Pong

CSEE4824 Final Project

Charles Hastings
Rachid Jeitani
May 13, 2011

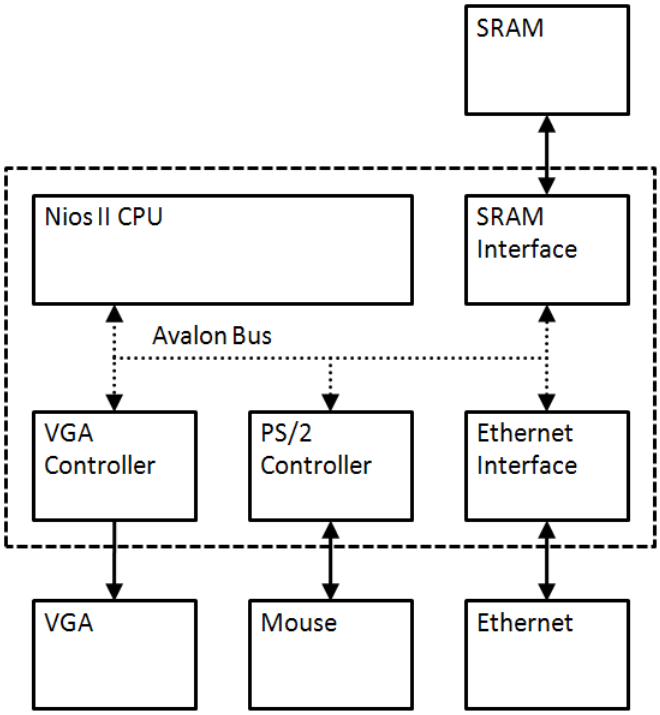
Introduction

Our project is a networked multiplayer version of the 1970s classic Atari video game Pong. Players sit at two separate stations, each consisting of a DE2 board, VGA monitor, and keyboard. The stations connect to each other, a master is chosen, and the game begins. “Exciting and fast-paced action” ensues.

You may have heard of Pong. It is an entertaining game that involves two players, each with a paddle that hits a ball back and forth around the screen. A player wins a point when s/he manages to put the ball beyond the opponent’s paddle.

The game keeps track of the position of the paddles and the game ball. Players move their paddles using a PS/2 mouse. If a ball hits a paddle, it bounces back to the other side of the screen at an angle equal to that at which it hit. The game continues until one player has scored 10 points. As time progresses, ball velocity increases to make the game more challenging.

Top-Level



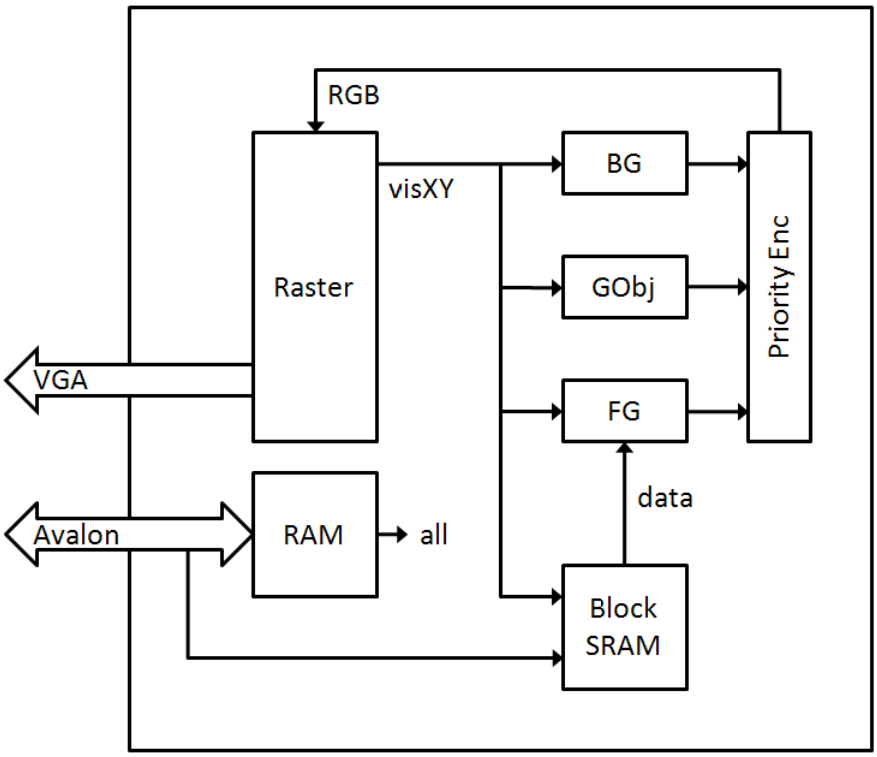
The system is based on the Altera Nios II CPU and implemented as a series of peripherals, connected using Altera’s proprietary Avalon bus and built with the assistance of the SOPC Builder tool.

The key peripherals in our design are:

1. VGA. Handles generation of foreground, background, and game objects
2. Mouse. User interface for system. Used to select player number at initialization and to control paddle movement during gameplay
3. Ethernet. Provides communication between consoles to enable multiplayer gaming

Implementation

VGA

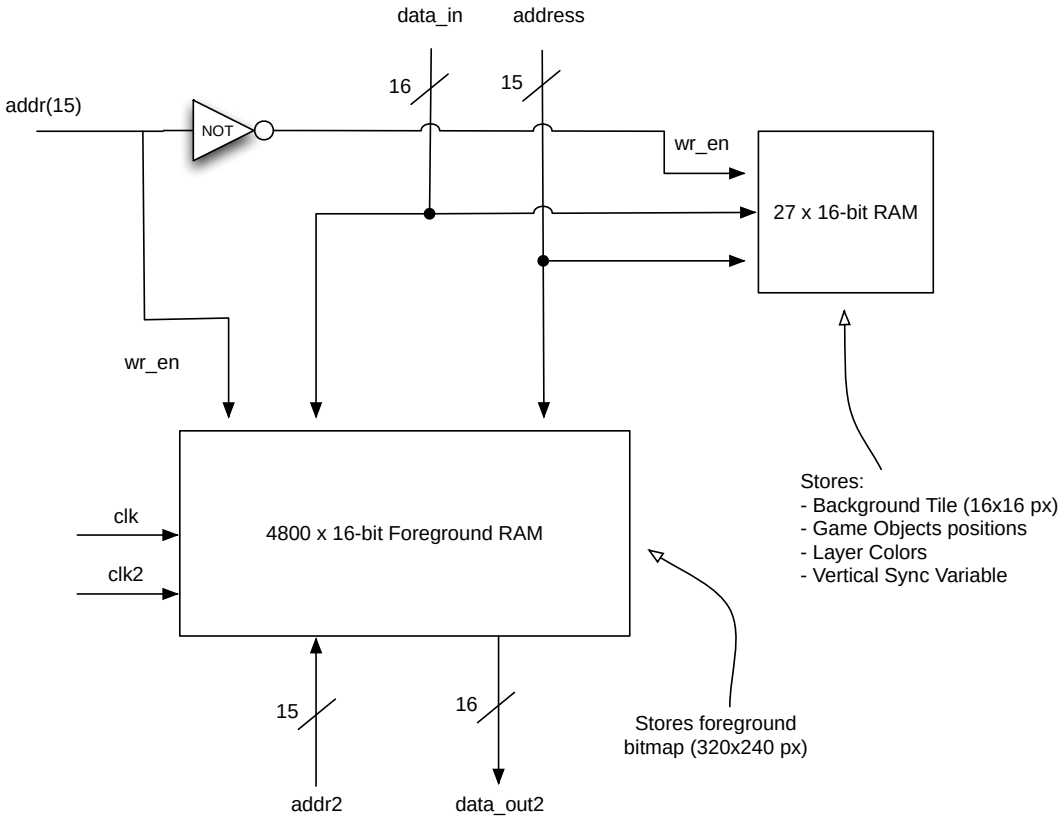


Our VGA peripheral consists of several key components, including two different memories, pixel generators for each of the three main graphic layers and a priority encoder to enforce the layering. Firstly, the peripheral utilizes two different memories. The first consists of 27 elements each 16-bits wide. It is used to store the 16x16 bit background tile as well as the locations of the game objects and the colors of the various layers. The second memory is used exclusively to store the foreground bitmap. Initially, our design intended to have 19,200 16-bit elements to be able to store the entire contents of the 640x480 display. However, repeated efforts to be able to fit such a memory on-chip were not successful and it was decided that reducing the size of the memory and only store a subset of the display was preferable. Therefore, we store the foreground into a 4,800-word memory, enough to store 320x240 pixels (1/4 of the screen). This bitmap is then stretched to cover the entire display, albeit with less detail. This foreground RAM is dual-ported, with a read/write interface on one side, and a read-only interface on the other. The Avalon bus is connected to the read/write side, allowing our software to be able to both read and write to the foreground RAM. To the other side we connected the foreground generator, which would only require read-access.

Since both of these memories were stored in the same peripheral (meaning they have the same base address) we needed a way to fully address both the smaller RAM and the foreground RAM independently. To solve this problem, we used the most significant bit of the address signal as a

selector. In order to address the smaller RAM, we would address the VGA peripheral normally (with the high bit equal to 0). When addressing the foreground RAM, however, we flipped the high bit to 1. The peripheral would then, depending on the value of this bit, interface with the correct RAM. This method worked because our address space didn't require the full 16-bits of the address signal.

Once all the data was in the correct location (background bitmap, colors, foreground bitmap), the three pixel generators would use this data as well as information regarding the current scan position from the VGA raster module to generate an enable signal. For example, the background generator would, by comparing the current scan x and y positions to the 16x16 bit background tile, would generate a 1 output if VGA raster is currently scanning a position that is present as a 1 (on) bit in the bitmap. All three pixel-generators are constantly doing this. Their collective enable signals are inputs to a priority encoder. This module will 'layer' the enable signals by outputting the color of the layer that should be visible at that particular pixel. Therefore, if the background and foreground are both enabled, the foreground takes priority and the colors corresponding to the foreground are output to the VGA raster to be drawn on the screen.



VGA Memory Map

Base Offset	Width	Description
0x00–0x30	16	Background Tile Bitmap (16x16 pixels)
0x32	10	Ball x position
0x34	10	Ball y position
0x36	10	Paddle 1 (left) start
0x38	10	Paddle 1 (left) end
0x40	10	Paddle 2 (right) start
0x42	10	Paddle 2 (right) end
0x44	1	Vertical Sync
0x46	16	Screen Color (Layer behind Background)
0x48	16	Background Tile Color
0x50	16	Foreground color
0x52	16	Game Object Color
0x10000–0x112c0	16	320x240 pixel foreground bitmap

We used the X-Bitmap (XBM) image format for our Pong logo and win/loss banners. This image format is in the form of a character array of hex values corresponding to the whether a particular bit is 1 or 0. This made it very easy to display these images in the foreground since it was just a matter of reading the bytes in the correct order and storing them into the foreground RAM. The VGA Controller would then just draw them to the screen.

Mouse

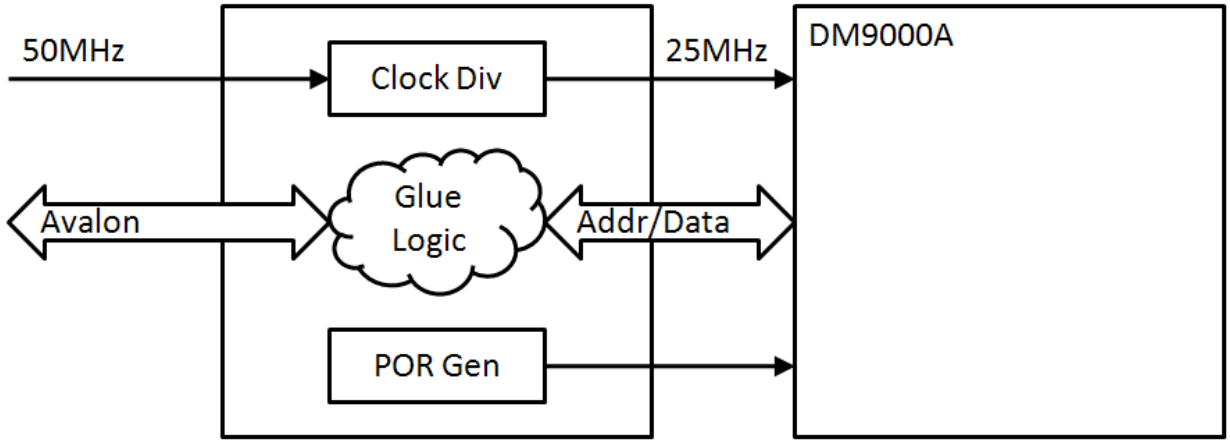
The PS/2 mouse peripheral is a standard Altera-provided IP core. It contains a 256-byte read FIFO and is accessed through a simple memory map.

Mouse Memory Map

Base Offset	Width	Description
0x00	32	Data
0x04	32	Control

One peculiarity of this interface is that there is no direct way to check the amount of data in the FIFO. The only way to obtain this information is by removing a token from the FIFO. This makes interfacing the PS/2 mouse inconvenient, as data is sent from the mouse in three-byte packets and an arbitrary read to the FIFO (which signals an interrupt when in any non-empty state) may only return a partial packet. Ideally the ISR would first check to see if a full packet has been received and then take action.

Ethernet



The Ethernet peripheral is a straightforward bridge between the Avalon bus and the DM9000A's interface, with a few notable peculiarities:

1. The address/data bus is not used in a conventional way to access DM9000A configuration registers. The address space is only two words, in fact. The first accesses the configuration register address, the second accesses the register referenced in the first
2. The Altera DE2 board does not provide a crystal oscillator for the controller. The signal must be generated by the FPGA. Fortunately, it is an even multiple of the clock frequency so this is not difficult to implement once its necessity is discovered
3. A power-on reset controller is important for repeatable performance. The DM9000A RST_N line does not map sensibly to an Avalon control signal so it was initially tied high to keep the device out of reset. This led to erratic functionality and eventually a power-on reset delay counter was implemented to keep the device in reset until shortly after system initialization (and, presumably, power supplies have stabilized)
4. The SOPC configuration for this peripheral is non-standard and very important to set correctly. The interface is unclocked, with 20ns setup/hold/read/write latency times specified

The DM9000A (and thus the Ethernet peripheral) memory map is well-documented in its datasheet.

Design Challenges

Here is a list of some of the significant challenges faced in implementing this design. The list is presented in no particular order, with a summary of solution presented to aide future development.

1. Ethernet initialization failures. Fixed by adding power-on reset controller
2. Ethernet controller unresponsive on bus. Fixed by adding clock
3. Corrupt PS/2 data. Improved by requesting data packets (in polling mode) or using ISR

4. Slight corruption of displayed foreground image. Fixed by clocking FG RAM at same rate as pixel generator
5. Hub dropping packets. Solved by directly connecting boards
6. Insufficient on-chip memory for full VGA resolution bitmap. Scaled bitmap to 320x240, doubled pixel size
7. Performance degradation with moderate network traffic. The video output became choppy at 30 packet/second load. Solved by revising specification to share less data over network

Lessons Learned

When dealing with hardware design, it is best to spend a significantly longer period of time before attempting to write any code, to plan out every detail of the design. This makes it easier writing the code later on and it also makes it easier to debug when problems inevitably arise. When you don't fully understand what a particular part of your system is doing, you end up resorting to trial and error to fix things, which will end up sucking up a lot of time.

The tools that you use have a huge impact on your success. Becoming familiar with the nuances of Nios, Quartus, and SOPC Builder helps you quickly solve the many problems you will encounter with these tools. Also, using a version control system like Subversion is highly recommended since it allows you to take risks knowing you can always go back. It also takes away the chore of having to email source code back and forth between teammates.

Division of Labor

Both members took an active role in the development and troubleshooting of both hardware and software. Rachid led the creation of hardware, focusing on the VGA raster module hierarchy, and also wrote the software to interface with it. Charles primarily developed the Nios application software and contributed to some of the other hardware peripherals.

