Project: Soft UART design and implementation
Development environment: Cortex M3 embedded system
Editor: Tom Hua
Date: January, 15, 2012

Possible improvement to my soft UART:

Robustness –

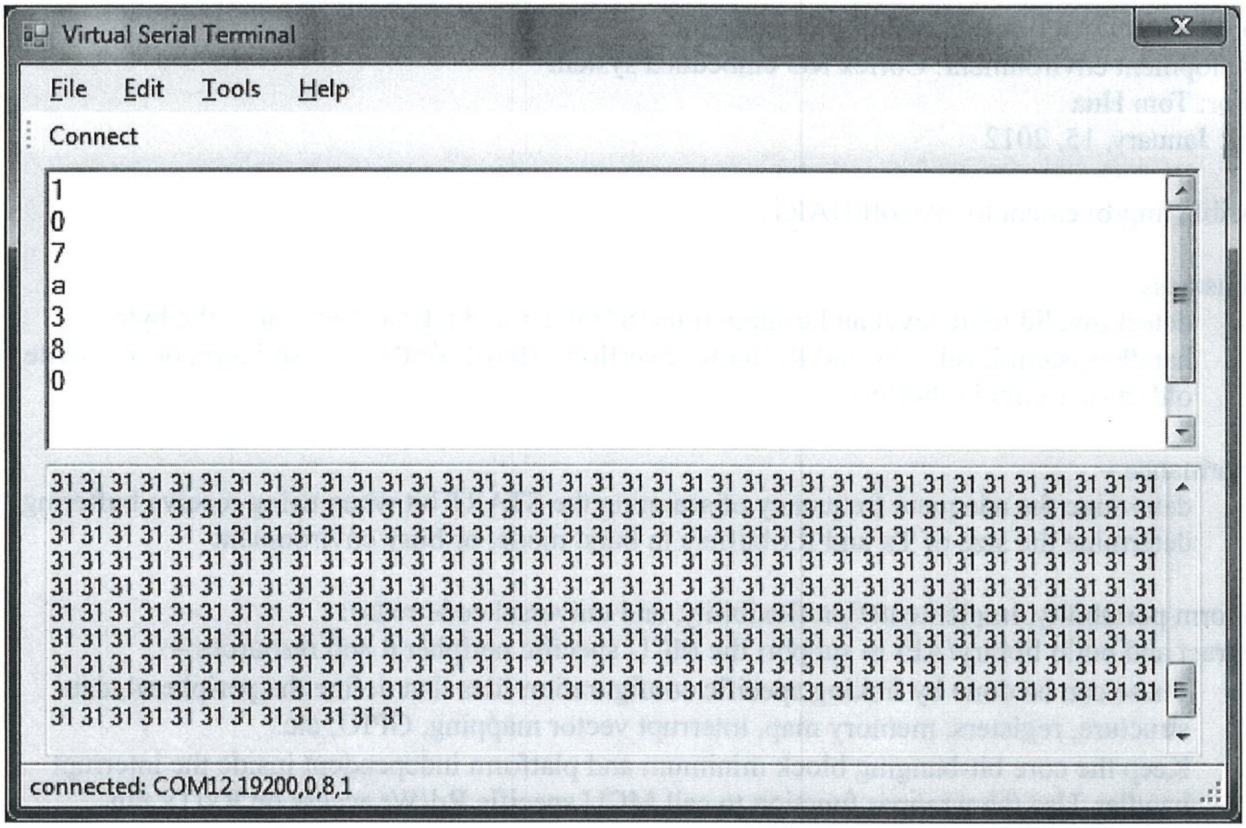
detect invalid logic level and timing of the STOP bit and take action – drop the byte.
handle system level – Tx and Rx buffer overflow – block further transmission or overwrite the oldest (use circular buffer)

Performance –

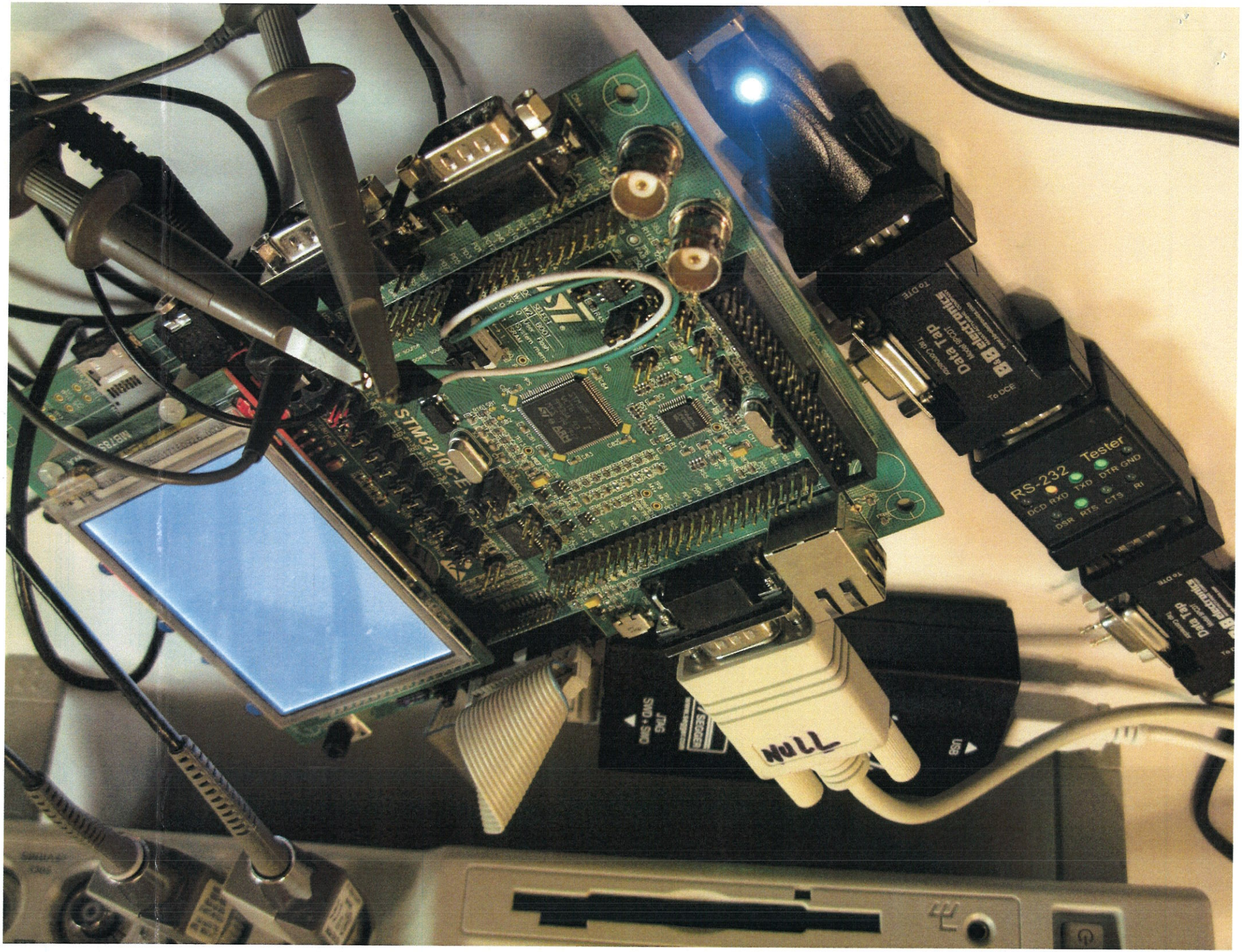
determine the adequate frequency of scanning the START bit when using receive buffering
determine the size of Tx and Rx buffers in burst mode, or busy environment

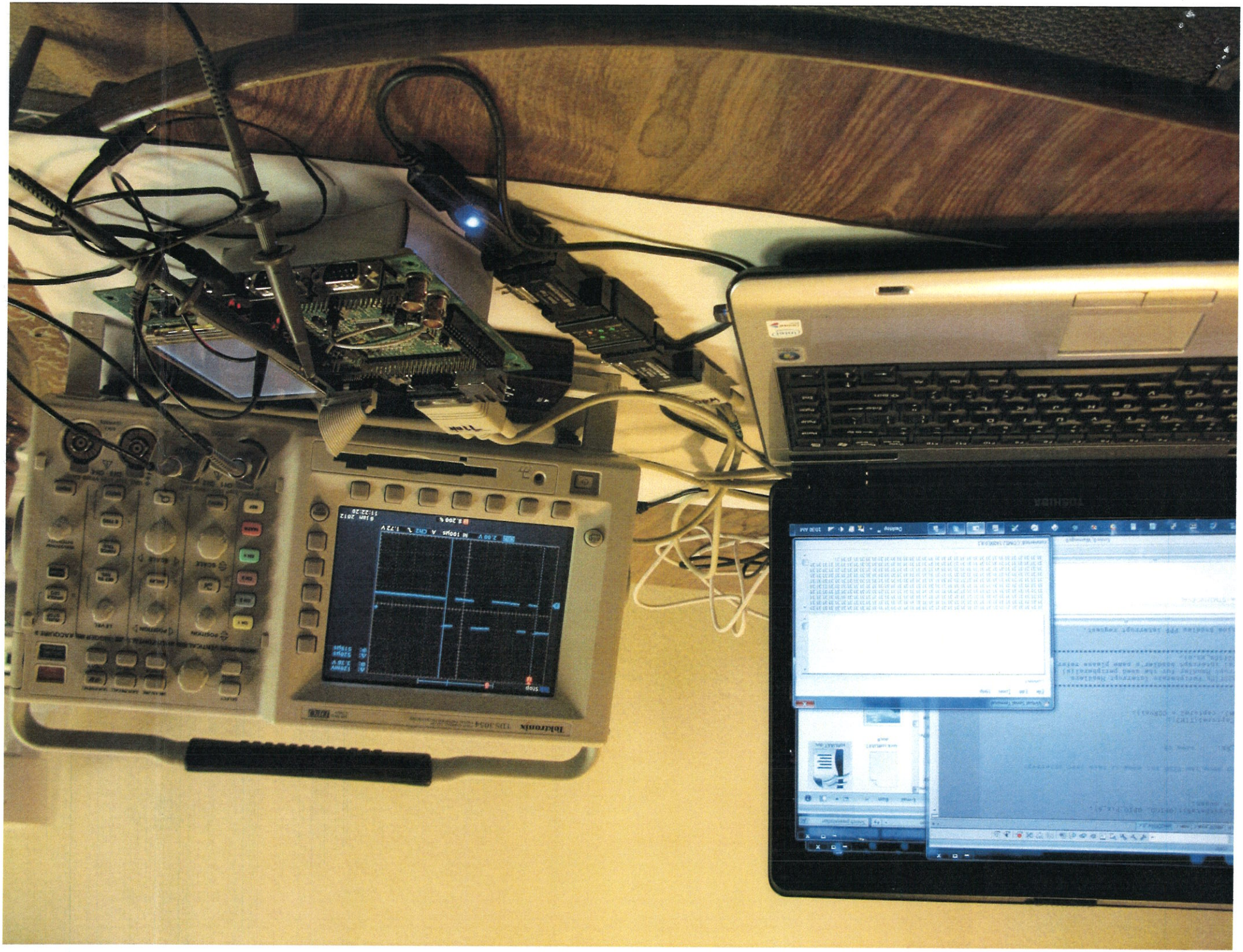
Platform portability, implementation flexibility, and universal code reuse:
Abstract and build library/API to support the MCU specific peripheral and resources –

- These can be done by linking specific configuration files that define the peripheral, data structure, registers, memory map, interrupt vector mapping, GPIO, etc.
- Keep the core bit-banging block minimum and platform independent inside the interrupt handler. Use the wrapper function to call MCU specific Rd/Wr access on Rx/Tx pin
- Timer and interrupt configuration for the required Baud rate –
  For example, theses two generic functions are the wrapper of MCU specific routines:
  TimerInit(MasterClk, TimerIDs, baud_rate)
  It configures the operation mode, count parameter value (C = TIMxCLK / baud)
  pass the calculated values into the MCU specific routines for configurations
  TimerStart(TimerID) is used to setup the timer and enable the interrupt
- In the interrupt handler bit-banging logic is based on the handshaking protocol, examples:
  generic TxConfig(protocol_struct*) sets up protocol – parity, number of data bits in the frame, etc. Then the Tx/Rx interrupt uses the structure parameter to do the job
- START bit detection, Tx/Rx buffering logic are also platform independent regardless any MCU
- Configure and rd/wr the GPIO pins used for Tx/Rx:
  GPIOconfig(IO_typedef* gpio) where gpio is a structure defines the pins and mode
  This generic function should call the MCU specific GPIO functions in their files.
  Implement a platform independent Wr_TxLine(IO_typedef* gpio) wrapper function
  Implement a platform independent Rd_RxLine(IO_typedef* gpio) wrapper function
- Global variables shared between main task and Tx/Rx tasks should be kept minimum.



