AN4834
Application note

Implementation of transmitters and receivers for infrared remote control protocols with STM32Cube

Introduction

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

Infrared radiation has two ranges. Near infrared light is closest in wavelength to visible light, while 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 simple 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 protocol standards used to transmit data via infrared light, such as (among others) RC5 and SIRC.

The purpose of this application note is to provide a generic solution for implementing an IR transmitter (a remote control device) and receiver using microcontrollers of the STM32F0, STM32F3, STM32G0, STM32G4, STM32L4, STM32L4+, STM32L5, STM32WB and STM32WL Series. An example of software implementation is provided for RC5 and SIRC protocols. Other protocols are similar and it's straightforward to adapt the example to support other manufacturer specific communication.

The infrared transmitter and receiver solutions described in this document are implemented using STM32Cube hardware abstraction layer and the X-CUBE-IRREMOTE firmware package, available on *www.st.com*.

Note: Although most STM32 MCUs feature the IRTIM, a typical STM32 EVAL board is not equipped with the necessary IR components. The X-CUBE-IRREMOTE therefore does not support this EVAL board, only F0, F3 and G0 examples are provided.

Contents

List of tables

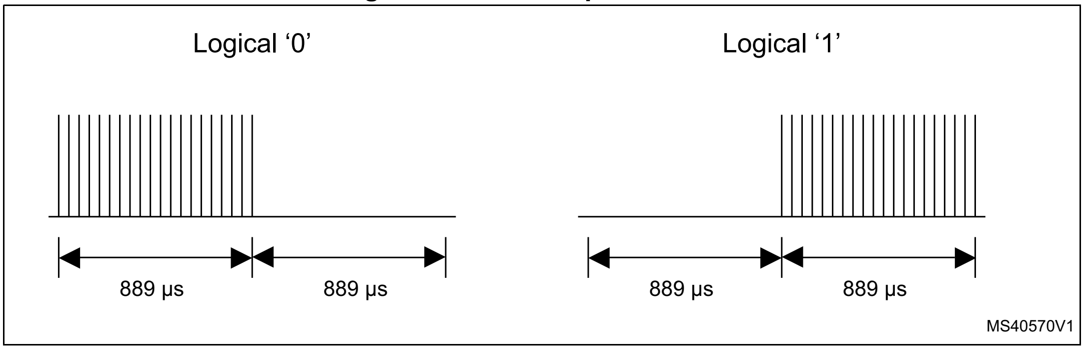
List of figures

1 Infrared protocol specification

1.1 RC5 protocol basics

The RC5 code is a 14-bit word that 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 "1" 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.

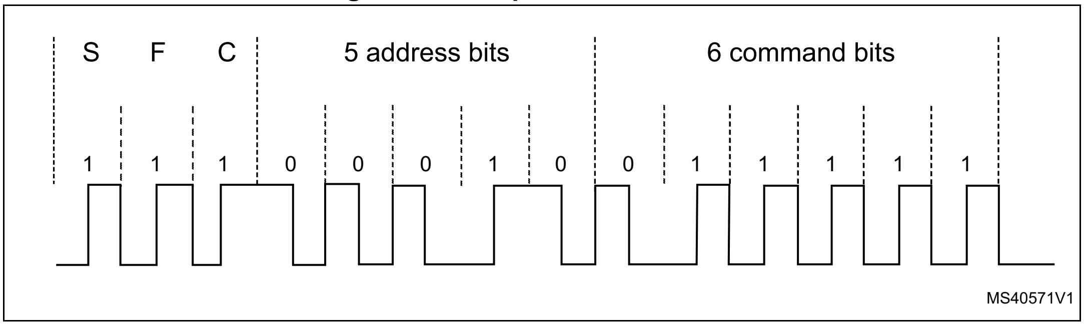
Figure 1. RC5 bit representation



The RC5 frame can generate 2048 (32 x 64) different commands organized in 32 groups. Each group has 64 different commands. An RC5 frame contains the following fields. An example of an RC5 frame is shown in *Figure 2*.

- **Start bit (S):** 1 bit length, always logic "1".
- **Field bit (F):** 1 bit length, denotes whether the command sent is in the lower field (logic 1 = 0 to 63 decimal) or in the upper field (logic 0 = 64 to 127 decimal). The field bit was added later when it was realized that 64 commands per device weren't sufficient. 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, 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, selects one of 32 possible systems.
- **Command:** 6 bits length, in conjunction with the field bit represents one of the 128 possible RC5 commands.

Figure 2. Example of an RC5 frame



To avoid frame collisions, an idle time is inserted between two successive frames with a specific width (see *Figure 3*).

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 3. RC5 idle time

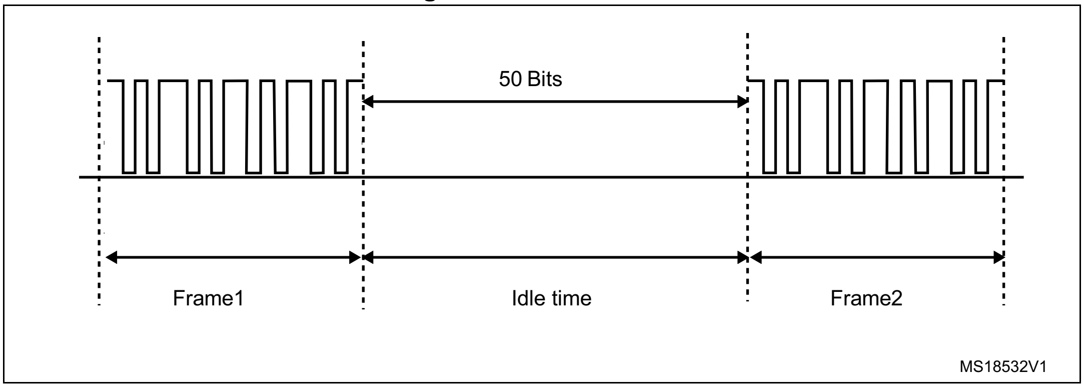


Table 1. RC5 timings

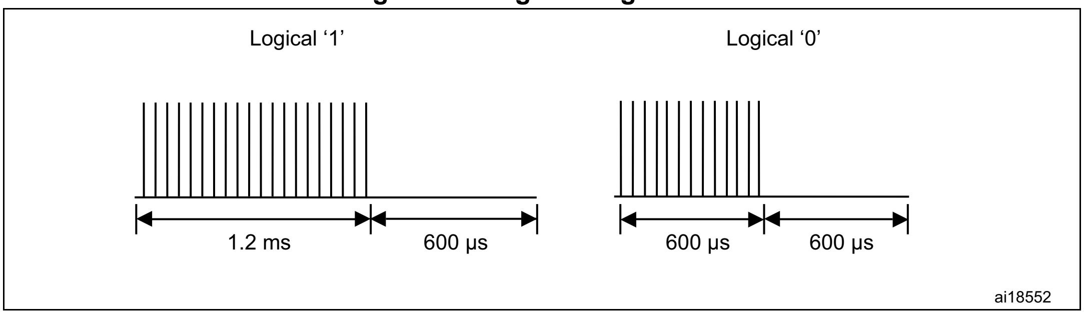
Description	Min	Typical	Max
RC5 half bit period	640 µs	889 µs	1140 µs
RC5 full bit period	1340 µs	1778 µs	2220 µs
RC5 message time	23.644 ms	24.889 µs	26.133 ms
RC5 message repetition time	108.089 ms	113.778 ms	119.467 ms
Carrier pulse bit time	27.233 µs	27.778 µs	28.349 µs

Note: The infrared protocol implementation is based on free RC5 specifications downloaded from http://www.sbprojects.com.

1.2 SIRC 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 (*Figure 4*).

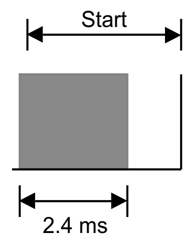
Figure 4. Length of logical bits



A SIRC frame contains the following fields.

- **Start bit:** the start burst is always 2.4 ms wide, followed by a standard space of 0.6 ms.
- **Command 7 bits length:** this field holds 7 bits that are used as command fields.
- **Address 5 bits length:** this field holds 5 bits that are used as address fields.

Figure 5. Length of start bit

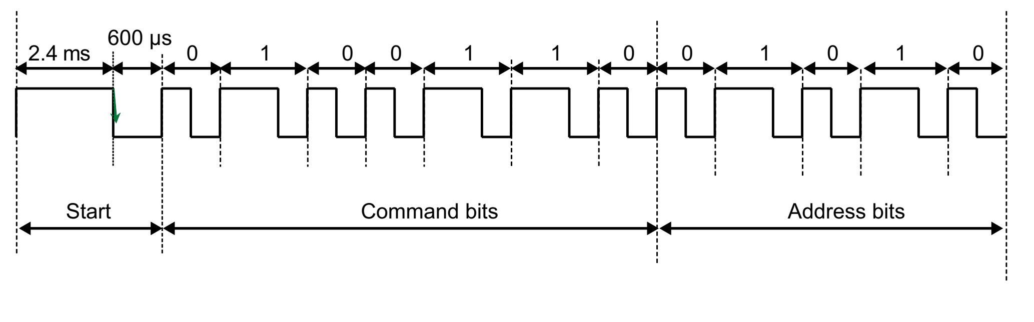


With this protocol, the LSB is transmitted first (the frame is assembled LSB to MSB). Since it is sent as 7 bits for command, followed by 5 bits for device address, the code must split the 12 received bits into two groups of 7 and 5 bits.

