实验四：串行IO接口设计

班级：**提高2301班**
姓名：**张禹阳**
学号：**U202314270**

实验目的

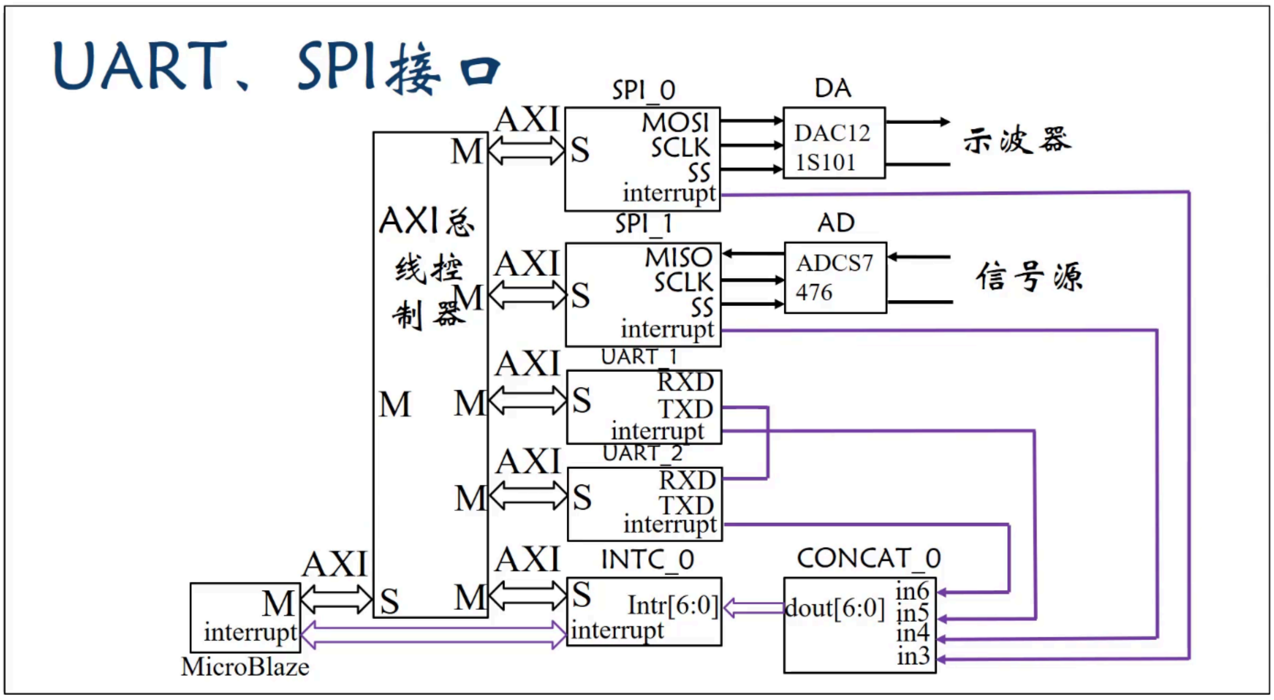
- 掌握GPIO IP核的工作原理和使用方法
- 掌握中断控制方式的 IO 接口设计原理
- 掌握中断程序设计方法
- 掌握 IO 接口程序控制方法