This is my version of serial data terminal I used for testing. The upper text-box field is for user to type in the data to send. The lower text-box field shows the received data in real time, and configured to show in hex value.


STM3210C-EVAL
B+B electronics
Data Tap
To DCE
To DTE
RS-232 Tester
DCD RXD TXD DTR GND
DSR RTS CTS RI
NULL
USB


Tektronix
TDS3054

```
/**
  ******************************************************************************
  * @file TIM/TimeBase/main.c
  * @author  MCD Application Team
  * @version V3.1.2
  * @date    09/28/2009
  * @brief   Main program body
  ******************************************************************************
  * @copy
  *
  * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
  * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE
  * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY
  * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING
  * FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE
  * CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
  *
  * <h2><center>&copy; COPYRIGHT 2009 STMicroelectronics</center></h2>
  */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

#include "stm32f10x.h"
#include "stm32_eval.h"

/** @addtogroup STM32F10x_StdPeriph_Examples
  * @{
  */

/** @addtogroup TIM_TimeBase
  * @{
  */

int RxIdle;
uint8_t Rx_byte = 0;

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
TIM_OCInitTypeDef  TIM_OCInitStructure;
__IO uint16_t CCRval  = 937;     // CCRval = TIMxCLK (Hz) / 19200 Hz
int e_ReadytoSend;
int e_DataReady;
volatile uint8_t PD6_data = 1;

static __IO uint32_t TimingDelay;

ErrorStatus HSEStartUpStatus;
```

```
/* Private function prototypes -----------------------------------------------*/
void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);



/* Private functions ---------------------------------------------------------*/

/**
  * @brief   Main program
  * @param  None
  * @retval None
  */
int main(void)
{
    e_ReadytoSend = FALSE;
    e_DataReady   = FALSE;
    RxIdle = TRUE;

    /* SysTick end of count event each 10ms with input clock equal to 18 MHz
    // SysTick is (HCLK/8, default) */
    SysTick_Config(HSE_Value / 14000);  // generate an interrupt every 25usec

    /* System Clocks Configuration */
    RCC_Configuration();

    /* NVIC Configuration */
    NVIC_Configuration();

    /* GPIO Configuration */
    GPIO_Configuration();

    GPIO_WriteBit(GPIOD, GPIO_Pin_5, Bit_RESET);
    Delay(1000);

  /* here RCC_Configuration() sets PCLK1 = HCLK/4 = 18 MHz
    because it calls: RCC_PCLK1Config(RCC_HCLK_Div4); // APB1_prescaler: 4
    from the manual, since APB1_prescaler is not 1 then the clock named
    TIM2CLK in Timer config block will be 2x of the PCLK1, 36 MHz.
    And the
  */
  /* ----------------------------------------------------------------
    TIMx Configuration: Output Compare Timing Mode:
    TIMxCLK = 36 MHz, Prescaler = 1, CCRval = 937
    TIMx counter clock = TIMxCLK / prescaler = 18 MHz
    CC1 update rate = TIMx counter clock / CCRval = 19200 Hz
    or, given required baud rate of 19200 bps:
    CCRval = TIMxCLK (Hz) / 19200 Hz
  ---------------------------------------------------------------- */

    /* Time base configuration */
```

```
    TIM_TimeBaseStructure.TIM_Period = 65535;
    TIM_TimeBaseStructure.TIM_Prescaler = 0;    // TIM_PrescalerConfig() sets it
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

    /* Prescaler configuration  prescaler:1 overwrites the previous value */
    TIM_PrescalerConfig(TIM2, 1, TIM_PSCReloadMode_Immediate); //Tim2Clk: 18 MHz
    TIM_PrescalerConfig(TIM3, 1, TIM_PSCReloadMode_Immediate); //Tim3Clk: 18 MHz

    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCInitStructure.TIM_Pulse       = CCRval;
    TIM_OCInitStructure.TIM_OCPolarity  = TIM_OCPolarity_High;

    /* Output Compare Timing Mode configuration: Timer 2 */
            TIM_OC1Init(TIM2,  &TIM_OCInitStructure);
    TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);

    /* Output Compare Timing Mode configuration: Timer 3 */
             TIM_OC1Init(TIM3,  &TIM_OCInitStructure);
     TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Disable);

    /* enable counter */
    TIM_Cmd(TIM2, ENABLE);
    TIM_Cmd(TIM3, ENABLE);

    Delay(1000);

    e_ReadytoSend = FALSE;
    /* TIM IT enable */
    TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE);
    TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE);

    RxIdle   = TRUE;
    Delay(1);

    while (1)
    {


        if (RxIdle == TRUE)
        {   // this is state 1, the logic in SysTick_Handler depends on PD6_data
            // 1st one: detect the 1-to-0 transition
            PD6_data = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
        }

        GPIO_WriteBit(GPIOD, GPIO_Pin_4,
                      (BitAction)(Rx_byte & 1));        // LED4
        GPIO_WriteBit(GPIOD, GPIO_Pin_3,
```

```
                    (BitAction)((Rx_byte >> 1) & 1)); // LED3
        GPIO_WriteBit(GPIOD, GPIO_Pin_13,
                    (BitAction)((Rx_byte >> 2) & 1)); // LED2
        GPIO_WriteBit(GPIOD, GPIO_Pin_7,
                    (BitAction)((Rx_byte >> 3) & 1)); // LED1
    }
}

/**
  * @brief  Configures the different system clocks.
  * @param  None
  * @retval None
  */
void RCC_Configuration(void)
{
  /* Setup the microcontroller system. Initialize the Embedded Flash Interface,
     initialize the PLL and update the SystemFrequency variable. */
  SystemInit();

  /* PCLK1 = HCLK/4 = 18 MHz*/
  RCC_PCLK1Config(RCC_HCLK_Div4);

  /* TIM2 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
  /* TIM2 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
  /* GPIOC clock enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
}

/**
  * @brief  Configure the GPIOD Pins.
  * @param  None
  * @retval None
  */
void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructureOut;
  GPIO_InitTypeDef GPIO_InitStructureIn;

  // pin 5 for Tx
  GPIO_InitStructureOut.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_3 | GPIO_Pin_4 |
                                   GPIO_Pin_7 | GPIO_Pin_13;
  GPIO_InitStructureOut.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructureOut.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOD, &GPIO_InitStructureOut);

  GPIO_InitStructureIn.GPIO_Pin = GPIO_Pin_6;
  GPIO_InitStructureIn.GPIO_Mode = GPIO_Mode_IPU;
  GPIO_Init(GPIOD, &GPIO_InitStructureIn);
```

```
  /* GPIOC Configuration: as input */
  // pint 13 is a push button to start Tx, pin 8 for Rx
  /*
  GPIO_InitStructureIn.GPIO_Pin  = GPIO_Pin_13;
  GPIO_InitStructureIn.GPIO_Mode = GPIO_Mode_IPU;
  GPIO_Init(GPIOC, &GPIO_InitStructureIn);

  GPIO_InitStructureIn.GPIO_Pin  = GPIO_Pin_8;
  GPIO_InitStructureIn.GPIO_Mode = GPIO_Mode_IPU;
  GPIO_Init(GPIOC, &GPIO_InitStructureIn);
  */
}


/**
  * @brief  Configure the nested vectored interrupt controller.
  * @param  None
  * @retval None
  */
void NVIC_Configuration(void)
{
  NVIC_InitTypeDef NVIC_InitStructure1;
  NVIC_InitTypeDef NVIC_InitStructure2;

  /* Enable the TIM2 gloabal Interrupt */
  NVIC_InitStructure1.NVIC_IRQChannel = TIM2_IRQn;
  NVIC_InitStructure1.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure1.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure1.NVIC_IRQChannelCmd = ENABLE;

  /* Enable the TIM2 gloabal Interrupt */
  NVIC_InitStructure2.NVIC_IRQChannel = TIM3_IRQn;
  NVIC_InitStructure2.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure2.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure2.NVIC_IRQChannelCmd = ENABLE;

  NVIC_Init(&NVIC_InitStructure1);
  NVIC_Init(&NVIC_InitStructure2);
}


/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
void Delay(unsigned int nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);
}
```

```
/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }
}

#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *   where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  while (1)
  {}
}
#endif

/**
  * @}
  */

/**
  * @}
  */

/******************* (C) COPYRIGHT 2009 STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * @file    TIM/TimeBase/stm32f10x_it.c
  * @author  MCD Application Team
  * @version V3.1.2
  * @date    09/28/2009
  * @brief   Main Interrupt Service Routines.
  *          This file provides template for all exceptions handler and peripherals
  *          interrupt service routine.
  ******************************************************************************
  * @copy
  *
  * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
  * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE
  * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY
  * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING
  * FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE
  * CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
  *
  * <h2><center>&copy; COPYRIGHT 2009 STMicroelectronics</center></h2>
  */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f10x_it.h"

/** @addtogroup STM32F10x_StdPeriph_Examples
  * @{
  */

/** @addtogroup TIM_TimeBase
  * @{
  */

/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/
#define START   0x80
#define STOP    0x01
#define IDLE    0x00

#define STARTBIT    0
#define STOPBIT     1

#define TXDATAFRAME(byte) {~((byte << 1) | STOPBIT) }

/* Private macro -------------------------------------------------------------*/

extern __IO uint16_t CCRval;
extern    int e_ReadytoSend;
extern uint8_t Rx_byte;
```

```
/* Private variables ---------------------------------------------------------*/
static int8_t data_Rx = 0;
static int TxIdle = TRUE;
static uint8_t nWait2Ticks = 0;
extern volatile uint8_t PD6_data;
static unsigned char data_Tx = 0x30;    // my test byte

// STARTBIT + data_Tx + STOPBIT: bit9=1, bit[8.1]=data_Tx, bit0=0
// the 10-bit data frame going into the UART then is: // 1 00110000 0
static unsigned int    Tx_frame = 0x0260;  // 10 0110 0000
// The transmission sequence from LSB is (P, next the data_Tx, then S):
// This is the frame going into the UART that it inverses all bits on pin3
// The UART Receiver inverses its data logic level

static volatile uint16_t PC13_UserKey = 0xFFFF;
volatile uint8_t bit = 0;


/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************/
/*            Cortex-M3 Processor Exceptions Handlers                         */
/******************************************************************************/

/**
  * @brief  This function handles NMI exception.
  * @param  None
  * @retval None
  */
void NMI_Handler(void)
{
}

/**
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @retval None
  */
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {}
}

/**
  * @brief  This function handles Memory Manage exception.
  * @param  None
  * @retval None
  */
void MemManage_Handler(void)
```