Figure 6 shows an example of a SIRC frame.

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

Figure 6. Example of SIRC framet



An idle time is inserted between two successive frames in order to avoid collisions. Every 45 ms a repeat code is transmitted.

Table 2. 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 1 period	1.8 ms	1.7 ms	1.9 ms
SIRC message reception time	45 ms	-	-
Carrier pulse bit time	25 µs	-	-

Note:
1. *The infrared protocol implementation is based on free SIRC specifications downloaded from http://www.sbprojects.com.*
2. *Table 2 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.*

2 Infrared transmitter

2.1 Hardware considerations

The TX-IR LED is an infrared transmitter designed for infrared serial data links and remote control applications. Data present is modulated at the selected carrier frequency (of 36 kHz or 40 kHz as in the example) providing a simple, single-chip solution for infrared data communications and remote control applications.

An infrared interface (IRTIM) for remote control is available on STM32F0xx, STM32F3xx and STM32L4xx devices. It can be used with an IR LED to perform remote control functionality.

The IR digital interface is designed to output a digital signal to an infrared diode driver circuit. It can output a signal using any of the existing modulation styles, the modulation style being dependent on the software algorithm.

The IR interface is very easy to configure and uses two signals provided by two STM32 timers (TIM16 and TIM17 for most of STM32 products, TIM15 and TIM16 for STM32L4x1/2/3 products).

TIM_HF (TIM17 typically) is used to provide the carrier frequency, while TIM_LF (TIM16 typically) provides the actual signal to be sent.

Figure 7. Hardware configuration for infrared transmitter

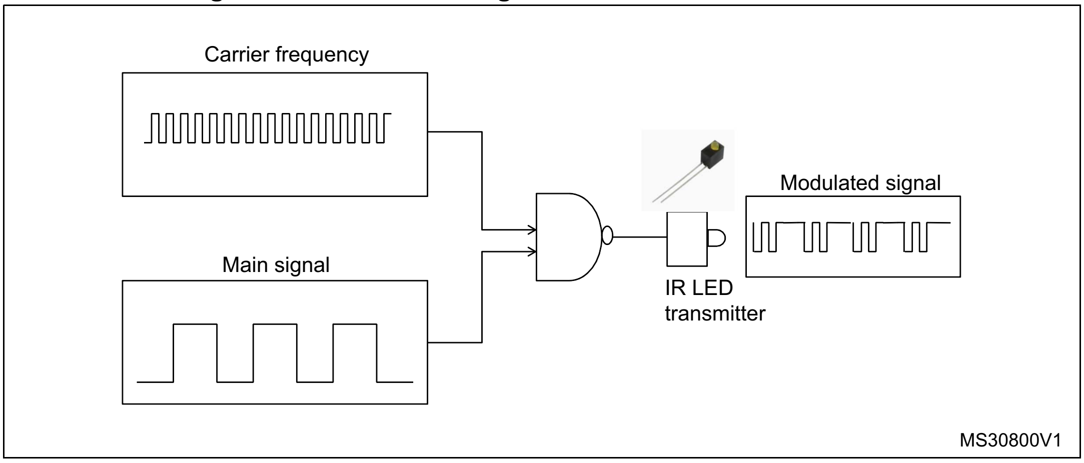
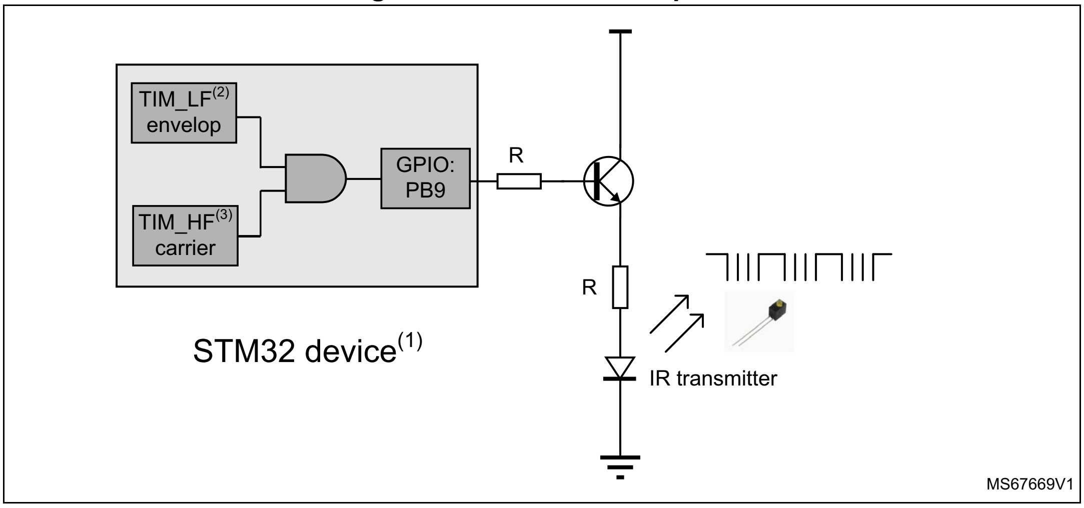


Figure 8. Hardware description



1. In STM32L41x/L42x/L43x/L44x/L45x/L46x products the IRTIM functionality is associated with TIM15 and TIM16.
2. TIM16 on most products.
3. TIM17 on most products.

2.2 IR transmitter: universal solution

The infrared transmitter solution based on the STM32 enables the user to send IR to various receiver devices.

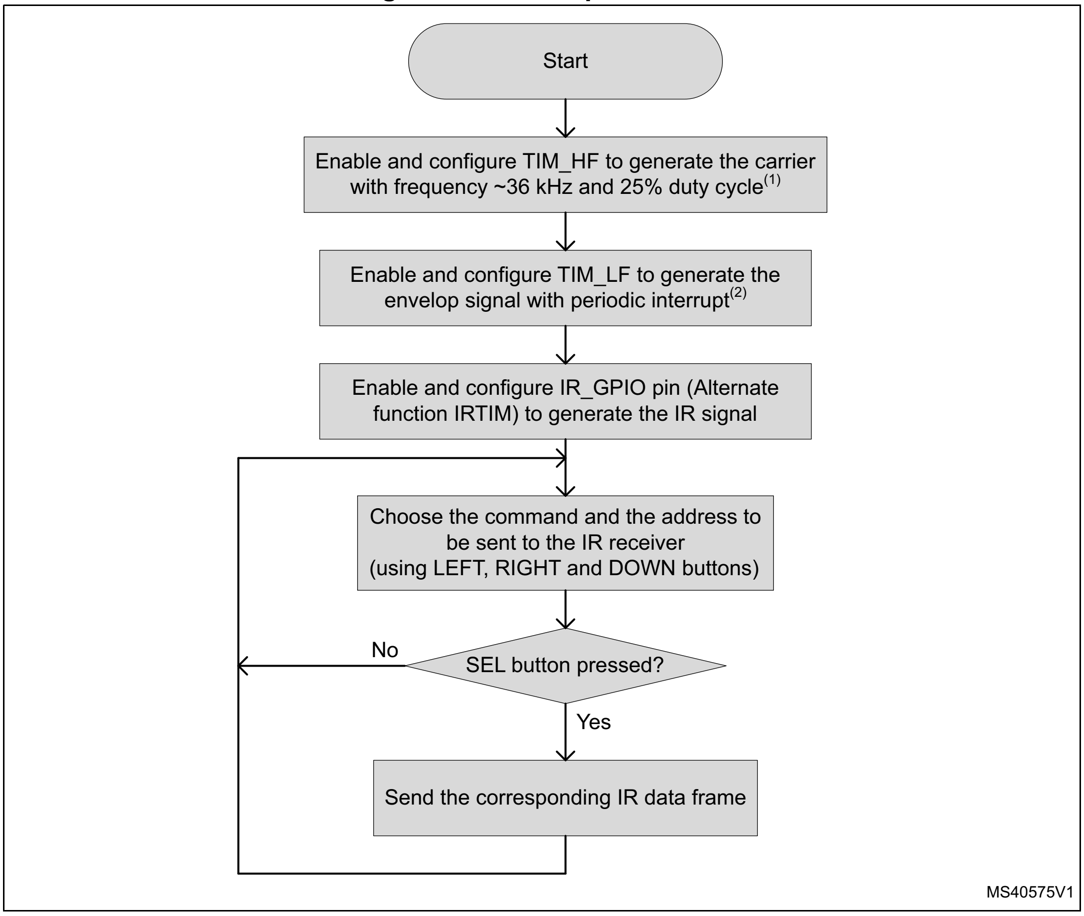
The application solution uses four peripherals:

- IRTIM: (IR interface with timers) generates the IR signal using TIM_HF and TIM_LF
  - TIM_HF (TIM17 on STM32Fxxx products): provides the carrier signal with a frequency of depending on the protocol specification
  - TIM-LF (TIM16 on STM32Fxxx products): provides the main signal to be sent (RC5 Frame, SIRC Frame or others protocol Frame)
- GPIO: (general-purpose I/O) provides the I/O to be connected to the buttons of the remote control and connected to the IR-LED
- CLK: (clock controller) enables the clocks and provides the correct clock frequency for the timers

To generate the infrared remote control signals, TIM_LF channel 1 (TIMX_OC1) and TIM_HF channel 1 (TIMX_OC1) must be properly configured to generate correct waveforms. All standard IR pulse modulation modes can be obtained by programming the two timer output compare channels. The infrared function is output on the TIM_IR pin. The activation of this function is done through the GPIOx_AFRx register by enabling the related alternate function bit. The reference manuals also mention the I2C_PB9_FMP bit in the SYSCFG_CFGR1 register to activate the high current sink capability for direct control of the Infra LED. With the circuitry used in the EVAL boards, this bit should remain reset.

