版本：1.8

SeruTek Ultrascale+TDC 17ch32b

快速指南

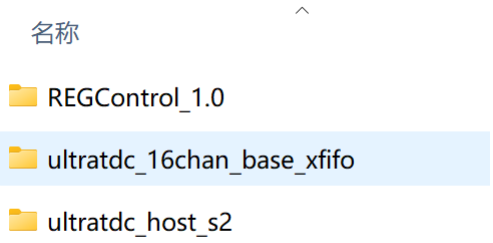
上海瑟如电子科技有限公司开发的基于 Ultrascale+ MPSoc(速度等级 1 或 2)的 17 通道 TDC IP 及附带的工程。本文旨在介绍该示例工程所包含的文件及使用方法。

开发环境：Vivado 2020.2

示例工程包括两部分内容：

1. TDC IP 核及 Vivado 最小系统示例工程
2. SDK 软件库及源程序

打开示例工程压缩包，目录结构如下：



Ultratdc_host_s2 为 TDC IP 工程目录。Ultratdc_16chan_base_xfifo 是示例工程目录。REGControl_1.0 是通过 AXI 控制寄存器的 IP。**建议先将示例工程跑通，再将 TDC IP 结合到自己的工程中去。**

1 快速测试

如果您从瑟如电子购买了评估板卡，那么评估板已经处于 SD 启动模式。TDC 的评估镜像应该已经复制到 SD 卡上了。所需要做的只是连接输入脉冲信号和串口线，接着接上电源。根据串口终端的提示，键入指令开始测量。需要注意的是固件中的脉冲输入端口电压为 3.3V LVCMOS，不要将 5V 脉冲直接接入，否则不仅影响测量还有可能会损坏开发板。

1.1 修改示例程序，综合实现后进行评估

示例工程基于 Enclustra ST1 ZU4 开发板制作。如目标板与此不同，用户需要修改工程，根据目标板的具体参数进行适配。需要修改的设备包括：PS 设置（DDR，UART 等）、TDC 参考时钟、脉冲输入，时序约束文件。

首先修改 Vivado 示例工程

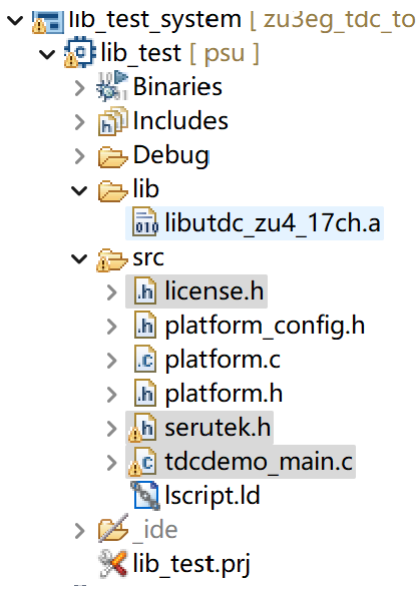
1. 打开 Vivado，打开示例工程。
2. **修改 PS 设置。**请根据板子的实际情况，重新配置 PS，需要检查、修改的内容有：DDR 型号、Uart、SD 等。
3. **修改 TDC 时钟。**在示例中 TDC IP 的输入时钟频率为 100M。示例工程中添加了 clock wizard，将时钟转化为 100MHz 输入到 TDC。如果用户开发板上的时钟频率与此不同，需要修改 clock wizard 的参数， **保证输入 TDC 的参考时钟为 100MHz。**
4. **修改脉冲输入。**示例工程只使用了 FPGA 内部产生的脉冲信号作为测试脉冲。当示例跑通后，请根据实际需求，将 TDC IP 上的脉冲输入端口引出，

并修改约束文件中 IO 的格式和电压。当同时测试 17 路外部脉冲输入时，建议使用 LVDS 等差分信号格式，（并添加相关的阻抗约束），降低输入脉冲信号之间的窜扰。

5. 约束文件中包含必须的时序约束，如果移植到自己的工程，**请将相关约束一起复制到新的工程，并根据实际工程名更改约束中的路径名。**如有疑问请随时联系瑟如电子。
6. 运行 Synthesis->Implementation->Generate Bitstream

