Matrix | 15.3 Introduction to Graphics | UT.6... (https://www.edx.org) https://courses.edx.org/courses/UTAustinX/UT...

**UTAustinX:** UT.6.01x Embedded Systems - Shape the World **KarenWest (/dashboard)** ▼

**Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware)** **Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)**

**Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum)** **Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)**

**Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)**

**Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)**

**Embedded Systems Community (/courses/UTAustinX/UT.6.01x/1T2014/e3df91316c544d3e8e21944fde3ed46c/)**

A **matrix** is a two-dimensional data structure accessed by row and column. Each element of a matrix is the same type and precision. In C, we create matrices using two sets of brackets. Figure 15.4 shows this byte matrix with six 8-bit elements. The figure also shows two possible ways to map the two-dimensional data structure into the linear address space of memory.

Help

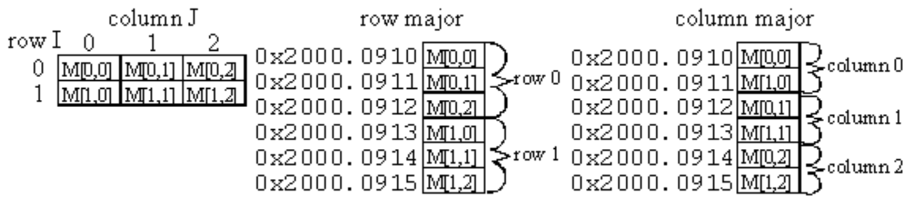

*Figure 15.4. A byte matrix with 2 rows and 3 columns.*

With **row-major allocation**, the elements of each row are stored together. Let **i** be the row index, **j** be the column index, **n** be the number of bytes in each row (equal to the number of columns), and **Base** is the base address of the byte matrix, then the address of the element at **i,j** is

```
Base+n*i+j
```

With a halfword matrix, each element requires two bytes of storage. Let **i** be the row index, **j** be the column index, **n** be the number of halfwords in each row (equal to the number of columns), and **Base** is the base address of the word matrix, then the address of the element at **i,j** is

```
Base+2*(n*i+j)
```

With a word matrix, each element requires four bytes of storage. Let **i** be the row index, **j** be the column index, **n** be the number of words in each row (equal to the number of columns), and **Base** is the base address of the word matrix, then the address of the element at **i,j** is

```
Base+4*(n*i+j)
```

As an example of a matrix, we will develop a set of driver functions to manipulate a 48 by 84 by 1-bit graphics display, see Figure 15.5.

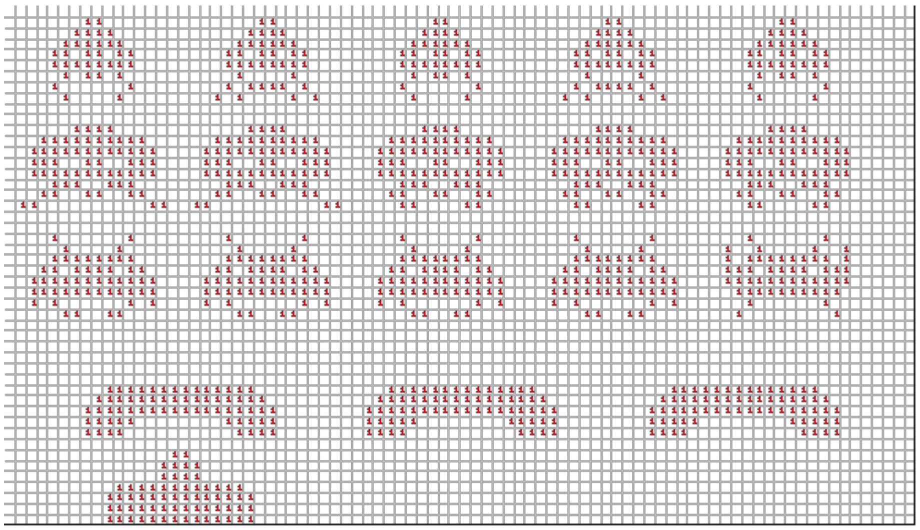

*Figure 15.5. A 1-bit matrix with 48 rows and 84 columns, each pixel is 1 bit.*

VIDEO 15.3 GRAPHICS

C15 3 Graphics

16:24 / 16:24

1.0x

Help

DR. JONATHAN VALVANO: Next, let's look at the graphics.

