**APRIL 2023, 4 BY ACKYE**

# Smart ZYNQ (SP&SL version) Engineering XVIII is based on timer interrupts on the PS side

Like all MCUs, the PS side of ZYNQ has its own timer interrupt function, which will be demonstrated in this section.

**(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)**

Timer interrupts can provide the system with a standard clock, heartbeat, and can also be used as a time reference for some calculations (such as pulse width, pulse period, etc.) are essential in real projects.

Each ARM core in ZYNQ has its own private timer, the operating frequency of the private timer is half of the CPU, if the operating frequency of ZYNQ is 666MHZ, the frequency of the timer is 333MHz, and you need to pay attention to this clock later.

## System Design:

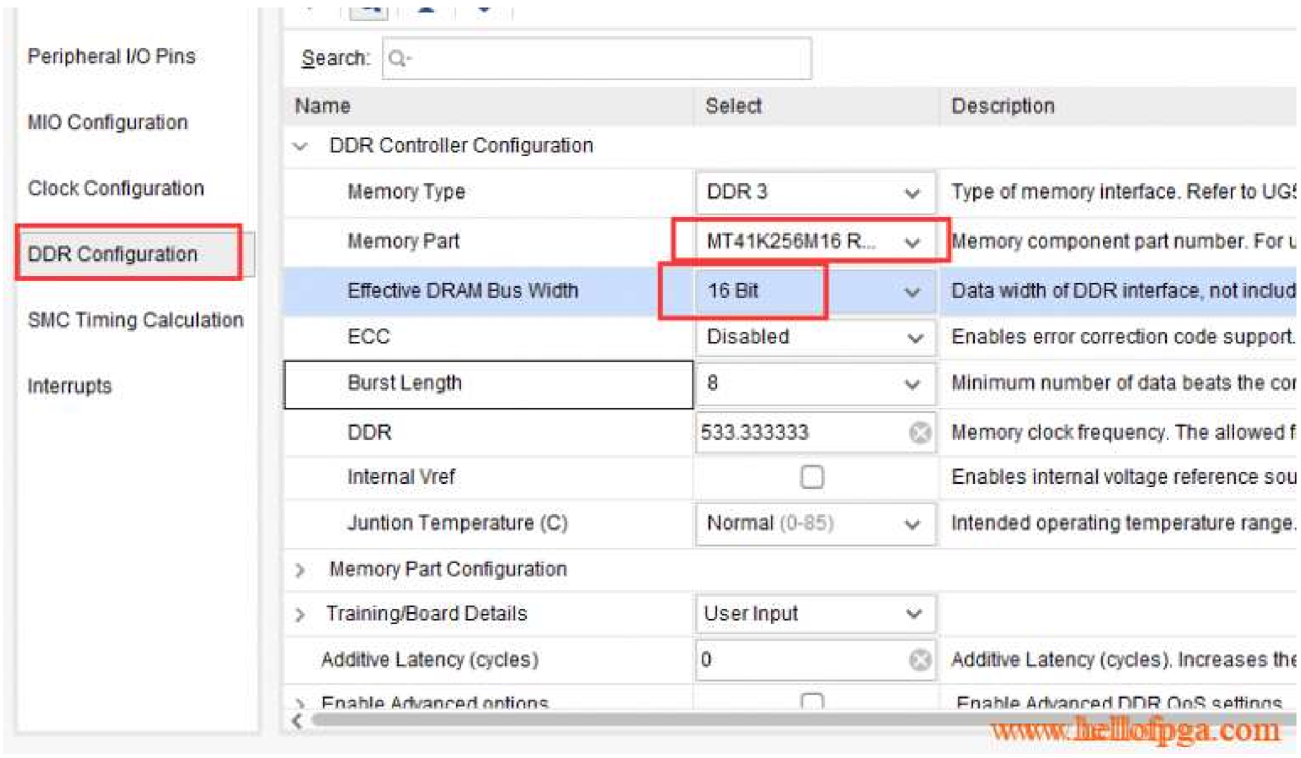
In terms of the system, we set a timer on the PS side, and let the timer trigger an interrupt every 1ms of counting, and turn on or off an LED light every cumulative second to achieve the demonstration effect.