The main program flow is shown in *Figure 9*.

Figure 9. Main loop flowchart



1. The carrier frequency is specific for each IR protocol (in the example it is 36 kHz for RC5 and 40 kHz for SIRC).
2. The envelop signal is specific for each IR protocol (in the example it is 889 µs for RC5 and 600 µs for SIRC).

The objective of TIM_HF is to generate the carrier signal.

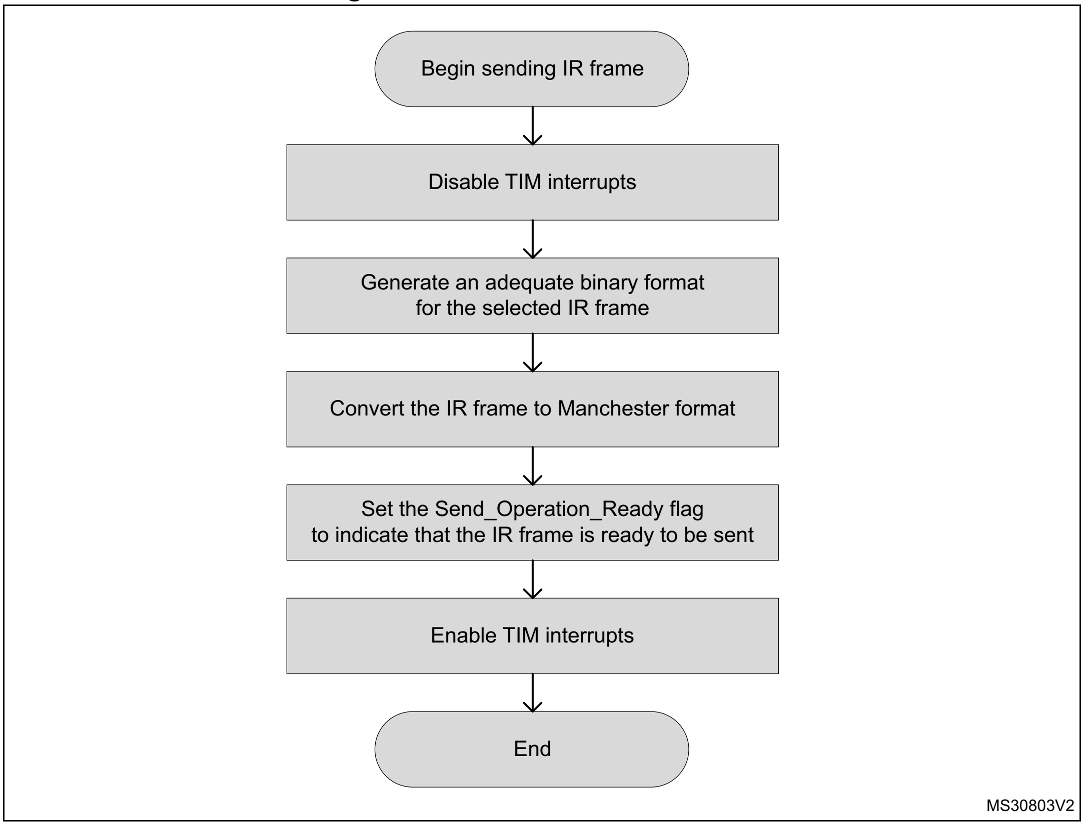
TIM_HF_Period = (SystemCoreClock / FrequencyCarrier) - 1

TIM_LF is used to generate the envelop signal.

TIM_LF_Period = (SystemCoreClock / FrequencyEnvelop) - 1

Once the modules are initialized (IRTIM, frame fields), the application waits for the SEL button to be pressed to send the IR data. *Figure 10* shows the *send frame* flowchart.

Figure 10. Send IR frame flowchart

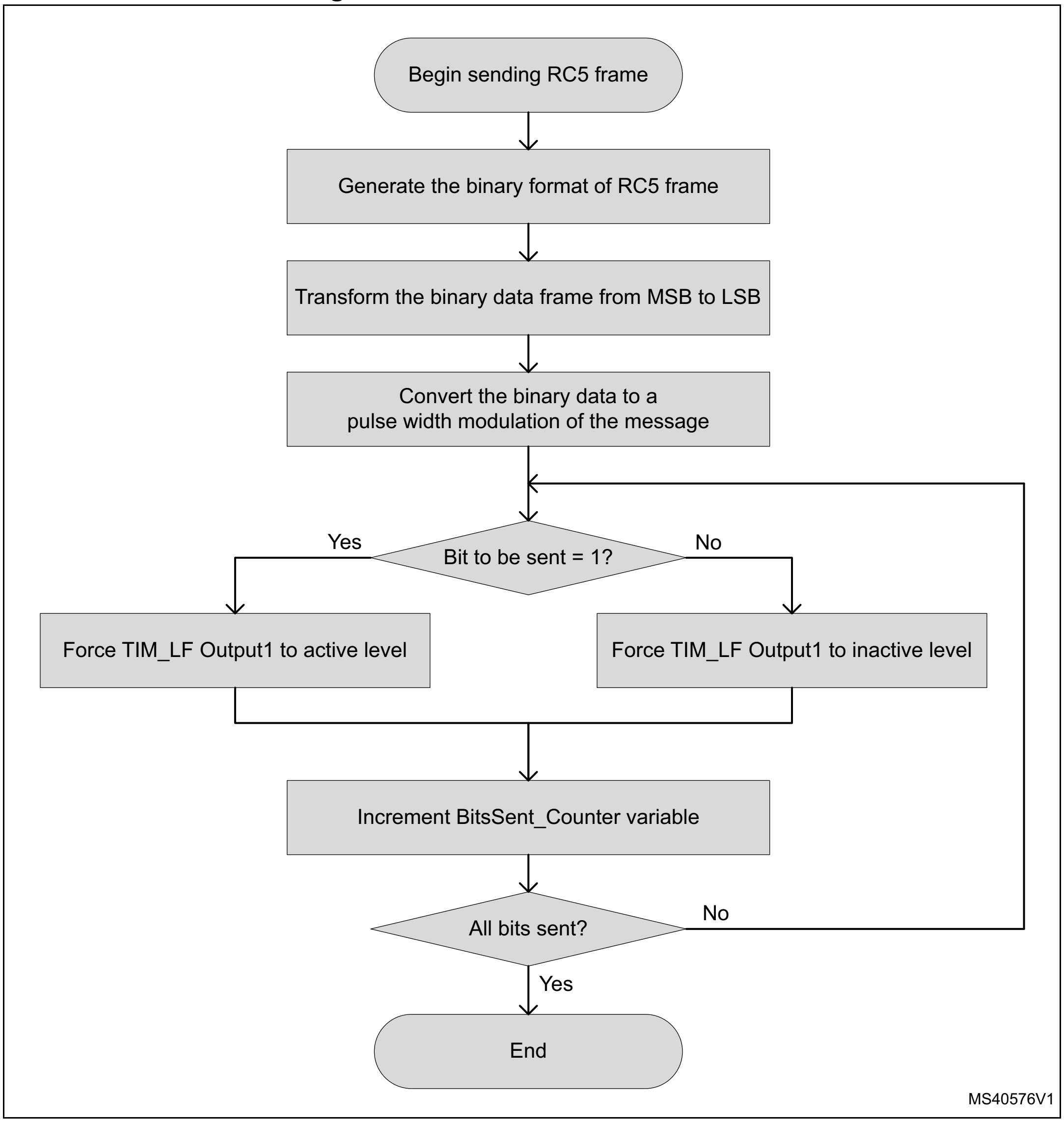


2.2.1 RC5 encoder solution

RC5 encoding mechanism

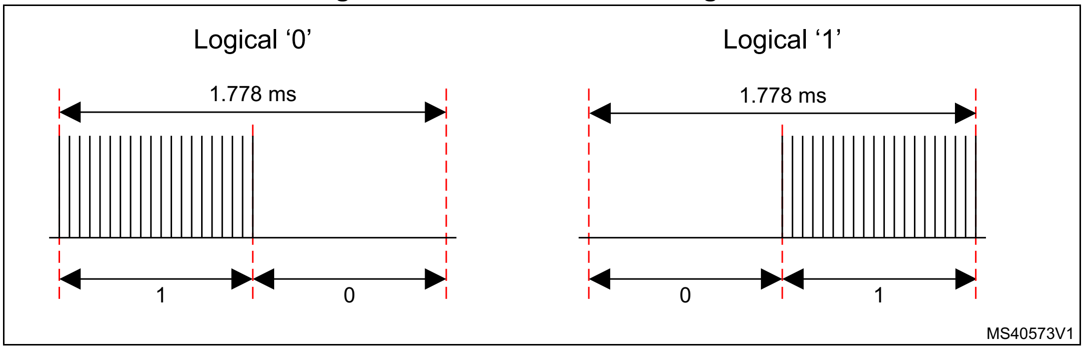
Figure 11 shows how the RC5 frame is generated. The described flowchart is called during the TIM16 update interrupt routine.

