AN3174
Application note

Implementing receivers for infrared remote control protocols using STM32F10xxx microcontrollers

Introduction

Infrared radiation is the region of the electromagnetic spectrum which lies between microwaves and visible light.

Infrared radiation has two ranges. Near infrared light is closest in wavelength to visible light and far infrared is closer to the microwave region of the electromagnetic spectrum.

The shorter waves are the ones used by remote controls. Information is transmitted and received using electromagnetic energy, without using wires.

Infrared technology offers important advantages as a form of wireless communication. Nowadays, almost all audio and video equipment can be controlled using an infrared remote control. At the receiving end, a receiver detects the light pulses, which are processed to retrieve/decode the information they contain.

There are many popular infrared protocols used to transmit data via infrared light, such as RC5, SIRC, ...

The purpose of this application note is to provide a generic solution for implementing an IR receiver in software using a STM32F10xxx microcontroller. Example of software implementation is provided for RC5 and SIRC protocols, other protocols are supported and available upon request (for further information contact your local STMicroelectronics sales office).

Contents

1 Hardware considerations

To improve noise rejection, the IR pulses are modulated at around 36 kHz, 38 kHz or 40 kHz. The easiest way to receive these pulses is to use an integrated IR-receiver/demodulator module like the TSOP1736 (5 V supply version) or TSOP34836 (3.3 V supply version) or other equivalent part number (refer to *Figure 1*).

These are 3-pin devices that receive the infrared burst and output the demodulated bit stream on the output pin which is connected directly to one of the STM32 microcontroller's GPIO pins or GP-Timers Input Capture channels. If TSOP1736 is used, the selected GPIO must be Five volt Tolerant (FT). The output of the IR module is inverted compared to the transmitted data (the data is idle high and logic '0' becomes logic '1' and vice versa).

Note: *The IR module needs two external components: a capacitor and a resistor (refer to the related IR module datasheet for their values).*

Figure 1.   Hardware configuration
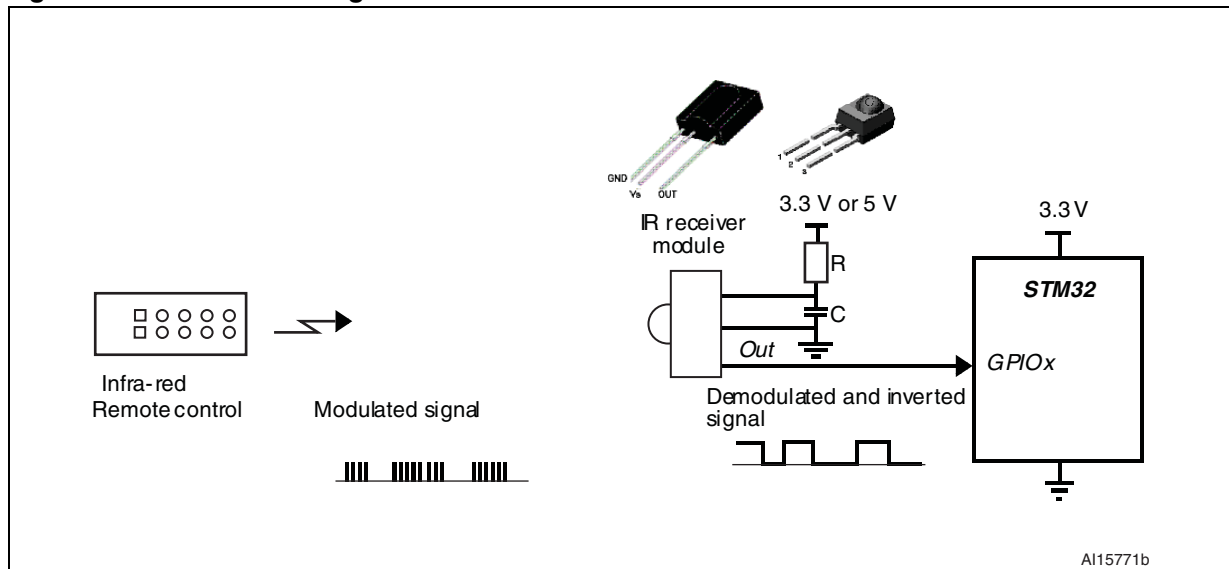
2 Generic solution

Each Infrared protocol can be decoded using one of the timer peripherals embedded in the STM32 microcontroller. This timer can be configured in PWM input mode and used to sample the Infrared frame bits. The timer input capture function is active on edges with opposite polarity.

The timer generates three types of interrupt:

● Interrupt at each falling edge: This can be used to measure the whole pulse (duration between two successive falling edges).

● Interrupt at each rising edge: This can be used to measure the low pulse (duration between falling and rising edges).

● Update event: This is used to put the Infrared packet into default state (Bit count, Data and Status) when the Timer counter overflows.

The low pulse and the whole pulse duration are used to determine the bit value. If the durations are within the tolerance range of the bit time, we identify the bit value (Logic0, Logic1 or Header).

The flowchart below gives an overview of the Infrared decoding procedure.

**Figure 2. Infrared decoding flowchart**
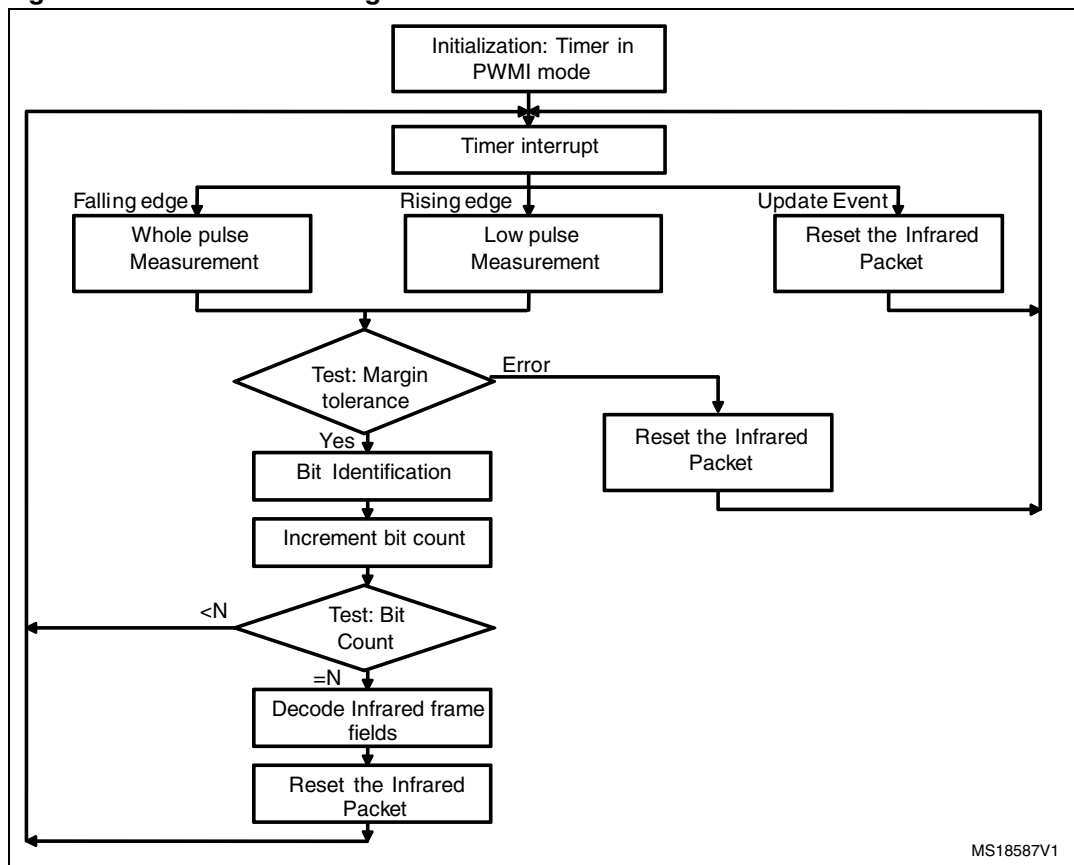
3   RC5 protocol solutions

3.1   Protocol basics

The RC5 code is a 14-bit word, it uses bi-phase modulation (also called Manchester coding) of a 36 kHz IR carrier frequency. All bits have an equal length of 1.778 ms, with half of the bit time filled with a burst of the 36 kHz carrier and the other half being idle. A logical zero is represented by a burst in the first half of the bit time. A logical one is represented by a burst in the second half of the bit time. The duty cycle of the 36 kHz carrier frequency is 33% or 25% which reduces power consumption.