SO THE WHOLE SYSTEM WE ONLY NEED TO EXPAND AN EMIO GPIO PORT IN THE ZYNQ CORE IN BLOCK DESIGN, AND USE THE TIMER OF THE PS SDK PROGRAM

TO DRIVE THIS GPIO EVERY SECOND.

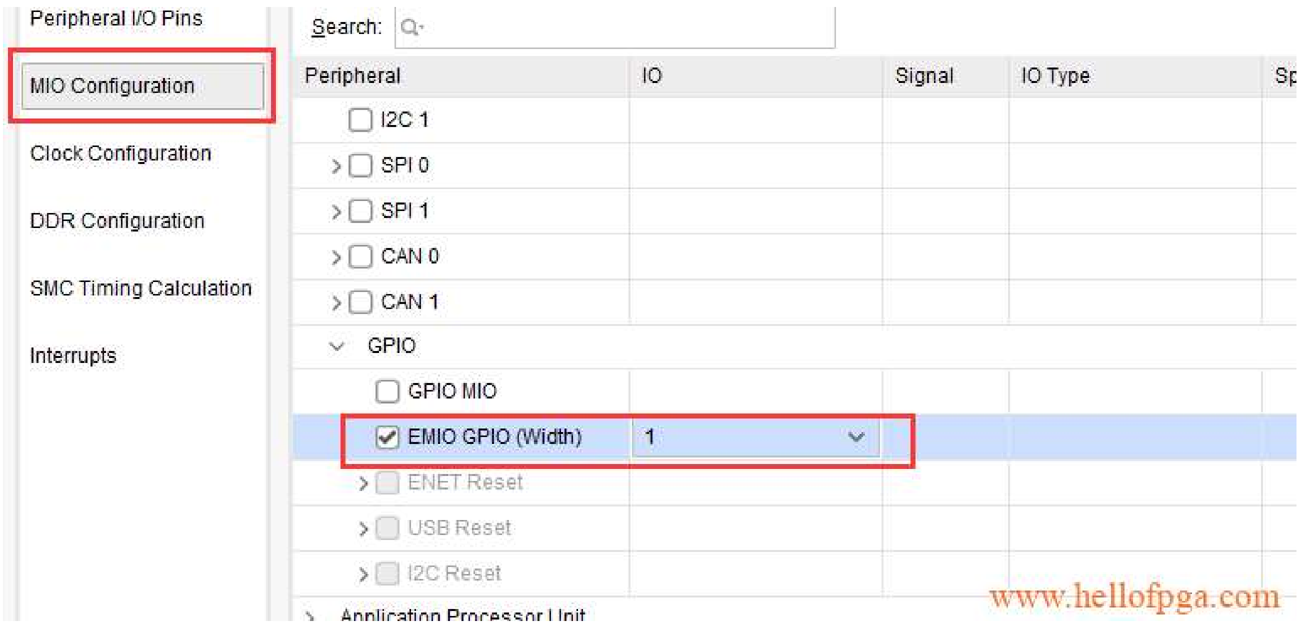
The project of EMIO is created here Abbreviated, the complete graphic tutorial can see the previous example, Smart ZYNQ (SP&SL version) Project VII Use ZYNQ's PS to light up the LED light connected to the PL end EMIO mode (recommended way).

