2(uint16_t x,uint16_t y,uint16_t num);
47.
48. extern void LCD_ShowChar_3(uint16_t x,uint16_t y,uint8_t num);
49. extern void LCD_ShowNum_3(uint16_t x,uint16_t y,uint16_t num);
50.
51. extern void LCD_ShowChar_4(uint16_t x,uint16_t y,uint8_t num);
52. extern void LCD_ShowNum_4(uint16_t x,uint16_t y,uint16_t num);
53.
54. extern void LCD_CallUp(uint16_t Color);
55. extern void LCD_CallDown(uint16_t Color);
56. extern void LCD_CallClean(void);
```

```

53.
54. extern void LCD_ShowMaintenance(void);
55. extern void LCD_ShowArrowUp(void);
56. extern void LCD_ShowArrowDw(void);
57. extern void LCD_ShowFloor(uint8_t piso);
58. extern void LCD_Confirm_Call_Up(void);
59. extern void LCD_Confirm_Call_Dw(void);
60.
61. extern void Draw_CircleFill(uint16_t x0,uint16_t y0,uint8_t r,uint8_t
    op);
62. extern void Draw_Circle(uint16_t x0,uint16_t y0,uint8_t r);
63.
64. extern void LCD_ScrollingDef(uint16_t topfix, uint16_t heightscroll, u
    int16_t botfix);
65. extern void LCD_ScrollingStart(uint16_t startline);
66.
67. #endif /* LCD_FUNC_H_ */

```

Código fuente 9.- Archivo con el programa de cabeceras de funciones relacionadas al manejo de pantalla tft lcf ili9341.

Archivo "uart_func.c"

```

1. /*
2.  * uart_func.c
3.  *
4.  * Created on: 2 de dic. de 2016
5.  * Author: Francisco
6.  */
7.
8. #ifndef UART_FUNC_C_
9. #define UART_FUNC_C_
10.
11. #include "uart_func.h"
12. #include "ges_sensor.h"
13. #include "lcd_func.h"
14. #include "colors_fig.h"
15.
16. #define TARGET_IS_BLIZZARD_RB1
17. #include <stdint.h>
18. #include <stdbool.h>
19. #include <string.h>
20.
21. #include "inc/hw_ints.h"
22. #include "inc/hw_memmap.h"
23. #include "inc/hw_types.h"
24. #include "inc/hw_gpio.h"
25.
26. #include "driverlib/systick.h"
27. #include "driverlib/debug.h"
28. #include "driverlib/fpu.h"
29. #include "driverlib/gpio.h"
30. #include "driverlib/interrupt.h"
31. #include "driverlib/pin_map.h"
32. #include "driverlib/rom.h"
33. #include "driverlib/sysctl.h"
34. #include "driverlib/uart.h"

```

```

35. #include "driverlib/rom_map.h"
36.
37. void UART2IntHandler(void){
38.     uint32_t ui32Status;
39.     ui32Status = MAP_UARTIntStatus(UART2_BASE, true);
40.     MAP_UARTIntClear(UART2_BASE, ui32Status);
41.
42.     while(MAP_UARTCharsAvail(UART2_BASE)) {
43.         rx_buffer[wr_ptr] = (uint8_t) MAP_UARTCharGetNonBlocking(UART2
44. _BASE);
45.         MAP_UARTCharPut(UART0_BASE, rx_buffer[wr_ptr++]);
46.         if(wr_ptr==RX_BUFFER_SZ) wr_ptr=0;
47.     }
48. }
49.
50. void process_uart2(void){
51.     while(rd_ptr != wr_ptr){
52.         if(rx_buffer[rd_ptr] > IRCMD_LOWLIMIT) {
53.             msg_ptr=0;
54.             msg[msg_ptr++] = rx_buffer[rd_ptr];
55.         }
56.
57.         else if(msg[0] == IRCMD_GESTURE){
58.             msg[msg_ptr++] = rx_buffer[rd_ptr];
59.             if(msg_ptr == LIM_GES_HOVMOV){
60.                 switch(msg[1]){
61.                     case GES_HVMV:
62.                         switch(msg[2]){
63.                             case HVMV_left:
64.                                 gest_hoverleft();
65.                                 break;
66.                             case HVMV_right:
67.                                 gest_hoverright();
68.                                 break;
69.                             case HVMV_center:
70.                                 gest_hovercenter();
71.                                 break;
72.                             default:
73.                                 break;
74.                         }
75.
76.                         break;
77.
78.                     default:
79.                         break;
80.                 }
81.             }
82.
83.         }
84.         if(++rd_ptr==RX_BUFFER_SZ) rd_ptr=0;
85.     }
86. }
87.
88. void processv02_uart2(void){
89.     while(rd_ptr != wr_ptr){
90.         if(rx_buffer[rd_ptr] > IRCMD_LOWLIMIT) {
91.             msg_ptr=0;
92.             msg[msg_ptr++] = rx_buffer[rd_ptr];

```

```

93.     }
94.
95.     else {
96.         if(msg[0] == IRCMD_GESTURE){
97.
98.             msg[msg_ptr++] = rx_buffer[rd_ptr];
99.             if(msg_ptr == LIM_GES_SWMOV){
100.                 switch(msg[1]){
101.                     //MAP_UARTCharPut(UART0_BASE,msg_ptr[0]);
102.                     case GES_SWRG:
103.                         //if(hv_count > 150) gest_hoverright();
104.
105.                         //else gest_swright();
106.                         gest_swright();
107.                         break;
108.                     case GES_SWLF:
109.                         //if(hv_count > 150) gest_hoverleft();
110.                         //else gest_swleft();
111.                         gest_swleft();
112.                         break;
113.                     case GES_SWDW:
114.                         //if(hv_count > 150) gest_hovercenter();
115.
116.                         //else gest_swcenter();
117.                         break;
118.                     default: break;
119.                 }
120.             }
121.
122.             //if(msg[0] == IRCMD_ZCOOR){
123.             // if(hv_count < 255) hv_count++;
124.             //}
125.
126.             //if(msg[0] == IRCMD_PENUP){
127.             // hv_count = 0;
128.             //}
129.         }
130.         if(++rd_ptr==RX_BUFFER_SZ) rd_ptr=0;
131.     }
132. }
133. uint32_t time = 0;
134. uint32_t cron = 0;
135. uint32_t aux = 0;
136. void processv03_uart2(void){
137.     if(gest_det == 1){
138.         if(time-cron > 3000){
139.             LCD_FillTriangle(67,10,50,100,YELLOW);
140.             LCD_FillTriangleD(67,250,50,100,YELLOW);
141.             gest_det = 0;
142.         }
143.     }
144.     while(rd_ptr != wr_ptr){
145.         if(rx_buffer[rd_ptr] > IRCMD_LOWLIMIT) {
146.             msg_ptr=0;
147.             msg[msg_ptr++] = rx_buffer[rd_ptr];
148.             if(msg[0] == IRCMD_PENUP){
149.

```

```

150.                Draw_CircleFill(190,160,5,2);
151.                if( ( (zcoor_cur -
zcoor_first) > LIM_HOVCENT)
152.                    && (hv_count > (LIM_HOVER-1)
153.                        ) )                gest_hovercenter();
154.                else if(hv_right == 1) {
155.                    if(time - aux > 5)
156.                        gest_hoverright();
157.                }
158.                else if(hv_left == 1) {
159.                    if(time - aux > 5)
160.                        gest_hoverleft();
161.                }
162.
163.                MAP_UARTCharPut(UART0_BASE,0xFF);
164.                hv_count = 0;
165.                hv_right_ctr = 0;
166.                hv_left_ctr = 0;
167.                zcoor_first = 0;
168.                zcoor_cur = 0;
169.            }
170.        }
171.
172.        else {
173.            if(zcoor_cur < 20){
174.                if(msg[0] == IRCMD_RANGES){
175.                    msg[msg_ptr++] = rx_buffer[rd_ptr];
176.                    if(msg_ptr == LIM_RAN_SZ){
177.                        if(msg[1] == 0){
178.                            if(hv_count > (LIM_HOVER-1)){
179.                                if(hv_left_ctr < LIM_LEFT_CTR) h
v_left_ctr++;
180.                                else{
181.                                    hv_right=0;
182.                                    hv_left=1;
183.                                    hv_center=0;
184.                                    Draw_CircleFill(190,160,5,1)
185.                                ;
186.                                    aux = time;
187.                                }
188.                            }
189.                        }
190.                        else if(msg[2] == 0){
191.                            if(hv_count > (LIM_HOVER-1)){
192.                                if(hv_right_ctr < LIM_RIGHT_CTR)
hv_right_ctr++;
193.                                else{
194.                                    hv_right=1;
195.                                    hv_left=0;
196.                                    hv_center=0;
197.                                    Draw_CircleFill(190,160,5,1)
198.                                ;
199.                                    aux = time;
200.                                }
201.                            }
202.                        }
203.                    }
                }
            }
        }
    }

```

```

204.             hv_left=0;
205.             hv_center=1;
206.             Draw_CircleFill(190,160,5,1);
207.         }
208.         else{
209.             if(hv_count<(LIM_HOVER))  hv_count++
210.         ;
211.             hv_right=0;
212.             hv_left=0;
213.             hv_center=0;
214.         }
215.     }
216. }
217. }
218.
219.     if(msg[0] == IRCMD_ZCOOR){
220.         msg[msg_ptr++] = rx_buffer[rd_ptr];
221.         zcoor_cur = msg[1];
222.         if(zcoor_first == 0) zcoor_first = msg[1];
223.
224.     }
225. }
226. if(++rd_ptr==RX_BUFFER_SZ)  rd_ptr=0;
227. }
228. }
229. uint16_t rd_ptr = 0;
230. uint16_t wr_ptr = 0;
231. uint8_t gest_up_det = 0;
232. uint8_t gest_dw_det = 0;
233. uint8_t gest_up_conf = 0;
234. uint8_t gest_dw_conf = 0;
235. uint32_t cron_up = 0;
236. uint32_t cron_dw = 0;
237.
238. void processv04_uart2(void){
239.     if(gest_up_conf == 0){
240.         if(gest_up_det == 1){
241.             if(time - cron_up > TIMEOUT_CLEAN){
242.                 LCD_FillTriangle(67,10,50,100,BRED);
243.                 gest_up_det = 0;
244.             }
245.         }
246.     }
247.     if(gest_dw_conf == 0){
248.         if(gest_dw_det == 1){
249.             if(time - cron_dw > TIMEOUT_CLEAN){
250.                 LCD_FillTriangleD(67,250,50,100,BRED);
251.                 gest_dw_det = 0;
252.             }
253.         }
254.     }
255.     if(rd_ptr != wr_ptr){
256.         extrae_rangos();
257.         //umbralizacion();
258.         //analiza_gesto();
259.     }
260.     //else{
261.     // punto_listo = 1;

```



```

262.         //}
263.         umbralizacion();
264.         analiza_gesto();
265.     }
266.
267.     uint8_t dato_u = INI_DATO_U;
268.     uint8_t dato_d = INI_DATO_D;
269.     uint8_t dato_z = INI_DATO_Z;
270.     uint8_t penup = INI_PEN_UP;
271.
272.     uint8_t umbral_u = INI_UMB_U;
273.     uint8_t umbral_d = INI_UMB_D;
274.     uint8_t umbral_z = INI_UMB_Z;
275.
276.     uint8_t cnt_puntos = 0;
277.     uint8_t cnt_puntos_analizados = 0;
278.     uint8_t punto_listo = 0;
279.
280.     uint8_t arreglo_puntosu[150];
281.     uint8_t arreglo_puntosd[150];
282.
283.     void extrae_rangos(){
284.         uint8_t byte_rxbuffer = rx_buffer[rd_ptr];
285.
286.         if(++rd_ptr==RX_BUFFER_SZ) rd_ptr=0;
287.
288.         if(byte_rxbuffer > IRCMD_LOWLIMIT) {
289.             msg_ptr=0;
290.             msg[msg_ptr++] = byte_rxbuffer;
291.             if(msg[0] == IRCMD_PENUP){
292.                 penup = 1;
293.                 cron= time;
294.                 return;
295.             }
296.             return;
297.         }
298.         penup = 0;
299.
300.
301.         if(msg[0] == IRCMD_ZCOOR){
302.             dato_z = byte_rxbuffer;
303.             return;
304.         }
305.
306.         //if(dato_z > LIM_ZCOOR) return;
307.
308.         if(msg[0] != IRCMD_RANGES) return;
309.
310.         msg[msg_ptr++] = byte_rxbuffer;
311.         if(msg_ptr < LIM_RAN_SZ) return;
312.
313.         dato_d = msg[1];
314.         dato_u = msg[2];
315.         punto_listo = 1;
316.
317.         arreglo_puntosu[cnt_puntos]= dato_u;
318.         arreglo_puntosd[cnt_puntos]= dato_d;
319.         cnt_puntos++;
320.         msg[0] = 0xFF;

```

```

321.         msg_ptr = 0;
322.         if(cnt_puntos==150)
323.             cnt_puntos = 0;
324.         //cron = time;
325.         return;
326.     }
327.
328.     void umbralizacion(){
329.         if(punto_listo == 0) return;
330.         /*
331.         if(penup == 1){
332.             if((time - cron) > TIMEOUT_ESPERA){
333.                 umbral_d = 0;
334.                 umbral_u = 0;
335.             }
336.             punto_listo = 1;
337.             umbral_z = 1;
338.             return;
339.         }
340.         */
341.         if(dato_d > UMBRAL_RANGOS) umbral_d = 1;
342.         else umbral_d = 0;
343.
344.         if(dato_u > UMBRAL_RANGOS) umbral_u = 1;
345.         else umbral_u = 0;
346.
347.         if(dato_z < UMBRAL_Z) umbral_z = 1;
348.         else umbral_z = 0;
349.         return;
350.     }
351.
352.     void analiza_gesto(){
353.
354.         if(rd_ptr == wr_ptr){
355.             punto_listo = 1;
356.             if(v_fases == idle ){
357.
358.                 return;
359.             }
360.             if(v_fases != espera) return;
361.             if(time-cron > TIMEOUT_GAP){
362.                 //v_fases = idle;
363.                 //punto_listo = 1;
364.                 umbral_d = 0;
365.                 umbral_u = 0;
366.             }
367.         }
368.
369.
370.         if(punto_listo == 0) {
371.             //MAP_UARTCharPut(UART0_BASE, (char)0xF2);
372.             //MAP_UARTCharPut(UART0_BASE, (char)0xF2);
373.             return;
374.         }
375.         punto_listo = 0;
376.         cnt_puntos_analizados++;
377.         //if(umbral_z == 0){
378.         //    return;
379.         //}

```

