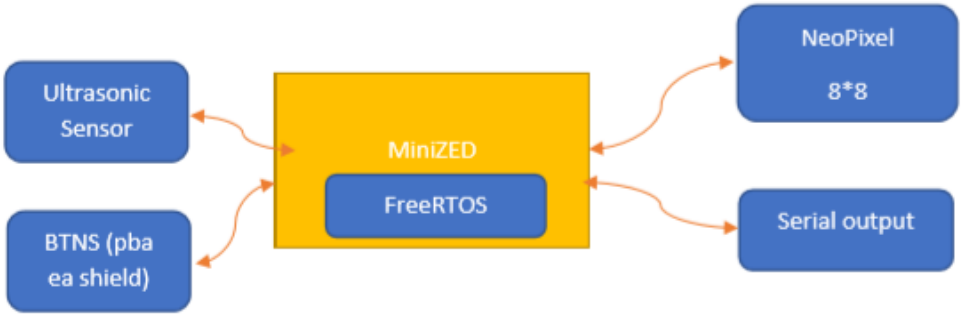
FINAL EXERCISE EOS

Dennis Keusters, Kris Teuwen, Kristof Heulsen

06-01-2020

Overzicht:

In dit project gaan we een Minized FPGA aansturen. De bedoeling is dat we als ingangen knoppen en een ultrasoon sensor gebruiken om zo een spelletje te programmeren op een Neonpixel (matrix 8*8). En deze waarde ook serieel kunnen uitprinten op een seriële uitgang. Het programeren van deze software gebeurt in vivado en in SDK.



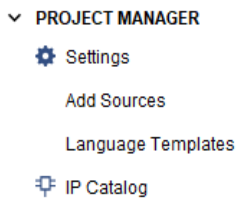
Uitleg spel:

Het spel dat we gemaakt hebben is Simon Says, waarbij een speler een kleur kan kiezen door de gebruik te maken van de ultrasoon sensor. Als hij de juiste kleur heeft gevonden kan hij door middel van de knop op de FPGA zeggen dat dit de kleur is. Als de keur juist is ga je naar het volgend level en anders verlies je de game.

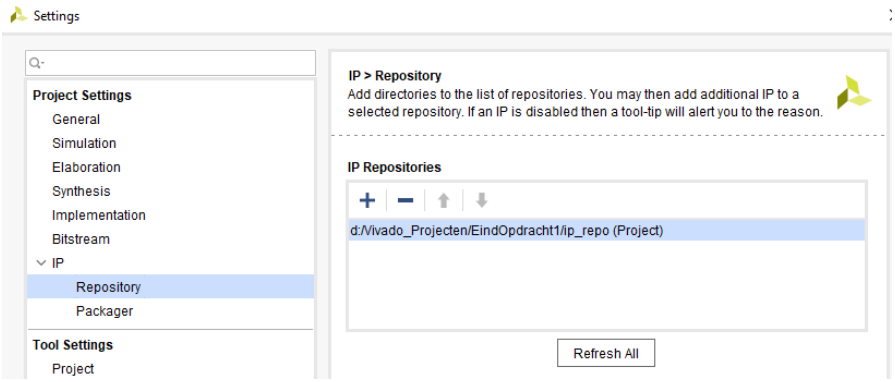
Vivado:

Hier hebben we een project aangemaakt waar we de vhdl code geschreven hebben om de ultrasoon sensor en de neonpixel aan te sturen. Vervolgens hebben we van deze code aparte IP-blokken gemaakt die we vervolgens in een blokschema kunnen gebruiken om ze daarna te kunnen aansturen in SDK.