On the ZYNQ setting interface, set the model and bit width of DDR MT41K256M16RE-125 to 16bit



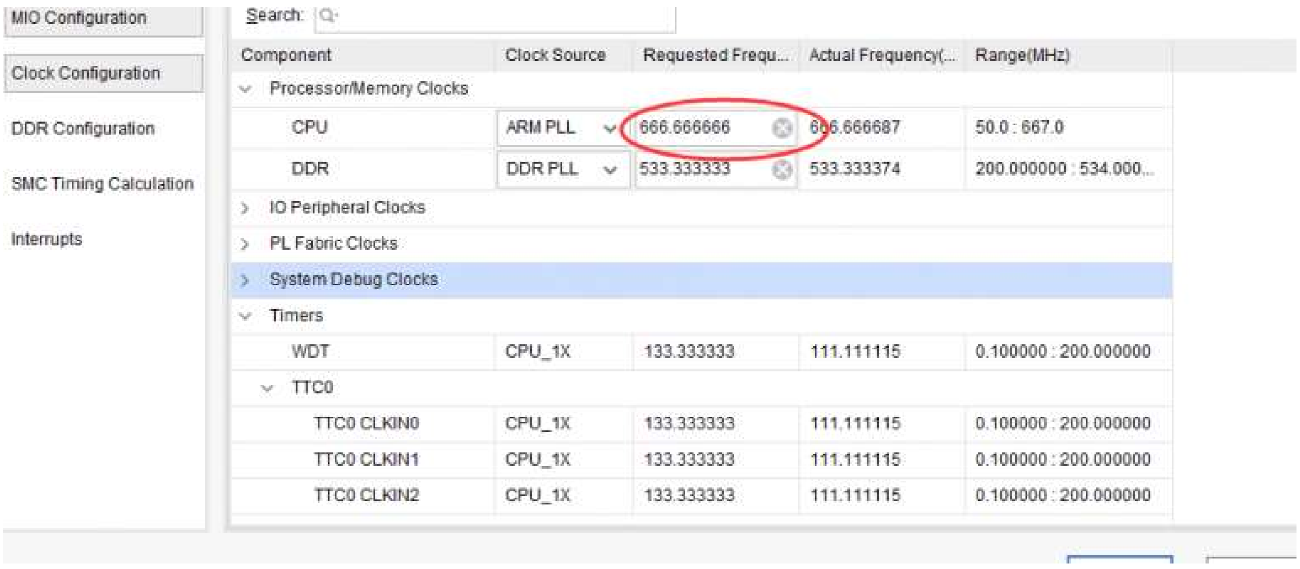
| Name | Select | Description |
|---|---|---|
| DDR Controller Configuration | | |
| Memory Type | DDR 3 | Type of memory interface. Refer to UG5 |
| Memory Part | MT41K256M16 R... | Memory component part number. For u |
| Effective DRAM Bus Width | 16 Bit | Data width of DDR interface, not includ |
| ECC | Disabled | Enables error correction code support. |
| Burst Length | 8 | Minimum number of data beats the co |
| DDR | 533.333333 | Memory clock frequency. The allowed f |
| Internal Vref | ☐ | Enables internal voltage reference sou |
| Juntion Temperature (C) | Normal (0-85) | Intended operating temperature range. |
| Memory Part Configuration | | |
| Training/Board Details | User Input | |
| Additive Latency (cycles) | 0 | Additive Latency (cycles). Increases the |
| Enable Advanced options | ☐ | Enable Advanced DDR QoS settings |