Figure 3.   RC5 bit representation
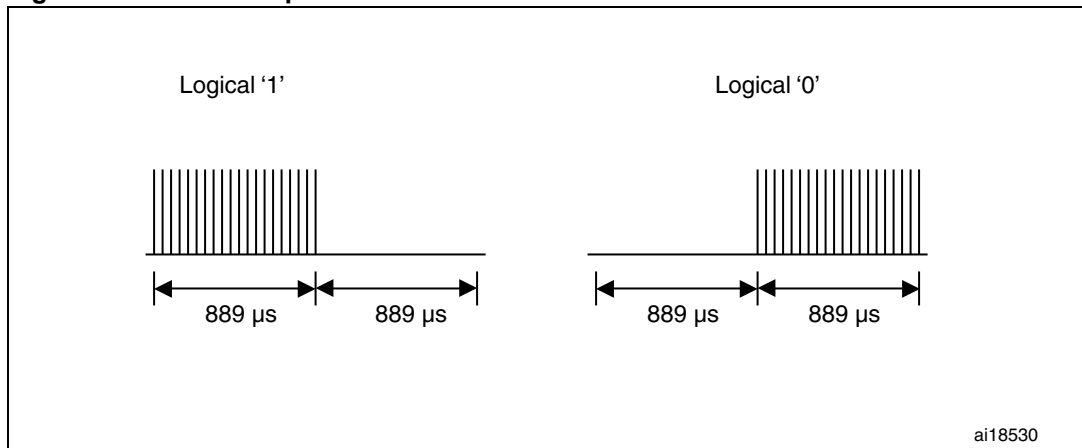
The RC5 frame can generate 2048 (32x64) different commands organized on 32 groups. Each group has 64 different commands. A RC5 frame contains the following fields. An example of RC5 frame is shown in *Figure 4*.

● **Start bit (S):** 1 bit length, always logic 1.

● **Field bit (F):** 1 bit length, which denotes whether the command sent is in the lower field (logic 1 = 0 to 63 decimal) or the upper field (logic 0 = 64 to 127 decimal). The field bit was added later when it was realized that 64 commands per device were insufficient. Previously, the field bit was combined with the start bit. Many devices still use this original system.

● **Control bit or Toggle bit (C):** 1 bit length, which toggles each time a button is pressed. This allows the receiving device to distinguish between two successive button presses (such as "1", "1" for "11").

● **Address:** 5 bits length, that selects one of 32 possible systems.

● **Command:** 6 bits length, that (in conjunction with the field bit) represents one of the 128 possible RC5 commands.

Figure 4.   Example of an RC5 frame
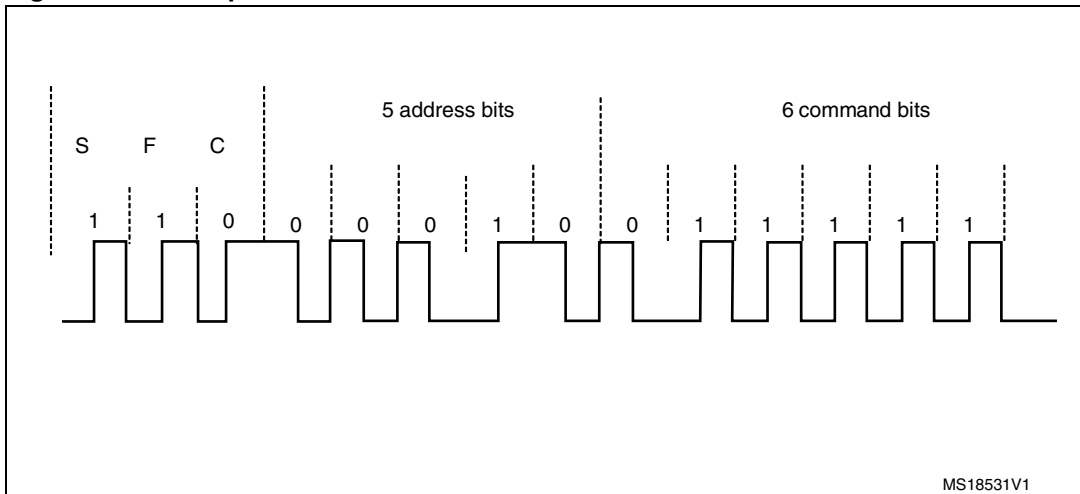
To avoid frame collisions, an idle time is inserted between two successive frames with a specific width (see *Figure 5*).

The idle time is defined as 50 bits wide. So, the periodicity of a frame is 64 x 1 bit width:

64 x 1.778 = 113.792 ms.

Figure 5.   RC5 idle time
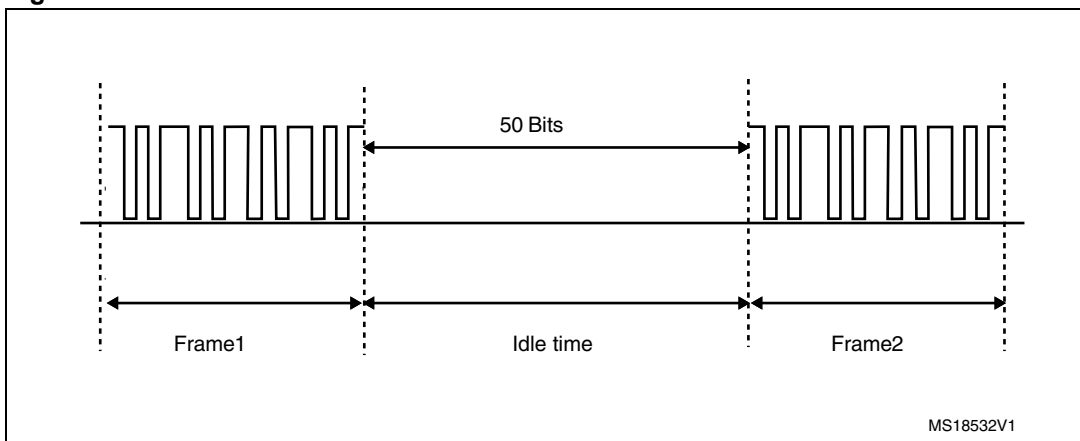
Table 1.   RC5 timings

| Description	Min.	Typical	Max.
RC5 Half bit period	640 µs	889 µs	1140 µs
RC5 Full bit period	1340 µs	1778 µs	2220 µs
RC5 message time	23.644 ms	24.889 µs	26.133 ms
RC5 message repetition time	108.089 ms	113.778 ms	119.467 ms
Carrier pulse bit time	27.233 µs	27.778 µs	28.349 µs

3.2 Software implementation using a single GPIO with a generic timer

3.2.1 RC5 frame reading mechanism

Figure 6 shows how the RC5 frame is received. Principally, two of the STM32 microcontroller's embedded peripherals are used for this purpose: EXTI and a timer (TIMx).

The STM32 pin connected to the IR module's output pin can be any GPIO selected by the user.

Figure 6. RC5 frame reception mechanism
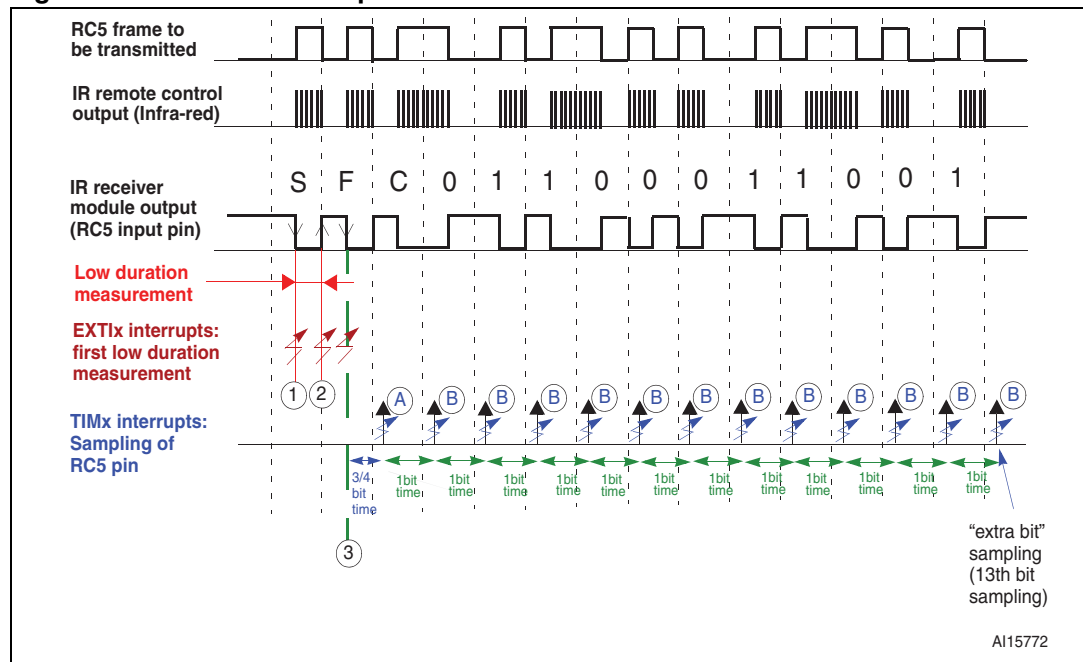
EXTI interrupt events

The EXTI interrupt is used to start and stop TIMx in order to measure the first low duration to validate the header timing of the RC5 frame:

● First EXTI interrupt event (1): the TIMx counter is initialized and enabled.