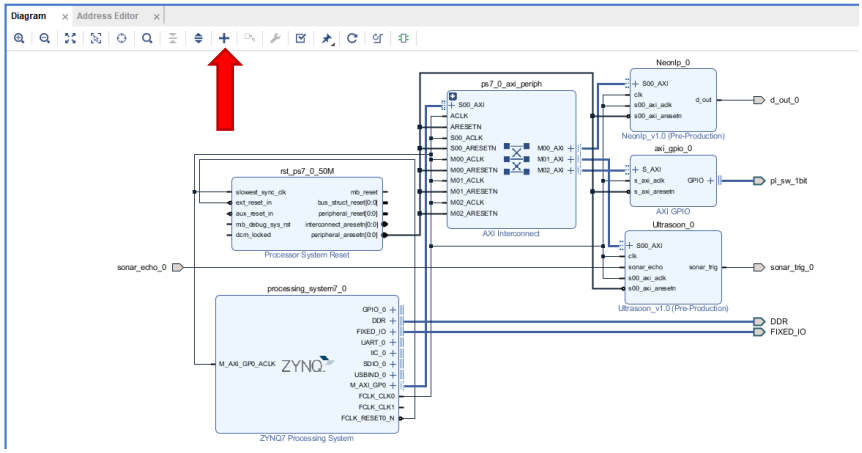
1. Open het programma dat het blokschema bevat om zo te controleren of alle blokken er in zitten.
    - Als ze er niet allemaal inzitten moet je ze toevoegen op de volgende manier.
        - Ga naar settings



        - Klik op IP -> vervolgens op Respository
        - Klik op de + om de IP-blokken toe te voegen.



- Dan kan je de map selecteren waar je al je IP-blokken gemaakt hebt om ze daarna toe te voegen in je blokdesign.
- In je blokdesign klik je op de **+** knop om alle nodige blokken toe te voegen.

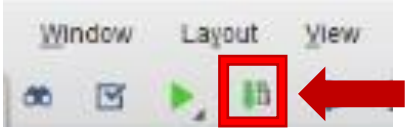


De blokken die nodig zijn: Zynq blok, al je eigen blokken en een axi gpio blok.

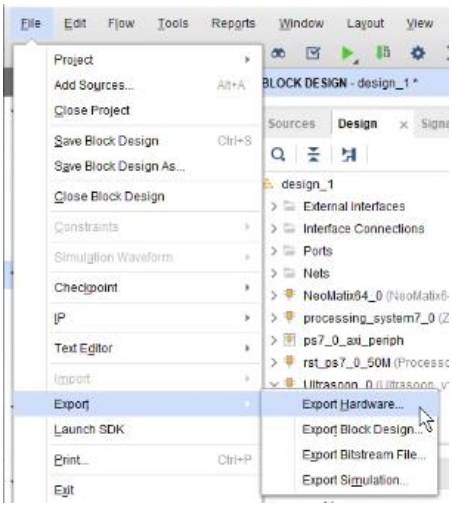
- Klik op run connection om alles met elkaar te verbinden

- Maak de nodige pinnen extern die nog niet verbonden zijn.

2. Als je ze allemaal hebt kan je een wrapper van je blokschema maken.
    - Rechtse muisknop op je blokschema design -> create hdl wrapper
3. Daarna maak je een bitstream van het programma.



    - Klik op File -> exporteer hardware :Vervolgens exporteer je de hardware samen met de bitstream naar SDK.

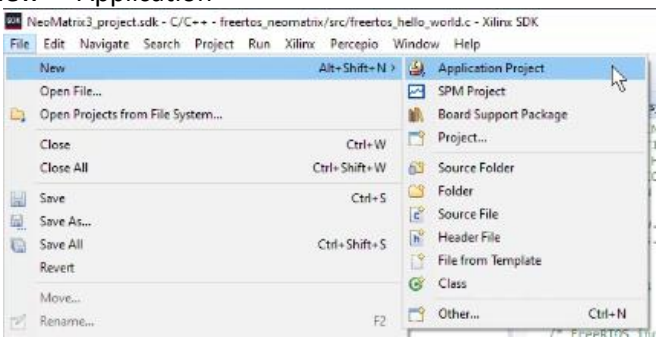


- Klik op File -> launch SDK: SDK gaat automatisch open.

