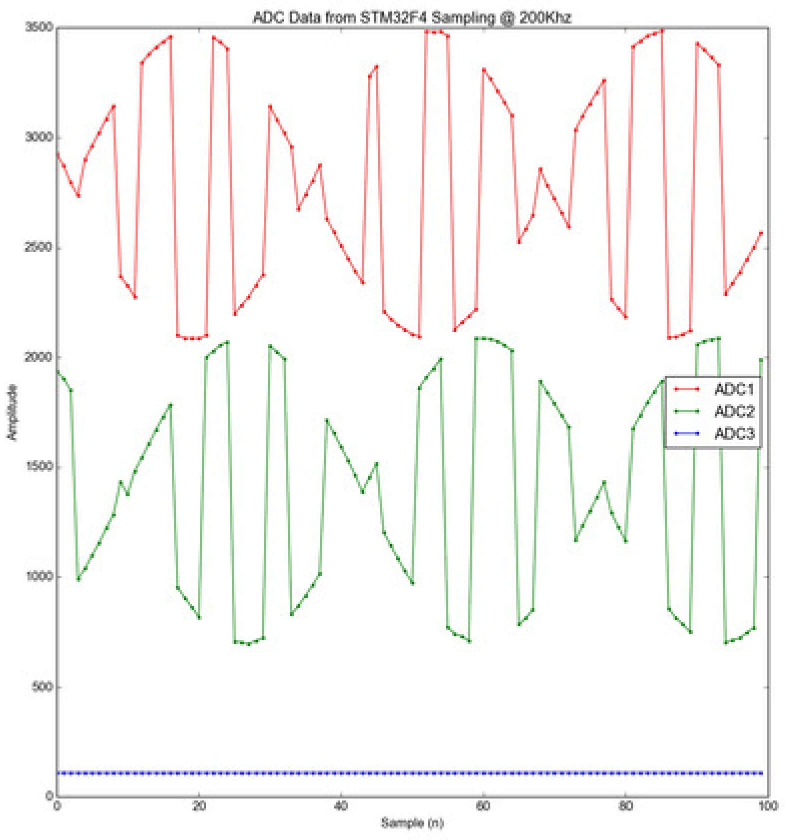
We are currently working on using the three ADCs on the STM32F407 to simultaneously sample three analog waveforms. Our target waveform is at about ~30Khz, but to illustrate our problem I have slowed the input waveforms down to 3Khz.

We have been capturing data by running our code through the debugger and exporting via gdb (using ST-UTIL in linux). This is our output. The waveform generator we are using to get these waveforms is also being measured with an oscilloscope; the sign inversion is not seen there. This leads us to believe it is being introduced somehow by the STM32F4. What is particularly curious is that the DC component of the individual signals is correct (i.e. ADC1 is offset above ADC2 and ADC3 is not connected to anything). Any suggestions would be greatly appreciated.



```
//Adapted From STM32 ADC IQ Sample @ 200 KHz (PC.1, PC.2) STM32F4 Discovery - 
sourcer32@gmail.com
// Assumptions per system_stm32f4xx.c CPU @ 168 MHz, APB2 @ 84 MHz (/2), APB1 @ 42 MHz (/4)
 #include "stm32f4_discovery.h"
#include "stm32f4xx_conf.h"

/**************************************************************************************/

void RCC_Configuration(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
}

/**************************************************************************************/

void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* ADC Channel 10 -> PC0
     ADC Channel 12 -> PC2
     ADC Channel 13 -> PC3
  */

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_2 |GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);
}

/**************************************************************************************/

void ADC_Configuration(void)
{
  ADC_CommonInitTypeDef ADC_CommonInitStructure;
  ADC_InitTypeDef ADC_InitStructure;

  /* ADC Common Init */
  ADC_CommonInitStructure.ADC_Mode = ADC_TripleMode_RegSimult;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;

  ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_1; // DMA mode 1 enabled (2 / 3 half-words one by one - 1 then 2 then 3)
  ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);

  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE; // 1 Channel
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; // Conversions Triggered
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_Rising;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T2_TRGO;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC1, &ADC_InitStructure);
  ADC_Init(ADC2, &ADC_InitStructure); // Mirror on ADC2
  ADC_Init(ADC3, &ADC_InitStructure); // Mirror on ADC3

  /* ADC1 regular channel 10 configuration */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_15Cycles); // PC1

  /* ADC2 regular channel 12 configuration */
  ADC_RegularChannelConfig(ADC2, ADC_Channel_12, 1, ADC_SampleTime_15Cycles); // PC2

  /* ADC3 regular channel 13 configuration */
  ADC_RegularChannelConfig(ADC3, ADC_Channel_13, 1, ADC_SampleTime_15Cycles); // PC3
  /* Enable DMA request after last transfer (Multi-ADC mode)  */
  ADC_MultiModeDMARequestAfterLastTransferCmd(ENABLE);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Enable ADC2 */
  ADC_Cmd(ADC2, ENABLE);

  /* Enable ADC3 */
  ADC_Cmd(ADC3, ENABLE);

}

/**************************************************************************************/

#define BUFFERSIZE 300  // I+Q 200KHz x2 HT/TC at 1KHz

__IO uint16_t ADCTripleConvertedValues[BUFFERSIZE]; // Filled as triple ADC1, ADC2, ADC3
```

```

static void DMA_Configuration(void)
{
  DMA_InitTypeDef DMA_InitStructure;

  DMA_InitStructure.DMA_Channel = DMA_Channel_0;
  DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&ADCTripleConvertedValues[0];
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)0x40012308; // CDR_ADDRESS; Packed
ADC1, ADC2 add pack for addr 3 ??