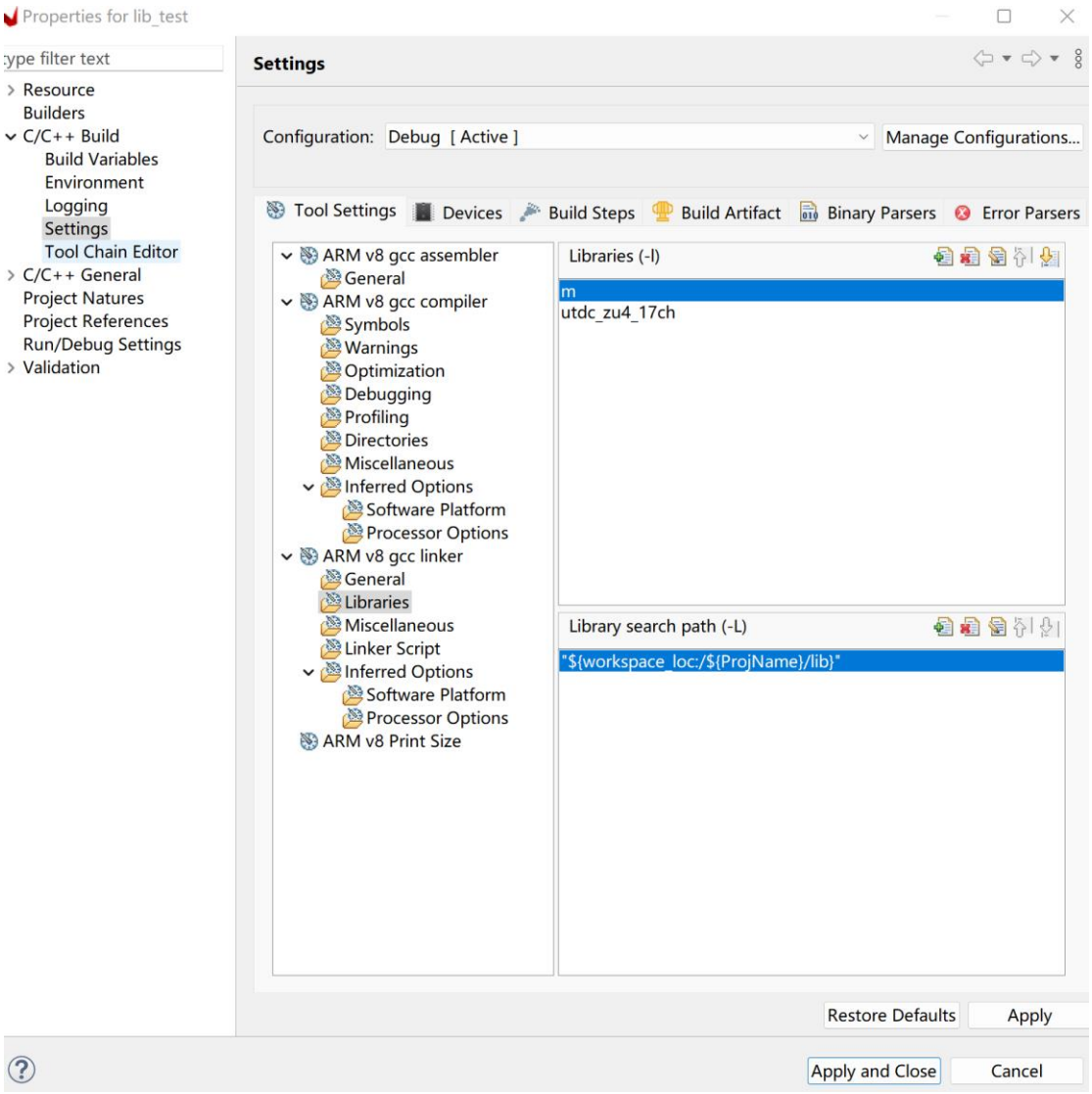
接着创建 SDK 工程

1. 导出 bitstream 并启动 SDK。
2. 在 SDK 中新建 helloworld Application 工程。并在工程下新建一个文件夹 Folder，取名 lib。将附带的 SDK 文件夹中的 libserutek_xxx.a 库文件复制到此处。
3. 将 serutek.h 复制到 src 文件夹，并将 SDK 包中的 tdcdemo_main.c 的内容替换原有 helloworld.c。helloworld 工程文件夹结构如下图所示：