实验任务

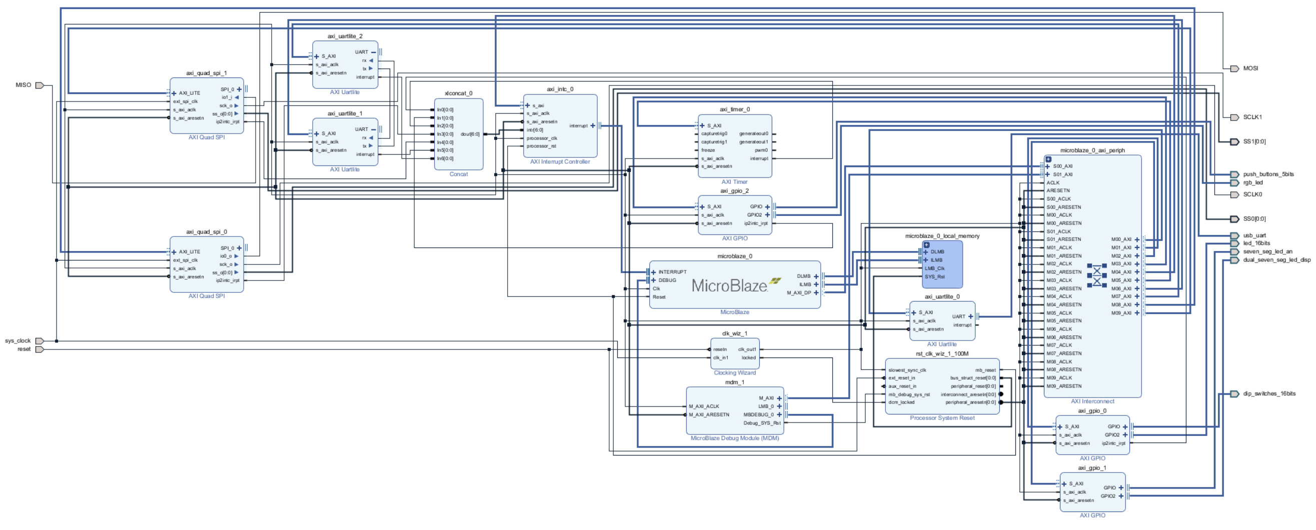
利用SPI IP 核，timer IP 核以及 DA 模块，控制 DA 模块输出周期可变锯齿波，且锯齿波周期由 switch 控制。锯齿波周期最长约为 1s，最短约为 60ms