● Second EXTI interrupt event (2): the TIMx counter is disabled, read and then initialized.

The value read from the counter gives the measured duration. The 3rd execution of the EXTI interrupt depends on the measured duration:

– If the duration is within the tolerance range of one half bit time, the EXTI is not disabled and the EXTI interrupt occurs for the 3rd time, which enables TIMx. TIMx then starts sampling the RC5 data. In this case the Field bit is recognized as a logical 1.

–   If the duration is within the tolerance range of one bit time, the EXTI is then disabled at this moment and the TIMx update event interrupt as well as the TIMx counter are enabled to start sampling the RC5 data. In this case the Field bit is recognized as a logical 0. If the duration seems to be a glitch, the system is initialized for the next RC5 frame.

●   Third EXTI interrupt event (3): this interrupt occurrence depends on the duration of the first low duration, see (2). When the interrupt occurs, TIMx is enabled and starts sampling the RC5 data.

TIMx interrupt events

TIMx is used to sample each bit of the RC5 frame after checking the timing of the first low duration of the frame.

TIMx interrupts are executed 13 times during an RC5 frame in order to sample all the bits.

The Start bit (S) and Field bit (F) are not sampled by TIMx and an "extra bit" is sampled at the end of the RC5 frame to be sure that all bits have been received and an idle state is present.

●   TIMx interrupt event (A): at this time, the RC5 pin is sampled by a single reading of the GPIO input data register. In this interrupt service routine, the TIMx is configured to generate a periodic interrupt each bit time.

●   TIMx interrupts event (B): at this time, the RC5 pin is sampled by a single reading of the GPIO input data register and the interrupt service routine checks if the number of data bits has reached 13 (n = 13: 14-2+1) If yes, the TIMx counter and the TIMx update interrupt are disabled.

As we can see, reading from the GPIO input data register directly reflects the value of the bit. If the read value is at low level this implies that the bit value is logic '0'. If the value read is at high level this implies that the bit value is logic'1'.

How to use the RC5 library

The RC5 driver is very simple to use. There are four functions available for the user.

RC5_Receiver_Init()

This function is intended to initialize the different peripherals used by the RC5 driver: GPIOs, EXTI and TIMx. It must be called after the user clock configuration.

RC5_Sample_Data()

This function is used to sample the RC5 data. It must be called in the RC5_TIM_IRQ_Handler routine (TIMx_IRQHandler) in the stm32f10x_it.c file. The RC5_IR_Receiver.h file must be included in the stm32f10x_it.c file. By default, TIM2 is used. You can use any timer by modifying the defines in the RC5_IR_Emul_Receiver.h file (path: \Project\InfraRed\RC5_Decoding_TIM_EXTI\inc) as follows:

Example:

If you want to use TIM3, make these modifications (highlighted in **bold**):

```
#define RC5_TIM TIM3
#define RC5_TIM_CLK RCC_APB1Periph_TIM3
#define RC5_TIM_IRQn TIM3_IRQn
#define RC5_TIM_IRQ_Handler TIM3_IRQHandler
```

You can choose any of the STM32F10x family timers.

RC5_MeasureFirstLowDuration()

This function measures and validates the first low duration of the RC5 frame. When this timing is in the range of the allowed timings, the function enables the RC5 frame sampling. This function must be called in the appropriate EXTI interrupt handler (in stm32f10x_it.c file) depending on the GPIO used for the RC5 input pin.

By default, GPIOB.01 is used as the RC5 input pin. You can use any timer by modifying the defines in the RC5_IR_Emul_Receiver.h file (path:\Project\InfraRed\RC5_Decoding_TIM_EXTI\inc).

Example:

If you want to use GPIOD.09, make these modifications (highlighted in **bold**):

`#define RC5_GPIO_PORT GPIO`**D**

`#define RC5_GPIO_CLK RCC_APB2Periph_GPIO`**D**

`#define RC5_GPIO_PIN GPIO_Pin_`**9**

`#define RC5_EXTI_PORT_SOURCE GPIO_PortSourceGPIO`**D**

`#define RC5_EXTI_PIN_SOURCE GPIO_PinSource`**9**

`#define RC5_EXTI_IRQn EXTI`**9_5**`_IRQn`

`#define RC5_EXTI_LINE EXTI_Line`**9**

`#define RC5_EXTI_IRQ_Handler EXTI`**9_5**`_IRQHandler`

RC5_MeasureFirstLowDuration must be called in the RC5_EXTI_IRQHandler.

RC5_Decode()

This function is intended to be called in the user application. It decodes the RC5 received messages. It returns a structure that contains the different values of the RC5 frame.

```
typedef struct
{
__IO uint8_t ToggleBit; /* Toggle bit field */
__IO uint8_t Address; /* Address field */
__IO uint8_t Command; /* Command field */
} RC5Frame_TypeDef ;
```

RC5_decode() must be called when the IR_FrameReceived flag is equal to YES.

Example:

```
/* System Clocks Configuration */
RCC_Configuration();
/* Initialize RC5 reception */
RC5_Receiver_Init();
while (1)
{
/* If RC5 frame has been received, then decode it */
if (IR_FrameReceived == YES)
{
/* Get the RC5 frame */
RC5_Frame = RC5_Decode();
}
}
```

3.3 Software implementation using a GP-Timer configured in PWM input mode

3.3.1 RC5 Frame decoding mechanism

Figure 7 shows how the RC5 frame is received. One of the STM32 microcontroller's embedded peripherals is used for this purpose: TIMER configured in mode PWM input.

This input can capture the current timer value both at falling and rising edges as well as generate an interrupt on both edges. This feature makes it easy to measure the RC5 pulse high and low times.

Figure 7.   RC5 frame decoding mechanism
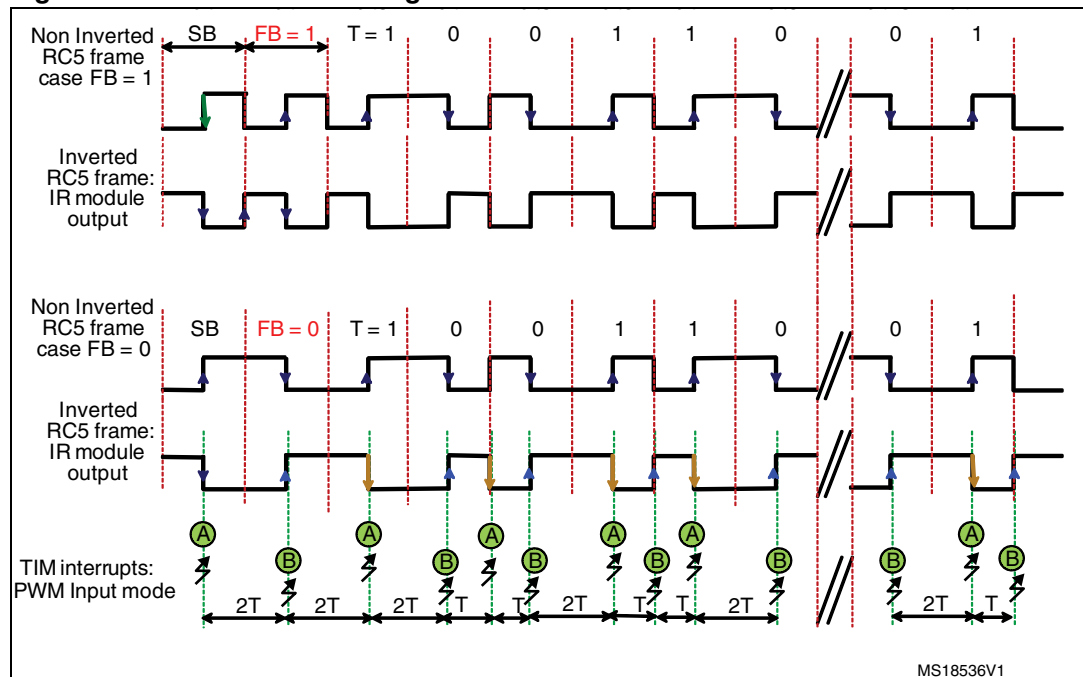
- **TIMER interrupt event:** Falling edge

  A: The TIMER interrupt is used to measure the period between two successive falling edges (The whole pulse duration).

- **TIMER interrupt event:** Rising edge

  B: TIMER is used to measure the duration between falling and rising edges (The low pulse duration).