Source Code Listing

The following pages contain source code listings for all files created or modified to build the Pong game system.

main.c

```
#include <stdio.h>
#include <system.h>
#include <stdlib.h>
#include <alt_types.h>
#include <sys/alt_irq.h>

#include "xio.h"

#include "mouse.h"
#include "game.h"
#include "packet.h"
#include "bg.h"
#include "dm9000a.h"
#include "graphics.h"


void init()
{
    mouse_start();
    printf( "Mouse init done.\n" );

    dm9000a_start();
    printf( "Ethernet init done.\n" );


    gs.pn = ui_getplayer();
//    ui_waitopponent();

    // clear the logos
    clr_fg();
    // draw vertical line
    draw_centerline();
    ui_updatescore();


    txpkt.enet.saddr[5] = gs.pn;

#ifdef PACKET_L3L4
    txpkt.ip.saddr = HTONL( IP( 192, 168, 0, pn ) );
#endif
    if (gs.pn==1)
    {
        gs.p2.c = 0;
        gs.p2.l = 0;
    }
    else
    {
        gs.p1.c = 0;
        gs.p1.l = 0;
    }
}


static inline void game()
```

```
{
    int vx, vy, y, b;

    // DURING ACTIVE SCAN --------------------------------
    while (IORD16(VGA_SYNC) == 0 )
        ;

    update_state();

    if ( mouse_event( &vx, &vy, &b ) )
    {
        y = gs_pl()->c;

        // mouse up is -y direction in paddle-space
        y += -1*vy;

        if ( y < (gs_pl()->l/2) )
            y = gs_pl()->l/2;
        else if ( y>(SCREEN_Y-gs_pl()->l/2) )
            y = (SCREEN_Y-gs_pl()->l/2);

        gs_pl()->c = y;
    }

    // when not to display the ball: when not on our screen, and on other
serve
    if ( gs.ball.side!=gs.pn || (gs.sn>0 && gs.sn!=gs.pn) )
    {
        gs.ball.x = 10;
        gs.ball.y = 10;
    }

    // local player has ball to serve/start
    if (gs.sn == gs.pn)
    {
        ui_updatescore();

        gs.ball.y = gs_pl()->c;

        gs.ball.x = (gs.pn==1) ? 60 : SCREEN_X-60;

        gs.ball.vx = 0;
        gs.ball.vy = 0;

        // button press, local player has serve
        if (b & MOUSE_LB)
        {
            gs.ball.vx = get_player()==1 ? -BALL_SPEED : BALL_SPEED;
            gs.sn = 0;
        }

    }


    // DURING SYNC INTERVAL ----------------------------
    while (IORD16(VGA_SYNC) != 0 )
        ;
```

```
    show_state();

}



int main()
{
    printf("NIOS boot successful.\n");

    IOWR16( VGA_FG_COLOR, 0x0fa5);
    IOWR16( VGA_SC_COLOR, 0x80ff);
    IOWR16( VGA_GO_COLOR, 0xffff);

    init();

    while (1)
    {
        game();
//        import_state();
//        transmit_state();
//        mouse_poll();

    }
}
```

banner.c

```
#include <io.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include "system.h"
#include "sys/alt_irq.h"
#include "vga.h"
#include "xio.h"
#include "banner.h"

#define pong_width 320
#define pong_height 100

static unsigned char pong_bits[] = {
   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, 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, 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, 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,
   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, 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, 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, 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,
   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, 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, 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, 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,
   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, 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, 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, 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,
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, 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, 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, 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,
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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 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, 0x80,
0xff, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff,
0xff, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0xff, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xf0, 0xff, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0,
0xff, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xfc, 0xff, 0xff, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc,
0xff, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0xff, 0x00, 0x00,
0x00, 0x00, 0x00, 0xf8, 0xff, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0x00, 0x00,
0x00, 0x00, 0xc0, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff,
0xff, 0xff, 0x01, 0x00, 0x00, 0x00, 0x80, 0xff, 0xff, 0x3f, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff,
0xff, 0xff, 0x03, 0x00, 0x00, 0x00, 0xf8, 0xff, 0xff, 0xff, 0x03, 0x00,
0x00, 0x00, 0xc0, 0xff, 0xff, 0xff, 0x07, 0x00, 0x00, 0x00, 0xc0, 0xff,
0xff, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0x07, 0x00, 0x00, 0x00, 0xfc, 0xff,
0xff, 0xff, 0x0f, 0x00, 0x00, 0x00, 0xe0, 0xff, 0xff, 0xff, 0x07, 0x00,
0x00, 0x00, 0xf0, 0xff, 0xff, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0x07, 0x00,
0x00, 0x00, 0xfe, 0xff, 0xff, 0xff, 0x3f, 0x00, 0x00, 0x00, 0xe0, 0xff,
0xff, 0xff, 0x1f, 0x00, 0x00, 0x00, 0xfc, 0xff, 0xff, 0x7f, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff,
0xff, 0xff, 0x1f, 0x00, 0x00, 0x80, 0xff, 0xff, 0xff, 0xff, 0x7f, 0x00,
0x00, 0x00, 0xf8, 0xff, 0xff, 0xff, 0x3f, 0x00, 0x00, 0x00, 0xfe, 0xff,
0xff, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0x1f, 0x00, 0x00, 0xe0, 0xff, 0xff,
0xff, 0xff, 0xff, 0x01, 0x00, 0x00, 0xf8, 0xff, 0xff, 0xff, 0x7f, 0x00,
0x00, 0x00, 0xff, 0xff, 0xff, 0x7f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0xff, 0xff, 0x1f, 0x00,
0x00, 0xf0, 0xff, 0xff, 0xff, 0xff, 0xff, 0x03, 0x00, 0x00, 0xfc, 0xff,
0xe7, 0xff, 0x7f, 0x00, 0x00, 0xc0, 0xff, 0xff, 0xff, 0x7f, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x7f,
0x00, 0xff, 0x7f, 0x00, 0x00, 0xf8, 0xff, 0xff, 0xff, 0xff, 0xff, 0x07,
0x00, 0x00, 0xfe, 0xff, 0x00, 0xfc, 0xff, 0x00, 0x00, 0xc0, 0xff, 0xff,
0x03, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xfc, 0x3f, 0x00, 0xfc, 0x7f, 0x00, 0x00, 0xfc, 0xff, 0xff,
0xff, 0xff, 0xff, 0x0f, 0x00, 0x00, 0xfe, 0x3f, 0x00, 0xf0, 0xff, 0x01,
0x00, 0xf0, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x1f, 0x00, 0xf8, 0x7f, 0x00,
0x00, 0xfe, 0xff, 0x0f, 0x00, 0xfe, 0xff, 0x1f, 0x00, 0x00, 0xff, 0x3f,
0x00, 0xf0, 0xff, 0x01, 0x00, 0xf8, 0xff, 0x0f, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f,
0x00, 0xf0, 0xff, 0x01, 0x00, 0xff, 0xff, 0x03, 0x00, 0xf8, 0xff, 0x3f,
0x00, 0x00, 0xff, 0x0f, 0x00, 0xe0, 0xff, 0x01, 0x00, 0xfc, 0xff, 0x07,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xfc, 0x0f, 0x00, 0xe0, 0xff, 0x01, 0x80, 0xff, 0xff, 0x00,
0x00, 0xc0, 0xff, 0x7f, 0x00, 0x00, 0xff, 0x07, 0x00, 0xc0, 0xff, 0x03,
0x00, 0xfe, 0xff, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f, 0x00, 0xe0, 0xff, 0x01,
0x80, 0xff, 0x3f, 0x00, 0x00, 0x80, 0xff, 0x7f, 0x00, 0x00, 0xff, 0x07,
0x00, 0xc0, 0xff, 0x03, 0x00, 0xfe, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f,
0x00, 0xc0, 0xff, 0x01, 0xe0, 0xff, 0x1f, 0x00, 0x00, 0x00, 0xff, 0xff,
0x00, 0x80, 0xff, 0x07, 0x00, 0x80, 0xff, 0x03, 0x00, 0xff, 0x7f, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xfc, 0x0f, 0x00, 0xc0, 0xff, 0x01, 0xe0, 0xff, 0x07, 0x00,
0x00, 0x00, 0xfc, 0xff, 0x01, 0x80, 0xff, 0x03, 0x00, 0x00, 0xff, 0x03,
0x00, 0xff, 0x1f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f, 0x00, 0xc0, 0xff, 0x01,
0xe0, 0xff, 0x07, 0x00, 0x00, 0x00, 0xf8, 0xff, 0x01, 0x80, 0xff, 0x03,
0x00, 0x00, 0xff, 0x03, 0x80, 0xff, 0x1f, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f,
0x00, 0xc0, 0xff, 0x01, 0xf0, 0xff, 0x01, 0x00, 0x00, 0x00, 0xf0, 0xff,
0x03, 0x80, 0xff, 0x03, 0x00, 0x00, 0xff, 0x07, 0x80, 0xff, 0x0f, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xfc, 0x0f, 0x00, 0xc0, 0xff, 0x01, 0xf0, 0xff, 0x01, 0x00,
0x00, 0x00, 0xe0, 0xff, 0x07, 0x80, 0xff, 0x03, 0x00, 0x00, 0xff, 0x07,
0xc0, 0xff, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f, 0x00, 0xc0, 0xff, 0x01,
0xf0, 0xff, 0x00, 0x00, 0x00, 0x00, 0xe0, 0xff, 0x07, 0x80, 0xff, 0x03,
0x00, 0x00, 0xff, 0x07, 0xe0, 0xff, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f,
0x00, 0xc0, 0xff, 0x01, 0xf8, 0x7f, 0x00, 0x00, 0x00, 0x00, 0xc0, 0xff,
0x07, 0x80, 0xff, 0x03, 0x00, 0x00, 0xff, 0x07, 0xe0, 0xff, 0x03, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xfc, 0x0f, 0x00, 0xc0, 0xff, 0x01, 0xfc, 0x3f, 0x00, 0x00,
0x00, 0x00, 0x80, 0xff, 0x07, 0x80, 0xff, 0x03, 0x00, 0x00, 0xff, 0x07,
0xe0, 0xff, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x0f, 0x00, 0xe0, 0xff, 0x01,
0xfc, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x80, 0xff, 0x0f, 0x80, 0xff, 0x03,
0x00, 0x00, 0xff, 0x07, 0xe0, 0xff, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0x1f,
```

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

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

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

```
   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, 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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00 };


#define winner_width 320
#define winner_height 100

static unsigned char winner_bits[] = {
   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, 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, 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, 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,
   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, 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, 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, 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,
   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, 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, 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, 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,
   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, 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, 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, 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,
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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0x7f,
0x00, 0x00, 0xe0, 0xff, 0x01, 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, 0x7f, 0x00, 0x00, 0xe0, 0xff, 0x01, 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, 0x7f, 0x00, 0x00, 0xe0, 0xff,
0x01, 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, 0x1c, 0x70,
0x00, 0x00, 0xe0, 0x80, 0x03, 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, 0x18, 0x70, 0x00, 0x00, 0xe0, 0x80, 0x03, 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, 0x1c, 0x70, 0x00, 0x00, 0xe0, 0x80,
0x03, 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, 0x18, 0xf0,
0xff, 0xff, 0xff, 0x80, 0x03, 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, 0x1c, 0xf0, 0xff, 0xff, 0xff, 0x80, 0x03, 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, 0x18, 0xf0, 0xff, 0xff, 0xff, 0x80,
0x03, 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, 0x1c, 0xf0,
0xff, 0xff, 0x7f, 0x80, 0x01, 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, 0x18, 0xf0, 0xff, 0xff, 0x7f, 0x80, 0x01, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x3e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0xf0, 0xff, 0xff, 0x7f, 0xc0,
0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3e, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0xe0,
0xff, 0xff, 0x7f, 0xc0, 0x01, 0xe0, 0x03, 0x00, 0xfc, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x00, 0x00, 0x1f,
```

```
0x00, 0x3e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x38, 0xe0, 0xff, 0xff, 0x7f, 0xc0, 0x01, 0xc0, 0x07, 0x00,
0x7c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x1f, 0x00, 0x00, 0x1f, 0x00, 0x3e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0xe0, 0xff, 0xff, 0x7f, 0xc0,
0x01, 0xc0, 0x07, 0x00, 0x3e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x1f, 0x00, 0x00, 0x1f, 0x00, 0x3e, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0xe0,
0xff, 0xff, 0x7f, 0xc0, 0x01, 0x80, 0x0f, 0x00, 0x3e, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1e, 0x00, 0x00, 0x0f,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x30, 0xe0, 0xff, 0xff, 0x7f, 0xe0, 0x00, 0x80, 0x0f, 0x00,
0x1f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x1e, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0xe0, 0xff, 0xff, 0x7f, 0xe0,
0x00, 0x00, 0x1f, 0x00, 0x1f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x1e, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x70, 0xe0,
0xff, 0xff, 0x7f, 0xe0, 0x00, 0x00, 0x1e, 0x80, 0x0f, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1e, 0x00, 0x00, 0x0f,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0xf0, 0xe0, 0xff, 0xff, 0x7f, 0x70, 0x00, 0x00, 0x3e, 0x80,
0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x3e, 0x00, 0x80, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0xe0, 0xff, 0xff, 0x3f, 0x70,
0x00, 0x00, 0x3c, 0xc0, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x3e, 0x00, 0x80, 0x0f, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0xc0,
0xff, 0xff, 0x3f, 0x78, 0x00, 0x00, 0x7c, 0xc0, 0x03, 0x00, 0xfc, 0x0f,
0x00, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x3e, 0x00, 0x80, 0x07,
0xf8, 0x3f, 0x00, 0x80, 0x8f, 0x7f, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0xc0, 0xc1, 0xff, 0xff, 0x3f, 0x38, 0x00, 0x00, 0xf8, 0xe0,
0x03, 0x00, 0xfe, 0x3f, 0x00, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00,
0x3c, 0xf0, 0x81, 0x07, 0xf8, 0x3f, 0x00, 0x80, 0xef, 0xff, 0x00, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0xc1, 0xff, 0xff, 0x3f, 0x3c,
0x00, 0x00, 0xf8, 0xf0, 0x01, 0x80, 0xff, 0x7f, 0x00, 0xf8, 0x00, 0x7c,
0x00, 0x00, 0x00, 0x00, 0x3c, 0xf0, 0x81, 0x07, 0xf8, 0x3f, 0x00, 0x80,
0xef, 0xff, 0x01, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xc3,
0xff, 0xff, 0x3f, 0x1c, 0x00, 0x00, 0xf0, 0xf1, 0x01, 0x80, 0x1f, 0xfe,
0x00, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x3c, 0xf8, 0x81, 0x07,
0xf8, 0x3f, 0x00, 0x80, 0xff, 0xf8, 0x03, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x80, 0xc3, 0xff, 0xff, 0x1f, 0x1e, 0x00, 0x00, 0xe0, 0xf9,
0x00, 0xc0, 0x07, 0xf8, 0x01, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00,
0x3c, 0xf8, 0x83, 0x07, 0x00, 0x3e, 0x00, 0x80, 0x3f, 0xe0, 0x03, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x87, 0xff, 0xff, 0x1f, 0x0e,
0x00, 0x00, 0xe0, 0x7b, 0x00, 0xe0, 0x03, 0xf0, 0x01, 0xf8, 0x00, 0x7c,
0x00, 0x00, 0x00, 0x00, 0x3c, 0xf8, 0x83, 0x07, 0x00, 0x3e, 0x00, 0x80,
0x1f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x8f,
0xff, 0xff, 0x1f, 0x0f, 0x00, 0x00, 0xc0, 0x7f, 0x00, 0xe0, 0x03, 0xe0,
0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x3c, 0xf8, 0xc3, 0x07,
0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x8e, 0xff, 0xff, 0x1f, 0x07, 0x00, 0x00, 0xc0, 0x3f,
0x00, 0xe0, 0x01, 0xe0, 0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00,
0x7c, 0xb8, 0xc3, 0x03, 0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1e, 0xff, 0xff, 0x8f, 0x03,
0x00, 0x00, 0x80, 0x3f, 0x00, 0xf0, 0x01, 0xe0, 0x03, 0xf8, 0x00, 0x7c,
0x00, 0x00, 0x00, 0x00, 0x78, 0xbc, 0xc7, 0x03, 0x00, 0x3e, 0x00, 0x80,
```

```
0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1c,
0xff, 0xff, 0xcf, 0x03, 0x00, 0x00, 0x00, 0x1f, 0x00, 0xf0, 0x01, 0xc0,
0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x78, 0xbc, 0xc7, 0x03,
0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x38, 0xff, 0xff, 0xcf, 0x01, 0x00, 0x00, 0x00, 0x1f,
0x00, 0xf0, 0x01, 0xc0, 0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00,
0x78, 0x1c, 0xc7, 0x03, 0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0xfe, 0xff, 0xe7, 0x01,
0x00, 0x00, 0x00, 0x1f, 0x00, 0xf0, 0x01, 0xc0, 0x03, 0xf8, 0x00, 0x7c,
0x00, 0x00, 0x00, 0x00, 0x78, 0x1c, 0xc7, 0x03, 0x00, 0x3e, 0x00, 0x80,
0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x70,
0xfe, 0xff, 0xf7, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x00, 0xf0, 0x01, 0xc0,
0x07, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x78, 0x1e, 0xc7, 0x03,
0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xe0, 0xfe, 0xff, 0x77, 0x00, 0x00, 0x00, 0x00, 0x1f,
0x00, 0xf0, 0x01, 0xc0, 0x07, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00,
0x78, 0x1e, 0xcf, 0x03, 0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0xfd, 0xff, 0x3b, 0x00,
0x00, 0x00, 0x00, 0x1f, 0x00, 0xf0, 0x01, 0xc0, 0x03, 0xf8, 0x00, 0x7c,
0x00, 0x00, 0x00, 0x00, 0x70, 0x0e, 0xce, 0x01, 0x00, 0x3e, 0x00, 0x80,
0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0,
0xff, 0xff, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x00, 0xf0, 0x01, 0xc0,
0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00, 0x70, 0x0e, 0xee, 0x01,
0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x80, 0xfb, 0xff, 0x1d, 0x00, 0x00, 0x00, 0x00, 0x1f,
0x00, 0xf0, 0x01, 0xc0, 0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00,
0xf0, 0x0e, 0xee, 0x01, 0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0x0f, 0x00,
0x00, 0x00, 0x00, 0x1f, 0x00, 0xf0, 0x01, 0xe0, 0x03, 0xf8, 0x00, 0x7c,
0x00, 0x00, 0x00, 0x00, 0xf0, 0x0f, 0xee, 0x01, 0x00, 0x3e, 0x00, 0x80,
0x0f, 0xc0, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xfe, 0xff, 0x07, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x00, 0xe0, 0x01, 0xe0,
0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00, 0xf0, 0x07, 0xfe, 0x01,
0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xfe, 0xff, 0x03, 0x00, 0x00, 0x00, 0x00, 0x1f,
0x00, 0xe0, 0x03, 0xe0, 0x03, 0xf8, 0x00, 0x7c, 0x00, 0x00, 0x00, 0x00,
0xf0, 0x07, 0xfc, 0x01, 0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x03, 0x00,
0x00, 0x00, 0x00, 0x1f, 0x00, 0xe0, 0x03, 0xf0, 0x01, 0xf8, 0x00, 0x7e,
0x00, 0x00, 0x00, 0x00, 0xf0, 0x07, 0xfc, 0x00, 0x00, 0x3e, 0x00, 0x80,
0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xf8, 0xff, 0x01, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x00, 0xc0, 0x07, 0xf8,
0x01, 0xf0, 0x01, 0x7f, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x07, 0xfc, 0x00,
0x00, 0x3e, 0x00, 0x80, 0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xf0, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f,
0x00, 0x80, 0x1f, 0xfe, 0x00, 0xf0, 0xc7, 0x7f, 0x00, 0x00, 0x00, 0x00,
0xe0, 0x07, 0xfc, 0x00, 0xfe, 0xff, 0x3f, 0x80, 0x0f, 0xc0, 0x07, 0x00,
0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x7f, 0x00, 0x00,
0x00, 0x00, 0x00, 0x1f, 0x00, 0x80, 0xff, 0x7f, 0x00, 0xe0, 0xff, 0x7d,
0x00, 0x00, 0x00, 0x00, 0xe0, 0x03, 0xf8, 0x00, 0xfe, 0xff, 0x3f, 0x80,
0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0xc0, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x00, 0x00, 0xff, 0x3f,
0x00, 0xc0, 0xff, 0x7d, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x03, 0xf8, 0x00,
0xfe, 0xff, 0x3f, 0x80, 0x0f, 0xc0, 0x07, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x80, 0x1f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f,
0x00, 0x00, 0xfc, 0x0f, 0x00, 0x80, 0x7f, 0x7c, 0x00, 0x00, 0x00, 0x00,
0xe0, 0x03, 0xf8, 0x00, 0xfe, 0xff, 0x3f, 0x80, 0x0f, 0xc0, 0x07, 0x00,
```