```

380.
381.     switch (v_fases){
382.     case idle:
383.         v_fases = (fases) analiza_idle();
384.         cron = time;
385.         break;
386.     case fase1u:
387.         if((time - cron)>TIMEOUT_FASE1UD){
388.             v_fases = espera;
389.             cron = time;
390.             break;
391.         }
392.         v_fases = (fases) analiza_fase1u();
393.         if(v_fases == fase2u) cron = time;
394.         break;
395.     case fase2u:
396.
397.         v_fases = (fases) analiza_fase2u();
398.         if(v_fases == fase3u){
399.             gest_hoverleft(); //Hover UP
400.             v_fases = espera;
401.             cron = time;          //BKPT 1
402.             gest_up_det = 1;
403.             cron_up = time;
404.
405.         }
406.
407.         if(v_fases == espera)
408.             cron = time;
409.         break;
410.     case fase3u:
411.         if((time - cron)>TIMEOUT_FASE3UD){
412.             v_fases = espera; //BKPT 1
413.             cron = time;
414.             break;
415.         }
416.         v_fases = (fases) analiza_fase3u();
417.         if(v_fases == ejecutau) {
418.             gest_hoverleft(); //Hover UP
419.             v_fases = espera;
420.             cron = time;          //BKPT 1
421.         }
422.         if(v_fases == espera) cron = time; //BKPT 1
423.         break;
424.     case fase1d:
425.         if((time - cron)>TIMEOUT_FASE1UD){
426.             v_fases = espera;
427.             cron = time;
428.             break;
429.         }
430.         v_fases = (fases) analiza_fase1d();
431.         if(v_fases == fase2d) cron = time;
432.         if(v_fases == espera) cron = time;
433.         break;
434.     case fase2d:
435.         if((time - cron)>TIMEOUT_FASE2UD){
436.             v_fases = espera;
437.             cron = time;
438.             break;

```

```

439.         }
440.         v_fases = (fases) analiza_fase2d();
441.         if(v_fases == fase3d){
442.             gest_hoverright(); //Hover DOWN
443.             v_fases = espera;
444.             cron = time;
445.             gest_dw_det = 1;
446.             cron_dw = time;
447.         }
448.         if(v_fases == espera) cron = time;
449.         break;
450.     case fase3d:
451.         if((time - cron)>TIMEOUT_FASE3UD){
452.             v_fases = espera;
453.             cron = time;
454.             break;
455.         }
456.         v_fases = (fases) analiza_fase3d();
457.         if(v_fases == ejecutad) {
458.             gest_hoverright(); //Hover DOWN
459.             v_fases = espera;
460.             cron = time;
461.         }
462.         if(v_fases == espera) cron = time;
463.         break;
464.     case hover1:
465.         //Draw_CircleFill(190,160,5,1);
466.         if((time - cron)>TIMEIN_HOVER){
467.             Draw_CircleFill(190,160,5,1);
468.         }
469.         if((time - cron)>TIMEOUT_HOVER){
470.             v_fases = espera;
471.             Draw_CircleFill(190,160,5,2);
472.             cron = time;
473.             break;
474.         }
475.
476.         v_fases = (fases) analiza_hover1();
477.         if(v_fases == hover2u)
478.             cron = time;
479.         if(v_fases == hover2d)
480.             cron = time;
481.         if(v_fases == espera){
482.             cron = time;
483.             Draw_CircleFill(190,160,5,2);
484.         }
485.         break;
486.     case hover2u:
487.         if((time - cron)>TIMEOUT_HOVERUD){
488.             v_fases = espera;
489.             Draw_CircleFill(190,160,5,2);
490.             cron = time;
491.             break;
492.         }
493.         v_fases = (fases) analiza_hover2u();
494.         if(v_fases == ejecutau) {
495.             Draw_CircleFill(190,160,5,2);
496.             gest_hoverleft(); //Hover UP
497.             v_fases = espera;

```

```

498.         cron = time;
499.         gest_up_det = 1;
500.         cron_up = time;
501.     }
502.     if(v_fases == espera){
503.         cron = time;
504.         Draw_CircleFill(190,160,5,2);
505.     }
506.
507.     break;
508. case hover2d:
509.     if((time - cron)>TIMEOUT_HOVERUD){
510.         v_fases = espera;
511.         Draw_CircleFill(190,160,5,2);
512.         cron = time;
513.         break;
514.     }
515.     v_fases = (fases) analiza_hover2d();
516.     if(v_fases == ejecutad) {
517.         Draw_CircleFill(190,160,5,2);
518.         gest_hoverright(); //Hover DOWN
519.         v_fases = espera;
520.         cron = time;
521.         gest_dw_det = 1;
522.         cron_dw = time;
523.     }
524.     if(v_fases == espera){
525.         cron = time;
526.         Draw_CircleFill(190,160,5,2);
527.     }
528.
529.     break;
530. case espera:
531.
532.     Draw_CircleFill(190,160,5,2);
533.     v_fases = (fases) analiza_espera();
534.     break;
535. default:
536.     v_fases = idle;
537.     break;
538. }
539. MAP_UARTCharPut(UART0_BASE, (char)0xF2);
540. MAP_UARTCharPut(UART0_BASE, (char)v_fases);
541. return;
542. }
543.
544. uint8_t analiza_idle(void){
545.     if(umbral_u == 0 && umbral_d == 0){
546.
547.         return idle;
548.     }
549.
550.     if(umbral_u == 1 && umbral_d == 1){
551.
552.         return hover1;
553.     }
554.
555.     if(umbral_u == 0 && umbral_d == 1){
556.

```

```

557.         return fase1u;
558.     }
559.
560.     if(umbral_u == 1 && umbral_d == 0){
561.
562.         return fase1d;
563.     }
564.     return idle;
565. }
566. uint8_t analiza_fase1u(void){
567.     static uint8_t cnt_fase = 1;
568.
569.     if(umbral_u == 0 && umbral_d == 1){
570.         cnt_fase++;
571.         cron = time;
572.         return fase1u;
573.     }
574.
575.     if(umbral_u == 1 && umbral_d == 1){
576.         if(cnt_fase > QTY_FASE_1) {
577.             cnt_fase = 1;
578.             return fase2u;
579.         }
580.         cnt_fase = 1;
581.         cron = time;
582.         return hover1;
583.     }
584.
585.     if(umbral_u == 1 && umbral_d == 0){
586.         cnt_fase= 1;
587.         cron = time;
588.         return espera;
589.     }
590.
591.     if(umbral_u == 0 && umbral_d == 0){
592.         cnt_fase = 1;
593.         cron = time;
594.         return espera;
595.     }
596.     return fase1u;
597. }
598.
599. uint8_t analiza_fase2u(void){
600.     static uint8_t cnt_fase = 1;
601.
602.     if(umbral_u == 0 && umbral_d == 1){
603.         cnt_fase = 1;
604.         cron = time;
605.         return espera;
606.     }
607.
608.     if(umbral_u == 1 && umbral_d == 1){
609.         cnt_fase++;
610.
611.         if((time - cron)>TIMEOUT_FASE2UD){
612.             cnt_fase = 1;
613.
614.             return hover1;
615.         }

```

```

616.
617.         return fase2u;
618.     }
619.
620.     if(umbral_u == 1 && umbral_d == 0){
621.         cnt_fase= 1;
622.         cron = time;
623.         return fase3u;
624.     }
625.
626.     if(umbral_u == 0 && umbral_d == 0){
627.         cnt_fase = 1;
628.         return espera;
629.     }
630.     return fase2u;
631. }
632. uint8_t analiza_fase3u(void){
633.     static uint8_t cnt_fase = 1;
634.
635.     if(umbral_u == 0 && umbral_d == 1){
636.         cnt_fase = 1;
637.         cron = time;
638.         return espera; //BKPT 1
639.     }
640.
641.     if(umbral_u == 1 && umbral_d == 1){
642.         cnt_fase = 1;
643.         cron = time;
644.         return espera; //BKPT 1
645.     }
646.
647.     if(umbral_u == 1 && umbral_d == 0){
648.         cnt_fase++;
649.
650.         if(cnt_fase > QTY_FASE_3) {
651.             cron = time;
652.             return ejecutau; //BKPT 1
653.         }
654.         return fase3u; //BKPT 1
655.     }
656.
657.     if(umbral_u == 0 && umbral_d == 0){
658.
659.         cnt_fase = 1;
660.         cron = time;
661.         return espera; //BKPT 1
662.     }
663.     return fase3u;
664. }
665.
666. uint8_t analiza_fase1d(void){
667.     static uint8_t cnt_fase = 1;
668.
669.     if(umbral_u == 1 && umbral_d == 0){
670.         cnt_fase++;
671.         cron = time;
672.         return fase1d;
673.     }
674.

```

```

675.         if(umbral_u == 1 && umbral_d == 1){
676.             if(cnt_fase > QTY_FASE_1) {
677.                 cnt_fase = 1;
678.                 cron = time;
679.                 return fase2d;
680.             }
681.             cnt_fase = 1;
682.             cron = time;
683.             return hover1;
684.         }
685.
686.         if(umbral_u == 0 && umbral_d == 1){
687.             cnt_fase= 1;
688.             cron = time;
689.             return espera;
690.         }
691.
692.         if(umbral_u == 0 && umbral_d == 0){
693.             cnt_fase = 1;
694.             cron = time;
695.             return espera;
696.         }
697.         return fase3d;
698.     }
699.     uint8_t analiza_fase2d(void){
700.         static uint8_t cnt_fase = 1;
701.
702.         if(umbral_u == 1 && umbral_d == 0){
703.             cnt_fase = 1;
704.             cron = time;
705.             return espera;
706.         }
707.
708.         if(umbral_u == 1 && umbral_d == 1){
709.             cnt_fase++;
710.
711.             if((time - cron)>TIMEOUT_FASE2UD){
712.                 return hover1;
713.             }
714.
715.             return fase2d;
716.         }
717.
718.         if(umbral_u == 0 && umbral_d == 1){
719.             cnt_fase= 1;
720.             cron = time;
721.             return fase3d;
722.         }
723.
724.         if(umbral_u == 0 && umbral_d == 0){
725.             cnt_fase = 1;
726.             cron = time;
727.             return espera;
728.         }
729.         return fase2d;
730.     }
731.     uint8_t analiza_fase3d(void){
732.         static uint8_t cnt_fase = 1;
733.

```



```

734.         if(umbral_u == 1 && umbral_d == 0){
735.             cnt_fase = 1;
736.             cron = time;
737.             return espera;
738.         }
739.
740.         if(umbral_u == 1 && umbral_d == 1){
741.             cnt_fase = 1;
742.             cron = time;
743.             return espera;
744.         }
745.
746.         if(umbral_u == 0 && umbral_d == 1){
747.             cnt_fase++;
748.             if(cnt_fase > QTY_FASE_3) return ejecutad;
749.             return fase3d;
750.         }
751.
752.         if(umbral_u == 0 && umbral_d == 0){
753.
754.             cnt_fase = 1;
755.             cron = time;
756.             return espera;
757.         }
758.         return fase3d;
759.     }
760.     uint8_t analiza_hover1(void){
761.         if(umbral_u == 0 && umbral_d == 0){
762.             //cron = time;
763.             return espera;
764.         }
765.
766.         if(umbral_u == 1 && umbral_d == 1){
767.             //cron = time;
768.             return hover1;
769.         }
770.
771.         if(umbral_u == 0 && umbral_d == 1){
772.             //cron = time;
773.             return hover2d;
774.         }
775.
776.         if(umbral_u == 1 && umbral_d == 0){
777.             //cron = time;
778.             return hover2u;
779.         }
780.
781.         return hover1;
782.     }
783.     uint8_t analiza_hover2u(void){
784.         static uint8_t cnt_fase = 1;
785.         if(umbral_u == 0 && umbral_d == 0){
786.             cnt_fase = 1;
787.             cron = time;
788.             return espera;
789.         }
790.
791.         if(umbral_u == 1 && umbral_d == 1){
792.             cnt_fase = 1;

```

```

793.         cron = time;
794.         return espera;
795.     }
796.
797.     if(umbral_u == 0 && umbral_d == 1){
798.         cnt_fase = 1;
799.         cron = time;
800.         return espera;
801.     }
802.
803.     if(umbral_u == 1 && umbral_d == 0){
804.         cnt_fase++;
805.         if(cnt_fase > QTY_HOVER_UMBRALE){
806.             cron = time;
807.             return ejecutau;
808.         }
809.         return hover2u;
810.     }
811.
812.     return espera;
813. }
814.
815. uint8_t analiza_hover2d(void){
816.     static uint8_t cnt_fase = 1;
817.     if(umbral_u == 0 && umbral_d == 0){
818.         cnt_fase = 1;
819.         cron = time;
820.         return espera;
821.     }
822.
823.     if(umbral_u == 1 && umbral_d == 1){
824.         cnt_fase = 1;
825.         cron = time;
826.         return espera;
827.     }
828.
829.     if(umbral_u == 1 && umbral_d == 0){
830.         cnt_fase = 1;
831.         cron = time;
832.         return espera;
833.     }
834.
835.     if(umbral_u == 0 && umbral_d == 1){
836.         cnt_fase++;
837.         if(cnt_fase > QTY_HOVER_UMBRALE){
838.             cron = time;
839.             return ejecutad;
840.         }
841.         return hover2d;
842.     }
843.     return espera;
844. }
845. uint8_t analiza_espera(void){
846.
847.     if(penup == 1){
848.         if((time - cron) < TIMEOUT_ESPERA) return espera;
849.         return idle;
850.     }
851.     if(umbral_u == 0 && umbral_d == 0){

```