And we will show you the high level functions

that we'll use to generate images.

Our Nokia image is a very small image.

The size is 48 pixels high by 84 pixels wide.

And each pixel can be a zero, which is off.

Or a one, which is on.

Which means the coordinates go from 0 to 83.

And in graphics, this pixel right here is located at coordinate 0, 0.

DR. RAMESH YERRABALLI: That is the origin.

DR. JONATHAN VALVANO: That's the origin.

And this pixel, up in this corner, is going to be at the x-position of 83

and the y-position of 0.

This one down in this corner is going to be at the x-position of 0

and the y-position of 47.

And this corner will be at 83, 47.

And so in this way we will define each pixel with a unique x, y-coordinate.

The basic function for producing graphics in this game

will be to create or to draw into this screen a BMP image.

A BMP file is a standard graphics file that we can produce.

And when we place a BMP file onto the screen we will specify which file,

which image to produce.

And we will specify the position of this lower left dot.

DR. RAMESH YERRABALLI: So each sprite over there,

it's an object that's an enemy ship, or the player's ship, or a bunker.

Each of these objects is a BMP image and we call that a sprite

DR. JONATHAN VALVANO: And we can have missiles, too.

DR. RAMESH YERRABALLI: And missiles too.

DR. JONATHAN VALVANO: So the particular function that we're going to call is

labeled Nokia5110_.

That means it's a function in the Nokia module.

And it's called PrintBMP.

And the next coordinate will be the x-position.

So if we write an 8 here then the x-position

of the lower left corner of the BMP will be 8.

And if we put it at 47, then the y-coordinate will be at 47.

That'll be the bottom row.

The next parameter of this function is a pointer to the image itself.

And we'll see that there are lots of images drawn for us.

The player ship zero, is one of the predrawn

images, which will look like the player ship.

And the last coordinate, the last parameter,

is a threshold which will decide which colors to produce--

which colors to be off and which will be on.

And so if we produce a zero in this, then any color which is zero

will be off.

And any color which will be 1 to 15 will actually turn on the pixel.

This is a threshold because our BMP images are going to be 4-bit color

and we only have a 1-bit display.

DR. RAMESH YERRABALLI: So if this display were a color display

then you could change the threshold and render colored

sprite in the same place.

So Jon, it looks like the print BMP, from what I understand,

is going to change a virtual imagine and not the actual physical image.

It seems like there is another thing called the Nokia5110_DisplayBuffer.

And this routine will be the one that

actually

displays the screen, entire screen, onto the Nokia device.

DR. JONATHAN VALVANO: There's a third one.

The Nokia5110_ClearBuffer will clear the array.

So there's an array.

You'll find it in the program.

It's called screen.

And this memory buffer contains the virtual image that we will draw next.

So this memory buffer contains the virtual image of the screen

that we will draw next.

And we can see that this 504 is 48 pixels by 84 pixels, divided

by 8 pixels per byte is 504.

So these 504 bytes contain the 48 by 84 pixel rendered image

that we're going to draw next.

So the software will first clear the buffer, which will clear this screen.

And then it will draw a whole bunch of BMPs

all over the screen where you want them.

And then when you're ready to create a new image, we will call this function,

and the screen buffer is then dumped onto the actual screen

and we'll see it.

This is the element that we're going to call 30 times a second.

DR. RAMESH YERRABALLI: So Jon, you said that the display buffer

routine is called 30 times a second.

The way we describe the flow chart, the display buffer is called in the main.

So if I'm displaying in the main, the main was not an interrupt-given one.

So how is it 30 hertz?