Figure 11. RC5 send frame flowchart



In the Manchester encoding, a logic "0" and a logic "1" are indicated, respectively, by a 0 to 1 transition and by a 1 to 0 transition at the center of the sequence, as visually summarized in *Figure 12*, where the red dashed lines indicate the time intervals.

Figure 12. Manchester encoding bits



RC5 encoding library

The RC5 encoder driver is based on the following functions.

RC5_Encode_Init()

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

RC5_Encode_SendFrame()

This function sends the Manchester format RC5 frame.

RC5_Encode_SignalGenerate()

This function generates the frame signal by monitoring the output level of TIM_LF. It is called during the TIM_LF update interrupt to handle the output signal.

2.2.2 How to use the RC5 encoder driver

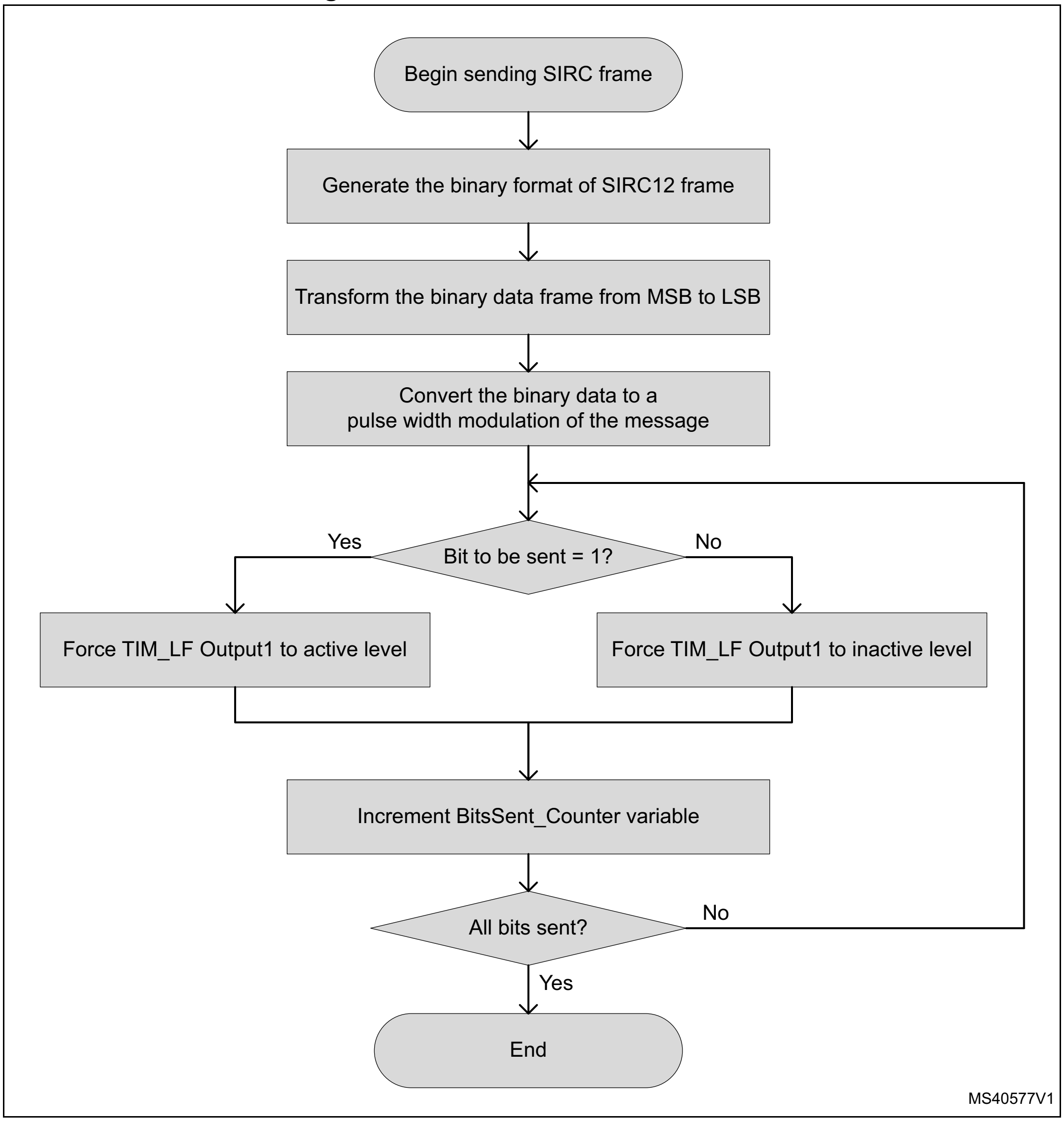
To use the RC5 encoder driver, proceed as follows.

- Call the function RC5_Encode_Init() to configure the timer and GPIO hardware resources needed for RC5 encoding.
- Call the function RC5_Encode_SendFrame() to send the RC5 frame.
- TIM_LF Update interrupts are used to encode the RC5 frame in pulse width modulation.

2.2.3 SIRC encoder solution

SIRC encoding mechanism

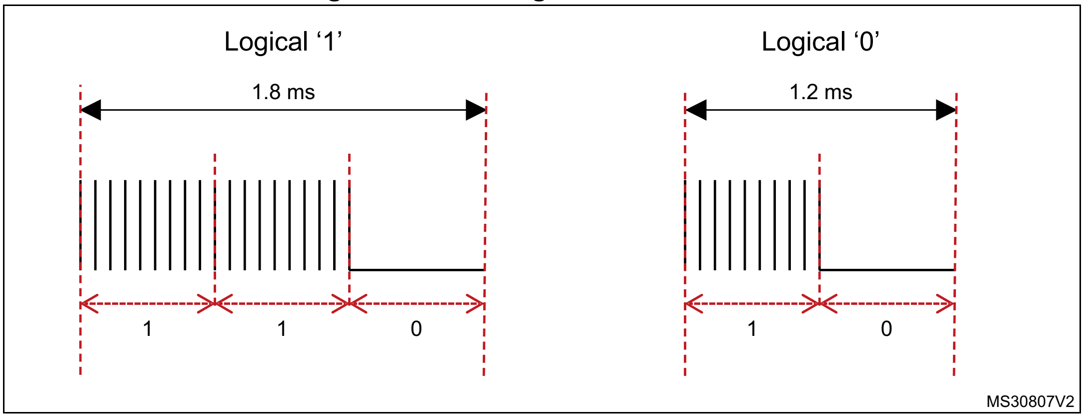
Figure 13. SIRC send frame flowchart



After generating the binary format frame, each logical bit is converted to a combination of "0"s and "1"s, representing a pulse width modulation format.

A logical "1" takes 1.8 ms to transmit, with 1.2 ms at high level and 600 µs at low level. For a logical "0", it takes 1.2 ms, with 600 µs at high level and 600 µs at low level (refer to *Figure 14*, where red dashed lines have been added to indicate the time intervals). The chosen base time is 600 µs, so the logical "1" is converted to 110, and the logical "0" to 10.

Figure 14. SIRC logical bit conversion



SIRC encoding library

The SIRC encoder driver is based on the following functions.

SIRC_Encode_Init()

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

SIRC_Encode_SendFrame()

This function sends the pulse width modulation of the SIRC12 Frame format.

SIRC_Encode_SignalGenerate()

This function generates the frame signal by monitoring the output level of TIM_LF. It is called in the TIM_LF update interrupt to handle the output signal.

2.2.4 How to use the SIRC encoder driver

To use the SIRC encoder driver, proceed as follows.

- Call the function SIRC_Encode_Init() to configure the timer and GPIO hardware resources needed for SIRC encoding.
- Call the function SIRC_Encode_SendFrame() to send the SIRC frame.
- TIM_LF Update interrupts are used to encode the SIRC frame in pulse width modulation.

3 Infrared receiver

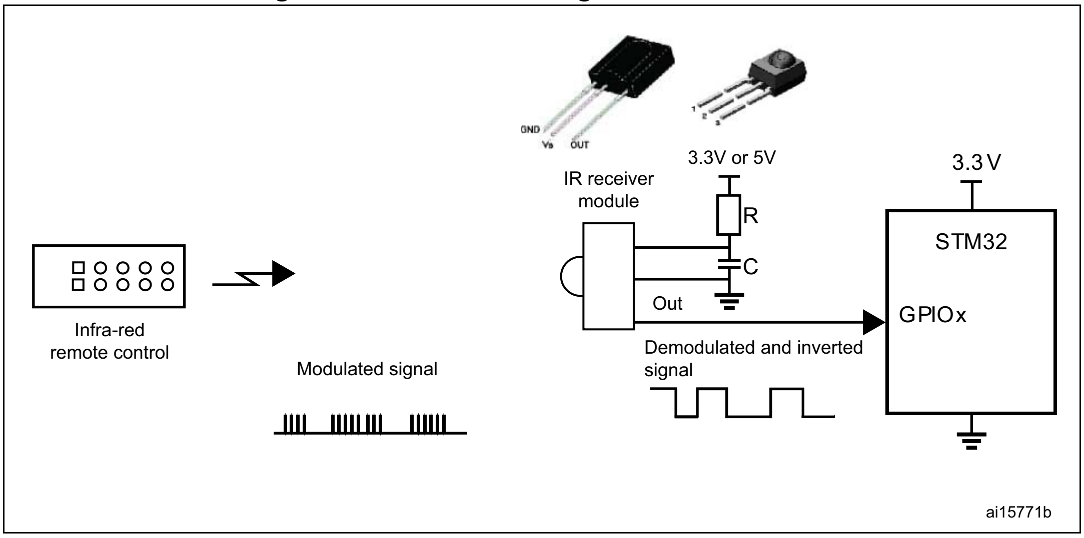
3.1 Hardware considerations

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), the TSOP34836 (3.3 V supply version), or other equivalent part numbers (see *Figure 15*).