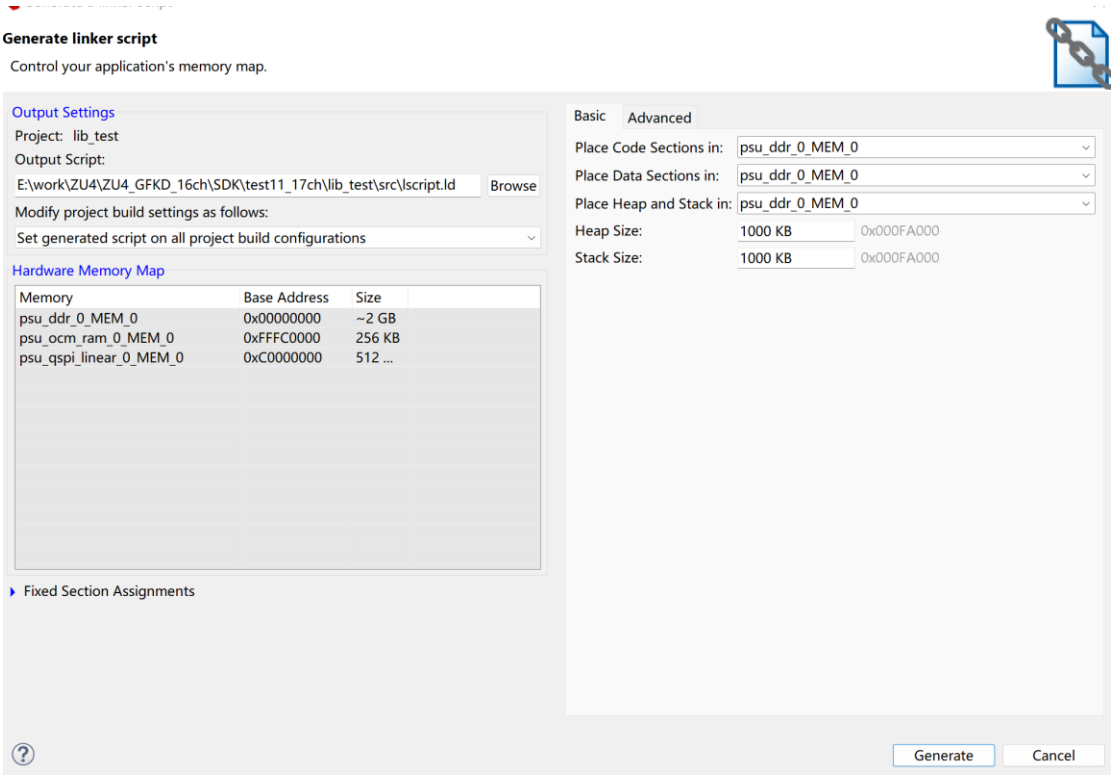


4. 右击项目，c/c++ building settings，如下图设置 libraries 和 library path，

注意，除了需要包含 tdc 库文件外还要包含系统内部库文件 m。



5. 调整 stack/heap size。示例中用到 printf 函数，需要更大的 stack/heap size，建议将 stack size 设为大于等于 64kB，示例中为 1000KB。选择工程，在顶层菜单中选择 xilinx->generate linker script



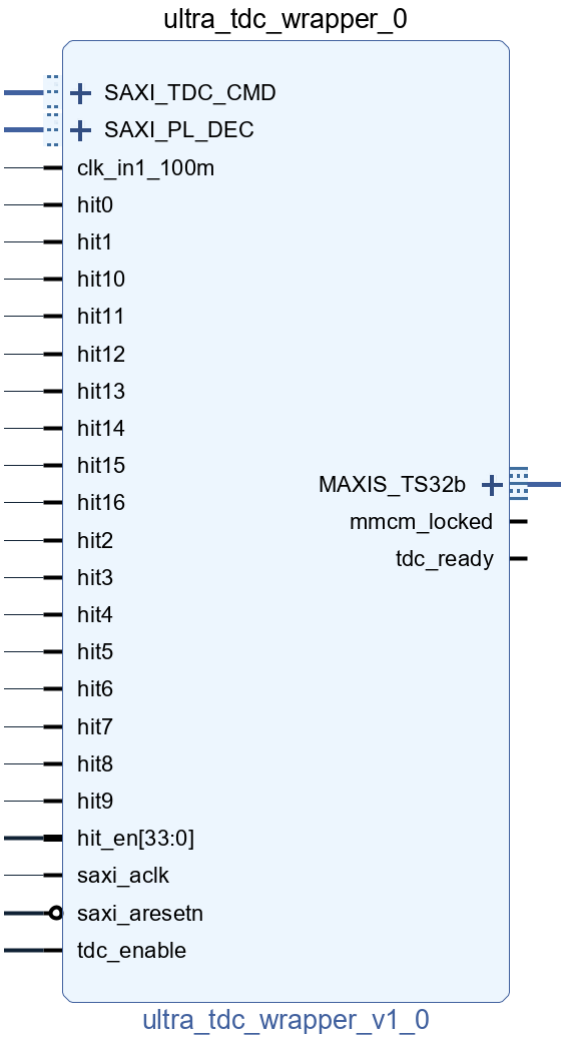
将 heap size 和 stack size 调节为如上图所示大小。

6. 编译工程。
7. 保证参考 时钟及 hit 信号连接正确，每路 hit 信号需要良好接地，不同信号间须具有良好的隔离。
8. 打开串口终端，设置波特率 115200，no parity，8 data bits。
9. 下载 bitstream。启动 SDK 的 helloworld 工程。