DR. JONATHAN VALVANO: It will be 30 Hertz

because we will set a flag or a semaphore

from the interrupt service routine into the main to say, draw me a new picture.

So that's how we'll make it to be 30 hertz.

The other question is, why is it 30 hertz?

If we make it a lot slower than 30 hertz, like 10 Hertz or 5 hertz,

the display will flicker.

And if we make it a lot faster than 30 hertz

we're essentially wasting time because our eyes can see that difference.

DR. RAMESH YERRABALLI: That sounds perfect.

DR. JONATHAN VALVANO: This is the starter file for Lab 15.

The first thing we see at the top of this C file

are the hardware requirements.

These are the specific connections you must

make in order for us to be able to share your game with other students.

Again the slide pot is tied to PE2.

You have two buttons on PE0 and 1.

Your DAC is on port B. If you want to flash some LEDs,

they can be here on port B4 and 5.

The Nokia display, which is either a red one or a blue one

depends upon where you buy it.

But it's connected to PA7, 6, 5, 3, and 2.

This is a sample BMP file.

And embedded in this file, in this BMP image, are the pixels.

And down in this part of the image we can see some F's and 0's.

0's are going to be off.

F's are going to be on.

This is a 4-bit BMP file.

And it will draw the small enemy number 30.

And so there are a lot of images that we can draw, small enemy 30, small enemy

20, small enemy 10, the player ship, a small enemy, extra small enemy, some

of the bunkers.

Bunker 0 is actually one that has fully functional.

And as the bunkers are being destroyed they can be changed shape.

Bunker 1, and 2, and 3 are bunkers being destroyed.

We have some explosions, big ones, small ones.

We have missiles that can go up.

And we can have missiles that can go down.

Missiles that go up and lasers going down.

And so now we're at the main loop, the main program.

The starter file is not the game, that's yours to do.

But what I like to illustrate in this main program is how the graphics work.

So this is not the solution to the lab but rather

an explanation of how the graphics work.

We're going to be running at 80 megahertz.

There is a random number generator that you can use if you want.

And these are the functions that we need to call.

Help

DR. RAMESH YERRABALLI: So Jon, why would I

use a random number generator in a game?

DR. JONATHAN VALVANO: Well, it makes the game more fun.

Every time you play it, it's just a little bit different.

All right, the initialization function is the hardware initialization

for the Nokia display.

We talked about there being a buffer.

And so the clear buffer function will clear the screen buffer.

And then if I display that cleared buffer what we have at this point

is a completely empty screen.

Next we'll show you some examples of calling.

Let's go ahead and download the code and run it.

So download, click.

Debug.

Step over, where's my step over?

There it is, step over.

We're running now at 80 megahertz.

The random number generator is

initialized.

The hardware for the Nokia is initialized.

Notice the screen has some displays from last time I ran it.

And when I call the clear buffer we can see that the screen didn't clear,

the software virtual array got cleared.

And it's only when I execute the display buffer.

I'm going to step over to the display buffer.

And we will see the display is now cleared.

In a similar way, the next function to call is a Print BMP.

And I'm going to produce or draw the player's ship in the middle, that's

32 somewhere sort of the middle.

But in the bottom row, 47.

So I'm going to execute this function.

And again you notice the display didn't change.

That's because I just drew that bump into the virtual variable,

the virtual array.

So we'll draw a bunker, step over, draw my bunker.

Draw a bunch enemies.

Here I'm going to draw a bunch enemies.

I'm going to put them on the top row.

A bunch of different enemies, step over, step over, step over.

Again nothing happens until I hit which--

DR. RAMESH YERRABALLI: Jon, can you pause there?

I want to take a look at what your passing

as the second parameter to a Print BMP.

DR. JONATHAN VALVANO: OK, here I'm going to draw an enemy.

64 is pretty much over towards the right hand side of the screen.

This define function will tell me how high the enemy is.

And so, as you can see, this enemy is 10 pixels high.

So 10 minus 1 is 9.

And if I put him in the y-position of 9, and he's 10 high,

this will be exactly on the top.