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


#define loser_width 320
#define loser_height 100

static unsigned char loser_bits[] = {
   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, 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, 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, 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,
   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, 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, 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, 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,
   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, 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, 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, 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,
   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, 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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xfc, 0x1f, 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, 0xc0, 0xff, 0xff, 0x03, 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, 0x1f,
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,
0x7e, 0x00, 0x00, 0x7e, 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, 0x80, 0x0f, 0x00, 0x00, 0xf0, 0x01, 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, 0x03, 0x00, 0x00, 0xc0,
0x07, 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,
0x00, 0x00, 0x00, 0x00, 0x0f, 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, 0x38, 0x00, 0x00, 0x00, 0x00, 0x1c, 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, 0x1e, 0x00, 0x00, 0x00, 0x00,
0x78, 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, 0x0e,
0x00, 0x00, 0x00, 0x00, 0xe0, 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, 0x80, 0x03, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x01, 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, 0x80, 0x01, 0x00, 0x00, 0x00, 0x00,
0x80, 0x03, 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, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0e, 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, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0e, 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, 0x38, 0x00, 0x00,
0x70, 0x00, 0x80, 0x01, 0x00, 0x1c, 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, 0x1c, 0x00, 0xf0, 0x00, 0xc0, 0x03, 0x00, 0x38, 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, 0x0c, 0x00, 0xf8, 0x00, 0xe0, 0x07,
0x00, 0x30, 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, 0x0e, 0x00,
0xfc, 0x01, 0xe0, 0x07, 0x00, 0x70, 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, 0x06, 0x00, 0xfc, 0x01, 0xe0, 0x07, 0x00, 0x60, 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, 0x03, 0x00, 0xfc, 0x00, 0xe0, 0x07,
0x00, 0xc0, 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, 0x03, 0x00,
0xf8, 0x00, 0xc0, 0x07, 0x00, 0xc0, 0x00, 0x80, 0x07, 0x00, 0x1f, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00,
0x00, 0x80, 0x03, 0x00, 0xf0, 0x00, 0xc0, 0x03, 0x00, 0xc0, 0x01, 0x00,
0x0f, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0f, 0x00, 0x00, 0x00, 0x80, 0x01, 0x00, 0x20, 0x00, 0x00, 0x01,
0x00, 0x80, 0x01, 0x00, 0x0f, 0x80, 0x07, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x03, 0x00, 0x1e, 0x80, 0x07, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00,
0x00, 0xc0, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x00,
0x1e, 0xc0, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0f, 0x00, 0x00, 0x00, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x03, 0x00, 0x3c, 0xc0, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0xc0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x00, 0x38, 0xe0, 0x01, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00,
0x00, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x00,
0x78, 0xe0, 0x01, 0x80, 0x1f, 0x00, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x80, 0x1f, 0x00, 0xc0, 0x3f, 0x00, 0x80, 0x1f, 0x00,
0x00, 0x0f, 0x00, 0x00, 0x00, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x06, 0x00, 0xf0, 0xf0, 0x00, 0xe0, 0x7f, 0x00, 0x3c, 0x80,
0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0xe0, 0x7f, 0x00, 0xf0, 0xff,
0x01, 0xe0, 0x7f, 0x00, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0xf0, 0x70, 0x00, 0xf0,
0xff, 0x00, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0xf0,
0xff, 0x00, 0xf8, 0xff, 0x01, 0xf8, 0xff, 0x00, 0x00, 0x0f, 0x00, 0x00,
0x00, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00,
0xe0, 0x79, 0x00, 0xf8, 0xf0, 0x01, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0xf8, 0xf0, 0x01, 0x78, 0xc0, 0x01, 0x7c, 0xf0, 0x01,
0x00, 0x0f, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x06, 0x00, 0xe0, 0x39, 0x00, 0x7c, 0xe0, 0x03, 0x3c, 0x80,
0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x7c, 0xe0, 0x03, 0x3c, 0x00,
0x00, 0x3c, 0xc0, 0x03, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0xc0, 0x3f, 0x00, 0x3c,
```

```
0xc0, 0x03, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x3c,
0xc0, 0x03, 0x3c, 0x00, 0x00, 0x1e, 0x80, 0x03, 0x00, 0x0f, 0x00, 0x00,
0x00, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00,
0x80, 0x1f, 0x00, 0x1c, 0x80, 0x03, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x1c, 0x80, 0x03, 0x3c, 0x00, 0x00, 0x1e, 0x80, 0x07,
0x00, 0x0f, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x06, 0x00, 0x80, 0x1f, 0x00, 0x1e, 0x80, 0x07, 0x3c, 0x80,
0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x1e, 0x80, 0x07, 0x3c, 0x00,
0x00, 0x0f, 0x80, 0x07, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x0f, 0x00, 0x1e,
0x80, 0x07, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x1e,
0x80, 0x07, 0x7c, 0x00, 0x00, 0x0f, 0x80, 0x07, 0x00, 0x0f, 0x00, 0x00,
0x00, 0x60, 0x00, 0x00, 0x00, 0x40, 0x05, 0x00, 0x00, 0x00, 0x06, 0x00,
0x00, 0x0f, 0x00, 0x1e, 0x80, 0x07, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x1e, 0x80, 0x07, 0xf8, 0x03, 0x00, 0xff, 0xff, 0x07,
0x00, 0x0f, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00, 0x80, 0xff, 0xff, 0x00,
0x00, 0x00, 0x06, 0x00, 0x00, 0x0f, 0x00, 0x1e, 0x80, 0x07, 0x3c, 0x80,
0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x1e, 0x80, 0x07, 0xf0, 0x3f,
0x00, 0xff, 0xff, 0x07, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x60, 0x00, 0x00,
0xe0, 0xff, 0xff, 0x0f, 0x00, 0x00, 0x06, 0x00, 0x00, 0x0f, 0x00, 0x1e,
0x80, 0x07, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x1e,
0x80, 0x07, 0xe0, 0xff, 0x00, 0xff, 0xff, 0x07, 0x00, 0x0f, 0x00, 0x00,
0x00, 0xe0, 0x00, 0x00, 0xfe, 0x17, 0xd0, 0x7f, 0x00, 0x00, 0x07, 0x00,
0x00, 0x0f, 0x00, 0x1e, 0x80, 0x07, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x1e, 0x80, 0x07, 0x00, 0xfe, 0x01, 0x0f, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x00, 0x00, 0x7f, 0x00, 0x00, 0xfe,
0x01, 0x00, 0x07, 0x00, 0x00, 0x0f, 0x00, 0x1e, 0x80, 0x07, 0x3c, 0x80,
0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x1e, 0x80, 0x07, 0x00, 0xf0,
0x01, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x00, 0xc0,
0x0f, 0x00, 0x00, 0xf0, 0x03, 0x00, 0x03, 0x00, 0x00, 0x0f, 0x00, 0x1e,
0x80, 0x07, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x1e,
0x80, 0x07, 0x00, 0xe0, 0x03, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0xc0, 0x00, 0xf0, 0x03, 0x00, 0x00, 0x80, 0x0f, 0x00, 0x03, 0x00,
0x00, 0x0f, 0x00, 0x1e, 0x80, 0x07, 0x3c, 0x80, 0x07, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x1e, 0x80, 0x07, 0x00, 0xc0, 0x03, 0x0e, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00,
0x1f, 0x00, 0x03, 0x00, 0x00, 0x0f, 0x00, 0x3c, 0xc0, 0x03, 0x3c, 0xc0,
0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x3c, 0xc0, 0x03, 0x00, 0xc0,
0x03, 0x1e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x01, 0x78,
0x00, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x03, 0x00, 0x00, 0x0f, 0x00, 0x3c,
0xc0, 0x03, 0x3c, 0xc0, 0x07, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x3c,
0xc0, 0x03, 0x00, 0xc0, 0x03, 0x1e, 0x00, 0x00, 0x00, 0x0f, 0x00, 0x00,
0x00, 0x80, 0x01, 0x1e, 0x00, 0x00, 0x00, 0x00, 0x78, 0x80, 0x01, 0x00,
0x00, 0x0f, 0x00, 0x78, 0xe0, 0x01, 0x78, 0xe0, 0x07, 0x00, 0x00, 0x00,
0x3c, 0x00, 0x00, 0x78, 0xe0, 0x01, 0x0c, 0xe0, 0x01, 0x7c, 0x80, 0x03,
0x00, 0x0f, 0x00, 0x00, 0x00, 0x80, 0x41, 0x1e, 0x00, 0x00, 0x00, 0x00,
0xf0, 0x85, 0x01, 0x00, 0x00, 0x0f, 0x00, 0xf8, 0xff, 0x01, 0xf8, 0xff,
0x07, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x0f, 0xf8, 0xff, 0x01, 0xfc, 0xff,
0x01, 0xf8, 0xff, 0x03, 0x00, 0x0f, 0x00, 0x00, 0x00, 0x80, 0xf3, 0x0f,
0x00, 0x00, 0x00, 0x00, 0xf0, 0xcf, 0x01, 0x00, 0x00, 0x0f, 0x00, 0xf0,
0xff, 0x00, 0xf0, 0xbf, 0x07, 0x00, 0x00, 0x00, 0xfc, 0xff, 0x0f, 0xf0,
0xff, 0x00, 0xfc, 0xff, 0x00, 0xf0, 0xff, 0x03, 0x00, 0x0f, 0x00, 0x00,
0x00, 0x00, 0xe3, 0x07, 0x00, 0x00, 0x00, 0x00, 0xe0, 0xc7, 0x00, 0x00,
0x00, 0x0f, 0x00, 0xc0, 0x3f, 0x00, 0xe0, 0x9f, 0x07, 0x00, 0x00, 0x00,
0xfc, 0xff, 0x0f, 0xc0, 0x3f, 0x00, 0xf8, 0x3f, 0x00, 0xc0, 0xff, 0x00,
0x00, 0x0f, 0x00, 0x00, 0x00, 0x00, 0xc7, 0x0f, 0x00, 0x00, 0x00, 0x00,
0xe0, 0xe3, 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, 0x06, 0x0f,
0x00, 0x00, 0x00, 0x00, 0xe0, 0x60, 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, 0x0e, 0x0e, 0x00, 0x00, 0x00, 0x00, 0x70, 0x70, 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, 0x0c, 0x0c, 0x00, 0x00, 0x00, 0x00,
0x60, 0x30, 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, 0x18, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 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, 0x38, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1c, 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, 0x70, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x0e, 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, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 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, 0xc0, 0x01, 0x00, 0x00, 0x00, 0x00, 0x80, 0x03, 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, 0x80, 0x03, 0x00, 0x00, 0x00, 0x00,
0x80, 0x03, 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, 0x07,
0x00, 0x00, 0x00, 0x00, 0xe0, 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, 0x0e, 0x00, 0x00, 0x00, 0x00, 0xf0, 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, 0x1c, 0x00, 0x00, 0x00, 0x00,
0x38, 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, 0x78,
0x00, 0x00, 0x00, 0x00, 0x1e, 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, 0x01, 0x00, 0x00, 0x80, 0x0f, 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, 0xc0, 0x07, 0x00, 0x00, 0xc0,
0x03, 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,
0x1f, 0x00, 0x00, 0xfc, 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, 0xfc, 0x03, 0xc0, 0x7f, 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, 0x07,
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, 0xff, 0xff, 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, 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, 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,
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, 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, 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, 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,
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, 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, 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, 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,
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, 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, 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, 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,
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, 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, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00 };



    // takes input in XBM format.  assumes 320px width

void ui_pong(unsigned int height)
    {

     unsigned int row;

     unsigned int x;
     unsigned int y;

     row = 0;
     unsigned int data;


     for (y = row; y < height; y++)
     {
         for (x = 0; x < 40; x+=2)
         {
             // for each row, copy into fg_ram
             data = pong_bits[(y * 40) + x + 1];
             data = data << 8;
             data = data | pong_bits[(y*40) + x];

             IOWR16 (FG_RAM_BASE + (40 * y) + x, data);

         }

     }

    }


void ui_win(unsigned int height)
    {

     unsigned int row;

     unsigned int x;
     unsigned int y;

     row = 0;
     unsigned int data;


     for (y = row; y < height; y++)
     {
         for (x = 0; x < 40; x+=2)
         {
             // for each row, copy into fg_ram
```

```
            data = winner_bits[(y * 40) + x + 1];
            data = data << 8;
            data = data | winner_bits[(y*40) + x];

            IOWR16 (FG_RAM_BASE + (40 * y) + x, data);

        }

    }

   }

void ui_lose(unsigned int height)
   {

    unsigned int row;

    unsigned int x;
    unsigned int y;

    row = 0;
    unsigned int data;


    for (y = row; y < height; y++)
    {
        for (x = 0; x < 40; x+=2)
        {
            // for each row, copy into fg_ram
            data = loser_bits[(y * 40) + x + 1];
            data = data << 8;
            data = data | loser_bits[(y*40) + x];

            IOWR16 (FG_RAM_BASE + (40 * y) + x, data);

        }

    }

   }
```

bg.c

```
#include "bg.h"
#include "xio.h"
#include "vga.h"

void bg_randomize()
{

}

void bg_randomflip()
{
}



void bg_line(int n)
{
      int i;
      for (i=0 ; i<16 ; i++)
      {
        int tmp = (n+i)%16;
        IOWR16( BG_BASE+2*i, (1<<tmp) );
        IOWR16( VGA_BG_COLOR, (1<< tmp));
      }

    n++;
}


void bg_set( unsigned short* buf )
{
      int i;

      for (i=0 ; i<16 ; i++)
      {
            IOWR16( BG_BASE+i, buf[i] );
      }
}
```

dm9000a.c

```
#include <stdio.h>
#include "dm9000a.h"
#include "dm9000a_io.h"

#include "packet.h"
#include "game.h"
#include "string.h"

unsigned char buf[1514];

void dm9000a_flush()
{
    dm9000a_iow( 0x25, 0 );     // rx sram high address
    dm9000a_iow( 0x24, 0 );     // rx sram low address
}

int dm9000a_poll()
{
    return dm9000a_ior(ISR) & 0x01;  // bit 1, packet received
}



int dm9000a_link()
{
    // read network status bit
    dm9000a_iow( NSR, 0xff );
    unsigned char tmp = dm9000a_ior(NSR);
    // link status bit
    return !!(tmp & (1<<6));
}


extern state_t rgs;

void dm9000a_isr()
{
    putchar( 'E' );

    unsigned int n = sizeof(buf);
    int status = dm9000a_rx(buf, &n);

    packet_t* pkt = (packet_t*)buf;

    if (status == DMFE_SUCCESS )
    {
        // make sure first five bytes of source MAC address match
        if (memcmp( pkt->enet.saddr, txpkt.enet.saddr, 5) == 0)
        {
            import_state( buf );
            printf( "sn=%d\n", gs.sn );
        }
        else
            printf( "enet: unrecognized packet\n" );
```

```
    }
    else
    {
        printf( "enet: packet rx failure %d\n", status );
        printf( "enet: nsr=%02x rxs=%02x rof=%02x\n",
            dm9000a_ior(0x01),
            dm9000a_ior(0x06),
            dm9000a_ior(0x07)
        );
    }