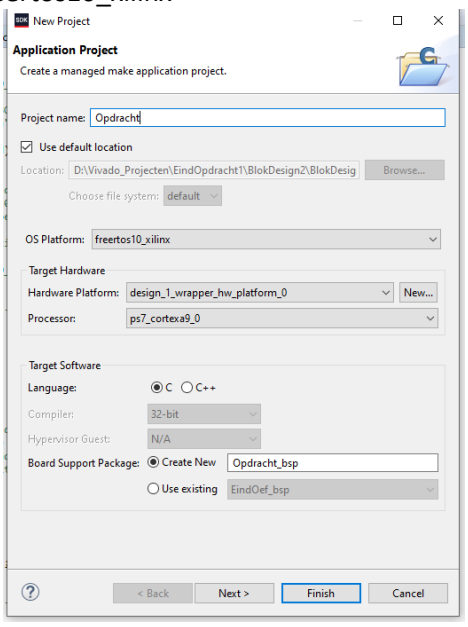
SDK:

Hier is er een applicatie aangemaakt die in C-taal geschreven word, en die werkt op Xilinx-FreeRTOS zodat we kunnen werken met verschillende soorten queues, timers, etc. .

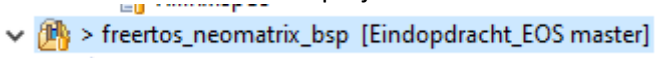
4. Maak een applicatie aan:
   - Klik op File -> New -> Application



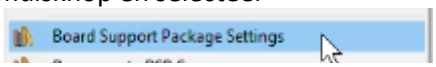
   - Geef de applicatie een naam.
   - Zet het OS platform Freertos10_xilinx



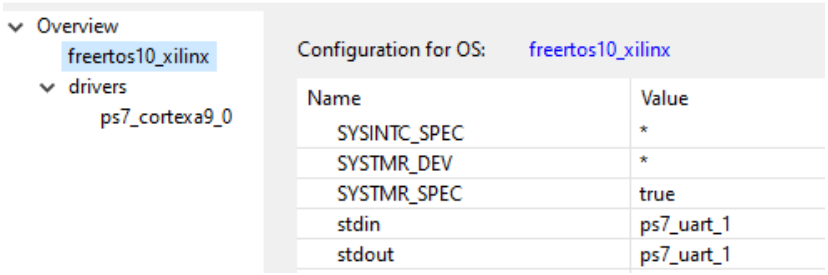
5. Ga naar de bsp file met dezelfde naam als uw project.



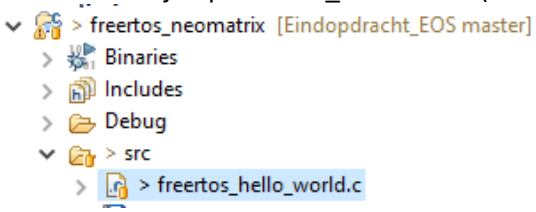
   - Klik hier met de rechtse muisknop en selecteer



   - Dan ga je naar Overview -> freertos10_xilinx en zet je de **stdin** en de **stdout** op ps7_uart_1.



6. Dan ga je naar de applicatie -> en klik je op de hello_world.c file (dit is je hoofdprogramma)



7. Extra pin info om alles aan te sluiten.

```
    1 tab == 4 spaces!
*/

/*
 Pin layout
 VCC:  5V Pin
 GND:  GND Pin
 Trig: F13 (AD1)
 Echo: F14 (AD0)
 Neomatrix : pin 2 van PMOD 1
 Vcc -> Pin 6/12 van PMOD 1
 GND -> pin 5/11 van PMOD 1
 */
```

8. Importeer al de nodige libraries, ook die van de eigen aangemaakte IP-blokken.

```
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "stdlib.h"
#include "stdio.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include "xgpiops.h"
/* includes for own IP and hardware */
#include "Ultrasoon.h"
#include "NeoMatix64.h"
#include "xil_io.h"
```

9. Maken de functies aan die nodig zijn voor de game te spelen als voor de game zelf.

```
/* The Tx and Rx tasks as described at the top of this file. */
//static void prvTxTask( void *pvParameters );
static void prvRxTask( void *pvParameters );
//tasks
static void prvSensor( void *pvParameters );
static void prvKnop(void *pvParameters);
//functies voor de timers
static void vTimerCallback( TimerHandle_t pxTimer );

// game functies
void drawScreen(int x,int y,int colorLED);
void clearScreen(void);
void drawArray(void);
void NextColorScreen (int kleur);
void generateLevel(int maxlength);
void set_player_color( u32 positie );
void gameOver(void);
```