2 Vivado 示例工程说明

示例工程由 17 通道 TDC IP、测试脉冲产生模块、Zynq PS、AXI Stream FIFO 及相关总线 IP 组成。目的在于演示 17 通道 TDC 的测量原理和使用方法。

2.1 TDC IP



根据功能，TDC IP 的 IO 口可以分为以下几类：

1. AXI 控制总线接口，SAXI_XXXX，用于读写 TDC IP 内部寄存器，控制 TDC IP 功能；
2. 参考时钟输入，clk_in1，该版本中输入时钟频率应为 100MHz；
3. 测量脉冲输入，hit0-hit16，共 17 通道；
4. hit_en，TDC 通道使能。该信号的低 17 位[16:0]对应 17 个通道，相应位为‘1’时，该通道使能，为‘0’时，该通道关闭；

   TDC IP 的通道也可以在通过 SDK 函数，对寄存器写入控制字来进行使

能。通过 hit_en 管脚使能和寄存器使能是逻辑“或”的关系，任意一种方式都能使能 TDC IP 的测量通道。通过 hit_en 使能具有低延迟的特性，延迟一般低于 3ns。而通过寄存器写入控制字的方式延迟较大，因为需要进行 AXI 总线的访问。

5. **tdc_enable**, TDC 使能，当检测到该信号上升沿时，TDC 使能，下降沿时，TDC 内部逻辑处于重置状态，无法工作；
6. MAXIS_TS32b，测量数据输出端口。
7. Mmcm_locked, 用于指示 TDC IP 内部 mmcm 的锁定情况。为‘0’表示内部时钟没有锁定，TDC 功能异常，应检查 TDC 参考时钟状态。

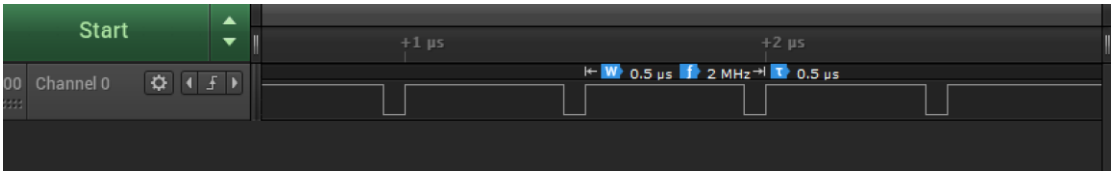
2.2 TDC IP 使能

此 32bit 版本内部粗计数器计数范围约为 6us(6144000 ps)。当超出计数范围时，计数器将回滚至 0，并重新开始计数。用户可定制时间戳量程， 最大可达 70 余年。

在使用 TDC IP 进行连续测量时，如果对 start/stop 脉冲信号不加控制，那么同一组 start\stop 脉冲有可能同时出现计数回滚之前和回滚之后。可以有两种方法应对这个问题。 方法一：周期性地使能 TDC 并控制 start 信号出现的时间，使其出现在计数器回滚之后的一定时间内；控制 TDC 通道使能信号， 使得在计数器回滚前一定时刻，TDC 就不接收 stop 脉冲。方法二， TDC 连续运行， 对有可能出现地一组 start/stop 脉冲跨越计数回滚地情况进行补偿。 无论哪种方法，都需要保证同一组 stop 信号不能延迟 start 大于一个回滚周期（6.144us），否则 stop 和 start 之间有可能经历 2 次计数器回滚，从而导致无法正确补偿。

第一种方法需要一个**同步信号**周期性地重置并使能 TDC，而被测脉冲在同步脉冲产生后的一定时间范围内进入 TDC IP，从而使得被测脉冲出现在 TDC 的同一个计数周期内。

在示例中，测量重复频率为 2MHz，那么 tdc_enable 信号的周期应当为 500ns，如下图所示。



在 tdc_enable 信号下降后，TDC IP 内部进入重置状态，重置过程需要一定的时间，在下一个 tdc_enable 上升沿前，tdc_enable 必须保证一段时间的低电平。如果 tdc_enable 上升时，内部重置过程还没有完成，那么 TDC IP 在这个上升沿就不能被正确地启动。

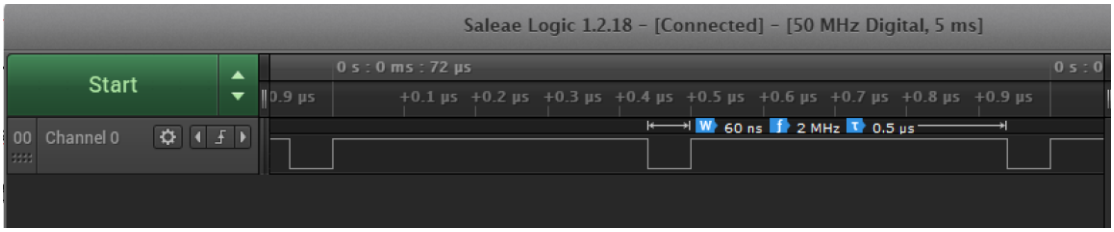
SDK 示例程序中下列代码设置内部重置信号的一些时序参数。

```
/*
 * configure the internal parameter
 */
u32 thres1 = 2;
u32 thres2 = 4;
u32 thres3 = ((thres2 << 8) | thres1) << 1;
Xil_Out32(CMDREG+6*4, thres3);
```

