AN4099
Application note

Implementing transmitters and receivers for infrared remote control protocols using STM32F0xx microcontrollers

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 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 protocol standards 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 transmitter (a remote control device) and receiver in software using a STM32F0xx microcontroller. An 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).

Note: *The infrared transmitter and receiver solutions described in this document are implemented in C language and are available within the STM320518-EVAL Demo Config2 package available on http://www.st.com.*

Contents

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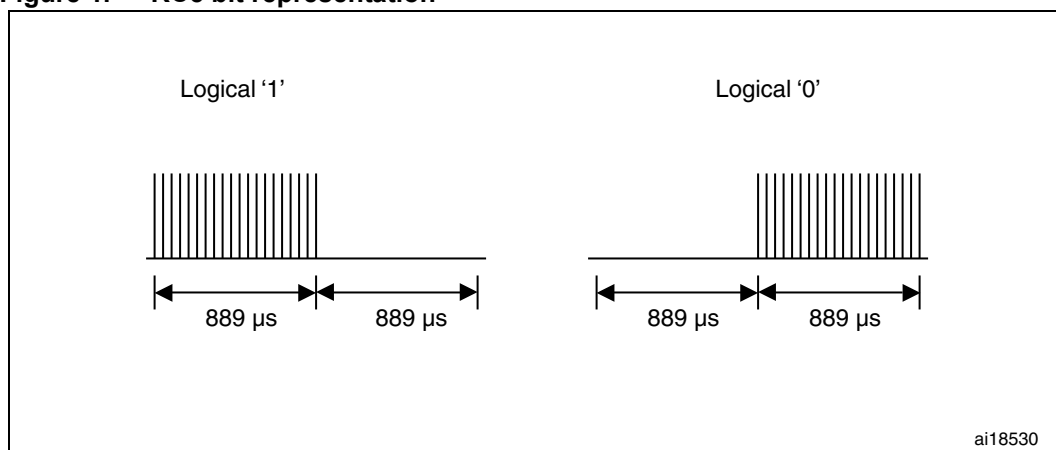
The RC5 frame can generate 2048 (32 x 64) different commands organized in 32 groups. Each group has 64 different commands. An RC5 frame contains the following fields. An example of an RC5 frame is shown in *Figure 2*.

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

● **Field bit (F):** 1 bit length, 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 2. Example of an RC5 frame

Figure 3. RC5 idle time
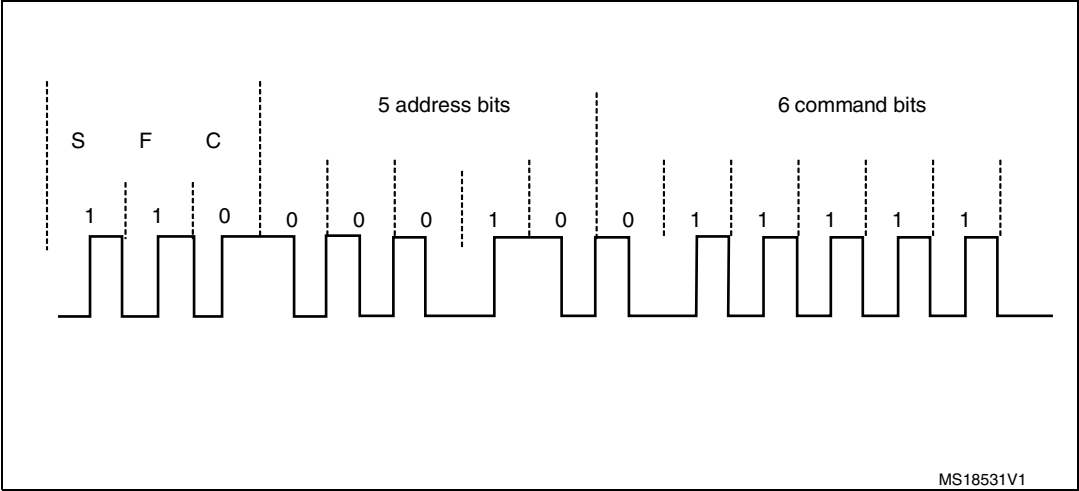
To avoid frame collisions, an idle time is inserted between two successive frames with a specific width (see *Figure 3*).