So he is as high as he can go onto the screen.

DR. RAMESH YERRABALLI: And the third parameter

is our array that contains the BMP.

DR. JONATHAN VALVANO: Yes, this is an array, stored in ROM,

which contains the BMP image.

And the last parameter is the threshold.

A zero will show up as off.

And any number bigger than zero will show up as an on.

DR. RAMESH YERRABALLI: OK, now we're ready to display the BMP.

DR. JONATHAN VALVANO: Yeah, come on.

Let's see what it looks like.

Step over.

And we see the imagine that was created by these seven calls to the PrintBMP

is now shown on the screen.

DR. RAMESH YERRABALLI: So there are three different kinds

of enemy ships, a bunker, and the player.

DR. JONATHAN VALVANO: Absolutely.

Now no game yet.

That's your job.

This function for the starter file is just going to wait five seconds.

Let's show you another feature of the graphics driver,

and that is to do character output.

The clear function is a function which will clear the screen.

And then if you want to output characters or numbers to the screen,

we can set the cursor in the row column location,

step over, and then draw characters to the screen.

So it's in row number 1, column 1.

And it output the string.

Step over, step over, step over, step over, step over.

And down here this could be your score.

As you can see, I don't have any score.

1,234 points.

DR. RAMESH YERRABALLI: So, in summary, what we saw in the starter file and the description

before is that there's a screen buffer and a virtual buffer.

We cleared the buffer, which is clearing the buffer that is our screen.

And then we have a display buffer which will render it.

So the first thing we did is cleared it, then displayed an empty buffer.

And then we added a bunch of BMPs to the screen buffer.

And eventually, when we were done with adding all the BMPs,

we displayed this buffer.

And once we did that, we delayed for a little bit of time

and then went on to show a text message on the screen

[illegible]

Placing a 0 into a pixel location will display that pixel in a color ranging from off (0) and a 1 is fully on. In this display, the first bit is the top left corner of the display, and the last bit is the bottom right corner. The graphical image on this 48 by 84 display will be stored in the 1-bit array called **`Screen`**. Since there are a total of 4032 pixels, and each byte can store 8 pixels, we need 504 bytes to store the entire image. The screen buffer is defined in Program 15.1.

```
char Screen[504]; // stores the next image to be printed on the screen
//********Nokia5110_Init*****************
// Initialize Nokia 5110 48x84 LCD by sending the proper
// commands to the PCD8544 driver.
// inputs: none
// outputs: none
// assumes: system clock rate of 50 MHz or less
void Nokia5110_Init(void);

//********Nokia5110_OutChar*****************
// Print a character to the Nokia 5110 48x84 LCD.  The
// character will be printed at the current cursor position,
// the cursor will automatically be updated, and it will
// wrap to the next row or back to the top if necessary.
// One blank column of pixels will be printed on either side
// of the character for readability.  Since characters are 8
// pixels tall and 5 pixels wide, 12 characters fit per row,
// and there are six rows.
// inputs: data  character to print
// outputs: none
// assumes: LCD is in default horizontal addressing mode (V = 0)
void Nokia5110_OutChar(unsigned char data);

//********Nokia5110_OutString*****************
// Print a string of characters to the Nokia 5110 48x84 LCD.
// The string will automatically wrap, so padding spaces may
// be needed to make the output look optimal.
// inputs: ptr  pointer to NULL-terminated ASCII string
// outputs: none
// assumes: LCD is in default horizontal addressing mode (V = 0)
void Nokia5110_OutString(char *ptr);

//********Nokia5110_OutUDec*****************
// Output a 16-bit number in unsigned decimal format with a
// fixed size of five right-justified digits of output.
// Inputs: n  16-bit unsigned number
// Outputs: none
// assumes: LCD is in default horizontal addressing mode (V = 0)
void Nokia5110_OutUDec(unsigned short n);

//********Nokia5110_SetCursor*****************
// Move the cursor to the desired X- and Y-position.  The
// next character will be printed here.  X=0 is the leftmost
// column.  Y=0 is the top row.
// inputs: newX  new X-position of the cursor (0<=newX<=11)
//         newY  new Y-position of the cursor (0<=newY<=5)
// outputs: none
void Nokia5110_SetCursor(unsigned char newX, unsigned char newY);

//********Nokia5110_Clear*****************
// Clear the LCD by writing zeros to the entire screen and
// reset the cursor to (0,0) (top left corner of screen).
// inputs: none
// outputs: none
void Nokia5110_Clear(void);

//********Nokia5110_DrawFullImage*****************
// Fill the whole screen by drawing a 48x84 bitmap image.
// inputs: ptr  pointer to 504 byte bitmap
// outputs: none
// assumes: LCD is in default horizontal addressing mode (V = 0)
void Nokia5110_DrawFullImage(const char *ptr);

//********Nokia5110_PrintBMP*****************
// Bitmaps contain their header data and may contain padding
// to preserve 4-byte alignment.  This function takes a
// bitmap in the previously described format and puts its
// image data in the proper location in the buffer so the
// image will appear on the screen after the next call to
//   Nokia5110_DisplayBuffer();
// inputs: xpos      horizontal position of bottom left corner of image,
//                     columns from the left edge
//                     must be less than 84
//                     0 is on the left; 82 is near the right
//         ypos      vertical position of bottom left corner of image,
//                     rows from the top edge
//                     must be less than 48
//                     2 is near the top; 47 is at the bottom
//         ptr       pointer to a 16 color BMP image
//         threshold grayscale colors above this number make pixel 'on'
//                     0 to 14
//                0 is fine for ships, explosions, projectiles, and bunkers
// outputs: none
void Nokia5110_PrintBMP(unsigned char xpos, unsigned char ypos,
const unsigned char *ptr, unsigned char threshold);

// There is a buffer in RAM that holds one screen
// This routine clears this buffer
void Nokia5110_ClearBuffer(void);

//********Nokia5110_DisplayBuffer*****************
// Fill the whole screen by drawing a 48x84 screen image.
// inputs: none
// outputs: none
// assumes: LCD is in default horizontal addressing mode (V = 0)
void Nokia5110_DisplayBuffer(void);
```