10. Maken verschillende define variabele aan, zodat we gemakkelijker kunnen werken in onze code, en zodat we maar een variabele moeten verander in plaats van diezelfde op meerdere plaatsen.

```
// Timers
#define TIMER_ID    1
#define TIMER_ID_KNOP 1
#define DELAY_10_SECONDS    10000UL
#define DELAY_1_SECOND      1000UL
#define DELAY_0_5_SECOND 500UL
#define DELAY_0_2_SECOND 200UL
#define TIMER_CHECK_THRESHOLD   9

// knop
#define GPIO_DEVICE_ID XPAR_XGPIOPS_0_DEVICE_ID
#define Knop 0
#define KnopData 0x00

// Sensor check afstand
#define CHECKBIT(var, pos)  ((var) & (1<<(pos)))
#define CHECKRDY(var) (CHECKBIT(var, 31))
#define CHECKOOB(var) (CHECKBIT(var, 30))

// Sensor adressen
#define ULTRASOON_ADDR XPAR_ULTRASOON_0_S00_AXI_BASEADDR
#define ULTRASOON_REG0 ULTRASOON_S00_AXI_SLV_REG0_OFFSET
#define ULTRASOON_REG1 ULTRASOON_S00_AXI_SLV_REG1_OFFSET

// Neonpixel adressen
#define NEON_ADDR XPAR_NEOMATIX64_0_S00_AXI_BASEADDR /* 0x43C00000 */
#define NEON_REG0 NEOMATIX64_S00_AXI_SLV_REG0_OFFSET
#define NEON_REG1 NEOMATIX64_S00_AXI_SLV_REG1_OFFSET
```

11. Definiëren de task parameters, queues en timers die we nodig hebben om de game te laten werken.

```
static TaskHandle_t xRxTask;
static TaskHandle_t xSensor;
//static TaskHandle_t xUartRead;
static TaskHandle_t xKnop;

static QueueHandle_t xQueue = NULL;
static QueueHandle_t xQueue2 = NULL;
static QueueHandle_t xQueue3 = NULL;

static TimerHandle_t xTimer = NULL;
static TimerHandle_t xTimerKnop = NULL;
```

Main() functie:

1. Maken de tasks aan die we koppelen aan de functies, die de codes bevatten om ons spel te kunnen spelen.

```
xTaskCreate( prvRxTask,                     /* The function that implements the task. */
             ( const char * ) "Receive",    /* Text name for the task, provided to assist debugging only. */
             configMINIMAL_STACK_SIZE,      /* The stack allocated to the task. */
             NULL,                          /* The task parameter is not used, so set to NULL. */
             tskIDLE_PRIORITY + 1,          /* The task runs at the idle priority. */
             &xRxTask );                    /* Used to pass a handle to the created task. */

xTaskCreate( prvSensor,
             ( const char * ) "Sensor",
             configMINIMAL_STACK_SIZE,
             NULL,
             tskIDLE_PRIORITY,
             &xSensor );

xTaskCreate( prvKnop,
             ( const char * ) "Knop",
             configMINIMAL_STACK_SIZE,
             NULL,
             tskIDLE_PRIORITY,
             &xKnop );

xTaskCreate( prvUartRead,
             ( const char * ) "UART",
             configMINIMAL_STACK_SIZE,
             NULL,
             tskIDLE_PRIORITY + 1,
             &xUartRead );
```

2. Maken de verschillende queues die nodig zijn voor de communicatie tussen de verschillende functies en de data die in elke functie verzonden/ontvangen word.

```
xQueue = xQueueCreate(  1,                      /* There is only one space in the queue. */
                        sizeof( HWstring ) );   /* Each space in the queue is large enough to hold a uint32_t. */

xQueue2 = xQueueCreate(1, sizeof( HWstring ));
xQueue3 = xQueueCreate(1, sizeof( HWstring ));

/* Check the queue was created. */
configASSERT( xQueue );
configASSERT( xQueue2 );
configASSERT( xQueue3 );
```

   a. xQueue: Is de verbinding tussen de ultrasoon sensor (speler1) en de neon pixel.
   b. xQueue2: Is de verbinding tussen de neon pixel die de kleur waarde verstuurd naar de UART.
   c. xQueue3: Is de verbinding tussen de knop(speler 2) en de neon pixel.

