_ONELINER

Platform for massive, character-based
communication using VGA monitors.

Arjan Scherpenisse, 2009.

TECHNICAL DOCUMENTATION

Graduation work Gerrit Rietveld Academie (DOGTIME)
Unstable Media / Interaction Design department. Amsterdam, July 2009.

System overview

`_oneliner` consists of 64 monitors and 9 boxes with custom electronics. 8 of these boxes are the so-called *VGA boards*, electronic circuits which can control 8 vga monitors at a time. The remaining box contains an *Arduino* board with a shield on which extra electronics are placed: an LED, 3 switches, and 2 EEPROM memory modules.

Every VGA monitor is capable of displaying a single character. Every monitor is run by a microcontroller which generates a VGA signal for a specific character.

The Arduino is the *controller* of the installation: it continuously decides which monitor in the row displays which character, thus creating animations. There are 8 possible "programs" which can be run. The program number can be set using the 3 switches on the arduino shield, and pressing the reset button on the arduino. The LED then blinks as an indicator that it is running.

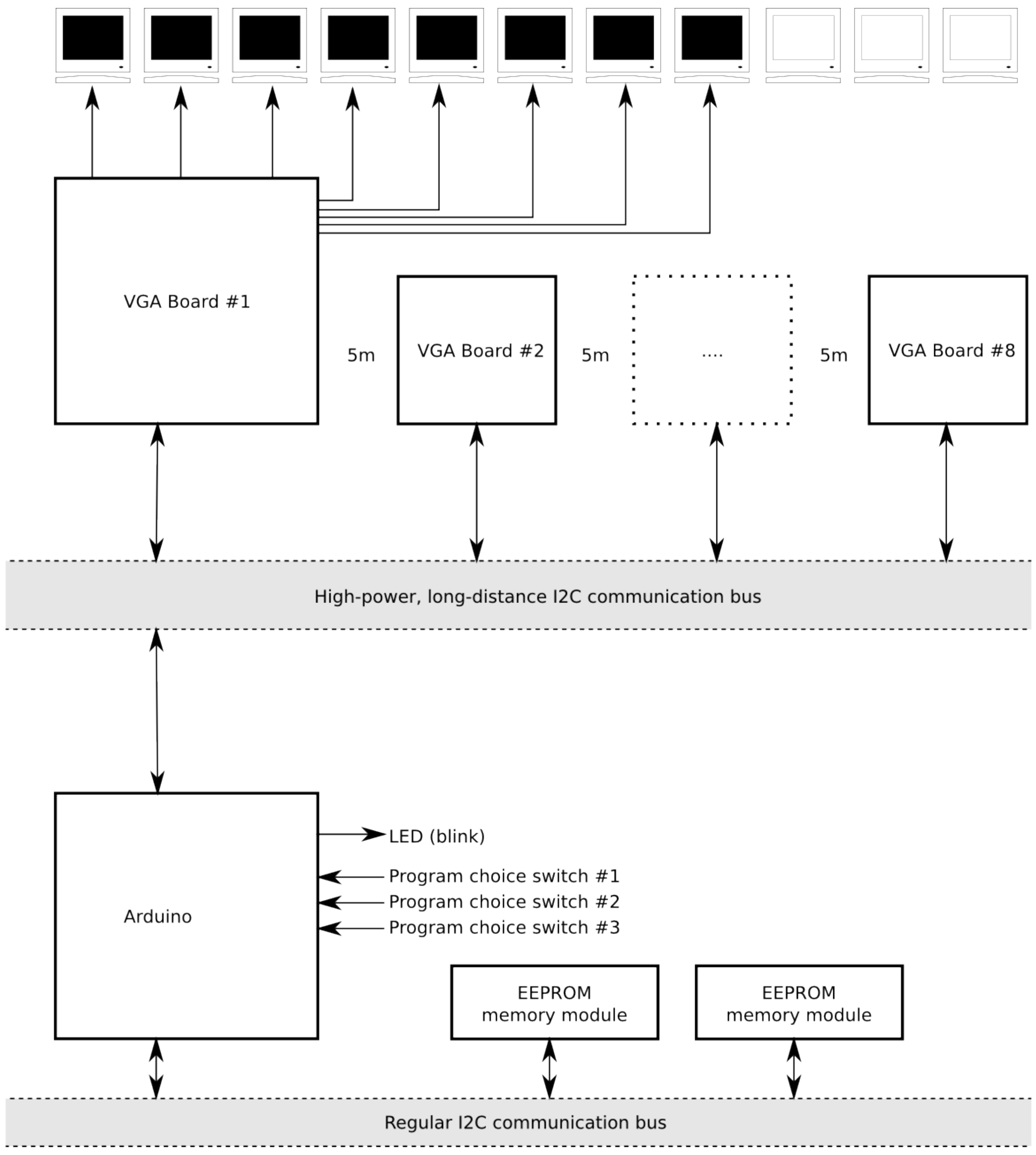

Figure 1: `_oneliner` *Overview*

Hardware: VGA board schematics

The *VGA Board* is a self-designed electronic circuit which occurs 8 times in `_oneliner`. It consists of 8 microcontrollers of type PIC16F627, each of which drives a single VGA monitor. Which character is displayed, is decided using a 7bit *latch* (the `DATA` bus in the schematic) which is tied to an I2C communication chip.

To be able to communicate over the entire 30 meters of the installation, the I2C signal needed to be amplified using P82B715 bus extenders.