```

852.         if((time - cron) < TIMEOUT_ESPERA) return espera;
853.         return idle;
854.     }
855.     return espera;
856. }
857.
858. void
859. UART0Send2(uint8_t *pui8Buffer)
860. {
861.     uint32_t ui32Count=0;
862.     while(*(pui8Buffer+ui32Count) != '\0') ui32Count++;
863.     while(ui32Count--
864. ) MAP_UARTCharPut(UART0_BASE, *pui8Buffer++);
865. }
866. void
867. UART0Send3(uint8_t data){
868.     uint8_t nhigh, nlow;
869.     nhigh = (data/16 > 9)? 0x37 + data/16 : 0x30 + data/16;
870.     nlow  = (data%16 > 9)? 0x37 + data%16 : 0x30 + data%16;
871.     MAP_UARTCharPut(UART0_BASE, nhigh);
872.     MAP_UARTCharPut(UART0_BASE, nlow);
873. }
874.
875. void
876. UART0Send4(uint32_t data32){
877.     uint8_t nhigh, nmedium, nlow;
878.     nhigh = ((uint8_t)(data32>>8)>9)? 0x37 + (uint8_t)(data32>>8
879. )%16 : 0x30 + (uint8_t)(data32>>8)%16;
880.     MAP_UARTCharPut(UART0_BASE, nhigh);
881.     uint8_t data = (uint8_t)data32;
882.     nmedium = (data/16 > 9)? 0x37 + data/16 : 0x30 + data/16;
883.     nlow  = (data%16 > 9)? 0x37 + data%16 : 0x30 + data%16;
884.     MAP_UARTCharPut(UART0_BASE, nmedium);
885.     MAP_UARTCharPut(UART0_BASE, nlow);
886. }
887.
888. void
889. conf_uart0(void)
890. {
891.
892.     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
893.
894.     #ifndef PERIPHERAL_GPIOA_ACTIVE
895.     #define PERIPHERAL_GPIOA_ACTIVE
896.     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
897.     #endif
898.
899.     GPIOPinConfigure(GPIO_PA0_U0RX);
900.     GPIOPinConfigure(GPIO_PA1_U0TX);
901.     MAP_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1
902. );
903.
904.     MAP_UARTDisable(UART0_BASE);
905.     ROM_UARTConfigSetExpClk(UART0_BASE, MAP_SysCtlClockGet(), UA
906. RT0_BRATE, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_
907. NONE));

```

```

906.         MAP_IntEnable(INT_UART0);
907.         MAP_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
908.
909.         MAP_UARTEnable(UART0_BASE);
910.     }
911.
912.     void
913.     conf_uart2(void)
914.     {
915.
916.         MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
917.
918.         #ifndef PERIPHERAL_GPIOD_ACTIVE
919.         #define PERIPHERAL_GPIOD_ACTIVE
920.         MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
921.         #endif
922.
923.         GPIOPinConfigure(GPIO_PD6_U2RX);
924.
925.         HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
926.         HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= GPIO_PIN_7;
927.         GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_7, GPIO_STRENGTH_
4MA, GPIO_PIN_TYPE_STD_WPU);
928.         GPIOPinConfigure(GPIO_PD7_U2TX);
929.         MAP_GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_6 | GPIO_PIN_7
);
930.         HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_M;
931.
932.         MAP_UARTDisable(UART2_BASE);
933.         MAP_UARTConfigSetExpClk(UART2_BASE, ROM_SysCtlClockGet(), UA
RT2_BRATE, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_
NONE));
934.         //MAP_UARTFIFOLevelSet(UART2_BASE, UART_FIFO_TX2_8, UART_FIFO_
RX1_8);
935.         MAP_IntEnable(INT_UART2);
936.         MAP_UARTIntEnable(UART2_BASE, UART_INT_RX | UART_INT_RT);
937.         MAP_UARTEnable(UART2_BASE);
938.     }
939.
940.
941.     #endif

```

Código fuente 10.- Archivo con el programa de implementación de funciones relacionadas a los periféricos UART.

Archivo "uart_func.h"

```

1.  /*
2.   * uart_func.h
3.   *
4.   * Created on: 2 de dic. de 2016
5.   * Author: Francisco
6.   */
7.
8.  #ifndef UART_FUNC_H_
9.  #define UART_FUNC_H_
10.
11. #define TARGET_IS_BLIZZARD_RB1

```

```

12. #include <stdint.h>
13. #include <stdbool.h>
14. #include <string.h>
15.
16. #include "inc/hw_ints.h"
17. #include "inc/hw_memmap.h"
18. #include "inc/hw_types.h"
19. #include "inc/hw_gpio.h"
20.
21. #include "driverlib/systick.h"
22. #include "driverlib/debug.h"
23. #include "driverlib/fpu.h"
24. #include "driverlib/gpio.h"
25. #include "driverlib/interrupt.h"
26. #include "driverlib/pin_map.h"
27. #include "driverlib/rom.h"
28. #include "driverlib/sysctl.h"
29. #include "driverlib/uart.h"
30. #include "driverlib/rom_map.h"
31.
32. #define UART0_BRATE 230400
33. #define UART2_BRATE 115200
34.
35. #define RX_BUFFER_SZ 1024
36. #define MSG_BUFFER_MAXSZ 3
37.
38. static volatile uint8_t rx_buffer[RX_BUFFER_SZ];
39. static volatile uint8_t msg[MSG_BUFFER_MAXSZ];
40.
41. extern uint16_t rd_ptr;
42. extern uint16_t wr_ptr;
43. static uint8_t msg_ptr = 0;
44.
45. extern uint32_t time;
46. extern uint32_t cron;
47. extern uint32_t aux;
48.
49.
50. #define LIM_ZCOOR          30 //UNO MENOS A LA COORDENADA Z MINIMA CONSIDERADA
51. #define QTY_FASE_1          1 //UNO MENOS A LA CANTIDAD DE PUNTOS USADOS
52. #define QTY_HOVER_UMBRAL   3 //UNO MENOS A LA CANTIDAD DE PUNTOS USADOS
53. #define QTY_FASE_3          0 //UNO MENOS A LA CANTIDAD DE PUNTOS USADOS
54.
55. #define INI_DATO_U          0 //PRIMER PUNTO SIN USAR
56. #define INI_DATO_D          0 //PRIMER PUNTO SIN USAR
57. #define INI_DATO_Z          0 //PRIMER PUNTO SIN USAR
58. #define INI_PEN_UP          1 //PRIMER PUNTO SIN USAR
59.
60. #define INI_UMB_U           0 //PRIMER PUNTO SIN USAR
61. #define INI_UMB_D           0 //PRIMER PUNTO SIN USAR
62. #define INI_UMB_Z           0 //PRIMER PUNTO SIN USAR
63.
64. #define UMBRAL_RANGOS       120 //PRIMER PUNTO SIN USAR
65. #define UMBRAL_Z            30 //PRIMER PUNTO SIN USAR
66.

```

```

67. #define TIMEOUT_FASE1UD    200 //TIMEOUT FASE 1 DE SUBIDA o BAJADA en
    mS
68. #define TIMEOUT_FASE2UD    1000 //TIMEOUT FASE 2 DE HOVER en SUBIDA O B
    AJADAS en mS
69. #define TIMEOUT_FASE3UD    200 //TIMEOUT FASE 3 DE SUBIDA O BAJADA en
    mS
70.
71. #define TIMEIN_HOVER        600 //TIMEOUT FASE HOVER en mS
72. #define TIMEOUT_HOVER      5000 //TIMEOUT FASE HOVER en mS
73. #define TIMEOUT_HOVERUD    200 //TIMEOUT FASE HOVER en mS
74. #define TIMEOUT_ESPERA      400 //TIMEOUT ESPERA en mS
75. #define TIMEOUT_GAP         500 //TIMEOUT ESPERA en mS
76.
77. extern uint8_t dato_d;
78. extern uint8_t dato_u;
79. extern uint8_t dato_z;
80. extern uint8_t penup;
81.
82. extern uint8_t umbral_u;
83. extern uint8_t umbral_d;
84. extern uint8_t umbral_z;
85.
86. extern uint8_t punto_listo;
87. extern uint8_t cnt_puntos;
88. extern uint8_t cnt_puntos_analizados;
89.
90. typedef enum {idle=0,
91.               fase1u=1, fase2u=2, fase3u=3, ejecutau=12,
92.               fase1d=5, fase2d=6, fase3d=7, ejecutad=13,
93.               hover1=8, hover2u=9, hover2d=10,
94.               espera=11} fases;
95.
96. static fases v_fases;
97.
98. #define TIMEOUT_CLEAN    3000 //TIMEOUT ESPERA en mS
99.
100. extern uint8_t gest_up_det;
101. extern uint8_t gest_dw_det;
102. extern uint8_t gest_up_conf;
103. extern uint8_t gest_dw_conf;
104. extern uint32_t cron_up;
105. extern uint32_t cron_dw;
106.
107. extern uint8_t analiza_idle(void);
108. extern uint8_t analiza_fase1u(void);
109. extern uint8_t analiza_fase2u(void);
110. extern uint8_t analiza_fase3u(void);
111. extern uint8_t analiza_fase1d(void);
112. extern uint8_t analiza_fase2d(void);
113. extern uint8_t analiza_fase3d(void);
114. extern uint8_t analiza_hover1(void);
115. extern uint8_t analiza_hover2u(void);
116. extern uint8_t analiza_hover2d(void);
117. extern uint8_t analiza_espera(void);
118.
119. extern void conf_uart0(void);
120. extern void conf_uart2(void);
121.
122. extern void UART2IntHandler(void);

```

```

123.     extern void UART0Send2(uint8_t *pui8Buffer);
124.     extern void UART0Send3(uint8_t data);
125.     extern void UART0Send4(uint32_t data32);
126.
127.     extern void process_uart2(void);
128.     extern void processv02_uart2(void);
129.     extern void processv03_uart2(void);
130.     extern void processv04_uart2(void);
131.
132.     extern void extrae_rangos();
133.     extern void umbralizacion();
134.     extern void analiza_gesto();
135.
136.     #endif /* UART_FUNC_H_ */

```

Código fuente 11.- Archivo con el programa de cabeceras de funciones relacionadas a los periféricos UART.

Archivo "uartstdio.c"

```

1.  //*****
2.  //
3.  // uartstdio.c -
4.  //   Utility driver to provide simple UART console functions.
5.  // Copyright (c) 2007-
6.  //   2016 Texas Instruments Incorporated. All rights reserved.
7.  // Software License Agreement
8.  // Texas Instruments (TI) is supplying this software for use solely an
9.  //   d
10. // exclusively on TI's microcontroller products. The software is owned
11. //   by
12. // TI and/or its suppliers, and is protected under applicable copyrigh
13. //   t
14. // laws. You may not combine this software with "viral" open-source
15. //   software in order to form a larger program.
16. //
17. // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
18. // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BU
19. //   T
20. // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS F
21. //   OR
22. // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER AN
23. //   Y
24. // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
25. //   DAMAGES, FOR ANY REASON WHATSOEVER.
26. //
27. // This is part of revision 2.1.3.156 of the Tiva Utility Library.
28. //
29. //*****
30.
31. #include <stdbool.h>
32. #include <stdint.h>
33. #include <stdarg.h>

```

```

28. #include "inc/hw_ints.h"
29. #include "inc/hw_memmap.h"
30. #include "inc/hw_types.h"
31. #include "inc/hw_uart.h"
32. #include "driverlib/debug.h"
33. #include "driverlib/interrupt.h"
34. #include "driverlib/rom.h"
35. #include "driverlib/rom_map.h"
36. #include "driverlib/sysctl.h"
37. #include "driverlib/uart.h"
38. #include "utils/uartstdio.h"
39.
40. //*****
41. //
42. //! \addtogroup uartstdio_api
43. //! @{
44. //
45. //*****
46.
47. //*****
48. //
49. // If buffered mode is defined, set aside RX and TX buffers and read/w
rite
50. // pointers to control them.
51. //
52. //*****
53. #ifdef UART_BUFFERED
54.
55. //*****
56. //
57. // This global controls whether or not we are echoing characters back
to the
58. // transmitter. By default, echo is enabled but if using this module
as a
59. // convenient method of implementing a buffered serial interface over
which
60. // you will be running an application protocol, you are likely to want
to
61. // disable echo by calling UARTEchoSet(false).
62. //
63. //*****
64. static bool g_bDisableEcho;
65.
66. //*****
67. //
68. // Output ring buffer. Buffer is full if g_ui32UARTTxReadIndex is one
ahead of
69. // g_ui32UARTTxWriteIndex. Buffer is empty if the two indices are the
same.
70. //
71. //*****

```



```

72. static unsigned char g_pcUARTTxBuffer[UART_TX_BUFFER_SIZE];
73. static volatile uint32_t g_ui32UARTTxWriteIndex = 0;
74. static volatile uint32_t g_ui32UARTTxReadIndex = 0;
75.
76. //*****
77. //
78. // Input ring buffer. Buffer is full if g_ui32UARTTxReadIndex is one
   ahead of
79. // g_ui32UARTTxWriteIndex. Buffer is empty if the two indices are the
   same.
80. //
81. //*****
82. static unsigned char g_pcUARTRxBuffer[UART_RX_BUFFER_SIZE];
83. static volatile uint32_t g_ui32UARTRxWriteIndex = 0;
84. static volatile uint32_t g_ui32UARTRxReadIndex = 0;
85.
86. //*****
87. //
88. // Macros to determine number of free and used bytes in the transmit b
   uffer.
89. //
90. //*****
91. #define TX_BUFFER_USED          (GetBufferCount(&g_ui32UARTTxReadIndex
   , \
92.                                               &g_ui32UARTTxWriteInde
   x, \
93.                                               UART_TX_BUFFER_SIZE))
94. #define TX_BUFFER_FREE          (UART_TX_BUFFER_SIZE -
   TX_BUFFER_USED)
95. #define TX_BUFFER_EMPTY        (IsEmptyBuffer(&g_ui32UARTTxReadIndex,
   \
96.                                               &g_ui32UARTTxWriteIndex
   ))
97. #define TX_BUFFER_FULL          (IsBufferFull(&g_ui32UARTTxReadIndex,
   \
98.                                               &g_ui32UARTTxWriteIndex,
   \
99.                                               UART_TX_BUFFER_SIZE))
100. #define ADVANCE_TX_BUFFER_INDEX(Index) \
101. (Index) = ((Index) + 1) % UART_T
   X_BUFFER_SIZE
102.
103. //*****
104. //
105. // Macros to determine number of free and used bytes in the rece
   ive buffer.
106. //
107. //*****
108. #define RX_BUFFER_USED          (GetBufferCount(&g_ui32UARTRxRea
   dIndex, \
109.                                               &g_ui32UARTRxWri
   teIndex, \

```