The two durations are used to determine the bit value. Each bit value is determined in relation to the last bit.

Figure 8. **Bit determination by the rising edge: low pulse**
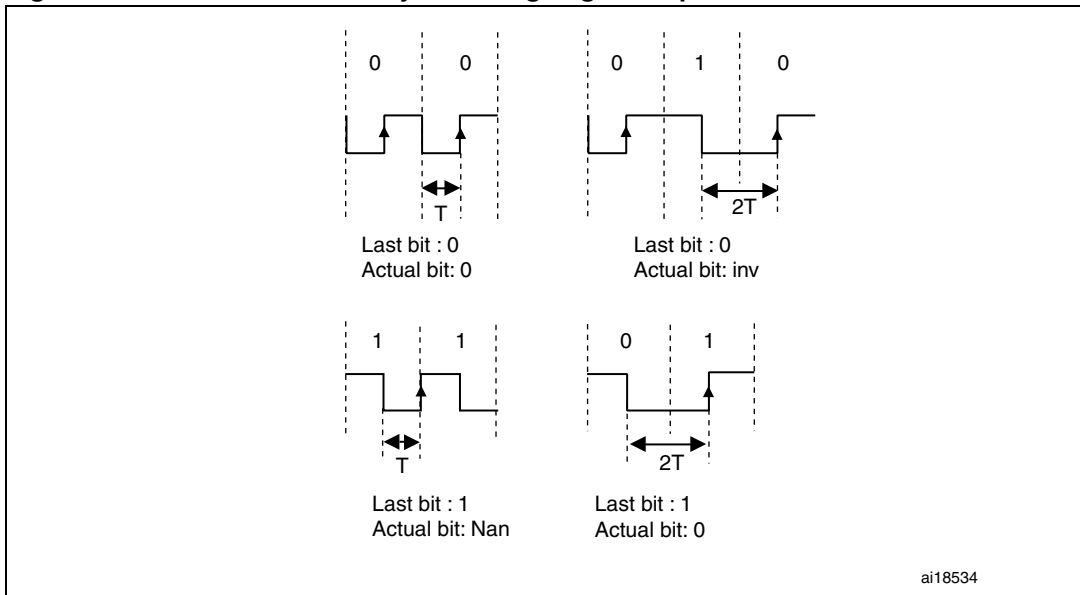
ai18534

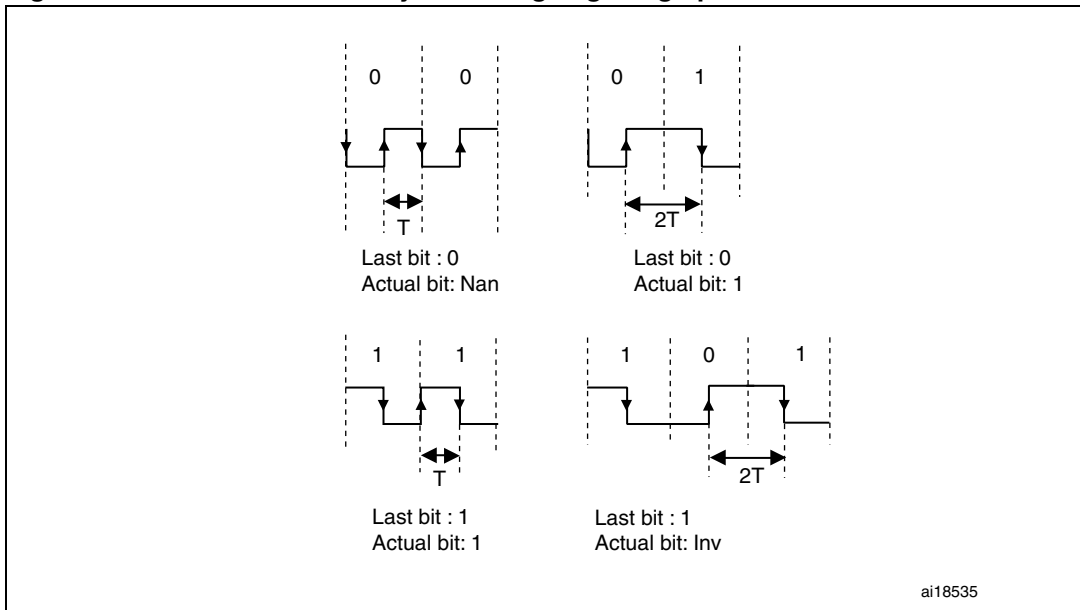- – If the low pulse duration is equal to T, and the last bit determined is '0' => the actual bit is *'logic0'*.
- – If the low pulse duration is equal to 2T, and the last bit determined is '0' => the actual bit is *'Inv'*. (**Invalid case**: this case cannot be released)
- – If the low pulse duration is equal to T, and the last bit determined is '1' => the actual bit is *'Nan'*. (**No bit:** this bit is determined at the next falling edge)
- – If the low pulse duration is equal to 2T, and the last bit determined is '1' => the actual bit is *'logic0'*.

Figure 9.   Bit determination by the falling edge: high pulse

ai18535

- If the high pulse duration is equal to T, and the last bit determined is '0' => the actual bit is *'Nan'*. (**No Bit:** this bit is determined at the next rising edge)
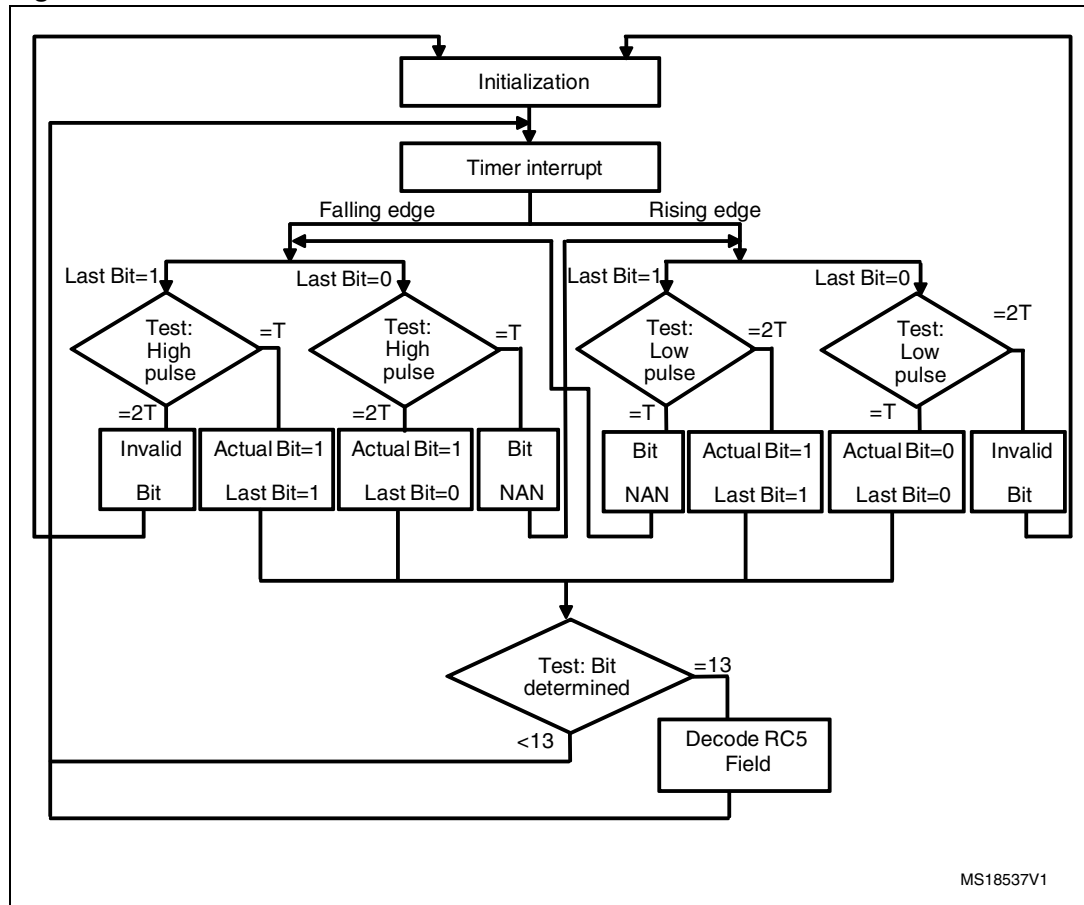- If the high pulse duration is equal to 2T, and the last bit determined is '0' => the actual bit is *'logic1'*.
- If the high pulse duration is equal to T, and the last bit determined is '1' => the actual bit is *'logic1'*.
- If the high pulse duration is equal to 2T, and the last bit determined is '1' => the actual bit is *'Inv'*. (**Invalid case**: this case cannot be released)

Figure 10. RC5 solution flowchart

3.3.2 RC5 library

The RC5 driver is very simple to use.

