Week 4: moving alien, stationary ship

Introduction week 4:

At this point you should have all of the functions at pixel.c file finished. Next thing is to figure out where to call them in your code to finish the LED-matrix interface.

After the interface for the LED-matrix is fully finished, it's time to move on to implementing the actual game. This week's topics are implementing moving alien and stationary ship.

Instructions:

Using pixel.c functions in code, game logic:

COMP.CE.100_exercise_guide.pdf, page 21

Interrupts:

COMP.CE.100_exercise_guide.pdf, page 11

Project_work_extra_material_v3, slides 2-6

Project_work_info, slides 25-30

Exercise requirements:

COMP.CE.100_exercise_guide.pdf, page 2

Setup function:

If you don't remember what the purpose of the setup()-function was, refer to the instructions for the first week. The setup()-function needs to be called only once when the program is executed. The calling of the setup()-function should happen before any of the other functions you have implemented have been called. The execution of the program starts from the main()-function in the main.c so that is where the setup() should be called. Before using other functions, some information about interrupts needs to be presented.

Interrupts:

Interrupts are a way of making the program to execute a set of instructions repeatedly after each specified time interval or to execute a set of instructions after some event has happened. In the project code there are three different interrupts already implemented for you. Two of them are timer interrupts and one of them is GPIO input interrupt. This week we'll only concentrate on the timer interrupts.

As said before there are two timer interrupts in the code template: one has frequency of 800 Hz and other has frequency of 10 Hz. The interrupt handelrs related to these are TickHandler() and TickHandler1() respectively. So the TickHandler() function gets automatically called 800 times per second and TickHandler1() gets called repeatedly 10 times per second. The 800 Hz interrupt handler will be used to control the LED-matrix and the 10 Hz interrupt handler will be used to create movement for the game.

Run and open_line:

For recap the run()-function will transfer color values of one channel to the DM163 chip's shift register. open_line()-function will open one channel, meaning that the color values at the DM163 shift register will be displayed on the channel which is currently open. Because the DM163 shift register only fits information of one channel at time, only one channel can be open at time. However, we want to make it seem so that all of the pixels in the LED-matrix seems to be on at the same time. To achieve this, we must change the currently open channel so frequently that it seems that all of the channels are open at the same time. To achieve this we use the 800 Hz timer interruption.

You need to implement TickHandler() so that each time it is called, it opens one channel and next time it is called it opens channel next to it. This way each channel is open 1/800 s at time and all of the eight channels are looped through every 1/100 seconds. To achieve this, the information about the currently open channel must be stored somewhere. A global variable can be used for this purpose. Then on each call of TickHandler() this channel variable must be changed to the next channel, and data on this new channel must be shown using run() and open_line() functions for this channel. Before calling the run() function it must be ensured that no channels are open. This will prevent so called ghost pixels from appearing because if an old channel is open, the data of the new channel will briefly flash on the old channel before the open channel is changed.

SetPixel:

Now you should have the LED-matrix interface fully implemented. After this, only thing you need to do to display something on the matrix is to call SetPixel()-function. SetPixel() sets color of one pixel. It takes as arguments x-position of the pixel, y-position of the pixel, red color intensity, green color intensity and blue color intensity. Color intensities are 8-bit integers so they range between 0 and 255. At this point you should testi if your interface is implemented correctly and working as it should. To do this you can call SetPixel() for example on main()-function after the setup() call. Test to set pixels in different positions on LED-matrix to different colors and make sure pixels are lit as expected. If something looks not to be right, you need to debug the code you have previously written.

Drawing ship and alien:

If everything was working at the last step, you can start to implement the game itself. You are free to design the game logic as you want. There are no premade functions or anything, you can, and should implement your own functions for the game logic. When designing the game, you may want to keep in mind all of the requirements for the game (see COMP.CE.100_exercise_guide.pdf, page 21). This way you can prevent some refactoring of the code later if you design carefully with all requirements in mind from the beginning.

Drawing the stationary ship is easy. You just need to set color for each of the ship's pixels with SetPixel() -function and that's it. For the alien you should do the same. However, alien should be moving. To create a movement, you can use the TickHandler1() interrupt handler which is called at 10 Hz frequency. So, when alien moves, clear the old pixels of the alien and draw new ones to the new location. Clearing pixel is obviously done by setting all of the pixel's colors to 0.

After this week you should *know and undesrstand*:

Where and why to use the pixel.c functions.

How timer interrups work.

How to draw items with SetPixel() -function and how to create movement.

After this week you *should have finished*:

Moving alien and stationary ship.