    /* Clear the DM9000A ISR: PRS, PTS, ROS, ROOS 4 bits, by RW/C1 */
    dm9000a_iow(ISR, 0x3F);

    /* Re-enable DM9000A interrupts */
    dm9000a_iow(IMR, INTR_set);
}

void dm9000a_start()
{
    if ( dm9000a_init(txpkt.enet.saddr) != DMFE_SUCCESS )
    {
      printf( "Ethernet init FAILED.\n" );
      while ( 1 )
         ;
    }

    usleep( 3000000 );

    if (!dm9000a_link())
    {
        printf("WARNING: no ethernet link.\n");
    }


  alt_irq_register(DE2_ENET_INST_IRQ, NULL, (void*)dm9000a_isr);
}



//
// below here directly from lab2...
//


void dm9000a_iow(unsigned int reg, unsigned int data)
{
  IOWR(DM9000A_BASE, IO_addr, reg);
  usleep(STD_DELAY);
  IOWR(DM9000A_BASE, IO_data, data);
}

unsigned int dm9000a_ior(unsigned int reg)
{
  IOWR(DM9000A_BASE, IO_addr, reg);
  usleep(STD_DELAY);
  return IORD(DM9000A_BASE, IO_data);
```

```
}


void phy_write(unsigned int reg, unsigned int value)
{
  /* set PHY register address into EPAR REG. 0CH */
  dm9000a_iow(0x0C, reg | 0x40); /* PHY register address setting,
                     and DM9000_PHY offset = 0x40 */

  /* fill PHY WRITE data into EPDR REG. 0EH & REG. 0DH */
  dm9000a_iow(0x0E, ((value >> 8) & 0xFF));   /* PHY data high_byte */
  dm9000a_iow(0x0D, value & 0xFF);            /* PHY data low_byte */

  /* issue PHY + WRITE command = 0xa into EPCR REG. 0BH */
  dm9000a_iow(0x0B, 0x8);                     /* clear PHY command first */
  IOWR(DM9000A_BASE, IO_data, 0x0A);  /* issue PHY + WRITE command */
  usleep(STD_DELAY);
  IOWR(DM9000A_BASE, IO_data, 0x08);  /* clear PHY command again */
  usleep(50);  /* wait 1~30 us (>20 us) for PHY + WRITE completion */
}

/* DM9000_init I/O routine */
unsigned int dm9000a_init(unsigned char *mac_address)
{
  unsigned int  i;

  /* set the internal PHY power-on (GPIOs normal settings) */
  dm9000a_iow(0x1E, 0x01);  /* GPCR REG. 1EH = 1 selected
                   GPIO0 "output" port for internal PHY */
  dm9000a_iow(0x1F, 0x00);  /* GPR  REG. 1FH GEPIO0
                   Bit [0] = 0 to activate internal PHY */
  msleep(5);        /* wait > 2 ms for PHY power-up ready */

  /* software-RESET NIC */
  dm9000a_iow(NCR, 0x03);   /* NCR REG. 00 RST Bit [0] = 1 reset on,
                   and LBK Bit [2:1] = 01b MAC loopback on */
  usleep(100);       /* wait > 10us for a software-RESET ok */
  dm9000a_iow(NCR, 0x00);   /* normalize */
  dm9000a_iow(NCR, 0x03);
  usleep(100);
  dm9000a_iow(NCR, 0x00);

  /* set GPIO0=1 then GPIO0=0 to turn off and on the internal PHY */
  dm9000a_iow(0x1F, 0x01);  /* GPR PHYPD Bit [0] = 1 turn-off PHY */
  dm9000a_iow(0x1F, 0x00);  /* PHYPD Bit [0] = 0 activate phyxcer */
  msleep(10);       /* wait >4 ms for PHY power-up */

  /* set PHY operation mode */
  phy_write(0,PHY_reset);   /* reset PHY registers back to the default state
*/
  usleep(100);               /* wait >30 us for PHY software-RESET ok */
  phy_write(16, 0x404);     /* turn off PHY reduce-power-down mode only */
  phy_write(4, PHY_txab);   /* set PHY TX advertised ability:
                        ALL + Flow_control */
  phy_write(0, 0x1200);     /* PHY auto-NEGO re-start enable
                        (RESTART_AUTO_NEGOTIATION +
                        AUTO_NEGOTIATION_ENABLE)
```

```
                        to auto sense and recovery PHY registers */
  msleep(5);               /* wait >2 ms for PHY auto-sense
                        linking to partner */

  /* store MAC address into NIC */
  for (i = 0; i < 6; i++)
    dm9000a_iow(16 + i, mac_address[i]);

  /* clear any pending interrupt */
  dm9000a_iow(ISR, 0x3F);  /* clear the ISR status: PRS, PTS, ROS, ROOS 4
bits,
                  by RW/C1 */
  dm9000a_iow(NSR, 0x2C);  /* clear the TX status: TX1END, TX2END, WAKEUP 3
bits,
                  by RW/C1 */

  /* program operating registers~ */
  dm9000a_iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions
                 (and disable this MAC loopback mode back to normal) */
  dm9000a_iow(0x08, BPTR_set); /* BPTR  REG.08  (if necessary) RX Back
Pressure
                    Threshold in Half duplex moe only:
                    High Water 3KB, 600 us */
  dm9000a_iow(0x09, FCTR_set);  /* FCTR  REG.09  (if necessary)
                     Flow Control Threshold setting
                     High/ Low Water Overflow 5KB/ 10KB */
  dm9000a_iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary)
                     RX/TX Flow Control Register enable TXPEN, BKPM
                     (TX_Half), FLCE (RX) */
  dm9000a_iow(0x0F, 0x00);      /* Clear the all Event */
  dm9000a_iow(0x2D, 0x80);      /* Switch LED to mode 1 */

  /* set other registers depending on applications */
  dm9000a_iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */

  /* enable interrupts to activate DM9000 ~on */
  dm9000a_iow(IMR, INTR_set);   /* IMR REG. FFH PAR=1 only,
                     or + PTM=1& PRM=1 enable RxTx interrupts */

  /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
  dm9000a_iow(RCR , RCR_set | RX_ENABLE | PASS_MULTICAST);
  /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */

  /* RETURN "DEVICE_SUCCESS" back to upper layer */
  return  (dm9000a_ior(0x2D)==0x80) ? DMFE_SUCCESS : DMFE_FAIL;
}

unsigned int dm9000a_tx(unsigned char *data_ptr, unsigned int tx_len)
{
  unsigned int i;

  /* mask NIC interrupts IMR: PAR only */
  dm9000a_iow(IMR, PAR_set);

  /* issue TX packet's length into TXPLH REG. FDH & TXPLL REG. FCH */
  dm9000a_iow(0xFD, (tx_len >> 8) & 0xFF);  /* TXPLH High_byte length */
  dm9000a_iow(0xFC, tx_len & 0xFF);         /* TXPLL Low_byte  length */
```

```
  /* wirte transmit data to chip SRAM */
  IOWR(DM9000A_BASE, IO_addr, MWCMD);  /* set MWCMD REG. F8H
                                  TX I/O port ready */
  for (i = 0; i < tx_len; i += 2) {
    usleep(STD_DELAY);
    IOWR(DM9000A_BASE, IO_data, (data_ptr[i+1]<<8)|data_ptr[i] );
  }

  /* issue TX polling command activated */
  dm9000a_iow(TCR , TCR_set | TX_REQUEST);  /* TXCR Bit [0] TXREQ auto clear
                                 after TX completed */

  /* wait TX transmit done */
  while(!(dm9000a_ior(NSR)&0x0C))
    usleep(STD_DELAY);

  /* clear the NSR Register */
  dm9000a_iow(NSR,0x00);

  /* re-enable NIC interrupts */
  dm9000a_iow(IMR, INTR_set);

  /* RETURN "TX_SUCCESS" to upper layer */
  return  DMFE_SUCCESS;
}

unsigned int dm9000a_rx(unsigned char *data_ptr, unsigned int *rx_len)
{
  unsigned char rx_READY, GoodPacket;
  unsigned int  Tmp, RxStatus, i;

  RxStatus = rx_len[0] = 0;
  GoodPacket=FALSE;

  /* mask NIC interrupts IMR: PAR only */
  dm9000a_iow(IMR, PAR_set);

  /* dummy read a byte from MRCMDX REG. F0H */
  rx_READY = dm9000a_ior(MRCMDX);

  /* got most updated byte: rx_READY */
  rx_READY = IORD(DM9000A_BASE,IO_data)&0x03;
  usleep(STD_DELAY);

  /* check if (rx_READY == 0x01): Received Packet READY? */
  if (rx_READY == DM9000_PKT_READY) {

    /* got RX_Status & RX_Length from RX SRAM */
    IOWR(DM9000A_BASE, IO_addr, MRCMD); /* set MRCMD REG. F2H
                                   RX I/O port ready */
    usleep(STD_DELAY);
    RxStatus = IORD(DM9000A_BASE,IO_data);
    usleep(STD_DELAY);
    rx_len[0] = IORD(DM9000A_BASE,IO_data);

    /* Check this packet_status GOOD or BAD? */
```

```
    if ( !(RxStatus & 0xBF00) && (rx_len[0] < MAX_PACKET_SIZE) ) {
      /* read 1 received packet from RX SRAM into RX buffer */
      for (i = 0; i < rx_len[0]; i += 2) {
      usleep(STD_DELAY);
      Tmp = IORD(DM9000A_BASE, IO_data);
      data_ptr[i] = Tmp & 0xFF;
      data_ptr[i+1] = (Tmp>>8) & 0xFF;
      }
      GoodPacket = TRUE;
    } else {
      /* this packet is bad, dump it from RX SRAM */
      for (i = 0; i < rx_len[0]; i += 2) {
      usleep(STD_DELAY);
      Tmp = IORD(DM9000A_BASE, IO_data);
      }
      printf("\nError\n");
      rx_len[0] = 0;
    }
  } else if (rx_READY) { /* status check first byte:
                      rx_READY Bit[1:0] must be "00"b or "01"b */

    /* software-RESET NIC */

    dm9000a_iow(NCR, 0x03);   /* NCR REG. 00 RST Bit [0] = 1 reset on,
                  and LBK Bit [2:1] = 01b MAC loopback on */
    usleep(20);       /* wait > 10us for a software-RESET ok */
    dm9000a_iow(NCR, 0x00);   /* normalize */
    dm9000a_iow(NCR, 0x03);
    usleep(20);
    dm9000a_iow(NCR, 0x00);
    /* program operating registers~ */
    dm9000a_iow(NCR, NCR_set); /* NCR REG. 00 enable the chip functions
                  (and disable this MAC loopback mode back to normal) */
    dm9000a_iow(0x08, BPTR_set);  /* BPTR  REG.08  (if necessary) RX Back
Pressure
                       Threshold in Half duplex moe only:
                       High Water 3KB, 600 us */
    dm9000a_iow(0x09, FCTR_set);  /* FCTR  REG.09  (if necessary)
                       Flow Control Threshold setting High/Low Water
                       Overflow 5KB/ 10KB */
    dm9000a_iow(0x0A, RTFCR_set); /* RTFCR REG.0AH (if necessary)
                       RX/TX Flow Control Register
                       enable TXPEN, BKPM (TX_Half), FLCE (RX) */
    dm9000a_iow(0x0F, 0x00);      /* Clear the all Event */
    dm9000a_iow(0x2D, 0x80);      /* Switch LED to mode 1 */
    /* set other registers depending on applications */
    dm9000a_iow(ETXCSR, ETXCSR_set); /* Early Transmit 75% */
    /* enable interrupts to activate DM9000 ~on */
    dm9000a_iow(IMR, INTR_set);   /* IMR REG. FFH PAR=1 only,
                       or + PTM=1& PRM=1 enable RxTx interrupts */
    /* enable RX (Broadcast/ ALL_MULTICAST) ~go */
    dm9000a_iow(RCR , RCR_set | RX_ENABLE | PASS_MULTICAST);
      /* RCR REG. 05 RXEN Bit [0] = 1 to enable the RX machine/ filter */
  }

  return GoodPacket ? DMFE_SUCCESS : DMFE_FAIL;
}
```

game.c

```
#include "game.h"
#include <system.h>
#include <io.h>
#include "packet.h"
#include <stdio.h>
#include "mouse.h"
#include "vga.h"
#include <stdlib.h>           // for abs()
#include <string.h>           // for memcpy()
#include "dm9000a.h"

#define IOWR16(base,data) \
    IOWR_16DIRECT(base,0,data)

state_t gs = { .sn=0, .p1 = {SCREEN_Y/2, 80, 0}, .p2={SCREEN_Y/2, 80, 0},
.ball={SCREEN_X/2, SCREEN_Y/2, -1*BALL_SPEED, 0, 1} };

// remote game state
state_t rgs;


player_t* gs_pl()
{
    if ( gs.pn == 1 )
        return &gs.p1;
    else
        return &gs.p2;
}

player_t* gs_pr()
{
    if ( gs.pn == 1 )
        return &gs.p2;
    else
        return &gs.p1;
}

void import_state(char* buf)
{
    state_t* ptr = (state_t*)((packet_t*)buf)->data;

    memcpy( &gs.ball, &ptr->ball, sizeof(ball_t) );
    gs.sn = ptr->sn;

    gs.p1.score = ptr->p1.score;
    gs.p2.score = ptr->p2.score;
}


void show_state()
{
    short tmp;

      IOWR16(VGA_BX, gs.ball.x);
```

```
      IOWR16(VGA_BY, gs.ball.y);

      tmp=gs.p1.c-gs.p1.l/2;
      IOWR16(VGA_P1A, tmp<0 ? 0 : tmp);
      IOWR16(VGA_P1B, gs.p1.c+gs.p1.l/2);

    tmp=gs.p2.c-gs.p2.l/2;
      IOWR16(VGA_P2A, tmp<0 ? 0 : tmp);
      IOWR16(VGA_P2B, gs.p2.c+gs.p2.l/2);
}



// returns player who gets point, NOT side on which ball crossed
int score_position()
{
    // ball next x-position
    int xn = gs.ball.x + gs.ball.vx;

    // right side: point for player 1
    if (gs.ball.x <= BALL_SCORE_RIGHT && xn > BALL_SCORE_RIGHT )
        return 1;

    // left side: point for player 2
    if (gs.ball.x >= BALL_SCORE_LEFT && xn < BALL_SCORE_LEFT )
        return 2;

    return 0;
}

void paddle_spin()
{
    // TODO: make sure it's the local player

    if (abs(mouse.vy) > 15)
        gs.ball.vy -= (mouse.vy/2);
    else if (abs(mouse.vy) > 7)
        gs.ball.vy -= (mouse.vy/8);
}

void handoff()
{
    printf( "handoff from %d vx=%d sn=%d\n", gs.pn, gs.ball.vx, gs.sn );

    // change screen of ball
    gs.ball.side = (gs.pn==1) ? 2 : 1;

    // mirror about x-axis
    gs.ball.x = SCREEN_X - gs.ball.x;
    printf( "new ball x=%d\n", gs.ball.x );

    // transmit packet
    transmit_state();

    // hide ball on our side
    gs.ball.x = 10;
    gs.ball.y = 10;
```

```
}

int paddle_bounce()
{
    int xn = gs.ball.x + gs.ball.vx;

    if (gs.ball.x >= PADDLE_EDGE_LEFT && xn < PADDLE_EDGE_LEFT
        && PADDLE_MIN(gs.p1) <= gs.ball.y && PADDLE_MAX(gs.p1) >= gs.ball.y )
       return 1;

    if (gs.ball.x <= PADDLE_EDGE_RIGHT && xn > PADDLE_EDGE_RIGHT
        && PADDLE_MIN(gs.p2) <= gs.ball.y && PADDLE_MAX(gs.p2) >= gs.ball.y )
        return 2;

    return 0;
}


int wall_bounce()
{
    // bounce off top and bottom
    return ( gs.ball.y < BALL_RADIUS || gs.ball.y > (SCREEN_Y-BALL_RADIUS) );
}

void update_state()
{
//    printf( "x=%d y=%d vx=%d vy=%d\n", gs.ball.x, gs.ball.y, gs.ball.vx,
gs.ball.vy );


      // to keep the math easy...
      gs.ball.x %= 1024;
      gs.ball.y %= 1024;

    int b;
    if (wall_bounce())
    {
        gs.ball.vy *= -1;
        printf( "wall bounce on %d\n", gs.pn );
    }

    else if ( (b=paddle_bounce()) == get_player())
    {
        gs.ball.vx *= -1;

        if (gs.ball.vx > 0)
            gs.ball.vx++;
        else
            gs.ball.vx--;

        paddle_spin();

        printf( "paddle bounce vy=%d on %d\n", mouse.vy, gs.pn );

    }
```

```
    int pn;
    if ( (pn = score_position())  )
    {
        // other player scored on  you
        if (pn!=get_player())
        {
            printf( "you lose a point\n" );

            gs.ball.vx = 0;
            gs.ball.vy = 0;
            gs_pr()->score++;
            gs.sn = (gs.pn == 1) ? 2 : 1;

            ui_updatescore();
            handoff();
        }
        else if ( (gs.pn==1 && gs.ball.vx>0) || (gs.pn==2 && gs.ball.vx<0) )
        {
            handoff();
            return;
        }
    }

    gs.ball.x += gs.ball.vx;
    gs.ball.y += gs.ball.vy;