Tdc_enable 保持低电平的最短时间可以通过以下公式计算：

$$(\text{thres2} + 2) * 10\text{ns}$$

由此可以计算出，Tdc_enable 保持低电平的最短时间为 60ns，如下图所示。



当 TDC IP 重置过程结束处于待机状态时，tdc_enable 的上升沿将会启动 TDC IP。从 tdc_enable 上升沿到 TDC 开始正常工作需要一定的时延。在示例中，通过测量得到这段时延略小于 30ns。在实际运行中，请适当留有 10-20ns 的裕量，也就是保证被测量的脉冲在 tdc_enable 上升沿的 40-50ns 后才到达 TDC IP。过早到达的脉冲有可能不能被正确地测量。

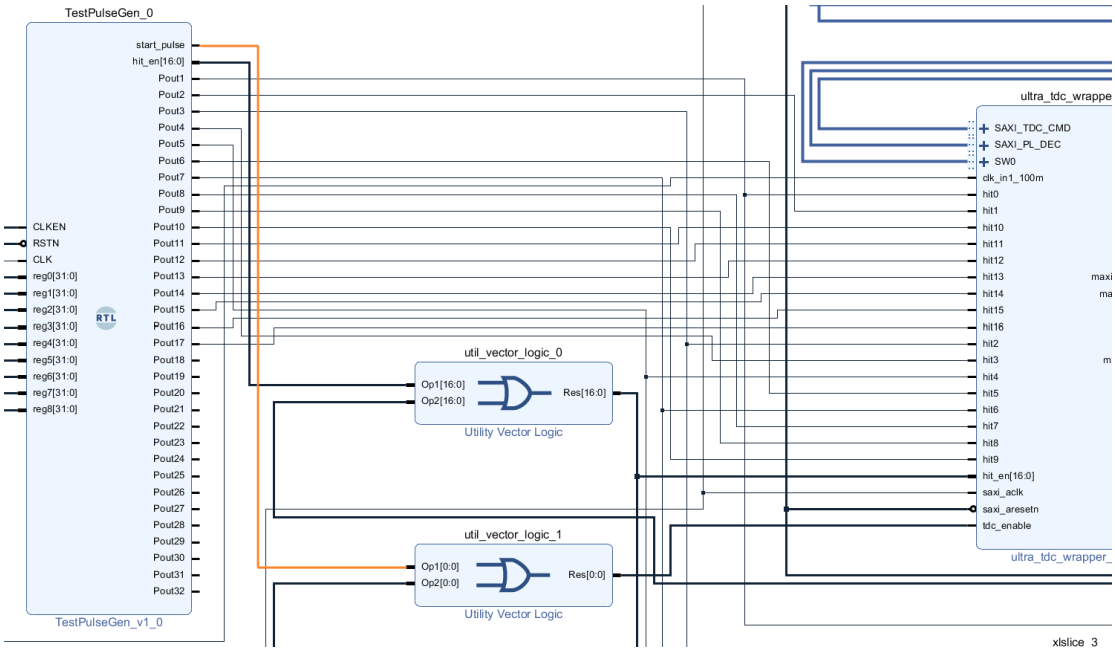
第二种方法是启动 TDC 后，使其连续工作。如果一组脉冲经历计数回滚，那么就会有 stop-start 的时间差为负数的情况。这时只要判断时间差是否为负，如果为负则加上一个回滚周期 6.144us，这样就能得到正确的时间差了。

2.3 测试脉冲产生

在示例中采用自定义模块产生测试脉冲，输入到 TDC IP 进行测量。 当跑通示例后， 用户可将 TDC IP 的 hit 端口引出到外部，用真是信号测量。

2.3.1 内部产生测试脉冲

测试脉冲产生模块 TestPulseGen 内置一个可配置阈值的 binary counter，并通过判断不同的计数器阈值，来产生 TDC 同步信号和一系列测试脉冲。



在示例中该模块的输出信号 start_pulse，用作同步信号，对 TDC 周期性使能。而 hit_en 信号则用来控制 TDC IP 通道的使能状态。Pout1-Pout17 是 17 路测试脉冲输出。

