I'm trying to understand DMA usage by testing a simple application. The idea is to allow USART peripheral to access read and write from and to memory directly, by using DMA technique. I'm testing this on STM32F0 Discovery board. After reading some generic literature on DMA and reference manual of this particular SOC (STM32F051XX), my understanding on DMA capable, USART operation for data reception (i.e. Data transfer direction is from Peripheral to Memory) is as follows:

1. Data arrives from external world (Say through USB Serial COM port on PC to GPIO pins on USART port. I'm using RealTerm on Windows to send and receive data). Peripheral sends, request signal to DMA controller. This leads to Interrupt request (DMA_IRQ) generation.(H/W)
2. DMA controller Ack's the requesting Peripheral. (H/W)
3. ISR associated with the DMA_IRQ gets invoked. Polling for 'transfer complete flag' starts.(S/W)
4. Bus Matrix, ask Cortex Core to relinquish the System Bus. Assuming a simple scenario here, where there are only two bus consumers; Core CPU and DMA Controller. (H/W)
5. Core relinquishes the System bus, Bus Matrix allows DMA controller to access the system bus. (H/W)
6. DMA controller access the Peripheral register. Based on DMA and Peripheral (In this case, USART1) configuration, data is transferred from and to memory(If both Tx and Rx directions were configured). (H/W)
7. This data transfer can happen over multiple separate channels (To achieve parallelism and to avoid contention, I guess). (H/W)
8. Once the data transfer (As configured during the DMA setup, such as data size, memory location, direction etc) is complete, DMA controller relinquish the bus. (H/W)
9. Bus Matrix grants, System bus access to CPU (Considering the simplest case. (H/W)
10. DMA controller sets 'Transfer complete flag' for Systems programming purposes(H/W)
11. In the ISR, this transfer complete flag is checked and once set, necessary action is taken and before leaving ISR, 'transfer complete flag' is reset.(S/W)

H/W = Handled by Hardware
S/W = Handled by Software

Based on above understanding and some sample code DMA related code distributed by STM and in this forum, Below is my simple application. The idea is to transfer some characters over USART1 interface, transfer this data into memory using DMA and toggle the LED on evaluation board to indicate the completion of transfer.

Below is the code snip:

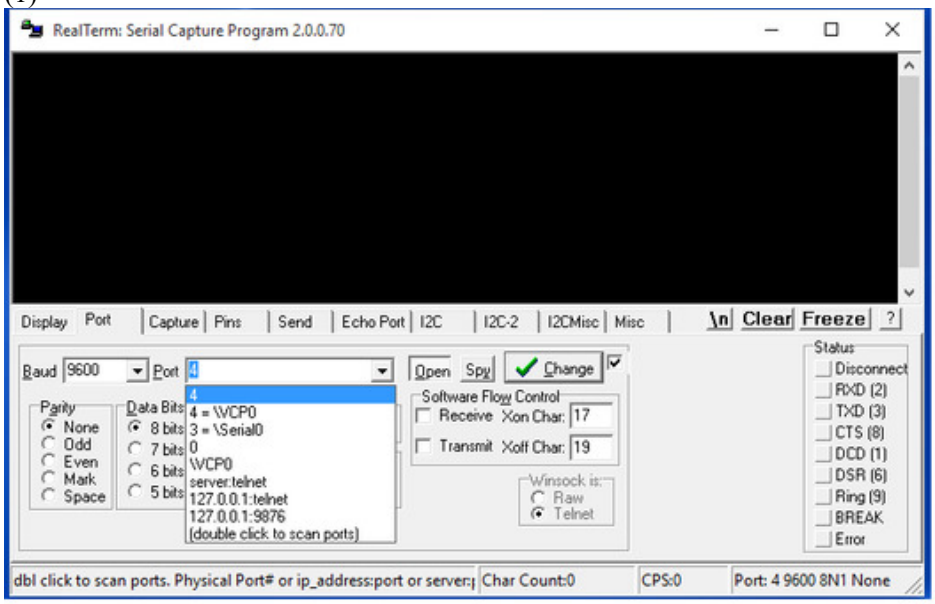
```
#include <stddef.h>
#include <stdint.h>
#include "stm32f0xx.h"
#include "stm32f0xx_it.h"

static __IO uint32_t val;

__INLINE void Configure_GPIOC(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  /* configure pins */
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;

  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);
  GPIO_Init(GPIOC, &GPIO_InitStruct);
}

__INLINE void Configure_GPIOA(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  /* configure pins */
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;

  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
  GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

```
void USART1_Configuration(void)
{
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
  USART_InitTypeDef USART_InitStructure;

  /* USART resources configuration (Clock, GPIO pins and USART registers) ----*/
  USART_InitStructure.USART_BaudRate = 9600;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