IR_RC5_Init()

This function initializes the different peripherals (GPIO, TIMER, NVIC,...).

IR_RC5_ResetPacket()

This function sets the packet structure to default state. Mainly, this function is called in the TIM3_IRQHandler routine. It occurs each TIMER overflow to reset the RC5 packet.

IR_RC5_Decode(IRFrame_TypeDef *ir_frame)

This function is intended to be called in the user application. It decodes the RC5 received messages. The following structure contains the different values of the RC5 frame.

```
typedef struct
{
__IO uint8_t FieldBit;  /* Field bit field  */
__IO uint8_t ToggleBit; /* Toggle bit field */
__IO uint8_t Address;   /* Address field    */
__IO uint8_t Command;   /* Command field    */
} IRFrame_TypeDef ;
```

IR_RC5_decode () is executed when the RC5FrameReceived flag is equal to YES.

IR_RC5_DeInit()

This function de-initializes the different peripherals (GPIO, TIMER, NVIC,...).

TIM3_IRQHandler ()

This function handles the TIM Capture Compare interrupt.

– **Timer Falling Edge Event:** This is used to measure the period between two successive falling edges (the whole pulse duration).

– **Timer Rising Edge Event:** This is used to measure the duration between falling and rising edges (the low pulse duration).

– **Update event (time-out event)** This resets the RC5 packet. The Timer Overflow is set to 3.6 ms .

The low pulse duration and the whole pulse duration are used to determine the bit value. Each bit value is determined in relation to the last bit.

3.3.3 How to use the RC5 decoder driver

To use the RC5 decoder driver, please proceed as follows:

– Call the function IR_RC5_Init() to configure the Timer and GPIO hardware resources needed for RC5 decoding.

– TIM3 Capture Compare and Update interrupts are used to decode the RC5 frame, if a frame is received correctly a global variable "RC5FrameReceived" will be set to inform the application.

– Then the application should call the function IR_RC5_Decode() to retrieve the received RC5 frame.

Code Example

```
#include "ir_decode.h"
#include <stdio.h>
#include "stm32100e_eval_lcd.h"

IR_Frame_TypeDef IR_FRAME;
/* Initialize the Color LCD mounted on STM32100E-EVAL */
STM32100E_LCD_Init();
```

```
/* Clear the LCD */
LCD_Clear(LCD_COLOR_WHITE);
IR_RC5_Init();
/* RC5 */
IR_RC5_Decode(&IR_FRAME);
```

Note: 1 *TIM3_IRQHandler ISRs are coded within the rc5_decode driver.*

● If you are using one or both interrupts in your application you have to proceed carefully:
– Either add your application code in these ISRs
– Or copy the contents of these ISRs in your application code

2 *You can easily tailor this application to your own hardware using different define declarations inside the "rc5_decode.h" file. Please refer to the following Table.*

Table 2. Example of implementation

Define name	Description	Allowed values	Example
#define IR_TIM	Timer used for IR decoding	TIM1, TIM2, TIM3, TIM4, TIM5, TIM8, TIM9, TIM12, TIM15	TIM3
#define TIM_PRESCALER	TIM prescaler	This parameter is computed in manner to have 1us as time base. TIM Frequency(MHz) / (Prescaler + 1)	23
#define IR_TIM_CLK	APB Clock of the used timer	Refer to RCC FW driver	RCC_APB1Periph_TIM3
#define IR_TIM_IRQn	IR TIM IRQ	Refer to stm32f10x_startup.s	TIM3_IRQn
#define IR_TIM_Channel	IR TIM Channel	TIM_Channel_1 or TIM_Channel_2	TIM_Channel_1
#define IR_GPIO_PORT	Port which IR output is connected	Refer to your product pinout[1]	GPIOC
#define IR_GPIO_PORT_CLK	IR pin GPIO Clock Port	Refer to RCC FW driver	RCC_APB2Periph_GPIOC
#define IR_GPIO_PIN	Pin which IR is connected	Refer to your product pinout[1]	GPIO_Pin_6

1. For more details on the available STM32 resources, please refer to your product datasheet.

3.4 Comparison of RC5 solutions

This application note provides two solutions for implementing a RC5 receiver in software one using an EXTI and a general purpose Timer (TIMx) and one using a Timer configured in mode PWM Input. The driver is very simple to use and it supports standard and extended RC5 formats, which is not the case for several of the RC5 drivers offered by other manufacturers.

**Table 3.   Comparison table**

Solution	Advantage	Drawback
First solution: Single GPIO with generic Timer	Any Timer and GPIO can be used	Two STM32 peripherals used (EXTI, TIMx)
Second solution: PWM Input	One STM32 peripheral used (TIMx)	TIMx must have two channels and only specific GPIOs can be used

4   SIRC infrared control solution

4.1   Protocol basics

The SIRC code is a 12-bit word; it uses modulation of a 40 kHz IR carrier frequency. The SIRC protocol uses pulse distance encoding of the bits. Each pulse is a 600 µs long 40 kHz carrier burst. A logical "1" takes 1.8 ms to transmit, while a logical "0 takes 1.2 ms to transmit. (refer to *Figure 11*).
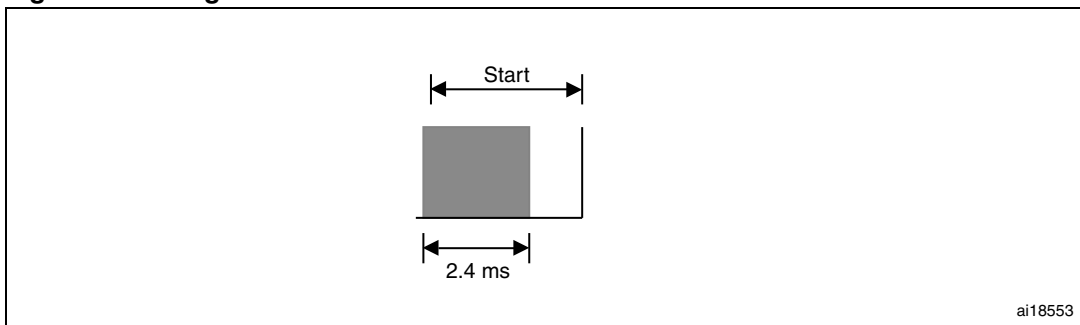
Figure 11.   Length of logical bits
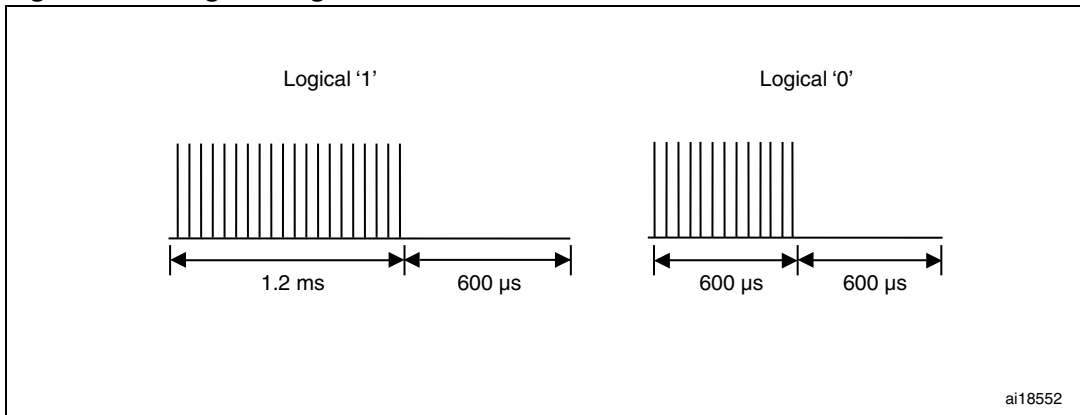
A SIRC frame contains the following fields:

●   **Start Bit:** The start burst is always 2.4 ms wide, followed by a standard space of 0.6ms.
●   **Command 7 bits length:** This field holds 7 bits which are used as command field.
●   **Address 5 bits length:** This field holds 5 bits which are used as address field.

Figure 12.   Length of start bit

With this protocol the LSB is transmitted first. So, it is assembled LSB to MSB. Since it is sent as a 7 bit for command, followed by 5 bit for device address, the code must split the 12 received bits into two groups of 7 and 5 bits.

The following figure shows an example of a SIRC frame.