3. Definiëren we de timer die we gebruiken om de delay in ons spel te maken zodat alles goed verloopt.

```
xTimer = xTimerCreate( (const char *) "Timer",
                       x0_5seconds,
                       pdTRUE,
                       (void *) TIMER_ID,
                       vTimerCallback);
```

   a. xTimer: word gebruikt voor het updaten van de neonpixel leds om de 0.5 seconden.

Ultrasoon sensor:

In deze functie staat alle code om de ultrasoon sensor in het programma te kunnen gebruiken.

1. De functie van de sensor
    a. Hier lezen we afstand in van de sensor door het basis adres en het slave register samen in te lezen en deze toe te kennen aan een variable samen met weergave van de waarde.
    b. Dan kijken we of de afstand meetbaar is of niet (licht de afstand binnen de ingestelde range van de sensor).
    c. Vervolgens steken we het adres van de afstand in de queue om zo de waarde van de afstand te verzenden naar de ontvanger om zo een uitgang aan te sturen (speler 1 op de neonpixel).

```
static void prvSensor( void *pvParameters ) //Werkt ....
{
    const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND );
    xil_printf("Ultrasonic test.\n\r");

    for( ;; )
    {
        // Delay for 1 second.
        vTaskDelay( x1second );
        Afstand = 0u;

        // Lees de gemeten afstand van de sensor in.
        Xil_Out32(ULTRASOON_ADDR + ULTRASOON_REG1, 0x00000001);
        Afstand = Xil_In32(ULTRASOON_ADDR + ULTRASOON_REG0);

        // Controleer of de afstand meetbaar is of niet.
        if(CHECKOOB(Afstand))
        {
            xil_printf("Object is too far away.\r\n");
        }

        // controle om de gemeten waarde te zien.
        xil_printf("Raw data: 0x%08x\r\n", Afstand);
        Xil_Out32(ULTRASOON_ADDR + ULTRASOON_REG1, 0x00000000);

        // Send the next value on the queue.  The queue should always be
        // empty at this point so a block time of 0 is used.
        xQueueSend( xQueue,          // The queue being written to.
                    &Afstand,        // The address of the data being sent.
                    0UL );           // The block time.
    }
}
```

Knop:

De functie van de knop bevat alles wat er nodig is om de knop te kunnen gebruiken in ons spel.
De knop dient om de kleur te bevestigen, die er gevonden is met de ultrasoon sensor.

1. De functie:
    a. We lezen de Knop in van onze FPGA omdat deze standaard op het bord stond.
        i. Configureren de pin en pindata.
    b. Controleren of de nieuwe data verschillend is van de oude data.
    c. Vervolgens zenden we data van de knop naar de neon pixel door gebruik te maken van een queue.
    d. De queue overwrite gebruiken we voor als er geen nieuwe data gedecteerd word.

```
static void prvKnop( void *pvParameters )
{
  u8 NewSwData = 0;
  u8 OldSwData = 0;
  int KnopValue;

    //const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND );
    //xil_printf("In Functie Knop.\n\r");

  XGpioPs_Config *ConfigPtr;
  ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
  XGpioPs_CfgInitialize(&Gpio, ConfigPtr, ConfigPtr -> BaseAddr);
  XGpioPs_SetDirectionPin(&Gpio, Knop, KnopData);

  for(;;)
    {
        NewSwData = XGpioPs_ReadPin(&Gpio, Knop);
        if((NewSwData != OldSwData) && (NewSwData == 1))
        {
            xil_printf("Button Pressed\r\n");
        }
        else if((NewSwData != OldSwData) && (NewSwData == 0))
        {
            xil_printf("Button Released\r\n");
            releaseButton = 1;
        }
        OldSwData = NewSwData;
        KnopValue = OldSwData;

    //xil_printf("Queue -> KnopWaarde: %d\n", KnopValue);


    xQueueOverwrite(xQueue3, //Als er iets in de queue zit, deze waarde overschrijven
                    &KnopValue);
    //xil_printf("Verzonden\n");
    }
}
```

prvRxTask:

in deze functie gaan we de waardes van de ultrasoon sensor uit een queue halen en gebruiken om de positie van speler 1 te bepalen en op de neonpixel aan te sturen.