实验原理

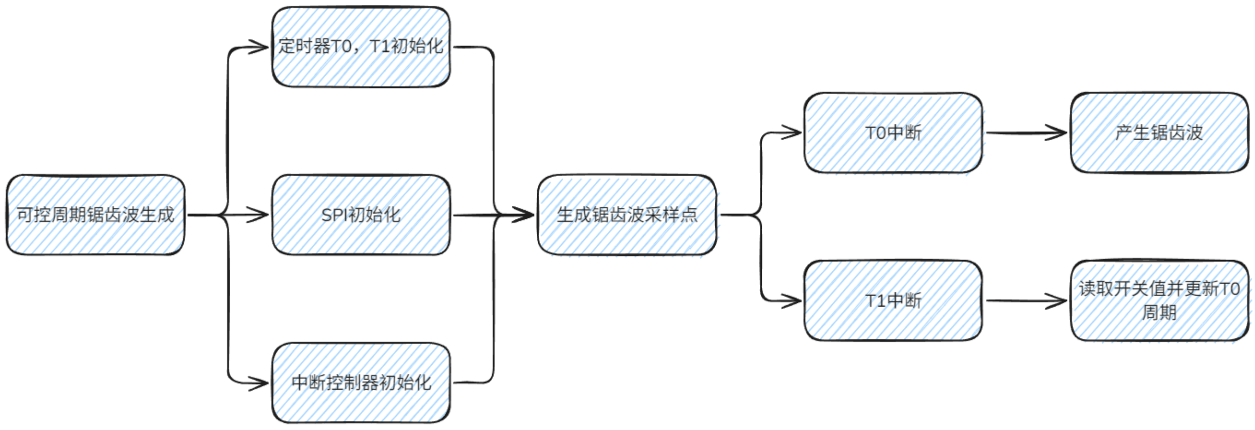
硬件电路框图



根据硬件电路框图搭建的硬件平台整体框图如下：



软件流程图

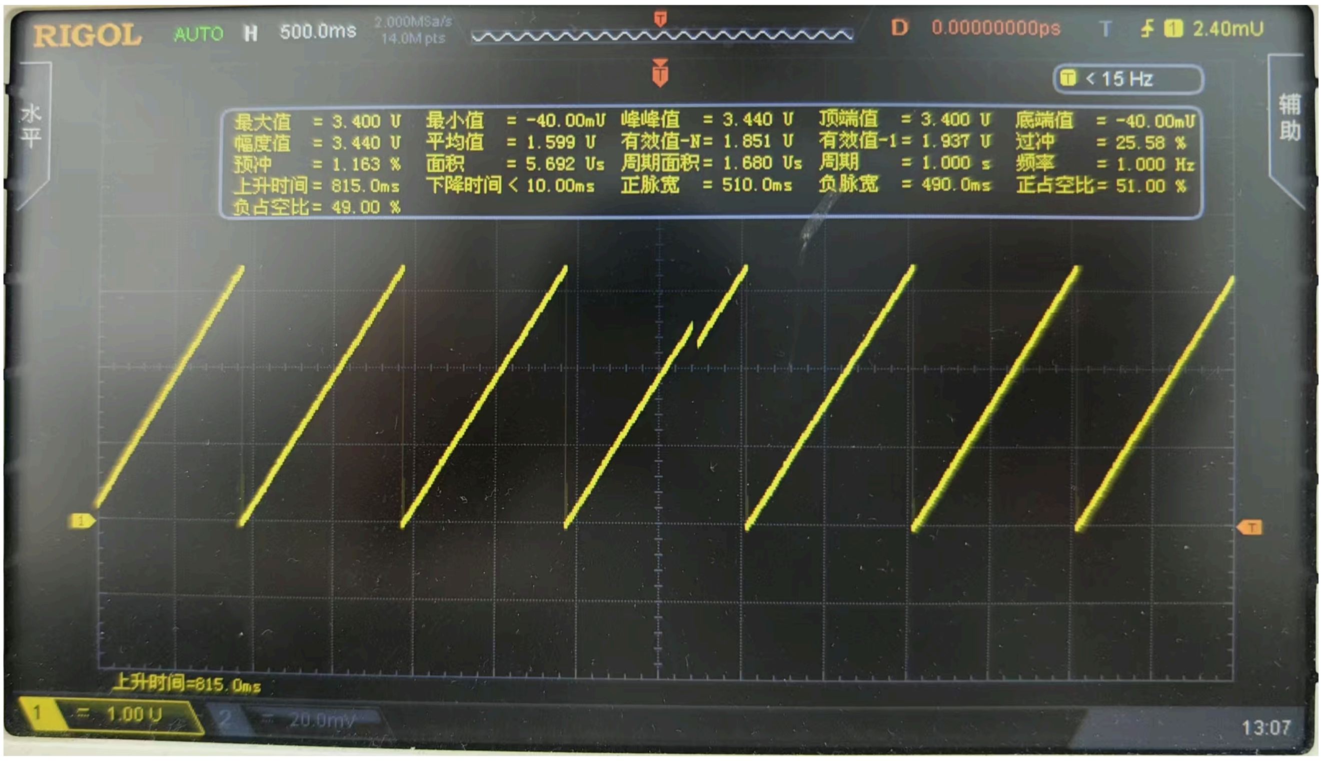


实验代码

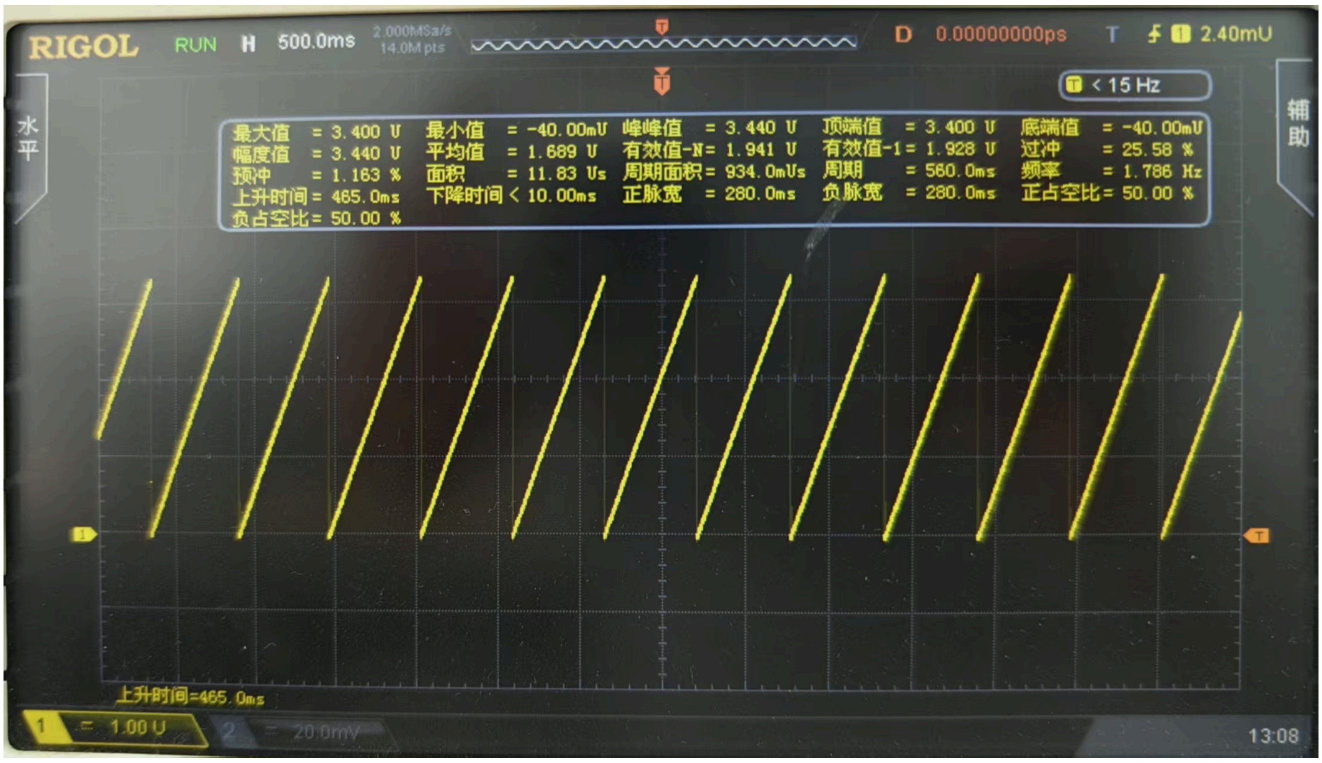
`chainsaw_wave.c`

```
//DAC:SPI0
# include "math.h"
# include "xgpio_l.h"
# include "stdio.h"
# include "xtmrctr_l.h"
# include "xspi_l.h"
# include "xintc_l.h"
# include "xil_io.h"
# include "xil_exception.h"

# define VALUE_switch 1000000 - 2
# define spi_T    46875-2
# define change_T 48958 //步长60ms

void timer_handle()  __attribute__ ((fast_interrupt));
void timer0_handle(); // 波形输出
void timer1_handle(); // 定时读取开关的值

int samples[128]; // 样本点，用于形成锯齿波
int int_times;
int T; // 开关值

int main()
{
    int status;
    Xil_Out16(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_TRI_OFFSET, 0xffff);//sw
    //定时器T0初始化：发送数据
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR + XTC_TLR_OFFSET, spi_T);
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR + XTC_TCSR_OFFSET,
XTC_CSR_INT_OCCURED_MASK | XTC_CSR_AUTO_RELOAD_MASK |
        XTC_CSR_DOWN_COUNT_MASK | XTC_CSR_LOAD_MASK |
XTC_CSR_ENABLE_INT_MASK);
    status = Xil_In32(XPAR_TMRCTR_0_BASEADDR + XTC_TCSR_OFFSET);
    status = (status & (~XTC_CSR_LOAD_MASK)) | XTC_CSR_ENABLE_TMR_MASK;
    Xil_Out32(XPAR_TMRCTR_0_BASEADDR + XTC_TCSR_OFFSET, status);

    //初始化T1：读取开关值
