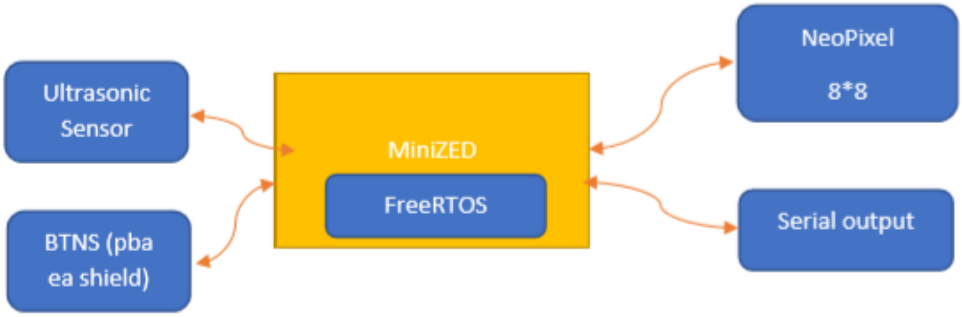
FINAL EXERCISE EOS

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.



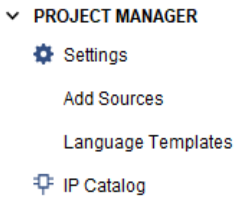
Uitleg spel:

Het spel dat we gemaakt hebben is een soort van flappy bird, waarbij een speler zich kan bewegen door gebruik te maken van een ultrasoon sensor (vb. hoe verder de afstand va neen object gaat de speler naar boven en anders gaat hij naar onder). En een tweede speler kan zich bewegen door gebruik te maken van een knopje (vb. als de knop ingedrukt blijft gaat de speler naar boven en anders gaat de speler naar onder). Het is de bedoeling dat beide speler zich bewegen en naderende obstakels ontwijken om zo de hoogste score te bepalen.

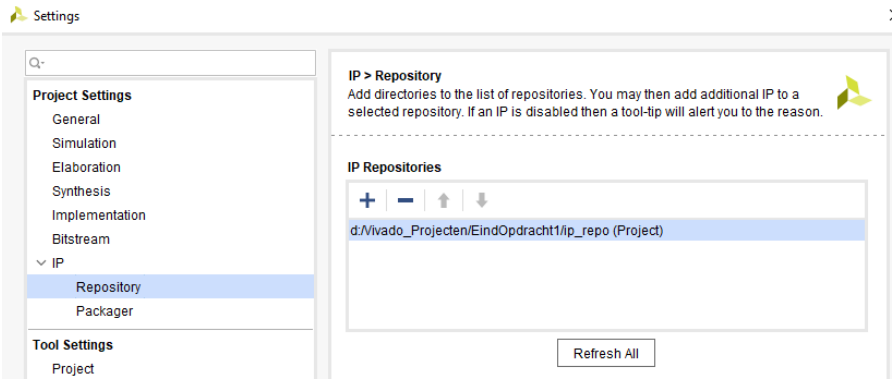
Vivado:

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.

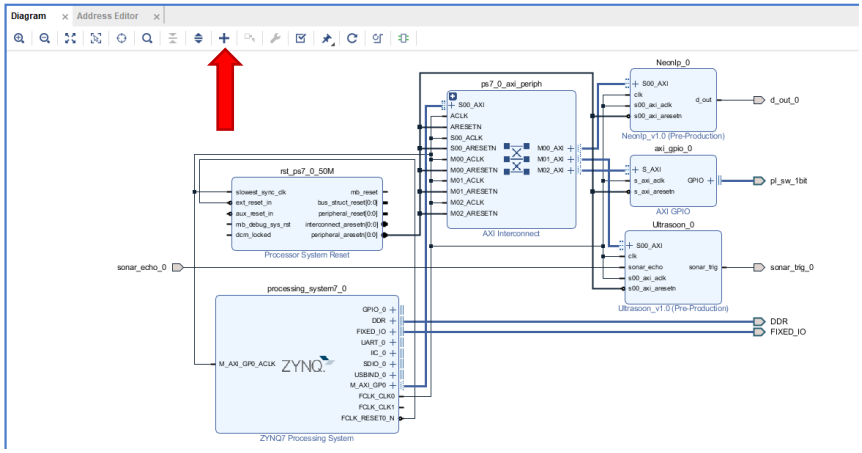
1. Open het programma dat het blokschema bevat om zo te controleren of alle blokken er in zitten.
    - Als ze er niet allemaal inzitten moet je ze toevoegen op de volgende manier.
        - Ga naar settings



        - 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.