1. Halen de data uit de queue die overeenkomt met de queue in de sensor functie. We printen de data uit om te controleren of de data overeenkomt met de data die we eerst hadden ingestuurd.
2. We voegen de game code toe, zodat de neonpixel automatisch geüpdatet word.
3. We voegen de functie toe waarmee we de positie van speler 1 bepalen in de game.
4. Er is ook een if else om te controleren of de data die we meten binnen het bereik van onze aftands grenzen van het spel licht. Indien dit waar is printen we groene leds en anders krijg je een error melding.

```
static void prvRxTask( void *pvParameters )
{
    xil_printf( "In prvRxTask \r\n" );
    int AfstandReceived;
    int KnopReceived;
    //int RGBvalue;
    /*int Temp;
    /* b10 ->   overschrijven

    for( ;; )
    {
        /*//Block to wait for data arriving on the queue.
        //xil_printf( "Knop waarde: %d \r\n", KnopReceived );

        // Block to wait for data arriving on the queue. */
        xQueueReceive(  xQueue,             /* The queue being read. */
                        &AfstandReceived,   /* Data is read into this address. */
                        portMAX_DELAY );    /* Wait without a timeout for data. */

        /* Print the received data. */
        xil_printf( "Afstand waarde: %d \r\n", AfstandReceived );

        // game toevoegen
        next_frame();
        set_player_position(AfstandReceived);


        /*NEONIP_mWriteReg(0x43c10000, NEON_REG0, 0b0);
        Temp = NEONIP_mReadReg(0x43c10000, NEON_REG0);*/

        if(AfstandReceived <= 15)
        {
            //groen in derde led
            NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, 0b00010000010);
            //klaarzetten in stage
            NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, 0b01010000010);
            NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, 0b00010000010);
            //buffer overschrijven
            NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, 0b10010000010);
            NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, 0b00010000010);
        }
        else if(AfstandReceived > 15)
        {
            xil_printf("Afstand te groot \r\n");
        }

        /*sleep_A9(1);
    }
}
```

vTimerCallback functie:

Deze functie word gebruikt om de matrix waardes te updaten.

1. We runnen de functie van de frames hier in zodat onze neon pixel altijd geüpdate word.
2. We voegen de pxTimer toe omdat deze ervoor zorgt de we pas om de 0.5 seconden gaan updaten wat het spel overzichtelijker maakt.
3. Er worden random kleuren gegenereerd om een level te maken.
4. En de afstand word bepaald met de sensor waardoor het speelbaar word.

```
static void vTimerCallback( TimerHandle_t pxTimer )
{
    //xil_printf( "0.5s voorbij. \r\n" );
    if ( InputToPlayer == 0 )
    {
        if( i_show_array < level )
        {
            int randomkleur = rand() % 3;
            array[i_show_array]= colors[randomkleur];
            NextColorScreen( array[ i_show_array ] );
            i_show_array++;
        }
        else
        {
            i_show_array = 0;
            InputToPlayer = 1;
        }
    }
    else
    {
        if( i_show_array < level )
        {
            set_player_color( AfstandReceived );
            if( releaseButton )
            {
                if( array[ i_show_array ] == player_color_buffer )
                {
                    i_show_array++;
                }
                else
                {
                    gameOver();
                }
            }
        }
        else
        {
            InputToPlayer = 0;
            i_show_array = 0;
            level++;
            xil_printf("LEVEL: %d\n", level );
        }
    }
    //next_frame;
    //drawArray();
    configASSERT( pxTimer );
}
```

Game:

Hier staan alle functies die betrekking hebben op het spelen van de game en de visualisatie daarvan.