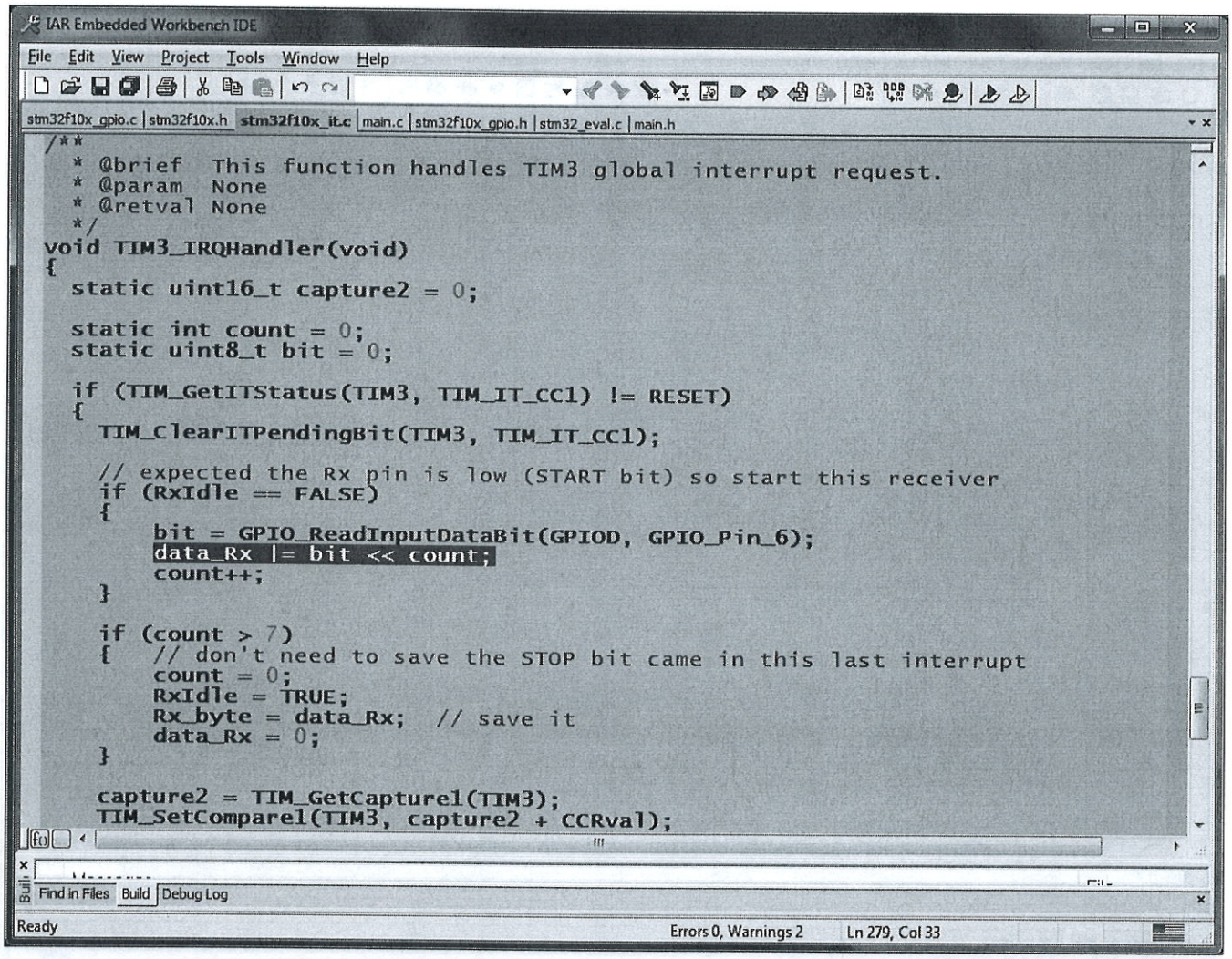
```
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {}
}

/**
  * @brief  This function handles Bus Fault exception.
  * @param  None
  * @retval None
  */
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {}
}

/**
  * @brief  This function handles Usage Fault exception.
  * @param  None
  * @retval None
  */
void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {}
}

/**
  * @brief  This function handles Debug Monitor exception.
  * @param  None
  * @retval None
  */
void DebugMon_Handler(void)
{}

/**
  * @brief  This function handles SVCall exception.
  * @param  None
  * @retval None
  */
void SVC_Handler(void)
{}

/**
  * @brief  This function handles PendSV_Handler exception.
  * @param  None
  * @retval None
  */
void PendSV_Handler(void)
```

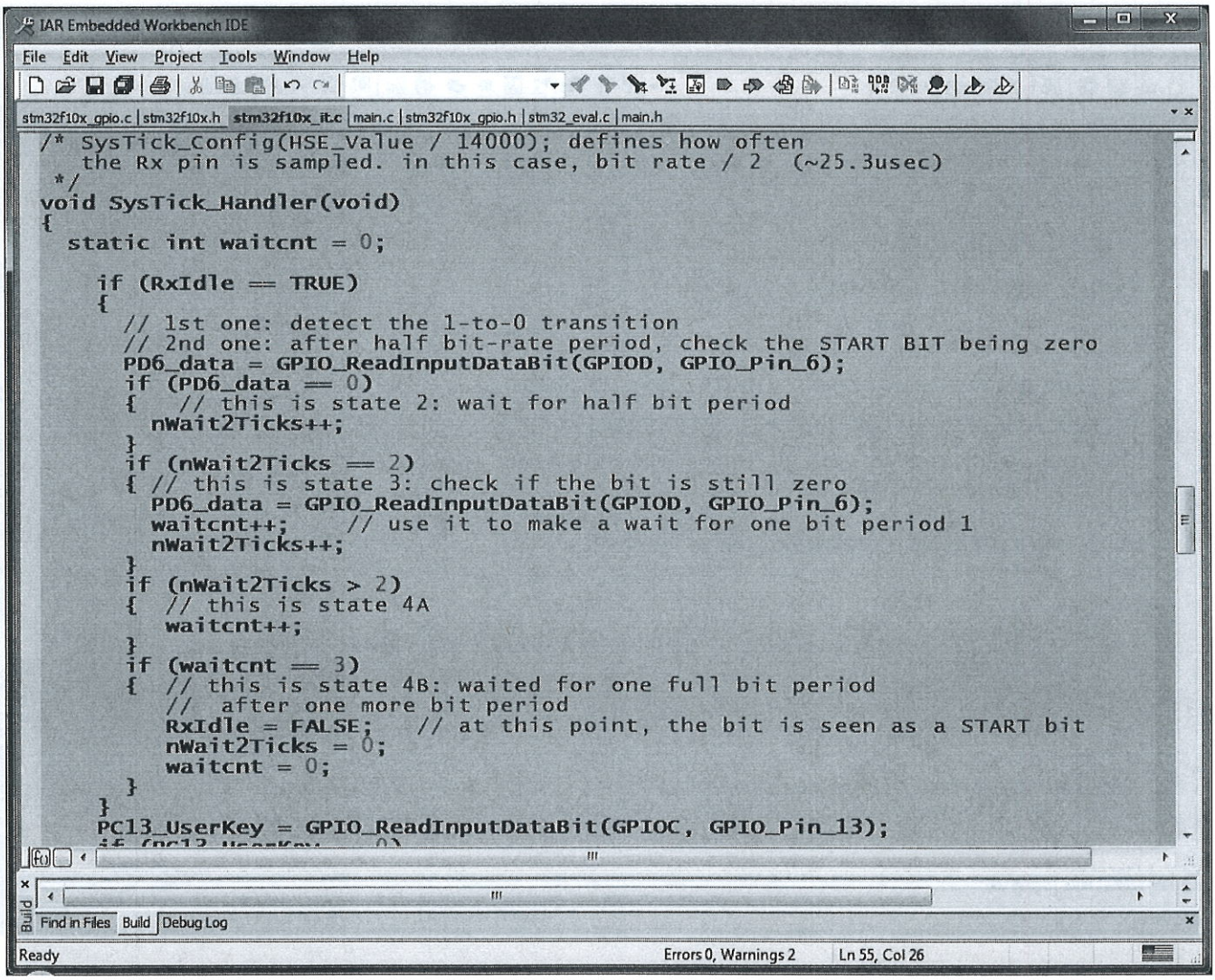
```
{}

/**
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @retval None
  */
/* SysTick_Config(HSE_Value / 14000); defines how often
   the Rx pin is sampled. in this case, bit rate / 2  (~25.3usec)
 */
void SysTick_Handler(void)
{
  static int waitcnt = 0;

    if (RxIdle == TRUE)
    {
      // 1st one: detect the 1-to-0 transition
      // 2nd one: after half bit-rate period, check the START BIT being zero
      PD6_data = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
      if (PD6_data == 0)
      {   // this is state 2: wait for half bit period
        nWait2Ticks++;
      }
      if (nWait2Ticks == 2)
      { // this is state 3: check if the bit is still zero
        PD6_data = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
        waitcnt++;    // use it to make a wait for one bit period 1
        nWait2Ticks++;
      }
      if (nWait2Ticks > 2)
      {  // this is state 4A
         waitcnt++;
      }
      if (waitcnt == 3)
      {  // this is state 4B: waited for one full bit period
         //  after one more bit period
         RxIdle = FALSE;   // at this point, the bit is seen as a START bit
         nWait2Ticks = 0;
         waitcnt = 0;
      }
    }
    PC13_UserKey = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13);
    if (PC13_UserKey == 0)
    {
      e_ReadytoSend = TRUE;
    }
    TimingDelay_Decrement();

}

/******************************************************************************/
/*            STM32F10x Peripherals Interrupt Handlers                        */
```