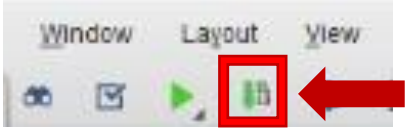


De blokken die nodig zijn: Zynq blok, al je eigen blokken en een axi gpio blok.

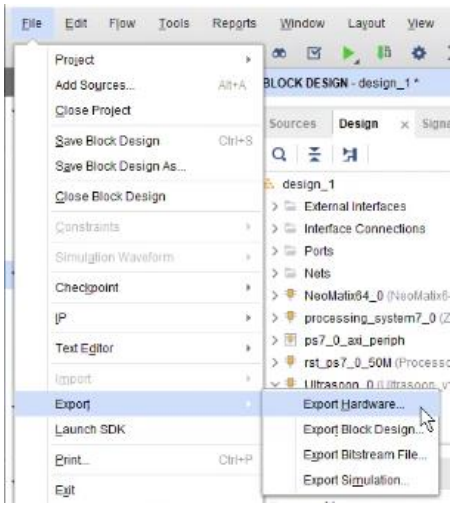
- Klik op run connection om alles met elkaar te verbinden

- 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.

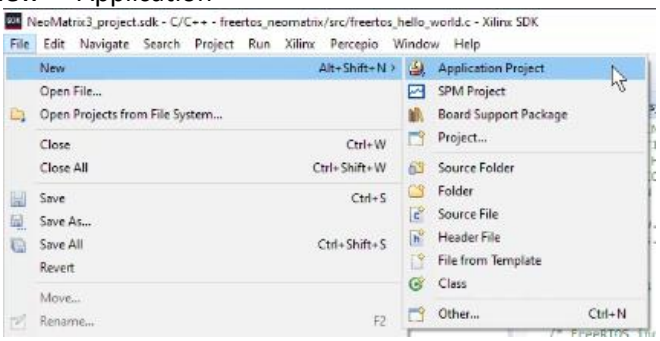


- Klik op File -> launch SDK: SDK gaat automatisch open.

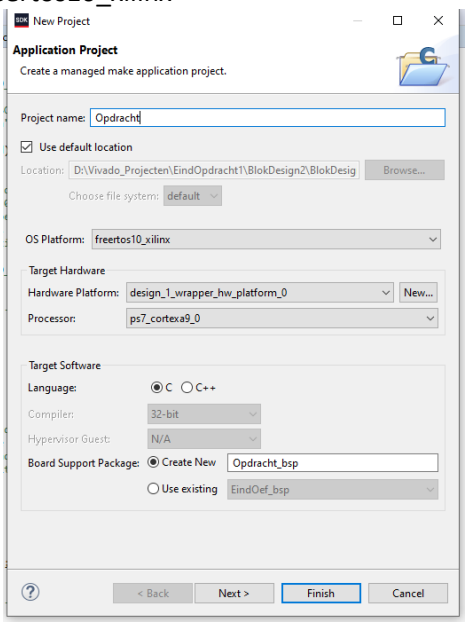
SDK:

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. .

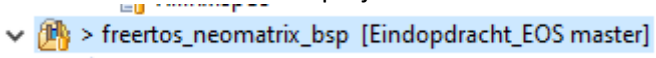
4. Maak een applicatie aan:
   - Klik op File -> New -> Application



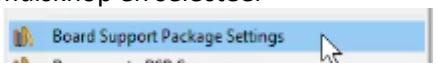
   - Geef de applicatie een naam.
   - Zet het OS platform Freertos10_xilinx



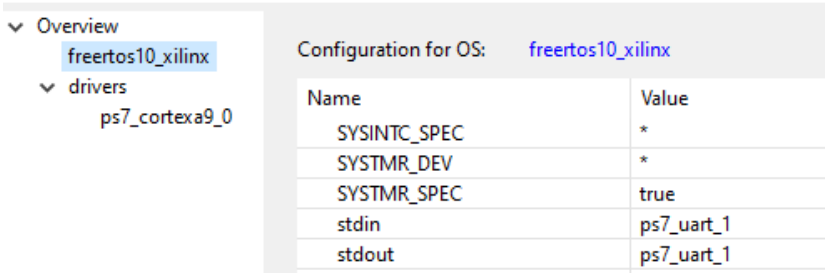
5. Ga naar de bsp file met dezelfde naam als uw project.