```

110.                                     UART_RX_BUFFER_S
    IZE))
111.     #define RX_BUFFER_FREE           (UART_RX_BUFFER_SIZE -
    RX_BUFFER_USED)
112.     #define RX_BUFFER_EMPTY         (IsBufferEmpty(&g_ui32UARTRxRead
    Index, \
113.                                     &g_ui32UARTRxWrit
    eIndex))
114.     #define RX_BUFFER_FULL           (IsBufferFull(&g_ui32UARTRxReadI
    ndex, \
115.                                     &g_ui32UARTRxWrite
    Index, \
116.                                     UART_RX_BUFFER_SIZ
    E))
117.     #define ADVANCE_RX_BUFFER_INDEX(Index) \
118.                                     (Index) = ((Index) + 1) % UART_R
    X_BUFFER_SIZE
119.     #endif
120.
121.     //*****
    *****
122.     //
123.     // The base address of the chosen UART.
124.     //
125.     //*****
    *****
126.     static uint32_t g_ui32Base = 0;
127.
128.     //*****
    *****
129.     //
130.     // A mapping from an integer between 0 and 15 to its ASCII chara
    cter
131.     // equivalent.
132.     //
133.     //*****
    *****
134.     static const char * const g_pcHex = "0123456789abcdef";
135.
136.     //*****
    *****
137.     //
138.     // The list of possible base addresses for the console UART.
139.     //
140.     //*****
    *****
141.     static const uint32_t g_ui32UARTBase[3] =
142.     {
143.         UART0_BASE, UART1_BASE, UART2_BASE
144.     };
145.
146.     #ifdef UART_BUFFERED
147.     //*****
    *****
148.     //
149.     // The list of possible interrupts for the console UART.
150.     //
151.     //*****
    *****

```

```

152.     static const uint32_t g_ui32UARTInt[3] =
153.     {
154.         INT_UART0, INT_UART1, INT_UART2
155.     };
156.
157.     //*****
158.     //
159.     // The port number in use.
160.     //
161.     //*****
162.     static uint32_t g_ui32PortNum;
163.     #endif
164.
165.     //*****
166.     //
167.     // The list of UART peripherals.
168.     //
169.     //*****
170.     static const uint32_t g_ui32UARTPeriph[3] =
171.     {
172.         SYSCTL_PERIPH_UART0, SYSCTL_PERIPH_UART1, SYSCTL_PERIPH_UART
173.         2
174.     };
175.     //*****
176.     //
177.     //! Determines whether the ring buffer whose pointers and size a
178.     re provided
179.     //! is full or not.
180.     //! \param pui32Read points to the read index for the buffer.
181.     //! \param pui32Write points to the write index for the buffer.
182.     //! \param ui32Size is the size of the buffer in bytes.
183.     //!
184.     //! This function is used to determine whether or not a given ri
185.     ng buffer is
186.     //! full. The structure of the code is specifically to ensure t
187.     hat we do not
188.     //! see warnings from the compiler related to the order of volat
189.     ile accesses
190.     //! being undefined.
191.     //!
192.     //! \return Returns \b true if the buffer is full or \b false ot
193.     herwise.
194.     //
195.     //*****
196.     #ifdef UART_BUFFERED
197.     static bool
198.     IsBufferFull(volatile uint32_t *pui32Read,
199.                 volatile uint32_t *pui32Write, uint32_t ui32Size)
200.     {
201.         uint32_t ui32Write;

```

```

198.         uint32_t ui32Read;
199.
200.         ui32Write = *pui32Write;
201.         ui32Read = *pui32Read;
202.
203.         return((((ui32Write + 1) % ui32Size) == ui32Read) ? true : f
false);
204.     }
205.     #endif
206.
207.     //*****
208.     //
209.     /*! Determines whether the ring buffer whose pointers and size a
re provided
210.     /*! is empty or not.
211.     /*!
212.     /*! \param pui32Read points to the read index for the buffer.
213.     /*! \param pui32Write points to the write index for the buffer.
214.     /*!
215.     /*! This function is used to determine whether or not a given ri
ng buffer is
216.     /*! empty. The structure of the code is specifically to ensure
that we do not
217.     /*! see warnings from the compiler related to the order of volat
ile accesses
218.     /*! being undefined.
219.     /*!
220.     /*! \return Returns \b true if the buffer is empty or \b false o
therwise.
221.     //
222.     //*****
223.     #ifdef UART_BUFFERED
224.     static bool
225.     IsBufferEmpty(volatile uint32_t *pui32Read,
226.                  volatile uint32_t *pui32Write)
227.     {
228.         uint32_t ui32Write;
229.         uint32_t ui32Read;
230.
231.         ui32Write = *pui32Write;
232.         ui32Read = *pui32Read;
233.
234.         return((ui32Write == ui32Read) ? true : false);
235.     }
236.     #endif
237.
238.     //*****
239.     //
240.     /*! Determines the number of bytes of data contained in a ring b
uffer.
241.     /*!
242.     /*! \param pui32Read points to the read index for the buffer.
243.     /*! \param pui32Write points to the write index for the buffer.
244.     /*! \param ui32Size is the size of the buffer in bytes.

```

```

245.    //!<
246.    //!< This function is used to determine how many bytes of data a
    given ring
247.    //!< buffer currently contains. The structure of the code is spe
    cifically to
248.    //!< ensure that we do not see warnings from the compiler related
    to the order
249.    //!< of volatile accesses being undefined.
250.    //!<
251.    //!< \return Returns the number of bytes of data currently in the
    buffer.
252.    //!<
253.    //!<*****
    *****
254.    #ifdef UART_BUFFERED
255.    static uint32_t
256.    GetBufferCount(volatile uint32_t *pui32Read,
257.                  volatile uint32_t *pui32Write, uint32_t ui32Size)
258.    {
259.        uint32_t ui32Write;
260.        uint32_t ui32Read;
261.
262.        ui32Write = *pui32Write;
263.        ui32Read = *pui32Read;
264.
265.        return((ui32Write >= ui32Read) ? (ui32Write - ui32Read) :
266.               (ui32Size - (ui32Read - ui32Write)));
267.    }
268.    #endif
269.
270.    //!<*****
    *****
271.    //!<
272.    //!< Take as many bytes from the transmit buffer as we have space
    for and move
273.    //!< them into the UART transmit FIFO.
274.    //!<
275.    //!<*****
    *****
276.    #ifdef UART_BUFFERED
277.    static void
278.    UARTPrimeTransmit(uint32_t ui32Base)
279.    {
280.        //!<
281.        //!< Do we have any data to transmit?
282.        //!<
283.        if(!TX_BUFFER_EMPTY)
284.        {
285.            //!<
286.            //!< Disable the UART interrupt. If we don't do this ther
    e is a race
287.            //!< condition which can cause the read index to be corrup
    ted.
288.            //!<
289.            MAP_IntDisable(g_ui32UARTInt[g_ui32PortNum]);
290.
291.            //!<

```

```

292.         // Yes -
        take some characters out of the transmit buffer and feed
293.         // them to the UART transmit FIFO.
294.         //
295.         while(MAP_UARTSpaceAvail(ui32Base) && !TX_BUFFER_EMPTY)
296.         {
297.             MAP_UARTCharPutNonBlocking(ui32Base,
298.                                         g_pcUARTTxBuffer[g_ui32UART
TTxReadIndex]);
299.             ADVANCE_TX_BUFFER_INDEX(g_ui32UARTTxReadIndex);
300.         }
301.
302.         //
303.         // Reenable the UART interrupt.
304.         //
305.         MAP_IntEnable(g_ui32UARTInt[g_ui32PortNum]);
306.     }
307. }
308. #endif
309.
310.     /*******
        *****
311.     //
312.     /**! Configures the UART console.
313.     /**!
314.     /**! \param ui32PortNum is the number of UART port to use for the
serial console
315.     /**! (0-2)
316.     /**! \param ui32Baud is the bit rate that the UART is to be confi
gured to use.
317.     /**! \param ui32SrcClock is the frequency of the source clock for
the UART
318.     /**! module.
319.     /**!
320.     /**! This function will configure the specified serial port to be
used as a
321.     /**! serial console. The serial parameters are set to the baud r
ate
322.     /**! specified by the \e ui32Baud parameter and use 8 bit, no par
ity, and 1 stop
323.     /**! bit.
324.     /**!
325.     /**! This function must be called prior to using any of the other
UART console
326.     /**! functions: UARTprintf() or UARTgets(). This function assume
s that the
327.     /**! caller has previously configured the relevant UART pins for
operation as a
328.     /**! UART rather than as GPIOs.
329.     /**!
330.     /**! \return None.
331.     //
332.     /*******
        *****
333.     void
334.     UARTStdioConfig(uint32_t ui32PortNum, uint32_t ui32Baud, uint32_
t ui32SrcClock)
335.     {

```

```

336.         //
337.         // Check the arguments.
338.         //
339.         ASSERT((ui32PortNum == 0) || (ui32PortNum == 1) ||
340.                (ui32PortNum == 2));
341.
342.         #ifdef UART_BUFFERED
343.             //
344.             // In buffered mode, we only allow a single instance to be o
345.             //
346.             ASSERT(g_ui32Base == 0);
347.         #endif
348.
349.         //
350.         // Check to make sure the UART peripheral is present.
351.         //
352.         if(!MAP_SysCtlPeripheralPresent(g_ui32UARTPeriph[ui32PortNum
353.         ])
354.         {
355.             return;
356.         }
357.         //
358.         // Select the base address of the UART.
359.         //
360.         g_ui32Base = g_ui32UARTBase[ui32PortNum];
361.
362.         //
363.         // Enable the UART peripheral for use.
364.         //
365.         MAP_SysCtlPeripheralEnable(g_ui32UARTPeriph[ui32PortNum]);
366.
367.         //
368.         // Configure the UART for 115200, n, 8, 1
369.         //
370.         MAP_UARTConfigSetExpClk(g_ui32Base, ui32SrcClock, ui32Baud,
371.
372.                                (UART_CONFIG_PAR_NONE | UART_CONFIG_
373.                                STOP_ONE |
374.                                UART_CONFIG_WLEN_8));
375.
376.         #ifdef UART_BUFFERED
377.             //
378.             // Set the UART to interrupt whenever the TX FIFO is almost
379.             // empty or
380.             // when any character is received.
381.             //
382.             MAP_UARTFIFOLevelSet(g_ui32Base, UART_FIFO_TX1_8, UART_FIFO_
383.             RX1_8);
384.
385.             //
386.             // Flush both the buffers.
387.             //
388.             UARTFlushRx();
389.             UARTFlushTx(true);
390.
391.             //
392.             // Remember which interrupt we are dealing with.

```

```

389.         //
390.         g_ui32PortNum = ui32PortNum;
391.
392.         //
393.         // We are configured for buffered output so enable the master interrupt
394.         // for this UART and the receive interrupts. We don't actually enable the
395.         // transmit interrupt in the UART itself until some data has been placed
396.         // in the transmit buffer.
397.         //
398.         MAP_UARTIntDisable(g_ui32Base, 0xFFFFFFFF);
399.         MAP_UARTIntEnable(g_ui32Base, UART_INT_RX | UART_INT_RT);
400.         MAP_IntEnable(g_ui32UARTInt[ui32PortNum]);
401.     #endif
402.
403.     //
404.     // Enable the UART operation.
405.     //
406.     MAP_UARTEnable(g_ui32Base);
407. }
408.
409.     //*****
410.     //
411.     /// Writes a string of characters to the UART output.
412.     ///
413.     /// \param pcBuf points to a buffer containing the string to transmit.
414.     /// \param ui32Len is the length of the string to transmit.
415.     ///
416.     /// This function will transmit the string to the UART output. The number of
417.     /// characters transmitted is determined by the \e ui32Len parameter. This
418.     /// function does no interpretation or translation of any characters. Since
419.     /// the output is sent to a UART, any LF (/n) characters encountered will be
420.     /// replaced with a CRLF pair.
421.     ///
422.     /// Besides using the \e ui32Len parameter to stop transmitting the string, if
423.     /// a null character (0) is encountered, then no more characters will be
424.     /// transmitted and the function will return.
425.     ///
426.     /// In non-buffered mode, this function is blocking and will not return until
427.     /// all the characters have been written to the output FIFO. In buffered mode,
428.     /// the characters are written to the UART transmit buffer and the call returns
429.     /// immediately. If insufficient space remains in the transmit buffer,
430.     /// additional characters are discarded.
431.     ///
432.     /// \return Returns the count of characters written.

```



```

433.    //
434.    //*****
435.    int
436.    UARTwrite(const char *pcBuf, uint32_t ui32Len)
437.    {
438.    #ifdef UART_BUFFERED
439.        unsigned int uIdx;
440.
441.        //
442.        // Check for valid arguments.
443.        //
444.        ASSERT(pcBuf != 0);
445.        ASSERT(g_ui32Base != 0);
446.
447.        //
448.        // Send the characters
449.        //
450.        for(uIdx = 0; uIdx < ui32Len; uIdx++)
451.        {
452.            //
453.            // If the character to the UART is \n, then add a \r bef
ore it so that
454.            // \n is translated to \n\r in the output.
455.            //
456.            if(pcBuf[uIdx] == '\n')
457.            {
458.                if(!TX_BUFFER_FULL)
459.                {
460.                    g_pcUARTTxBuffer[g_ui32UARTTxWriteIndex] = '\r';
461.                    ADVANCE_TX_BUFFER_INDEX(g_ui32UARTTxWriteIndex);
462.                }
463.                else
464.                {
465.                    //
466.                    // Buffer is full -
discard remaining characters and return.
467.                    //
468.                    break;
469.                }
470.            }
471.
472.            //
473.            // Send the character to the UART output.
474.            //
475.            if(!TX_BUFFER_FULL)
476.            {
477.                g_pcUARTTxBuffer[g_ui32UARTTxWriteIndex] = pcBuf[uId
x];
478.                ADVANCE_TX_BUFFER_INDEX(g_ui32UARTTxWriteIndex);
479.            }
480.            else
481.            {
482.                //
483.                // Buffer is full -
discard remaining characters and return.
484.                //

```