Generate level:

Hier maken we het level dat geladen word in de neonpixel.

```
void generateLevel(int maxlength)
{
    for (int i= 0 ; i< maxlength; i++)
    {
        int randomkleur = rand() % 3;
        array[i]= colors[randomkleur];
    }
}
```

Next Color Screen:

Hier bepalen we de kleur van de volgende leds die moeten aangaan.

```
void NextColorScreen ( int kleur )
{
    for ( int i = 0; i< 8 ; i++ )
    {
        for ( int j = 0; j< 8 ; j++ )
        {
            drawScreen( i, j,kleur );
        }
    }
}
```

Draw Screen:

Hier bepalen we de kleuren van het spel, en we schrijven ze weg naar de neonpixel.

```
void drawScreen( int x, int y, int colorLED )
{
    /*positie is zero index based dus van 0-7 , 0-7
    //xil_printf("positieled: %d\n", positieLED);
    //xil_printf("colorled: %d\n", (uint )colorLED << 6);

    uint positieLED = y*8 + x;
    uint sturen = (uint) positieLED | ((uint) colorLED << 6);

    u32 send1 = 0b00000000000;
    u32 send2 = 0b01000000000;
    u32 send3 = 0b00000000000;
    u32 send4 = 0b10000000000;
    u32 send5 = 0b00000000000;

    //xil_printf("sturen: %x\n", sturen);

     NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, send1 | sturen);
     //xil_printf("send 1: %x\n", send1);
     NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, send2 | sturen);
     //xil_printf("%x\n", send2 | sturen);
     NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, send3| sturen);
     //xil_printf("%x\n", send3| sturen);
     NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, send4 | sturen);
     //xil_printf("%x\n",send4 | sturen);
     NEOMATIX64_mWriteReg(NEON_ADDR, NEON_REG0, send5 | sturen);
     //xil_printf("send 5: %x\n",send5 | sturen);
}
```

Clear Screen:

Maken we de volledige neonpixel leeg, om daarna een nieuwe kleur mee te geven.

```
void clearScreen(void)
{
    for(int i = 0; i< 8 ; i++)
    {
        for(int j =0; j<8 ; j++)
        {
            drawScreen(i,j,0);
        }
    }
}
```

Simon Says:

Dit bevat al de code om het spel te spelen met de sensor, knop op de neonpixel. En je kan controleren in een terminal welk level dat je al bereikt hebt.

```
void Simon_Says(int level)
{
    int i = 0;
    clearScreen();
    while(i < level)
    {

        //set_player_color(ultrasone);
        if(releaseButton && i <= level)
        {
            releaseButton = 0;
            if(array[i] == player_color_buffer)
            {
                i++;
                //check of einde bereikt is dan level up
                }
            else
            {
                gameOver();
                xil_printf("Je bent dood");
                break;
            }
        }
    }
    xil_printf("Goed zo, level up");
}
```

Set Player Color:

Hier definiëren we welke kleur het is als we een bepaalde afstand meten met de ultrasoon sensor. Om vervolgens te kunnen controleren of deze overeenkomt met het level.

```
void set_player_color( u32 positie )
{
    //static u32 lastPosition = 0;
    xil_printf("positie sturen : %d", positie);
    switch( positie )
  {
        case 0:
          // statements
            NextColorScreen(5);
            player_color_buffer = 5;
        break;
        case 1:
          // statements
              NextColorScreen(5);
              player_color_buffer = 5;
          break;
        case 2:
          // statements

            //player pos 1
                //y1 x0
            NextColorScreen(7);
            player_color_buffer = 7;
        break;
        case 3:
          // statements

          //player pos 1
            //y1 x0
              NextColorScreen(7);
              player_color_buffer = 7;
          break;
```

```
        case 4:

            //player pos 2
            NextColorScreen(3);
            player_color_buffer = 3;
        break;
        case 5:

          //player pos 2
              NextColorScreen(3);
              player_color_buffer = 3;
          break;
          case 6:
            NextColorScreen(2);
            player_color_buffer = 2;
        break;
        case 7:
              NextColorScreen(2);
              player_color_buffer = 2;
          break;

      default: break;
    }
}
```

Game Over:

Hier word bepaald wanneer een speler de game verloren heeft.

```
void gameOver(void)
{
    clearScreen();
    xil_printf("RIP you died... \n");
    level = 1;
}
```