*Program 15.1. Functions that display images on the LCD.*

## CHECKPOINT 15.6

An image is essentially a two-dimensional object, but memory is defined in a one-dimensional array. How do we store images in memory?

Hide Answer

The 2-array is broken into individual rows and rows are packed one row after another into memory. This format is called **row major**.

CHECKPOINT 15.7

On our system the Nokia 5110 is interfaced to the TM4C123 using the synchronous serial interface (SSI), which is running at 3.33 MHz. This means it transfers 3.33 million bits of data per second. The function **Nokia5110_DisplayBuffer** calls the function **Nokia5110_DrawFullImage**. The function **Nokia5110_DrawFullImage** sends an entire screen's worth of data. Estimate how long it takes this function to execute.

**Hide Answer**

There are 48 by 84 pixels, which is 4,032 bits of data. 4,032 divided by 3,330,000 bits/sec is about 1.22ms. Because this LCD output takes so long to perform, we will place it in the main program, or foreground, of our game.

About (https://www.edx.org/about-us) Jobs (https://www.edx.org/jobs)
Press (https://www.edx.org/press) FAQ (https://www.edx.org/student-faq)
Contact (https://www.edx.org/contact)

Help

EdX is a non-profit created by founding partners Harvard and MIT whose mission is to bring the best of higher education to students of all ages anywhere in the world, wherever there is Internet access. EdX's free online MOOCs are interactive and subjects include computer science, public health, and artificial intelligence.

(http://www.meetup.com/edX-Global-Community/)
(http://www.facebook.com/EdxOnline)
(https://twitter.com/edXOnline)
(https://plus.google.com/108235383044095082735/posts)
(http://youtube.com/user/edxonline)
© 2014 edX, some rights reserved.

Terms of Service and Honor Code -
Privacy Policy (https://www.edx.org/edx-privacy-policy)