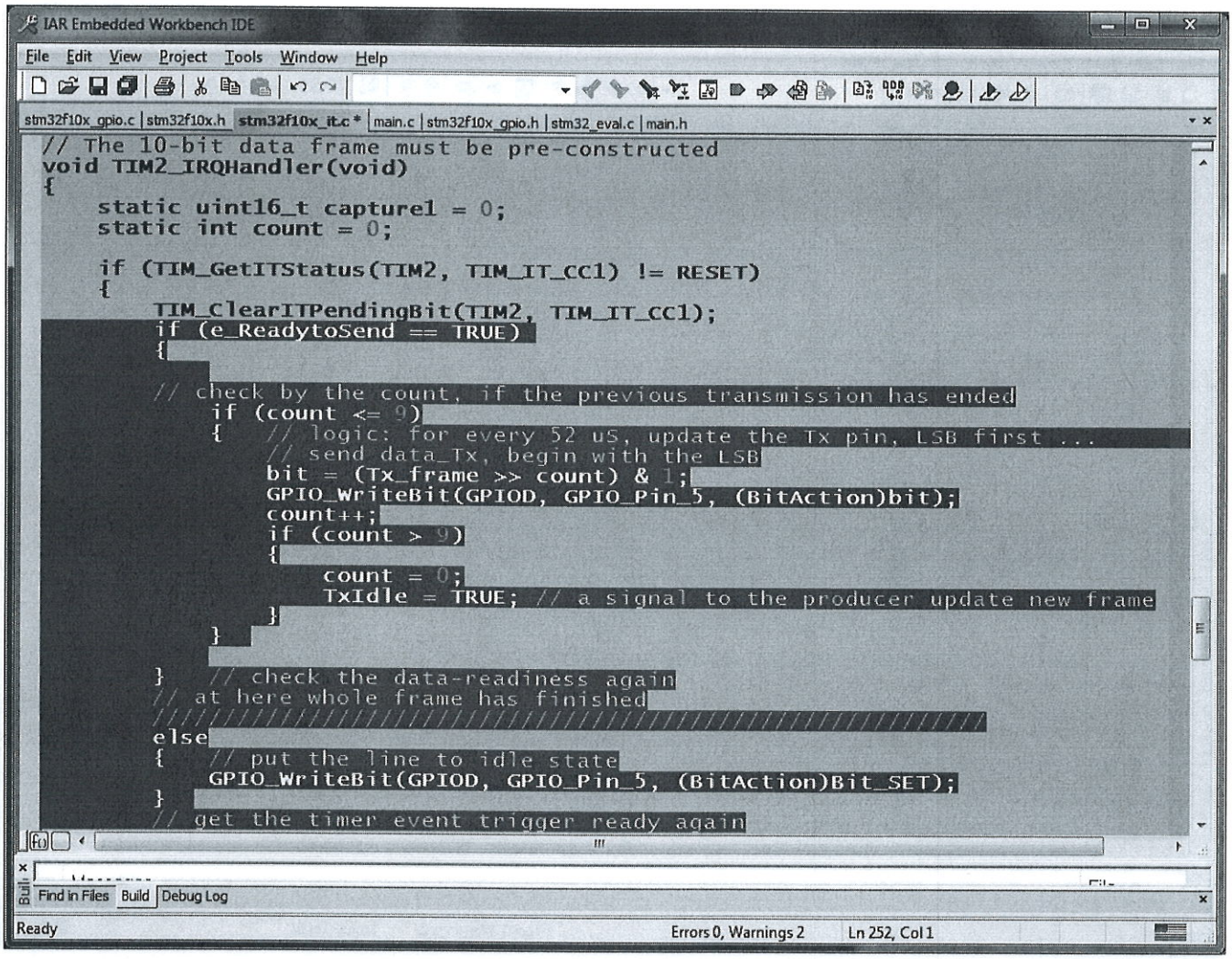
```
/******************************************************************************/

/**
  * @brief  This function handles TIM2 global interrupt request.
  * @param  None
  * @retval None
  */
// signal level inversion (logic 1 is -12V and logic 0 is +12V).
// RS232 protocol logic level:
// idle state:  for more than two clock cycle
// Start bit  one clock cycle
// stop  bit  one cycle
// The 10-bit data frame must be pre-constructed
void TIM2_IRQHandler(void)
{
    static uint16_t capture1 = 0;
    static int count = 0;

    if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
    {
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
        if (e_ReadytoSend == TRUE)
        {

        // check by the count, if the previous transmission has ended
            if (count <= 9)
            {   // logic: for every 52 uS, update the Tx pin, LSB first ...
                // send data_Tx, begin with the LSB
                bit = (Tx_frame >> count) & 1;
                GPIO_WriteBit(GPIOD, GPIO_Pin_5, (BitAction)bit);
                count++;
                if (count > 9)
                {
                    count = 0;
                    TxIdle = TRUE; // a signal to the producer update new frame
                }
            }

        }   // check the data-readiness again
        // at here whole frame has finished
        ////////////////////////////////////////////////////////
        else
        {   // put the line to idle state
            GPIO_WriteBit(GPIOD, GPIO_Pin_5, (BitAction)Bit_SET);
        }
        // get the timer event trigger ready again
        capture1 = TIM_GetCapture1(TIM2);
        TIM_SetCompare1(TIM2, capture1 + CCRval);
    }

}
```

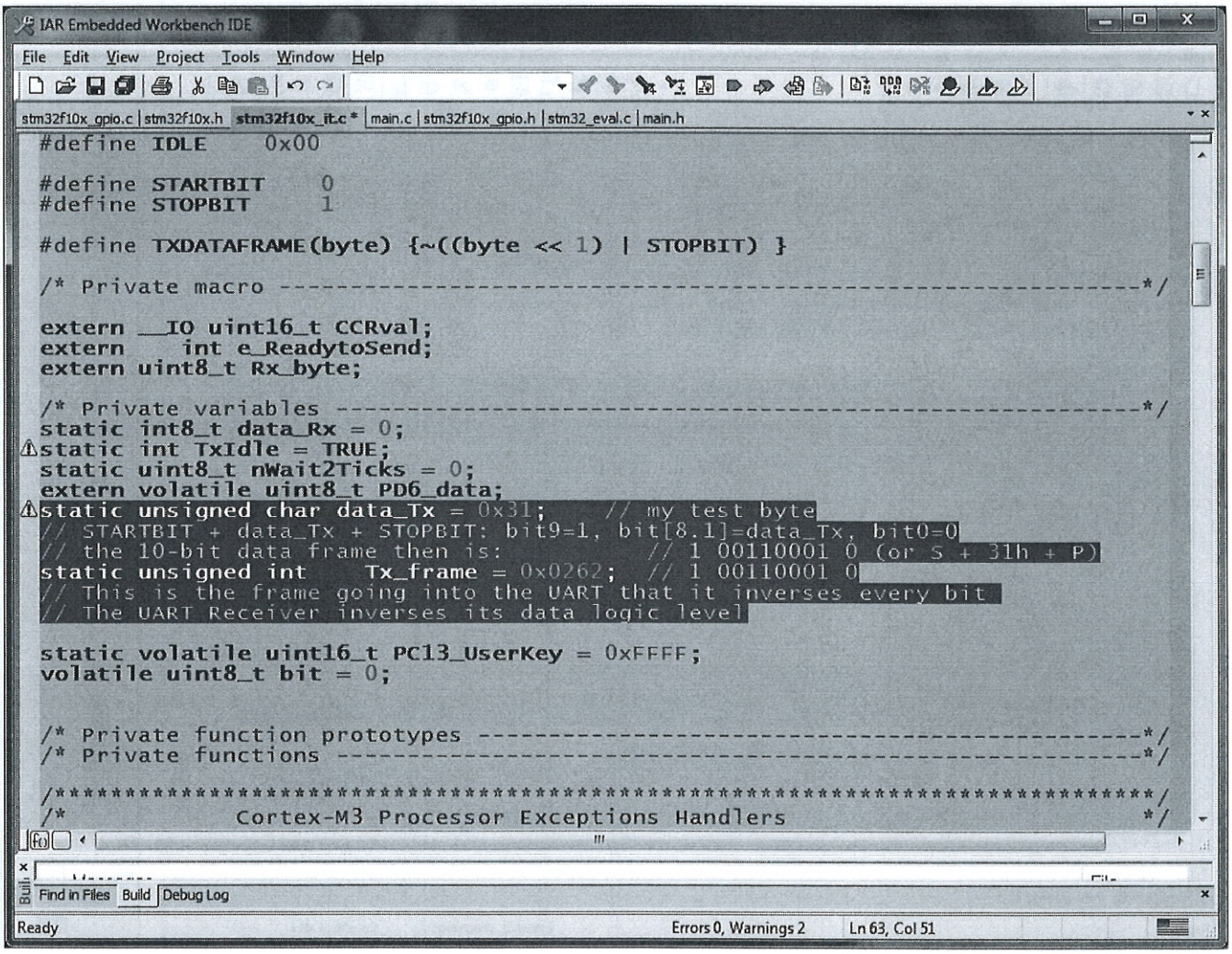
```
/**
  * @brief  This function handles TIM3 global interrupt request.
  * @param  None
  * @retval None
  */
void TIM3_IRQHandler(void)
{
  static uint16_t capture2 = 0;

  static int count = 0;
  static uint8_t bit = 0;

  if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET)
  {
    TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);

    // expected the Rx pin is low (START bit) so start this receiver
    if (RxIdle == FALSE)
    {
        bit = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
        data_Rx |= bit << count;
        count++;
    }

    if (count > 7)
    {   // don't need to save the STOP bit came in this last interrupt
        count = 0;
        RxIdle = TRUE;
        Rx_byte = data_Rx;  // save it
        data_Rx = 0;
    }

    capture2 = TIM_GetCapture1(TIM3);
    TIM_SetCompare1(TIM3, capture2 + CCRval);
  }
}

/******************************************************************************/
/*                 STM32F10x Peripherals Interrupt Handlers                   */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f10x_xx.s).                                            */
/******************************************************************************/

/**
  * @brief  This function handles PPP interrupt request.
  * @param  None
  * @retval None
  */
/*void PPP_IRQHandler(void)
{
}*/
```

