Background

In this project, we will design a complete embedded system using Xilinx Vivado and Vitis. The system is built around a MicroBlaze soft-core processor and uses standard AXI-based IP blocks to perform peripheral communication.

The project integrates three main components:

- An AXI Timer that generates periodic interrupts to keep time and control logic timing.
- An AXI GPIO module that connects to external switches (inputs) and LEDs (outputs).
- An Interrupt Controller (AXI INTC) that receives interrupt requests from the timer and forwards them to the processor.

The goal is to create a real-time embedded application that:

- Controls LED behavior based on the state of input switches.
- Displays a live time counter (hours, minutes, seconds) over UART.
- Demonstrates interrupt-driven design using AXI peripherals.

This design highlights the integration of hardware (in Vivado) and software (in Vitis), and showcases how AXI IP blocks and interrupts can be combined to build interactive, time-aware FPGA applications.

What You Will Use in the HW Design :

Tool/Component	Purpose
Vivado	To design the hardware system using IP blocks (Block Design) and generate the bitstream
Vitis	To write and deploy embedded C code to the MicroBlaze processor
MicroBlaze V	Xilinx's soft-core processor, used to run the application code
AXI Timer	To generate regular timed interrupts and drive system timing
AXI GPIO	To read input from switches and control output to LEDs
Interrupt Controller (AXI INTC)	To manage and handle interrupt signals (e.g., from the timer) and notify the processor
Interrupts	Enable event-driven execution, allowing the system to respond immediately when the timer expires
UART	To view time updates, debug messages, and system output over serial connection

Let's start!

Step 1: Create a New Project in Vivado

◈ Launch Vivado

◈ Create a New Project as usual

◈ Project Name and Location

- **Project Name**: Choose a name like axi_timer_project
- **Project Location**: Choose a workspace directory you can easily find
- Click **Next**

◈ Project Type

- Select **"RTL Project"**
- Check **"Do not specify sources at this time"**
- Click **Next**

◈ Default Part or Board Selection

Now select your FPGA target: **Arty s7- 50**

Option A: Using a board (recommended)

- Click on **"Boards"**
- Filter by vendor: digilent
- Choose your board (e.g. Arty S7-50)

Click **Next**, then **Finish**

◈ Change the programming language to VHDL

Step 2: Create the Block Design

◈ Open the Block Design Tool

In the **Flow Navigator** on the left, go to:
IP Integrator → Create Block Design

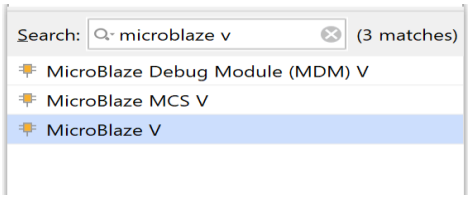
◈ Name the Block Design

Give your block design a name, for example: system_design
Click **OK** to proceed.

Vivado now opens the Block Design canvas.

◈ Add the MicroBlaze V Processor

Click **"Add IP"** or right-click on the canvas and choose **Add IP**
Search for MicroBlaze v and double-click it to add it to the canvas.

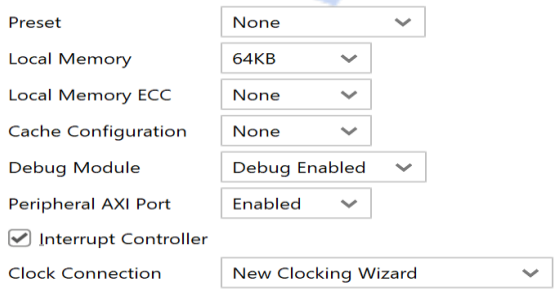


◈ Run Block Automation

After MicroBlaze appears on the canvas, click the **"Run Block Automation"** banner or button.