Add a GPIO interface (EMIO mode) to prepare the LED for lighting up later



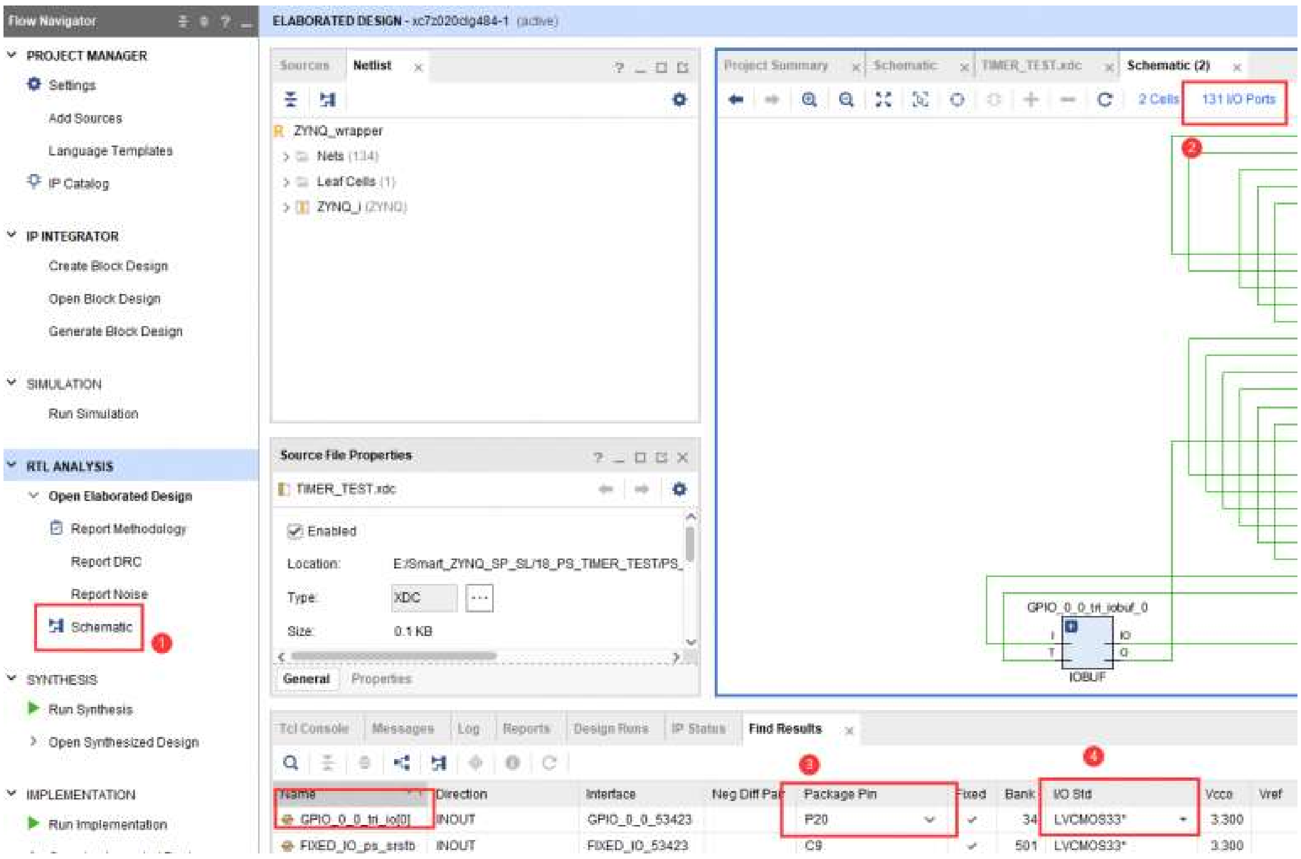
| Peripheral | IO | Signal | IO Type | Sp |
|---|---|---|---|---|
| ☐ I2C 1 | | | | |
| ☐ SPI 0 | | | | |
| ☐ SPI 1 | | | | |
| ☐ CAN 0 | | | | |
| ☐ CAN 1 | | | | |
| GPIO | | | | |
| ☐ GPIO MIO | | | | |
| ✓ EMIO GPIO (Width) | 1 | | | |
| ☐ ENET Reset | | | | |
| ☐ USB Reset | | | | |
| ☐ I2C Reset | | | | |
| Application Processor Unit | | | | |

Third, record the clock of the CPU (there is no need to modify here, just look at the current clock frequency recorded)



| Component | Clock Source | Requested Frequ... | Actual Frequency(... | Range(MHz) |
|---|---|---|---|---|
| Processor/Memory Clocks | | | | |
| CPU | ARM PLL | 666.666666 | 666.666687 | 50.0 : 667.0 |
| DDR | DDR PLL | 533.333333 | 533.333374 | 200.000000 : 534.000... |
| IO Peripheral Clocks | | | | |
| PL Fabric Clocks | | | | |
| System Debug Clocks | | | | |
| Timers | | | | |
| WDT | CPU_1X | 133.333333 | 111.111115 | 0.100000 : 200.000000 |
| TTC0 | | | | |
| TTC0 CLKIN0 | CPU_1X | 133.333333 | 111.111115 | 0.100000 : 200.000000 |
| TTC0 CLKIN1 | CPU_1X | 133.333333 | 111.111115 | 0.100000 : 200.000000 |
| TTC0 CLKIN2 | CPU_1X | 133.333333 | 111.111115 | 0.100000 : 200.000000 |