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
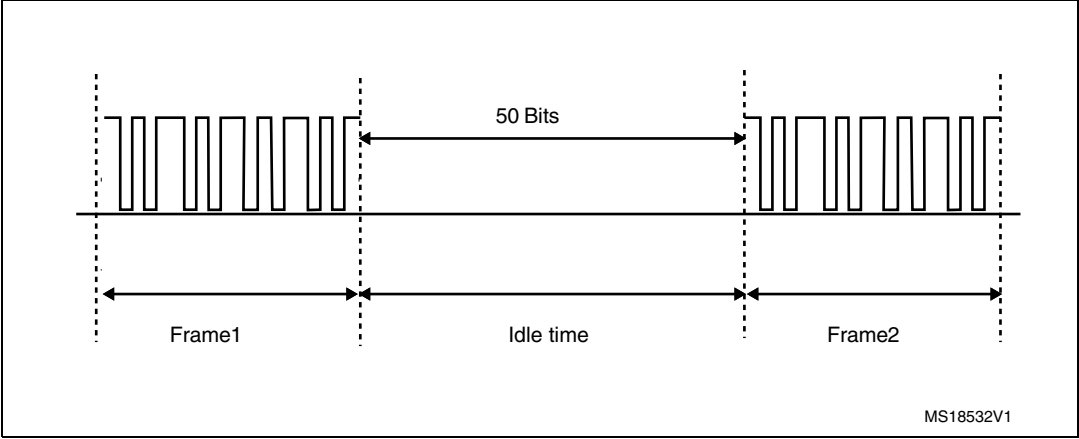
Table 1. RC5 timings

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/knowledge/ir/rc5.php.*

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
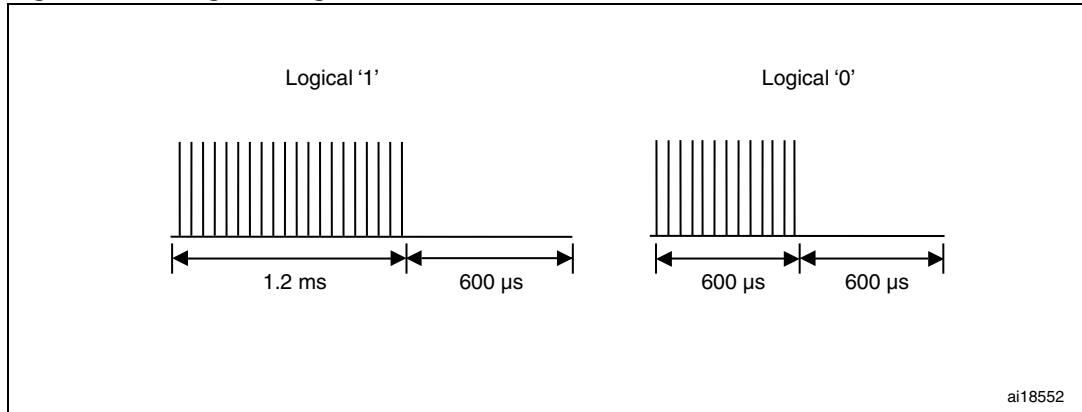
A SIRC frame contains the following fields.

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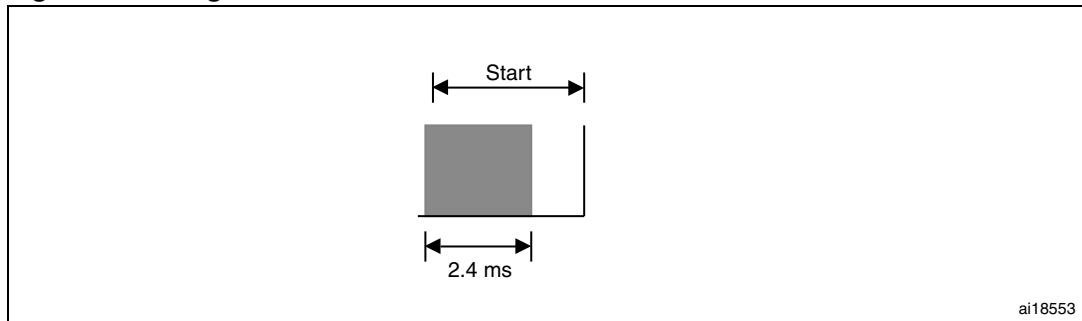
With this protocol, the LSB is transmitted first. As such, it is assembled LSB to MSB. Since it is sent as 7 bits for command, followed by 5 bits for device address, the code must split the 12 received bits into two groups of 7 and 5 bits.