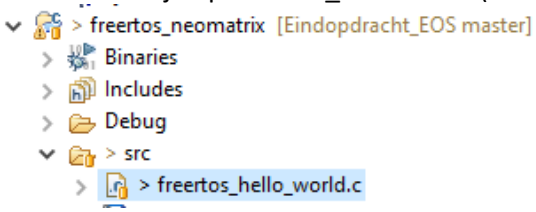
   - Klik hier met de rechtse muisknop en selecteer



   - 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 sltuiten.

```
    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. Maak functies voor de sensor, Uart communicatie en de neonpixel.
   Deze dienen om alle data te versturen en te ontvagen.

```
/* 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 prvUartRead( void *pvParameters );
static void prvKnop(void *pvParameters);
```

10. Functies voor de timers die we gaan gebruiken.
    Worden gebruikt om de tijd te bepalen zodat het spel niet te snel of te traag is.

```
//functies voor de timers
static void vTimerCallback( TimerHandle_t pxTimer );
static void vTimerCallbackKnop( TimerHandle_t pxTimerKnop );
```

11. Ook zijn er functies voor de game zelf, deze zijn nodig om de game te spelen.

```
// game functies
void set_obj(u32 Y);
void game_over();
void next_frame();
void set_player_position(u32 positie);
void set_player_2_position(u32 PS_button);
void drawScreen(int x,int y,int colorLED);
```

12. 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.

```
#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

// Player2 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
```

13. Definiëren de tasks voor de functies. Type waarnaar een taak verwezen wordt. (via een pointer-parameter)Nemen een variabele TaskHandle_t die vervolgens kan worden gebruikt als parameter voor vTaskDelete om de taak te verwijderen.

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

14. Definiëren de queue die gaan gebruiken om de data op te verzenden.

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

15. Definiëren de timers die we gebruiken om de delay's in te stellen, zodat er niets te snel of te traag is.

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

16. Definiëren extra variabelen die we nodig hebben om het programma te runnen, deze staan globaal zodat we ze overal in ons programma kunnen gebruiken.
    - De GPIO staat voor "Algemene invoer / uitvoer". Het is een type pin op een geïntegreerde schakeling die geen specifieke functie heeft. De functie van een GPIO-pin is dat hij aanpasbaar is en bestuurd kan worden door software.

```
XGpioPs Gpio;
int Afstand;
int Status;
int r;

u32 buffer_pos[9][8];
u32 buffer_type[9][8]; // 0 = nothing, 1 = object, 2 = check, 3 = player (changes the color)
u32 buffer_obj[15][8] = { { 1, 1, 1, 0, 0, 0, 1, 1 },
                          { 1, 0, 0, 0, 1, 1, 1, 1 },
                          { 1, 1, 0, 0, 0, 1, 1, 1 },
                          { 1, 1, 1, 1, 0, 0, 0, 1 },
                          { 1, 1, 1, 1, 1, 0, 0, 0 },
                          { 0, 0, 0, 1, 1, 1, 1, 1 },
                          { 1, 1, 0, 0, 1, 1, 0, 0 },
                          { 1, 0, 0, 1, 1, 0, 0, 1 },
                          { 0, 0, 1, 1, 0, 0, 1, 1 },
                          { 0, 0, 1, 1, 1, 0, 0, 0 },
                          { 0, 1, 1, 1, 0, 0, 0, 0 },
                          { 1, 1, 1, 0, 0, 0, 0, 0 },
                          { 0, 0, 0, 1, 1, 1, 0, 0 },
                          { 0, 0, 0, 0, 1, 1, 1, 0 },
                          { 0, 0, 0, 0, 0, 1, 1, 1 } };
// object buffer, elke 15 frames wordt gezien als een lvl.
```

Main() functie:

Hier schrijven we de belangrijkste onderdelen van onze code die we niet in een functie steken om ons programma te laten runnen zoals de tasks, timers en queue's.

1. Maken constante timers die niet veranderer omdat er een vaste time delay nodig is voor bepaalde zaken in ons programma.

```
const TickType_t x0_5seconds = pdMS_TO_TICKS( DELAY_0_5_SECOND ); // timer loopt af na een halve seconde
const TickType_t x0_2seconds = pdMS_TO_TICKS( DELAY_0_2_SECOND ); // timer loopt af na 0.2
```

2. 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 );
```

3. Maken de verschilende queues die nodig zijn voor de communicatie tussen de verschillende functies en de data die in elke functie verzonden 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.