Add and set the pin constraint of the LED (P20 LVCMOS33)



Source File Properties

TIMER_TEST.xdc

Enabled

| | |
|---|---|
| Location: | E:/Smart_ZYNQ_SP_SL/18_PS_TIMER_TEST/PS_ |
| Type: | XDC |
| Size: | 0.1 KB |

General Properties

| Name | Direction | Interface | Neg Diff Pair | Package Pin | Fixed | Bank | I/O Std | Vcco | Vref |
|---|---|---|---|---|---|---|---|---|---|
| GPIO_0_0_tri_io[0] | INOUT | GPIO_0_0_53423 | | P20 | ✓ | 34 | LVCMOS33* | 3.300 | |
| FIXED_IO_ps_srstb | INOUT | FIXED_IO_53423 | | C9 | ✓ | 501 | LVCMOS33* | 3.300 | |

设下的对项目行编译综合就好

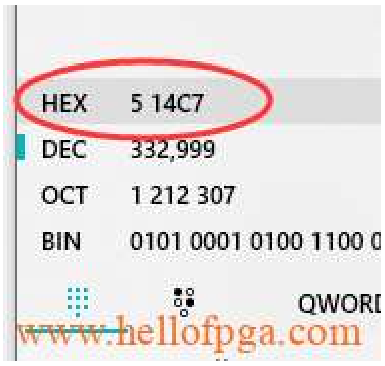
## 程序设计：

建立一个TIMER_TEST空的工程，并且添加main.c 添加如下代码：

```
#include "xparameters.h"
#include "xgpiops.h"
#include "xstatus.h"
#include "xplatform_info.h"
#include "xscutimer.h"
#include "Xscugic.h"

#define LED     54

#define GPIO_DEVICE_ID          XPAR_XGPIOPS_0_DEVICE_ID
XGpioPs Gpio;


void Gpio_Init(void){
        XGpioPs_Config *ConfigPtr;

        ConfigPtr = XGpioPs_LookupConfig(GPIO_DEVICE_ID);
        XGpioPs_CfgInitialize(&Gpio, ConfigPtr,ConfigPtr-
>BaseAddr);

        XGpioPs_SetDirectionPin(&Gpio, LED, 1);
        XGpioPs_SetOutputEnablePin(&Gpio, LED, 1);
        XGpioPs_WritePin(&Gpio, LED, 0);
}



#define TIMER_DEVICE_ID XPAR_XSCUTIMER_0_DEVICE_ID
#define INTC_DEVICE_ID XPAR_SCUGIC_SINGLE_DEVICE_ID
#define TIMER_IRPT_INTR XPAR_SCUTIMER_INTR
#define TIMER_LOAD_VALUE 0x514c7 //666*1000*1/2 666mhz
static XScuGic Intc; //GIC
static XScuTimer Timer;//timer

static void SetupInterruptSystem(XScuGic *GicInstancePtr,
XScuTimer *TimerInstancePtr, u16 TimerIntrId);
static void TimerIntrHandler(void *CallBackRef);

void SetupInterruptSystem(XScuGic *GicInstancePtr,XScuTimer
*TimerInstancePtr, u16 TimerIntrId){
        XScuGic_Config *IntcConfig; //GIC config
        Xil_ExceptionInit();
        //initialise the GIC
        IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
        XScuGic_CfgInitialize(GicInstancePtr,
IntcConfig,IntcConfig->CpuBaseAddress);
        //connect to the hardware
        Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
                (Xil_ExceptionHandler)XScuGic_InterruptHandler,
                GicInstancePtr);
        //set up the timer interrupt
        XScuGic_Connect(GicInstancePtr, TimerIntrId,
                (Xil_ExceptionHandler)TimerIntrHandler,
                (void *)TimerInstancePtr);
        //enable the interrupt for the Timer at GIC
        XScuGic_Enable(GicInstancePtr, TimerIntrId);
        //enable interrupt on the timer
        XScuTimer_EnableInterrupt(TimerInstancePtr);
        // Enable interrupts in the Processor.
        Xil_ExceptionEnableMask(XIL_EXCEPTION_IRQ);
}

volatile int ms_count;
unsigned char led_state=0;

static void TimerIntrHandler(void *CallBackRef){
        static int ms_count = 0; //计数
        XScuTimer *TimerInstancePtr = (XScuTimer *) CallBackRef;
        XScuTimer_ClearInterruptStatus(TimerInstancePtr);

        ms_count++;
        if(ms_count>=1000){
                ms_count=0;
                if(led_state==0)led_state=1;
                else led_state=0;
                XGpioPs_WritePin(&Gpio, LED, led_state);
        }
}


void Timer_Init(){
        XScuTimer_Config *TMRConfigPtr;
        TMRConfigPtr = XScuTimer_LookupConfig(TIMER_DEVICE_ID);
        XScuTimer_CfgInitialize(&Timer, TMRConfigPtr,TMRConfigPtr-
>BaseAddr);
        XScuTimer_SelfTest(&Timer);
        XScuTimer_LoadTimer(&Timer, TIMER_LOAD_VALUE);
        //自动装载
        XScuTimer_EnableAutoReload(&Timer);
        //启动定时器
        XScuTimer_Start(&Timer);
        //set up the interrupts
        SetupInterruptSystem(&Intc,&Timer,TIMER_IRPT_INTR);
}
int main(void)
{
        Gpio_Init();
        Timer_Init();

        while(1);

        return 0;
}
```