Figure 6 shows an example of a SIRC frame.

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

Figure 6. Example of SIRC frame

An idle time is inserted between two successive frames in order to avoid collisions. Every 45 ms a repeat code is transmitted.
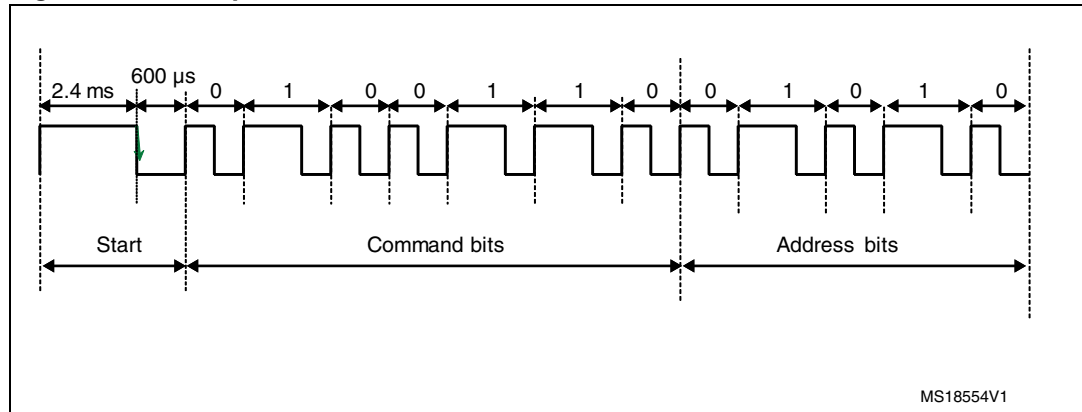
Table 2. SIRC timings

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 1period	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/knowledge/ir/sirc.php.*

2 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.

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 providing a simple, single-chip solution for infrared data communications and remote control applications.

An infrared interface (IRTIM) for remote control is available on the device STM32F0xx. 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).

TIM17 is used to provide the carrier frequency and TIM16 provides the actual signal to be sent.