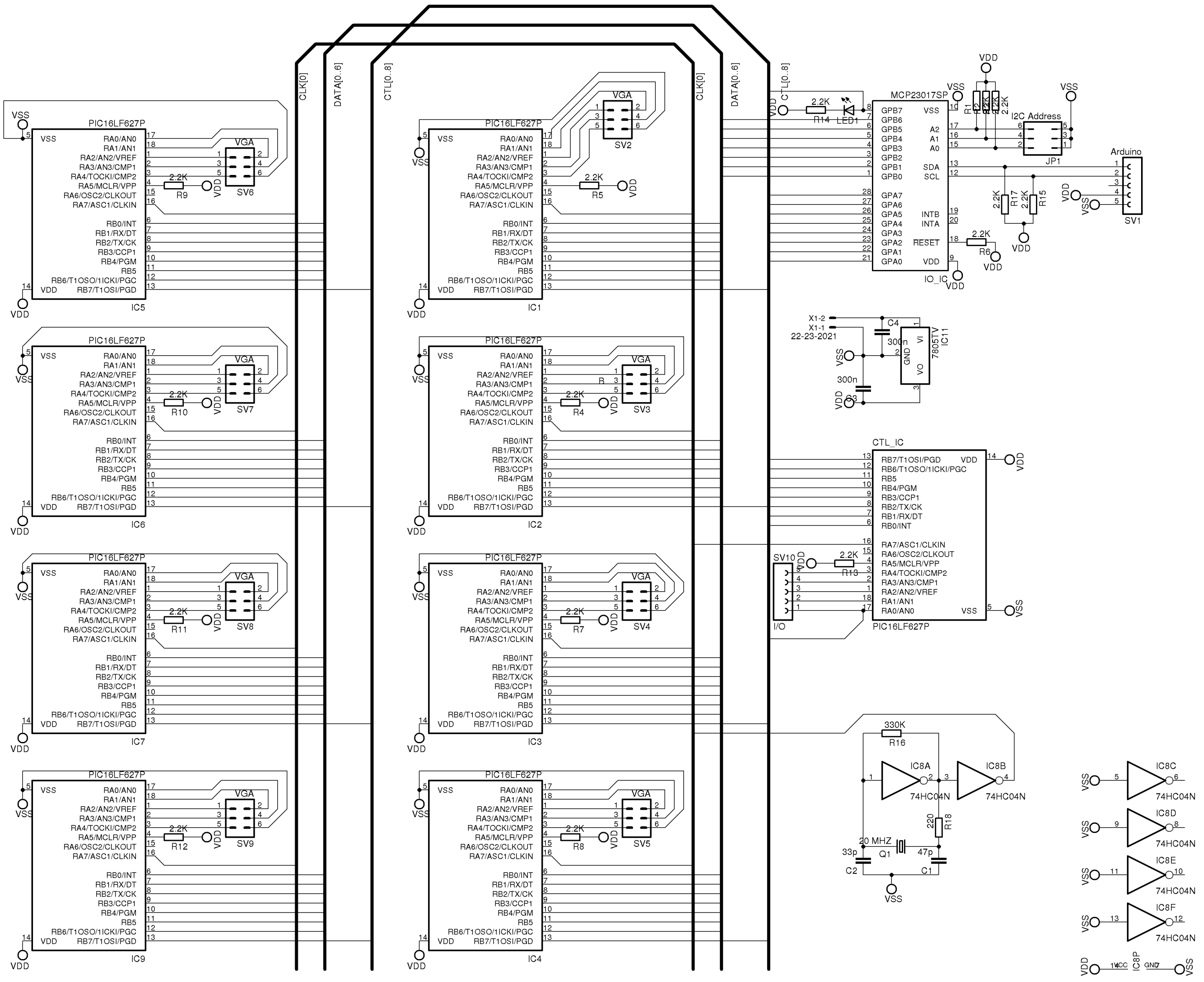

Figure 2: *VGA Board overview*

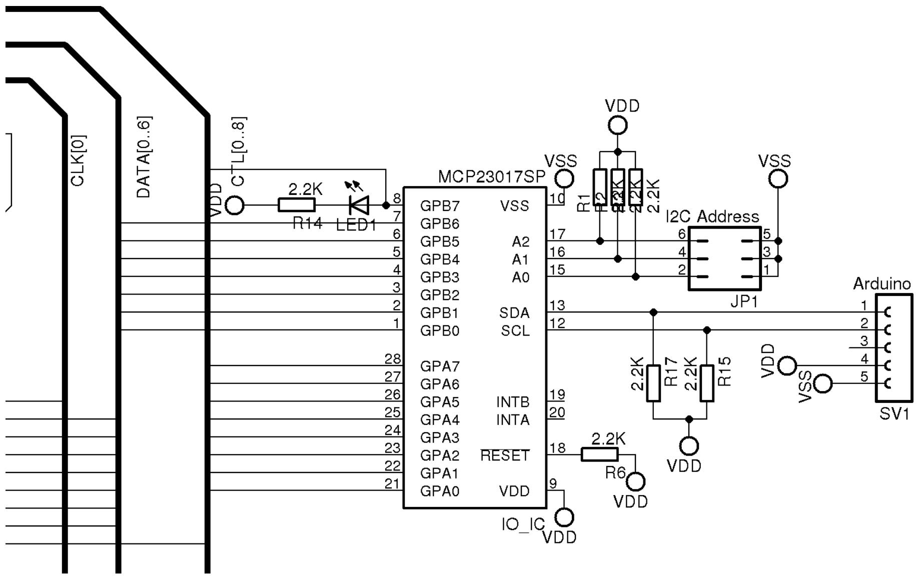

Figure 3: *I^2C Communication chip (MCP23017) with latches to DATA and CTL bus*

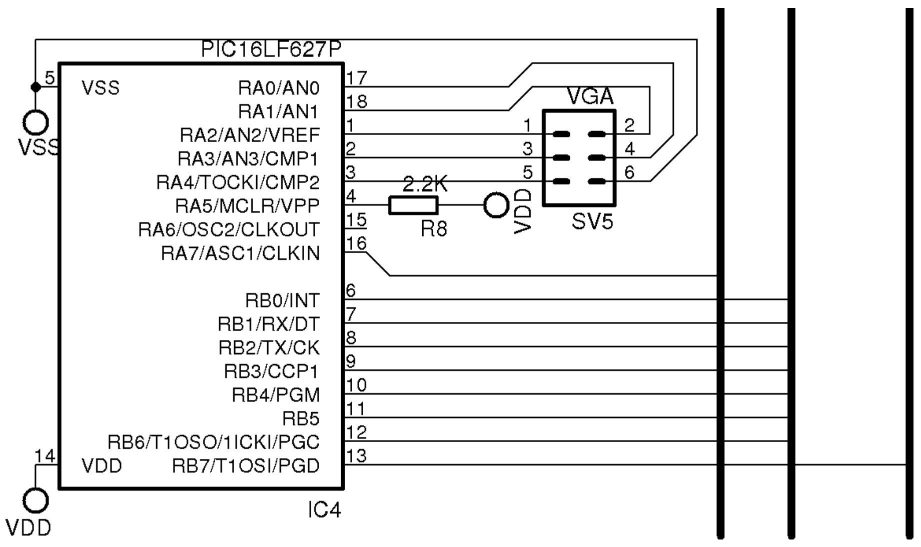

Figure 4: *Microchip control (8x PIC16F627)*

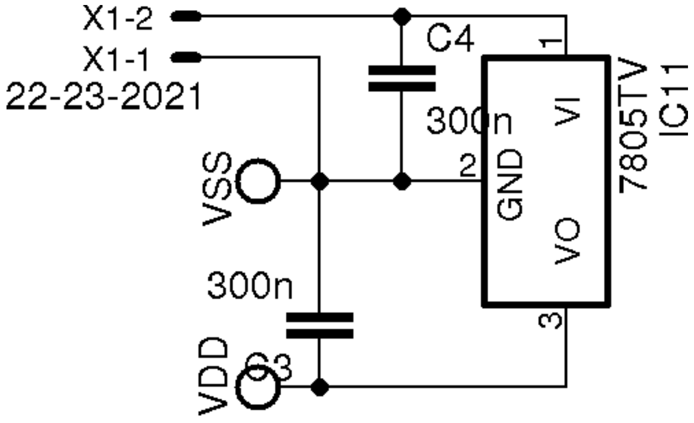

Figure 5: *Power circuit*

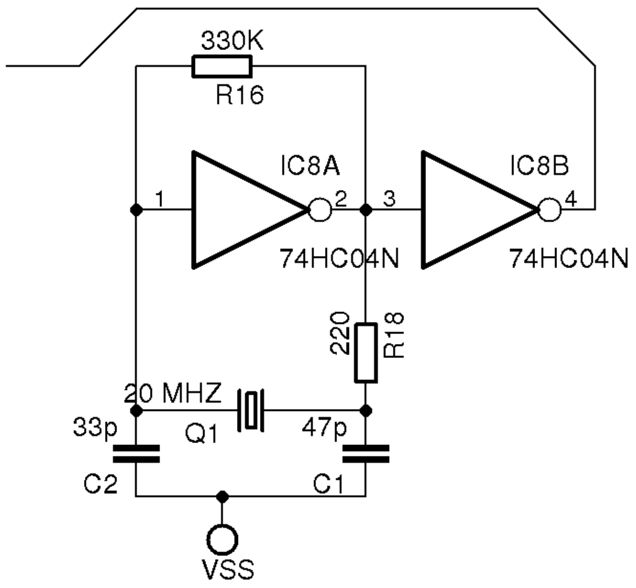

Figure 6: *Clock circuit*

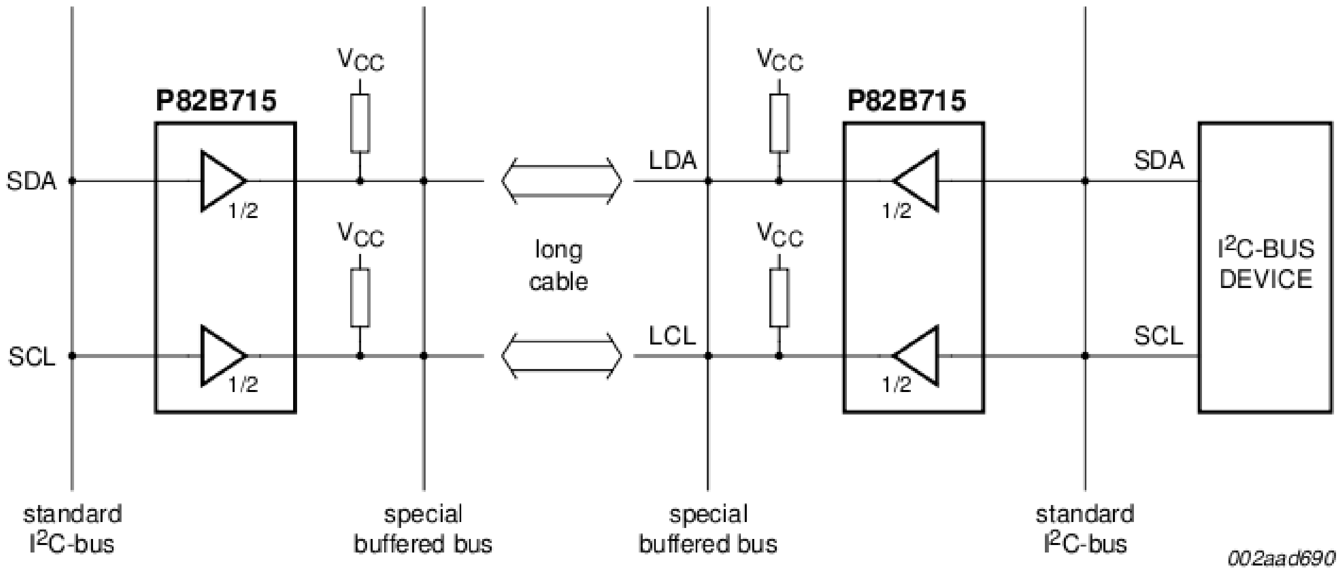

Figure 7: *High-powered I2C bus (using P82B715 ICs)*

Hardware: VGA circuit board

The board design was created using *Eagle* from the electronic schematic on the previuos pages. It was manufactured by Eurocircuits, a PCB manufacturing company from Belgium. The size of the board is 8cm x 10cm, which is exactly the maximum allowable size which can be created with the free evaluation version of Eagle.

The board designs are shown here on a 1:1 scale.

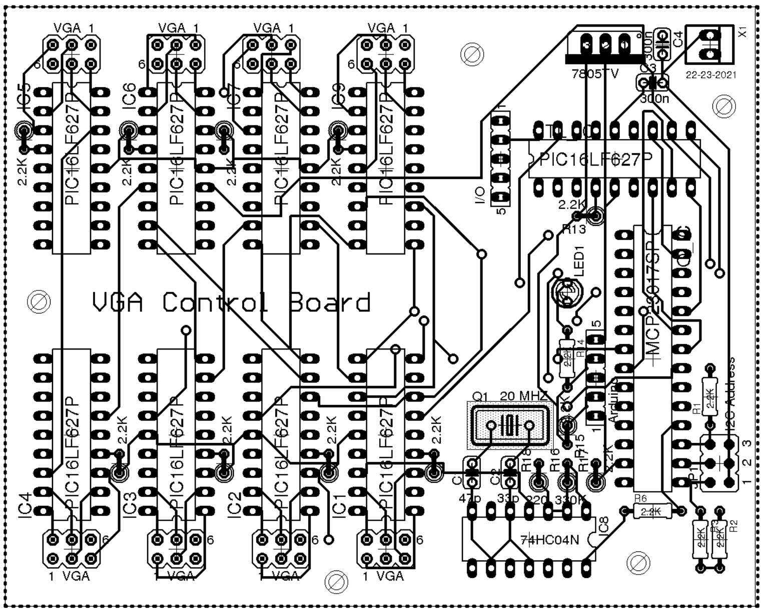

Figure 8: *Board design (front side)*

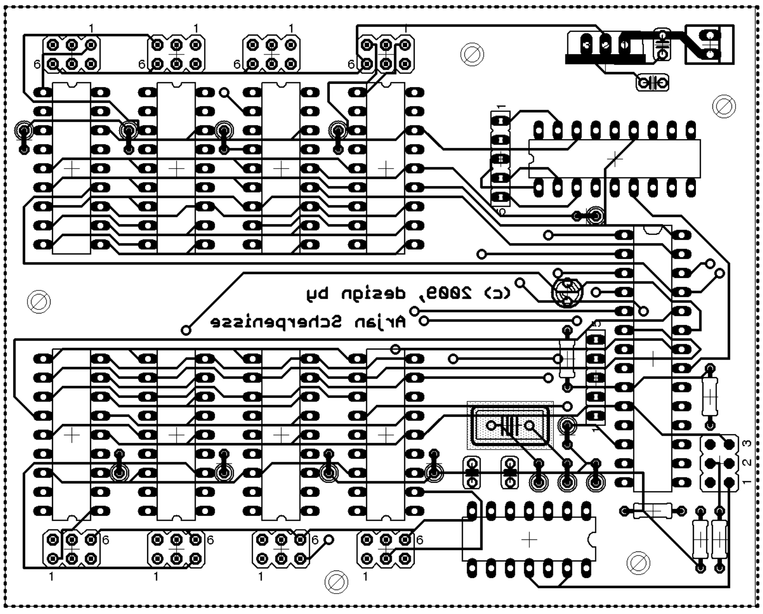

Figure 9: *Board design (back side)*

Software: the fontset

The fontset was designed and implemented in a custom ascii file format. Utilities were then written to read this format, generate pictures of the fonts (as seen below), generate assembly code for usage on the PIC microchips, and to generate a `switch` statement for use on the controller.

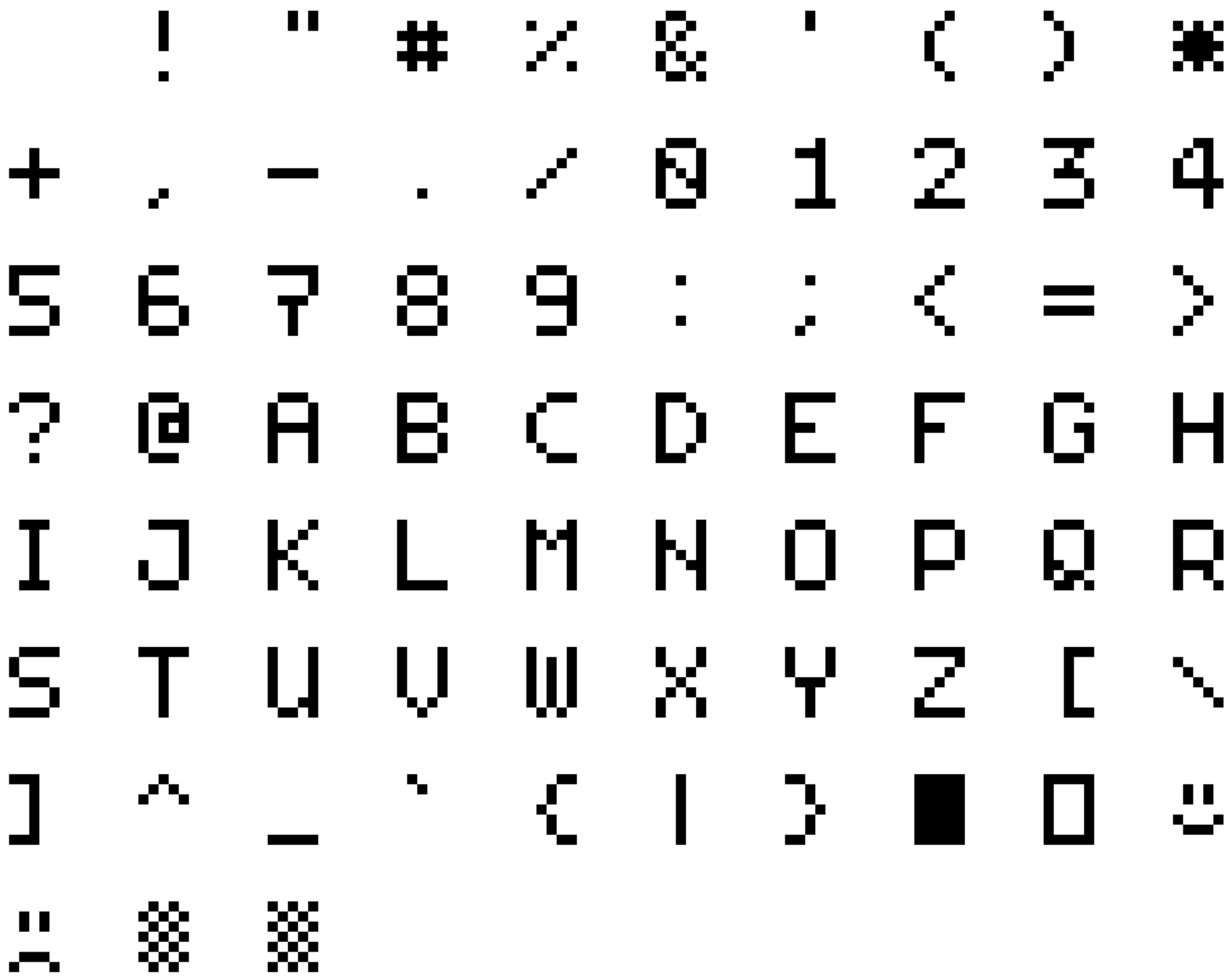

Figure 10: *The entire fontset*

Software: controller code

The controller code runs on an *Arduino* board: a board with a Atmel-based microchip which is very easily accessible and *open hardware*. The Arduino has been extended with a shield on which 2 EEPROM chips are placed which stores the texts and software which is displayed on the screens of `_oneliner`.

