**Small led game**

An interesting application for serial to parallel shift registers would be controlling multiple leds with less pins used from the board. In this case we will build a small run-from-a-monster game.

**What are serial to parallel shift registers?**

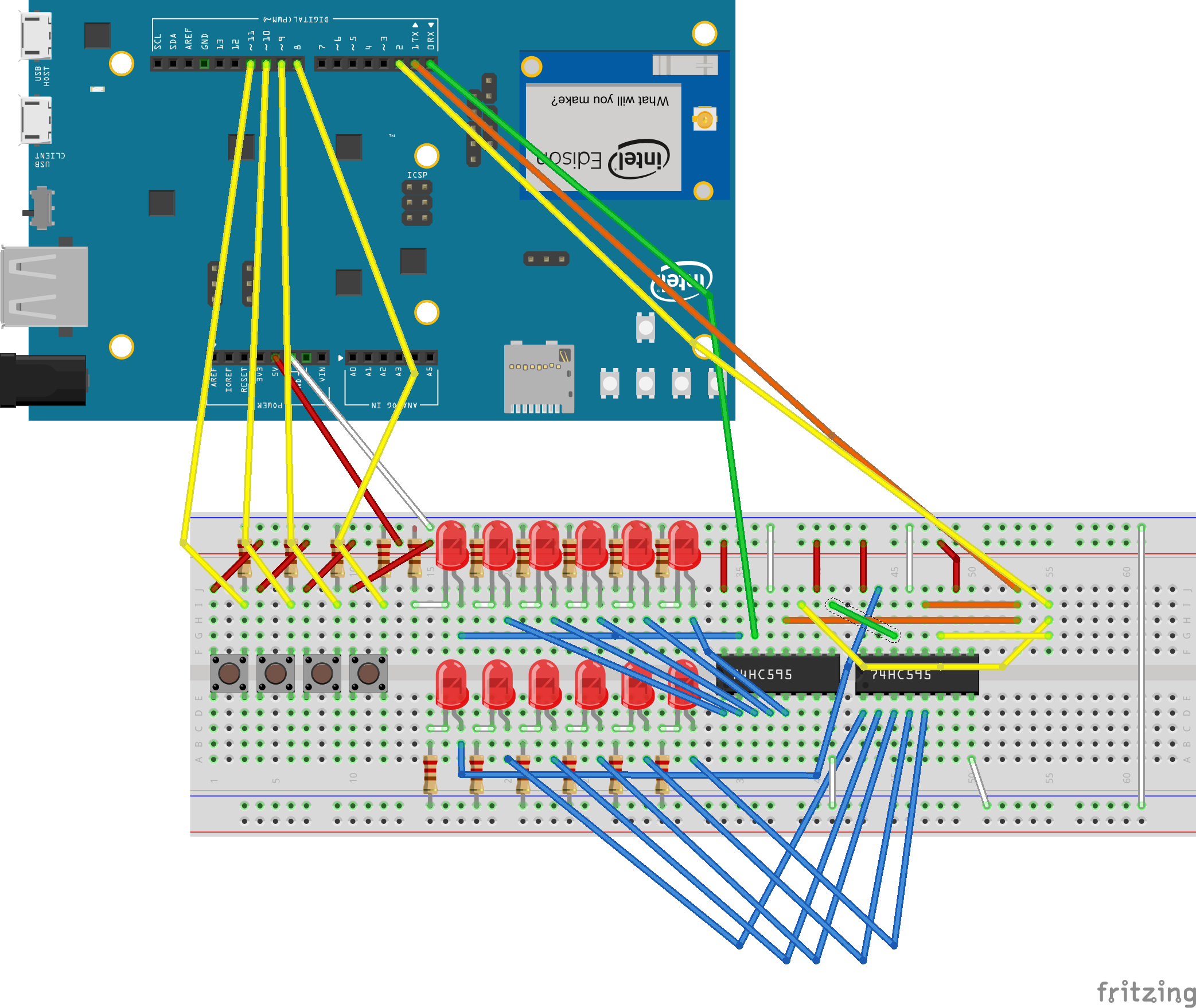
Well, instead of sending the data in parallel, which would require multiple pins from the board to be used, you simply use one line for sending the data serially. Then the register converts it to a parallel output.

**Description of the application**

The player starts in the upper left corner. In the lower right corner a monster is spawned and it will try to catch the player. Our player must do anything he can to avoid the monster. When the monster catches him, a fancy screen is displayed and then the game resets.

**Setup**

We have a matrix of leds, in our case 2x6 leds which represent positions in the map. We also need four push-buttons to control the player. We then used two 8 bit serial to parallel shift registers SN74HC595. Also, don’t forget about the needed pull-down resistors for the pushbutton and the resistors for the leds(we used 220omhs resistors). For the embedded board we used the Intel Edison v2, but the project should work on the Galileo, Raspberry Pi, Beagle Bone and Red Pitaya.



**Code**

The full code is available at <https://github.com/AndreiRO/smallledgame.git>. Only the most important parts of the code will be shown here.

We first setup the used pins to OUTPUT/INPUT mode accordingly. Two important functions for “drawing” are clear(value) and drawMap(x, y).

def clear(v):

global SER, SRCLCK, RCLCK, W, H

digitalWrite(RCLCK, LOW)

for j in range(H – 1, -1, -1):

digitalWrite(SER, v)

for i in range(W – 1, -1, -1):

digitalWrite(CLCK, LOW)

digitalWrite(CLCK, HIGH)

digitalWrite(RCLCK, HIGH)

def draw\_map(x, y):

global SER, SRCLCK, RCLCK, W, H

digitalWrite(RCLCK, LOW)

for j in range(H - 1, -1, -1):

for i in range(W - 1, -1, -1):

flag = False

for it in range(len(x)):

if x[it] == i and y[it] == j:

digitalWrite(SER, HIGH)

flag = True

break

if flag == False:

digitalWrite(SER, LOW)

digitalWrite(SRCLCK, LOW)

digitalWrite(SRCLCK, HIGH)

digitalWrite(RCLCK, HIGH)

Clear simply sets each bit to the specified value, drawMap will put zeros except for the pair of coordinates from the vectors x and y.

Important is to notice that the program first sends the value from the lower right corner finishing transmission with the upper left corner.

The main function is the following:

player = Movement(1, 0, 6, 2, 11, 10, 9, 8)

# def \_\_init\_\_(self, x, y, w, h, rate):

monster = Monster(W - 1, H - 1, W, H, 1)

ct = 0

while True:

clear(LOW)

player.Update()

if ct >= 100:

ct = 0

monster.Update(player.x, player.y)

draw\_map([player.x, monster.x], [player.y, monster.y])

ct = ct + 1

if player.x == monster.x and player.y == monster.y:

clear(HIGH)

delay(3000)

for i in range(3):

clear(LOW)

delay(500)

clear(HIGH)

delay(500)

monster.x = W - 1

monster.y = H - 1

player.x = 0

player.y = 0

delay(10)

We create the four buttons which map onto the physical buttons. The class button provides an easier interface to the physical buttons. They can be queried to check if they are pushed. holding down the button instead of clicking it(however you should add a small fraction to the player’s x and then give it as an integer parameter to drawMap).

The monster class is the most simple AI form ever. It simply tries to update its coordinates by adding or subtracting in order to get closer to the target.

The movement class handles the presses and moves the player accordingly. It the left button was pressed, the player moves a slight to the left and so on.