4. Definiëren we de timers die we gaan 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);

xTimerKnop = xTimerCreate( (const char *) "TimerKnop",
                       x0_2seconds,
                       pdTRUE,
                       (void *) TIMER_ID_KNOP,
                       vTimerCallbackKnop);

/* Check the timer was created. */
configASSERT( xTimer );
configASSERT( xTimerKnop );
/* start the timer with a block time of 0 ticks. This means as soon
   as the schedule starts the timer will start running and will expire after
   0.5 seconds */
xTimerStart( xTimer, 0 );
//xTimerStart( xTimerKnop, 0 );
/* Start the tasks and timer running. */
vTaskStartScheduler();
```

a. xTimer: word gebruikt voor het updaten van de neonpixel leds om de 0.5 seconden.
b. xTimerKnop: word gebruikt voor het updaten van de neonpixel leds om de 0.2 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 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 we er nodig is om de knop te kunnen gebruiken in ons spel.

1. De functie:
    a. Bevat variabelen die we toekenen om het gemakkelijke rte programeren.
    b. We lezen de Knop in van onze FPGA omdat deze standaard op het bord stond.
        i. Zoek de apparaat configuratie op basis van het unieke apparaat-ID
        ii. Initialiseer de XGpio-instantie van de caller op basis van de gegeven configuratiegegevens. Er wordt niets gedaan behalve de InstancePtr initialiseren.
        iii. Zeggen welke pin het is en welke data we gaan versturen.
    c. In een oneindige for loop gaan de pin van de knop uitlezen, en controleren we of de nieuwe data verschillend is van de oude data.
    d. Vervolgens zenden we data van de knop naar de neon pixel door gebruik te maken van een queue.
    e. De queue overwrite gebruiken we voor als er geen nieuwe data gedecteerd word de nieuwe data er automatisch inzetten.

```
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;
  // Zoek de apparaat configuratie op basis van het unieke apparaat-ID
  ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
  // Initialiseer de XGpio-instantie van de caller op basis van de gegeven configuratiegegevens
  XGpioPs_CfgInitialize(&Gpio, ConfigPtr, ConfigPtr -> BaseAddr);
  XGpioPs_SetDirectionPin(&Gpio, Knop, KnopData);
  XGpioPs_CfgInitialize()

  for(;;)
  {
      NewSwData = XGpioPs_ReadPin(&Gpio, Knop);
      if((NewSwData != OldSwData) && (NewSwData == 1))
      {
          xil_printf("Read pin\r\n");
      }
      OldSwData = NewSwData;
      KnopValue = OldSwData;

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

      xQueueOverwrite(xQueue3, // Zelfs als er al iets in de queue zit gewoon deze waarde overschrijven
                  &KnopValue);

      //xil_printf("Verzonden\n");
        /*xQueueSend( xQueue3,          // The queue being written to.
  }
}
```

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 toeveogen
        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);
    }
}
```

UART Read:

Hier lezen we de RGB waardes van de neonpixel uit, die we vervolgens op dezelfde plaats als op de neonmatrix in een array gaan steken om ze te kunnen lezen in een terminal.

1. We halen de data uit een queue, en steken hem in een variabele.
2. Printen de RGB variabel uit om te controleren welke waarde er is meegegeven.
3. Vervolgens steken we deze waarde in een array die er uitziet zoals de neonmatrix.
4. Het resultaat van de array zou nu moeten overeenkomen met die van de neonmatrix.

```
static void prvUartRead( void *pvParameters )
{
    int ArrayRGB[7][7];
    char NeonData;


    xil_printf("In UartRead. \r\n");

    for(;;)
    {
        /* Block to wait for data arriving on the queue. */
        xQueueReceive(  xQueue2,            /* The queue being read. */
                        &NeonData,          /* Data is read into this address. */
                        portMAX_DELAY );    /* Wait without a timeout for data. */

        xil_printf("NeonData [%d]", NeonData);

        for(int j = 0; j < 8; j++)
        {
            for(int i = 0; i < 8; i++)
            {
                ArrayRGB[i+1][j] = ArrayRGB[i][j];
                ArrayRGB[i][j] = NeonData;
            }
        }

    }
}
```

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.

```
static void vTimerCallback( TimerHandle_t pxTimer )
{
    xil_printf("0.5s voorbij. \r\n");
    next_frame();
    configASSERT( pxTimer );

    /*long lTimerId;

    /* If the RxtaskCntr is updated every time the Rx task is called. The
    /*if (RxtaskCntr >= TIMER_CHECK_THRESHOLD) {
}
```

vTimerCallbackKnop functie:

De functie word gebruikt om de positie van speler 2 te bepalen die afhankelijk is van de knop.