}


void transmit_state()
{
    memcpy( txpkt.data, &gs, sizeof(gs) );
    dm9000a_tx( (u8*)&txpkt, sizeof(txpkt) );
}

int get_player()
{
    return gs.pn;
}
```

graphics.c

```
/* Rudimentary character generator */

#include <io.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "system.h"
#include "sys/alt_irq.h"
#include "vga.h"
#include "xio.h"
#include "graphics.h"


/* Font from /lib/kbd/consolefonts/lat0-16.psfu.gz

od --address-radix=n --width=16 -v -t x1 -j 4 -N 2048 lat0-16.psfu

*/

unsigned char font[] = {
  0x00, 0x00, 0x7e, 0xc3, 0x99, 0x99, 0xf3, 0xe7, 0xe7, 0xff, 0xe7, 0xe7,
0x7e, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xdc, 0x00, 0x76, 0xdc, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x6e, 0xf8, 0xd8, 0xd8, 0xdc, 0xd8, 0xd8, 0xd8, 0xf8, 0x6e,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x6e, 0xdb, 0xdb, 0xdf, 0xd8, 0xdb, 0x6e,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x10, 0x38, 0x7c, 0xfe, 0x7c, 0x38, 0x10, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x88, 0x88, 0xf8, 0x88, 0x88, 0x00, 0x3e, 0x08, 0x08, 0x08, 0x08,
0x00, 0x00, 0x00, 0x00,
  0x00, 0xf8, 0x80, 0xe0, 0x80, 0x80, 0x00, 0x3e, 0x20, 0x38, 0x20, 0x20,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x70, 0x88, 0x80, 0x88, 0x70, 0x00, 0x3c, 0x22, 0x3c, 0x24, 0x22,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x80, 0x80, 0x80, 0x80, 0xf8, 0x00, 0x3e, 0x20, 0x38, 0x20, 0x20,
0x00, 0x00, 0x00, 0x00,
  0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44, 0x11, 0x44,
0x11, 0x44, 0x11, 0x44,
  0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa,
0x55, 0xaa, 0x55, 0xaa,
  0xdd, 0x77, 0xdd, 0x77, 0xdd, 0x77, 0xdd, 0x77, 0xdd, 0x77, 0xdd, 0x77,
0xdd, 0x77, 0xdd, 0x77,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff, 0xff,
0xff, 0xff, 0xff, 0xff,
  0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0, 0xf0,
0xf0, 0xf0, 0xf0, 0xf0,
```

```
  0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f, 0x0f,
0x0f, 0x0f, 0x0f, 0x0f,
  0x00, 0x88, 0xc8, 0xa8, 0x98, 0x88, 0x00, 0x20, 0x20, 0x20, 0x20, 0x3e,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x88, 0x88, 0x50, 0x50, 0x20, 0x00, 0x3e, 0x08, 0x08, 0x08, 0x08,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x0e, 0x38, 0xe0, 0x38, 0x0e, 0x00, 0xfe, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xe0, 0x38, 0x0e, 0x38, 0xe0, 0x00, 0xfe, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x06, 0x0c, 0xfe, 0x18, 0x30, 0xfe, 0x60, 0xc0, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x06, 0x1e, 0x7e, 0xfe, 0x7e, 0x1e, 0x06, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xc0, 0xf0, 0xfc, 0xfe, 0xfc, 0xf0, 0xc0, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x18, 0x3c, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x7e, 0x3c, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x0c, 0xfe, 0x0c, 0x18, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x60, 0xfe, 0x60, 0x30, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x18, 0x3c, 0x7e, 0x18, 0x18, 0x18, 0x18, 0x7e, 0x3c, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x28, 0x6c, 0xfe, 0x6c, 0x28, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x06, 0x36, 0x66, 0xfe, 0x60, 0x30, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xfe, 0x6e, 0x6c, 0x6c, 0x6c, 0x6c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x18, 0x3c, 0x3c, 0x3c, 0x18, 0x18, 0x18, 0x00, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x66, 0x66, 0x66, 0x24, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x6c, 0x6c, 0xfe, 0x6c, 0x6c, 0x6c, 0xfe, 0x6c, 0x6c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x10, 0x10, 0x7c, 0xd6, 0xd0, 0xd0, 0x7c, 0x16, 0x16, 0xd6, 0x7c,
0x10, 0x10, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xc2, 0xc6, 0x0c, 0x18, 0x30, 0x60, 0xc6, 0x86,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x38, 0x6c, 0x6c, 0x38, 0x76, 0xdc, 0xcc, 0xcc, 0xcc, 0x76,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x18, 0x18, 0x18, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x0c, 0x18, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x18, 0x0c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x30, 0x18, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x18, 0x30,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x3c, 0xff, 0x3c, 0x66, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x7e, 0x18, 0x18, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
```

```
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x18,
0x30, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x0c, 0x18, 0x30, 0x60, 0xc0, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xce, 0xce, 0xd6, 0xd6, 0xe6, 0xe6, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x18, 0x38, 0x78, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x7e,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0x06, 0x0c, 0x18, 0x30, 0x60, 0xc0, 0xc6, 0xfe,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0x06, 0x06, 0x3c, 0x06, 0x06, 0x06, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x0c, 0x1c, 0x3c, 0x6c, 0xcc, 0xfe, 0x0c, 0x0c, 0x0c, 0x1e,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xfe, 0xc0, 0xc0, 0xc0, 0xfc, 0x06, 0x06, 0x06, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x38, 0x60, 0xc0, 0xc0, 0xfc, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xfe, 0xc6, 0x06, 0x06, 0x0c, 0x18, 0x30, 0x30, 0x30, 0x30,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0x7e, 0x06, 0x06, 0x06, 0x0c, 0x78,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x18, 0x18, 0x30,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x06, 0x0c, 0x18, 0x30, 0x60, 0x30, 0x18, 0x0c, 0x06,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0x00, 0x00, 0xfe, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x60, 0x30, 0x18, 0x0c, 0x06, 0x0c, 0x18, 0x30, 0x60,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xc6, 0x0c, 0x18, 0x18, 0x18, 0x00, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xde, 0xde, 0xde, 0xdc, 0xc0, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x10, 0x38, 0x6c, 0xc6, 0xc6, 0xfe, 0xc6, 0xc6, 0xc6, 0xc6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c, 0x66, 0x66, 0x66, 0x66, 0xfc,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0, 0xc0, 0xc0, 0xc2, 0x66, 0x3c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xf8, 0x6c, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66, 0x6c, 0xf8,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xfe, 0x66, 0x62, 0x68, 0x78, 0x68, 0x60, 0x62, 0x66, 0xfe,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xfe, 0x66, 0x62, 0x68, 0x78, 0x68, 0x60, 0x60, 0x60, 0xf0,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x3c, 0x66, 0xc2, 0xc0, 0xc0, 0xde, 0xc6, 0xc6, 0x66, 0x3a,
0x00, 0x00, 0x00, 0x00,
```

```
  0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xfe, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x3c, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x1e, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0xcc, 0xcc, 0xcc, 0x78,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xe6, 0x66, 0x66, 0x6c, 0x78, 0x78, 0x6c, 0x66, 0x66, 0xe6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xf0, 0x60, 0x60, 0x60, 0x60, 0x60, 0x60, 0x62, 0x66, 0xfe,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xc6, 0xee, 0xfe, 0xfe, 0xd6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xc6, 0xe6, 0xf6, 0xfe, 0xde, 0xce, 0xc6, 0xc6, 0xc6, 0xc6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c, 0x60, 0x60, 0x60, 0x60, 0xf0,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xd6, 0xde, 0x7c,
0x0c, 0x0e, 0x00, 0x00,
  0x00, 0x00, 0xfc, 0x66, 0x66, 0x66, 0x7c, 0x6c, 0x66, 0x66, 0x66, 0xe6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7c, 0xc6, 0xc6, 0x60, 0x38, 0x0c, 0x06, 0xc6, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x7e, 0x7e, 0x5a, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x6c, 0x38, 0x10,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xd6, 0xd6, 0xd6, 0xfe, 0xee, 0x6c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xc6, 0xc6, 0x6c, 0x7c, 0x38, 0x38, 0x7c, 0x6c, 0xc6, 0xc6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3c, 0x18, 0x18, 0x18, 0x18, 0x3c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xfe, 0xc6, 0x86, 0x0c, 0x18, 0x30, 0x60, 0xc2, 0xc6, 0xfe,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x3c, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x3c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x60, 0x30, 0x18, 0x0c, 0x06, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x3c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x0c, 0x3c,
0x00, 0x00, 0x00, 0x00,
  0x10, 0x38, 0x6c, 0xc6, 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, 0xff, 0x00,
  0x00, 0x30, 0x30, 0x30, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x78, 0x0c, 0x7c, 0xcc, 0xcc, 0xcc, 0x76,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0xe0, 0x60, 0x60, 0x78, 0x6c, 0x66, 0x66, 0x66, 0x66, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc0, 0xc0, 0xc0, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
```

```
  0x00, 0x00, 0x1c, 0x0c, 0x0c, 0x3c, 0x6c, 0xcc, 0xcc, 0xcc, 0xcc, 0x76,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xfe, 0xc0, 0xc0, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x38, 0x6c, 0x64, 0x60, 0xf0, 0x60, 0x60, 0x60, 0x60, 0xf0,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc, 0x7c,
0x0c, 0xcc, 0x78, 0x00,
  0x00, 0x00, 0xe0, 0x60, 0x60, 0x6c, 0x76, 0x66, 0x66, 0x66, 0x66, 0xe6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x18, 0x18, 0x00, 0x38, 0x18, 0x18, 0x18, 0x18, 0x18, 0x3c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x06, 0x06, 0x00, 0x0e, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
0x66, 0x66, 0x3c, 0x00,
  0x00, 0x00, 0xe0, 0x60, 0x60, 0x66, 0x6c, 0x78, 0x78, 0x6c, 0x66, 0xe6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x70, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x34, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xec, 0xfe, 0xd6, 0xd6, 0xd6, 0xd6, 0xc6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x66, 0x66, 0x66, 0x66, 0x66, 0x66,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x66, 0x66, 0x66, 0x66, 0x66, 0x7c,
0x60, 0x60, 0xf0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x76, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc, 0x7c,
0x0c, 0x0c, 0x1e, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xdc, 0x76, 0x66, 0x60, 0x60, 0x60, 0xf0,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x7c, 0xc6, 0x60, 0x38, 0x0c, 0xc6, 0x7c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x10, 0x30, 0x30, 0xfc, 0x30, 0x30, 0x30, 0x30, 0x36, 0x1c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc, 0xcc, 0x76,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x66, 0x66, 0x66, 0x66, 0x66, 0x3c, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xd6, 0xd6, 0xd6, 0xfe, 0x6c,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0x6c, 0x38, 0x38, 0x38, 0x6c, 0xc6,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0xc6, 0x7e,
0x06, 0x0c, 0xf8, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xfe, 0xcc, 0x18, 0x30, 0x60, 0xc6, 0xfe,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x0e, 0x18, 0x18, 0x18, 0x70, 0x18, 0x18, 0x18, 0x18, 0x0e,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x70, 0x18, 0x18, 0x18, 0x0e, 0x18, 0x18, 0x18, 0x18, 0x70,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x76, 0xdc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
  0x00, 0x66, 0x00, 0x66, 0x66, 0x66, 0x66, 0x3c, 0x18, 0x18, 0x18, 0x3c,
0x00, 0x00, 0x00, 0x00,
};
```

```
void put_vga_char(char c, unsigned int column, unsigned int row)
{
  int i;
  unsigned pixels;
  unsigned int byte;

  byte = 0;

    if (column % 2 != 0)
    {
        byte = 1;
        column--;
    }

  c &= 0x7f;
  int row_offset = row * 40;

      for (i = 0 ; i < 15 ; ++i)
       {
        int j;
        pixels = font[c * 16 + i];

        unsigned short tmp = IORD16( (FG_RAM_BASE+row_offset) + (40*i) +
column );
        unsigned short bak = tmp;

        for (j = 0 ; j < 8 ; ++j)
         {
            if (pixels & (0x80>>j))
                tmp |= (1<<j);
            else
                tmp &= ~(1<<j);
         }

         if (byte == 0)
         {
            tmp = tmp << 1;

         }
         else
         {
            tmp = tmp << 9;
            tmp = tmp | (bak & 0xff);

         }

        IOWR16( (FG_RAM_BASE + row_offset) + (40*i) + column, tmp );
      }


}

void put_vga_string(char *str, unsigned int column, unsigned int row)
{
```

```
  char c;
  while ( (c = *str++) )
  {
     put_vga_char(c, column, row);
     column +=1;
  }
}


void draw_vline(int pos, int start, int end)
{
    int i;

    if (start > end)
    {
        start = 0;
        end = 240;
    }

    for (i = start; i < end; i++)
    {
        unsigned short tmp = IORD16 (FG_RAM_BASE + ((40 * i) + pos));
        tmp = tmp | 0x02;
        IOWR16 (FG_RAM_BASE + ((40 * i ) + pos), tmp);
    }
}



void draw_vline_dotted(int pos, int start, int end)
{
    int i;

    if (start > end)
    {
        start = 0;
        end = 240;
    }

    for (i = start; i < end; i++)
    {
        unsigned short tmp = IORD16 (FG_RAM_BASE + ((40 * i) + pos));

        tmp = tmp | (i % 2 ? 0x02 : 0x00);

        IOWR16 (FG_RAM_BASE + ((40 * i ) + pos), tmp);
    }
}


void draw_hline(int pos, int start, int end)
{

    int i;

    if (start > end)
    {
```

```
        start = 0;
        end = 40;
    }

    for (i = start; i < end; i+=2)
    {
        unsigned short tmp = IORD16 (FG_RAM_BASE + ((40 * pos)) + i);
        tmp = tmp | 0xffff;
        IOWR16 (FG_RAM_BASE + ((40 * pos)) + i, tmp);
    }


}


void draw_centerline()
{
    draw_vline_dotted( 20, 0, 240 );
}



void draw_hline_dotted(int pos, int start, int end)
{

    int i;

    if (start > end)
    {
        start = 0;
        end = 40;
    }

    for (i = start; i < end; i+=2)
    {
        unsigned short tmp = IORD16 (FG_RAM_BASE + ((40 * pos)) + i);
        tmp = tmp | 0xaaaa;
        IOWR16 (FG_RAM_BASE + ((40 * pos)) + i, tmp);
    }

}




void inv_vga_char(char c, unsigned int column, unsigned int row, int last)
{
  int i;
  unsigned pixels;
  unsigned int byte;

  byte = 0;

    if (column % 2 != 0)
    {
        byte = 1;
        column--;
```

```
    }

  c &= 0x7f;
  int row_offset = row * 40;

      for (i = 0 ; i < 15 ; ++i)
       {
        int j;
        pixels = font[c * 16 + i];

        unsigned short tmp = IORD16( (FG_RAM_BASE+row_offset) + (40*i) +
column );
        unsigned short bak = ~tmp;

        for (j = 0 ; j < 8 ; ++j)
         {
            if (pixels & (0x80>>j))
                tmp |= (1<<j);
            else
                tmp &= ~(1<<j);
         }
         if (byte == 0)
         {
            tmp = tmp << 1;

         }
         else
         {
            tmp = tmp << 9;
            tmp = tmp | bak;
         }

         tmp = ~tmp;
         IOWR16( (FG_RAM_BASE + row_offset) + (40*i) + column, tmp );
      }


}



void inv_vga_string(char *str, unsigned int column, unsigned int row)
{
  char c;
  int length = strlen(str);
  int count = 0;
  int last = 0;

  while ( (c = *str++) )
  {

     if (count == (length - 1))
     {
        last = 1;
     }

     inv_vga_char(c, column, row, last);
```

```
      column +=1;
      count++;
   }

}


//
// Clears all Foreground RAM
//

void clr_fg ()
{
    int i;
    for (i = 0; i < 9600; i+=2)
    {
        IOWR16(( FG_RAM_BASE + i ), 0x0000);
    }
}


/*
 * Clears a subset of the Foreground RAM from 'start' to 'end'
 */

void clr_fg_sub (int start, int end)
{
    if (end < start)
    {
        start = 0;
        end = 0;
    }
    if (start % 2 != 0) start--; // make even
    if ((start < 0) | (start > 9600)) start = 0; // invalid range
    if (end > 9600) end = 9600;

    int i;
    for (i = start; i < end; i+=2)
    {
        IOWR16(( FG_RAM_BASE + i ), 0x0000);
    }

}



void fg_color (unsigned short color)
{
    IOWR16 (VGA_FG_COLOR, color);
}

void bg_color (unsigned short color)
{
    IOWR16 (VGA_BG_COLOR, color);
}

void go_color (unsigned short color)
```

```
{
    IOWR16 (VGA_GO_COLOR, color);
}

void sc_color (unsigned short color)
{
    IOWR16 (VGA_SC_COLOR, color);
}
```

mouse.c

```
#include <system.h>
#include <alt_types.h>
#include <sys/alt_irq.h>
#include <stdio.h>

#include "mouse.h"
#include "xio.h"