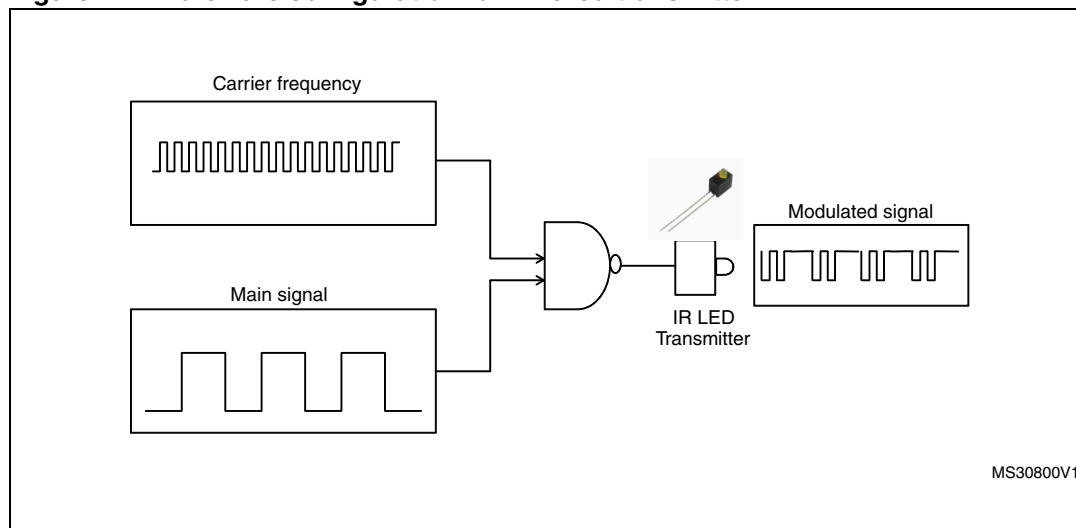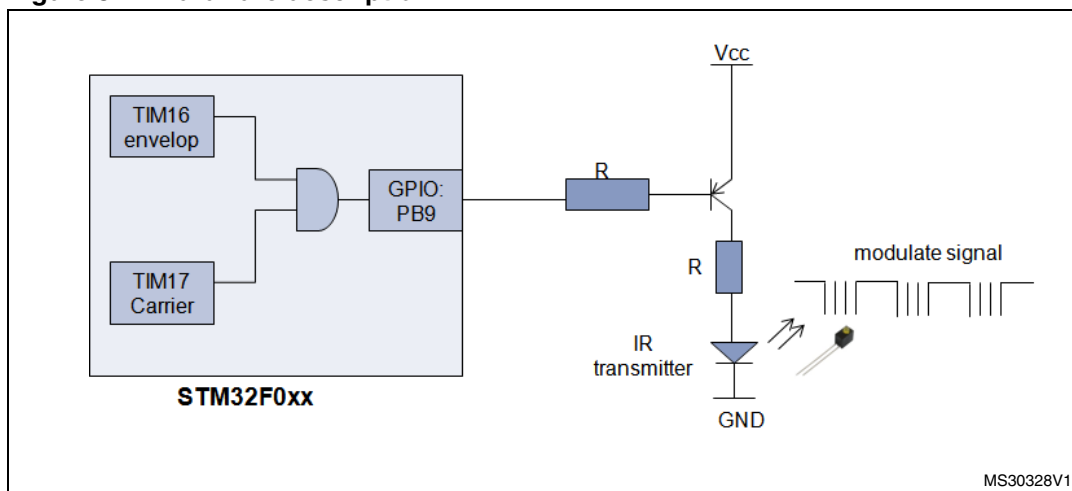**Figure 7. Hardware configuration for infrared transmitter**

Figure 8.   Hardware description

2.2   IR transmitter: universal solution

The infrared transmitter solution based on the STM32 enables the user to send all RC5 and SIRC instructions to all RC5 and SIRC receiver devices.

The application solution uses four peripherals.

● IRTIM: (IR interface with timers) generates the IR signal using TIM16 and TIM17

– TIM17: (Timer17) provides the carrier signal with a frequency of 36 kHz for RC5 and 40 kHz for the SIRC protocol.

– TIM16: (Timer16) provides the main signal to be sent (RC5 Frame or SIRC 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, the IR interface must be enabled, TIM16 channel 1 (TIM16_OC1) and TIM17 channel 1 (TIM17_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 (PB9 directly controls the Infrared LED).