The following code is uploaded onto the Arduino two times. First in compiled form, because the arduino needs to execute this code to execute the `_oneliner` installation. Secondly the code is transformed into uppercase and then uploaded onto the EEPROM memory-chip on the shield of the arduino: during the running of the installation, the program will read these texts and display them on the 64 screens of the `_oneliner`.

```
// Start with the standard includes
#include <stdio.h>    // is this really needed?
```

```
#include <stdlib.h>   // for malloc()
#include <string.h>   // for strlen()

#include "config.h"   // configuration -- how many monitors do we have, how many control boards.

#include "ctl.h" // threading control
#include "machine.h" // the state of the machine (character line)
#include "animate.h" // animation stuff
#include "eeprom.h"  // eeprom i^2c memory reading

CTL_Thread *start; // the start of our linked list of "threads"
int tick; // the current tick; int; will wrap around, but that's not really an issue.

EEPROM *mem;

int machine_tick_speed = 10;       // The smallest unit of execution; controls the speed (in milliseconds)
int i2c_delay; //

#ifdef CFG_I2C
#include <Wire.h>   // include Wire library for I^2C

#define                      IODIRA                       0x00                          // MCP23017 address of I/O direction
#define                      IODIRB                       0x01                          // MCP23017 1=input
#define                      GPIOA                        0x12                          // MCP23017 address of GP Value
#define                      GPIOB                        0x13                          // MCP23017 address of GP Value


// Do the actual i2c communication.
void i2c_send(char address, char reg, char data)
{
    Wire.beginTransmission(address);
    Wire.send(reg);
    Wire.send(data);
    Wire.endTransmission();
}

int i2c_char2code (char ch)
{
    switch (ch) {
#include "char2letter.h" // generated lookup function
    }
}

#endif


/* The decide_program() function is called once at startup, to
 * indicate that the program has reset. It reads the 3 input pins, and
 * then decides what program to run.
 */

void decide_program ()
{
    pinMode(2, INPUT);
    pinMode(3, INPUT);
    pinMode(4, INPUT);
    char program = (digitalRead(2) << 2) + (digitalRead(3) << 1) + digitalRead(4);

    switch(program)
    {
    case 0:
            // the main program -- sourcecode shower.
            machine_tick_speed = 5000; // once every 5 seconds
            ctl_thread_add(start, ctl_thread_new(&tick_next_sourcecode_line, 0, 1));
            break;

    case 1:
            // the FRANTIC version of the sourcecode display. every 50ms a new line of goodness!
            machine_tick_speed = 50; // omg...
            ctl_thread_add(start, ctl_thread_new(&tick_next_sourcecode_line, 0, 1));
            break;

    case 2:
            // version of main prog which does everything on one line.
            machine_tick_speed = 5000; // once every 5 seconds
            ctl_thread_add(start, ctl_thread_new(&tick_next_sourcecode_line2, 0, 1));
            break;

    case 3:
            machine_tick_speed = 50;
            ctl_thread_add(start, ctl_thread_new(&tick_next_sourcecode_line2, 0, 1));
            break;

    case 4:
    {
            // VIDUM welcome greeter
            int d1 = 23; // pos of the first door
            int d2 = 37; // pos of the second door

            //machine_set(d1, 5, "VIDUM");
            //machine_set(d2, 5, "VIDUM");

            ctl_thread_add_welcome_scroll(start, 10, 0, d1, 1);
            ctl_thread_add_welcome_scroll(start, 10, d1+5, d2, 1);

            ctl_thread_add_text_blink(start, 30, d1, "VIDUM", "*****");
            ctl_thread_add_text_blink(start, 30, d2, "*****", "VIDUM");

            ctl_thread_add_welcome_scroll(start, 10, d2+5, 64, -1);
    }
            break;

    case 5:
            // FIXME -- there's more room for programs
            break;

    case 6:
            // some silly test program
            ctl_thread_add_text_scroll(start, 2, " GRATIS BIER ", 0, -1);
            ctl_thread_add_text_scroll(start, 2, ". BIJ IDUM", 10, 1);
            break;

    case 7:
            machine_tick_speed = 1000; // once every second
            ctl_thread_add(start, ctl_thread_new(&tick_testmode, 0, 1));
            break;
    }

    // Send a "blink" signal so I'll physically know it has booted.
    // this can probably be done more efficiently ;-)
    digitalWrite(13, 1); delay(50);
    digitalWrite(13, 0); delay(50);
    digitalWrite(13, 1); delay(50);
    digitalWrite(13, 0); delay(50);
    digitalWrite(13, 1); delay(50);
    digitalWrite(13, 0); delay(50);
    digitalWrite(13, 1); delay(50);
    digitalWrite(13, 0); delay(50);
}

void setup ()
{

    start = ctl_thread_new(0, 0, 0); // empty starting point

    machine_clear();  // reset the machine to a clean slate.

    decide_program(); // decide what program to run.

    tick = 0;

#ifdef CFG_I2C
    Wire.begin();   // start Wire library as I2C—Bus Master

    char board;
    for (board = 0; board < MACHINE_NUM_BOARDS; board++)
    {
              char addr = (0x4 << 3 | board);
              i2c_send(addr, IODIRA, B00000000); // A all output
              i2c_send(addr, IODIRB, B00000000); // B all output
    }
#endif

#ifdef CFG_SERIAL
    Serial.begin(115200);
#endif

    mem = eeprom_init(0x51);

    digitalWrite(13, 0); // led off
}

void tick_testmode (CTL_Thread *thr)
{
    static int x = 0;
    if (x)
    {
              machine_set(0, 64, "0123456701234567012345670123456701234567012345670123456701234567");
    }
    else
    {
              machine_set(0, 64, "ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");
    }
    x = (x+1) % 2;
}

void tick_next_sourcecode_line (CTL_Thread *thr)
{
    char *line = eeprom_read_line(mem);

    machine_clear();
    machine_set_nowrap(0, strlen(line), line);
    //Serial.println(line); // print the line also on the serial port, for debugging purposes.
    // (commented out because this is now done in machine_sync())

    free(line);                                                                 // make sure we cleanup after us! otherwise the memory will fill up very...⇒
... quicly.
}

int sourcepos = 0;
// alternate version of the sourcecode display: everything on "one line"; e.g. no newlines.
void tick_next_sourcecode_line2 (CTL_Thread *thr)
{
    char *line = eeprom_read_line(mem);

    sourcepos = machine_set(sourcepos, strlen(line), line);
    sourcepos = machine_set(sourcepos, 6, "      ");

    // these spaces will be overwritten
    //machine_set(sourcepos, 3, "   ");

    free(line); // do not forget!
}


void loop ()
{
    CTL_Thread *thr = start;
    do
    {
              if (thr—>thread_function && thr—>wakeup && !(tick%thr—>wakeup))
              {
                  thr—>thread_function(thr);
              }
              thr = thr—>next;
    } while (thr);
    tick++;

    //digitalWrite(13, tick%2); // blink the led

    if (_machine_dirty)
    {
#ifdef CFG_SERIAL
              Serial.write(0xff); // indicator that data is coming
              Serial.write(MACHINE_N);
              Serial.print(_machine);
#endif

#ifdef CFG_I2C
              // I^2C state synchronization here
              char board = 0; // The current address of the board
              char monitor = 0; // The current monitor
              char i = 0;

              for (monitor=0; monitor<MACHINE_MONITORS_PER_BOARD; monitor++)
              {
                  for (board=0; board<MACHINE_NUM_BOARDS; board++)
                  {
                            char addr = (0x4 << 3 | board);

                            i = monitor + (MACHINE_MONITORS_PER_BOARD * board);
#ifdef CFG_MIRRORED
                            i = MACHINE_NUM_BOARDS — i — 1;
#endif

                            i2c_send(addr, GPIOA, 0xff);
```

```
                              i2c_send(addr, GPIOB, 0xff ^ (i2c_char2code(_machine[i])));
                              i2c_send(addr, GPIOA, 0xff ^ (1 << monitor));

                              delayMicroseconds(I2C_DELAY);
                  }
              }
#endif
              _machine_dirty = 0;
    }
    delay(machine_tick_speed);
}
```

```
#ifndef ANIMATE_H
#define ANIMATE_H

// Text animations
#include "config.h"
#include "ctl.h"
#include "machine.h"

#include <stdlib.h> // for malloc
#include <string.h> // for strlen

struct TXT_Scroll
{
              char *text;
              int text_length;

              int offset;
              int p;
              int dir;
};

CTL_Thread *ctl_thread_add_text_scroll( CTL_Thread *parent, int wakeup, char *text, int offset, int dir);


struct Welcome_Scroll
{
              // A scrolling thing that displays "arrows" going to one side.
              int from;
              int to;
              int dir;
              int i;
};

CTL_Thread *ctl_thread_add_welcome_scroll( CTL_Thread *parent, int wakeup, int from, int to, int dir);


struct TXT_Blink
{
              char *text_a;
              char *text_b;
              int text_length;

              int offset;
              int i;
};

CTL_Thread *ctl_thread_add_text_blink( CTL_Thread *parent, int wakeup, int offset, char *text_a, char *text_b);


#endif
```

```
case 32: return 0;
case 33: return 1;
case 34: return 2;
case 35: return 3;
case 37: return 4;
case 38: return 5;
case 39: return 6;
case 40: return 7;
case 41: return 8;
case 42: return 9;
case 43: return 10;
case 44: return 11;
case 45: return 12;
case 46: return 13;
case 47: return 14;
case 48: return 15;
case 49: return 16;
case 50: return 17;
case 51: return 18;
case 52: return 19;
case 53: return 20;
case 54: return 21;
case 55: return 22;
case 56: return 23;
case 57: return 24;
case 58: return 25;
case 59: return 26;
case 60: return 27;
case 61: return 28;
case 62: return 29;
case 63: return 30;
case 64: return 31;
case 65: return 32;
case 66: return 33;
case 67: return 34;
case 68: return 35;
case 69: return 36;
case 70: return 37;
case 71: return 38;
case 72: return 39;
case 73: return 40;
case 74: return 41;
case 75: return 42;
case 76: return 43;
case 77: return 44;
case 78: return 45;
case 79: return 46;
case 80: return 47;
case 81: return 48;
case 82: return 49;
case 83: return 50;
case 84: return 51;
case 85: return 52;
case 86: return 53;
case 87: return 54;
case 88: return 55;
case 89: return 56;
case 90: return 57;
case 91: return 58;
case 92: return 59;
case 93: return 60;
case 94: return 61;
case 95: return 62;
case 96: return 63;
case 123: return 64;
case 124: return 65;
case 125: return 66;
case 128: return 67;
case 129: return 68;
case 130: return 69;
case 131: return 70;
case 132: return 71;
case 133: return 72;


#define CFG_SERIAL // use serial I/O for monitoring
#define CFG_I2C    // send to the vga boards

//#define CFG_MIRRORED //  mirrored output

//#define MACHINE_MONITORS_PER_BOARD 6  // The nr of monitors per vga control board
//#define MACHINE_N   40                // The number of monitors
//#define MACHINE_NUM_BOARDS 7          // Number of boards; MUST match ceil(N / (monitors per board)) !!

#define MACHINE_MONITORS_PER_BOARD 8  // The nr of monitors per vga control board
#define MACHINE_N   64                // The number of monitors
#define MACHINE_NUM_BOARDS 8          // Number of boards; MUST match ceil(N / (monitors per board)) !!

#define I2C_DELAY 20

// look up bill spinhoven! (willem)


#ifndef CTL_H
#define CTL_H


struct CTL_Thread
{
               void (*thread_function)(CTL_Thread *); // The "tick" function
               int wakeup; // When to wake up -- every X cycles: 1 is the fastests (every cycle)
               void *state; // Thread state -- can by anything.
               struct CTL_Thread *next; // A pointer to the next thread
};


CTL_Thread *ctl_thread_new( void (*thread_function)(CTL_Thread *), void *state, int wakeup);
void ctl_thread_add( CTL_Thread *thread, CTL_Thread *next);
int ctl_thread_del( CTL_Thread *thread, CTL_Thread *to_delete);
void ctl_thread_set_wakeup( CTL_Thread *thread, int wakeup);

#endif



/*