```

485.             break;
486.         }
487.     }
488.
489.     //
490.     // If we have anything in the buffer, make sure that the UAR
T is set
491.     // up to transmit it.
492.     //
493.     if(!TX_BUFFER_EMPTY)
494.     {
495.         UARTPrimeTransmit(g_ui32Base);
496.         MAP_UARTIntEnable(g_ui32Base, UART_INT_TX);
497.     }
498.
499.     //
500.     // Return the number of characters written.
501.     //
502.     return(uIdx);
503. #else
504.     unsigned int uIdx;
505.
506.     //
507.     // Check for valid UART base address, and valid arguments.
508.     //
509.     ASSERT(g_ui32Base != 0);
510.     ASSERT(pcBuf != 0);
511.
512.     //
513.     // Send the characters
514.     //
515.     for(uIdx = 0; uIdx < ui32Len; uIdx++)
516.     {
517.         //
518.         // If the character to the UART is \n, then add a \r bef
ore it so that
519.         // \n is translated to \n\r in the output.
520.         //
521.         if(pcBuf[uIdx] == '\n')
522.         {
523.             MAP_UARTCharPut(g_ui32Base, '\r');
524.         }
525.
526.         //
527.         // Send the character to the UART output.
528.         //
529.         MAP_UARTCharPut(g_ui32Base, pcBuf[uIdx]);
530.     }
531.
532.     //
533.     // Return the number of characters written.
534.     //
535.     return(uIdx);
536. #endif
537. }
538.
539. //*****
540. //

```

```

541.    //!< A simple UART based get string function, with some line proc
    essing.
542.    //!<
543.    //!< \param pcBuf points to a buffer for the incoming string from
    the UART.
544.    //!< \param ui32Len is the length of the buffer for storage of th
    e string,
545.    //!< including the trailing 0.
546.    //!<
547.    //!< This function will receive a string from the UART input and
    store the
548.    //!< characters in the buffer pointed to by \e pcBuf. The charac
    ters will
549.    //!< continue to be stored until a termination character is recei
    ved. The
550.    //!< termination characters are CR, LF, or ESC. A CRLF pair is t
    reated as a
551.    //!< single termination character. The termination characters ar
    e not stored in
552.    //!< the string. The string will be terminated with a 0 and the
    function will
553.    //!< return.
554.    //!<
555.    //!< In both buffered and unbuffered modes, this function will bl
    ock until
556.    //!< a termination character is received. If non-
    blocking operation is required
557.    //!< in buffered mode, a call to UARTPeek() may be made to determ
    ine whether
558.    //!< a termination character already exists in the receive buffer
    prior to
559.    //!< calling UARTgets().
560.    //!<
561.    //!< Since the string will be null terminated, the user must ensu
    re that the
562.    //!< buffer is sized to allow for the additional null character.

563.    //!<
564.    //!< \return Returns the count of characters that were stored, no
    t including
565.    //!< the trailing 0.
566.    //!<
567.    //!<*****
    *****

568.    int
569.    UARTgets(char *pcBuf, uint32_t ui32Len)
570.    {
571.    #ifdef UART_BUFFERED
572.        uint32_t ui32Count = 0;
573.        int8_t cChar;
574.
575.        //
576.        // Check the arguments.
577.        //
578.        ASSERT(pcBuf != 0);
579.        ASSERT(ui32Len != 0);
580.        ASSERT(g_ui32Base != 0);
581.
582.        //

```

```

583.         // Adjust the length back by 1 to leave space for the trailing
584.         // null terminator.
585.         //
586.         ui32Len--;
587.         //
588.         // Process characters until a newline is received.
589.         //
590.         while(1)
591.         {
592.             //
593.             // Read the next character from the receive buffer.
594.             //
595.             if(!RX_BUFFER_EMPTY)
596.             {
597.                 cChar = g_pcUARTRxBuffer[g_ui32UARTRxReadIndex];
598.                 ADVANCE_RX_BUFFER_INDEX(g_ui32UARTRxReadIndex);
599.                 //
600.                 // See if a newline or escape character was received
601.                 //
602.                 if((cChar == '\r') || (cChar == '\n') || (cChar == 0
603.                    x1b))
604.                 {
605.                     //
606.                     // Stop processing the input and end the line.
607.                     //
608.                     break;
609.                 }
610.                 //
611.                 // Process the received character as long as we are
612.                 // not at the end
613.                 // of the buffer. If the end of the buffer has been
614.                 // reached then
615.                 // all additional characters are ignored until a new
616.                 // line is
617.                 // received.
618.                 if(ui32Count < ui32Len)
619.                 {
620.                     //
621.                     // Store the character in the caller supplied buffer.
622.                     //
623.                     pcBuf[ui32Count] = cChar;
624.                     //
625.                     // Increment the count of characters received.
626.                     //
627.                     ui32Count++;
628.                 }
629.             }
630.         }
631.     }
632.     //
633.     // Add a null termination to the string.
634.

```

```

635.         //
636.         pcBuf[ui32Count] = 0;
637.
638.         //
639.         // Return the count of int8_ts in the buffer, not counting t
        he trailing 0.
640.         //
641.         return(ui32Count);
642.     #else
643.         uint32_t ui32Count = 0;
644.         int8_t cChar;
645.         static int8_t bLastWasCR = 0;
646.
647.         //
648.         // Check the arguments.
649.         //
650.         ASSERT(pcBuf != 0);
651.         ASSERT(ui32Len != 0);
652.         ASSERT(g_ui32Base != 0);
653.
654.         //
655.         // Adjust the length back by 1 to leave space for the traili
        ng
656.         // null terminator.
657.         //
658.         ui32Len--;
659.
660.         //
661.         // Process characters until a newline is received.
662.         //
663.         while(1)
664.         {
665.             //
666.             // Read the next character from the console.
667.             //
668.             cChar = MAP_UARTCharGet(g_ui32Base);
669.
670.             //
671.             // See if the backspace key was pressed.
672.             //
673.             if(cChar == '\b')
674.             {
675.                 //
676.                 // If there are any characters already in the buffer
        , then delete
677.                 // the last.
678.                 //
679.                 if(ui32Count)
680.                 {
681.                     //
682.                     // Rub out the previous character.
683.                     //
684.                     UARTwrite("\b \b", 3);
685.
686.                     //
687.                     // Decrement the number of characters in the buf
        fer.
688.                     //
689.                     ui32Count--;

```

```

690.         }
691.
692.         //
693.         // Skip ahead to read the next character.
694.         //
695.         continue;
696.     }
697.
698.     //
699.     // If this character is LF and last was CR, then just go
    bble up the
700.     // character because the EOL processing was taken care o
    f with the CR.
701.     //
702.     if((cChar == '\n') && bLastWasCR)
703.     {
704.         bLastWasCR = 0;
705.         continue;
706.     }
707.
708.     //
709.     // See if a newline or escape character was received.
710.     //
711.     if((cChar == '\r') || (cChar == '\n') || (cChar == 0x1b)
    )
712.     {
713.         //
714.         // If the character is a CR, then it may be followed
    by a LF which
715.         // should be paired with the CR. So remember that a
    CR was
716.         // received.
717.         //
718.         if(cChar == '\r')
719.         {
720.             bLastWasCR = 1;
721.         }
722.
723.         //
724.         // Stop processing the input and end the line.
725.         //
726.         break;
727.     }
728.
729.     //
730.     // Process the received character as long as we are not
    at the end of
731.     // the buffer. If the end of the buffer has been reache
    d then all
732.     // additional characters are ignored until a newline is
    received.
733.     //
734.     if(ui32Count < ui32Len)
735.     {
736.         //
737.         // Store the character in the caller supplied buffer
    .
738.         //
739.         pcBuf[ui32Count] = cChar;

```

```

740.
741.          //
742.          // Increment the count of characters received.
743.          //
744.          ui32Count++;
745.
746.          //
747.          // Reflect the character back to the user.
748.          //
749.          MAP_UARTCharPut(g_ui32Base, cChar);
750.      }
751.  }
752.
753.      //
754.      // Add a null termination to the string.
755.      //
756.      pcBuf[ui32Count] = 0;
757.
758.      //
759.      // Send a CRLF pair to the terminal to end the line.
760.      //
761.      UARTwrite("\r\n", 2);
762.
763.      //
764.      // Return the count of int8_ts in the buffer, not counting t
he trailing 0.
765.      //
766.      return(ui32Count);
767.  #endif
768.  }
769.
770.  //*****
771.      //
772.      //! Read a single character from the UART, blocking if necessary
.
773.      //!
774.      //! This function will receive a single character from the UART
and store it at
775.      //! the supplied address.
776.      //!
777.      //! In both buffered and unbuffered modes, this function will bl
ock until a
778.      //! character is received. If non-
blocking operation is required in buffered
779.      //! mode, a call to UARTRxAvail() may be made to determine wheth
er any
780.      //! characters are currently available for reading.
781.      //!
782.      //! \return Returns the character read.
783.      //
784.      //*****
785.      unsigned char
786.      UARTgetc(void)
787.      {
788.      #ifdef UART_BUFFERED
789.          unsigned char cChar;
790.

```

```

791.         //
792.         // Wait for a character to be received.
793.         //
794.         while(RX_BUFFER_EMPTY)
795.         {
796.             //
797.             // Block waiting for a character to be received (if the
buffer is
798.             // currently empty).
799.             //
800.         }
801.
802.         //
803.         // Read a character from the buffer.
804.         //
805.         cChar = g_pcUARTRxBuffer[g_ui32UARTRxReadIndex];
806.         ADVANCE_RX_BUFFER_INDEX(g_ui32UARTRxReadIndex);
807.
808.         //
809.         // Return the character to the caller.
810.         //
811.         return(cChar);
812.     #else
813.         //
814.         // Block until a character is received by the UART then retu
rn it to
815.         // the caller.
816.         //
817.         return(MAP_UARTCharGet(g_ui32Base));
818.     #endif
819. }
820.
821.     //*****
*****
822.     //
823.     /// A simple UART based vprintf function supporting %c, %d, %
p, %s, %u,
824.     /// %x, and %X.
825.     ///
826.     /// \param pcString is the format string.
827.     /// \param vaArgP is a variable argument list pointer whose cont
ent will depend
828.     /// upon the format string passed in \e pcString.
829.     ///
830.     /// This function is very similar to the C library <tt>vprintf()
</tt> function.
831.     /// All of its output will be sent to the UART. Only the follow
ing formatting
832.     /// characters are supported:
833.     ///
834.     /// - %c to print a character
835.     /// - %d or %i to print a decimal value
836.     /// - %s to print a string
837.     /// - %u to print an unsigned decimal value
838.     /// -
      %x to print a hexadecimal value using lower case letters
839.     /// -
      %X to print a hexadecimal value using lower case letters (not upper
case

```



```

840.    /// letters as would typically be used)
841.    /// - %p to print a pointer as a hexadecimal value
842.    /// - %% to print out a % character
843.    ///
844.    /// For %s, %d, %i, %u, %p, %x, and %X, an optional number may reside
845.    /// between the % and the format character, which specifies the
846.    /// minimum number
847.    /// of characters to use for that value; if preceded by a 0 then
848.    /// the extra
849.    /// characters will be filled with zeros instead of spaces. For
850.    /// example,
851.    /// %%\%8d' will use eight characters to print the decimal value
852.    /// with spaces
853.    /// added to reach eight; %%\%08d' will use eight characters as
854.    /// well but will
855.    /// add zeroes instead of spaces.
856.    ///
857.    /// The type of the arguments in the variable arguments list must
858.    /// match the
859.    /// requirements of the format string. For example, if an integer
860.    /// was passed
861.    /// where a string was expected, an error of some kind will most
862.    /// likely occur.
863.    ///
864.    /// \return None.
865.    //
866.    //*****
867.    void
868.    UARTvprintf(const char *pcString, va_list vaArgP)
869.    {
870.        uint32_t ui32Idx, ui32Value, ui32Pos, ui32Count, ui32Base, ui32Neg;
871.        char *pcStr, pcBuf[16], cFill;
872.        //
873.        // Check the arguments.
874.        //
875.        ASSERT(pcString != 0);
876.        //
877.        // Loop while there are more characters in the string.
878.        //
879.        while(*pcString)
880.        {
881.            //
882.            // Find the first non-
883.            // % character, or the end of the string.
884.            //
885.            for(ui32Idx = 0;
886.                (pcString[ui32Idx] != '%') && (pcString[ui32Idx] !=
887.                '\0');
888.                ui32Idx++)
889.            {
890.            }
891.            //
892.            // Write this portion of the string.

```

```

886.          //
887.          UARTwrite(pcString, ui32Idx);
888.
889.          //
890.          // Skip the portion of the string that was written.
891.          //
892.          pcString += ui32Idx;
893.
894.          //
895.          // See if the next character is a %.
896.          //
897.          if(*pcString == '%')
898.          {
899.              //
900.              // Skip the %.
901.              //
902.              pcString++;
903.
904.              //
905.              // Set the digit count to zero, and the fill character to space
906.              // (in other words, to the defaults).
907.              //
908.              ui32Count = 0;
909.              cFill = ' ';
910.
911.              //
912.              // It may be necessary to get back here to process more characters.
913.              // Goto's aren't pretty, but effective. I feel extremely dirty for
914.              // using not one but two of the beasts.
915.              //
916.              again:
917.
918.              //
919.              // Determine how to handle the next character.
920.              //
921.              switch(*pcString++)
922.              {
923.                  //
924.                  // Handle the digit characters.
925.                  //
926.                  case '0':
927.                  case '1':
928.                  case '2':
929.                  case '3':
930.                  case '4':
931.                  case '5':
932.                  case '6':
933.                  case '7':
934.                  case '8':
935.                  case '9':
936.                  {
937.                      //
938.                      // If this is a zero, and it is the first digit, then the
939.                      // fill character is a zero instead of a space.

```