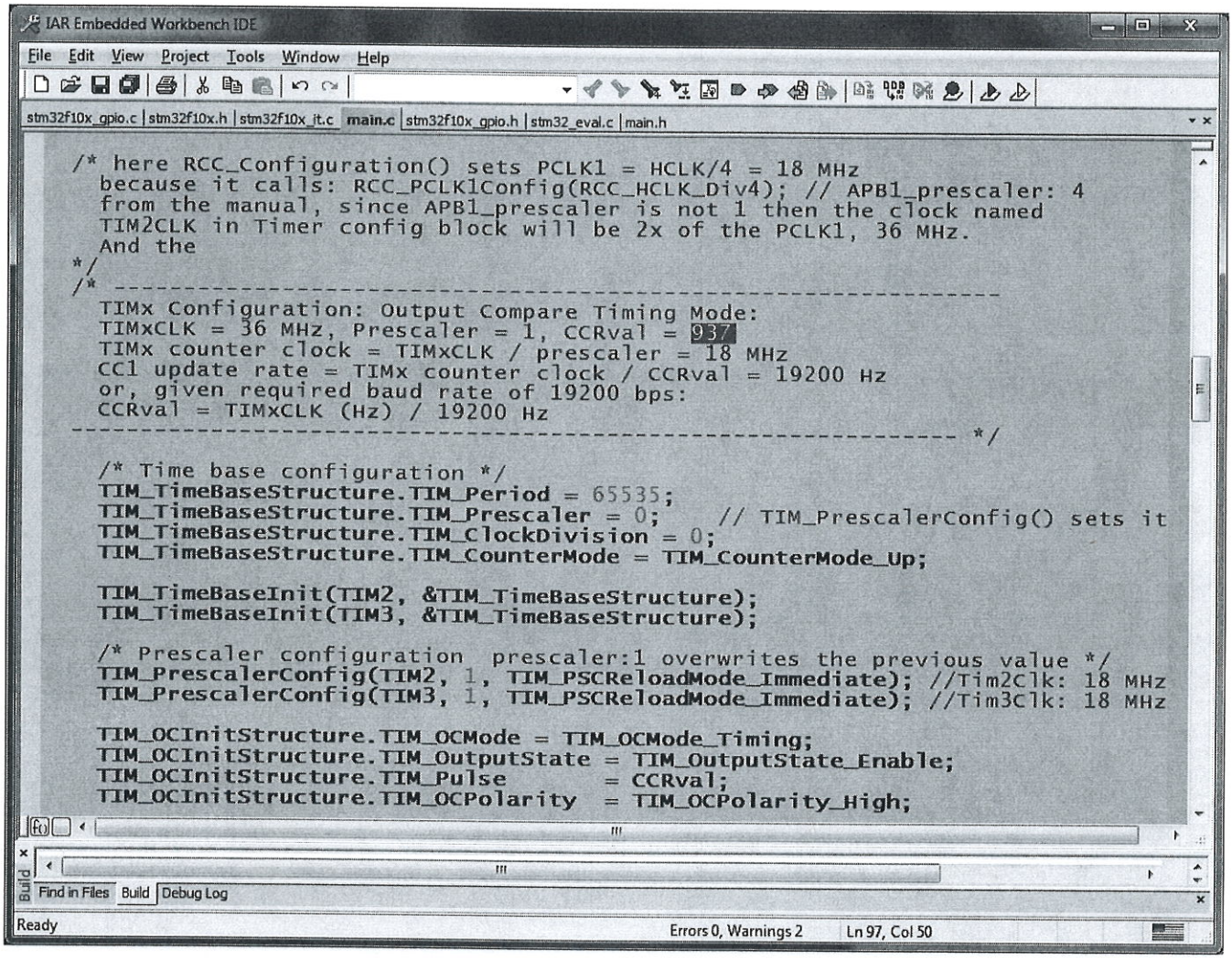
```
/**
  * @}
  */

/**
  * @}
  */

/****************** unused:


////////////////////////////////////////

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

/******************* (C) COPYRIGHT 2009 STMicroelectronics *****END OF FILE****/
```


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
/**
* @brief This function handles TIM3 global interrupt request.
* @param None
* @retval None
*/
void TIM3_IRQHandler(void)
{
static uint16_t capture2 = 0;
static int count = 0;
static uint8_t bit = 0;
if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET)
{
TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
// expected the Rx pin is low (START bit) so start this receiver
if (RxIdle == FALSE)
{
bit = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
data_Rx |= bit << count;
count++;
}
if (count > 7)
{ // don't need to save the STOP bit came in this last interrupt
count = 0;
RxIdle = TRUE;
Rx_byte = data_Rx; // save it
data_Rx = 0;
}
capture2 = TIM_GetCapture1(TIM3);
TIM_SetCompare1(TIM3, capture2 + CCRval);
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 279, Col 33


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
/* SysTick_Config(HSE_Value / 14000); defines how often
the Rx pin is sampled. in this case, bit rate / 2 (~25.3usec)
*/
void SysTick_Handler(void)
{
static int waitcnt = 0;
if (RxIdle == TRUE)
{
// 1st one: detect the 1-to-0 transition
// 2nd one: after half bit-rate period, check the START BIT being zero
PD6_data = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
if (PD6_data == 0)
{ // this is state 2: wait for half bit period
nWait2Ticks++;
}
if (nWait2Ticks == 2)
{ // this is state 3: check if the bit is still zero
PD6_data = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
waitcnt++; // use it to make a wait for one bit period 1
nWait2Ticks++;
}
if (nWait2Ticks > 2)
{ // this is state 4A
waitcnt++;
}
if (waitcnt == 3)
{ // this is state 4B: waited for one full bit period
// after one more bit period
RxIdle = FALSE; // at this point, the bit is seen as a START bit
nWait2Ticks = 0;
waitcnt = 0;
}
}
PC13_UserKey = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13);
Build
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 55, Col 26


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c * | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
// The 10-bit data frame must be pre-constructed
void TIM2_IRQHandler(void)
{
static uint16_t capture1 = 0;
static int count = 0;
if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
{
TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
if (e_ReadytoSend == TRUE)
{
// check by the count, if the previous transmission has ended
if (count <= 9)
{ // logic: for every 52 uS, update the Tx pin, LSB first ...
// send data_Tx, begin with the LSB
bit = (Tx_frame >> count) & 1;
GPIO_WriteBit(GPIOD, GPIO_Pin_5, (BitAction)bit);
count++;
if (count > 9)
{
count = 0;
TxIdle = TRUE; // a signal to the producer update new frame
}
}
} // check the data-readiness again
// at here whole frame has finished
//
else
{ // put the line to idle state
GPIO_WriteBit(GPIOD, GPIO_Pin_5, (BitAction)Bit_SET);
}
// get the timer event trigger ready again
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 252, Col 1


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c * | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
#define IDLE 0x00
#define STARTBIT 0
#define STOPBIT 1
#define TXDATAFRAME(byte) {~((byte << 1) | STOPBIT) }
/* Private macro ---*/
extern __IO uint16_t CCRval;
extern int e_ReadytoSend;
extern uint8_t Rx_byte;
/* Private variables ---*/
static int8_t data_Rx = 0;
static int TxIdle = TRUE;
static uint8_t nWait2Ticks = 0;
extern volatile uint8_t PD6_data;
static unsigned char data_Tx = 0x31; // my test byte
// STARTBIT + data_Tx + STOPBIT: bit9=1, bit[8.1]=data_Tx, bit0=0
// the 10-bit data frame then is: // 1 00110001 0 (or S + 31h + P)
static unsigned int Tx_frame = 0x0262; // 1 00110001 0
// This is the frame going into the UART that it inverses every bit
// The UART Receiver inverses its data logic level
static volatile uint16_t PC13_UserKey = 0xFFFF;
volatile uint8_t bit = 0;
/* Private function prototypes ---*/
/* Private functions ---*/
/**/
/* Cortex-M3 Processor Exceptions Handlers */
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 63, Col 51


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
/* here RCC_Configuration() sets PCLK1 = HCLK/4 = 18 MHz
because it calls: RCC_PCLK1Config(RCC_HCLK_Div4); // APB1_prescaler: 4
from the manual, since APB1_prescaler is not 1 then the clock named
TIM2CLK in Timer config block will be 2x of the PCLK1, 36 MHz.
And the
*/
/* --
TIMx Configuration: Output Compare Timing Mode:
TIMxCLK = 36 MHz, Prescaler = 1, CCRval = 937
TIMx counter clock = TIMxCLK / prescaler = 18 MHz
CC1 update rate = TIMx counter clock / CCRval = 19200 Hz
or, given required baud rate of 19200 bps:
CCRval = TIMxCLK (Hz) / 19200 Hz
-- */
/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = 65535;
TIM_TimeBaseStructure.TIM_Prescaler = 0; // TIM_PrescalerConfig() sets it
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
/* Prescaler configuration prescaler:1 overwrites the previous value */
TIM_PrescalerConfig(TIM2, 1, TIM_PSCReloadMode_Immediate); //Tim2Clk: 18 MHz
TIM_PrescalerConfig(TIM3, 1, TIM_PSCReloadMode_Immediate); //Tim3Clk: 18 MHz
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCRval;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
Build
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 97, Col 50

Project: Soft UART design and implementation
Development environment: Cortex M3 embedded system
Editor: Tom Hua

Possible improvement to my soft UART:

Robustness –

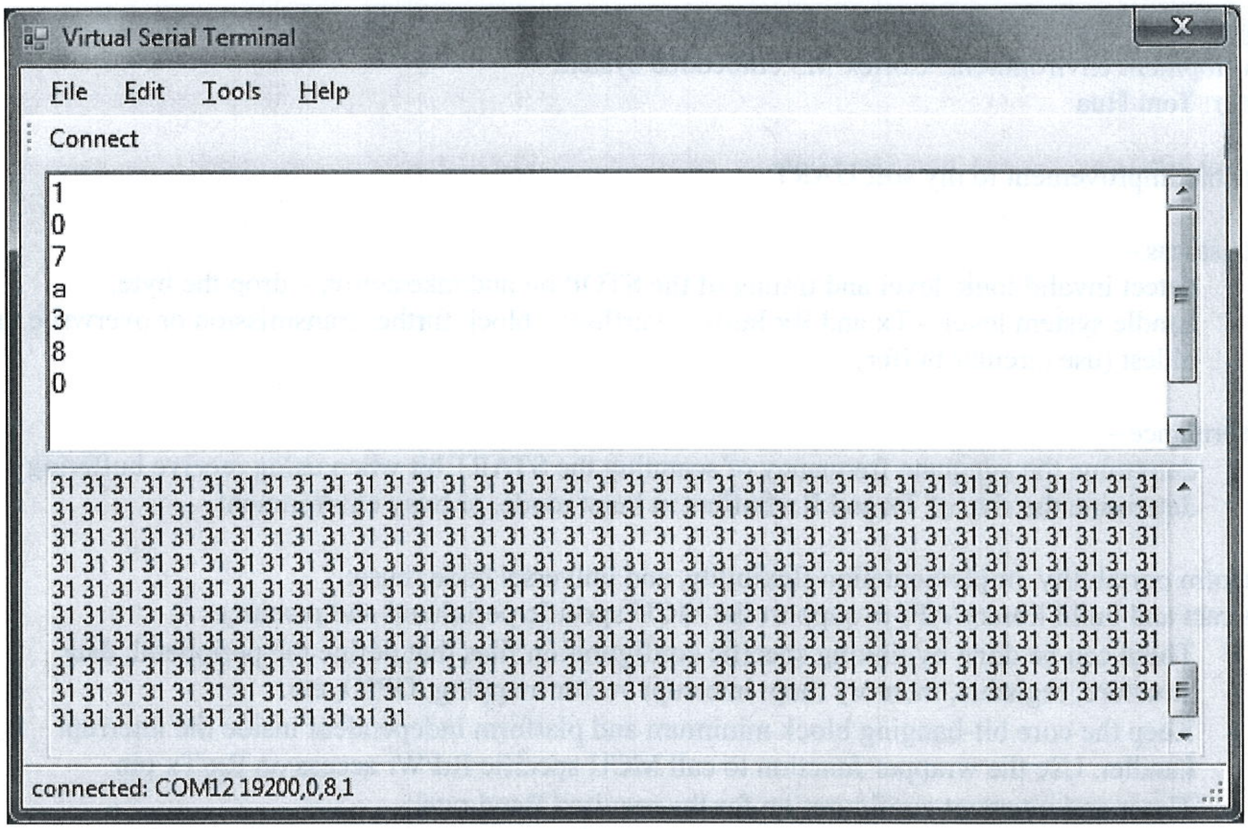
detect invalid logic level and timing of the STOP bit and take action – drop the byte.
handle system level – Tx and Rx buffer overflow – block further transmission or overwrite the oldest (use circular buffer)