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET,
        Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET) & ~XTC_CSR_ENABLE_TMR_MASK);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TLR_OFFSET, VALUE_switch);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET,
        Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET) | XTC_CSR_LOAD_MASK);
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET,
        (Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET)
        & ~XTC_CSR_LOAD_MASK) | XTC_CSR_ENABLE_TMR_MASK |
XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_DOWN_COUNT_MASK);

    //spi初始化
    Xil_Out32(XPAR_SPI_0_BASEADDR + XSP_SRR_OFFSET, XSP_SRR_RESET_MASK);
    Xil_Out32(XPAR_SPI_0_BASEADDR + XSP_CR_OFFSET, XSP_CR_ENABLE_MASK |
XSP_CR_MASTER_MODE_MASK |
        XSP_CR_CLK_POLARITY_MASK | XSP_CR_TXFIFO_RESET_MASK |
XSP_CR_RXFIFO_RESET_MASK);
    Xil_Out32(XPAR_SPI_0_BASEADDR + XSP_SSR_OFFSET, 0x0);
    Xil_Out32(XPAR_SPI_0_BASEADDR + XSP_DTR_OFFSET, 0x0);


    //中断控制器初始化
    Xil_Out32(XPAR_INTC_0_BASEADDR +
XIN_IAR_OFFSET,Xil_In32(XPAR_INTC_0_BASEADDR + XIN_ISR_OFFSET));
    Xil_Out32(XPAR_INTC_0_BASEADDR +
XIN_IER_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR +
XIN_IMR_OFFSET,XPAR_AXI_TIMER_0_INTERRUPT_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR +
XIN_MER_OFFSET,XIN_INT_MASTER_ENABLE_MASK|XIN_INT_HARDWARE_ENABLE_MASK);
    Xil_Out32(XPAR_INTC_0_BASEADDR + XIN_IVAR_OFFSET + 4 *
XPAR_INTC_0_TMRCTR_0_VEC_ID, (int)timer_handle);

    microblaze_enable_interrupts();
    for (int i = 0 ; i < 128; i++)
    {
        samples[i] = 32 * i;
    }
    return 0;
}

void timer_handle() {
    int status;
    // 判断 T0 中断
    status = Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET);
    if ((status & XTC_CSR_INT_OCCURED_MASK) == XTC_CSR_INT_OCCURED_MASK) {
        timer0_handle();
    }
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TCSR_OFFSET, status);

    // 判断 T1 中断
    status = Xil_In32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET);
    if ((status & XTC_CSR_INT_OCCURED_MASK) == XTC_CSR_INT_OCCURED_MASK) {
        timer1_handle();
    }
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TIMER_COUNTER_OFFSET +
XTC_TCSR_OFFSET, status);
}

// T0 中断事务处理
void timer0_handle() {
    int_times++;
    if(int_times == 128)
       int_times = 0;
    Xil_Out32(XPAR_SPI_0_BASEADDR + XSP_DTR_OFFSET, samples[int_times]);
}
// T1 中断事务处理
void timer1_handle() {
    T = Xil_In16(XPAR_AXI_GPIO_0_BASEADDR + XGPIO_DATA_OFFSET) & 0xf;
    Xil_Out32(XPAR_AXI_TIMER_0_BASEADDR + XTC_TLR_OFFSET, spi_T + T *
change_T);
}
```