In the dialog that appears:

- Use the default settings
- Ensure **Local Memory** is selected and choose 32 or 64 KB to get more memory
- Optionally enable **Debug Module** if needed
- Check on the **interrupt controller**
- Choose **New Clocking Wizard**

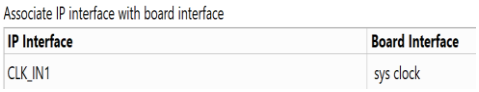


Click **OK**.
Vivado will now add required components such as:

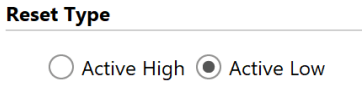
- AXI Interconnect
- Local Memory Block (BRAM)
- Clocking Wizard (if not already present)

◈ adjust clk and reset

- Dubbel click on **CLC_WIZ** to change the settings of the clock and reset**.**
- In the **Board Interface** choose the **Sys clock** for **IP Interface CLK_IN1**



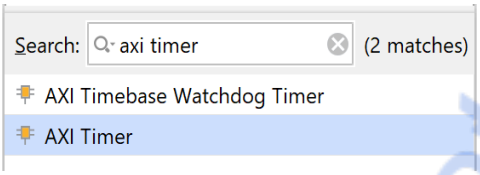
- Click **Outputs Clocks** and then put the reset on **Active Low**
  Active Low Reset:
  The reset signal on the Arty S7-50 is active low, meaning that the reset is asserted when the signal is low and deasserted when it is high.



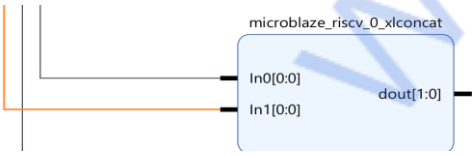
Now click on **Run Connection Automation** to connect the clock and reset with the processor.

- Right click on the reset and clk pin and choose make external.

◈ Add Axi timer

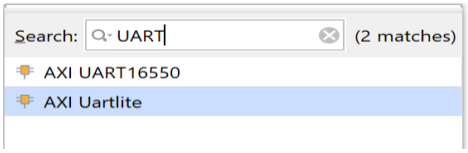


- Connect the Interrupt on Axi Timer to the interrupt xlconact block

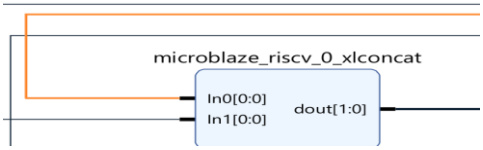


And again, click on **Run Connection Automation.**

◈ Add UART



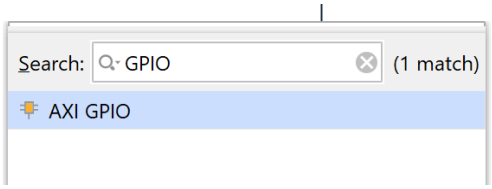
- Connect the Interrupt on UART to the interrupt xlconact block



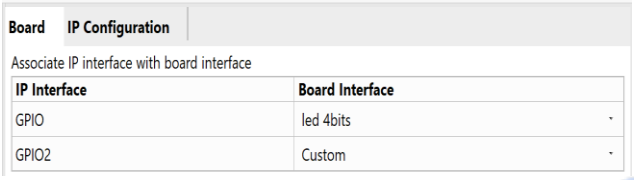
Now click on **Run Connection Automation** to connect the UART with

◈ Add Axi GPIO

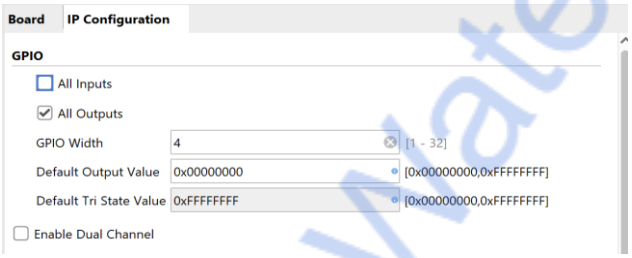
- Click **"Add IP" and write Axi GPIO** then add one or more**,** optional!



- You can neither use the same **GPIO** for both inputs and outputs, nor add multiple GPIOs and specify each block to a specific interface on the board.
- Choose the interface bord for the IP Interface (in this case you cant change the width oof the interface).