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 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 15. Hardware configuration



3.2 Universal solution: software implementation using a GP-Timer configured in PWM input mode

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.

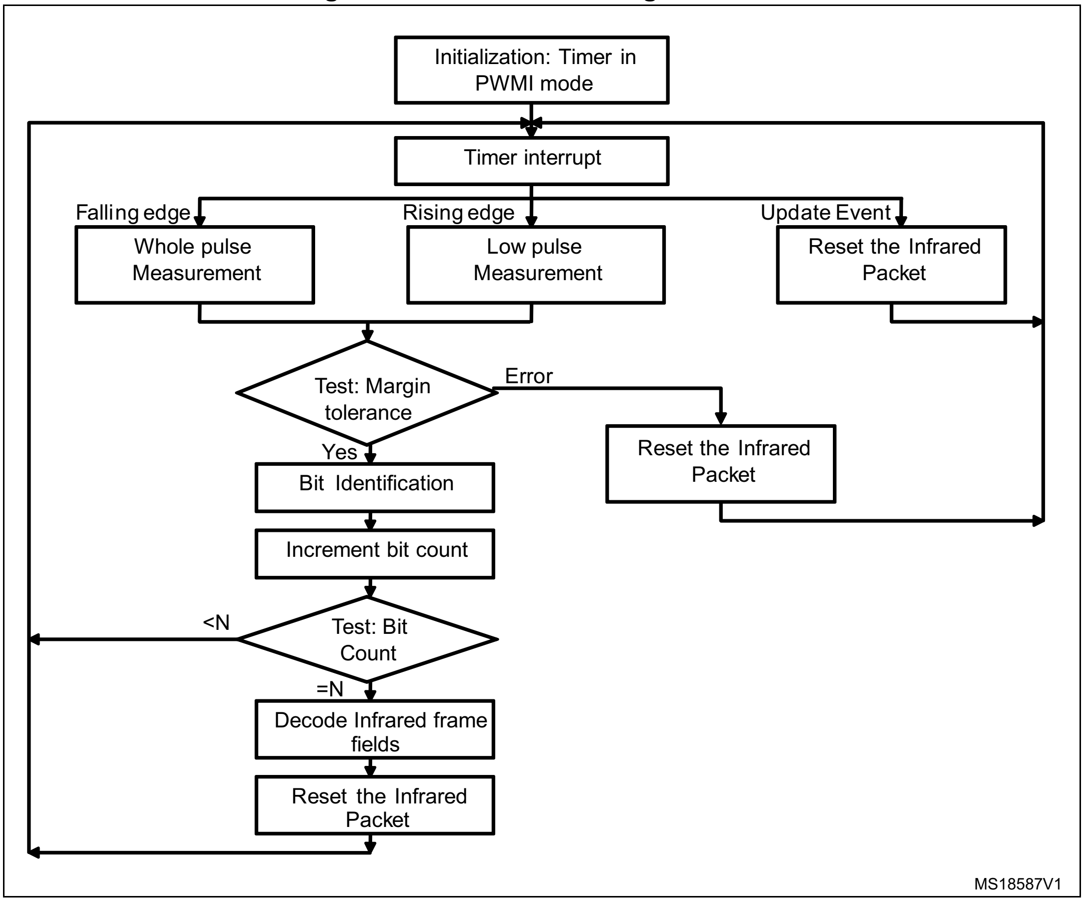
The timer generates three interrupt types

- Interrupt at each falling edge: this can be used to measure the entire 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 whole pulse duration are used to determine the bit value. If the durations are within the tolerance range of the bit time, one identifies the bit value (Logic0, Logic1 or Header).

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

Figure 16. Infrared decoding flowchart



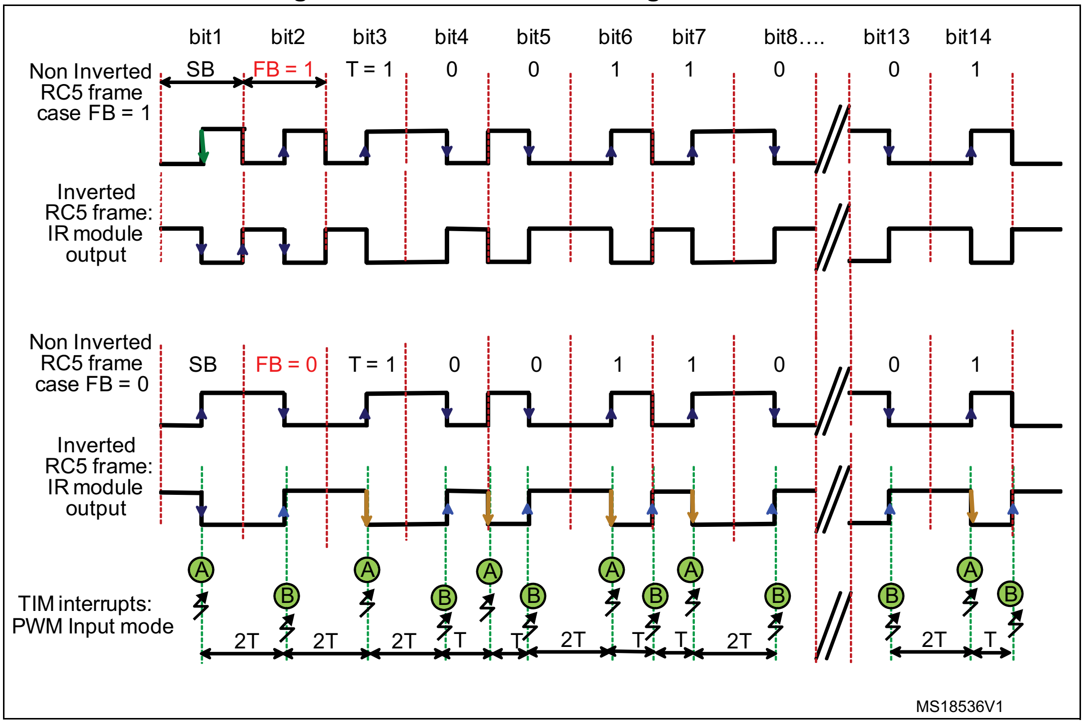
3.3 RC5 protocol solutions

3.3.1 RC5 frame decoding mechanism

Figure 17 shows how the RC5 frame is received. One of the general purpose timer of STM32 microcontrollers is 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 17. RC5 frame decoding mechanism



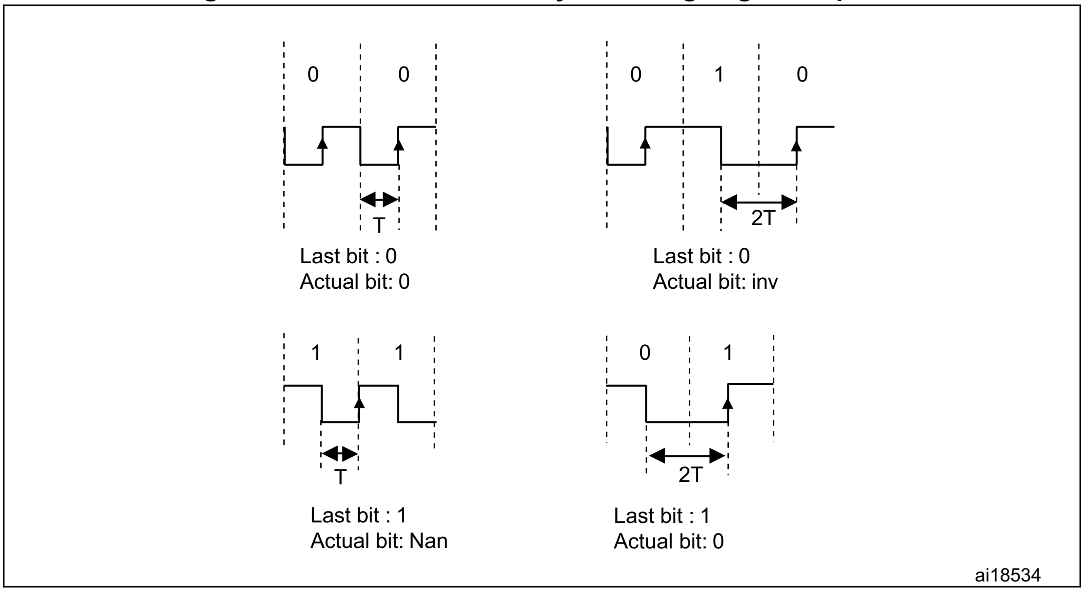
- **TIMER interrupt event:** falling edge

  A: the TIMER interrupt is used to measure the period between two successive falling edges (one or one and a half pulse duration).
- **TIMER interrupt event:** rising edge