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
  DMA_InitStructure.DMA_BufferSize = BUFFERSIZE; // Count of 16-bit words
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Enable;
  DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
  DMA_Init(DMA2_Stream0, &DMA_InitStructure);


  /* Enable DMA Stream Half / Transfer Complete interrupt */
  DMA_ITConfig(DMA2_Stream0, DMA_IT_TC | DMA_IT_HT, ENABLE);

  /* DMA2_Stream0 enable */
  DMA_Cmd(DMA2_Stream0, ENABLE);
}
__IO uint16_t ADCTripleConvertedValues[BUFFERSIZE]; // Filled as triples ADC1, ADC2, ADC3

/**************************************************************************************/

void TIM2_Configuration(void)
{
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

  /* Time base configuration */
  TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
  TIM_TimeBaseStructure.TIM_Period = (84000000 / 200000) - 1; // 200 KHz, from 84 MHz
TIM2CLK (ie APB1 = HCLK/4, TIM2CLK = HCLK/2)
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

  /* TIM2 TRGO selection */
  TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update); // ADC_ExternalTrigConv_T2_TRGO

  /* TIM2 enable counter */
  TIM_Cmd(TIM2, ENABLE);
}

/**************************************************************************************/

void NVIC_Configuration(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Enable the DMA Stream IRQ Channel */
  NVIC_InitStructure.NVIC_IRQChannel = DMA2_Stream0_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

```

```
/*****************************************************************************************/

void DMA2_Stream0_IRQHandler(void) // Called at 1 KHz for 200 KHz sample rate, LED Toggles at 500 Hz
{
  /* Test on DMA Stream Half Transfer interrupt */
  if(DMA_GetITStatus(DMA2_Stream0, DMA_IT_HTIF0))
  {
    /* Clear DMA Stream Half Transfer interrupt pending bit */
    DMA_ClearITPendingBit(DMA2_Stream0, DMA_IT_HTIF0);

    /* Turn LED3 off: Half Transfer */
    STM_EVAL_LEDOff(LED3);

        // Add code here to process first half of buffer (ping)
  }

  /* Test on DMA Stream Transfer Complete interrupt */
  if(DMA_GetITStatus(DMA2_Stream0, DMA_IT_TCIF0))
  {
    /* Clear DMA Stream Transfer Complete interrupt pending bit */
    DMA_ClearITPendingBit(DMA2_Stream0, DMA_IT_TCIF0);


    /* Turn LED3 on: End of Transfer */
    STM_EVAL_LEDOn(LED3);

        // Add code here to process second half of buffer (pong)
  }
}

/*****************************************************************************************/

int main(void)
{
  RCC_Configuration();

  GPIO_Configuration();

  NVIC_Configuration();

  TIM2_Configuration();

  DMA_Configuration();

  ADC_Configuration();

  STM_EVAL_LEDInit(LED3); /* Configure LEDs to monitor program status */
  STM_EVAL_LEDInit(LED4); /* Configure LEDs to monitor program status */

  STM_EVAL_LEDOn(LED3); /* Turn LED3 on, 500 Hz means it working */

  /* Start ADC1 Software Conversion */
  ADC_SoftwareStartConv(ADC1);
  uint32_t t = 0;

  while(1) // Don't want to exit
  {
      STM_EVAL_LEDToggle(LED4);
      for(t=0; t < 5000000; t++ );
  }
}
```