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

Figure 9.   Main loop flowchart
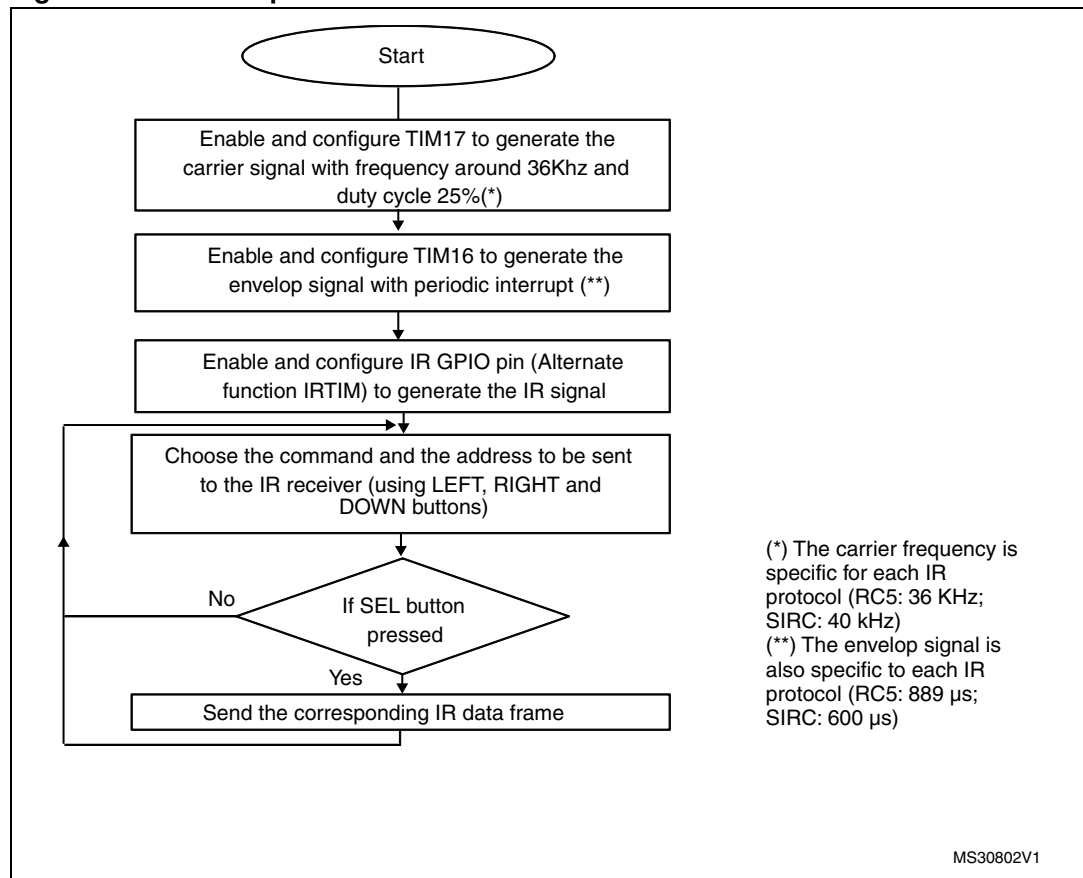
The objective of TIM17 is to generate the carrier signal.

TIM17_Period = (SystemCoreClock / FrequencyCarrier) - 1

TIM16 is used to generate the envelop signal.

TIM16_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
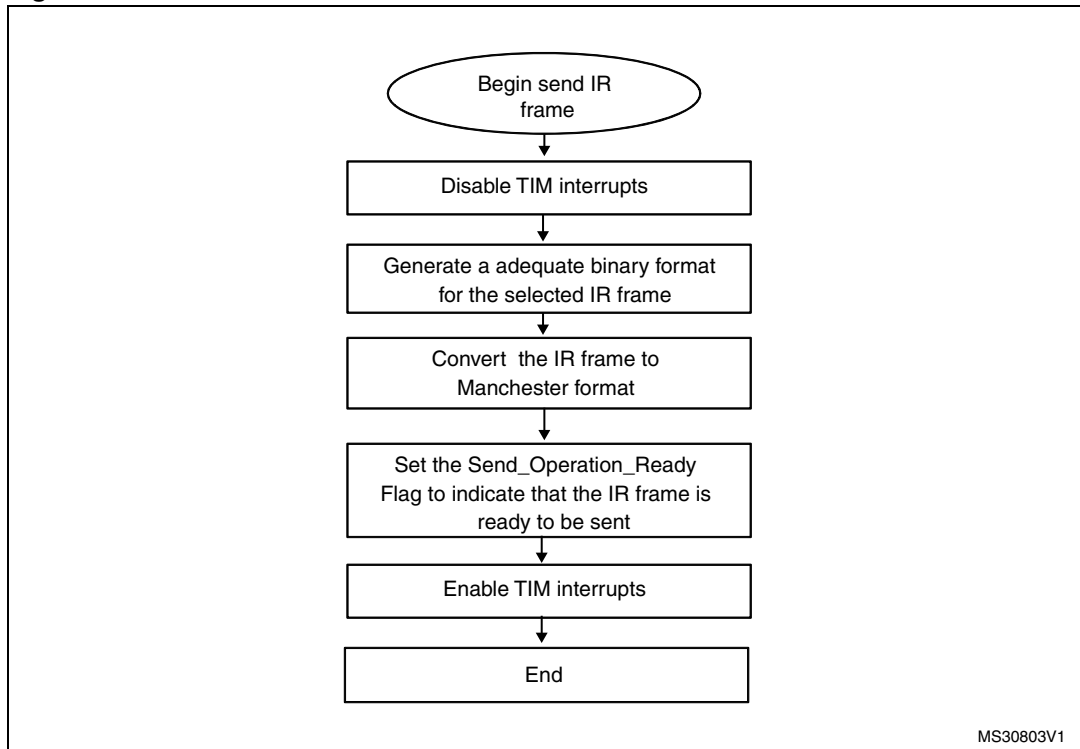
2.2.1 RC5 encoder solution