TestpulseGen 模块产生 17 路脉冲。 第一路用来模拟 start 脉冲， 其余 16 路用来模拟回波脉冲(stop)信号。 每一个 start 脉冲，在每个通道会产生对应的 6 个回波脉冲。这些脉冲相对于 start 的延迟和脉冲之间的间隔可通过寄存器配置， 详情请参照源代码。

其中

reg0 用来控制内置计数器回滚阈值；

reg1 控制 start_pulse（TDC 使能）的时序；

reg2 控制 hit_en（通道使能）的时序；

2.4 时间戳数据的读取

示例中用 axi stream fifo 来获取 TDC IP 输出的时间戳信号。PS 从 axi stream

fifo 读取测量值后，通过串口打印。 由于 axi stream fifo 本身缓存容量有限， 且串口打印速度很慢，所以当进行高速测量时， 每次测量的数据点数不能超过 axi stream fifo 本身的缓存点数。

当示例跑通之后， 用户可自行添加 axi stream switch IP（1 slave port, 2 master port)。当 TDC 初始化时， 发送指令将 axi stream switch 切换到第一路输出。 当进行测量时， 切换到第二路。 在第二路 Master 输出之后可以接高速 DMA 传输模块或自定义 PL 端时间戳处理模块。

3 SDK 示例程序说明

在 SDK 示例程序中提供了 get_Timestamp_rt_ts2mm()函数，用于通过 DMA 模块从 ts32 接口读取时间戳并打印。打印之后的时间戳可以用 matlab 或 python 程序进行处理分析。

3.1 时间戳测量流程

3.1.1 初始化及 TDC 校准

首先需要确认 SDK 中寄存器的地址与 Vivado 工程中的地址一致，特别是当将 TDC IP 集成到用户原有工程中后，寄存器地址一般都会与示例中定义的不一致，因此必须修改。

在 tdcdemo_main.c 中，通过宏定义了控制 TDC 的寄存器地址，如下图：

```
#define PL_DECD_REG         0xA0020000
#define CMDREG              0xA0070000
#define REGCTRL_REG         0xA0030000
//#define SW0REG                0xA0050000
//#define SW1REG                0xA0060000
```

确认以上地址与 vivado 工程中的地址一致。如果后续修改 vivado 工程之后，也需要检查这些地址是否与 vivado 工程中定义的地址一致，如果不一致，需要在此 C 文件中修改这些地址。

在测量主程序中，测试流程如下：

- 初始化外设，包括 AXI Stream FIFO，ts2mm DMA 模块，AXI GPIO。

```
//gpio init
int Status = XGpio_Initialize(&Gpio0, 0);
if (Status != XST_SUCCESS)  {
    xil_printf("gpio init failed\r\n");
  return XST_FAILURE;
}
//axi stream fifo init
Status = XLlFifo_Init(&FifoInstance, XPAR_AXI_FIFO_0_DEVICE_ID);
if (Status != XST_SUCCESS)  {
    xil_printf("Xfifo init failed\r\n");
    return XST_FAILURE;
}
/* Set the direction for all signals to be outputs */
XGpio_SetDataDirection(&Gpio0, 1, 0x0);
XGpio_SetDataDirection(&Gpio0, 2, 0x0);
```

- 配置内部参数并拉高 TDC IP 的 tdc_enable 端口，使能 TDC，为初始化校准 TDC 做准备：

```
  /*
   * configure the rest sequencer parameter
   */
  u32 thres1 = 1;
  u32 thres2 = 3;
  u32 thres3 = ((thres2 << 8) | thres1) << 1;
  Xil_Out32(CMDREG+6*4, thres3);

  //time stamp decoder in passthrough mode
  Xil_Out32(PL_DECD_REG, 0x00);
  Xil_Out32(PL_DECD_REG, 0x00);

  /*
   *Enable TDC for calibration
   */
  XGpio_DiscreteWrite(&Gpio0, 2, 0x04);
```

- 设置授权码

```
struct devkey_char tdc_dev = tdc_keyhash_list[1];
```

- 设置通道间静差。TDC IP 由于内部走线不同， 每个通道之间会有 3ns 的静差。 当 bitstream 不变且外部电路是同一个电路时，重复上电，通道间的静差不变。 但是不同的 PCB 上工艺和器件的时延总会略有不同，因此每台设备建议都要做通道静差校准。 校准后的值可以用来设置通道静差，从而保证各个通道之间 skew 在 50ps 以内。

```
    for(int i = 0; i < CHN_CNT; i++){
//      set_chanoff(i+1, 0);
        set_chanoff(i+1, i*10000);
    }
```

- 调用 init_tdc 对所有通道的 TDC 进行初始化和校准。
- 关闭 TDC

```
    /*
     * Disable TDC
     */
    XGpio_DiscreteWrite(&Gpio0, 2, 0x00);
```