 * Read EEPROM chips on arduino over I2C.
 */

#ifndef EEPROM_H
#define EEPROM_H

// I have bought 2 chips of type 24LC512, that are each 65 kilobytes.
// They are attached to a prototyping board on top of the arduino.
#define MAX_ADDR 65535

// we read from the memory modules in chunks of 24 bytes.
#define PAGE_SIZE 24

// a data structure to read an EEPROM chip.
struct EEPROM
{
               int addr; // the address;
               // either 0x50 for the first chip,
               // or 0x51 for the second chip.
               // I might add more chips in the future..

               int cur; // current address we're reading
               int max; // the limit
};


// function definitions
EEPROM *eeprom_init(int addr);
char *eeprom_read_line(EEPROM *e);

#endif



#ifndef MACHINE_H
#define MACHINE_H

#include "config.h"

extern char _machine[MACHINE_N];      // The machine state
extern char _machine_dirty;          //  The "dirty" flag


#ifdef SIMULATOR
#include <stdio.h>
#endif

// returns the position of the last character printed.
int machine_set(int x, int l, const char *data);

void machine_set_nowrap(int x, int l, const char *data);
void machine_sync ();
void machine_clear ();

#endif




// Animations, scrolling, text display. All kinds of tick functions.
//
// Arjan Scherpenisse <arjan@scherpenisse.net>, June 2009
//

#include "animate.h" // the definitions
```

```c
extern char _machine[MACHINE_N];      // The machine state
extern char _machine_dirty;          //  The "dirty" flag


void txt_scroll_tick(CTL_Thread *t)
{
    TXT_Scroll *txt = (TXT_Scroll *)t->state;

    // advance the scroller one step
    txt->p += txt->dir;

    // set the machine text
    machine_set(txt->p + txt->offset, txt->text_length, txt->text);

    if (txt->p < -MACHINE_N)
               txt->p += 2*MACHINE_N;
    else if (txt->p > 2*MACHINE_N)
               txt->p -= 2*MACHINE_N;
}


CTL_Thread *ctl_thread_add_text_scroll( CTL_Thread *parent, int wakeup, char *text, int offset, int dir)
{
    TXT_Scroll *txt;
    CTL_Thread *thr;
    int l = strlen(text);

    txt = (TXT_Scroll *)malloc(sizeof(TXT_Scroll));

    txt->text = (char *)malloc(l * sizeof(char));
    strncpy(txt->text, text, l);
    txt->text_length = l;

    txt->offset = offset;
    txt->dir    = dir;
#ifdef CFG_MIRRORED
    txt->dir    = -1 * txt->dir;
#endif

    txt->p      = 0;

    thr = ctl_thread_new(&txt_scroll_tick, (void *)txt, wakeup);
    ctl_thread_add(parent, thr);
    return thr;
}


char *t0 = ">  ";
char *t1 = " > ";
char *t2 = "  >";
char *t3 = "  <";
char *t4 = " < ";
char *t5 = "<  ";

void welcome_scroll_tick(CTL_Thread *t)
{
    Welcome_Scroll *wel = (Welcome_Scroll *)t->state;

    char* txt;

    // advance the scroller one step
    wel->i = (wel->i + 1) % 3;
    if (wel->dir > 0)
    {
               switch(wel->i) {
               case 0: txt = t0; break;
               case 1: txt = t1; break;
               case 2: txt = t2; break;
               }
    }
    else
    {
               switch(wel->i) {
               case 0: txt = t3; break;
               case 1: txt = t4; break;
               case 2: txt = t5; break;
               }
    }

    for (int i=wel->from; i<wel->to; i++)
    {
               _machine[i] = txt[i % 3];
    }

    _machine_dirty = 1;
}


CTL_Thread *ctl_thread_add_welcome_scroll( CTL_Thread *parent, int wakeup, int from, int to, int dir)
{
    Welcome_Scroll *wel;
    CTL_Thread *thr;

    wel = (Welcome_Scroll *)malloc(sizeof(Welcome_Scroll));

    wel->from = from;
    wel->to   = to;
    wel->dir  = dir;
    wel->i    = 0;
    wel->dir    = dir;

#ifdef CFG_MIRRORED
    wel->dir    = -1 * wel->dir;
#endif

    thr = ctl_thread_new(&welcome_scroll_tick, (void *)wel, wakeup);
    ctl_thread_add(parent, thr);
    return thr;
}


/// Text blinker

void txt_blink_tick(CTL_Thread *t)
{
    TXT_Blink *txt = (TXT_Blink *)t->state;

    // advance the scroller one step
    txt->i++;

    if (txt->i % 2)
    {
```

```
		// set the machine text a
		machine_set(txt—>offset, txt—>text_length, txt—>text_a);
    }
    else
    {
		// set the machine text a
		machine_set(txt—>offset, txt—>text_length, txt—>text_b);
    }
}


CTL_Thread *ctl_thread_add_text_blink( CTL_Thread *parent, int wakeup, int offset, char *text_a, char *text_b)
{
    TXT_Blink *txt;
    CTL_Thread *thr;
    int l = strlen(text_a);

    txt = (TXT_Blink *)malloc(sizeof(TXT_Blink));

    txt—>text_a = (char *)malloc(l * sizeof(char));
    strncpy(txt—>text_a, text_a, l);
    txt—>text_b = (char *)malloc(l * sizeof(char));
    strncpy(txt—>text_b, text_b, l);

    txt—>text_length = l;

    txt—>i      = 0;
    txt—>offset = offset;

    thr = ctl_thread_new(&txt_blink_tick, (void *)txt, wakeup);
    ctl_thread_add(parent, thr);
    return thr;
}


#include <stdlib.h>

#include "ctl.h"

// the main threading code

CTL_Thread *ctl_thread_new( void (*thread_function)(CTL_Thread *), void *state, int wakeup)
{
		CTL_Thread *t;
		t = (CTL_Thread *)malloc(sizeof(CTL_Thread));
		t—>thread_function = thread_function;

		t—>state  = state;
		t—>wakeup = wakeup;
		t—>next   = 0;

		return t;
}

/**
 * Add <next> to the tail of the linked list
 */
void ctl_thread_add( CTL_Thread *thread, CTL_Thread *next)
{
		while (thread—>next) thread = thread—>next;
		thread—>next = next;
}

/**
 * Delete <to_del> from the linked list
 */
int ctl_thread_del( CTL_Thread *thread, CTL_Thread *to_delete)
{
		CTL_Thread *prev = 0;

		while (thread != to_delete)
		{
				if (!thread—>next)
				{
						return 0; // failed
				}

				prev   = thread;
				thread = thread—>next;
		}

		// found it, lets delete it
		if (thread—>next && prev)
		{
				prev—>next = thread—>next;
		}

		free(thread);
		return 1;
}

void ctl_thread_set_wakeup( CTL_Thread *thread, int wakeup)
{
		thread—>wakeup = wakeup;
}





#include <Wire.h>
#include <WProgram.h>

#include "eeprom.h"


// Low—level EEPROM read functions.
void i2c_eeprom_write_byte( int deviceaddress, unsigned int eeaddress, byte data )
{
    int rdata = data;
    Wire.beginTransmission(deviceaddress);
    Wire.send((int)(eeaddress >> 8)); // MSB
    Wire.send((int)(eeaddress & 0xFF)); // LSB
    Wire.send(rdata);
    Wire.endTransmission();
		delay(4); // give the chip some time to recover!
}

// WARNING: address is a page address, 6—bit end will wrap around
// also, data can be maximum of about 30 bytes, because the Wire library has a buffer of 32 bytes
void i2c_eeprom_write_page( int deviceaddress, unsigned int eeaddresspage, byte* data, byte length )
{
    Wire.beginTransmission(deviceaddress);
    Wire.send((int)(eeaddresspage >> 8)); // MSB
    Wire.send((int)(eeaddresspage & 0xFF)); // LSB
    byte c;
    for ( c = 0; c < length; c++) {
               Wire.send(data[c]);
    }
    Wire.endTransmission();
    delay(4); // give the chip some time to recover!
}

byte i2c_eeprom_read_byte( int deviceaddress, unsigned int eeaddress )
{
    byte rdata = 0xFF;
    Wire.beginTransmission(deviceaddress);
    Wire.send((int)(eeaddress >> 8)); // MSB
    Wire.send((int)(eeaddress & 0xFF)); // LSB
    Wire.endTransmission();
    Wire.requestFrom(deviceaddress,1);
    if (Wire.available()) rdata = Wire.receive();
    return rdata;
}

// let's not read more than 30 or 32 bytes at a time;
// there seems to be an issue in the wire library...
void i2c_eeprom_read_buffer( int deviceaddress, unsigned int eeaddress, byte *buffer, int length )
{
    Wire.beginTransmission(deviceaddress);
    Wire.send((int)(eeaddress >> 8)); // MSB
    Wire.send((int)(eeaddress & 0xFF)); // LSB
    Wire.endTransmission();
    Wire.requestFrom(deviceaddress,length);
    int c = 0;
    for ( c = 0; c < length; c++ ) {
               if (Wire.available()) buffer[c] = Wire.receive();
    }
}

// end lowlevel functions

// initialize an eeprom structure
EEPROM *eeprom_init(int addr)
{
    EEPROM *e = (EEPROM *)malloc(sizeof(EEPROM));

    e->addr = addr;
    e->cur  = 0;    // set current position to start of the chip
    e->max = MAX_ADDR; // The maximum address
    return e;
}


// read a line from the EEPROM module, until we read either 120 chars,
// or read a newline (character 10), or the end-of-memory marker(character 0).
// memory should be free()'d by the caller!!!!
char *eeprom_read_line(EEPROM *e)

{
    byte buf[PAGE_SIZE];

    // output buffer..
    char *b = (char *)malloc(120 * sizeof(char));

    int i; // byte counter
    int s; // how much bytes are we gonna read?
    int cont = 1; // continue flag
    int read = 0; // counter, how much have we read

    while (cont)
    {
               //s = e->cur + PAGE_SIZE < e->max ? PAGE_SIZE : (e->max - e->cur);
               //if (s < 0) break;
               s = PAGE_SIZE;

               i2c_eeprom_read_buffer(e->addr, e->cur, buf, s);

               for (i=0; i<s; i++) {
                   // loop over all read bytes; we need to check if we encounter any markers.
                   if (buf[i] == 10) {
                              // encountered end-of-line, lets stop reading
                              b[read++] = 0; // mark end-of-string
                              cont = 0;
                              i++; // progress one so we'll skip over the newline the next time
                              break;
                   }
                   else if (buf[i] == 0) {
                              // encountered end-of-memory
                              b[read++] = 0; // mark end of string
                              cont = 0;
                              i = -1; // restart address
                              break;
                   }
                   else {
                              // add the read char to the buffer
                              b[read++] = buf[i];
                   }
               }
               if (read >= 120) cont = 0;

               if (i >= 0) {
                   // advance current-location pointer
                   e->cur += i;
               } else {
                   // restart
                   e->cur = 0;
               }
    }

    return b;
}


#include "machine.h"

char _machine_dirty = 0;

char _machine[MACHINE_N];

//
// set the machine to a certain string
//
int machine_set(int x, int l, const char *data)
{
               int i; // just a counter
```

```
        int p; // the wrapping point

        int ret; // return value

        //printf("%d\n", x);

        while (x<0)  x += MACHINE_N;  // We need to make sure that X is wrapped
        while (x>=MACHINE_N) x -= MACHINE_N;


        if (x+l <= MACHINE_N)
        {
                // no wrapping
                for (i=0; i<l; i++)
                {
                        _machine[x+i] = data[i];
                }
                ret = x + i;
        }
        else
        {
                // calculate wrapping point
                p = l- ((x+l)-MACHINE_N);

                for (i=0; i<p; i++)
                {
                        _machine[x+i] = data[i];
                }
                ret = x + i;

                for (i=p; i<l; i++)
                {
                    if (i-p >= MACHINE_N) break;
                    _machine[i-p] = data[i];
                    ret = i-p;
                }
        }
        _machine_dirty = 1;

        return ret % MACHINE_N; // fix for out-of-bounds...
}

// set the machine to a certain string, but without wrapping the string.
void machine_set_nowrap(int x, int l, const char *data)
{
        int i; // just a counter
        int p; // the wrapping point

        //printf("%d\n", x);

        while (x<0)  x += MACHINE_N;  // We need to make sure that X is wrapped
        while (x>=MACHINE_N) x -= MACHINE_N;

        for (i=0; i<l && i+x < MACHINE_N; i++)
        {
            _machine[x+i] = data[i];
        }
        _machine_dirty = 1;
}


void machine_sync ()
{
#ifdef SIMULATOR
        printf("%d - [%s]\n", MACHINE_N, _machine);
#else
#ifdef SERIAL
        Serial.println(_machine);
#endif

        // FIXME: I^2C state synchronization here

#endif
}



void machine_clear ()
{
        int i;
        for (i=0; i<MACHINE_N; i++) _machine[i] = ' ';
        _machine_dirty = 1;
}
```