  USART_Init(USART1, &USART_InitStructure);
  /* Enable USART1 */
  USART_Cmd(USART1, ENABLE);
}

void DMA1_Configure(void)
{
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
  DMA_InitTypeDef DMA_InitStructure;

  DMA_DeInit(DMA1_Channel3);

  /* Configure Channel Parameters */
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&USART1->RDR;
  DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)val;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_InitStructure.DMA_BufferSize = 4;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  DMA_InitStructure.DMA_Priority = DMA_Priority_Low;
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

  /* Initialize the DMA Channel */
  DMA_Init(DMA1_Channel3, &DMA_InitStructure);

  /* Enable the USART's DMA receive request interface */
  USART_DMACmd(USART1, USART_DMAReq_Rx, ENABLE);

  /* Enable DMA Stream Transfer Complete interrupt */
  DMA_ITConfig(DMA1_Channel3, DMA_IT_TC, ENABLE);

  DMA_Cmd(DMA1_Channel3, ENABLE);
}

void DMA1_Channel2_3_IRQHandler(void)
{
  /* Test on DMA Transfer Complete interrupt */
  if (DMA_GetITStatus(DMA1_IT_TC3))
  {
    /* Toggle the Blue LED */
    GPIOC->ODR ^= GPIO_ODR_9;

    /* Clear DMA Transfer Complete interrupt pending bit */
    DMA_ClearITPendingBit(DMA1_IT_TC3);
  }
}

int main(void)
{
  RCC_ClocksTypeDef RCC_Clocks;

  /* SysTick end of count event each 1ms */
  RCC_GetClocksFreq(&RCC_Clocks);

  /* initialize the System Timer and its interrupt, and start the
   * System Tick Timer. Refer SysTick_Handler in startup_stm32f072.s
   * and stm32f0xx_it.c
   */
```

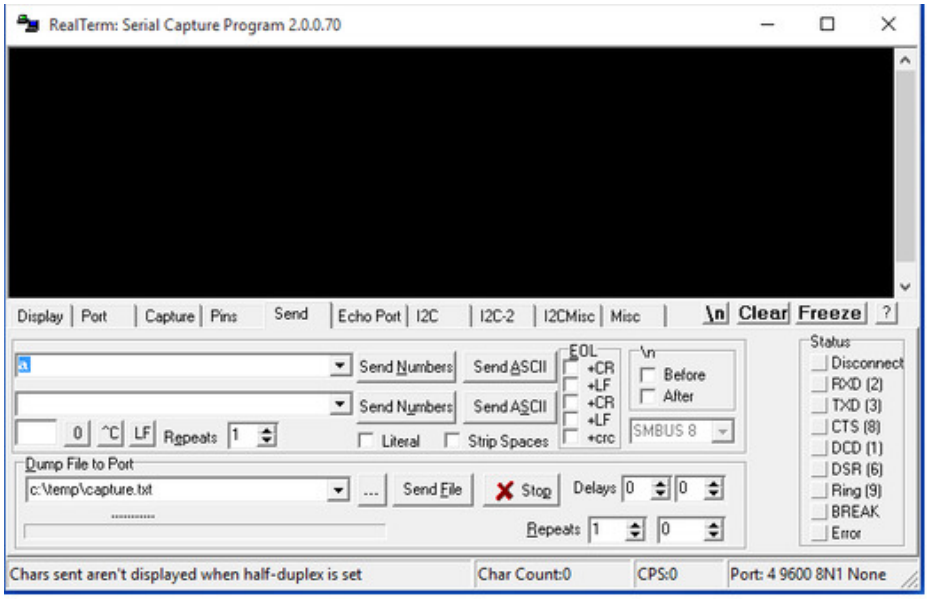
```
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 1000);

  /* Configure GPIOC port and pins: Green and Blue LED On evaluation board */
  Configure_GPIOC();

  /* Configure GPIOA port and pins: PA.9 and PA.1O as USART On evaluation board */
  Configure_GPIOA();
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_1);
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_1);

  /* Configure USART1 */
  USART1_Configuration();
  DMA1_Configure();

  /* Reference manual, pg 176:
   * Bit 10 USART1_RX_DMA_RMP
   *    0: No remap (USART1_RX DMA request mapped on DMA channel 3)
   */
  SYSCFG_DMAChannelRemapConfig(SYSCFG_DMARemap_USART1Rx, DISABLE);

  while (1)
  {
    __WFI();
  }
}
```

Compilation happens fine, but it seems that the *DMA1_Channel2_3_IRQHandler* is not getting invoked and hence I do not see the LED getting Toggled :(

Also attach here is the Screenshot of RealTerm Settings (1) and the send tab in RealTerm, where I try to send a character (2).
(1)



(2)



Thank you for your help.