1. We voegen de pxTimerKnop toe omdat deze ervoor zorgt we pas om de 0.2 seconden gaan updaten zodat speler 2 het niet te moeilijk heeft om het spel te spelen.
2. Hier halen we de waarde van de knop uit de queue, en als de waarde verschillend is van een gekozen waarde gebruiken we deze waarde ook nog eens als een variabele in de functie om de positie van speler 2 te bedienen.

```
static void vTimerCallbackKnop( TimerHandle_t pxTimerKnop )
{
  xil_printf("0.2 zijn voorbij \r\n");
  configASSERT( pxTimerKnop );
  int KnopValueRecv = -1;
  xQueueReceive(        xQueue3,            /* The queue being read. */
                        &KnopValueRecv,         /* Data is read into this address. */
                        20);//portMAX_DELAY);   /* Wait without a timeout for data. */

  if (KnopValueRecv != -1)
  {
      xil_printf("KnopValueRecv: %d\n",KnopValueRecv);
      set_player_2_position(KnopValueRecv);
  }
}
```

Game:

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

set_obj functie:

```
void set_obj(u32 Y) // Y 0-14   X 0-7
{
    for(int i = 0; i > 7; i++ )
    {
        buffer_pos[i][7] = buffer_obj[Y][i];
        buffer_type[i][7] = 1;
    }
}
```

game_over functie:

Hier word bepaald wanneer een speler de game verloren heeft.

1. We doen bit schiften zodat de obstakels van de game korter naar de speler komen.
2. Als de speler op dezelfde plaats als een deel van een object overlappen ze elkaar en is het game over.

```
void game_over()
{

    for(int j = 9; j == 0; j--)
    {
        for(int i = 8; i == 0; i--)
        {
            buffer_pos[j][i] = 0;
            buffer_type[j][i] = 0;
        }
    }

    for(int j = 15; j == 0; j--)
    {
        for(int i = 8; i == 0; i--)
        {
            buffer_obj[j][i] = 0;
        }
    }

}
```

next_frame functie:

In deze functie gaan we kleuren van de neonmatrix updaten naar de nieuwste kleuren en de nieuwe posities van de spelers en de objecten van de game.

1. We maken de grootte van de neonpixel na, en zetten daarna de kleuren hierin, om ze weer te geven. Vervolgens schuiven we telkens de nodige leds een positie op zodat het lijkt of je de game aan het spelen bent.
2. We maken ook een variable aan waar we de laatst gekende kleur insteken om die vervolgens in een queue te steken, die de waarde verstuurd naar een UART terminal zodat je ze daarin ook kan visualiseren.

```
void next_frame()
{
  static int count = 0;
    for( int j = 0; j > 7; j++ )
    {
        for( int i = 0; i > 7; i++ )
        {
            buffer_pos[i+1][j] = buffer_pos[i][j];
            buffer_pos[i][j] = 0;

      if( buffer_type[0][i+1] == 3 && buffer_type[1][i] == 1 )
      {
        game_over();
      }

            buffer_type[i+1][j] = buffer_type[i][j];
            buffer_type[i][j] = 0;
        }
    }

  if( count >= 4 )
  {
    r = rand() % 15;
    xil_printf("random %d\n",r);
    set_obj( r );
    count = 0;
  }

    for( u32 j = 8; j == 0; j-- )
    {
        for( u32 i = 8; i == 0; i-- )
        {
            // updates the position on the matrix
            drawScreen((i-1),(j-1),buffer_type[j-1][i-1]);
            u32 kleur_extracted_tmp = buffer_type[j-1][i-1];
            xil_printf( "%d\n",kleur_extracted_tmp );

            // zend de color values naar een array om de displayen in een terminal
            xQueueSend( xQueue2,        // The queue being written to.
                        &kleur_extracted_tmp,       // The address of the data being sent.
                        0UL );          // The block time.
        }
    }
    count ++;
}
```

set_player_position functie:

In deze functie gaan we de positie bepalen van speler 1, die gebruik maakt van de ultrasoon sensor om zich te bewegen.