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

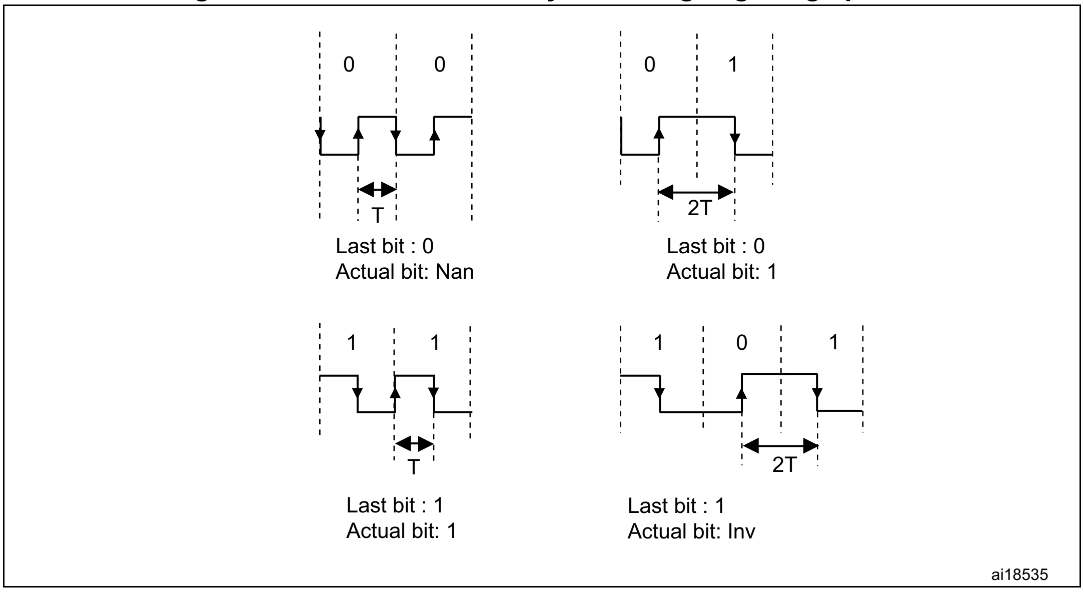
The two durations are used to determine the bit value. Each bit value is determined in relation to the last bit.

Figure 18. Bit determination by the rising edge: low pulse



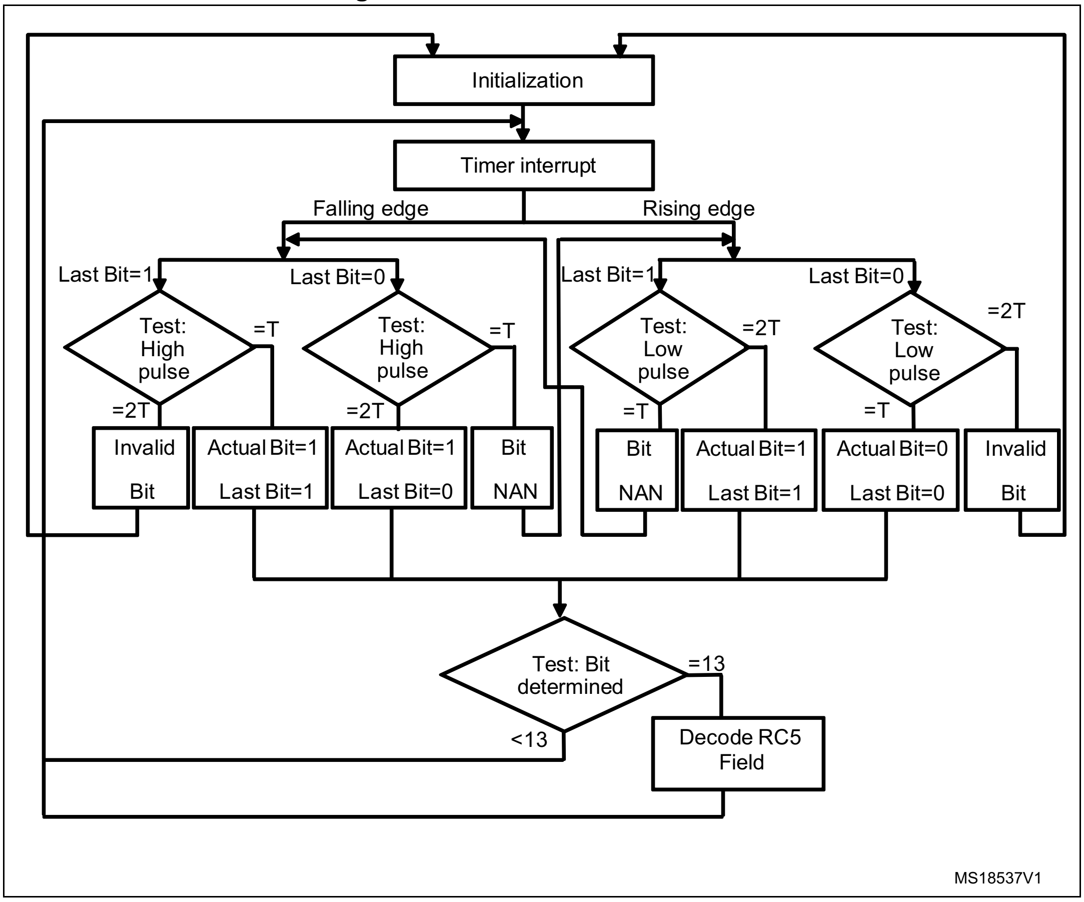
- If the low pulse duration is equal to T and the last bit determined is "0", then the actual bit is ***logic0***.
- If the low pulse duration is equal to 2T and the last bit determined is "0", then the actual bit is ***Inv*** (**invalid case**: this case cannot be released).
- If the low pulse duration is equal to T and the last bit determined is "1", then 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", then the actual bit is ***logic0***.

Figure 19. Bit determination by the falling edge: high pulse



- If the high pulse duration is equal to T and the last bit determined is "0", then 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", then the actual bit is ***logic1***.
- If the high pulse duration is equal to T and the last bit determined is "1", then the actual bit is ***logic1***.
- If the high pulse duration is equal to 2T and the last bit determined is "1", then the actual bit is ***Inv*** (**invalid case**: this case cannot be released).

Figure 20. RC5 solution flowchart



3.3.2 RC5 decoding library

The RC5 driver is very simple to use.

RC5_Init()

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

RC5_ResetPacket()

This function sets the packet structure to the default state. This function is mainly called in the HAL_TIM_PeriodElapsedCallback function. It occurs at each TIMER overflow, to reset the RC5 packet.

RC5_Decode(RC5_Frame_TypeDef *rc5_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    */
  } RC5_Frame_TypeDef ;
```

RC5_Decode () is executed when the RC5FrameReceived flag is equal to YES.

RC5_DeInit()

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

HAL_TIM_CaptureCallback ()

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

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.

HAL_TIM_PeriodElapsedCallback ()

This function handles TIM Update interrupt.

– **Update event (time-out event):** this resets the RC5 packet. The Timer Overflow is set to 3.7 ms.

3.3.3 How to use the RC5 decoder driver

To use the RC5 decoder driver, proceed as follows.

- Call the function RC5_Init() to configure the timer and GPIO hardware resources needed for RC5 decoding.
- TIM2 Capture Compare and Update interrupts are used to decode the RC5 frame, if a frame is received correctly a global variable "RC5FrameReceived" is set to inform the application.
- The application should then call the function RC5_Decode() to retrieve the received RC5 frame.

Code example

```
#include "rc5_decode.h"

/* IR_FRAME will hold the RC5 frame (Address, Command,...) */
RC5_Frame_TypeDef IR_FRAME;

/* Initialize the RC5 driver */
RC5_Init();

while(1)
{
```

```
        /* Decode the received RC5 frame and store it in IR_FRAME variable
        */
        RC5_Decode(&IR_FRAME);