- 设置测试脉冲产生模块。

```
u32 pulse_period = 903;//225->1M *(193) , 903->250k*193
u32 tdcen_lo = 2;  // set high from
u32 tdcen_hi = 202; // set high until

Xil_Out32(REGCTRL_REG+0*4, pulse_period);
Xil_Out32(REGCTRL_REG+1*4, (tdcen_hi << 16) | (tdcen_lo) ); //start_pulse
Xil_Out32(REGCTRL_REG+2*4, (tdcen_hi << 16) | (tdcen_lo) ); //hit_en
Xil_Out32(REGCTRL_REG+3*4, 20);
Xil_Out32(REGCTRL_REG+4*4, 2);
```

- 根据用户输出，选择对应的测量模式，进行测量与计算。

3.1.2 测量及时间戳数据读取

示例程序中包含了两种模式的测量/处理方式：第一种 reset 模式，周期性地

重置 TDC，并保证发射和回波脉冲都在同一个计数周期中。 第二种：

保持 TDC 连续工作，对时间差进行补偿地 non-reset 模式

以 reset 模式进行测量，在 get_PLtimstamp_rt()中：

- 使能时间戳解码：

```
// enable decoder
Xil_Out32(PL_DECD_REG, 0x01);
Xil_Out32(PL_DECD_REG+4, 0x00);
```

- 使能测试脉冲产生模块，等待一段时间后关闭该模块：

```
XGpio_DiscreteWrite(&Gpio0, 2, 0x03);// start Test Pulse
usleep(40);
XGpio_DiscreteWrite(&Gpio0, 2, 0x0);
```

测量期间， TDC 使能和通道使能都由测试脉冲产生模块控制。

- 读取 AXI Stream FIFO，并对时间戳进行解析：

```
    occp = XLlFifo_RxOccupancy(&FifoInstance);
    xil_printf("sample %u\r\n", occp);
    for(int j = 0; j< occp; j++){
/       occp = XLlFifo_RxOccupancy(&FifoInstance);

        if(bytes >= MAX_TDC_SAMPLE*4 )
        {
            XLlFifo_Read(&FifoInstance, empty_buffer, 4);
        }
        else
        {
            XLlFifo_Read(&FifoInstance, ts_buffer+bytes, 4);
            bytes += 4;
            samplecnt++;
        }
    }
    xil_printf("copy %u bytes %d samples\n\r", bytes, samplecnt);
    u8* ts_ptr;
    u32 offset = 0;
    while (offset < bytes) {
        ts_ptr = ts_buffer+offset;
        u8 id = *(ts_ptr+3);
        u32 ts = *((u32*) ts_ptr);
        ts = (ts & 0xFFFFFF) << 8;
        s32 ts_int = ((s32)ts) >> 8;
        ts_samples[id-1][SaCnt[id-1]++] = ts_int;
/       printf("%2u, %12u\r\n", id, ts);
        offset +=4;
    }
```

可以取消 上述代码中 printf 地注释， 这样就能在串口中断看到具体地时间戳了。

- 计算各回波相对发射脉冲的时间差， 并计算标准差
- 重置 axi stream fifo（清空可能残留在 fifo 里的测量值）

以 non-reset 模式进行测量：

流程与 reset 模式基本一致，有两个地方有差别

在 get_PLtimstamp_nonreset()中：

1. 拉高 tdc_enable 和 tdc_en 信号，使得 TDC 还有它所有的通道保持一直使能的状态

```
XGpio_DiscreteWrite(&Gpio0, 2, 0x04); //tdc always enabled during measurement
XGpio_DiscreteWrite(&Gpio0, 1, 0xFFFFFFFF);//tdc channels always enabled
```

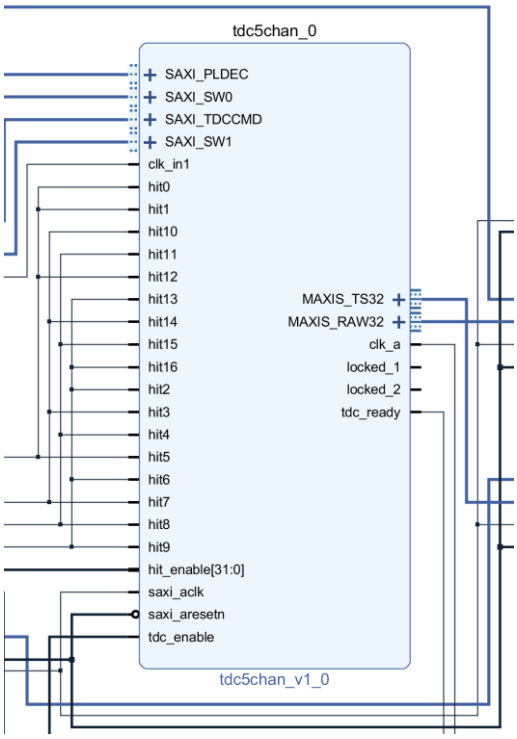
2. 在计算时间差时对负的时间差进行补偿：

```
u32 threshold = 6144000;

for(int k=1;k<CHN_CNT;k++)
{
    for(int i=0;(i*6)<SaCnt[k];i++)
    {
        for(int j= 0; j<6;j++)
        {
            s32 td = ts_samples[k][i*6+j] - ts_samples[0][i];
            if(td < 0)
            {
                td = td + threshold;
            }
            td_records[k][j][i] = td;
            if(td > tdmax[k][j])
            {
                tdmax[k][j] = td;
            }
            else if (td < tdmin[k][j])
            {
                tdmin[k][j] = td;
            }
        }
        SingleReturnCnt[k]++;
    }
}
```