In this case: address 26h (0100110b) and command Ah (01010b)

Figure 13.   Example of SIRC frame
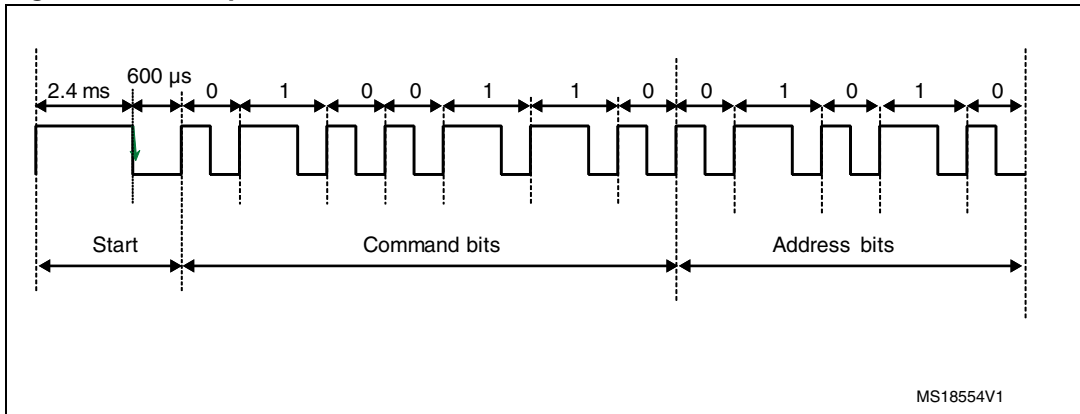
An idle time is inserted between two successive frames in order to avoid collisions. Every 45 ms a repeat code is transmitted.

Table 4.   SIRC timings

Description	Typical	Min.	Max.
Syn pulse high level	2.4 ms	2.3 ms	2.6 ms
Syn pulse low level	0.6 ms	0.55 ms	0.7 ms
bit 0 period	1.2 ms	1.1 ms	1.3 ms
bit 1period	1.8 ms	1.7 ms	1.9 ms
SIRC message reception time	45 ms	-	-
Carrier pulse bit time	25 µs	-	-

Note: *The table shows an overview of the data pulse-width tolerances, used in this application note. The min-max SIRC timing can be specified by the user.*

4.2 Software implementation

**Figure 14. SIRC frame reception mechanism**

MS18555V1
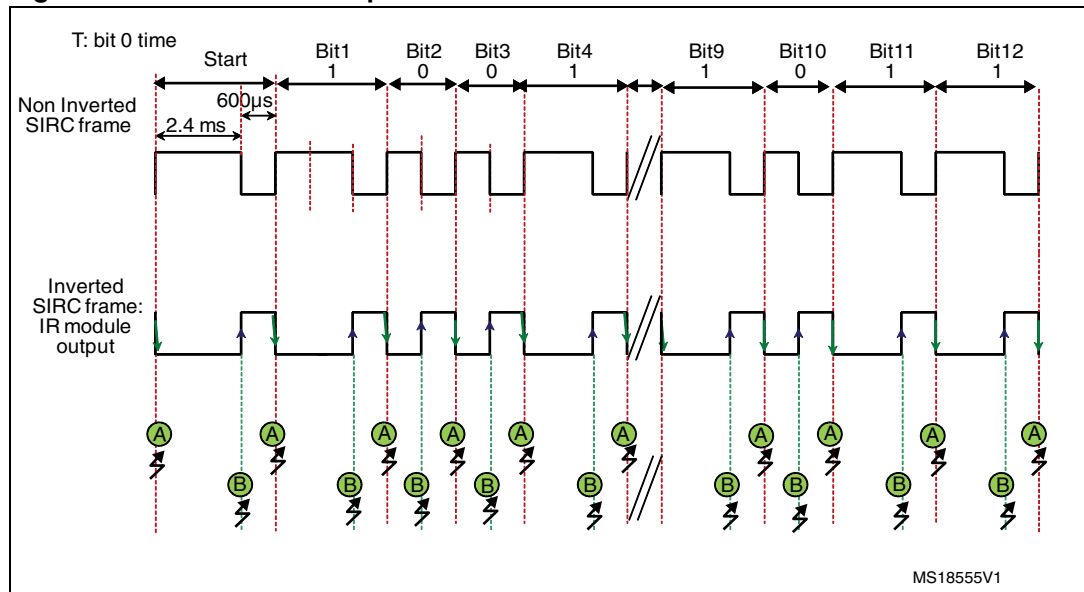
TIMER interrupt: in PWM input mode

The TIMER is used to sample the different bits of the SIRC frame. We capture the current timer value both at each falling and rising edges as well as generate an interrupt on both edges. This feature makes it easy to measure the SIRC pulse whole and low times.

- If the period measured is equal to T = 1200 μs and the low pulse duration is equal to T/2 = 600 μs => the bit is **_logic '0'._**

- If the period measured is equal to 3T/2 = 1800 μs and the low pulse duration is equal to T = 1200 μs => the bit is **_logic '1'._**

- If the whole period measured is equal to 3000 μs and the low pulse duration is equal to 2400 μs => the bit is **_'start bit'._**

Figure 15.   SIRC solution flowchart
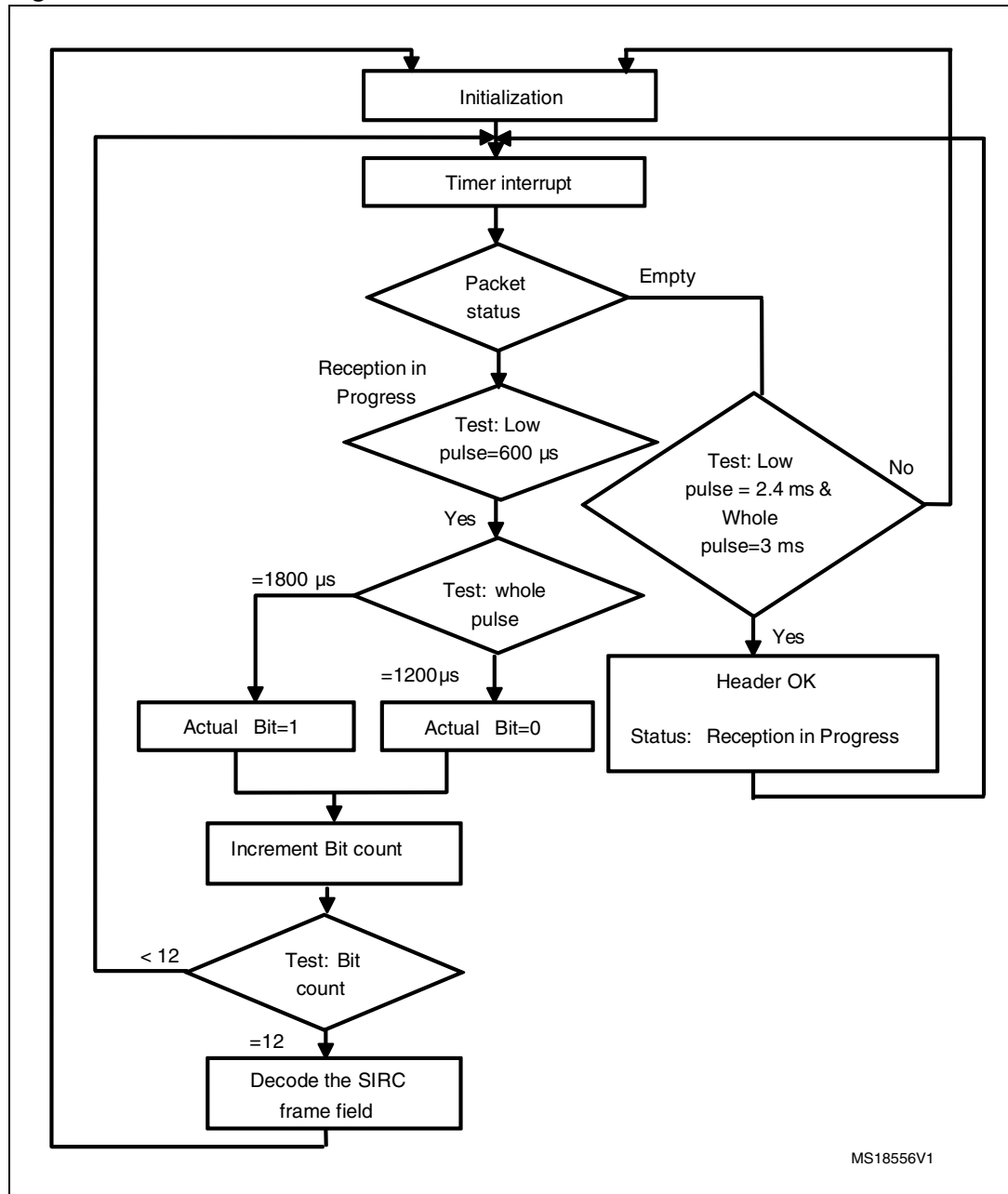
MS18556V1

4.3 SIRC library

IR_Init()

This function initializes the different peripherals used for SIRC protocol.