- If you want to edit and change the width and the interface, then you need to keep it in **Custom.** Then click on **IP Configuration** and you will see you can see that now you could change as you wish.



- Click on **Run Connection Automation** and Validate the design**.**

◈ Save the Design

Click the **save icon** or press **Ctrl + S** to save your design.
It's a good habit to save often during your progress.

- Generate Bitstream file and export the hardware **xsa** file.

Step 3: Creation of a Platform and Application in Vitis

◈ Open Vitis and choose or create a new Workspace.

◈ Create a platform for the project and choose the hardware file (xsa) that you exported.

◈ After you build the Platform go ahead and select HelloWorld template and create a new application.

◈ C code and Project Description – FPGA LED Timer Controller

This program is designed to run on an FPGA board using a MicroBlaze or RISC-V softcore processor. The goal is to control LEDs using switches and a timer interrupt. Depending on the switch settings, different LED patterns are activated. The code uses peripherals like GPIOs, timers, and interrupts provided by Xilinx.

Main Functionality

The application continuously reads the state of 4 switches. Based on which switch is active (SW0 to SW3), it performs one of the following LED behaviors:

1. **Slow Blink (SW0): Toggles LEDs every 1 second (1 Hz)**
2. **Fast Blink (SW1): Toggles LEDs every 0.25 seconds (4 Hz)**
3. **Sequence Pattern (SW2): Shifts a single lit LED back and forth**
4. **Binary Count (SW3): Counts from 0 to 15 in binary**

It also keeps track of time in hours, minutes, and seconds, and prints it to the terminal once every second.

Key Modules and Concepts

1. Hardware Definitions

The program begins by importing hardware-specific libraries and defining the base addresses for GPIOs, timer, and interrupt controller. These addresses come from the xparameters.h file, generated by the Vivado hardware platform.

2. GPIO Initialization

- **Switch GPIO is set as input.**
- **LED GPIO is set as output.**
  This allows reading switch states and controlling LEDs.

3. Timer Setup

A timer (XTmrCtr) is configured with:

- **Auto-reload: It automatically resets after reaching zero.**
- **Interrupt mode: Triggers an interrupt when the timer expires.**
  The reset value of the timer is set based on the selected mode (e.g., fast or slow blink).

4. Interrupt Controller Setup

The interrupt controller (XIntc) is configured to:

- **Listen to the timer interrupt.**
- **Call the TimerISR() function every time the timer expires.**
- **CPU-level exceptions are enabled to allow interrupt handling.**

5. Timer Interrupt Service Routine (ISR)

The TimerISR() function sets a global flag timerExpired = 1 and resets the timer. This flag is polled in the main loop to know when it's time to update LEDs and timekeeping.

6. LED Pattern Logic

The update_led_pattern() function updates the LED behavior depending on which switch is active:

- **Blink modes toggle the lower 4 LEDs.**
- **Sequence mode shifts a bit left and right to create a moving light.**
- **Binary count mode increments the LED output from 0 to 15.**

It also adjusts the timer speed for each mode.

7. Timekeeping

The function timer_hms_tick() is called every time the timer has expired twice (i.e., once per second). It increments seconds, minutes, and hours accordingly and prints the time live to the terminal using xil_printf().

Program Flow in main()

1. **Initialization phase:**
   - **Platform init (UART etc.)**
   - **GPIOs for switch and LED are set up.**
   - **Timer and interrupt controller are initialized and started.**
2. **User information output:**
   - **Welcome messages and instructions are printed via UART.**
3. **Infinite main loop:**

- **Reads switch state (note: the switches are active-low).**
- **Sets the correct mode based on which switch is active.**
- **Waits for the timer interrupt (polls the timerExpired flag).**
- **When the timer expires:**
  - **Updates the LED pattern based on mode.**
  - **Writes new pattern to the LEDs.**
  - **Recalculates and reloads the timer speed if needed.**
  - **Updates and prints current time.**
  - **Resets the timerExpired flag.**

Conclusion

This program is an interactive, interrupt-driven LED controller that demonstrates:

- **Using GPIOs for inputs and outputs**
- **Configuring and using a hardware timer**
- **Handling interrupts**
- **Basic timekeeping**
- **Real-time user interaction through hardware switches**

◈Demonstration and Video on testein on Arty S7- 50

- https://www.youtube.com/watch?v=j9NKezEkaWw
- **https://www.youtube.com/shorts/IcRadAvstR8**

C Code:

```
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xgpio.h"
#include "xtmrctr.h"
#include "xintc.h"
#include "xil_exception.h"

// Base addresses from xparameters.h (defined by the hardware design)
#define GPIO_SW_BASEADDR XPAR_AXI_GPIO_0_BASEADDR       // Switch GPIO base address
#define GPIO_LED_BASEADDR XPAR_AXI_GPIO_1_BASEADDR      // LED GPIO base address
#define TMR_BASEADDR XPAR_AXI_TIMER_0_BASEADDR          // Timer base address
#define INTC_BASEADDR XPAR_MICROBLAZE_RISCV_0_AXI_INTC_BASEADDR // Interrupt controller base address
#define TMR_INTR_ID XPAR_FABRIC_AXI_TIMER_0_INTR        // Timer interrupt ID

```

```c
#define SW_CHANNEL 1     // Switch GPIO channel
#define LED_CHANNEL 1    // LED GPIO channel

// Switch modes - each switch activates a specific LED pattern mode
#define MODE_NONE         0x0
#define MODE_BLINK_SLOW   0x1
#define MODE_BLINK_FAST   0x2
#define MODE_SEQUENCE     0x4
#define MODE_BINARY_COUNT 0x8

// Global peripheral instances
XGpio GpioSw, GpioLed;
XTmrCtr Timer;
XIntc InterruptController;

// Global control variables
volatile int timerExpired = 0;           // Flag set by the timer ISR
volatile u32 led_pattern = 0x1;          // Current LED pattern
volatile u32 timer_speed = 50000000;     // Default timer speed (0.5 seconds)

// Timekeeping variables
u32 hours = 0;
u32 minutes = 0;
u32 seconds = 0;
u32 tick_counter = 0;

// Function that updates and prints the current time once every second
void timer_hms_tick()
{
   tick_counter++;

   if (tick_counter >= 2) {  // 2 * 0.5s = 1 second
      tick_counter = 0;
      seconds++;

      if (seconds >= 60) {
         seconds = 0;
         minutes++;
         if (minutes >= 60) {
            minutes = 0;
            hours++;
         }
      }

      // Print current time in-place on the terminal
      xil_printf("\rTime -> Hours: %02d  Minutes: %02d  Seconds: %02d", hours, minutes, seconds);
      fflush(stdout);
   }
}

// Timer Interrupt Service Routine (ISR)
void TimerISR(void *InstancePtr)
{
   XTmrCtr *TimerInstancePtr = (XTmrCtr *)InstancePtr;

   if (XTmrCtr_IsExpired(TimerInstancePtr, 0)) {
      timerExpired = 1;                  // Signal that the timer has expired
      XTmrCtr_Reset(TimerInstancePtr, 0);  // Reset the timer for the next interval
   }
}

// Function to update LED pattern based on current mode (set by switch input)
void update_led_pattern(u32 switch_mode)
{
   static int direction = 1;  // Used for back-and-forth LED shifting

   switch (switch_mode) {
      case MODE_BLINK_SLOW:
         led_pattern ^= 0xF;            // Toggle lower 4 bits
         timer_speed = 100000000;       // 1 second
         break;

      case MODE_BLINK_FAST:
         led_pattern ^= 0xF;            // Toggle lower 4 bits
         timer_speed = 25000000;        // 0.25 seconds
         break;

      case MODE_SEQUENCE:
         if (direction) {
            led_pattern <<= 1;          // Shift left
            if (led_pattern & 0x10) {   // If overflow, reverse direction
               led_pattern = 0x8;
               direction = 0;
            }
         } else {
            led_pattern >>= 1;          // Shift right
            if (led_pattern == 0) {