```

940.                //
941.                if((pcString[-
1] == '0') && (ui32Count == 0))
942.                {
943.                    cFill = '0';
944.                }
945.
946.                //
947.                // Update the digit count.
948.                //
949.                ui32Count *= 10;
950.                ui32Count += pcString[-1] - '0';
951.
952.                //
953.                // Get the next character.
954.                //
955.                goto again;
956.            }
957.
958.            //
959.            // Handle the %c command.
960.            //
961.            case 'c':
962.            {
963.                //
964.                // Get the value from the varargs.
965.                //
966.                ui32Value = va_arg(vaArgP, uint32_t);
967.
968.                //
969.                // Print out the character.
970.                //
971.                UARTwrite((char *)&ui32Value, 1);
972.
973.                //
974.                // This command has been handled.
975.                //
976.                break;
977.            }
978.
979.            //
980.            // Handle the %d and %i commands.
981.            //
982.            case 'd':
983.            case 'i':
984.            {
985.                //
986.                // Get the value from the varargs.
987.                //
988.                ui32Value = va_arg(vaArgP, uint32_t);
989.
990.                //
991.                // Reset the buffer position.
992.                //
993.                ui32Pos = 0;
994.
995.                //
996.                // If the value is negative, make it positiv
e and indicate

```

```

997.          // that a minus sign is needed.
998.          //
999.          if((int32_t)ui32Value < 0)
1000.         {
1001.             //
1002.             // Make the value positive.
1003.             //
1004.             ui32Value = -(int32_t)ui32Value;
1005.
1006.             //
1007.             // Indicate that the value is negative.
1008.
1009.             //
1010.             ui32Neg = 1;
1011.         }
1012.         else
1013.         {
1014.             //
1015.             // Indicate that the value is positive s
1016.             o that a minus
1017.             // sign isn't inserted.
1018.             //
1019.             ui32Neg = 0;
1020.         }
1021.
1022.         //
1023.         // Set the base to 10.
1024.         //
1025.         ui32Base = 10;
1026.
1027.         //
1028.         // Convert the value to ASCII.
1029.         //
1030.         goto convert;
1031.     }
1032.
1033.     //
1034.     // Handle the %s command.
1035.     //
1036.     case 's':
1037.     {
1038.         //
1039.         // Get the string pointer from the varargs.
1040.
1041.         //
1042.         pcStr = va_arg(vaArgP, char *);
1043.
1044.         //
1045.         // Determine the length of the string.
1046.         //
1047.         for(ui32Idx = 0; pcStr[ui32Idx] != '\0'; ui3
1048.             2Idx++)
1049.         {
1050.             //
1051.             // Write the string.
1052.             //
1053.             UARTwrite(pcStr, ui32Idx);

```

```

1052.
1053.          //
1054.          // Write any required padding spaces
1055.          //
1056.          if(ui32Count > ui32Idx)
1057.          {
1058.              ui32Count -= ui32Idx;
1059.              while(ui32Count--)
1060.              {
1061.                  UARTwrite(" ", 1);
1062.              }
1063.          }
1064.
1065.          //
1066.          // This command has been handled.
1067.          //
1068.          break;
1069.      }
1070.
1071.      //
1072.      // Handle the %u command.
1073.      //
1074.      case 'u':
1075.      {
1076.          //
1077.          // Get the value from the varargs.
1078.          //
1079.          ui32Value = va_arg(vaArgP, uint32_t);
1080.
1081.          //
1082.          // Reset the buffer position.
1083.          //
1084.          ui32Pos = 0;
1085.
1086.          //
1087.          // Set the base to 10.
1088.          //
1089.          ui32Base = 10;
1090.
1091.          //
1092.          // Indicate that the value is positive so th
          at a minus sign
1093.          // isn't inserted.
1094.          //
1095.          ui32Neg = 0;
1096.
1097.          //
1098.          // Convert the value to ASCII.
1099.          //
1100.          goto convert;
1101.      }
1102.
1103.      //
1104.      // Handle the %x and %X commands. Note that the
          y are treated
1105.          // identically; in other words, %X will use lowe
          r case letters
1106.          // for a-
          f instead of the upper case letters it should use. We

```

```

1107.          // also alias %p to %x.
1108.          //
1109.          case 'x':
1110.          case 'X':
1111.          case 'p':
1112.          {
1113.              //
1114.              // Get the value from the varargs.
1115.              //
1116.              ui32Value = va_arg(vaArgP, uint32_t);
1117.              //
1118.              // Reset the buffer position.
1119.              //
1120.              //
1121.              ui32Pos = 0;
1122.              //
1123.              //
1124.              // Set the base to 16.
1125.              //
1126.              ui32Base = 16;
1127.              //
1128.              //
1129.              // Indicate that the value is positive so th
    at a minus sign
1130.              // isn't inserted.
1131.              //
1132.              ui32Neg = 0;
1133.              //
1134.              //
1135.              // Determine the number of digits in the str
    ing version of
1136.              // the value.
1137.              //
1138.          convert:
1139.              for(ui32Idx = 1;
1140.                  (((ui32Idx * ui32Base) <= ui32Value) &&
1141.                     (((ui32Idx * ui32Base) / ui32Base) == u
    i32Idx));
1142.                  ui32Idx *= ui32Base, ui32Count--)
1143.                  {
1144.                  }
1145.              //
1146.              // If the value is negative, reduce the coun
    t of padding
1147.              // characters needed.
1148.              //
1149.              if(ui32Neg)
1150.              {
1151.                  ui32Count--;
1152.              }
1153.              //
1154.              //
1155.              // If the value is negative and the value is
    padded with
1156.              // zeros, then place the minus sign before t
    he padding.
1157.              //
1158.              //

```

```

1159.         if(ui32Neg && (cFill == '0'))
1160.         {
1161.             //
1162.             // Place the minus sign in the output buffer.
1163.             //
1164.             pcBuf[ui32Pos++] = '-';
1165.             //
1166.             // The minus sign has been placed, so turn off the
1167.             // negative flag.
1168.             //
1169.             ui32Neg = 0;
1170.         }
1171.         //
1172.         // Provide additional padding at the beginning of the
1173.         // string conversion if needed.
1174.         //
1175.         if((ui32Count > 1) && (ui32Count < 16))
1176.         {
1177.             for(ui32Count--; ui32Count; ui32Count--)
1178.             {
1179.                 pcBuf[ui32Pos++] = cFill;
1180.             }
1181.             //
1182.             // If the value is negative, then place the
1183.             // minus sign
1184.             // before the number.
1185.             //
1186.             if(ui32Neg)
1187.             {
1188.                 //
1189.                 // Place the minus sign in the output buffer.
1190.                 //
1191.                 pcBuf[ui32Pos++] = '-';
1192.             }
1193.             //
1194.             // Convert the value into a string.
1195.             //
1196.             for(; ui32Idx; ui32Idx /= ui32Base)
1197.             {
1198.                 pcBuf[ui32Pos++] =
1199.                     g_pCHex[(ui32Value / ui32Idx) % ui32
1200.                         Base];
1201.             }
1202.             //
1203.             // Write the string.
1204.             //
1205.             UARTwrite(pcBuf, ui32Pos);
1206.         }
1207.

```

```

1211.          //
1212.          // This command has been handled.
1213.          //
1214.          break;
1215.      }
1216.
1217.          //
1218.          // Handle the %% command.
1219.          //
1220.          case '%':
1221.          {
1222.              //
1223.              // Simply write a single %.
1224.              //
1225.              UARTwrite(pcString - 1, 1);
1226.
1227.              //
1228.              // This command has been handled.
1229.              //
1230.              break;
1231.          }
1232.
1233.          //
1234.          // Handle all other commands.
1235.          //
1236.          default:
1237.          {
1238.              //
1239.              // Indicate an error.
1240.              //
1241.              UARTwrite("ERROR", 5);
1242.
1243.              //
1244.              // This command has been handled.
1245.              //
1246.              break;
1247.          }
1248.      }
1249.  }
1250.  }
1251.  }
1252.
1253.  //*****
1254.  //
1255.  ///! A simple UART based printf function supporting %c, %d, %p
1256.  ///! %s, %u,
1257.  ///! %x, and %X.
1258.  ///!
1259.  ///! \param pcString is the format string.
1260.  ///! \param ... are the optional arguments, which depend on the c
1261.  ///! contents of the
1262.  ///! format string.
1263.  ///!
1264.  ///! This function is very similar to the C library <tt>fprintf()
1265.  ///! function.
1266.  ///! All of its output will be sent to the UART. Only the follow
1267.  ///! ing formatting
1268.  ///! characters are supported:

```



```

1265.    //!<
1266.    //!< - %c to print a character
1267.    //!< - %d or %i to print a decimal value
1268.    //!< - %s to print a string
1269.    //!< - %u to print an unsigned decimal value
1270.    //!< -
1271.    //!< -
1272.    //!< %x to print a hexadecimal value using lower case letters
1273.    //!< %X to print a hexadecimal value using lower case letters (not upper
1274.    case
1275.    //!< letters as would typically be used)
1276.    //!< - %p to print a pointer as a hexadecimal value
1277.    //!< - %% to print out a % character
1278.    //!<
1279.    //!< For %s, %d, %i, %u, %p, %x, and %X, an optional number may reside
1280.    //!< between the % and the format character, which specifies the
1281.    minimum number
1282.    //!< of characters to use for that value; if preceded by a 0 then
1283.    the extra
1284.    //!< characters will be filled with zeros instead of spaces. For
1285.    example,
1286.    //!< %%8d will use eight characters to print the decimal value with
1287.    spaces
1288.    //!< added to reach eight; %%08d will use eight characters as well but
1289.    will
1290.    //!< add zeroes instead of spaces.
1291.    //!<
1292.    //!< The type of the arguments after %pcString must match the requirements
1293.    of
1294.    //!< the format string. For example, if an integer was passed where a
1295.    string
1296.    //!< was expected, an error of some kind will most likely occur.
1297.    //!<
1298.    //!<
1299.    //!< \return None.
1300.    //!<
1301.    //!<
1302.    //!<*****
1303.    void
1304.    UARTprintf(const char *pcString, ...)
1305.    {
1306.        va_list vaArgP;
1307.
1308.        //
1309.        // Start the varargs processing.
1310.        //
1311.        va_start(vaArgP, pcString);
1312.
1313.        UARTvprintf(pcString, vaArgP);
1314.
1315.        //
1316.        // We're finished with the varargs now.
1317.        //
1318.        va_end(vaArgP);
1319.    }
1320.
1321.    //!<*****

```

```

1310.    //
1311.    /// Returns the number of bytes available in the receive buffer.

1312.    ///
1313.    /// This function, available only when the module is built to op
    erate in
1314.    /// buffered mode using \b UART_BUFFERED, may be used to determi
    ne the number
1315.    /// of bytes of data currently available in the receive buffer.

1316.    ///
1317.    /// \return Returns the number of available bytes.
1318.    //
1319.    //*****
    *****
1320.    #if defined(UART_BUFFERED) || defined(DOXYGEN)
1321.    int
1322.    UARTRxBytesAvail(void)
1323.    {
1324.        return(RX_BUFFER_USED);
1325.    }
1326.    #endif
1327.
1328.    #if defined(UART_BUFFERED) || defined(DOXYGEN)
1329.    //*****
    *****
1330.    //
1331.    /// Returns the number of bytes free in the transmit buffer.
1332.    ///
1333.    /// This function, available only when the module is built to op
    erate in
1334.    /// buffered mode using \b UART_BUFFERED, may be used to determi
    ne the amount
1335.    /// of space currently available in the transmit buffer.
1336.    ///
1337.    /// \return Returns the number of free bytes.
1338.    //
1339.    //*****
    *****
1340.    int
1341.    UARTRxBytesFree(void)
1342.    {
1343.        return(TX_BUFFER_FREE);
1344.    }
1345.    #endif
1346.
1347.    //*****
    *****
1348.    //
1349.    /// Looks ahead in the receive buffer for a particular character
    .
1350.    ///
1351.    /// \param ucChar is the character that is to be searched for.
1352.    ///
1353.    /// This function, available only when the module is built to op
    erate in
1354.    /// buffered mode using \b UART_BUFFERED, may be used to look ah
    ead in the

```

```

1355.    //!< receive buffer for a particular character and report its pos
        tion if found.
1356.    //!< It is typically used to determine whether a complete line of
        user input is
1357.    //!< available, in which case ucChar should be set to CR ('\r')
        which is used
1358.    //!< as the line end marker in the receive buffer.
1359.    //!<
1360.    //!< \return Returns -
        1 to indicate that the requested character does not exist
1361.    //!< in the receive buffer. Returns a non-
        negative number if the character was
1362.    //!< found in which case the value represents the position of the
        first instance
1363.    //!< of \e ucChar relative to the receive buffer read pointer.
1364.    //!<
1365.    //!<*****
        *****
1366.    #if defined(UART_BUFFERED) || defined(DOXYGEN)
1367.    int
1368.    UARTPeek(unsigned char ucChar)
1369.    {
1370.        int iCount;
1371.        int iAvail;
1372.        uint32_t ui32ReadIndex;
1373.
1374.        //
1375.        // How many characters are there in the receive buffer?
1376.        //
1377.        iAvail = (int)RX_BUFFER_USED;
1378.        ui32ReadIndex = g_ui32UARTRxReadIndex;
1379.
1380.        //
1381.        // Check all the unread characters looking for the one passe
        d.
1382.        //
1383.        for(iCount = 0; iCount < iAvail; iCount++)
1384.        {
1385.            if(g_pcUARTRxBuffer[ui32ReadIndex] == ucChar)
1386.            {
1387.                //
1388.                // We found it so return the index
1389.                //
1390.                return(iCount);
1391.            }
1392.            else
1393.            {
1394.                //
1395.                // This one didn't match so move on to the next char
        acter.
1396.                //
1397.                ADVANCE_RX_BUFFER_INDEX(ui32ReadIndex);
1398.            }
1399.        }
1400.
1401.        //
1402.        // If we drop out of the loop, we didn't find the character
        in the receive
1403.        // buffer.

```

```

1404.      //
1405.      return(-1);
1406.  }
1407.  #endif
1408.
1409.  //*****
1410.  //
1411.  //! Flushes the receive buffer.
1412.  //!
1413.  //! This function, available only when the module is built to op
    erate in
1414.  //! buffered mode using \b UART_BUFFERED, may be used to discard
    any data
1415.  //! received from the UART but not yet read using UARTgets().
1416.  //!
1417.  //! \return None.
1418.  //
1419.  //*****
1420.  #if defined(UART_BUFFERED) || defined(DOXYGEN)
1421.  void
1422.  UARTFlushRx(void)
1423.  {
1424.      uint32_t ui32Int;
1425.
1426.      //
1427.      // Temporarily turn off interrupts.
1428.      //
1429.      ui32Int = MAP_IntMasterDisable();
1430.
1431.      //
1432.      // Flush the receive buffer.
1433.      //
1434.      g_ui32UARTRxReadIndex = 0;
1435.      g_ui32UARTRxWriteIndex = 0;
1436.
1437.      //
1438.      // If interrupts were enabled when we turned them off, turn
    them
1439.      // back on again.
1440.      //
1441.      if(!ui32Int)
1442.      {
1443.          MAP_IntMasterEnable();
1444.      }
1445.  }
1446.  #endif
1447.
1448.  //*****
1449.  //
1450.  //! Flushes the transmit buffer.
1451.  //!
1452.  //! \param bDiscard indicates whether any remaining data in the
    buffer should
1453.  //! be discarded (\b true) or transmitted (\b false).
1454.  //

```