代码简单介绍：

Timer_Init(); 是定时器的初始化函数，在系统运行的时候，需要对定时器初始化才能正常工作，其中TIMER_LOAD_VALUE 用来控制 每一次定时器触发的时间， 还记得我们前面的CPU时钟为666mhz吗，那我们的定时器时钟就相当于工作在1/2的CPU时钟即333mhz频率下。 我们的系统设置是每1ms进入一次中断，那1ms就相当于需要计数（333000000）*1/1000=333000个周期，那我们就设置TIMER_LOAD_VALUE 为 333000-1，即0x514c7 (这里代表1ms的数值)


```
HEX   5 14C7
DEC   332,999
OCT   1 212 307
BIN   0101 0001 0100 1100 0
```

TimerIntrHandler 是定时器的回调函数，相当于单片机的中断服务函数，每当中断被触发，就会调用一次该回调函数。 因为是1ms进入一次，所以假设我们需要计数1S钟，就需要额外增加一个变量ms_count

每次进入该回调函数 ms_count都自增，当大于等于1000的时候让ms_count归零完成一次循环，同时执行我们的LED变换操作。 如果我们有其他程序需要添加，也可以增加到TimerIntrHandler的回调函数钟。

```
ms_count++;
if(ms_count>=1000){
    ms_count=0;

    if(led_state==0)led_state=1;
    else led_state=0;
    XGpioPs_WritePin(&Gpio, LED, led_state);
}
```

For the main function, we only need to perform the initialization of the GPIO and timer

```
int main(void){
        Gpio_Init();
        Timer_Init();
        while(1);
        return 0;
}
```

Download it to our board, you can see that the LED light on the motherboard flashes every second, indicating that our timer program is working normally.

The program is relatively simple, only for reference and learning, for ZYNQ's timer, there are many functions (including using PL to trigger timer interrupts, etc.), which need to be expanded by yourself when actually doing the project.

The complete project is as follows (for reference purposes):

18_PS_TIMER_TEST_XC7Z020 **Download**

**SMART ZYNQ SP & SL**