Software: VGA code

The 64 PIC16F726 microcontrollers which are in use in the installation, all run the following program (written in assembler) to generate a VGA signal. Due to the vertical nature of assembler, the source code is displayed in two columns.

```
;;; *********************************************************
;;; VGA control code for a PIC16F62x running at a 20mhz crystal.
;;; Arjan Scherpenisse<arjan@scherpenisse.net>, may 2009
;;; This code is released in the public domain.
;;; *********************************************************

        LIST P=16F627, R=DEC    ; Use the PIC16F628 and decimal system

        #include "p16f627.inc" ; Include header file

        __config _EXTCLK_OSC & _LVP_OFF & _WDT_OFF & _PWRTE_ON & _BODEN_ON

        CBLOCK 0x20             ; Declare variable addresses starting at 0x20
          hloop, vloop          ; hsync and vsync loop variables

          dataline              ; data-variable for the b/w pixel data

          data0, data1, data2, data3, data4, data5, data6
                                ; 7 data variables for the pixel rows

          charaddr, offset
                                ; address of current char
        ENDC

#define P_HSYNC PORTA,1
#define P_VSYNC PORTA,0
#define P_GREEN PORTA,3
#define P_BLUE  PORTA,2

;
; -----------
; INITIALIZE
; -----------
;
        ORG    0x000           ; Program starts at 0x000


        CLRF   PORTA           ; Initialize port A
        CLRF   PORTB           ; Initialize port B

        BSF    STATUS,RP0      ; RAM bank 1

        CLRF   TRISA           ; All pins port A output
        MOVLW   0xFF
        MOVWF   TRISB           ; PORTB input

        BCF    STATUS,RP0      ; RAM bank 0
;
; ------------------------
; FUNCTION OF PORT A PINS
; ------------------------
;
        MOVLW    7
        MOVWF    CMCON         ; Comparators off, all pins digital I/O

        goto start


start:

;;; set some data...

        ;; checkerboard
;;      MOVLW b'10101010'
;;      MOVWF data0
;;      MOVLW b'01010101'
;;      MOVWF data1
;;      MOVLW b'10101010'
;;      MOVWF data2
;;      MOVLW b'01010101'
;;      MOVWF data3
;;      MOVLW b'10101010'
;;      MOVWF data4
;;      MOVLW b'01010101'
;;      MOVWF data5

        ;; sample, for now:
        movlw 28                ; put char in address
        movwf charaddr



;;; VERTICAL SYNC
;;; -------------
;;; visble area = 640 lines
;;; front porch = 16 lines
;;; sync pulse  = 96 lines
;;; back porch  = 48 lines

mainloop:

        ;; visible area = 640 lines ;; we make 7 pixel rows, 65 per
        ;; piece. we need 12 blanks at the top, 13 at the bottom.

;;; pixel row 0; 6 pixels = 80 lines
        ;; (data for row 0 is set in the vsync pulse, to save space.
        MOVLW   64
        MOVWF   vloop

_v_px0: CALL memhsync
        NOP
        NOP
        NOP
        DECFSZ  vloop,F
        GOTO _v_px0
        NOP

        CALL memhsync

;;; pixel row 1; 6 pixels = 80 lines
        MOVF data1, W
        MOVWF dataline
        NOP
        NOP

        MOVLW   64
        MOVWF   vloop

_v_px1: CALL memhsync
        NOP
        NOP
        NOP
        DECFSZ  vloop,F
        GOTO _v_px1
        NOP

        CALL memhsync

;;; pixel row 2; 6 pixels = 80 lines
        MOVF data2, W
        MOVWF dataline

        NOP
        NOP

        MOVLW   64
        MOVWF   vloop

_v_px2: CALL memhsync
        NOP
        NOP
        NOP
        DECFSZ  vloop,F
        GOTO _v_px2
        NOP

        CALL memhsync

;;; pixel row 3; 6 pixels = 80 lines
        MOVF data3, W
        MOVWF dataline

        NOP
        NOP

        MOVLW   64
        MOVWF   vloop

_v_px3: CALL memhsync
        NOP
        NOP
        NOP
        DECFSZ  vloop,F
        GOTO _v_px3
        NOP

        CALL memhsync

;;; pixel row 4; 6 pixels = 80 lines
        MOVF data4, W
        MOVWF dataline

        NOP
        NOP

        MOVLW   64
        MOVWF   vloop

_v_px4: CALL memhsync
        NOP
        NOP
        NOP
        DECFSZ  vloop,F
        GOTO _v_px4
        NOP

        CALL memhsync

;;; pixel row 5; 6 pixels = 80 lines
        MOVF data5, W
        MOVWF dataline

        NOP
        NOP

        MOVLW   64
        MOVWF   vloop

_v_px5: CALL memhsync
        NOP
        NOP
        NOP
        DECFSZ  vloop,F
        GOTO _v_px5
        NOP

        CALL memhsync

;;; pixel row 6; 6 pixels = 80 lines
        MOVF data6, W
        MOVWF dataline

        NOP
        NOP

        MOVLW   64
        MOVWF   vloop

_v_px6: CALL memhsync
        NOP
        NOP
        NOP
        DECFSZ  vloop,F
        GOTO _v_px6
        NOP

        CALL memhsync

        NOP
        NOP
        NOP
        NOP

;;; done with the pixels

        ;; front porch = 10 lines
        MOVLW   9
        MOVWF   vloop
_v_in3: CALL blankhsync152
        NOP
        DECFSZ  vloop,F
        GOTO _v_in3
        NOP

        CALL blankhsync152
        NOP
        BCF P_VSYNC             ;set vsync port low

        ;; vsync pulse = 2 lines
        NOP
        NOP
        CALL blankhsync152

        MOVF data0, W
        MOVWF dataline          ;we got some space left, set the data for row 0

        BSF P_VSYNC             ;set hsync port high again
        NOP

        ;; back porch = 33 lines
        MOVLW   32
        MOVWF   vloop
_v_in5: CALL blankhsync152
        NOP
        DECFSZ  vloop,F
        GOTO _v_in5
        NOP

        ;; load the character data
        CALL loaddatahsync

        GOTO mainloop           ; back to the top



;;; HORIZONTALSYNC
;;; ---------------
;;; One horizontal line is 31.7us -- 158 instructions.
;;; Within this horizontal line:
;;; vis area    - 25.42us = 127 instr
;;; front porch -  0.63us = 3 instr
;;; sync pulse  -  3.81us = 19 ins
;;; back porch  -  1.9us  = 9 ins

;;; for the function, I take 6 instructions out from the backporch,
;;; so there are 152 instructions left. (backporch = 3 ins)
;;; CALL takes up 2 instructions, so there are 6 left to do looping and stuff.


blankhsync152:
        ;; visible area + front porch = 130 instr
        MOVLW   25              ;25*5 = 125
        MOVWF   hloop           ; + these: 125+2 = 127
_h_inner1
        NOP
        NOP
        DECFSZ  hloop,F
        GOTO    _h_inner1         ; Inner loop = 5 usec.
        ;; 3 left

        NOP
        NOP
        NOP
        NOP

        ;; now for the hsync pulse
        BCF     P_HSYNC
        ;; 17 inbetween

        MOVLW   3               ;3*5 = 15
        MOVWF   hloop           ; + these two: 15+2 = 17
_h_inner2 NOP
        NOP
        DECFSZ  hloop,F
        GOTO    _h_inner2         ; Inner loop = 5 usec.

        BSF     P_HSYNC

        ;; back porch, nstr
        NOP
        NOP
        NOP
        RETURN



loaddatahsync:
        ;; visible area + front porch = 130 instr;
        ;; loading data takes 91, so 39 left.

        MOVLW   7               ;7*5=35
        MOVWF   hloop           ; + these: 37
_ld_inner1
        NOP
        NOP
        DECFSZ  hloop,F
        GOTO    _ld_inner1         ; Inner loop = 5 usec.
        NOP
        ;; 2 left
        NOP
        NOP

        ;; these take up 14*7 = 98 instr
        call t0                 ;load data for line 0
        movwf data0             ;put in databyte 0
        call t1                 ;etc
        movwf data1
        call t2
        movwf data2
        call t3
        movwf data3
        call t4
        movwf data4
        call t5
        movwf data5
        call t6
        movwf data6


        ;; now for the hsync pulse
        BCF     P_HSYNC
        ;; 17 inbetween

        MOVLW   3               ;3*5 = 15
        MOVWF   hloop           ; + these two: 15+2 = 17
_ld_inner2 NOP
        NOP
        DECFSZ  hloop,F
        GOTO    _ld_inner2         ; Inner loop = 5 usec.

        BSF     P_HSYNC

        ;; back porch, nstr
        NOP
        NOP
        NOP
        RETURN



datahsync152:
        ;; visible area + front porch = 130 instr
        MOVLW   25              ;25*5 = 125
        MOVWF   hloop           ; + these: 125+2 = 127
_hd_inner1
        BCF PORTA,3
        BSF PORTA,3
        DECFSZ  hloop,F
        GOTO    _hd_inner1         ; Inner loop = 5 usec.
        ;; 3 left
        NOP

        NOP
        NOP
        NOP

        ;; now for the hsync pulse
        BCF     P_HSYNC
        ;; 17 inbetween

        MOVLW   3               ;3*5 = 15
        MOVWF   hloop           ; + these two: 15+2 = 17
_hd_inner2 NOP
        NOP
        DECFSZ  hloop,F
        GOTO    _hd_inner2         ; Inner loop = 5 usec.

        BSF     P_HSYNC

        ;; back porch, nstr
        NOP
        NOP
        NOP
        RETURN

memhsync:
        ;; visible area + front porch = 130 instr
        ;; let's make it 128; so each pixel will take 16 instructions (8 wide)
        ;; data has to fit in here....
        NOP

#include px.inc

        BCF     P_GREEN         ; always clear the data at the end of the line
        ;; now for the hsync pulse
        BCF     P_HSYNC

        ;; 17 inbetween

        BTFSS   PORTB,7
        GOTO    _filladdress

        NOP
        NOP
        NOP
        NOP
        GOTO    _cont

_filladdress:
        ;; Fill the current address from the data from ports 0-4; only if data i...⇒
..not zero (e.g. all high)
        MOVF    PORTB,W
        XORLW   0xFF            ; Invert
        ANDLW   0x7F            ; Only look at lower 7 bits
        MOVWF   charaddr
        NOP
_cont:
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

        BSF     P_HSYNC

        ;; back porch, nstr
        NOP
        RETURN



        ;;
t0:
        movlw HIGH charline0
        movwf PCLATH
        movlw LOW charline0
        addwf charaddr,W
        btfsc STATUS,C
        incf PCLATH,F
        MOVWF PCL
t1:
        movlw HIGH charline1
        movwf PCLATH
        movlw LOW charline1
        addwf charaddr,W
        btfsc STATUS,C
        incf PCLATH,F
        MOVWF PCL
t2:
        movlw HIGH charline2
        movwf PCLATH
        movlw LOW charline2
        addwf charaddr,W
        btfsc STATUS,C
        incf PCLATH,F
        MOVWF PCL
t3:
        movlw HIGH charline3
        movwf PCLATH
        movlw LOW charline3
        addwf charaddr,W
        btfsc STATUS,C
        incf PCLATH,F
        MOVWF PCL
t4:
        movlw HIGH charline4
        movwf PCLATH
        movlw LOW charline4
        addwf charaddr,W
        btfsc STATUS,C
        incf PCLATH,F
        MOVWF PCL
t5:
        movlw HIGH charline5
        movwf PCLATH
        movlw LOW charline5
        addwf charaddr,W
        btfsc STATUS,C
        incf PCLATH,F
        MOVWF PCL
t6:
        movlw HIGH charline6
        movwf PCLATH
        movlw LOW charline6
        addwf charaddr,W
        btfsc STATUS,C
        incf PCLATH,F
        MOVWF PCL


;;; The in-program-romcharactermap

        org 0x200
#include <charset.inc>

        end                     ; end of program



;;; These pixels are meant to render 5 pixels wide, centered in the
;;; middle, for the letter display.

;;; 130 instructionsfor the data, but we need 5 pixels of 65px wide
;;; 640/130 * 65 = 13 instructionsper pixel
;;; 130-(5*13) = 65, so we need 33 black on one side, 32 black on the other.

;;; 33 padding instructionsgo here
        MOVLW   6              ;6*5 = 30
        MOVWF   hloop           ; + these: 30+2 = 32
_px_inner1
        NOP
        NOP
        DECFSZ  hloop,F
        GOTO    _px_inner1         ; Inner loop = 5 usec.
        NOP

;;; start pixel p0
        BTFSC   dataline, 0
        GOTO _data_set_p0
        GOTO _data_clr_p0

_data_set_p0:
        NOP
        BSF     P_GREEN
        GOTO _nxt_p0
_data_clr_p0:
        BCF     P_GREEN
        GOTO _nxt_p0
_nxt_p0:
        ;; the stuff above took 7 instructions;we need to add (13-7)=6 more
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

;;; start pixel p1
        BTFSC   dataline, 1
        GOTO _data_set_p1
        GOTO _data_clr_p1

_data_set_p1:
        NOP
        BSF     P_GREEN
        GOTO _nxt_p1
_data_clr_p1:
        BCF     P_GREEN
        GOTO _nxt_p1
_nxt_p1:
        ;; the stuff above took 7 instructions;we need to add (13-7)=6 more
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
```

```
;;; start pixel p2
        BTFSC   dataline, 2
        GOTO _data_set_p2
        GOTO _data_clr_p2

_data_set_p2:
        NOP
        BSF     P_GREEN
        GOTO _nxt_p2
_data_clr_p2:
        BCF     P_GREEN
        GOTO _nxt_p2
_nxt_p2:
        ;; the stuff above took 7 instructions;we need to add (13-7)=6 more
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

;;; start pixel p3
        BTFSC   dataline, 3
        GOTO _data_set_p3
        GOTO _data_clr_p3

_data_set_p3:
        NOP
        BSF     P_GREEN
        GOTO _nxt_p3
_data_clr_p3:
        BCF     P_GREEN
        GOTO _nxt_p3
_nxt_p3:
        ;; the stuff above took 7 instructions;we need to add (13-7)=6 more
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP

;;; start pixel p4
        BTFSC   dataline, 4
        GOTO _data_set_p4
        GOTO _data_clr_p4

_data_set_p4:
        NOP
        BSF     P_GREEN
        GOTO _nxt_p4
_data_clr_p4:
        BCF     P_GREEN
        GOTO _nxt_p4
_nxt_p4:
        ;; the stuff above took 7 instructions;we need to add (13-7)=6 more
        NOP
        NOP
        NOP

        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        BCF     P_GREEN
        MOVLW   5              ;6*5 = 30
        MOVWF   hloop           ; + these: 30+2 = 32
        NOP

;;; 32 padding instructionsgo here
_px_inner2
        NOP
        NOP
        DECFSZ  hloop,F
        GOTO    _px_inner2         ; Inner loop = 5 usec.




; -------------------------
; Font file fonts/5x7.txt,8 x 7
; Program-datacharactermap

; 73 symbols, in order:
; ' ', '!', '"', '#', '%', '&', ''', '(', ')', '*', '+', ',', '-', '.', '/', '0'...⇒
..'1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@'...⇒
..'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P'...⇒
..'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\', ']', '^', '_', '`'...⇒
..'{', '|', '}', 'Ă', 'Ą', 'Ć', 'Č', 'Ď', 'Ě'

; Jump offset: 73

charline0:
    retlw   0       ;     (#0, byte 0)
    retlw   4       ;  !  (#1, byte 0)
    retlw  20       ;  "  (#2, byte 0)
    retlw   0       ;  #  (#3, byte 0)
    retlw   0       ;  %  (#4, byte 0)
    retlw   6       ;  &  (#5, byte 0)
    retlw   4       ;  '  (#6, byte 0)
    retlw   8       ;  (  (#7, byte 0)
    retlw   1       ;  )  (#8, byte 0)
    retlw   0       ;  *  (#9, byte 0)
    retlw   0       ;  +  (#10, byte 0)
    retlw   0       ;  ,  (#11, byte 0)
    retlw   0       ;  -  (#12, byte 0)
    retlw   0       ;  .  (#13, byte 0)
    retlw   0       ;  /  (#14, byte 0)
    retlw  14       ;  0  (#15, byte 0)
    retlw   8       ;  1  (#16, byte 0)
    retlw  14       ;  2  (#17, byte 0)
    retlw  31       ;  3  (#18, byte 0)
    retlw  12       ;  4  (#19, byte 0)
    retlw  31       ;  5  (#20, byte 0)
    retlw  14       ;  6  (#21, byte 0)
    retlw  31       ;  7  (#22, byte 0)
    retlw  14       ;  8  (#23, byte 0)
    retlw  14       ;  9  (#24, byte 0)
    retlw   0       ;  :  (#25, byte 0)
    retlw   0       ;  ;  (#26, byte 0)
    retlw   8       ;  <  (#27, byte 0)
    retlw  0        ;  =  (#28, byte 0)
    retlw  1        ;  >  (#29, byte 0)
    retlw 14        ;  ?  (#30, byte 0)
    retlw 14        ;  @  (#31, byte 0)
    retlw 14        ;  A  (#32, byte 0)
    retlw 15        ;  B  (#33, byte 0)
    retlw 28        ;  C  (#34, byte 0)
    retlw  7        ;  D  (#35, byte 0)
    retlw 31        ;  E  (#36, byte 0)
    retlw 31        ;  F  (#37, byte 0)
    retlw 14        ;  G  (#38, byte 0)
    retlw 17        ;  H  (#39, byte 0)
    retlw 14        ;  I  (#40, byte 0)
    retlw 30        ;  J  (#41, byte 0)
    retlw 17        ;  K  (#42, byte 0)
    retlw  1        ;  L  (#43, byte 0)
    retlw 17        ;  M  (#44, byte 0)
    retlw 17        ;  N  (#45, byte 0)
    retlw 14        ;  O  (#46, byte 0)
    retlw 15        ;  P  (#47, byte 0)
    retlw 14        ;  Q  (#48, byte 0)
    retlw 15        ;  R  (#49, byte 0)
    retlw 30        ;  S  (#50, byte 0)
    retlw 31        ;  T  (#51, byte 0)
    retlw 17        ;  U  (#52, byte 0)
    retlw 17        ;  V  (#53, byte 0)
    retlw 17        ;  W  (#54, byte 0)
    retlw 17        ;  X  (#55, byte 0)
    retlw 17        ;  Y  (#56, byte 0)
    retlw 31        ;  Z  (#57, byte 0)
    retlw 28        ;  [  (#58, byte 0)
    retlw  0        ;  \  (#59, byte 0)
    retlw  7        ;  ]  (#60, byte 0)
    retlw  4        ;  ^  (#61, byte 0)
    retlw  0        ;  _  (#62, byte 0)
    retlw  2        ;  `  (#63, byte 0)
    retlw 24        ;  {  (#64, byte 0)
    retlw  4        ;  |  (#65, byte 0)
    retlw  3        ;  }  (#66, byte 0)
    retlw 31        ;  Ă  (#67, byte 0)
    retlw 31        ;  Ą  (#68, byte 0)
    retlw  0        ;  Ć  (#69, byte 0)
    retlw  0        ;  Č  (#70, byte 0)
    retlw 10        ;  Ď  (#71, byte 0)
    retlw 21        ;  Ě  (#72, byte 0)
charline1:
    retlw  0        ;     (#0, byte 1)
    retlw  4        ;  !  (#1, byte 1)
    retlw 20        ;  "  (#2, byte 1)
    retlw 10        ;  #  (#3, byte 1)
    retlw 17        ;  %  (#4, byte 1)
    retlw  9        ;  &  (#5, byte 1)
    retlw  4        ;  ’  (#6, byte 1)
    retlw  4        ;  (  (#7, byte 1)
    retlw  2        ;  )  (#8, byte 1)