RC5 encoding mechanism

Figure 11 on page 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
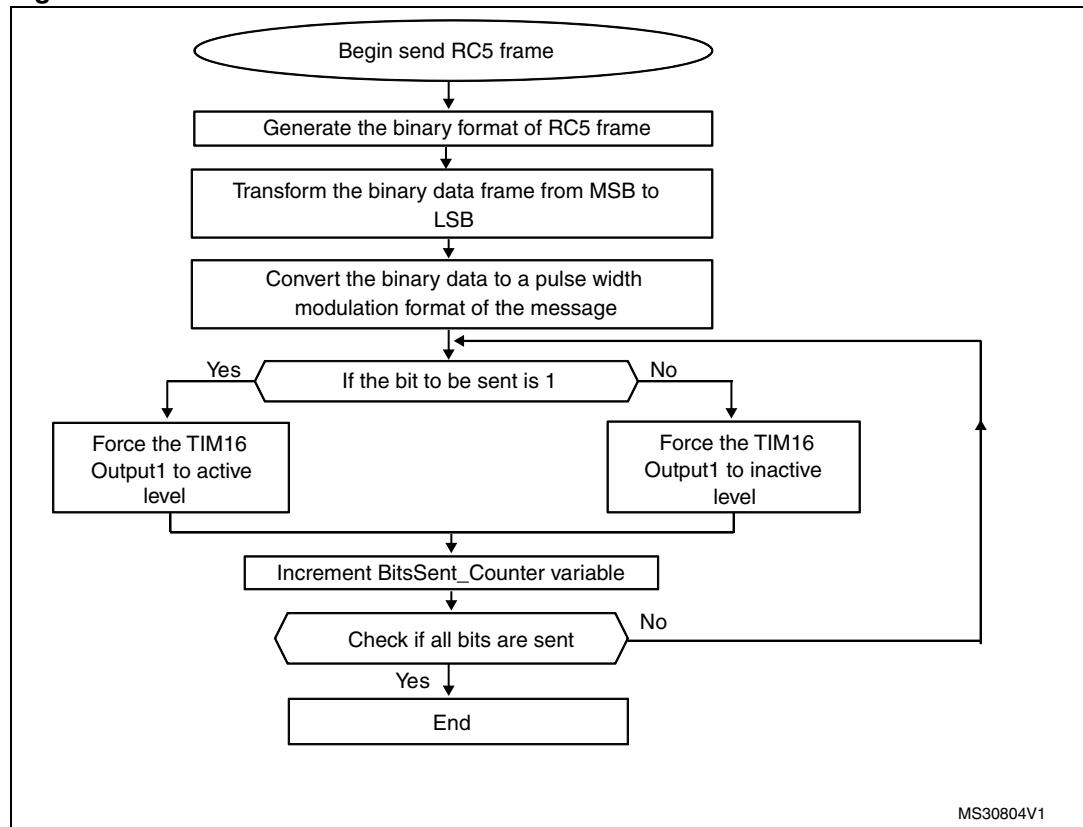
In the Manchester encoding, a logic "0" is indicated by a 0 to 1 transition at the centre of the bit and a logic "1" is indicated by a 1 to 0 transition at the centre of the bit. The Manchester encoding rules are summarized in *Figure 12*.