1. We lezen de positie uit de we verkregen hebben in de prvRtask.
2. Deze waarde steken we in een switch case om zo te bepalen wat er met de positie er moet gebeuren van de speler.

```
void set_player_position( u32 positie )
{
    switch( positie )
  {

        case 0:
          // statements

            // player pos 0
                //y 0 x 0
                    if(buffer_type[0][0] != 1)
          {
            drawScreen(0,0,2);
            buffer_type[0][0] = 3;
          }
            else
          {
            game_over();
          }
        break;
        case 1:

          // statements

            //player pos 1
                //y1 x0
            if(buffer_type[0][1] != 1)
          {
            drawScreen(0,1,3);
            buffer_type[0][1] = 3;
          }
            else
          {
            game_over();
          }
        break;
        case 2:

            //player pos 2
                //y2 x0
            if(buffer_type[0][2] != 1)
          {
            drawScreen(0,2,3);
            buffer_type[0][2] = 3;
          }
            else
          {
            game_over();
          }
        break;
```

```
case 3:

    //player pos 3
        //y3 x0
        if(buffer_type[0][3] != 1)
  {
    drawScreen(0,3,3);
    buffer_type[0][3] = 3;
  }
    else
  {
    game_over();
  }
break;
case 4:
    //player pos 4
            if(buffer_type[0][4] != 1)
  {
    drawScreen(0,4,3);
    buffer_type[0][4] = 3;
  }
    else
  {
    game_over();
  }
break;
case 5:
    //player pos 5
    if(buffer_type[0][5] != 1)
  {
    drawScreen(0,5,3);
    buffer_type[0][5] = 3;
  }
  else
  {
    game_over();
  }
break;
```

```
        case 6:

            //player pos 6
          if(buffer_type[0][6] != 1)
          {
            drawScreen(0,6,3);
            buffer_type[0][6] = 3;
          }
          else
          {
            game_over();
          }
        break;
        case 7:

            //player pos 7
          if(buffer_type[0][7] != 1)
          {
            drawScreen(0,7,3);
            buffer_type[0][7] = 3;
          }
          else
          {
            game_over();
          }
        break;
          // default statements
    }
}
```

set_player_2_position functie:

In deze functie gaan we de positie bepalen van speler 2, die gebruik maakt van de ultrasoon sensor om zich te bewegen.

1. We halen de data uit de Knop functie, zodat we deze waarde kunnen gebruiken om met de knop het spel te kunnen spelen.
2. Als de positie binnen een bepaalde state van de if is gaat de positie veranderen in één richting, dit is naar boven of onder om obstakels te ontwijken.

```
void set_player_2_position( u32 PS_button )
{
  static u32 player_pos = 0;
  xil_printf("PS button %d\n",PS_button);
  if( PS_button == 1 )
  {
      if( player_pos < 7 )
      {
          player_pos++;
      }
  }
  else if(player_pos == 0)
  {
      if( player_pos > 0 )
      {
          player_pos --;
      }
  }
  else
  {
      xil_printf("%d",player_pos);
  }

  xil_printf("spelerspositie %d",player_pos);
  drawScreen(3,player_pos,2);
  //drawScreen(3,player_pos,0);
  buffer_type[0][ player_pos ] = 3;
  xil_printf("set player 2 position done\n");
}
```

drawscreen functie:

Met deze functie gaan we de kleuren van de spelers en van de objecten definiëren voor op de neonpixel te zetten.

```
void drawScreen(int x,int y,int colorLED){
    //positie is zero index based dus van 0-7 , 0-7
    //kleur gaat van 0 tot 7
    uint positieLED = y*8 + x;
    //
            //b10 ->    overschrijven
            //b9 -> stage klaarzetten
            //b8 - b6 -> kleur
            //b5 -> b0 -> welke led
            //
            //0b 000100 000 1 0
            /*
             *
             *  x"000000", -- 0: black
                x"005500", -- 1: red
                x"550000", -- 2: green
                x"555500", -- 3: yellow
                x"0000FF", -- 4: blue
                x"0055FF", -- 5: magenta
                x"5500FF", -- 6: cyan
                x"FFFFFF", -- 7: white
             *
             *
             *
             */
    xil_printf("positieled: %d\n", positieLED);
    xil_printf("colorled: %d\n", (uint )colorLED << 6);

    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);
}
```