```

1455.    //!< This function, available only when the module is built to op
        erate in
1456.    //!< buffered mode using \b UART_BUFFERED, may be used to flush t
        he transmit
1457.    //!< buffer, either discarding or transmitting any data received
        via calls to
1458.    //!< UARTprintf() that is waiting to be transmitted. On return,
        the transmit
1459.    //!< buffer will be empty.
1460.    //!<
1461.    //!< \return None.
1462.    //!<
1463.    //!<*****
        *****
1464.    #if defined(UART_BUFFERED) || defined(DOXYGEN)
1465.    void
1466.    UARTFlushTx(bool bDiscard)
1467.    {
1468.        uint32_t ui32Int;
1469.
1470.        //
1471.        // Should the remaining data be discarded or transmitted?
1472.        //
1473.        if(bDiscard)
1474.        {
1475.            //
1476.            // The remaining data should be discarded, so temporaril
                y turn off
1477.            // interrupts.
1478.            //
1479.            ui32Int = MAP_IntMasterDisable();
1480.
1481.            //
1482.            // Flush the transmit buffer.
1483.            //
1484.            g_ui32UARTTxReadIndex = 0;
1485.            g_ui32UARTTxWriteIndex = 0;
1486.
1487.            //
1488.            // If interrupts were enabled when we turned them off, t
                urn them
1489.            // back on again.
1490.            //
1491.            if(!ui32Int)
1492.            {
1493.                MAP_IntMasterEnable();
1494.            }
1495.        }
1496.        else
1497.        {
1498.            //
1499.            // Wait for all remaining data to be transmitted before
                returning.
1500.            //
1501.            while(!TX_BUFFER_EMPTY)
1502.            {
1503.            }
1504.        }
1505.    }

```

```

1506.     #endif
1507.
1508.     //*****
1509.     //
1510.     //! Enables or disables echoing of received characters to the tr
1511.     nsmitter.
1512.     //! \param bEnable must be set to \b true to enable echo or \b f
1513.     else to
1514.     //! disable it.
1515.     //! This function, available only when the module is built to op
1516.     erate in
1517.     //! buffered mode using \b UART_BUFFERED, may be used to control
1518.     whether or not
1519.     //! received characters are automatically echoed back to the tra
1520.     nsmitter. By
1521.     //! default, echo is enabled and this is typically the desired b
1522.     ehavior if
1523.     //! the module is being used to support a serial command line.
1524.     In applications
1525.     //! where this module is being used to provide a convenient, buf
1526.     fered serial
1527.     //! interface over which application-
1528.     specific binary protocols are being run,
1529.     //! however, echo may be undesirable and this function can be us
1530.     ed to disable
1531.     //! it.
1532.     //!
1533.     //! \return None.
1534.     //
1535.     //*****
1536.     #if defined(UART_BUFFERED) || defined(DOXYGEN)
1537.     void
1538.     UAREchoSet(bool bEnable)
1539.     {
1540.         g_bDisableEcho = !bEnable;
1541.     }
1542.     #endif
1543.
1544.     //*****
1545.     //
1546.     //! Handles UART interrupts.
1547.     //!
1548.     //! This function handles interrupts from the UART. It will cop
1549.     y data from the
1550.     //! transmit buffer to the UART transmit FIFO if space is availa
1551.     ble, and it
1552.     //! will copy data from the UART receive FIFO to the receive buf
1553.     fer if data is
1554.     //! available.
1555.     //!
1556.     //! \return None.
1557.     //
1558.     //*****

```

```

1548.     #if defined(UART_BUFFERED) || defined(DOXYGEN)
1549.     void
1550.     UARTStdioIntHandler(void)
1551.     {
1552.         uint32_t ui32Ints;
1553.         int8_t cChar;
1554.         int32_t i32Char;
1555.         static bool bLastWasCR = false;
1556.
1557.         //
1558.         // Get and clear the current interrupt source(s)
1559.         //
1560.         ui32Ints = MAP_UARTIntStatus(g_ui32Base, true);
1561.         MAP_UARTIntClear(g_ui32Base, ui32Ints);
1562.
1563.         //
1564.         // Are we being interrupted because the TX FIFO has space av
1565.         //
1566.         if(ui32Ints & UART_INT_TX)
1567.         {
1568.             //
1569.             // Move as many bytes as we can into the transmit FIFO.
1570.             //
1571.             UARTPrimeTransmit(g_ui32Base);
1572.
1573.             //
1574.             // If the output buffer is empty, turn off the transmit
1575.             //
1576.             if(TX_BUFFER_EMPTY)
1577.             {
1578.                 MAP_UARTIntDisable(g_ui32Base, UART_INT_TX);
1579.             }
1580.         }
1581.
1582.         //
1583.         // Are we being interrupted due to a received character?
1584.         //
1585.         if(ui32Ints & (UART_INT_RX | UART_INT_RT))
1586.         {
1587.             //
1588.             // Get all the available characters from the UART.
1589.             //
1590.             while(MAP_UARTCharsAvail(g_ui32Base))
1591.             {
1592.                 //
1593.                 // Read a character
1594.                 //
1595.                 i32Char = MAP_UARTCharGetNonBlocking(g_ui32Base);
1596.                 cChar = (unsigned char)(i32Char & 0xFF);
1597.
1598.                 //
1599.                 // If echo is disabled, we skip the various text fil
1600.                 //
1601.                 // operations that would typically be required when
1602.                 // supporting a
1603.                 // command line.

```

```

1602.          //
1603.          if(!g_bDisableEcho)
1604.          {
1605.              //
1606.              // Handle backspace by erasing the last character
1607.              // in the
1608.              // buffer.
1609.              //
1610.              if(cChar == '\b')
1611.              {
1612.                  //
1613.                  // If there are any characters already in the
1614.                  // buffer, then
1615.                  // delete the last.
1616.                  //
1617.                  if(!RX_BUFFER_EMPTY)
1618.                  {
1619.                      //
1620.                      // Rub out the previous character on the
1621.                      // users
1622.                      // terminal.
1623.                      //
1624.                      UARTwrite("\b \b", 3);
1625.                      //
1626.                      // Decrement the number of characters in
1627.                      // the buffer.
1628.                      //
1629.                      if(g_ui32UARTRxWriteIndex == 0)
1630.                      {
1631.                          g_ui32UARTRxWriteIndex = UART_RX_BUFFER_SIZE - 1;
1632.                      }
1633.                      else
1634.                      {
1635.                          g_ui32UARTRxWriteIndex--;
1636.                      }
1637.                  }
1638.                  //
1639.                  // Skip ahead to read the next character.
1640.                  //
1641.                  continue;
1642.              }
1643.              //
1644.              // If this character is LF and last was CR, then
1645.              // just gobble up
1646.              // the character since we already echoed the previous
1647.              // CR and we
1648.              // don't want to store 2 characters in the buffer
1649.              // if we don't
1650.              // need to.
1651.              //
1652.              if((cChar == '\n') && bLastWasCR)
1653.              {
1654.                  bLastWasCR = false;
1655.                  continue;
1656.              }

```



```

1653.
1654.          //
1655.          // See if a newline or escape character was rece
ived.
1656.          //
1657.          if((cChar == '\r') || (cChar == '\n') || (cChar
== 0x1b))
1658.          {
1659.              //
1660.              // If the character is a CR, then it may be
followed by an
1661.              // LF which should be paired with the CR. S
o remember that
1662.              // a CR was received.
1663.              //
1664.              if(cChar == '\r')
1665.              {
1666.                  bLastWasCR = 1;
1667.              }
1668.
1669.              //
1670.              // Regardless of the line termination charac
ter received,
1671.              // put a CR in the receive buffer as a marke
r telling
1672.              // UARTgets() where the line ends. We also
send an
1673.              // additional LF to ensure that the local te
rminal echo
1674.              // receives both CR and LF.
1675.              //
1676.              cChar = '\r';
1677.              UARTwrite("\n", 1);
1678.          }
1679.      }
1680.
1681.      //
1682.      // If there is space in the receive buffer, put the
character
1683.      // there, otherwise throw it away.
1684.      //
1685.      if(!RX_BUFFER_FULL)
1686.      {
1687.          //
1688.          // Store the new character in the receive buffer
1689.          //
1690.          g_pcUARTRxBuffer[g_ui32UARTRxWriteIndex] =
(unsigned char)(i32Char & 0xFF);
1691.          ADVANCE_RX_BUFFER_INDEX(g_ui32UARTRxWriteIndex);
1692.
1693.          //
1694.          // If echo is enabled, write the character to th
e transmit
1695.          // buffer so that the user gets some immediate f
eedback.
1696.          //
1697.          if(!g_bDisableEcho)
1698.

```

```

1699.          {
1700.              UARTwrite((const char *)&cChar, 1);
1701.          }
1702.      }
1703.  }
1704.
1705.      //
1706.      // If we wrote anything to the transmit buffer, make sur
1707.      e it actually
1708.      // gets transmitted.
1709.      //
1710.      UARTPrimeTransmit(g_ui32Base);
1711.      MAP_UARTIntEnable(g_ui32Base, UART_INT_TX);
1712.  }
1713.  #endif
1714.
1715.  /*******
1716.  //
1717.  // Close the Doxygen group.
1718.  //! @}
1719.  //
1720.  /*******

```

Código fuente 12.- Archivo con el programa de implementación de funciones relacionadas a algunas funciones de comunicación UART.

Archivo "watchdog.c"

```

1.  /*
2.   * watchdog_func.c
3.   *
4.   * Created on: 10 de dic. de 2016
5.   * Author: Francisco
6.   */
7.
8.  #include <stdint.h>
9.  #include <stdbool.h>
10. #include "inc/hw_ints.h"
11. #include "inc/hw_memmap.h"
12. #include "inc/hw_types.h"
13. #include "driverlib/debug.h"
14. #include "driverlib/fpu.h"
15. #include "driverlib/gpio.h"
16. #include "driverlib/interrupt.h"
17. #include "driverlib/sysctl.h"
18. #include "driverlib/watchdog.h"
19. #include "driverlib/rom_map.h"
20. #include "driverlib/rom.h"
21.
22. #include "watchdog_func.h"
23.
24. void WatchdogIntHandler(void)
25. {
26.     //

```

```

27. // If we have been told to stop feeding the watchdog, return immedi-
    ately
28. // without clearing the interrupt. This will cause the system to
    reset
29. // next time the watchdog interrupt fires.
30. //
31. //if(!feedWatchdog2)
32. //{
33.     //feedWatchdog2 = true;
34.     // return;
35. //}
36.
37. if(reset_sys == 8){
38.
39.     return;
40. }
41.
42. reset_sys++;
43. //
44. // Clear the watchdog interrupt.
45. //
46. WatchdogIntClear(WATCHDOG0_BASE);
47.}
48.
49. void habilita_watchdog0(uint8_t segun){
50.
51.     // Enable the watchdog peripheral
52.     MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);
53.
54.     // Set timeout of watchdog to 2 sec
55.     MAP_WatchdogReloadSet(WATCHDOG0_BASE, SysCtlClockGet() * segun);
56.
57.     // Reset when watchdog expires
58.     MAP_WatchdogResetEnable(WATCHDOG0_BASE);
59.
60.     // Register the watchdog interrupt handler
61.     WatchdogIntRegister(WATCHDOG0_BASE, &WatchdogIntHandler);
62.
63.     // Enable the watchdog
64.     MAP_WatchdogEnable(WATCHDOG0_BASE);
65.}

```

Código fuente 13.- Archivo con el programa de implementación de funciones relacionadas al manejo del watchdog.

Archivo "watchdog.h"

```

1. /*
2.  * watchdog_func.h
3.  *
4.  * Created on: 10 de dic. de 2016
5.  * Author: Francisco
6.  */
7.
8. #ifndef WATCHDOG_FUNC_H_
9. #define WATCHDOG_FUNC_H_
10.

```

```

11. #include <stdint.h>
12. #include <stdbool.h>
13. #include "inc/hw_ints.h"
14. #include "inc/hw_memmap.h"
15. #include "inc/hw_types.h"
16. #include "driverlib/debug.h"
17. #include "driverlib/fpu.h"
18. #include "driverlib/gpio.h"
19. #include "driverlib/interrupt.h"
20. #include "driverlib/sysctl.h"
21. #include "driverlib/watchdog.h"
22. #include "driverlib/rom_map.h"
23. #include "driverlib/rom.h"
24.
25. uint8_t reset_sys;
26.
27. void habilita_watchdog0(uint8_t segun);
28. void WatchdogIntHandler(void);
29.
30.
31. #endif /* WATCHDOG_FUNC_H_ */

```

Código fuente 14.- Archivo con el programa de cabeceras de funciones relacionadas al manejo del watchdog.

ANEXO B

Archivos del proyecto para toma de datos, fases del programa principal y gráfica de datos del módulo sensor de gestos.

Programa en Python para gráfica de datos y depurar código