#define MOUSE_BASE  DE2_PS2_INST_BASE

#include <stdio.h>
#include <unistd.h>


mouse_t mouse;

inline void mouse_ie(int en)
{
    if (en)
        IOWR32(MOUSE_BASE+4, 0x0001);
    else
        IOWR32(MOUSE_BASE+4, 0x0000);
}




void mouse_isr()
{
    putchar( 'M' );

    static int n=0;
    unsigned char d;

    // magic sleep.  without it, third byte of mouse data packet gets lost in
FIFO,
    // only to be read at next interrupt.  this is bad.  long live magic
sleep!
    // (putting a sleep in an ISR is probably not an entirely great idea)

    int event=0;

    int i;
    for (i=0 ; i<2000 ; i++)
        ;

    while ( mouse_read( &d ) )
    {
        mouse.data[n] = d;
        if ( ++n==3 )
        {
            mouse.n++;
```

```
            n=0;

            // process new packet...
            int x, y;

            x = (int)MOUSE_X;
            if ( MOUSE_CTL & MOUSE_XSI )
                x |= 0xffffff00;

            y = (int)MOUSE_Y;
            if ( MOUSE_CTL & MOUSE_YSI )
                y |= 0xffffff00;

            mouse.vx = x;
            mouse.vy = y;

            mouse.x += x;
            mouse.y += y;

            mouse.b |= (MOUSE_CTL & (MOUSE_LB | MOUSE_RB | MOUSE_MB) );

            event++;
        }
    }
}



int mouse_event( int* x, int* y, int* button )
{
    static unsigned int n = 0;

    // no new events from isr
    if ( n == mouse.n )
        return 0;

    n = mouse.n;

//    if ( button != 0 )
//        *button = MOUSE_CTL & (MOUSE_LB | MOUSE_RB | MOUSE_MB);

    *x = mouse.x;
    *y = mouse.y;

    if ( button )
        *button = mouse.b;

    // disable interrupts, clear global mouse position counter
    mouse_ie(0);
    mouse.x = 0;
    mouse.y = 0;
    mouse.b = 0;
    mouse_ie(1);

    return 1;
}
```

```
int mouse_write( unsigned char cmd )
{
    IOWR32(MOUSE_BASE, cmd);

    return 1;
}


int mouse_read( unsigned char* c )
{
    unsigned int tmp = IORD32(MOUSE_BASE);

    if ( (tmp>>16) != 0 )
    {
        *c = (tmp&0xff);
        return 1;
    }
    else
    {
        return 0;
    }
}

int mouse_read_blocking( unsigned char* buf )
{
    unsigned int tmp;

    // that's right, real men poll!
    do {
        tmp = IORD32(MOUSE_BASE);
//        printf( "%08x\n", tmp );
//        usleep( 100000 );
    } while ( (tmp>>16) == 0 );


    if (buf == 0)
        return 0;

    *buf = (unsigned char)tmp;
    return 1;
}



void mouse_flush()
{
    unsigned int tmp;

    // that's right, real men poll!
    do {
        tmp = IORD32(MOUSE_BASE);
    } while ( (tmp>>16) != 0 );
}

void mouse_stream()
```

```
{
    char buf;

    mouse_write( 0xf3 );    // set sample rate
    mouse_read_blocking(&buf);
    mouse_write( 20 );      // 10Hz
    mouse_read_blocking(&buf);

//    mouse_write( 0xe8 );    // set resolution
//    mouse_read(0);          // ack
//    mouse_write( 3 );       // 8 counts/mm
//    mouse_read(0);          // ack


    mouse_write( 0xf4 );    // enable
    mouse_read_blocking(&buf);

}

int mouse_data( char* buf )
{
    mouse_read( &buf[0] );
    mouse_read( &buf[1] );
    mouse_read( &buf[2] );

    return 1;
}

int mouse_init()
{
    unsigned char ack, res, id;
    int i;

    mouse_flush();

    for (i=0 ; i<4 ; i++)
    {
      mouse_write( 0xff );

        if ( mouse_read_blocking(&ack) && mouse_read_blocking(&res) &&
mouse_read_blocking(&id) )
        {
            if ( ack!=-0xfa && res!=0xaa )
                return 0;
        }
    }

    printf( "Mouse detected.\n" );
    return 1;
}


void mouse_start()
{
    mouse_ie(0);

    mouse_init();
```

```
    mouse_stream();

//    usleep(100000);
//    mouse_flush();

    // register interrupt
    alt_irq_register( DE2_PS2_INST_IRQ, NULL, (void*)mouse_isr );
    // enable interrupt in peripheral
    mouse_ie(1);
}


void mouse_poll()
{
    unsigned char buf[4];
    int i;

    while (1)
    {
        mouse_write( 0xeb );

        // mouse sends 0xfa then movement packet
        for (i=0 ; i<4 ; i++)
        {
            mouse_read_blocking( &buf[i] );

            usleep(2000);

            if (i==0 && buf[0] != 0xfa)
                break;

        }

        if (buf[0] == 0xfa)
            break;
        else
        {
            printf( "error\n" );
            mouse_flush();
//            mouse_write(0xff);
        }
    }

    mouse.n++;

//    printf( "%02x %02x %02x %02x\n", buf[0], buf[1], buf[2], buf[3] );

    //                  0    1    2  3
    // packet contains: ack, ctl, x, y

    int x, y;

    x = (int)buf[2];
    if ( buf[1] & MOUSE_XSI )
        x |= 0xffffff00;

    y = (int)buf[3];
```

```
    if ( buf[1] & MOUSE_YSI )
        y |= 0xffffff00;

    mouse.vx = x;
    mouse.vy = y;

    mouse.x += x;
    mouse.y += y;

    mouse.b |= (buf[1] & (MOUSE_LB | MOUSE_RB | MOUSE_MB) );

}
```

packet.c

```
#include "types.h"
#include "packet.h"
#include <string.h>

packet_t txpkt =
{
      .enet =
      {
            .daddr = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff },
            .saddr = { 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa },

#ifdef  PACKET_L3L4
            .type = HTONS(0x0800)         // ip
#else
        .type = HTONS(0xdead)
#endif

      },

#ifdef  PACKET_L3L4
      .ip =
      {
            .vhl = 0x45,
            .ds = 0,
            .len =                                    // 20+8+256 = 0x011c

      HTONS(sizeof(txpkt.ip)+sizeof(txpkt.udp)+sizeof(txpkt.data)),

            .id = HTONS(0xabcd),
            .flag = 0,
            .frag = 0,
            .ttl = 0xff,
            .proto = 0x11,                        // udp
//          .cksum = HTONS(0x8ab2),       // header sum = 0x3754a, ones comp
= 0x754d, cksum = 0x8ab2
        .cksum = 0xffff,
            .saddr = HTONL(IP(192,168,1,1)),    // c0 a8 01 01
            .daddr = HTONL(IP(192,168,1,255))   // c0 a8 01 ff
      },

      .udp =
      {
            .sport = HTONS(1234),
            .dport = HTONS(5678),

            .len =                                // length of udp header and
data
                  HTONS(sizeof(txpkt.udp)+sizeof(txpkt.data)),

            .cksum = 0
      }
#endif

};
```

```
packet_t rxpkt;


void ip_cksum( packet_t* pkt )
{
#ifdef  PACKET_L3L4
      pkt->ip.cksum = 0;
      u32 cksum=0;
      u16* pt;
      u16* pe;

    // size in 16-bit words
      pe = (u16*)&pkt->ip + sizeof(pkt->ip)/2;

      for (pt=(u16*)&pkt->ip ; pt!=pe ; pt++)
            cksum += *pt;

    // the peculiar thing about checksums... don't need HTONS conversion,
    // since this is already taken care of by counters
      pkt->ip.cksum = ~((cksum & 0xffff) + (cksum >> 16) );

#endif
}



void udp_data( packet_t* pkt, u8* buf, u16 len )
{
      memcpy( pkt->data, buf, len );
      pkt->ip.len = HTONS( sizeof(pkt->udp)+len );
}


int pkt_valid( packet_t* pkt )
{
      return
            pkt->enet.type == NTOHS(0x0800)
            && pkt->ip.proto == 0x11;
}


int pkt_datalen( packet_t* pkt )
{
      return NTOHS(pkt->udp.len) - sizeof(pkt->udp);
}
```

ui.c

```
#include "packet.h"
#include "game.h"
#include "mouse.h"
#include <stdio.h>

#include "ui.h"
#include "graphics.h"

#include "banner.h"
#include <unistd.h>
#include "dm9000a.h"

void ui_updatescore()
{
    char score1[4];
    char score2[4];

    sprintf(score1, "%d", gs.p1.score);
    sprintf(score2, "%2d", gs.p2.score);

    // clear the top
//    clr_fg_sub(0, 800);

    put_vga_string( "  ", 1, 0 );
    put_vga_string (score1, 1, 0);

    put_vga_string( "  ", 37, 0);
    put_vga_string (score2, 37, 0);

}

int ui_getplayer()
{
    clr_fg();

    // display menu

    ui_pong(100);

    // default is player 1

    draw_vline_dotted(20, 100, 160);
    draw_vline_dotted(20, 180, 240);

    inv_vga_string ("Player 1", 16, 160);
    put_vga_string ("Player 2", 16, 180);

    int sel = 1;
    int done = 0;

    int vx, vy, b;
```

```
    while (done == 0)
    {
 //       mouse_poll();
        usleep(1000);

        if (mouse_event( &vx, &vy, &b))
        {
            if (sel == 1 && (vy < -10))
            {
                // clear the screen
                clr_fg_sub(6400,7820);
                put_vga_string ("Player 1", 16, 160);
                inv_vga_string ("Player 2", 16, 180);
                sel = 2;

            }

            if (sel == 2 && (vy > 10))
            {
                clr_fg_sub(6400,7820);
                inv_vga_string ("Player 1", 16, 160);
                put_vga_string ("Player 2", 16, 180);
                sel = 1;
            }

            if (b & MOUSE_LB)
            {

               if (sel == 1)
               {
                  return 1;
               }
               if (sel == 2)
               {
                  return 2;
               }

            }

        }
    }

    return -1;
}


void ui_waitopponent()
{
    rgs.ball.x = 0xdead;

    inv_vga_string( "Waiting for Opponent...", 8, 120 );

    while ( rgs.ball.x == 0xdead )
    {
        transmit_state();
        usleep( 1000000 );
    }
```

```
    put_vga_string( "                          ", 8, 120 );

    draw_centerline();
    ui_updatescore();

    dm9000a_flush();


}

void ui_updatescore_text()
{
    printf( "Score: P1=%d P2=%d\n", gs.p1.score, gs.p2.score );

}


void ui_winner()
{
    ui_win(100);

    usleep(1000000);
}

void ui_loser()
{
    ui_lose(100);

    usleep(1000000);
}
```

banner.h

```
#ifndef     BANNER_H
#define     BANNER_H

void  ui_pong(unsigned int height);
void  ui_win(unsigned int height);
void  ui_lose(unsigned int height);

#endif
```

bg.h

```
#ifndef     BG_H
#define     BG_H

#include <system.h>

#define     BG_BASE     (DE2_VGA_RASTER_INST_BASE+0)

#endif
```

dm9000a.h

```
#ifndef   __DM9000A_H__
#define   __DM9000A_H__

#define IO_addr     0
#define IO_data     1

#define NCR    0x00  /* Network  Control Register REG. 00 */
#define NSR    0x01  /* Network  Status Register  REG. 01 */
#define TCR    0x02  /* Transmit Control Register REG. 02 */
#define RCR    0x05  /* Receive  Control Register REG. 05 */
#define ETXCSR 0x30  /* TX early Control Register REG. 30 */
#define MRCMDX 0xF0  /* RX FIFO I/O port command: READ a byte from RX SRAM */
#define MRCMD  0xF2  /* RX FIFO I/O port command READ  from RX SRAM */
#define MWCMD  0xF8  /* TX FIFO I/O port command WRITE into TX FIFO */
#define ISR    0xFE  /* NIC Interrupt Status Register REG. FEH */
#define IMR    0xFF  /* NIC Interrupt Mask   Register REG. FFH */

#define NCR_set    0x00
#define TCR_set    0x00
#define TX_REQUEST 0x01  /* TCR REG. 02 TXREQ Bit [0] = 1 polling
                         Transmit Request command */
#define TCR_long   0x40  /* packet disable TX Jabber Timer */
#define RCR_set    0x30  /* skip CRC_packet and skip LONG_packet */
#define RX_ENABLE  0x01  /* RCR REG. 05 RXEN
                         Bit [0] = 1 to enable RX machine */
#define RCR_long   0x40  /* packet disable RX Watchdog Timer */
#define PASS_MULTICAST 0x08 /* RCR REG. 05 PASS_ALL_MULTICAST
                         Bit [3] = 1: RCR_set value ORed 0x08 */
#define BPTR_set   0x3F  /* BPTR REG. 08 RX Back Pressure Threshold:
                         High Water Overflow Threshold setting
                         3KB and Jam_Pattern_Time = 600 us */
#define FCTR_set   0x5A  /* FCTR REG. 09 High/ Low Water Overflow Threshold
                         setting 5KB/ 10KB */
#define RTFCR_set  0x29  /* RTFCR REG. 0AH RX/TX Flow Control Register
                         enable TXPEN + BKPM(TX_Half) + FLCE(RX) */
#define ETXCSR_set 0x83  /* Early Transmit Bit [7] Enable and
                         Threshold 0~3: 12.5%, 25%, 50%, 75% */
#define INTR_set   0x81  /* IMR REG. FFH: PAR +PRM, or 0x83: PAR + PRM + PTM
*/
#define PAR_set    0x80  /* IMR REG. FFH: PAR only, RX/TX FIFO R/W
                         Pointer Auto Return enable */

#define PHY_reset  0x8000  /* PHY reset: some registers back to
                           default value */
#define PHY_txab   0x05e1  /* set PHY TX advertised ability:
                           Full-capability + Flow-control (if necessary) */
#define PHY_mode   0x3100  /* set PHY media mode: Auto negotiation
                           (AUTO sense) */

// #define STD_DELAY       20      /* standard delay 20 us */
#define STD_DELAY 50

#define DMFE_SUCCESS    0
#define DMFE_FAIL       1
```

```
#define TRUE            1
#define FALSE           0

#define DM9000_PKT_READY  0x01  /* packets ready to receive */
#define PACKET_MIN_SIZE   0x40  /* Received packet min size */
#define MAX_PACKET_SIZE   1522  /* RX largest legal size packet
                           with fcs & QoS */
#define DM9000_PKT_MAX    3072  /* TX 1 packet max size without 4-byte CRC */
//------------------------------------------------------------------------
void  dm9000a_flush( void );
int   dm9000a_poll( void );
int   dm9000a_link( void );
void  dm9000a_isr( void );
void  dm9000a_start( void );
void  dm9000a_iow(unsigned int reg, unsigned int data);
unsigned    int dm9000a_ior(unsigned int reg);
void  phy_write(unsigned int reg, unsigned int value);
unsigned    int dm9000a_init(unsigned char *mac_address);
unsigned    int dm9000a_tx(unsigned char *data_ptr, unsigned int tx_len);
unsigned    int dm9000a_rx(unsigned char *data_ptr, unsigned int *rx_len);
//------------------------------------------------------------------------

int dm9000a_poll();

#endif
```

dm9000a_io.h

```
#ifndef   __basic_io_H__
#define   __basic_io_H__

#include <io.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include "system.h"
#include "sys/alt_irq.h"

#define DM9000A_BASE    DE2_ENET_INST_BASE

//  for GPIO
#define inport(base)                                    IORD(base, 0)
#define outport(base, data)                             IOWR(base, 0, data)
#define get_pio_dir(base)                               IORD(base, 1)
#define set_pio_dir(base, data)                         IOWR(base, 1, data)
#define get_pio_irq_mask(base)                          IORD(base, 2)
#define set_pio_irq_mask(base, data)                    IOWR(base, 2, data)
#define get_pio_edge_cap(base)                          IORD(base, 3)
#define set_pio_edge_cap(base, data)                    IOWR(base, 3, data)

//  for SEG7 Display
#define seg7_show(base,data)                            IOWR(base, 0, data)

//  for Time Delay
#define msleep(msec)                                    usleep(1000*msec);
#define Sleep(sec)                                      msleep(1000*sec);

#endif
```

game.h

```
#ifndef     GAME_H
#define     GAME_H

#include "ui.h"

#define     PADDLE_MIN(p) \
      (p.c-(p.l/2))

#define     PADDLE_MAX(p) \
      (p.c+(p.l/2))

#define     PADDLE_WIDTH      5
#define     PADDLE_OFFSET     20
#define     BALL_RADIUS       8

#define SCREEN_X        640
#define SCREEN_Y        480

#define     PADDLE_EDGE_LEFT  (0+PADDLE_OFFSET+PADDLE_WIDTH+BALL_RADIUS)
#define     PADDLE_EDGE_RIGHT (SCREEN_X-(PADDLE_OFFSET+PADDLE_WIDTH)-
BALL_RADIUS)

#define BALL_SCORE_LEFT (1)
#define BALL_SCORE_RIGHT (SCREEN_X-1)

#define BALL_SPEED      4


#include "vga.h"
#include "types.h"

typedef struct
{
      u16 x, y;
      short int vx, vy;

    short int side;
} ball_t;

typedef struct
{
      u16 c;
      u8    l;
      u8    score;
} player_t;

typedef struct
{
      ball_t      ball;

      player_t p1;
      player_t p2;

    short int pn;         // player number
```

```
    short int sn;         // server number (0=none)