Performance –

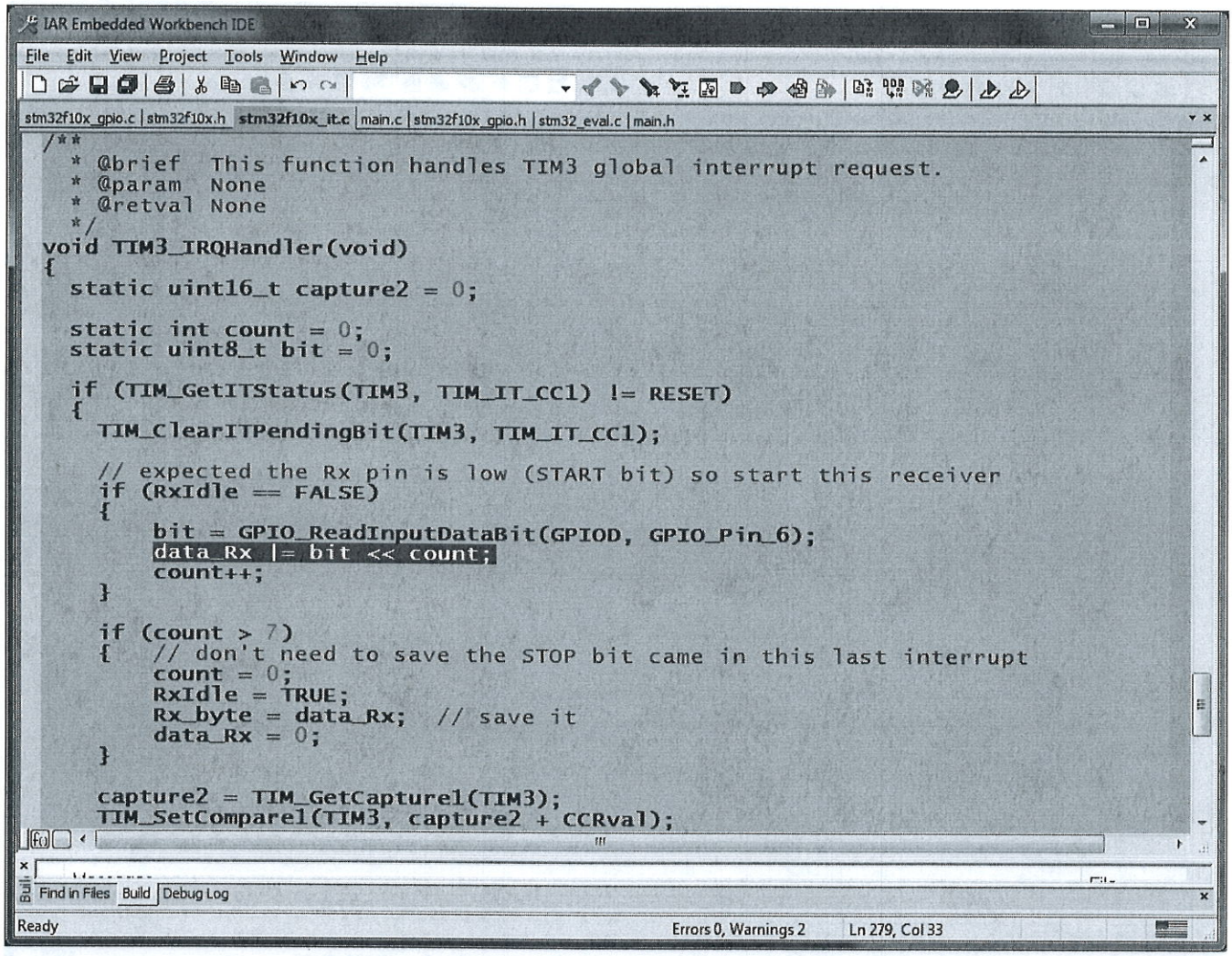
determine the adequate frequency of scanning the START bit when using receive buffering
determine the size of Tx and Rx buffers in burst mode, or busy environment

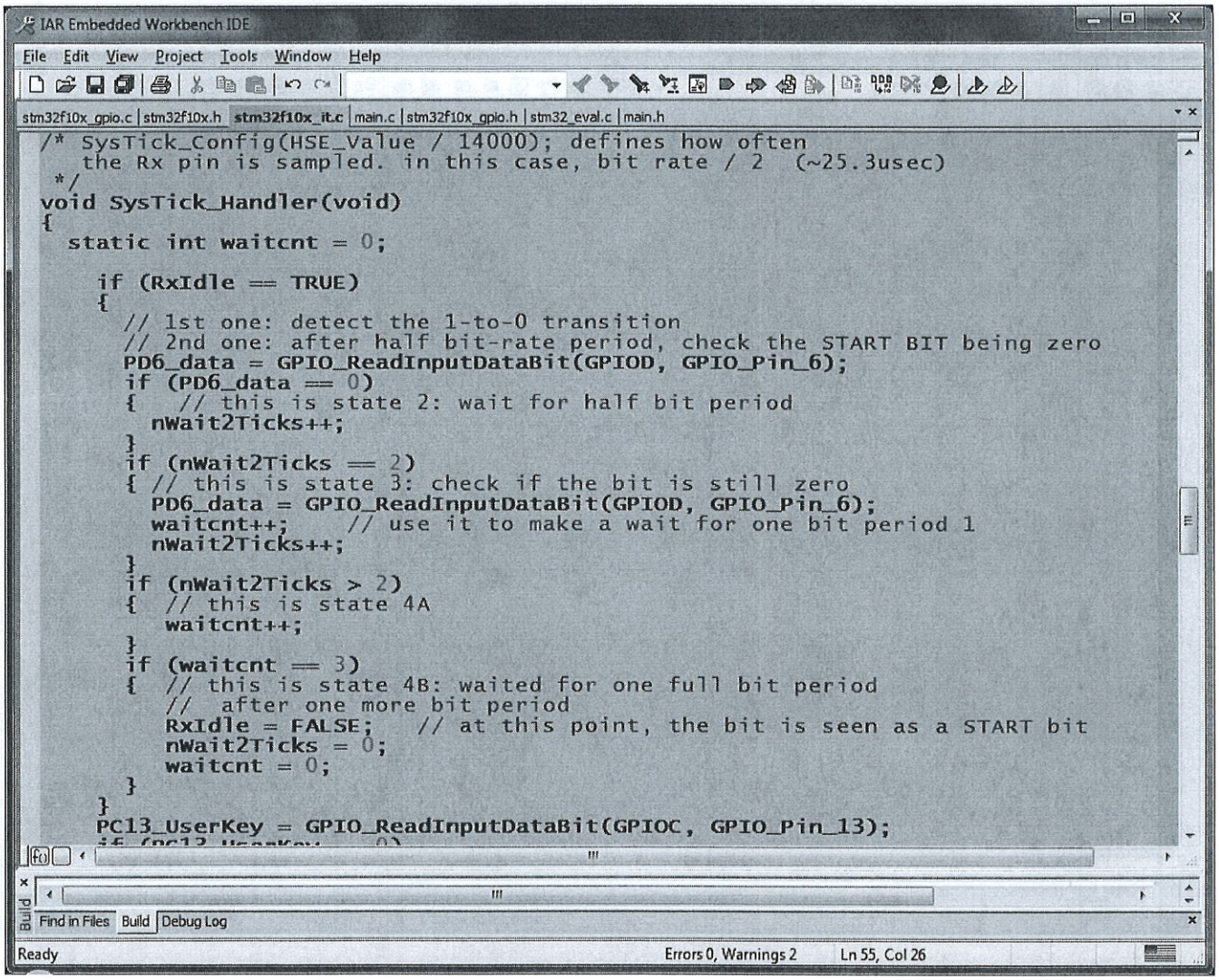
Platform portability, implementation flexibility, and universal code reuse:
Abstract and build library/API to support the MCU specific peripheral and resources –

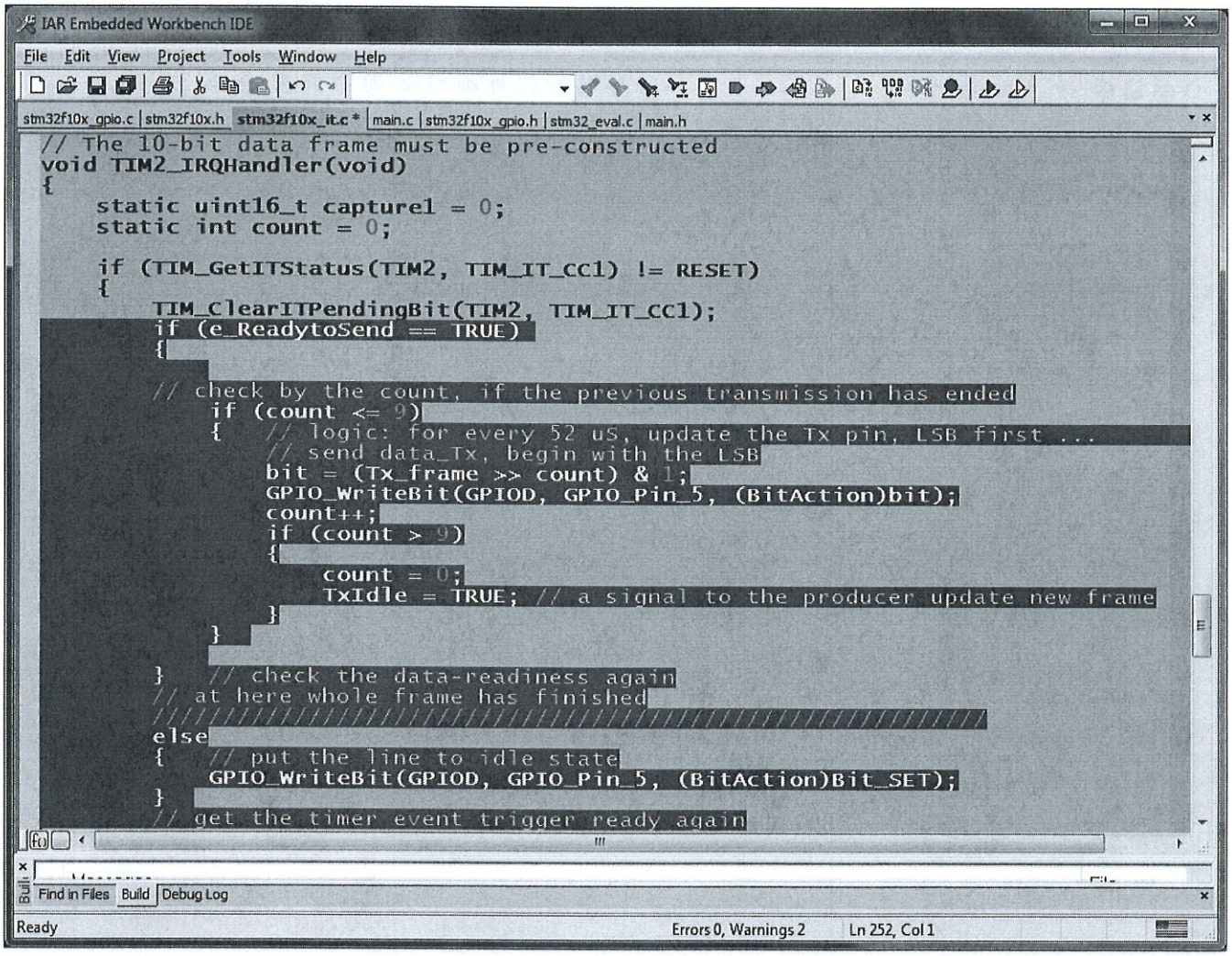
- These can be done by linking specific configuration files that define the peripheral, data structure, registers, memory map, interrupt vector mapping, GPIO, etc.
- Keep the core bit-banging block minimum and platform independent inside the interrupt handler. Use the wrapper function to call MCU specific Rd/Wr access on Rx/Tx pin
- Timer and interrupt configuration for the required Baud rate –
  For example, theses two generic functions are the wrapper of MCU specific routines:
  TimerInit(MasterClk, TimerIDs, baud_rate)
  It configures the operation mode, count parameter value (C = TIMxCLK / baud)
  pass the calculated values into the MCU specific routines for configurations
  TimerStart(TimerID) is used to setup the timer and enable the interrupt
- In the interrupt handler bit-banging logic is based on the handshaking protocol, examples:
  generic TxConfig(protocol_struct*) sets up protocol – parity, number of data bits in the frame, etc. Then the Tx/Rx interrupt uses the structure parameter to do the job
- START bit detection, Tx/Rx buffering logic are also platform independent regardless any MCU
- Configure and rd/wr the GPIO pins used for Tx/Rx:
  GPIOconfig(IO_typedef* gpio) where gpio is a structure defines the pins and mode
  This generic function should call the MCU specific GPIO functions in their files.
  Implement a platform independent Wr_TxLine(IO_typedef* gpio) wrapper function
  Implement a platform independent Rd_RxLine(IO_typedef* gpio) wrapper function
- Global variables shared between main task and Tx/Rx tasks should be kept minimum.