Figure 12. Manchester encoding bits
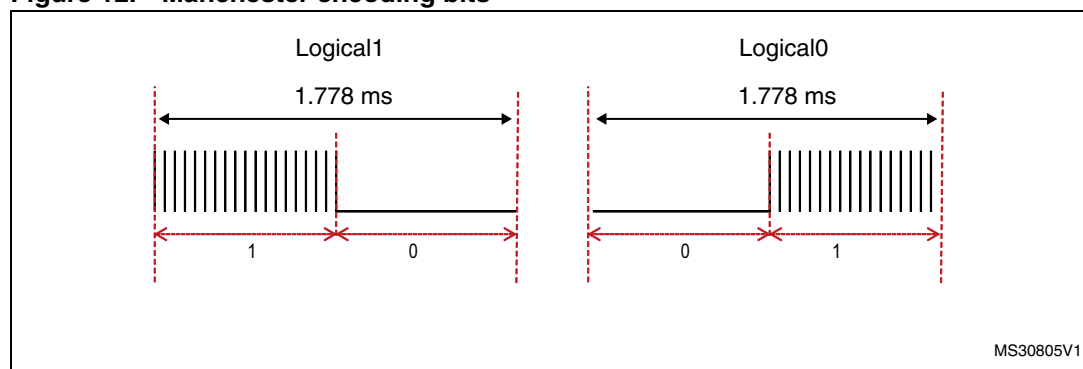
RC5 encoding library

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 TIM16. It is called during the TIM16 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.
– TIM16 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
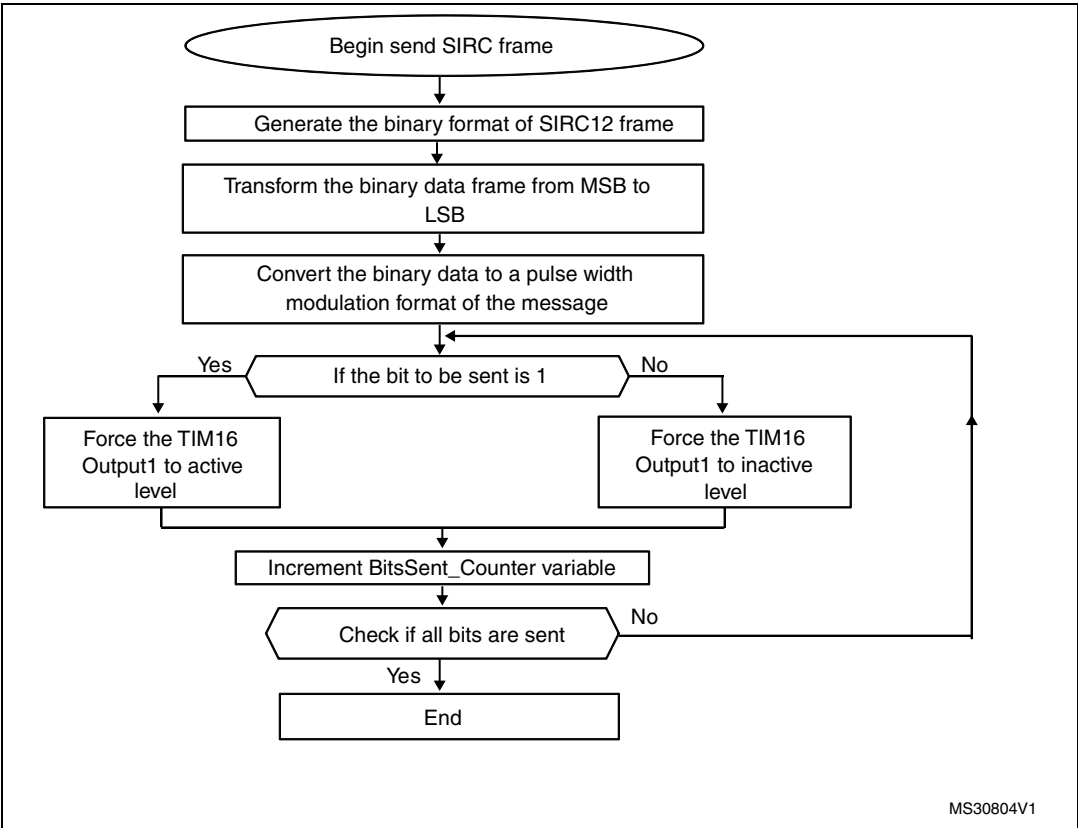
After generating the binary format frame, each logical bit is converted to a combination of "0" and "1" which represents a pulse width modulation format.

For example, a logical "1" takes 1.8 ms to transmit with 1.2 ms high level and 600 μs as low level. The chosen base time is 600 μs, so the logical "1" is converted to 110.

For a logical "0", it takes 1.2 ms with 600 μs as high level and 600 μs as low level. It is converted to 10 (refer to *Figure 14*).