4 TDC IP 接口说明

4.1 TDC 配置及数据输出接口



4.1.1 Hit_enable TDC 通道使能接口

使能 TDC 通道可通过两种方式，一种为内部寄存器使能，另一种是通过使能引脚。

一、使用内部寄存器：在 SDK 中调用 startMeasure4chan 函数，对所有通道进行使能；stopMeasure4chan 关闭所有通道。

二、也可通过使能管脚 hit_en 进行使能。Hit_en 的低 17 位（[16:0]）对应 17 个 TDC 通道。通过管脚使能延迟更小，延迟一般小于 3ns。而通过内部寄存器使能的延迟在微秒量级。

4.1.2 TDC 配置接口

AXI 开头 slave AXI-Lite 接口是 TDC IP 的配置接口。这些接口的时钟和复位信号分别是 saxi_aclk 和 saxi_aresetn。

SAXI_CLKCMD 用来控制 TDC IP 各测量通道的使能。

AXI_PL_DECODER 是用来控制 PL 时间戳解码模块的寄存器。默认情况下，该模块处于 pass through 模式，输出原始数据，不进行解码。当 TDC 处于自校准模式时，必须将解码模块设为 pass through 模式，输出原始测量数据。当解码模块激活时，TDC 输出解码后的时间戳。

下图向解码模块下达指令，置于 passthrough 模式，输出原始数据用于内部校准：

```
// enable decoder's passthrough mode, calibration needs raw measurement
Xil_Out32(PL_DECD_REG, 0x00);
Xil_Out32(PL_DECD_REG+4, 0x00);
```

下图表示解码模块激活，输出解码后的时间戳：

```
Xil_Out32(PL_DECD_REG, 0x01);   // disable decoder's passthrough mode
Xil_Out32(PL_DECD_REG+4, 0x00);
```

4.1.3 TDC 数据输出接口

TDC IP 定义一组 Master AXI Stream 数据输出接口:TS32b。它们的时钟和复位信号都是 ***saxi_aclk 和 saxi_aresetn*** 。

TS32b 输出时间戳的原始码，在 TDC 自校准时使用。

TS32b 输出解码后的时间戳。该时间戳是由 PL 内部的解码逻辑计算输出的。能够提供于时钟相当的解码速度。

4.2 时间戳格式

默认时间戳长度为 4 字节，最高的 8 位为通道标识，用来表示产生该时间戳的通道编号。在本示例中，通道编号范围是 1-17。

时间戳的低 24 位用来表示时间戳的值，数据类型为有符号整型。最小位 LSB 对应的单位是皮秒 ps。

[31:24]	[23:0]
通道号	时间戳，无符号整型，LSB 单位： ps

4.3 最小脉冲间隔

最大测量频率为 90M。这个参数决定了 TDC 的单个通道能够测量两个脉冲之间的最小间隔。90M 的测量速率对应的最小脉冲间隔为 11ns。当第二个脉冲与前一个脉冲的时间间隔小于 11ns，第二个脉冲则不会被测量。

注：不同通道之间没有最小脉冲间隔。

4.4 定制化选项

SeruTek TDC 可定制化选项有

1. 通道数
2. 通道高速 FIFO 点数
3. 最小脉冲间隔
4. 数据接口形式

4.4.1 通道数

向所有用户开放 1-4 通道 TDC 的 IP 试用。

如需定制 4 通道以上的 TDC， 请咨询瑟如电子。最大通道数，需要根据具体 FPGA 型号进行评估。

4.4.2 接口形式

如果需要持续的高频率测量及高吞吐量的数据读出，可定制每个通道独立的 AXI Stream 接口输出。此种形式搭配 PCIE，以上位机的大容量内存作为缓存，可实现惊人的连续测量能力。详情请咨询上海瑟如电子科技。