    retlw 21        ;  *  (#9, byte 1)
    retlw  4        ;  +  (#10, byte 1)
    retlw  0        ;  ,  (#11, byte 1)
    retlw  0        ;  -  (#12, byte 1)
    retlw  0        ;  .  (#13, byte 1)
    retlw 16        ;  /  (#14, byte 1)
    retlw 17        ;  0  (#15, byte 1)
    retlw 14        ;  1  (#16, byte 1)
    retlw 17        ;  2  (#17, byte 1)
    retlw  8        ;  3  (#18, byte 1)
    retlw 10        ;  4  (#19, byte 1)
    retlw  1        ;  5  (#20, byte 1)
    retlw  1        ;  6  (#21, byte 1)
    retlw 16        ;  7  (#22, byte 1)
    retlw 17        ;  8  (#23, byte 1)
    retlw 17        ;  9  (#24, byte 1)
    retlw  4        ;  :  (#25, byte 1)
    retlw  4        ;  ;  (#26, byte 1)
    retlw  4        ;  <  (#27, byte 1)
    retlw  0        ;  =  (#28, byte 1)
    retlw  2        ;  >  (#29, byte 1)
    retlw 17        ;  ?  (#30, byte 1)
    retlw 17        ;  @  (#31, byte 1)
    retlw 17        ;  A  (#32, byte 1)
    retlw 17        ;  B  (#33, byte 1)
    retlw  2        ;  C  (#34, byte 1)
    retlw  9        ;  D  (#35, byte 1)
    retlw  1        ;  E  (#36, byte 1)
    retlw  1        ;  F  (#37, byte 1)
    retlw 17        ;  G  (#38, byte 1)
    retlw 17        ;  H  (#39, byte 1)
    retlw  4        ;  I  (#40, byte 1)
    retlw 16        ;  J  (#41, byte 1)
    retlw  9        ;  K  (#42, byte 1)
    retlw  1        ;  L  (#43, byte 1)
    retlw 27        ;  M  (#44, byte 1)
    retlw 17        ;  N  (#45, byte 1)
    retlw 17        ;  O  (#46, byte 1)
    retlw 17        ;  P  (#47, byte 1)
    retlw 17        ;  Q  (#48, byte 1)
    retlw 17        ;  R  (#49, byte 1)
    retlw  1        ;  S  (#50, byte 1)
    retlw  4        ;  T  (#51, byte 1)
    retlw 17        ;  U  (#52, byte 1)
    retlw 17        ;  V  (#53, byte 1)
    retlw 21        ;  W  (#54, byte 1)
    retlw 17        ;  X  (#55, byte 1)
    retlw 17        ;  Y  (#56, byte 1)
    retlw 16        ;  Z  (#57, byte 1)
    retlw  4        ;  [  (#58, byte 1)
    retlw  1        ;  \  (#59, byte 1)
    retlw  4        ;  ]  (#60, byte 1)
    retlw 10        ;  ^  (#61, byte 1)
    retlw  0        ;  _  (#62, byte 1)
    retlw  4        ;  `  (#63, byte 1)
    retlw  4        ;  {  (#64, byte 1)
    retlw  4        ;  |  (#65, byte 1)
    retlw  4        ;  }  (#66, byte 1)
    retlw 31        ;  Ă  (#67, byte 1)
    retlw 17        ;  Ą  (#68, byte 1)
    retlw 10        ;  Ć  (#69, byte 1)
    retlw 10        ;  Č  (#70, byte 1)
    retlw 21        ;  Ď  (#71, byte 1)
    retlw 10        ;  Ě  (#72, byte 1)
charline2:
    retlw  0        ;     (#0, byte 2)
    retlw  4        ;  !  (#1, byte 2)
    retlw  0        ;  "  (#2, byte 2)
    retlw 31        ;  #  (#3, byte 2)
    retlw  8        ;  %  (#4, byte 2)
    retlw  5        ;  &  (#5, byte 2)
    retlw  0        ;  ’  (#6, byte 2)
    retlw  2        ;  (  (#7, byte 2)
    retlw  4        ;  )  (#8, byte 2)
    retlw 14        ;  *  (#9, byte 2)
    retlw  4        ;  +  (#10, byte 2)
    retlw  0        ;  ,  (#11, byte 2)
    retlw  0        ;  -  (#12, byte 2)
    retlw  0        ;  .  (#13, byte 2)
    retlw  8        ;  /  (#14, byte 2)
    retlw 19        ;  0  (#15, byte 2)
    retlw  8        ;  1  (#16, byte 2)
    retlw 16        ;  2  (#17, byte 2)
    retlw  4        ;  3  (#18, byte 2)
    retlw  9        ;  4  (#19, byte 2)
    retlw  1        ;  5  (#20, byte 2)
    retlw  1        ;  6  (#21, byte 2)
    retlw 16        ;  7  (#22, byte 2)
    retlw 17        ;  8  (#23, byte 2)
    retlw 17        ;  9  (#24, byte 2)
    retlw  0        ;  :  (#25, byte 2)
    retlw  0        ;  ;  (#26, byte 2)
    retlw  2        ;  <  (#27, byte 2)
    retlw 31        ;  =  (#28, byte 2)
    retlw  4        ;  >  (#29, byte 2)
    retlw 16        ;  ?  (#30, byte 2)
    retlw 29        ;  @  (#31, byte 2)
    retlw 17        ;  A  (#32, byte 2)
    retlw 17        ;  B  (#33, byte 2)
    retlw  1        ;  C  (#34, byte 2)
    retlw 17        ;  D  (#35, byte 2)
    retlw  1        ;  E  (#36, byte 2)
    retlw  1        ;  F  (#37, byte 2)
    retlw  1        ;  G  (#38, byte 2)
    retlw 17        ;  H  (#39, byte 2)
    retlw  4        ;  I  (#40, byte 2)
    retlw 16        ;  J  (#41, byte 2)
    retlw  5        ;  K  (#42, byte 2)
    retlw  1        ;  L  (#43, byte 2)
    retlw 21        ;  M  (#44, byte 2)
    retlw 19        ;  N  (#45, byte 2)
    retlw 17        ;  O  (#46, byte 2)
    retlw 17        ;  P  (#47, byte 2)
    retlw 17        ;  Q  (#48, byte 2)
    retlw 17        ;  R  (#49, byte 2)
    retlw  1        ;  S  (#50, byte 2)
    retlw  4        ;  T  (#51, byte 2)
    retlw 17        ;  U  (#52, byte 2)
    retlw 17        ;  V  (#53, byte 2)
    retlw 21        ;  W  (#54, byte 2)
    retlw 10        ;  X  (#55, byte 2)
    retlw 17        ;  Y  (#56, byte 2)
    retlw  8        ;  Z  (#57, byte 2)
    retlw  4        ;  [  (#58, byte 2)
    retlw  2        ;  \  (#59, byte 2)
    retlw  4        ;  ]  (#60, byte 2)
    retlw 17        ;  ^  (#61, byte 2)
    retlw  0        ;  _  (#62, byte 2)
    retlw  0        ;  `  (#63, byte 2)
    retlw  4        ;  {  (#64, byte 2)
    retlw  4        ;  |  (#65, byte 2)
    retlw  4        ;  }  (#66, byte 2)
    retlw 31        ;  Ă  (#67, byte 2)
    retlw 17        ;  Ą  (#68, byte 2)
    retlw 10        ;  Ć  (#69, byte 2)
    retlw 10        ;  Č  (#70, byte 2)
    retlw 10        ;  Ď  (#71, byte 2)
    retlw 21        ;  Ě  (#72, byte 2)
charline3:
    retlw  0        ;     (#0, byte 3)
    retlw  4        ;  !  (#1, byte 3)
    retlw  0        ;  "  (#2, byte 3)
    retlw 10        ;  #  (#3, byte 3)
    retlw  4        ;  %  (#4, byte 3)
    retlw  2        ;  &  (#5, byte 3)
    retlw  0        ;  ’  (#6, byte 3)
    retlw  2        ;  (  (#7, byte 3)
    retlw  4        ;  )  (#8, byte 3)
    retlw 31        ;  *  (#9, byte 3)
    retlw 31        ;  +  (#10, byte 3)
    retlw  0        ;  ,  (#11, byte 3)
    retlw 31        ;  -  (#12, byte 3)
    retlw  0        ;  .  (#13, byte 3)
    retlw  4        ;  /  (#14, byte 3)
    retlw 21        ;  0  (#15, byte 3)
    retlw  8        ;  1  (#16, byte 3)
    retlw  8        ;  2  (#17, byte 3)
    retlw 14        ;  3  (#18, byte 3)
    retlw  9        ;  4  (#19, byte 3)
    retlw 14        ;  5  (#20, byte 3)
    retlw 15        ;  6  (#21, byte 3)
    retlw 14        ;  7  (#22, byte 3)
    retlw 14        ;  8  (#23, byte 3)
    retlw 30        ;  9  (#24, byte 3)
    retlw  0        ;  :  (#25, byte 3)
    retlw  0        ;  ;  (#26, byte 3)
    retlw  1        ;  <  (#27, byte 3)
    retlw  0        ;  =  (#28, byte 3)
    retlw  8        ;  >  (#29, byte 3)
    retlw  8        ;  ?  (#30, byte 3)
    retlw 21        ;  @  (#31, byte 3)
    retlw 31        ;  A  (#32, byte 3)
    retlw 15        ;  B  (#33, byte 3)
    retlw  1        ;  C  (#34, byte 3)
    retlw 17        ;  D  (#35, byte 3)
    retlw  7        ;  E  (#36, byte 3)
    retlw  7        ;  F  (#37, byte 3)
    retlw 25        ;  G  (#38, byte 3)
    retlw 31        ;  H  (#39, byte 3)
    retlw  4        ;  I  (#40, byte 3)
    retlw 16        ;  J  (#41, byte 3)
    retlw  3        ;  K  (#42, byte 3)
    retlw  1        ;  L  (#43, byte 3)
    retlw 17        ;  M  (#44, byte 3)
    retlw 21        ;  N  (#45, byte 3)
    retlw 17        ;  O  (#46, byte 3)
    retlw 17        ;  P  (#47, byte 3)
    retlw 17        ;  Q  (#48, byte 3)
    retlw 17        ;  R  (#49, byte 3)
    retlw 14        ;  S  (#50, byte 3)
    retlw  4        ;  T  (#51, byte 3)
    retlw  17        ;  U  (#52, byte 3)
    retlw  17        ;  V  (#53, byte 3)
    retlw  21        ;  W  (#54, byte 3)
    retlw   4        ;  X  (#55, byte 3)
    retlw  14        ;  Y  (#56, byte 3)
    retlw   4        ;  Z  (#57, byte 3)
    retlw   4        ;  [  (#58, byte 3)
    retlw   4        ;  \  (#59, byte 3)
    retlw   4        ;  ]  (#60, byte 3)
    retlw   0        ;  ˆ  (#61, byte 3)
    retlw   0        ;  _  (#62, byte 3)
    retlw   0        ;  `  (#63, byte 3)
    retlw   2        ;  {  (#64, byte 3)
    retlw   4        ;  |  (#65, byte 3)
    retlw   8        ;  }  (#66, byte 3)
    retlw  31        ;  Ă  (#67, byte 3)
    retlw  17        ;  Ą  (#68, byte 3)
    retlw   0        ;  Ć  (#69, byte 3)
    retlw   0        ;  Č  (#70, byte 3)
    retlw  21        ;  Ď  (#71, byte 3)
    retlw  10        ;  Ě  (#72, byte 3)
charline4:
    retlw   0        ;     (#0, byte 4)
    retlw   0        ;  !  (#1, byte 4)
    retlw   0        ;  "  (#2, byte 4)
    retlw  31        ;  #  (#3, byte 4)
    retlw   2        ;  %  (#4, byte 4)
    retlw  21        ;  &  (#5, byte 4)
    retlw   0        ;  ’  (#6, byte 4)