} state_t;

extern state_t gs;
extern state_t rgs;

void  import_state(char*);
void  show_state( void );
int   score_position( void );
void  paddle_spin( void );
int   paddle_bounce( void );
int   wall_bounce( void );
void  update_state( void );
void  transmit_state( void );
int   get_player( void );
player_t* gs_pl();
player_t* gs_pr();
void handoff( void );

#endif
```

graphics.h

```
#ifndef     GRAPHICS_H
#define     GRAPHICS_H

void  put_vga_char(char c, unsigned int column, unsigned int row);
void  put_vga_string(char *str, unsigned int column, unsigned int row);
void  draw_vline(int pos, int start, int end);
void  draw_vline_dotted(int pos, int start, int end);
void  draw_hline(int pos, int start, int end);
void  draw_centerline( void );
void  draw_hline_dotted(int pos, int start, int end);
void  inv_vga_char(char c, unsigned int column, unsigned int row, int last);
void  inv_vga_string(char *str, unsigned int column, unsigned int row);
void  clr_fg ( void );
void  clr_fg_sub (int start, int end);
void  fg_color (unsigned short color);
void  bg_color (unsigned short color);
void  go_color (unsigned short color);
void  sc_color (unsigned short color);


#endif
```

mouse.h

```
#ifndef     MOUSE_H
#define     MOUSE_H

typedef struct
{
      int x, y, b, vx, vy;
      char data[3];
      unsigned int n;
} mouse_t;

extern mouse_t mouse;

#define     MOUSE_YOF   (1<<7)
#define     MOUSE_XOF   (1<<6)
#define     MOUSE_YSI   (1<<5)
#define     MOUSE_XSI   (1<<4)
#define     MOUSE_MB    (1<<2)
#define     MOUSE_RB    (1<<1)
#define     MOUSE_LB    (1<<0)

#define     MOUSE_CTL   (mouse.data[0])
#define     MOUSE_X           (mouse.data[1])
#define     MOUSE_Y           (mouse.data[2])

inline void mouse_ie(int en);
void  mouse_isr( void );
int   mouse_event( int* x, int* y, int* button );
int   mouse_write( unsigned char cmd );
int   mouse_read( unsigned char* c );
int   mouse_read_blocking( unsigned char* buf );
void  mouse_flush( void );
void  mouse_stream( void );
int   mouse_data( char* buf );
int   mouse_init( void );
void  mouse_start( void );
void  mouse_poll( void );



#endif
```

mouse_mm.h

```
#ifndef     MOUSE_MM_H
#define     MOUSE_MM_H

#define     MOUSE_BASE  0

#define     MOUSE_CTL   (MOUSE_BASE+0)
#define     MOUSE_X           (MOUSE_BASE+1)
#define     MOUSE_Y           (MOUSE_BASE+2)

#endif
```

packet.h

```
#ifndef     PACKET_H
#define     PACKET_H

#include "types.h"

// nios is byteswap?
#define     BYTESWAP

#ifdef BYTESWAP
#define HTONS(a) \
      ((((a) & 0xff00) >> 8) | (((a) & 0x00ff) << 8))
#define HTONL(a) \
      ((((a) & 0xff000000) >> 24)   | \
       (((a) & 0x00ff0000) >> 8)    | \
       (((a) & 0x0000ff00) << 8)    | \
       (((a) & 0x000000ff) << 24) )
#else
#define     HTONS(a)    (a)
#define     HTONL(a)    (a)
#endif

#define     NTOHS(a)    HTONS(a)
#define     NTOHL(a)    HTONL(a)

// always in HOST order
#define     IP(a,b,c,d) ( ((a)<<24) | ((b)<<16) | ((c)<<8) | ((d)<<0) )


typedef struct
{
      // ethernet
      struct
      {
            u8    daddr[6];
            u8    saddr[6];
            u16   type;

      } __attribute__((packed)) enet;

      struct
      {
            u8    vhl;        // 7-4: version 3-0: hdr len, in 32-bit words
            u8    ds;               // diff services
            u16   len;        // length of entire IP datagram, header
included

            u16   id;
            u8    flag;       // 7-5: reserved, df, mf 4-0: high frag bits
            u8    frag;

            u8    ttl;
            u8    proto;
            u16   cksum;            // header checksum only
```

```
            u32   saddr;
            u32   daddr;
      } __attribute__((packed)) ip;

      struct
      {
            u16   sport;
            u16   dport;

            u16   len;        // length of udp header and data
            u16   cksum;

      } __attribute__((packed)) udp;


      u8 data[64];

} __attribute__((packed)) packet_t;


extern packet_t txpkt;
extern packet_t rxpkt;

#endif
```

types.h

```
#ifndef     _TYPES_H_
#define     _TYPES_H_

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;

typedef short i16;
typedef int i32;


#endif
```

ui.h

```
#ifndef     UI_H
#define     UI_H

void  ui_updatescore( void );
int   ui_getplayer( void );
void  ui_waitopponent( void );
void  ui_getplayer_text( void );
void  ui_updatescore_text( void );
void  ui_winner( void );
void  ui_loser( void );


#endif
```

vga.h

```
#ifndef     VGA_H
#define     VGA_H

#define VGA_WIDTH   160
#define VGA_HEIGHT  120

#define     VGA_BASE    DE2_VGA_RASTER_INST_BASE

#define     VGA_BX                (VGA_BASE+32)
#define     VGA_BY                (VGA_BASE+34)
#define     VGA_P1A           (VGA_BASE+36)
#define     VGA_P1B           (VGA_BASE+38)
#define     VGA_P2A               (VGA_BASE+40)
#define     VGA_P2B           (VGA_BASE+42)
#define     VGA_SYNC          (VGA_BASE+44)
#define VGA_SC_COLOR    (VGA_BASE+46)
#define VGA_BG_COLOR    (VGA_BASE+48)
#define VGA_FG_COLOR    (VGA_BASE+50)
#define VGA_GO_COLOR    (VGA_BASE+52)

#define FG_RAM_BASE (0x10000 | VGA_BASE)

extern void put_vga_char(char c, unsigned int column, unsigned int row);
extern void put_vga_string(char *str, unsigned int column, unsigned int row);
extern void clr_fg ();
extern void clr_fg_sub(int start, int end);



#endif
```

xio.h

```
#ifndef     XIO_H
#define     XIO_H

#include <io.h>

#define IOWR8(base,data) \
    IOWR_8DIRECT(base,0,data)

#define IORD8(base) \
    IORD_8DIRECT(base,0)

#define IOWR16(base,data) \
    IOWR_16DIRECT(base,0,data)

#define IORD16(base) \
    IORD_16DIRECT(base,0)

#define IOWR32(base,data) \
    IOWR_32DIRECT(base,0,data)

#define IORD32(base) \
      IORD_32DIRECT(base,0)


#endif
```

top.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity top is
      port (
            CLOCK_50 : in std_logic;

            PS2_CLK : inout std_logic;
            PS2_DAT : inout std_logic;

            SRAM_ADDR : OUT STD_LOGIC_VECTOR (17 DOWNTO 0);
            SRAM_CE_N : OUT STD_LOGIC;
            SRAM_DQ : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
            SRAM_LB_N : OUT STD_LOGIC;
            SRAM_OE_N : OUT STD_LOGIC;
            SRAM_UB_N : OUT STD_LOGIC;
            SRAM_WE_N : OUT STD_LOGIC;


        VGA_CLK,
        VGA_HS,
        VGA_VS,
        VGA_BLANK,
        VGA_SYNC : out std_logic;
        VGA_R,
        VGA_G,
        VGA_B : out STD_LOGIC_VECTOR(9 downto 0);

        ENET_DATA : inout std_logic_vector(15 downto 0);
        ENET_CMD,
        ENET_CS_N,
        ENET_WR_N,
        ENET_RD_N,
        ENET_RST_N,
        ENET_CLK : out std_logic;
        ENET_INT : in std_logic
      );
end top;

architecture rtl of top is
      signal clk25 : std_logic;
begin
        process (CLOCK_50)
        begin
            if rising_edge(CLOCK_50) then
                        clk25 <= not clk25;
                  end if;
            end process;

            ENET_CLK <= clk25;

            U1 : entity work.pong port map
```

```
            (
              -- 1) global signals:
                 clk => CLOCK_50,
                 reset_n => '1',

              -- the_ps2slave_inst
                 PS2_CLK_to_and_from_the_de2_ps2_inst => PS2_CLK,
                 PS2_DAT_to_and_from_the_de2_ps2_inst => PS2_DAT,


              -- the_sram_inst
                 SRAM_ADDR_from_the_sram_inst => SRAM_ADDR,
                 SRAM_CE_N_from_the_sram_inst => SRAM_CE_N,
                 SRAM_DQ_to_and_from_the_sram_inst => SRAM_DQ,
                 SRAM_LB_N_from_the_sram_inst => SRAM_LB_N,
                 SRAM_OE_N_from_the_sram_inst => SRAM_OE_N,
                 SRAM_UB_N_from_the_sram_inst => SRAM_UB_N,
                 SRAM_WE_N_from_the_sram_inst => SRAM_WE_N,


              -- the_de2_vga_raster_inst
                 CLOCK_50_to_the_de2_vga_raster_inst => CLOCK_50,
                 VGA_BLANK_from_the_de2_vga_raster_inst => VGA_BLANK,
                 VGA_B_from_the_de2_vga_raster_inst => VGA_B,
                 VGA_CLK_from_the_de2_vga_raster_inst => VGA_CLK,
                 VGA_G_from_the_de2_vga_raster_inst => VGA_G,
                 VGA_HS_from_the_de2_vga_raster_inst => VGA_HS,
                 VGA_R_from_the_de2_vga_raster_inst => VGA_R,
                 VGA_SYNC_from_the_de2_vga_raster_inst => VGA_SYNC,
                 VGA_VS_from_the_de2_vga_raster_inst => VGA_VS,


              -- the_de2_enet_inst

                 ENET_CMD_from_the_de2_enet_inst => ENET_CMD,
                 ENET_CS_N_from_the_de2_enet_inst => ENET_CS_N,
                 ENET_DATA_to_and_from_the_de2_enet_inst => ENET_DATA,
                 ENET_INT_to_the_de2_enet_inst => ENET_INT,
                 ENET_RD_N_from_the_de2_enet_inst => ENET_RD_N,
                 ENET_RST_N_from_the_de2_enet_inst => ENET_RST_N,
                 ENET_WR_N_from_the_de2_enet_inst => ENET_WR_N
            );

end rtl;
```