        /* Here add the code that will process the just received frame, i.e.
           IR_FRAME variable, otherwise it will be overwritten by the next frame
           */
        ...
    }
```

Note: 1 TIMx_IRQHandler ISRs are coded within the stm32f0xx_it.c, stm32g0xx_it.c or stm32f3xx_it.c

- If one or both interrupts are used in the application special care must be taken:
  - either add the application code in these ISRs, or
  - copy the contents of these ISRs in the application code.

2 User can easily tailor this application to the hardware using different define declarations in the "ir_common.h" file. Refer to Table 3.

Table 3. Example of implementation

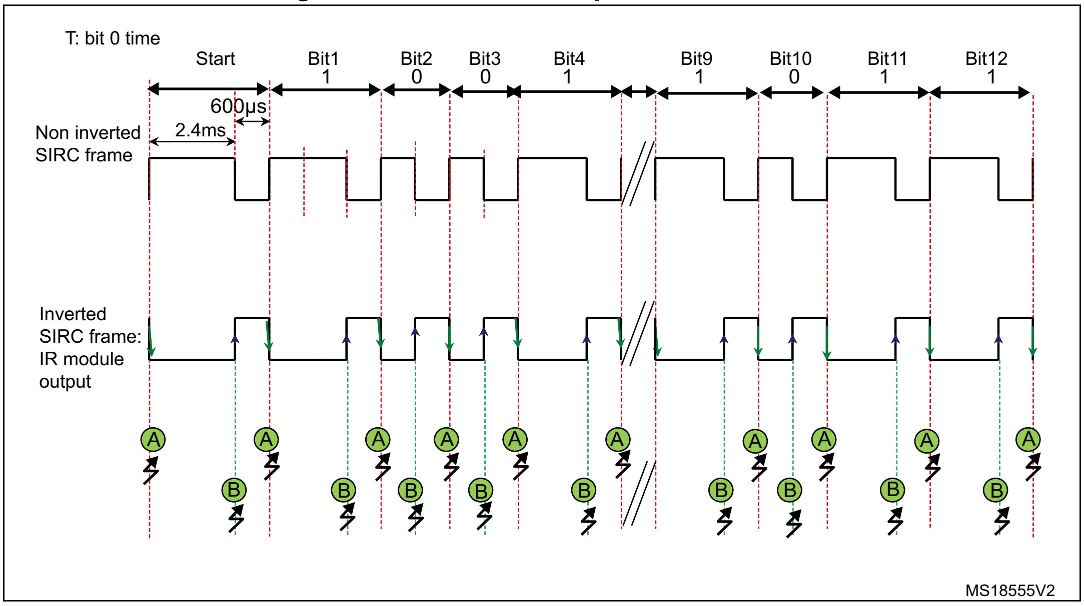
Define name	Description	STM32F07x STM32G0 Series	STM32F30x	STM32F37x
#define IR_TIM	Timer used for IR decoding(1)	TIM3	TIM1	TIM3
#define TIM_PRESCALER	TIM prescaler This parameter is computed to have 1 µs as time base. TIM frequency (in MHz) / (prescaler+1)	47	71	71
#define IR_TIM_CLK	APB clock of the used timer	__HAL_RCC_TIM3_CLK_ENABLE	__HAL_RCC_TIM1_CLK_ENABLE	__HAL_RCC_TIM3_CLK_ENABLE
#define IR_TIM_IRQn	IR TIM IRQ	TIM3_IRQn	TIM1_CC_IRQn	TIM3_IRQn
#define IR_TIM_Channel	IR TIM channel	TIM_CHANNEL_1	TIM_CHANNEL_2	TIM_CHANNEL_2
#define IR_GPIO_PORT	Port to whom the IR output is connected(1)	GPIOC	GPIOA	GPIOB
#define IR_GPIO_PORT_CLK	IR pin GPIO clock port	__HAL_RCC_GPIOC_CLK_ENABLE	__HAL_RCC_GPIOA_CLK_ENABLE	__HAL_RCC_GPIOB_CLK_ENABLE
#define IR_GPIO_PIN	Pin to whom the IR is connected(1)	GPIO_PIN_6	GPIO_PIN_9	GPIO_PIN_5

1. For more details on the available STM32 resources, refer to the product datasheets.

3.4 SIRC infrared control solution

3.4.1 Software implementation

Figure 21. IRC frame reception mechanism

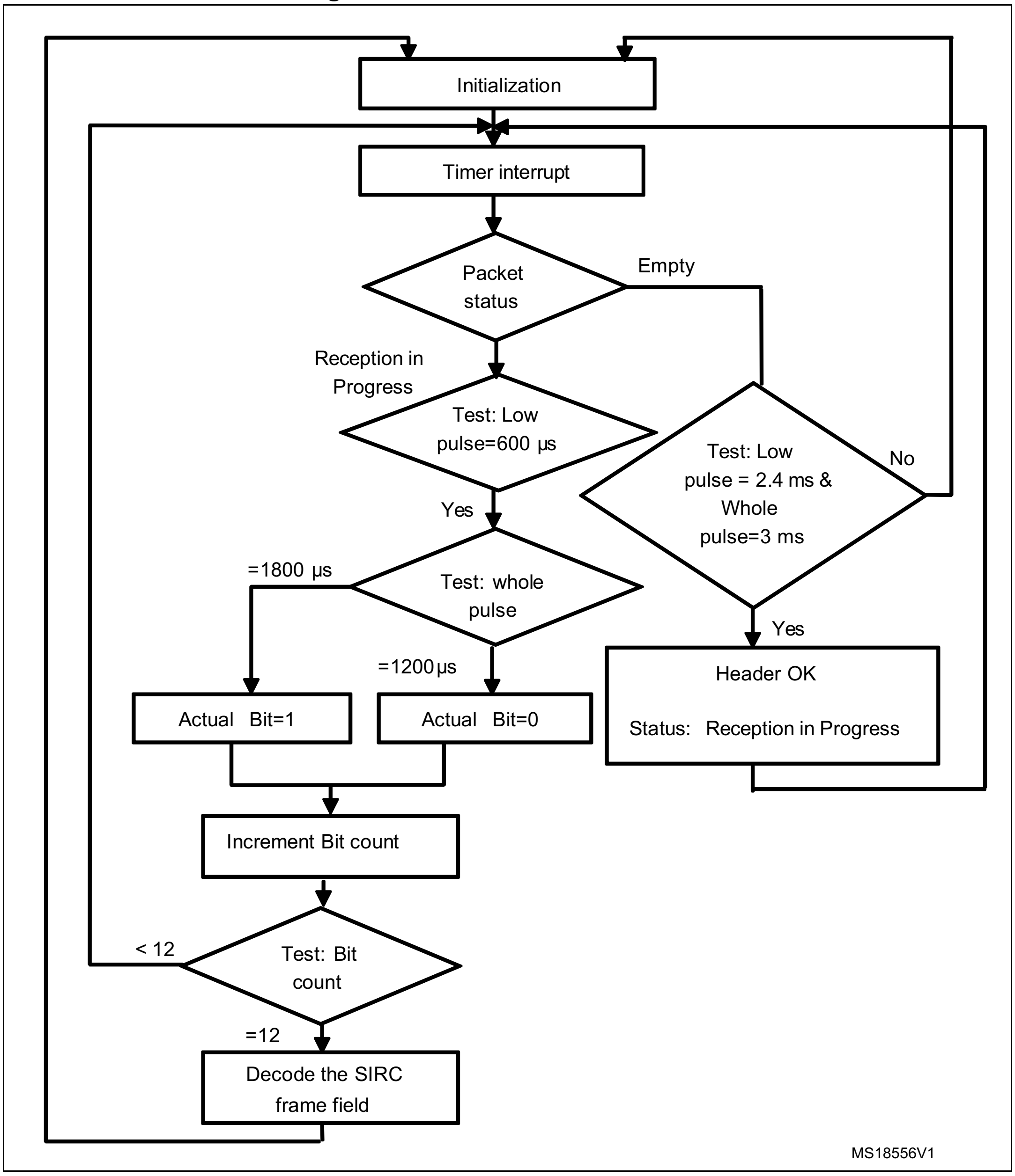


TIMER interrupt: in PWM input mode

The TIMER is used to sample the different bits of the SIRC frame. The current timer value is captured both at falling and rising edges, and an interrupt is generated 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, then 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, then 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, then the bit is ***"start bit"***.

Figure 22. SIRC solution flowchart



3.4.2 SIRC library

SIRC_Init()

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

SIRC_Decode (SIRC_Frame_TypeDef *sirc_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 Command;          /* Command field    */
   __IO uint8_t Address;          /* Address field    */
} SIRC_Frame_TypeDef;
```

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

SIRC_ResetPacket()

This function puts the IR packet to the default state. This function is called in the TIM2_IRQHandler routine. It occurs each timer overflow to reset the IR packet.

SIRC_DeInit()

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

HAL_TIM_IC_CaptureCallback

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 (low pulse duration).

The bit value is determined from these two durations.

HAL_TIM_PeriodElapsedCallback ()

This function handles TIM Update interrupt.

- **Update event (time-out event)**: this resets the RC5 packet. The timer overflow is set to 4 ms.

3.4.3 How to use the SIRC decoder driver

To use the SIRC decoder driver, proceed as follows.

- TIM2 Capture Compare and Update interrupts are used to decode the IR frame. If a frame is received correctly a global variable "IRFrameReceived" is set to inform the application.
- The application should then call the function SIRC_Decode() to retrieve the received IR frame.
- User can easily tailor this driver to any other infrared protocol by simply 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.

Code example

```
#include "sirc_decode.h"

/* SIRC_FRAME will hold the SIRC frame (Address,Command,...) */
SIRC_Frame_TypeDef SIRC_FRAME;

/* Initialize the SIRC driver */
SIRC_Init();

while(1)
   {
   /* Decode the received SIRC frame and store it in SIRC_FRAME variable */
   SIRC_Decode(&SIRC_FRAME);

   /* Here add the code that will process the just received frame, i.e.
      SIRC_FRAME variable, otherwise it will be overwritten by the next
      frame */
   ...
   }
```

Note: 1 *TIMx_IRQHandler ISRs are coded within the stm32f0xx_it.c, stm32g0xx_it.c or stm32f3xx_it.c driver.*

- If one or both interrupts are used in the application special care must be taken:
  - either add your application code in these ISRs, or
  - copy the contents of these ISRs in your application code.