IR_Decode (IRFrame_TypeDef *ir_frame)

This function is intended to be called in the user application. It decodes the SIRC received messages. It has as a parameter a structure that contains the different values of the IR frame.

```
typedef struct
{
__IO uint8_t Address;          /* Address field    */
__IO uint8_t Command;          /* Command field    */
} IRFrame_TypeDef;
```

IR_decode () must be executed when the IRFrameReceived flag is equal to YES.

IR_ResetPacket()

This function puts the IR packet to default state. This function is called in the TIM3_IRQHandler routine. It occurs each timer overflow to reset the IR packet.

IR_DeInit()

This function de-initializes the different peripherals used for SIRC protocol.

TIM3_IRQHandler ()

This function handles the TIM Capture Compare interrupt.

– **Timer Falling Edge Event**: This is used to measure the different periods between two successive falling edges in order to identify the frame bits.

– **Timer Rising Edge Event**: This is used to measure the duration between falling and rising edges (The low pulse duration).

– **Update event (time-out event)** This resets the RC5 packet. The Timer Overflow is set to 4 ms .

From these two durations, we determine the bit value.

4.4 How to use the SIRC decoder driver

To use the SIRC decoder driver, please proceed as follows:

- TIM3 Capture Compare and Update interrupts are used to decode the IR frame, if a frame is received correctly a global variable "IRFrameReceived" will be set to inform the application.
- Then the application should call the function IR_Decode() to retrieve the received IR frame.
- You can easily tailor this driver to any other InfraRed protocol, by just adapting the defines from sirc_decode.h to the InfraRed protocol specification(Bit Duration, Header Duration, Marge Tolerance, Number of bits...) and the command and device tables from ir_commands.c

Code Example

```
#include "ir_decode.h"
#include <stdio.h>
#include "stm32100e_eval_lcd.h"
IR_Frame_TypeDef IR_FRAME;
/* Initialize the Color LCD mounted on STM32100E-EVAL */
STM32100E_LCD_Init();
/* Clear the LCD */
LCD_Clear(LCD_COLOR_WHITE);
IR_Init();
/* SIRC */
IR_Decode(&IR_FRAME);
```

Note: *1* *TIM3_IRQHandler ISRs are coded within the ir_decode.c driver.*

- If you are using one or both interrupts in your application you have to proceed carefully:
  - Either add your application code in these ISRs
  - Or copy the contents of these ISRs in your application code

2 *You can easily tailor this application to your own hardware using different define declarations inside the "ir_decode.h" file.*

Table 5. Example of implementation

| Define name	Description	Allowed values	Example
#define IR_TIM	Timer used for IR decoding	TIM1, TIM2, TIM3, TIM4, TIM5, TIM8, TIM9, TIM12, TIM15	TIM3
#define TIM_PRESCALER	TIM prescaler	This parameter is computed in manner to have 1us as time base. TIM Frequency(MHz) / (Prescaler + 1)	23
#define IR_TIM_CLK	APB Clock of the used timer	Refer to RCC FW driver	RCC_APB1Periph_TIM3

Define name	Description	Allowed values	Example
#define IR_TIM_IRQn	IR TIM IRQ	Refer to stm32f10x_startup.s	TIM3_IRQn
#define IR_TIM_Channel	IR TIM Channel	TIM_Channel_1 or TIM_Channel_2	TIM_Channel_1
#define IR_GPIO_PORT	Port which IR output is connected	Refer to your product pinout[1]	GPIOC
#define IR_GPIO_PORT_CLK	IR pin GPIO Clock Port	Refer to RCC FW driver	RCC_APB2Periph_GPIOC
#define IR_GPIO_PIN	Pin which IR is connected	Refer to your product pinout[1]	GPIO_Pin_6

1. For more details on the available STM32 resources, please refer to your product datasheet.

5   Interface layer

To facilitate the use of the infrared protocol drivers, an interface layer (IR_decode) is used.

Figure 16.   Application layer architecture
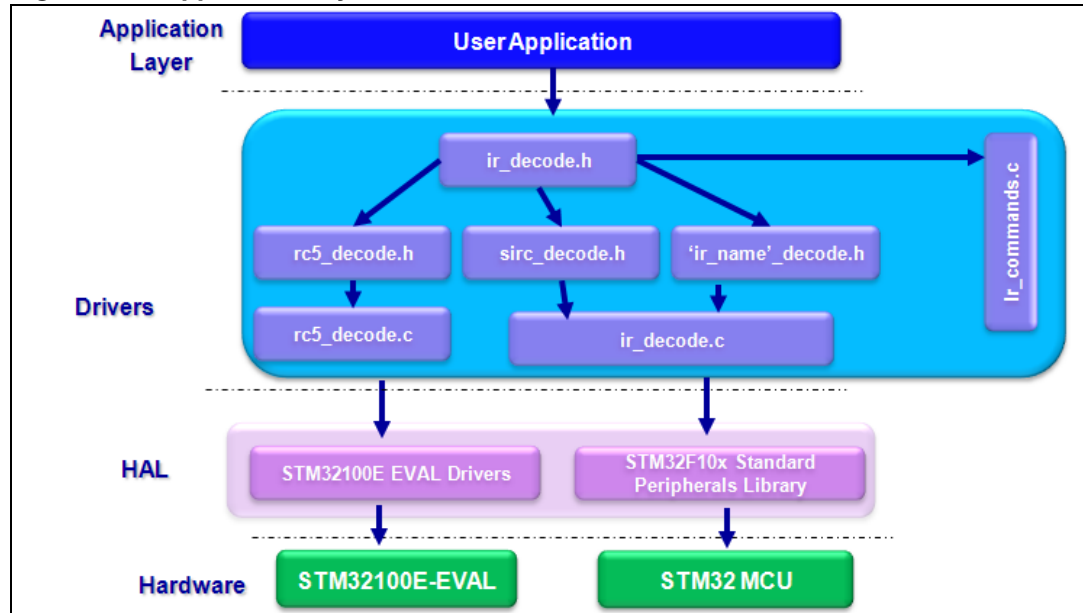
Infrared Protocol is selectable through firmware defines in IR_decode.h.

```
#define IR_RC5_PROTOCOL
#define IR_SIRC_PROTOCOL
```

There are many similar Infrared protocols which are differentiated by timing parameters like the SIRC protocol. These protocols are handled by ir_decode.c functions. You need only to update the timing values.

There are others that are quite different and are managed by specific functions like RC5 and its associated driver rc5_decode.c

Each protocol has a specific structure frame. IR_FRAME is a pointer to the selected infrared protocol structure and it contains the main information needed for communication (Device-Address and Command).

This layer offers an easy way to use the firmware.

```
int32_t main(void)
  {
#if defined IR_RC5_PROTOCOL
  IR_RC5_Init();
#else
  IR_Init();
#endif

  while(1)
   {
   #if defined (IR_RC5_PROTOCOL)
   /* RC5 */
   IR_RC5_Decode(&IR_FRAME);
   #else
   IR_Decode(&IR_FRAME);
   #endif
   }
  }
```

6   How to use the IR driver

To run the InfraRed applications examples, please proceed as follows:

– Open the project named "IR_Decoding_PWMI"
  (path:\Project\InfraRed\IR_Decoding_PWMI)

– Select your preferred toolchain

– Select the corresponding workspace in your preferred toolchain project. You can select IR_RC5_Decode or IR_SIRC_Decode. These workspaces are already configured.

6.1   Demonstration programs

### 6.1.1   Demo using GP-Timer configured in PWM mode

This demo consists of receiving IR messages and sending them to the LCD.

Each IR message is displayed in 2 parts:

● The device which transmitted the IR frame

● The command to be executed

The following figure shows the RC5 Decoder using the PWMI method (*Section 3.3: Software implementation using a GP-Timer configured in PWM input mode on page 10*).