    retlw   2        ;  (  (#7, byte 4)
    retlw   4        ;  )  (#8, byte 4)
    retlw  14        ;  *  (#9, byte 4)
    retlw   4        ;  +  (#10, byte 4)
    retlw   0        ;  ,  (#11, byte 4)
    retlw   0        ;  -  (#12, byte 4)
    retlw   0        ;  .  (#13, byte 4)
    retlw   2        ;  /  (#14, byte 4)
    retlw  25        ;  0  (#15, byte 4)
    retlw   8        ;  1  (#16, byte 4)
    retlw   4        ;  2  (#17, byte 4)
    retlw  16        ;  3  (#18, byte 4)
    retlw  31        ;  4  (#19, byte 4)
    retlw  16        ;  5  (#20, byte 4)
    retlw  17        ;  6  (#21, byte 4)
    retlw   4        ;  7  (#22, byte 4)
    retlw  17        ;  8  (#23, byte 4)
    retlw  16        ;  9  (#24, byte 4)
    retlw   0        ;  :  (#25, byte 4)
    retlw   0        ;  ;  (#26, byte 4)
    retlw   2        ;  <  (#27, byte 4)
    retlw  31        ;  =  (#28, byte 4)
    retlw   4        ;  >  (#29, byte 4)
    retlw   4        ;  ?  (#30, byte 4)
    retlw  29        ;  @  (#31, byte 4)
    retlw  17        ;  A  (#32, byte 4)
    retlw  17        ;  B  (#33, byte 4)
    retlw   1        ;  C  (#34, byte 4)
    retlw  17        ;  D  (#35, byte 4)
    retlw   1        ;  E  (#36, byte 4)
    retlw   1        ;  F  (#37, byte 4)
    retlw  17        ;  G  (#38, byte 4)
    retlw  17        ;  H  (#39, byte 4)
    retlw   4        ;  I  (#40, byte 4)
    retlw  17        ;  J  (#41, byte 4)
    retlw   5        ;  K  (#42, byte 4)
    retlw   1        ;  L  (#43, byte 4)
    retlw  17        ;  M  (#44, byte 4)
    retlw  25        ;  N  (#45, byte 4)
    retlw  17        ;  O  (#46, byte 4)
    retlw  15        ;  P  (#47, byte 4)
    retlw  19        ;  Q  (#48, byte 4)
    retlw  15        ;  R  (#49, byte 4)
    retlw  16        ;  S  (#50, byte 4)
    retlw   4        ;  T  (#51, byte 4)
    retlw  17        ;  U  (#52, byte 4)
    retlw  17        ;  V  (#53, byte 4)
    retlw  21        ;  W  (#54, byte 4)
    retlw  10        ;  X  (#55, byte 4)
    retlw   4        ;  Y  (#56, byte 4)
    retlw   2        ;  Z  (#57, byte 4)
    retlw   4        ;  [  (#58, byte 4)
    retlw   8        ;  \  (#59, byte 4)
    retlw   4        ;  ]  (#60, byte 4)
    retlw   0        ;  ˆ  (#61, byte 4)
    retlw   0        ;  _  (#62, byte 4)
    retlw   0        ;  `  (#63, byte 4)
    retlw   4        ;  {  (#64, byte 4)
    retlw   4        ;  |  (#65, byte 4)
    retlw   4        ;  }  (#66, byte 4)
    retlw  31        ;  Ă  (#67, byte 4)
    retlw  17        ;  Ą  (#68, byte 4)
    retlw  17        ;  Ć  (#69, byte 4)
    retlw   0        ;  Č  (#70, byte 4)
    retlw  10        ;  Ď  (#71, byte 4)
    retlw  21        ;  Ě  (#72, byte 4)
charline5:
    retlw   0        ;     (#0, byte 5)
    retlw   0        ;  !  (#1, byte 5)
    retlw   0        ;  "  (#2, byte 5)
    retlw  10        ;  #  (#3, byte 5)
    retlw  17        ;  %  (#4, byte 5)
    retlw   9        ;  &  (#5, byte 5)
    retlw   0        ;  ’  (#6, byte 5)
    retlw   4        ;  (  (#7, byte 5)
    retlw   2        ;  )  (#8, byte 5)
    retlw  21        ;  *  (#9, byte 5)
    retlw   4        ;  +  (#10, byte 5)
    retlw   4        ;  ,  (#11, byte 5)
    retlw   0        ;  -  (#12, byte 5)
    retlw   4        ;  .  (#13, byte 5)
    retlw   1        ;  /  (#14, byte 5)
    retlw  17        ;  0  (#15, byte 5)
    retlw   8        ;  1  (#16, byte 5)
    retlw   2        ;  2  (#17, byte 5)
    retlw  16        ;  3  (#18, byte 5)
    retlw   8        ;  4  (#19, byte 5)
    retlw  16        ;  5  (#20, byte 5)
    retlw  17        ;  6  (#21, byte 5)
    retlw   4        ;  7  (#22, byte 5)
    retlw  17        ;  8  (#23, byte 5)
    retlw  16        ;  9  (#24, byte 5)
    retlw   4        ;  :  (#25, byte 5)
```

```
    retlw   4        ;  ;  (#26, byte 5)
    retlw   4        ;  <  (#27, byte 5)
    retlw   0        ;  =  (#28, byte 5)
    retlw   2        ;  >  (#29, byte 5)
    retlw   0        ;  ?  (#30, byte 5)
    retlw   1        ;  @  (#31, byte 5)
    retlw  17        ;  A  (#32, byte 5)
    retlw  17        ;  B  (#33, byte 5)
    retlw   2        ;  C  (#34, byte 5)
    retlw   9        ;  D  (#35, byte 5)
    retlw   1        ;  E  (#36, byte 5)
    retlw   1        ;  F  (#37, byte 5)
    retlw  17        ;  G  (#38, byte 5)
    retlw  17        ;  H  (#39, byte 5)
    retlw   4        ;  I  (#40, byte 5)
    retlw  17        ;  J  (#41, byte 5)
    retlw   9        ;  K  (#42, byte 5)
    retlw   1        ;  L  (#43, byte 5)
    retlw  17        ;  M  (#44, byte 5)
    retlw  17        ;  N  (#45, byte 5)
    retlw  17        ;  O  (#46, byte 5)
    retlw   1        ;  P  (#47, byte 5)
    retlw  13        ;  Q  (#48, byte 5)
    retlw   9        ;  R  (#49, byte 5)
    retlw  16        ;  S  (#50, byte 5)
    retlw   4        ;  T  (#51, byte 5)
    retlw  25        ;  U  (#52, byte 5)
    retlw  10        ;  V  (#53, byte 5)
    retlw  21        ;  W  (#54, byte 5)
    retlw  17        ;  X  (#55, byte 5)
    retlw   4        ;  Y  (#56, byte 5)
    retlw   1        ;  Z  (#57, byte 5)
    retlw   4        ;  [  (#58, byte 5)
    retlw  16        ;  \  (#59, byte 5)
    retlw   4        ;  ]  (#60, byte 5)
    retlw   0        ;  ˆ  (#61, byte 5)
    retlw   0        ;  _  (#62, byte 5)
    retlw   0        ;  `  (#63, byte 5)
    retlw   4        ;  {  (#64, byte 5)
    retlw   4        ;  |  (#65, byte 5)
    retlw   4        ;  }  (#66, byte 5)
    retlw  31        ;  Ă  (#67, byte 5)
    retlw  17        ;  Ą  (#68, byte 5)
    retlw  14        ;  Ć  (#69, byte 5)
    retlw  14        ;  Č  (#70, byte 5)
    retlw  21        ;  Ď  (#71, byte 5)
    retlw  10        ;  Ě  (#72, byte 5)
charline6:
    retlw   0        ;     (#0, byte 6)
    retlw   4        ;  !  (#1, byte 6)
    retlw   0        ;  "  (#2, byte 6)
    retlw   0        ;  #  (#3, byte 6)
    retlw   0        ;  %  (#4, byte 6)
    retlw  22        ;  &  (#5, byte 6)
    retlw   0        ;  ’  (#6, byte 6)
    retlw   8        ;  (  (#7, byte 6)
    retlw   1        ;  )  (#8, byte 6)
    retlw   0        ;  *  (#9, byte 6)
    retlw   0        ;  +  (#10, byte 6)
    retlw   2        ;  ,  (#11, byte 6)
    retlw   0        ;  -  (#12, byte 6)
    retlw   0        ;  .  (#13, byte 6)
    retlw   0        ;  /  (#14, byte 6)
    retlw  14        ;  0  (#15, byte 6)
    retlw  30        ;  1  (#16, byte 6)
    retlw  31        ;  2  (#17, byte 6)
    retlw  15        ;  3  (#18, byte 6)
    retlw   8        ;  4  (#19, byte 6)
    retlw  15        ;  5  (#20, byte 6)
    retlw  14        ;  6  (#21, byte 6)
    retlw   4        ;  7  (#22, byte 6)
    retlw  14        ;  8  (#23, byte 6)
    retlw  14        ;  9  (#24, byte 6)
    retlw   0        ;  :  (#25, byte 6)
    retlw   2        ;  ;  (#26, byte 6)
    retlw   8        ;  <  (#27, byte 6)
    retlw   0        ;  =  (#28, byte 6)
    retlw   1        ;  >  (#29, byte 6)
    retlw   4        ;  ?  (#30, byte 6)
    retlw  14        ;  @  (#31, byte 6)
    retlw  17        ;  A  (#32, byte 6)
    retlw  15        ;  B  (#33, byte 6)
    retlw  28        ;  C  (#34, byte 6)
    retlw   7        ;  D  (#35, byte 6)
    retlw  31        ;  E  (#36, byte 6)
    retlw   1        ;  F  (#37, byte 6)
    retlw  14        ;  G  (#38, byte 6)
    retlw  17        ;  H  (#39, byte 6)
    retlw  14        ;  I  (#40, byte 6)
    retlw  14        ;  J  (#41, byte 6)
    retlw  17        ;  K  (#42, byte 6)
    retlw  31        ;  L  (#43, byte 6)
    retlw  17        ;  M  (#44, byte 6)
    retlw  17        ;  N  (#45, byte 6)
    retlw  14        ;  O  (#46, byte 6)
    retlw   1        ;  P  (#47, byte 6)
    retlw  22        ;  Q  (#48, byte 6)
    retlw  17        ;  R  (#49, byte 6)
    retlw  15        ;  S  (#50, byte 6)
    retlw   4        ;  T  (#51, byte 6)
    retlw  22        ;  U  (#52, byte 6)
    retlw   4        ;  V  (#53, byte 6)
    retlw  10        ;  W  (#54, byte 6)
    retlw  17        ;  X  (#55, byte 6)
    retlw   4        ;  Y  (#56, byte 6)
    retlw  31        ;  Z  (#57, byte 6)
    retlw  28        ;  [  (#58, byte 6)
    retlw   0        ;  \  (#59, byte 6)
    retlw   7        ;  ]  (#60, byte 6)
    retlw   0        ;  ˆ  (#61, byte 6)
    retlw  31        ;  _  (#62, byte 6)
    retlw   0        ;  `  (#63, byte 6)
    retlw  24        ;  {  (#64, byte 6)
    retlw   4        ;  |  (#65, byte 6)
    retlw   3        ;  }  (#66, byte 6)
    retlw  31        ;  Ă  (#67, byte 6)
    retlw  31        ;  Ą  (#68, byte 6)
    retlw   0        ;  Ć  (#69, byte 6)
    retlw  17        ;  Č  (#70, byte 6)
    retlw  10        ;  Ď  (#71, byte 6)
    retlw  21        ;  Ě  (#72, byte 6)
```