de2_vga_raster.vhd

```
--
-- circle generator
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity pong_circle is
      port (
            x,y                     : in unsigned (9 downto 0);
            cx,cy             : in unsigned (9 downto 0);
            do                      : out std_logic
      );
end pong_circle;

architecture rtl of pong_circle is
      type bitmap_t is array(0 to 14,0 to 14) of std_logic;
      constant bitmap : bitmap_t :=
        ( "000001111100000",
          "000111111111000",
        "001111111111100",
        "011111111111110",
        "011111111111110",
        "111111111111111",
        "111111111111111",
        "111111111111111",
        "111111111111111",
        "111111111111111",
        "011111111111110",
        "011111111111110",
        "001111111111100",
        "000111111111000",
        "000001111100000" );

    -- offset variables
      signal ix,iy : unsigned(9 downto 0);

      -- valid, in range of bitmap
      signal vx, vy : std_logic;
begin
      ix <= (x - (cx-7));
      iy <= (y - (cy-7));

      vx <= '1' when (x >= (cx-7) and x <= (cx+7)) else '0';
      vy <= '1' when (y >= (cy-7) and y <= (cy+7)) else '0';

      do <= vx and vy and bitmap(TO_INTEGER(ix),TO_INTEGER(iy));
end rtl;


--
-- paddle generator
```

```
--



library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity pong_paddle is
      port (
            x,y,cx                  : in unsigned (9 downto 0);
            a,b                     : in unsigned (9 downto 0);
            do                      : out std_logic
      );
end pong_paddle;

architecture rtl of pong_paddle is
  signal vx, vy : std_logic;
  signal vis : std_logic;
begin
  vx <= '1' when (x >= cx and x<(cx+5)) else '0';
  vy <= '1' when (y >= a and y <= b) else '0';

  vis <= '0' when (a = b) else '1';


  do <= vx and vy and vis;
end rtl;




--
-- Game Objects Generator
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gobj_gen is
      port (
            --x,y             : in unsigned (9 downto 0);
            visx, visy  : in unsigned (9 downto 0);
            ballx,
            bally,
            p1a,
            p1b,
            p2a,
            p2b               : in unsigned (9 downto 0);

            do                : out std_logic
            );

end gobj_gen;
```

```
architecture rtl of gobj_gen is
-- signals
  signal active_pixel,circle_pixel,padl1_pixel,padl2_pixel : std_logic;

begin


U1: entity work.pong_circle port map (
    x => visx,
    y => visy,
    cx => ballx, --"0001010000",
    cy => bally, -- "0001010000",
    do => circle_pixel
  );

  U2: entity work.pong_paddle port map (
    x => visx,
    y => visy,
    a => p1a, --"0011110000",
    b => p1b, --"0011111111",
    cx => "0000010100", -- 20 pixel offset for left paddle
    do => padl1_pixel
  );

  U3: entity work.pong_paddle port map (
    x => visx,
    y => visy,
    a => p2a, --"0000001000",
    b => p2b, --"0000110000",
    cx => "1001100111", -- VGA_X-(PADL_OFFSET+PADL_WIDTH)=640-20-5, give or
take
    do => padl2_pixel
  );

  active_pixel <= circle_pixel or padl1_pixel or padl2_pixel; -- or bg_pixel;
  do <= active_pixel;


end rtl;




--1001101111
-- Background Generator
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bg_gen is
      port (
            x,y         : in unsigned (9 downto 0);
            --tx,ty     : in unsigned (9 downto 0);
            tile_map : in unsigned (255 downto 0);
```

```
            do          : out std_logic
            );
end bg_gen;

architecture rtl of bg_gen is

      --type bitmap_t is array(0 to 15,0 to 15) of std_logic;

      type bitmap_t is array (15 downto 0) of unsigned (15 downto 0);
      signal bitmap : bitmap_t;


            signal vx, vy : std_logic;
            signal bit_x,bit_y : unsigned(9 downto 0);
            signal current_line : unsigned (15 downto 0);
      --    signal count_x, count_y : unsigned (3 downto 0);

      begin

            bitmap (0) <= tile_map (255 downto 240);
            bitmap (1) <= tile_map (239 downto 224);
            bitmap (2) <= tile_map (223 downto 208);
            bitmap (3) <= tile_map (207 downto 192);
            bitmap (4) <= tile_map (191 downto 176);
            bitmap (5) <= tile_map (175 downto 160);
            bitmap (6) <= tile_map (159 downto 144);
            bitmap (7) <= tile_map (143 downto 128);
            bitmap (8) <= tile_map (127 downto 112);
            bitmap (9) <= tile_map (111 downto 96);
            bitmap (10) <= tile_map (95 downto 80);
            bitmap (11) <= tile_map (79 downto 64);
            bitmap (12) <= tile_map (63 downto 48);
            bitmap (13) <= tile_map (47 downto 32);
            bitmap (14) <= tile_map (31 downto 16);
            bitmap (15) <= tile_map (15 downto 0);

            current_line <= bitmap(TO_INTEGER(x(3 downto 0)));
            do <= '1' when current_line(TO_INTEGER(y(3 downto 0))) = '1' else
'0';

--    do <= '1' when bitmap(TO_INTEGER(x(3 downto 0)),TO_INTEGER(y(3 downto
0)))='1' else '0';

end rtl;




library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


--
-- Foreground Generator
--
```

```
entity fg_gen is
      port (
                  clk               : in std_logic;
                  visx, visy  : in unsigned (9 downto 0);
                  rd_address  : out unsigned (14 downto 0);
                  data_in           : in unsigned (15 downto 0);
                  do                : out std_logic
            );

end fg_gen;

architecture rtl of fg_gen is


      --signal byte_pos : unsigned (14 downto 0);
      signal bit_pos : unsigned (3 downto 0);

      -- new stuff

      signal data : unsigned (14 downto 0);
      --signal data_long : unsigned (19 downto 0);

      signal x,y : unsigned (14 downto 0);


      begin

      -- convert x,y to byte offset
      -- Position is calculated as => 80 * (visy) + (visx/8);

      --data_long <= (40 * visy) + (visx/16);

      x <= "000000000000000" + visx(9 downto 1);
      y <= "000000000000000" + visy(9 downto 1);

      data <= "000000000000000" + (((y sll 4) + (y sll 2)) + (x srl 4));

      --data <= data_long (14 downto 0);

      rd_address <= data;
      bit_pos <= x(3 downto 0); -- lower 4 bits of visx


      -- update on the clock

--    process (clk)
--          begin
--                if rising_edge(clk) then
--                      if data_in(to_integer(bit_pos)) = '1' then
--                            do <= '1';
--                      else
--                            do <= '0';
--                      end if;
--                end if;
--          end process;

      do <= '1' when data_in(TO_INTEGER(bit_pos)) = '1' else '0';
```

```
end rtl;



--
-- Priority MUX
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pmux is
port (
            en0, en1, en2                       : in std_logic;
            -- incoming colors
            sc_color,
            bg_color,
            fg_color,
            go_color                            : in unsigned (15 downto 0);
            vga_r, vga_g, vga_b                 : out unsigned (9 downto 0);
            clk                                       : in std_logic
            );
end pmux;

architecture rtl of pmux is
-- signals
signal red, green, blue       : unsigned (9 downto 0);

signal
      sc_r, bg_r, fg_r, go_r,
      sc_g, bg_g, fg_g, go_g,
      sc_b, bg_b, fg_b, go_b  : unsigned (9 downto 0);

begin

-- Screen Colors
sc_r <= sc_color(14 downto 10) & "11111" when sc_color(15) = '1'
            else sc_color(14 downto 10) & "00000";
sc_g <= sc_color(9 downto 5) & "11111" when sc_color(15) = '1'
            else sc_color(9 downto 5) & "00000";
sc_b <= sc_color(4 downto 0) & "11111" when sc_color(15) = '1'
            else sc_color(4 downto 0) & "00000";

-- Background Colors
bg_r <= bg_color(14 downto 10) & "11111" when bg_color(15) = '1'
            else bg_color(14 downto 10) & "00000";
bg_g <= bg_color(9 downto 5) & "11111" when bg_color(15) = '1'
            else bg_color(9 downto 5) & "00000";
bg_b <= bg_color(4 downto 0) & "11111" when bg_color(15) = '1'
            else bg_color(4 downto 0) & "00000";

-- Foreground Colors
fg_r <= fg_color(14 downto 10) & "11111" when fg_color(15) = '1'
            else fg_color(14 downto 10) & "00000";
```

```
fg_g <= fg_color(9 downto 5) & "11111" when fg_color(15) = '1'
            else fg_color(9 downto 5) & "00000";
fg_b <= fg_color(4 downto 0) & "11111" when fg_color(15) = '1'
            else fg_color(4 downto 0) & "00000";

-- Game Object Colors
go_r <= go_color(14 downto 10) & "11111" when go_color(15) = '1'
            else go_color(14 downto 10) & "00000";
go_g <= go_color(9 downto 5) & "11111" when go_color(15) = '1'
            else go_color(9 downto 5) & "00000";
go_b <= go_color(4 downto 0) & "11111" when go_color(15) = '1'
            else go_color(4 downto 0) & "00000";



red <=      go_r when en0 = '1' else -- game objects
            fg_r when en1 = '1' else -- foreground
            bg_r when en2 = '1' else -- background
            sc_r;

green <= go_g when en0 = '1' else
            fg_g when en1 = '1' else
            bg_g when en2 = '1' else
            sc_g;

blue <=  go_b when en0 = '1' else
            fg_b when en1 = '1' else
            bg_b when en2 = '1' else
            sc_b;

      process (clk) begin
            if rising_edge(clk) then
                  vga_r <= red;
                  vga_g <= green;
                  vga_b <= blue;
            end if;
      end process;

end rtl;


--------------------------------------------------------------------------------
--
--
-- Simple VGA raster display
--
-- Stephen A. Edwards
-- sedwards@cs.columbia.edu
--
--------------------------------------------------------------------------------
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_vga_raster is
```

```
  port (
    reset : in std_logic;
    clk   : in std_logic;                    -- Should be 25.125 MHz

    -- circle position
    cx : in unsigned(9 downto 0);
    cy : in unsigned(9 downto 0);

      visx, visy : out unsigned(9 downto 0); -- visible screen x and y
position
      pgen_vga_r, pgen_vga_g, pgen_vga_b : in unsigned (9 downto 0);

      vbi : out std_logic;               -- anywhere in vertical blank
interval

    -- vga interface

    VGA_CLK,                           -- Clock
    VGA_HS,                            -- H_SYNC
    VGA_VS,                            -- V_SYNC
    VGA_BLANK,                         -- BLANK
    VGA_SYNC : out std_logic;          -- SYNC
    VGA_R,                             -- Red[9:0]
    VGA_G,                             -- Green[9:0]
    VGA_B : out unsigned(9 downto 0) -- Blue[9:0]
    );

end de2_vga_raster;

architecture rtl of de2_vga_raster is

  -- Video parameters

  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
  constant VSYNC        : integer := 2;
  constant VBACK_PORCH  : integer := 33;
  constant VACTIVE      : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  constant RECTANGLE_HSTART : integer := 100;
  constant RECTANGLE_HEND   : integer := 540;
  constant RECTANGLE_VSTART : integer := 100;
  constant RECTANGLE_VEND   : integer := 380;

  -- Signals for the video controller
  signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
  signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
  signal EndOfLine, EndOfField : std_logic;
  signal visx_sig, visy_sig : unsigned(9 downto 0); -- visible screen x and y
position
```

```
  signal vga_hblank, vga_hsync,
    vga_vblank, vga_vsync : std_logic;  -- Sync. signals

  signal active_pixel,circle_pixel,padl1_pixel,padl2_pixel : std_logic;

  --signal pgen_vga_r, pgen_vga_g, pgen_vga_b : unsigned (4 downto 0); -- 5-
bit color output from pixel gen

  signal bg_pixel : std_logic;

begin

  -- Horizontal and vertical counters
  visx_sig <= b"0000000000" when Hcount<(HSYNC+HBACK_PORCH) else (Hcount -
(HSYNC+HBACK_PORCH));
  visy_sig <= b"0000000000" when Vcount<(VSYNC + VBACK_PORCH+1) else (Vcount
- (VSYNC+VBACK_PORCH+1));

  visx <= visx_sig;
  visy <= visy_sig;



  HCounter : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        Hcount <= (others => '0');
      elsif EndOfLine = '1' then
        Hcount <= (others => '0');
      else
        Hcount <= Hcount + 1;
      end if;
    end if;
  end process HCounter;

  EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

  VCounter: process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        Vcount <= (others => '0');
      elsif EndOfLine = '1' then
        if EndOfField = '1' then
          Vcount <= (others => '0');
        else
          Vcount <= Vcount + 1;
        end if;
      end if;
    end if;
  end process VCounter;

  EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

  -- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK
```

```
  HSyncGen : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' or EndOfLine = '1' then
        vga_hsync <= '1';
      elsif Hcount = HSYNC - 1 then
        vga_hsync <= '0';
      end if;
    end if;
  end process HSyncGen;

  HBlankGen : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        vga_hblank <= '1';
      elsif Hcount = HSYNC + HBACK_PORCH then
        vga_hblank <= '0';
      elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
        vga_hblank <= '1';
      end if;
    end if;
  end process HBlankGen;

  VSyncGen : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        vga_vsync <= '1';
      elsif EndOfLine ='1' then
        if EndOfField = '1' then
          vga_vsync <= '1';
        elsif Vcount = VSYNC - 1 then
          vga_vsync <= '0';
        end if;
      end if;
    end if;
  end process VSyncGen;

  VBlankGen : process (clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        vga_vblank <= '1';
      elsif EndOfLine = '1' then
        if Vcount = VSYNC + VBACK_PORCH - 1 then
          vga_vblank <= '0';
        elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
          vga_vblank <= '1';
        end if;
      end if;
    end if;
  end process VBlankGen;

-- for testing purposes
active_pixel <= '1';
```

```
  -- Registered video signals going to the video DAC

  VideoOut: process (clk, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk'event and clk = '1' then
      if (vga_hblank='1' or vga_vblank='1') then
        VGA_R <= "0000000000";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      elsif (visx_sig <= 640) and (visy_sig <= 480) then
        VGA_R <= pgen_vga_r;
        VGA_G <= pgen_vga_g;
        VGA_B <= pgen_vga_b;
      else
        VGA_R <= "1111111111";
        VGA_G <= "0000000000";
        VGA_B <= "0000000000";
      end if;
    end if;
  end process VideoOut;

  VGA_CLK <= clk;
  VGA_HS <= not vga_hsync;
  VGA_VS <= not vga_vsync;
  VGA_SYNC <= '0';
  VGA_BLANK <= not (vga_hsync or vga_vsync);

  vbi <= not vga_vblank;

end rtl;



library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity vga_top is

  port (
    -- avalon interface
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  unsigned(15 downto 0);
    readdata   : out unsigned(15 downto 0);
    writedata  : in  unsigned(15 downto 0);

    -- vga interface

    CLOCK_50 : std_logic;
```

```
    VGA_CLK,                                                -- Clock
    VGA_HS,                                                 -- H_SYNC
    VGA_VS,                                                 -- V_SYNC
    VGA_BLANK,                                              -- BLANK
    VGA_SYNC : out std_logic;                               -- SYNC
    VGA_R,                                                  -- Red[9:0]
    VGA_G,                                                  -- Green[9:0]
    VGA_B : out unsigned(9 downto 0)                        -- Blue[9:0]


    );

end vga_top;

architecture rtl of vga_top is

  type ram_type is array(0 to 26) of unsigned(15 downto 0);
  signal RAM : ram_type;
  signal ram_address : unsigned(15 downto 0);
  signal clk25 : std_logic;
  signal vbi : std_logic;

  signal gobj_en, fg_en, bg_en : std_logic; -- enable signals for each layer
  signal pgen_vga_r, pgen_vga_g, pgen_vga_b : unsigned (9 downto 0);
  signal tile_map : unsigned (255 downto 0);

  signal
            ballx,
            bally,
            p1a,
            p1b,
            p2a,
            p2b   : unsigned (9 downto 0);

-- color signals
  signal
            sc_color,
            bg_color,
            fg_color,
            go_color : unsigned (15 downto 0);

  signal visx, visy : unsigned(9 downto 0); -- visible screen x and y
position

      -- VGA memory signals

      -- into memory
      signal address_avalon, address_vga : std_logic_vector (14 downto 0);
      signal write_avalon : std_logic; -- write signal for VGA hardwired to 0
since always read
      signal write_data : unsigned (15 downto 0);
      signal rd_address : unsigned (14 downto 0);

      -- from memory
      signal readdata_avalon, readdata_vga : unsigned (15 downto 0);
```

```
      signal mem_cs : std_logic;


      --test stuff
      signal clk16  : std_logic; -- slow clock for VGA memory


      component ram_fg is
            port (

                  -- avalon side
                  clk         : in std_logic; -- dual clocked
                  addr        : in unsigned (14 downto 0);
                  cs                : in std_logic; -- chipselect
                  w_en        : in std_logic; -- write enable
                  din               : in unsigned (15 downto 0);
                  dout        : out unsigned (15 downto 0);

                  -- vga side
                  clk2        : in std_logic; -- dual clocked
                  addr2       : in unsigned (14 downto 0);
                  dout2       : out unsigned (15 downto 0)

                  );
end component;


begin

  --
  -- avalon interface
  --

  -- memory map: BASE:xpos BASE+2:ypos BASE+4:blank

  ram_address <= address(15 downto 0);
  write_avalon <= (write and address(15));

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        readdata <= (others => '0');
      else
        if chipselect = '1' then
          if read = '1' then
                  if ( address(15) = '0' ) then
                        RAM(22) <= b"000000000000000" & vbi;
                        readdata <= RAM(to_integer(ram_address(4 downto 0)));
                        else
                        readdata <= UNSIGNED(readdata_avalon);
              end if;
              elsif write = '1' then
                        if address(15) = '0' then
                        RAM(to_integer(ram_address)) <= writedata;
                        else
                        write_data <= writedata;
```

```
                        end if;
              end if;
       end if;

       end if;
   end if;
 end process;


 --
 -- vga raster logic
 --

 process (CLOCK_50)
 begin
   if rising_edge(CLOCK_50) then
     clk25 <= not clk25;
   end if;
 end process;

process (clk25)
     begin
           if rising_edge(clk25) then
                 clk16 <= visx(4);
           end if;
     end process;

 V1: entity work.de2_vga_raster port map (
   reset => '0',
   clk => clk25,

     visx => visx,
     visy => visy,
     pgen_vga_r => pgen_vga_r,
     pgen_vga_g => pgen_vga_g,
     pgen_vga_b => pgen_vga_b,
     vbi => vbi,

   cx => RAM(0)(9 downto 0),
   cy => RAM(1)(9 downto 0),

   VGA_CLK => VGA_CLK,
   VGA_HS => VGA_HS,
     VGA_VS => VGA_VS,
   VGA_BLANK => VGA_BLANK,
   VGA_SYNC => VGA_SYNC,
   VGA_R => VGA_R,
   VGA_G => VGA_G,
   VGA_B => VGA_B
 );

           tile_map (255 downto 240) <= RAM(0);
           tile_map (239 downto 224) <= RAM(1);
           tile_map (223 downto 208) <= RAM(2);
           tile_map (207 downto 192) <= RAM(3);
           tile_map (191 downto 176) <= RAM(4);
           tile_map (175 downto 160) <= RAM(5);
```

```
tile_map (159 downto 144) <= RAM(6);
tile_map (143 downto 128) <= RAM(7);
tile_map (127 downto 112) <= RAM(8);
tile_map (111 downto 96) <= RAM(9);
tile_map (95 downto 80) <= RAM(10);
tile_map (79 downto 64) <= RAM(11);
tile_map (63 downto 48) <= RAM(12);
tile_map (47 downto 32) <= RAM(13);
tile_map (31 downto 16) <= RAM(14);
tile_map (15 downto 0) <= RAM(15);


ballx <= RAM(16)(9 downto 0);
bally <= RAM(17)(9 downto 0);
p1a <= RAM(18)(9 downto 0);
p1b <= RAM(19)(9 downto 0);
p2a <= RAM(20)(9 downto 0);
p2b   <= RAM(21)(9 downto 0);

sc_color <= RAM(23);
bg_color <= RAM(24);
fg_color <= RAM(25);
go_color <= RAM(26);


U1: entity work.gobj_gen port map (
      visx => visx,
      visy => visy,
      ballx => ballx,
      bally => bally,
      p1a => p1a,
      p1b => p1b,
      p2a => p2a,
      p2b   => p2b,
      do => gobj_en
);


U2: entity work.bg_gen port map (
      x => visx,
      y => visy,
      tile_map => tile_map,
      do => bg_en
);


U3: entity work.fg_gen port map(
      clk => clk25,
      visx => visx,
      visy => visy,
      rd_address => rd_address,
      data_in => readdata_vga,
      do => fg_en
      );


U4: entity work.pmux port map (
```

```
                clk => clk25,
                en0 => gobj_en,
                en1 => fg_en,
                en2 => bg_en,
                sc_color => sc_color,
                bg_color => bg_color,
                fg_color => fg_color,
                go_color => go_color,
                vga_r => pgen_vga_r,
                vga_g => pgen_vga_g,
                vga_b => pgen_vga_b
                );

                mem_cs <= std_logic(address(15)) and chipselect; -- used in
ram_fg

        U5: ram_fg port map (

                -- avalon side
                clk => clk,
                addr => address(14 downto 0),
                cs => mem_cs,
                w_en => write_avalon,
                din => write_data,
                dout => readdata_avalon,

                -- vga side
                clk2 => clk,
                addr2 => rd_address,
                dout2 => readdata_vga

                );

end rtl;
```

ram_fg.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


--
-- 307,200 bit memory to store foreground bitmap
-- Stored as 19,200 16-bit half-words
--

entity ram_fg is
port (

      -- avalon side
      clk         : in std_logic; -- dual clocked
      addr        : in unsigned (14 downto 0);
      cs                : in std_logic; -- chipselect
      w_en        : in std_logic; -- write enable
      din               : in unsigned (15 downto 0);
      dout        : out unsigned (15 downto 0);

      --vga side
      clk2        : in std_logic; -- dual clocked
      addr2       : in unsigned (14 downto 0);
      dout2       : out unsigned (15 downto 0)

      );
end ram_fg;


architecture rtl of ram_fg is

      type ram_type is array (0 to 4800) of unsigned (15 downto 0);
      signal RAM : ram_type;

      begin

            process (clk) begin
                  if rising_edge(clk) then
                        if w_en = '1' then RAM(to_integer(addr)) <= din;
                                    dout <= din;
                        else dout <= RAM(to_integer(addr));
                        end if;
                  end if;
            end process;


            process (clk2) begin
                  if rising_edge(clk2) then
                         dout2 <= RAM(to_integer(addr2));
                        end if;
            end process;

      end rtl;
```

enet/dm9000a.v

```
module DM9000A_IF(      //    HOST Side
                              iDATA,oDATA,iCMD,
                              iRD_N,iWR_N,
                              iCS_N,//iRST_N,
                              oINT,
                              //    DM9000A Side
                              ENET_DATA,ENET_CMD,
                              ENET_RD_N,ENET_WR_N,
                              ENET_CS_N,ENET_RST_N,
                              ENET_INT
                              );
//    HOST Side
input [15:0]      iDATA;
input             iCMD;
input             iRD_N;
input             iWR_N;
input             iCS_N;
//input                 iRST_N;
output      [15:0]      oDATA;
output                  oINT;
//    DM9000A Side
inout [15:0]      ENET_DATA;
output                  ENET_CMD;
output                  ENET_RD_N;
output                  ENET_WR_N;
output                  ENET_CS_N;
output                  ENET_RST_N;
input             ENET_INT;

assign      ENET_DATA   =     ENET_WR_N   ?     16'hzzzz    :     iDATA ;
assign      oDATA       =     ENET_DATA   ;

assign      ENET_CMD    =     iCMD;
assign      ENET_RD_N   =     iRD_N;
assign      ENET_WR_N   =     iWR_N;
assign      ENET_CS_N   =     iCS_N;
assign      ENET_RST_N  =     1;
assign      oINT        =     ENET_INT;

endmodule
```