               led_pattern = 0x1;
               direction = 1;
            }
         }
         timer_speed = 50000000;        // 0.5 seconds
         break;

      case MODE_BINARY_COUNT:
         led_pattern = (led_pattern + 1) & 0xF; // Count 0–15 in binary
         timer_speed = 50000000;                // 0.5 seconds
         break;

      default:
         led_pattern = 0x0;             // Turn off LEDs
```

```
            timer_speed = 50000000;         // Default speed
            break;
    }
}

int main()
{
    u32 switch_state, switch_mode;

    init_platform(); // Initialize UART and low-level hardware

    // Initialize GPIO for switches (input)
    XGpio_Initialize(&GpioSw, GPIO_SW_BASEADDR);
    XGpio_SetDataDirection(&GpioSw, SW_CHANNEL, 0xF); // All 4 bits as input

    // Initialize GPIO for LEDs (output)
    XGpio_Initialize(&GpioLed, GPIO_LED_BASEADDR);
    XGpio_SetDataDirection(&GpioLed, LED_CHANNEL, 0x0); // All 4 bits as output

    // Initialize Timer with auto-reload and interrupt mode
    XTmrCtr_Initialize(&Timer, TMR_BASEADDR);
    XTmrCtr_SetOptions(&Timer, 0, XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);
    XTmrCtr_SetResetValue(&Timer, 0, 0xFFFFFFFF - timer_speed); // Configure initial timer value

    // Initialize and configure the interrupt controller
    XIntc_Initialize(&InterruptController, INTC_BASEADDR);
    XIntc_Connect(&InterruptController, TMR_INTR_ID, (XInterruptHandler)TimerISR, &Timer);
    XIntc_Enable(&InterruptController, TMR_INTR_ID);
    XIntc_Start(&InterruptController, XIN_REAL_MODE);

    // Enable CPU-level interrupts
    Xil_ExceptionInit();
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
       (Xil_ExceptionHandler)XIntc_InterruptHandler,
       &InterruptController);
    Xil_ExceptionEnable();

    // Start the timer
    XTmrCtr_Start(&Timer, 0);

    // Welcome and instructions
    xil_printf("Welcome To FPGA-World\r\n");
    xil_printf("Engineer : Mnyar Hees\r\n");

    xil_printf("\r\nLED Timer Controller Ready\r\n");
    xil_printf("SW0: Slow Blink (1Hz)\r\n");
    xil_printf("SW1: Fast Blink (4Hz)\r\n");
    xil_printf("SW2: Sequence Pattern\r\n");
    xil_printf("SW3: Binary Count\r\n");

    xil_printf("Time -> Hours: 00  Minutes: 00  Seconds: 00");

    // Main loop
    while (1) {
        // Read the 4 switches (active-low, so invert result)
        switch_state = ~XGpio_DiscreteRead(&GpioSw, SW_CHANNEL) & 0xF;

        // Set mode based on lowest active switch
        if (switch_state & 0x1) {
            switch_mode = MODE_BLINK_SLOW;
        } else if (switch_state & 0x2) {
            switch_mode = MODE_BLINK_FAST;
        } else if (switch_state & 0x4) {
            switch_mode = MODE_SEQUENCE;
        } else if (switch_state & 0x8) {
            switch_mode = MODE_BINARY_COUNT;
        } else {
            switch_mode = MODE_NONE;
        }

        // If timer expired (flag set by ISR)
        if (timerExpired) {
            update_led_pattern(switch_mode); // Update pattern based on selected mode
            XGpio_DiscreteWrite(&GpioLed, LED_CHANNEL, led_pattern); // Write pattern to LEDs

            // Update timer value if speed changed
            XTmrCtr_SetResetValue(&Timer, 0, 0xFFFFFFFF - timer_speed);
            XTmrCtr_Reset(&Timer, 0);

            // Update time and print
            timer_hms_tick();

            timerExpired = 0; // Reset flag
        }
    }

    cleanup_platform(); // Optional cleanup (not usually reached)
    return XST_SUCCESS;
}
```