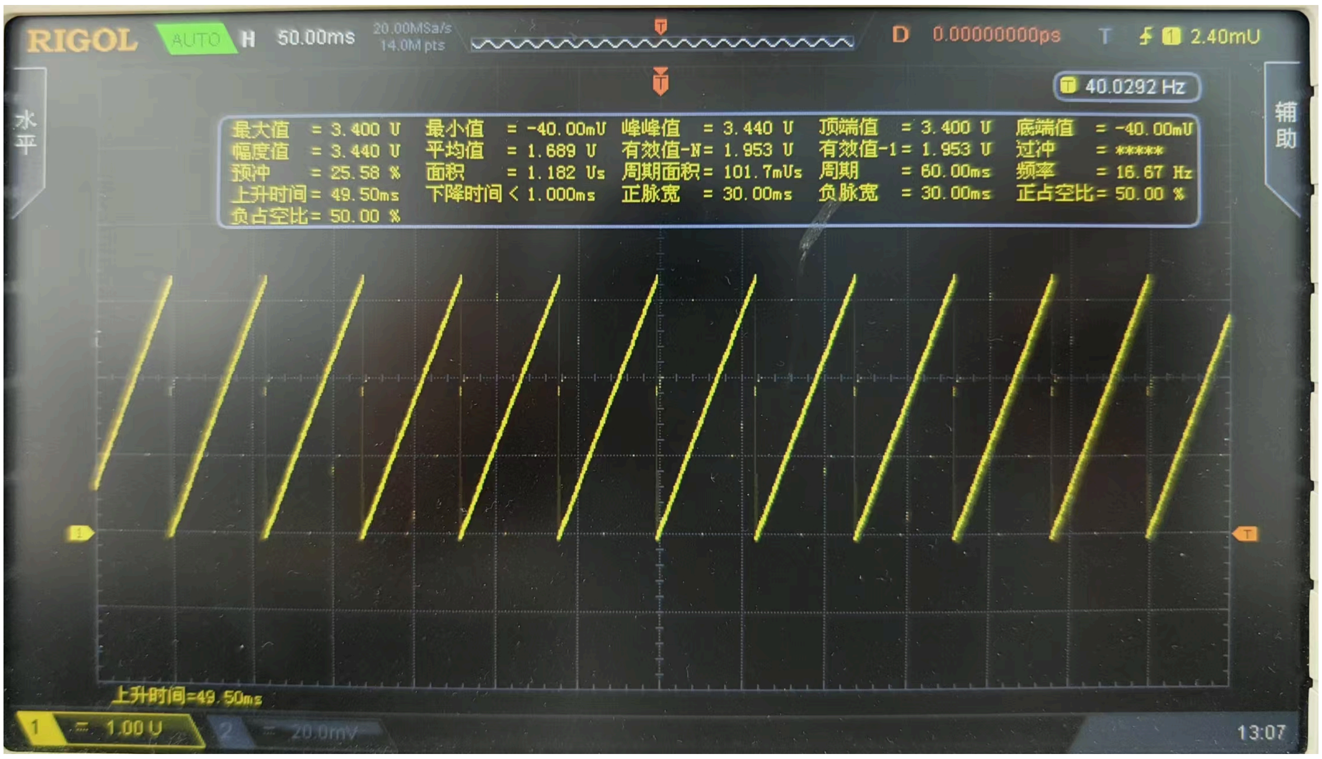
实验结果



```
最大值 = 3.400 U    最小值 = -40.00mU     峰峰值 = 3.440 U     顶端值 = 3.400 U     底端值 = -40.00mU
幅度值 = 3.440 U    平均值 = 1.599 U      有效值-N= 1.851 U    有效值-1= 1.937 U    过冲 = 25.58 %
预冲 = 1.163 %      面积 = 5.692 Us       周期面积= 1.680 Us   周期 = 1.000 s       频率 = 1.000 Hz
上升时间= 815.0ms   下降时间 < 10.00ms    正脉宽 = 510.0ms     负脉宽 = 490.0ms     正占空比= 51.00 %
负占空比= 49.00 %
```



```
最大值 = 3.400 U    最小值 = -40.00mU     峰峰值 = 3.440 U     顶端值 = 3.400 U     底端值 = -40.00mU
幅度值 = 3.440 U    平均值 = 1.689 U      有效值-N= 1.941 U    有效值-1= 1.928 U    过冲 = 25.58 %
预冲 = 1.163 %      面积 = 11.83 Us       周期面积= 934.0mUs   周期 = 560.0ms       频率 = 1.786 Hz
上升时间= 465.0ms   下降时间 < 10.00ms    正脉宽 = 280.0ms     负脉宽 = 280.0ms     正占空比= 50.00 %
负占空比= 50.00 %
```



```
最大值   = 3.400 U   最小值   = -40.00mU 峰峰值   = 3.440 U   顶端值   = 3.400 U   底端值   = -40.00mU
幅度值   = 3.440 U   平均值   = 1.689 U   有效值-N= 1.953 U   有效值-1= 1.953 U   过冲     = *****
预冲     = 25.58 %   面积     = 1.182 Us  周期面积= 101.7mUs  周期     = 60.00ms   频率     = 16.67 Hz
上升时间= 49.50ms   下降时间 < 1.000ms   正脉宽   = 30.00ms   负脉宽   = 30.00ms   正占空比= 50.00 %
负占空比= 50.00 %
```

上升时间=49.50ms