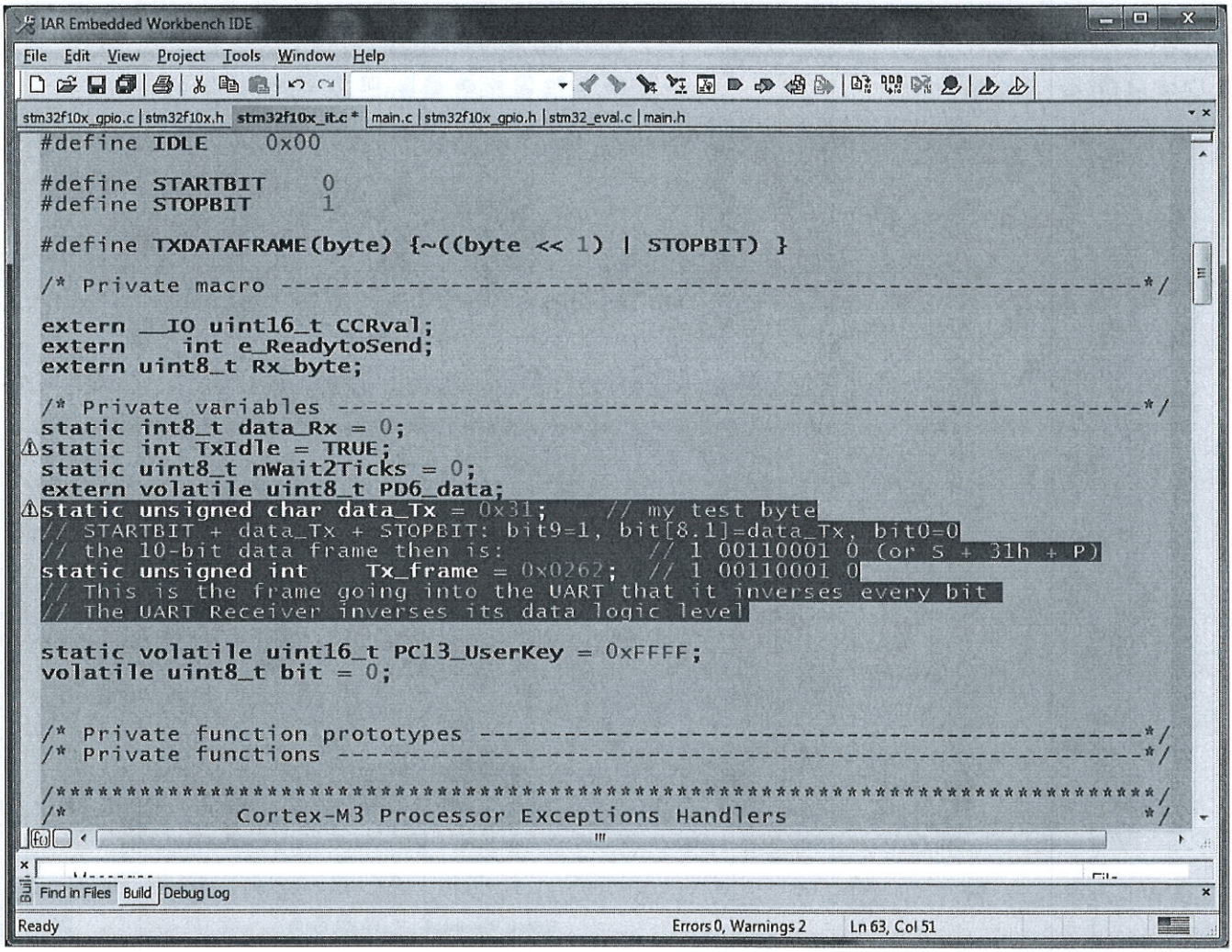


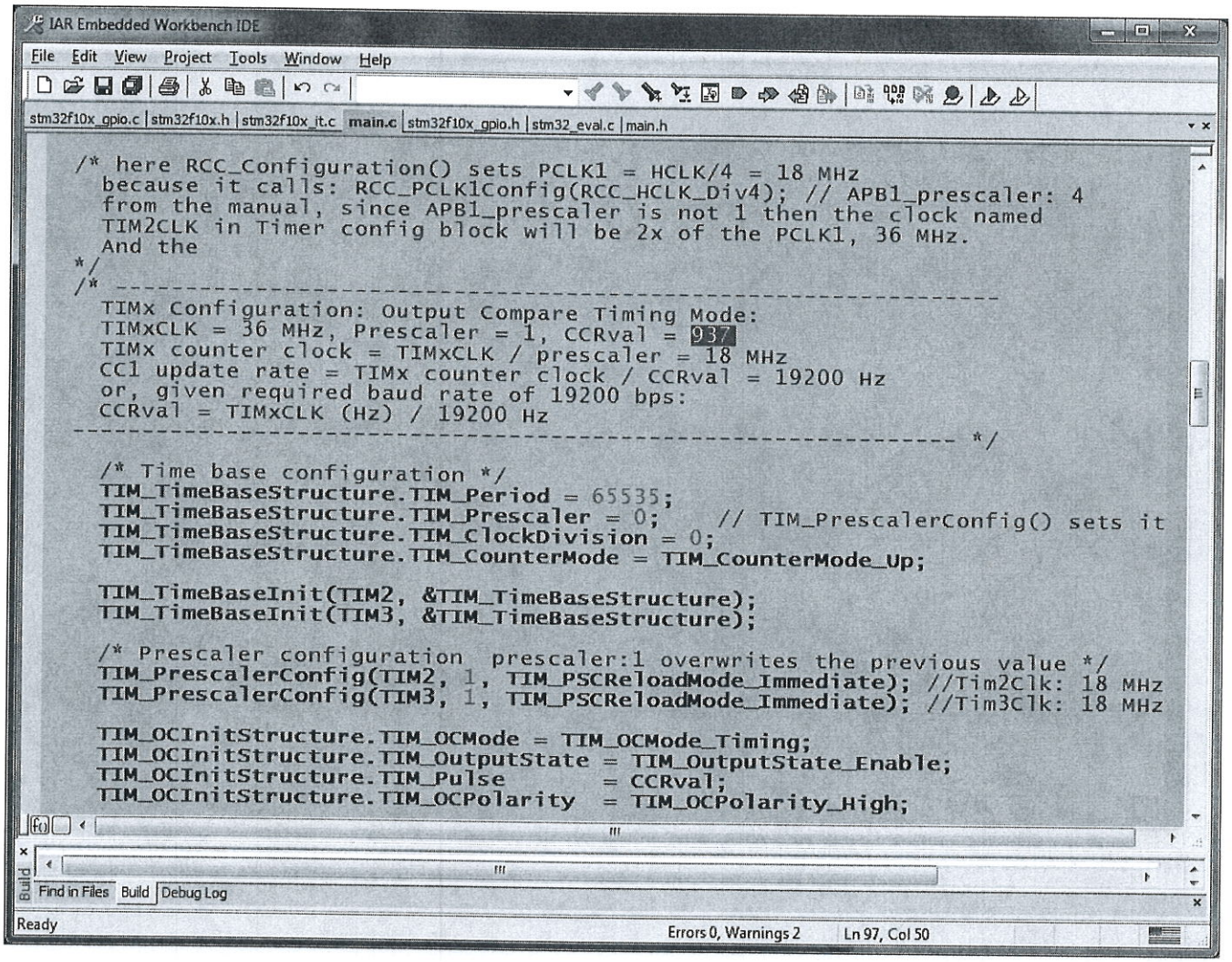
This is my version of serial data terminal I used for testing. The upper text-box field is for user to type in the data to send. The lower text-box field shows the received data in real time, and configured to show in hex value.