```
1.  ....
2.  Created on 28/3/2018
3.
4.  @author: Francisco
5.  '''
6.  import serial
7.  import numpy
8.  import matplotlib
9.  #matplotlib.use('TkAgg')
10. import matplotlib.pyplot as plt
11.
12. from drawnow import *
13. #import struct
14.
15. rangosleft  = []
16. rangosright = []
17. cnt = 0
18.
19. tivadata=serial.Serial('com20',230400,timeout=1)
20. plt.ion()
21.
22. UMBRAL = 110
23.
24. def makeFig():
25.     #plt.ylim(-1,1.5)
26.     plt.ylim(-1,300)
27.     plt.xlim(-1,100)
28.     plt.title("Lectura de Rangos del Sensor de gestos")
29.     plt.grid(True)
30.     plt.ylabel("Valores")
31.     plt.plot(rangosleft,'bo-',label="valores Abajo")
32.     plt.legend(loc="upper left")
33.     plt2=plt.twinx()
34.     #plt.ylim(-1,1.5)
35.     plt.ylim(-1,300)
36.     plt2.plot(rangosright,'r^-',label="valores Arriba")
37.     plt2.ticklabel_format(useOffset=False)
38.     plt2.legend(loc="upper right")
39.     plt.pause(.000001)
40.
41. while True:
42.     while(tivadata.inWaiting()==0):
43.         pass
44.     tivabyte = tivadata.read(1)
45.     if(tivabyte==bytes(b'\xF2')):
46.         arg3=tivadata.read(1)
47.         print(" v_fase: "+str(arg3.hex()))
48.     #print(tivabyte.hex())
49.     #drawnow(makeFig)
50.     if(int.from_bytes(tivabyte,byteorder='big')>240):
51.         if(tivabyte==bytes(b'\xFE')):
```

```

52.         arg1=tivadata.read(1)
53.         arg2=tivadata.read(1)
54.
55.
56.         #if(int(arg1.hex(),16)>UMBRAL):
57.         #     rangosleft.append(1)
58.         #else:
59.         #     rangosleft.append(0)
60.         #if(int(arg2.hex(),16)>UMBRAL):
61.         #     rangosright.append(1)
62.         #else:
63.         #     rangosright.append(0)
64.         rangosleft.append(float(int(arg1.hex(),16)))
65.         rangosright.append(float(int(arg2.hex(),16)))
66.
67.         #mensaje = str(tivabyte) + str(arg1) + str(arg2)
68.         #print("Comando "+mensaje)
69.
70.         mensajehex = str(tivabyte.hex()) + str(arg1.hex()) + s
tr(arg2.hex())
71.         print("Comando "+mensajehex)
72.         drawnow(makeFig)
73.         plt.pause(.000001)
74.         cnt=cnt+1
75.         if(cnt>100):
76.             rangosleft.pop(0)
77.             rangosright.pop(0)

```

Código fuente 15.- Archivo con el programa de depuración y análisis de datos desde sensor de gestos.

ANEXO C

Código principal para obtener arreglo de bits con presencia o no de color de un caracter alfanumérico.

Programa en Python para obtener arreglo de presencia o no de colores de un caracter

```
1.  """
2.  Created on 9/10/2016
3.
4.  @author: Francisco
5.  """
6.
7.
8.  font_size = 96
9.  WIDTH_font, HEIGHT_font = 64, 71
10. offset_x_font, offset_y_font = WIDTH_font/2, HEIGHT_font/2
11.
12.
13. cont = 1
14. f = open('C:\\Users\\Francisco\\Pictures\\Test_Fonts\\workfile_'+str(c
    ont)+'.txt', 'w')
15.
16. from PIL import Image, ImageFont, ImageDraw
17. import os
18.
19.
20. im = Image.new('RGBA', (WIDTH_font, HEIGHT_font), 'black')
21. draw = ImageDraw.Draw(im)
22.
23. fontsFolder = "C:\\Users\\Francisco\\Documents\\Francisco_tesis\\REDAC
    CIONES_IMPLEMENTACION\\TFT_display\\Python_Raspberry\\arial_narrow_7_9
    348"
24. arialFont = ImageFont.truetype(os.path.join(fontsFolder, 'arial.ttf'),
    font_size)
25.
26. text = draw.text((0, -17), "S", fill='white', font=arialFont)
27.
28. print(arialFont.getsize("S"))
29.
30.
31. im.save("C:\\Users\\Francisco\\Pictures\\Test_Fonts\\font_num_1.png",
    "PNG")
32.
33.
34. def to_8bits(datas):
35.     """Convert red, green, blue components to a 16-
        bit 565 RGB value. Components
36.     should be values 0 to 255.
37.     """
38.
39.
40.
41. im = im.convert("RGBA")
42. datas = im.getdata()
43.
44. #print(datas[1][0])
```

```

45.resul_8bits = 0
46.newData = []
47.j = 0
48.i = 0
49.cont = 0
50.
51.newData_file = []
52.
53.for item in datas:
54.    #print(item)
55.
56.    if item[0] == 0 and item[1] == 0 and item[2] == 0:
57.        resul_8bits = 0+(resul_8bits<<1)
58.    else:
59.        resul_8bits = 1+(resul_8bits<<1)
60.
61.    if(j==7):
62.        newData.append((resul_8bits,cont))
63.        newData_file.append(resul_8bits)
64.        resul_8bits = 0
65.        j = 0
66.    else:
67.        j += 1
68.
69.
70.    if(((i+1)%WIDTH_font)==0):
71.        cont += 1
72.        i = 0
73.    else:
74.        i += 1
75.
76.for item in newData_file:
77.    f.write("0x")
78.    f.write(str(format(item, '02x')))
79.    f.write(",")
80.
81.print (newData)
82.#f.write("Hola\n")
83.#f.write("Mundo")
84.#f.write(str(newData))
85.
86.
87.from PIL import ImageTk
88.from tkinter import Tk, Canvas, mainloop
89.#from math import sin
90.
91.
92.
93.WIDTH_canvas, HEIGHT_canvas = 56, 71
94.
95.window = Tk()
96.canvas = Canvas(window, width=WIDTH_canvas, height=HEIGHT_canvas, bg="
    #000000")
97.
98.
99.im = Image.open(r'C:\\Users\\Francisco\\Pictures\\Test_Fonts\\font_num
    _1.png')
100.
101.    tking = ImageTk.PhotoImage(im)

```



```

102.
103.
104.     #Centra la imagen en WIDTH/2 y HEIGHT/2
105.     canvas.create_image((WIDTH_canvas//2+2, HEIGHT_canvas//2+2), ima
    ge=tkimg, state="normal")
106.
107.     canvas.pack()
108.
109.     mainloop()

```

Código fuente 16.- Archivo con el programa para obtener arreglo de presencia o no de color de un caracter.

```

1.  ....
2.  Created on 8/10/2016
3.
4.  @author: Francisco
5.  '''
6.
7.  from PIL import Image,ImageFont, ImageDraw
8.  import os
9.
10.
11. im = Image.new('RGBA', (133, 149), 'black')
12. draw = ImageDraw.Draw(im)
13.
14. fontsFolder = "C:\\Users\\Francisco\\Documents\\Francisco_tesis\\REDAC
    CIONES_IMPLEMENTACION\\TFT_display\\Python_Raspberry\\arial_narrow_7_9
    348"
15. arialFont = ImageFont.truetype(os.path.join(fontsFolder, 'arial.ttf'),
    200)
16.
17. draw.text((0, -36), "1",fill='white', font=arialFont)
18. im.save("C:\\Users\\Francisco\\Pictures\\Test_Fonts\\img2.png", "PNG")
19.
20.
21. def to_8bits(datas):
22.     """Convert red, green, blue components to a 16-
    bit 565 RGB value. Components
23.     should be values 0 to 255.
24.     """
25.
26.
27.
28. im = im.convert("RGBA")
29. datas = im.getdata()
30.
31. #print(datas[1][0])
32. resul_8bits = 0
33. newData = []
34. j = 0
35. i = 0
36. cont = 0
37.
38. for item in datas:
39.     #print(item)
40.

```

```

41.     if item[0] == 0 and item[1] == 0 and item[2] == 0:
42.         resul_8bits = 0+(resul_8bits<<1)
43.     else:
44.         resul_8bits = 1+(resul_8bits<<1)
45.
46.     if(j==7):
47.         newData.append((resul_8bits,cont))
48.         resul_8bits = 0
49.         j = 0
50.     else:
51.         j += 1
52.
53.
54.     if(((i+1)%200)==0):
55.         cont += 1
56.         i = 0
57.     else:
58.         i += 1
59.
60. print (newData)
61.
62.
63.
64. from PIL import ImageTk
65. from tkinter import Tk, Canvas, PhotoImage, mainloop
66. #from math import sin
67.
68.
69.
70. WIDTH, HEIGHT = 240, 320
71.
72. window = Tk()
73. canvas = Canvas(window, width=WIDTH, height=HEIGHT, bg="#000000")
74.
75. im = Image.open(r'C:\\Users\\Francisco\\Pictures\\Test_Fonts\\img2.png
76. ')
77. tkimg = ImageTk.PhotoImage(im)
78. img = PhotoImage(width=WIDTH, height=HEIGHT)
79.
80. ....
81. for y in range(96):
82.     print (y)
83.     for x in range(56):
84.         print (x,y)
85.         img.put("#ffffff", (x+92,y+32))
86. ...
87.
88. #para dibujar una linea en la mitad de la pantalla de 240x320
89. for x in range(WIDTH):
90.     y = int(HEIGHT/2)
91.     img.put("#ffffff", (x,y))
92.
93.
94. for x in range(100):
95.     y = 191+x
96.     #print(y)
97.     #print(x)
98.     for i in range(x):

```

```

99.         img.put("#ffffff", (i+120,y))
100.         img.put("#ffffff", (120-i,y))
101.
102.
103.         #Centra la imagen en WIDTH/2 y HEIGHT/2
104.         canvas.create_image((119, 123-
41), image=tkimg, state="normal")
105.         canvas.create_image((WIDTH/2, HEIGHT/2), image=img, state="norma
1")
106.
107.         canvas.pack()
108.
109.         #canvas_id = canvas.create_text(10, 10, anchor="nw")
110.         #canvas.itemconfig(canvas_id, text="this is the text")
111.         #canvas.insert(canvas_id, 12, "new ")
112.
113.
114.
115.
116.
117.         mainloop()
118.
119.
120.         ....
121.         Tiva code to store data in flash
122.         HWREG(FLASH_FMA) = ADDRESS; // Program Address to write
123.
124.         HWREG(FLASH_FMD) = DATA; // Program Data to write
125.
126.         HWREG(FLASH_FMC) = FLASH_FMC_WRKEY|FLASH_FMC_WRITE); // Start wr
ite to flash
127.
128.         while((HWREG(FLASH_FCRIS) & FLASH_FCRIS_PRIS) != FLASH_FCRIS_PRI
S); // Wait for operation to complete
129.
130.         HWREG(FLASH_FCRIS) = FLASH_FCRIS_PRIS; // clear the completion f
lag for the next iteration
131.
132.         ...

```

Código fuente 17.- Archivo con el programa para representar una imagen o caracter en un lienzo de resolución deseada.

ANEXO D

Lista de materiales y precios para el dispositivo de piso

Reference	Description	Value	Package	Manufacture Part Number	Type	Cost	Cost (x1000)
X1, CAN	TERM BLOCK HDR 2POS 90DEG 5MM	1757475	MALE 2POS - PITCH 5MM	1757475	THRU-HOLE	0.66	0.4675
C3, C4, C6, C9, C10, C11, C13	CAP CER 0.1UF 50V Y5V 0603	100n	0603	CL10F104ZB8NNNC	SMT	0.1	0.0043
D22	TVS DIODE 24VWM 40VC SOT23-3L	ESDCAN24-2BLY	SOT23-3	ESDCAN24-2BLY	SMT	0.45	0.11124
D1, D4, D5	DIODE SCHOTTKY 40V 1A USC	CUS10S40,H3F	SOD-323	CUS10S40,H3F	SMT	0.43	0.0796
PPTC1	PTC RESTTBLE 0.20A 30V CHIP 1210	200mA	1812	OZCB0020FF2E	SMT	0.24	0.13224
IC1	CONV DC/DC 5V 500MA OUT THRU	R-78E5.0-0.5	3-SIP Module	R-78E5.0-0.5	THRU-HOLE	2.84	2.268
D3	TVS DIODE 28VWM 45.4VC SMA	SMAJ28A-E3/61	DO-214AC, SMA	SMAJ28A-E3/5A	SMT	0.4	0.12796
L1	FIXED IND 10UH 550MA 500 MOHM	10u	L1812	NLC453232T-100K-PF	SMT	0.46	0.2724
U1	IC TRANSCEIVER CAN HI-SPD 8-DIP	MCP2551-I/P	8-DIP (0.300", 7.62mm)	MCP2551-I/P	THRU-HOLE	1.05	0.8034
CON1	2x CONN HEADER .100" DUAL STR 20POS	PRP0010DAAN-RC	MALE 20 POS - 0.1 INCH - 2 ROWS	PRP0010DAAN-RC	THRU-HOLE	1.18	0.59184
R6	RES SMD 120 OHM 1% 1/10W 0603	120	0603	RC0603FR-07120RL	SMT	0.1	0.00268
R5	RES SMD 10K OHM 1% 1/10W 0603	10k	0603	RC0603FR-0710KL	SMT	0.1	0.00268
C1	CAP ALUM 100UF 20% 35V RADIAL	100u	Diam: 6.3mm	UWT1H101MNL1GS	SMT	0.47	0.2201
C5	CAP CER 22PF 50V 00G/NP0 0603	22p	0603	06035A220JAT2A	SMT	0.1	0.00765
C16	CAP CER 4.7UF 50V X7R 1206	4.7u	1206	UMK316AB7475KL-T	SMT	0.24	0.05832
C15	CAP CER 10UF 50V X7R 1206	10u	1206	CL31B106KBHNNNE	SMT	0.31	0.08261
C2, C7, C8, C12, C14	CAP CER 10UF 35V X5R 0805	10u	0805	GRM21B6YA106KE43L	SMT	0.38	0.9976
SENSOR, U2(1)	CONN HEADER FEMALE 9POS .1" GOLD	PPPC091LFBN-RC	9 Pos Pitch: 0.1"	PPPC091LFBN-RC	THRU-HOLE	0.83	0.3712
U2(2)	CONN HEADER FEMALE 5POS .1" GOLD	PPPC051LFBN-RC	5 Pos Pitch: 0.1"	PPPC051LFBN-RC	THRU-HOLE	0.65	0.272
PCB		PCB 1/16", FR4, IMMERSION GOLD (ENIG), 1 oz copper, GREEN MASK			-	10.99	6.87
LCD	Display LCD TFT ILI9341	ILI9341	-	-	FPC	10.6	7.35
U3	Evaluation board TM4C123	TM4C123	-	-	THRU-HOLE	13.49	13.49
U4	Gesture Sensor ZX	-	-	-	THRU-HOLE	24.95	24.95
					TOTAL	71.02	59.53332