**C15 Systems Approach to Game Design**

**C15.3 Introduction to Graphics**

**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.

**C15.5 Periodic Interrupt Using Timer 2A**

In this section we use Timer2A as a means to create interrupts. It essentially has two registers. A COUNTER and a reload register called TAILR. This reload value unlike SysTick (16 bits), is 32 bits wide.

When COUNTER counts up to the period loaded into the reload register TAILR, it sets the trigger flag (RIS flag). This bit exists as bit 0 in the **TIMER2\_RIS\_R** register. On the next count, the 32-bit timer is reloaded with the 32-bit value in **TIMER2\_TAILR\_R**. We select the periodic timer mode by setting the 2-bit **TAMR** field of the **TIMER2\_TAMR\_R** to 0x02. In periodic mode, the timer runs continuously. The timers can be used to create PWM outputs and measure pulse width, period, or frequency.

Each periodic timer module has:

            A clock enable bit, bit 2 in **SYSCTL\_RCGCTIMER\_R**            A control register, **TIMER2\_CTL\_R** (set to 0 to disable, 1 to enable)  
            A configuration register, **TIMER2\_CFG\_R** (set to 0 for 32-bit mode)  
            A mode register, **TIMER2\_TAMR\_R** (set to 2 for periodic mode)  
            A 32-bit reload register, **TIMER2\_TAILR\_R**            A resolution register, **TIMER2\_TAPR\_R** (set to 0 for 12.5ns)  
            An interrupt clear register, **TIMER2\_ICR\_R** (bit 0)  
            An interrupt arm bit, TATOIM, **TIMER2\_IM\_R** (bit 0)  
            A flag bit, TATORIS, **TIMER2\_RIS\_R** (bit 0)

**unsigned long TimerCount;  
void Timer2\_Init(unsigned long period){   
  unsigned long volatile delay;  
  SYSCTL\_RCGCTIMER\_R |= 0x04;   // 0) activate timer2  
  delay = SYSCTL\_RCGCTIMER\_R;  
  TimerCount = 0;  
  TIMER2\_CTL\_R = 0x00000000;   // 1) disable timer2A  
  TIMER2\_CFG\_R = 0x00000000;   // 2) 32-bit mode  
  TIMER2\_TAMR\_R = 0x00000002;  // 3) periodic mode  
  TIMER2\_TAILR\_R = period-1;   // 4) reload value  
  TIMER2\_TAPR\_R = 0;           // 5) clock resolution  
  TIMER2\_ICR\_R = 0x00000001;   // 6) clear timeout flag  
  TIMER2\_IMR\_R = 0x00000001;   // 7) arm timeout  
  NVIC\_PRI5\_R = (NVIC\_PRI5\_R&0x00FFFFFF)|0x80000000;   
// 8) priority 4  
  NVIC\_EN0\_R = 1<<23;          // 9) enable IRQ 23 in  
  TIMER2\_CTL\_R = 0x00000001;   // 10) enable timer2A  
}  
// trigger is Timer2A Time-Out Interrupt  
// set periodically TATORIS set on rollover  
void Timer2A\_Handler(void){  
  TIMER2\_ICR\_R = 0x00000001;  // acknowledge  
  TimerCount++;  
// run some background stuff here  
}  
void Timer2A\_Stop(void){   
  TIMER2\_CTL\_R &= ~0x00000001; // disable  
}  
void Timer2A\_Start(void){   
  TIMER2\_CTL\_R |= 0x00000001;   // enable  
}**Program 15.6. Periodic interrupts using Timer2A (included in C15\_Timer2 starter project).

**Converting .wva files**

Say you want to convert a wav file **(blah.wav**) you want to use in your game. Here are the sequence of steps you can do in Matlab (or a free alternative to Matlab from GNU called Octave – [http://octave.org](http://octave.org/))  to make this happen:

1. Read the file and extract the samples as well as the frequency:

[Spls, fs] = wavread('blah.wav');

2. Downsample it, to get it to the frequency you want (11.025 kHz):

Spls = downsample(Spls, round(fs/11025));

3. Adjust the samples (originals are fractions between -1 and 1) to range between 0 and 15 (4-bit)

Spls = round((Spls+ 1)\* 7.5);

4. write it to a file so you can cut and paste it into your C code:

file = fopen('blah.txt', 'w');

fprintf(file, 'unsigned char blah[] = {');

fprintf(file, '%d,', Spls);

fprintf(file, '};\n');

That's it, you should have a file (called **blah.txt**) with a declaration you can cut and paste in your code. There may be an extra commas in the output that you would have to remove.  You could make a convert.m file with these statements inside a subroutine and have a converter that can be invoked for conversion from matlab. (From Mitch Crooks)