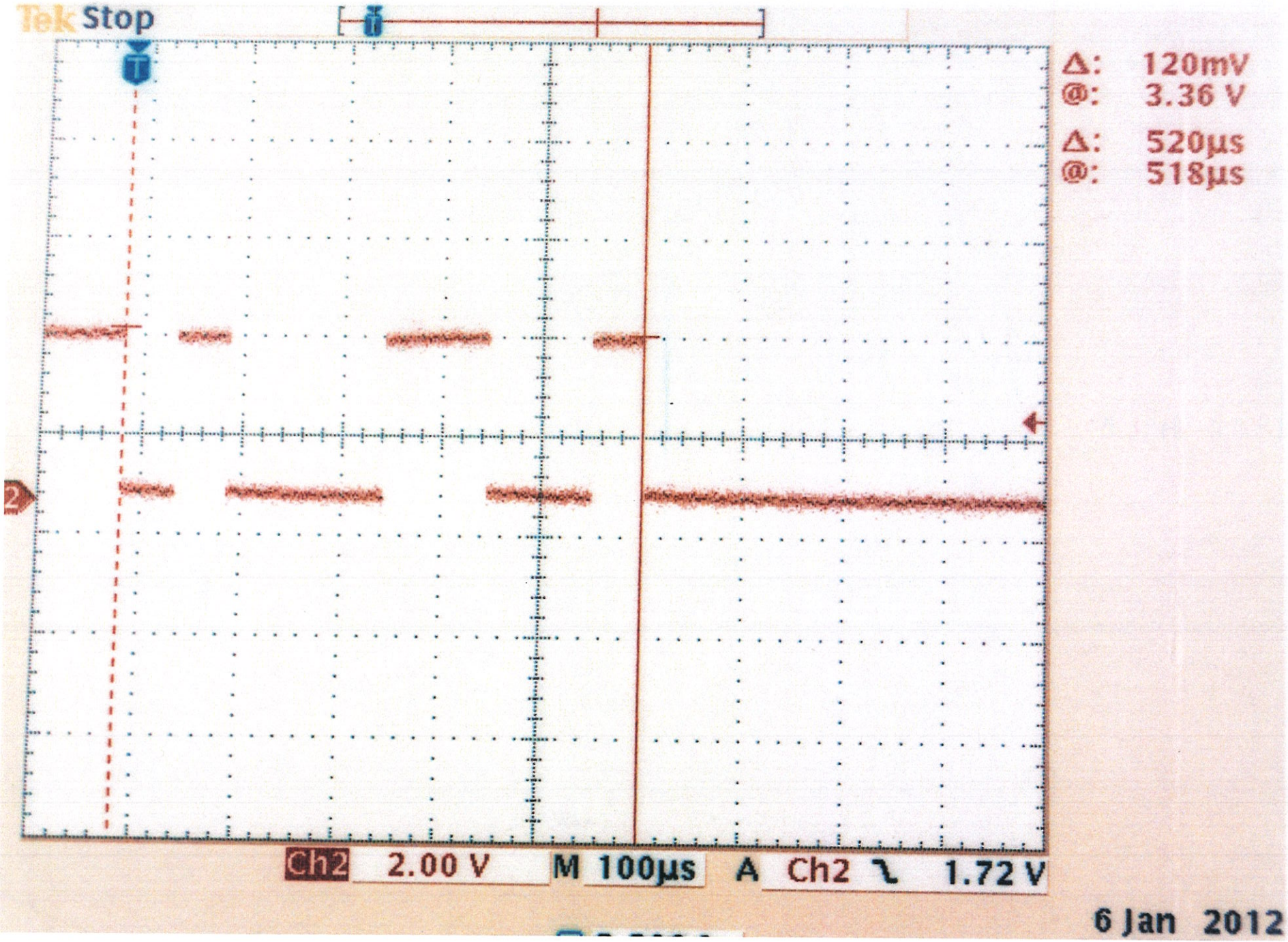

IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
/**
* @brief This function handles TIM3 global interrupt request.
* @param None
* @retval None
*/
void TIM3_IRQHandler(void)
{
static uint16_t capture2 = 0;
static int count = 0;
static uint8_t bit = 0;
if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET)
{
TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
// expected the Rx pin is low (START bit) so start this receiver
if (RxIdle == FALSE)
{
bit = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
data_Rx |= bit << count;
count++;
}
if (count > 7)
{ // don't need to save the STOP bit came in this last interrupt
count = 0;
RxIdle = TRUE;
Rx_byte = data_Rx; // save it
data_Rx = 0;
}
capture2 = TIM_GetCapture1(TIM3);
TIM_SetCompare1(TIM3, capture2 + CCRval);
Build
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 279, Col 33

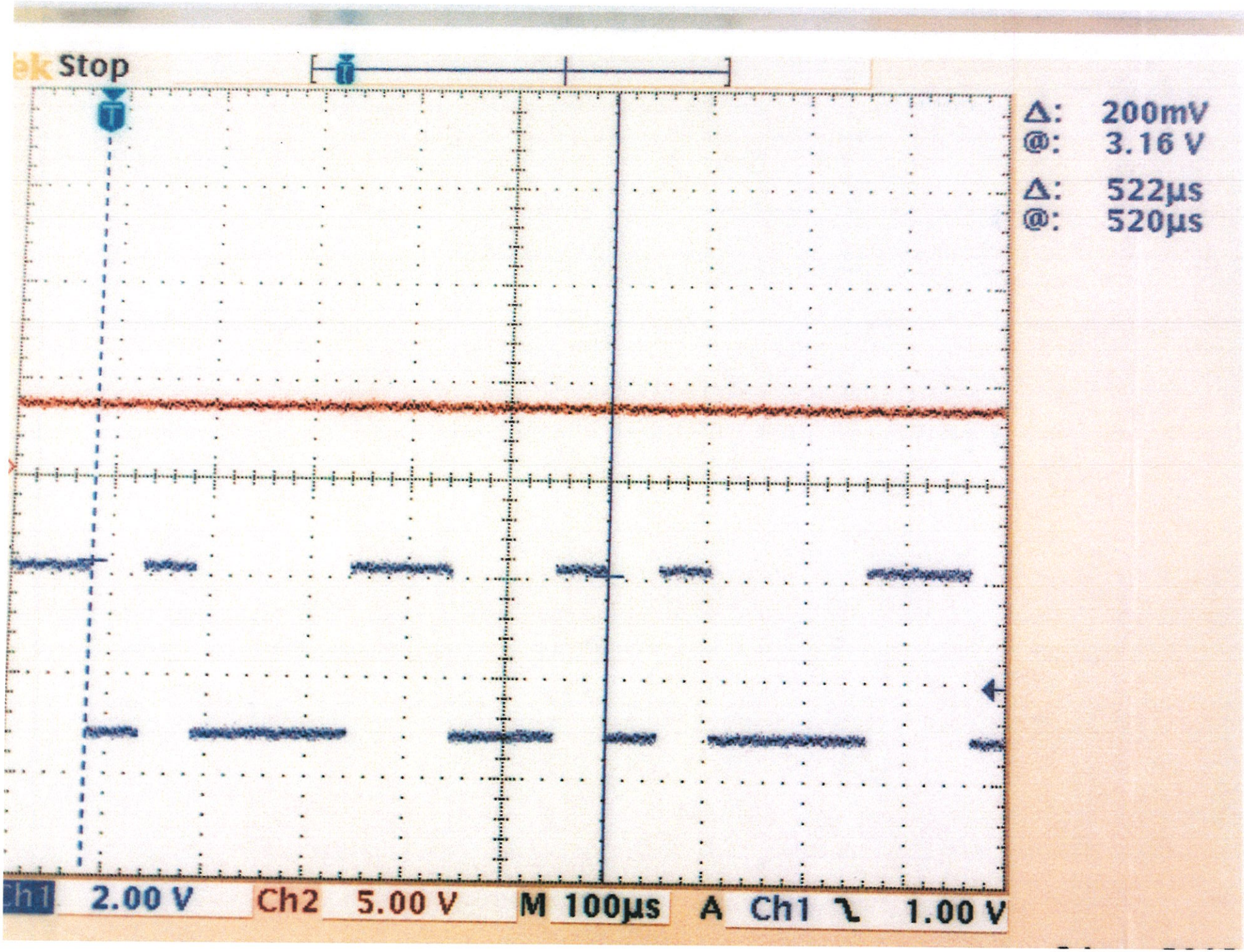

IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
/* SysTick_Config(HSE_Value / 14000); defines how often
the Rx pin is sampled. in this case, bit rate / 2 (~25.3usec)
*/
void SysTick_Handler(void)
{
static int waitcnt = 0;
if (RxIdle == TRUE)
{
// 1st one: detect the 1-to-0 transition
// 2nd one: after half bit-rate period, check the START BIT being zero
PD6_data = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
if (PD6_data == 0)
{ // this is state 2: wait for half bit period
nWait2Ticks++;
}
if (nWait2Ticks == 2)
{ // this is state 3: check if the bit is still zero
PD6_data = GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_6);
waitcnt++; // use it to make a wait for one bit period 1
nWait2Ticks++;
}
if (nWait2Ticks > 2)
{ // this is state 4A
waitcnt++;
}
if (waitcnt == 3)
{ // this is state 4B: waited for one full bit period
// after one more bit period
RxIdle = FALSE; // at this point, the bit is seen as a START bit
nWait2Ticks = 0;
waitcnt = 0;
}
}
PC13_UserKey = GPIO_ReadInputDataBit(GPIOC, GPIO_Pin_13);
Build
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 55, Col 26


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c* | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
// The 10-bit data frame must be pre-constructed
void TIM2_IRQHandler(void)
{
static uint16_t capture1 = 0;
static int count = 0;
if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
{
TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
if (e_ReadytoSend == TRUE)
{
// check by the count, if the previous transmission has ended
if (count <= 9)
{ // logic: for every 52 uS, update the Tx pin, LSB first ...
// send data_Tx, begin with the LSB
bit = (Tx_frame >> count) & 1;
GPIO_WriteBit(GPIOD, GPIO_Pin_5, (BitAction)bit);
count++;
if (count > 9)
{
count = 0;
TxIdle = TRUE; // a signal to the producer update new frame
}
}
} // check the data-readiness again
// at here whole frame has finished
//
else
{ // put the line to idle state
GPIO_WriteBit(GPIOD, GPIO_Pin_5, (BitAction)Bit_SET);
}
// get the timer event trigger ready again
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 252, Col 1


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c * | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
#define IDLE 0x00
#define STARTBIT 0
#define STOPBIT 1
#define TXDATAFRAME(byte) {~((byte << 1) | STOPBIT) }
/* Private macro ---*/
extern __IO uint16_t CCRval;
extern int e_ReadytoSend;
extern uint8_t Rx_byte;
/* Private variables ---*/
static int8_t data_Rx = 0;
static int TxIdle = TRUE;
static uint8_t nWait2Ticks = 0;
extern volatile uint8_t PD6_data;
static unsigned char data_Tx = 0x31; // my test byte
// STARTBIT + data_Tx + STOPBIT: bit9=1, bit[8.1]=data_Tx, bit0=0
// the 10-bit data frame then is: // 1 00110001 0 (or S + 31h + P)
static unsigned int Tx_frame = 0x0262; // 1 00110001 0
// This is the frame going into the UART that it inverses every bit
// The UART Receiver inverses its data logic level
static volatile uint16_t PC13_UserKey = 0xFFFF;
volatile uint8_t bit = 0;
/* Private function prototypes ---*/
/* Private functions ---*/
/***/
/* Cortex-M3 Processor Exceptions Handlers */
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 63, Col 51


IAR Embedded Workbench IDE
File Edit View Project Tools Window Help
stm32f10x_gpio.c | stm32f10x.h | stm32f10x_it.c | main.c | stm32f10x_gpio.h | stm32_eval.c | main.h
/* here RCC_Configuration() sets PCLK1 = HCLK/4 = 18 MHz
because it calls: RCC_PCLK1Config(RCC_HCLK_Div4); // APB1_prescaler: 4
from the manual, since APB1_prescaler is not 1 then the clock named
TIM2CLK in Timer config block will be 2x of the PCLK1, 36 MHz.
And the
*/
/* --
TIMx Configuration: Output Compare Timing Mode:
TIMxCLK = 36 MHz, Prescaler = 1, CCRval = 937
TIMx counter clock = TIMxCLK / prescaler = 18 MHz
CC1 update rate = TIMx counter clock / CCRval = 19200 Hz
or, given required baud rate of 19200 bps:
CCRval = TIMxCLK (Hz) / 19200 Hz
-- */
/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = 65535;
TIM_TimeBaseStructure.TIM_Prescaler = 0; // TIM_PrescalerConfig() sets it
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
/* Prescaler configuration prescaler:1 overwrites the previous value */
TIM_PrescalerConfig(TIM2, 1, TIM_PSCReloadMode_Immediate); //Tim2Clk: 18 MHz
TIM_PrescalerConfig(TIM3, 1, TIM_PSCReloadMode_Immediate); //Tim3Clk: 18 MHz
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = CCRval;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
Build
Find in Files Build Debug Log
Ready
Errors 0, Warnings 2
Ln 97, Col 50


Tek Stop
Δ: 120mV
@: 3.36 V
Δ: 520µs
@: 518µs
Ch2 2.00 V
M 100µs
A Ch2 ↘ 1.72 V
6 Jan 2012


Stop
Δ: 200mV
@: 3.16 V
Δ: 522µs
@: 520µs
Ch1 2.00 V
Ch2 5.00 V
M 100µs
A Ch1 1.00 V