2 *User can easily tailor this application to the hardware using different define declarations in the “ir_common.h” file.*

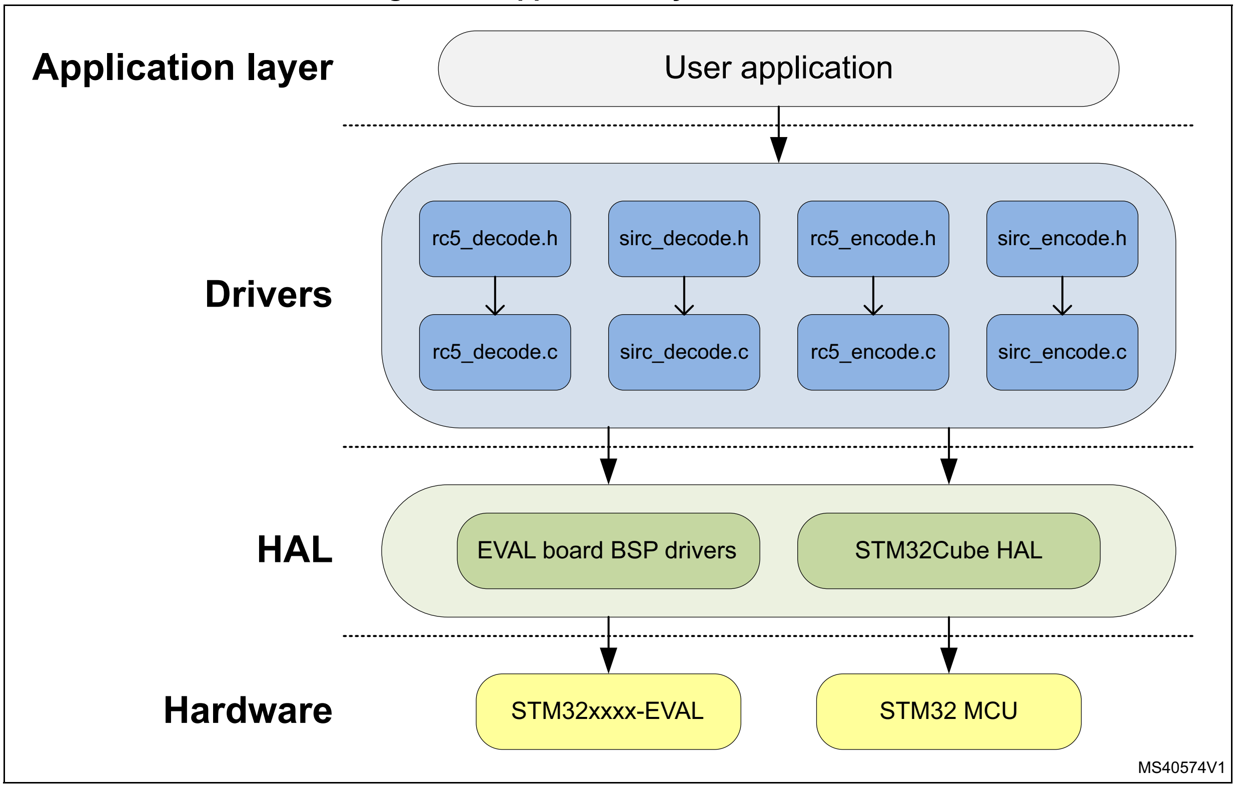
Table 4. Example of implementation

Define name	Description	STM32F07x STM32G0 Series	STM32F30x	STM32F37x
#define IR_TIM	Timer used for IR decoding(1)	TIM3	TIM1	TIM3
#define TIM_PRESCALER	TIM prescaler This parameter is computed to have 1 µs as time base. TIM frequency (in MHz) / (prescaler+1)	47	71	71
#define IR_TIM_CLK	APB clock of the used timer	__HAL_RCC_TIM3_CLK_ENABLE	__HAL_RCC_TIM1_CLK_ENABLE	__HAL_RCC_TIM3_CLK_ENABLE
#define IR_TIM_IRQn	IR TIM IRQ	TIM3_IRQn	TIM1_CC_IRQn	TIM3_IRQn
#define IR_TIM_Channel	IR TIM channel	TIM_CHANNEL_2	TIM_CHANNEL_2	TIM_CHANNEL_2
#define IR_GPIO_PORT	Port to whom the IR output is connected(1)	GPIOC	GPIOA	GPIOB
#define IR_GPIO_PORT_CLK	IR pin GPIO clock port	__HAL_RCC_GPIOC_CLK_ENABLE	__HAL_RCC_GPIOA_CLK_ENABLE	__HAL_RCC_GPIOB_CLK_ENABLE
#define IR_GPIO_PIN	Pin which IR is connected(1)	GPIO_PIN_6	GPIO_PIN_9	GPIO_PIN_5

1. For more details on the available STM32 resources, refer to the product datasheets.

4 Interface layer

Figure 23. Application layer architecture



There are many similar infrared protocols, differentiated from the SIRC protocol by timing parameters. These protocols are handled by sirc_decode.c/sirc_encode.c functions. User only needs to update the timing values.

There are others that are quite different and are managed by specific functions such as RC5 and its associated driver rc5_decode.c/rc5_encode.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).

4.1 Demonstration programs

To ensure a quick start, the infrared transmitter and receiver presented in this document are implemented in C language and are available for free download as X-CUBE-IRREMOTE package.

4.1.1 Transmitter demonstration using IRTIM

This demonstration consists in transmitting IR messages displayed on an LCD.

Each IR message is displayed in two parts

- The IR device receiver
- The command to be executed

4.1.2 Receiver demonstration using GP-Timer configured in PWM mode

This demonstration consists in receiving IR messages and sending them to the LCD.

Each IR message is displayed in two parts

- The device that transmitted the IR frame
- The command to be executed

4.2 How to customize the IR drivers

4.2.1 IR receiver drivers

To include an infrared decoder driver based on the PWM input solution in a user application, follow these steps:

1. Add the header file of the appropriate IR protocol to the project
   Example: rc5_decode.h.
2. Add the file.c corresponding to the IR protocol to the project
   Example: rc5_decode.c.
3. Call the function of protocol initiation in the main()
   Example: `RC5_Init();`
4. Add the TIMx interrupt functions to *stm32f0xx_it.c, stm32g0xx_it.c or stm32f3xx_it.c*
   Example:
   ```
   void TIM2_IRQHandler (void)
   {
      HAL_TIM_IRQHandler(&TimHandleDEC);
   }
   ```
5. Define a structure for the IR protocol in the file main.c
   Example:
   ```
   RC5_Frame_TypeDef  IR_FRAME;
   ```
6. Call the decoding function in main()
   Example:
   ```
   void main(void)
   {
      ...
      RC5_Init();
      while(1)
      {
         RC5_Decode(&IR_Frame);
      }
   }
   ```

Changes needed to support any IR protocol

This solution can be used 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 5. List of defines in the header file for the IR protocol parameters

Defines	Meaning	Example settings for SIRC protocol
SIRC_TIME_OUT_US	Timeout in µs	4050
SIRC_BITS_COUNT	Number of bits	11
SIRC_TOTAL_BITS _COUNT	Total number of bits	11
SIRC_ONTIME_MIN_US	Min low pulse in µs	(600 - 60)
SIRC_ONTIME_MAX_US	Max low pulse in µs	(1200 + 60)
SIRC_HEADER_LOW_MIN_US	Min header low pulse in µs	(2400 - 150)
SIRC_HEADER_LOW_MAX_US	Max header low pulse in µs	(2400 + 150)
SIRC_HEADER_WHOLE_MIN_US	Min header whole duration in µs	(2400 + 600 - 60)
SIRC_HEADER_WHOLE_MAX_US	Max header whole duration in µs	(2400 + 600 + 60)
SIRC_VALUE_STEP_US	Step value between bit0 and bit1 in µs	600
SIRC_VALUE_MARGIN_US	Margin in µs	100
SIRC_VALUE_00_US	Bit0 duration in µs	1200

– Change the IR protocol frame field in IR_Frame_TypeDef structure

```
typedef struct
{

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

} IR_Frame_TypeDef;
```

– in the sirc_decode.c file, add the appropriate IR_Commands and IR_devices tables for the IR protocol

4.2.2 IR transmitter drivers

To include an infrared encoder driver based on the IRTIM solution in a user application follow these steps:

1. Add the header file of the appropriate IR protocol to the project
   Example: rc5_encode.h
2. Add the file.c corresponding to the IR protocol to the project
   Example: rc5_encode.c
3. Call the protocol initiation function in the main()
   Example: `RC5_Encode_Init();`
4. Add the TIMx interrupt functions to *stm32f0xx_it.c, stm32g0xx_it.c or stm32f3xx_it.c*
   Example:

```
void TIM16_IRQHandler(void)
{
  HAL_TIM_IRQHandler(&TimHandleLF);

}
```

5. Call the encoding function in main()
   Example:

```
void main(void)
{
  ...
  RC5_Encode_Init();
  while(1)
  {
    RC5_Encode_SendFrame(Address, Instruction, Control);
  }
}
```

5 Conclusion

This application note provides a solution for software implementation of an IR transmitter/receiver using timers available on STM32 microcontrollers.

The IR encoding application uses microcontrollers of the STM32F0, STM32F3 STM32Gx, STM32Lx, and STM32Wx Series and takes advantage of the hardware modulator called IRTIM that combines signals from two internal timers to drive the IR interface. This feature makes the microcontroller especially well suited for applications that require IR signal generation capability.

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

The IR decoder implementation described in this application note works with a general purpose timer and can be ported to any STM32 microcontroller.

6 Revision history

Table 6. Document revision history

Date	Revision	Changes
14-Mar-2016	1	Initial release.
24-Nov-2020	2	Updated: – *Introduction* – *Section 2.1: Hardware considerations* – *Figure 8: Hardware description* – *Section 2.2: IR transmitter: universal solution* – *Section 3.3.3: How to use the RC5 decoder driver* – *Table 3: Example of implementation* – *Section 3.4.3: How to use the SIRC decoder driver* – *Table 4: Example of implementation* – *Section 4.2.1: IR receiver drivers* – *Section 4.2.2: IR transmitter drivers* – *Section 5: Conclusion*