Figure 17.   RC5 received frame shown in the LCD
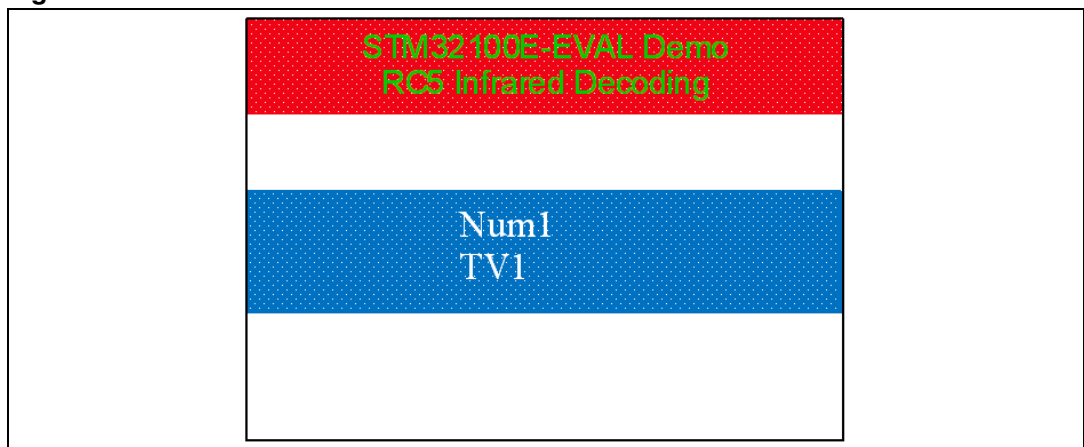
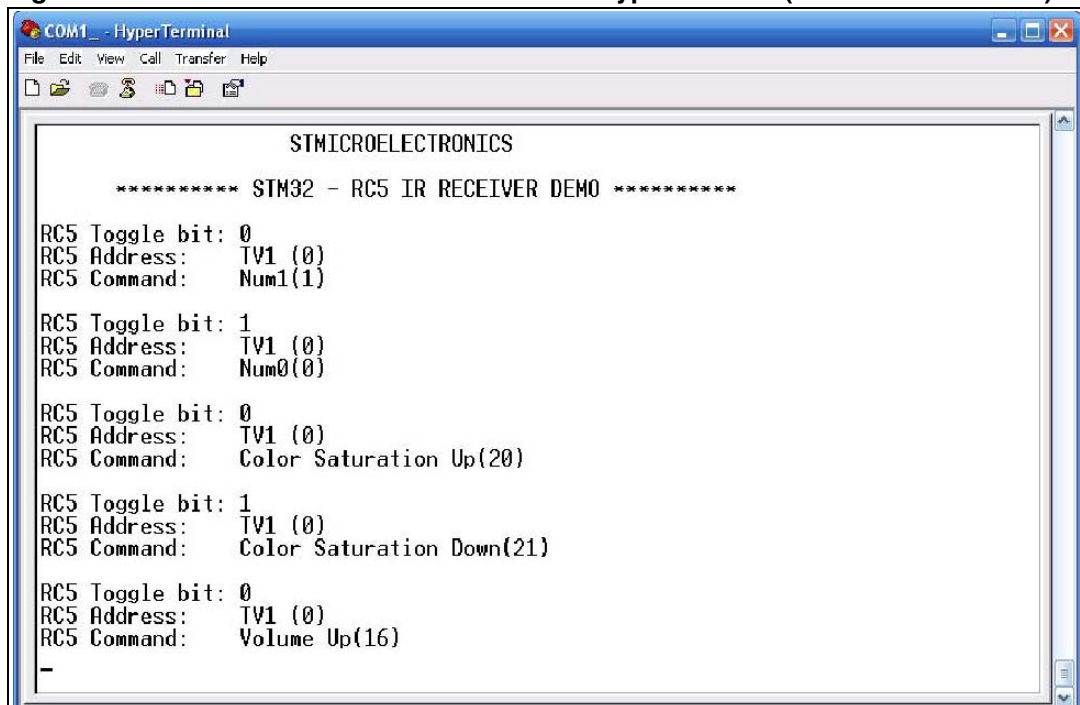### 6.1.2   RC5 demo using a single GPIO with generic timer

This demo consists of receiving RC5 messages and sending them to the hyperterminal (see *Figure 18*) using USART1 (115200 baud, 8-bit data, No parity, No Flow Control). Each RC5 message is displayed in 3 parts:

● The value of the toggle bit.

● The device which transmitted the RC5 with its decimal value in brackets.

● The command to be executed and its decimal value in brackets.

When an RC5 message is received, LED1 toggles on the board.

See also *Section 3.2: Software implementation using a single GPIO with a generic timer on page 7*).

Figure 18.   RC5 received frames shown in the hyperterminal (basic timer solution)

7 How to customize the IR drivers

To include an Infrared driver based on the PWM input solution in a user application, you have to:

1. Open the project named "IR_Decoding_PWMI"
2. Add the header file of the appropriate IR protocol to your project.
    Example: rc5_decode.h.
3. Add the file.c corresponding to the IR protocol to your project.
    Example: rc5_decode.c.
4. Call the function of protocol initiation in the main()
    Example: `IR_RC5_Init();`
5. Add to stm32f10x_it.c : the TIMx interrupt functions.

Example:

```
void TIM3_IRQHandler (void)
{
   static uint32_t ICValue1;
   static uint32_t ICValue2;
   /* IC1 Interrupt*/
   if((TIM_GetFlagStatus(IR_TIM, TIM_FLAG_CC1) != RESET))
   {
      TIM_ClearFlag(IR_TIM, TIM_FLAG_CC1);
      /* Get the Input Capture value */
      ICValue2 = TIM_GetCapture1(IR_TIM);
      /* RC5 */
   IR_RC5_DataSampling(ICValue2 - ICValue1, 0);
   }
   /* IC2 Interrupt */
   else if((TIM_GetFlagStatus(IR_TIM, TIM_FLAG_CC2) != RESET))
   {
      TIM_ClearFlag(IR_TIM, TIM_FLAG_CC2);
      /* Get the Input Capture value */
      ICValue1 = TIM_GetCapture2(IR_TIM);
      IR_RC5_DataSampling(ICValue1 , 1);
   }
   /* Checks whether the IR_TIM flag is set or not.*/
   else if ((TIM_GetFlagStatus(IR_TIM, TIM_FLAG_Update) != RESET))
   {
   /* Clears the IR_TIM's pending flags*/
   TIM_ClearFlag(IR_TIM, TIM_FLAG_Update);
   IR_RC5_ResetPacket();
   }
}
```

6. Define a structure for the IR protocol in the file main.c.
    Example:

```
IRFrame_TypeDef  IR_FRAME.
```
7. Call the decoding function in main().

 Example:

```
void main(void)
{
    IR_RC5_Init();
    while(1)
    {
    IR_RC5_Decode(&IR_Frame).
    }
}
```

For the RC5 decoder solution on GPIO with a generic timer, you have to:

1.   Open the project named 'RC5_Decoding_TIM_EXTI'
2.   Add the header file of the RC5 protocol to your project: RC5_IR_Emul_Receiver.h
3.   Add the source file to your project: RC5_IR_Emul_Receiver.c
4.   Define a sturcture for RC5.

Example:

RC5Frame_TypeDef RC5_FRAME.

5.   Add the stm32f10x_it.c file which contains the TIMx interrupt and the EXTI functions.
6.   Call the decoding function

Example:

```
void main(void)
{
RC5_Receiver_Init();
while(1)
{
if (RC5_Frame_Received==YES)
  {
  RC5_FRAME = IR_RC5_Decode().
  }
}
}
```

Changes needed to support any IR protocol

You can use this solution to support any Infrared protocol by making only a few changes in the header file and updating the command and device tables.

–   Create a header file (exp: ir_protocol_name.h) similar to the sirc_decode.h file. Change the defines to adapt it to the specifications of the chosen IR protocol (Bit duration Min/Max, Header duration Min/Max, Total Bits number, TimeOut...)

Table 6. List of defines in the header file for the IR protocol parameters

Defines	Meaning	Example settings for SIRC protocol
IR_Time_OUT_US	Timeout in µs	4000
IR_BITS_COUNT	Number of bits	11
IR_TOTAL_BITS _COUNT	Total number of bits	11
IR_ONTIME_MIN_US	Min Low pulse in µs	(600 - 50)
IR_ONTIME_MAX_US	Max Low pulse in µs	(1200 + 50)
IR_HEADER_LOW_MIN_US	Min Header Low pulse in µs	(2400 - 100)
IR_HEADER_LOW_MAX_US	Max Header Low pulse in µs	(2400 + 100)
IR_HEADER_WHOLE_MIN_US	Min Header whole duration in µs	(2400 + 600 - 50)
IR_HEADER_WHOLE_MAX_US	Max Header whole duration in µs	(2400 + 600 + 50)
IR_VALUE_STEP_US	Step value between bit0 and bit1 in µs	600
IR_VALUE_MARGIN_US	Margin in µs	100
IR_VALUE_00_US	Bit0 duration in µs	1200

– Change the the IR protocol frame field in IR_Frame_TypeDef structure.

```
typedef struct
{

/* Structure of the IR frame ( Address, Command,....)*/

} IR_Frame_TypeDef;
```

– in the ir_commands.c file, add the appropriate IR_Commands and IR_devices tables for the IR protocol.
– in the ir_decode.c file, update the function IR_Decode(IR_Frame_TypeDef *ir_frame) with the frame structure of the target protocol.

8 Conclusion

This application note provides a solution for implementing an IR receiver in software using a general purpose timer.

The IR driver allows the IR solution to be integrated in the HDMI-CEC module in order to support high-level control functions for all of the various audiovisual products in a given environment.

9   Revision history

Table 7.   Document revision history

Date	Revision	Changes
01-Apr-2010	1	Initial release.
17-Feb-2012	2	Updated with generic solution to support different IR protocols

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2012 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com