Figure 14. SIRC logical bit conversion
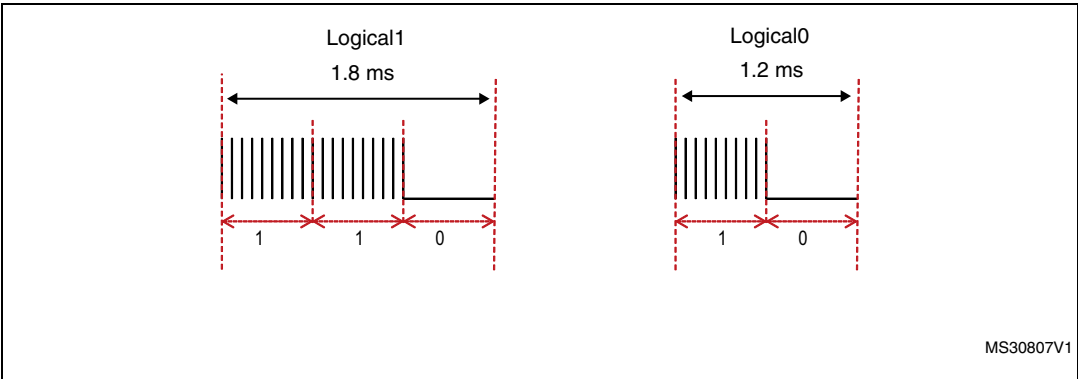
SIRC encoding library

The SIRC encoder driver is based on the following functions.

SIRC12_Encode_Init()

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

SIRC12_Encode_SendFrame()

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

SIRC12_Encode_SignalGenerate()

This function generates the frame signal by monitoring the output level of TIM16. It is called in the TIM16 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 SIRC12_Encode_Init() to configure the timer and GPIO hardware resources needed for SIRC encoding.

– Call the function SIRC12_Encode_SendFrame() to send the SIRC frame.

– TIM16 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) or TSOP34836 (3.3 V supply version) or other equivalent part number (refer to *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'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 15. Hardware configuration
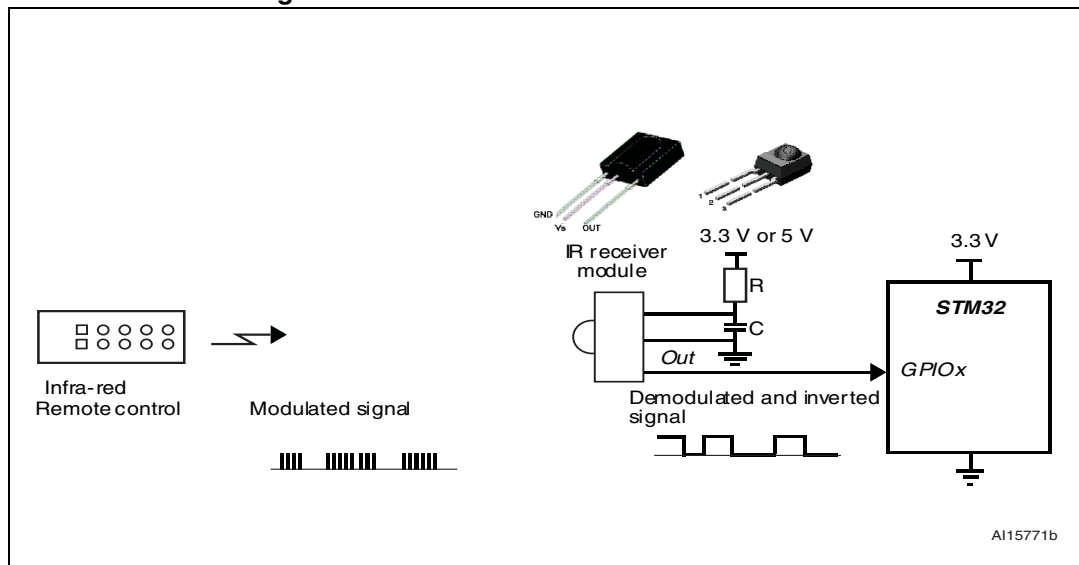
3.2 Universal solution: Software implementation using a GP-Timer configured in PWM input mode

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 types of interrupt.

● 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
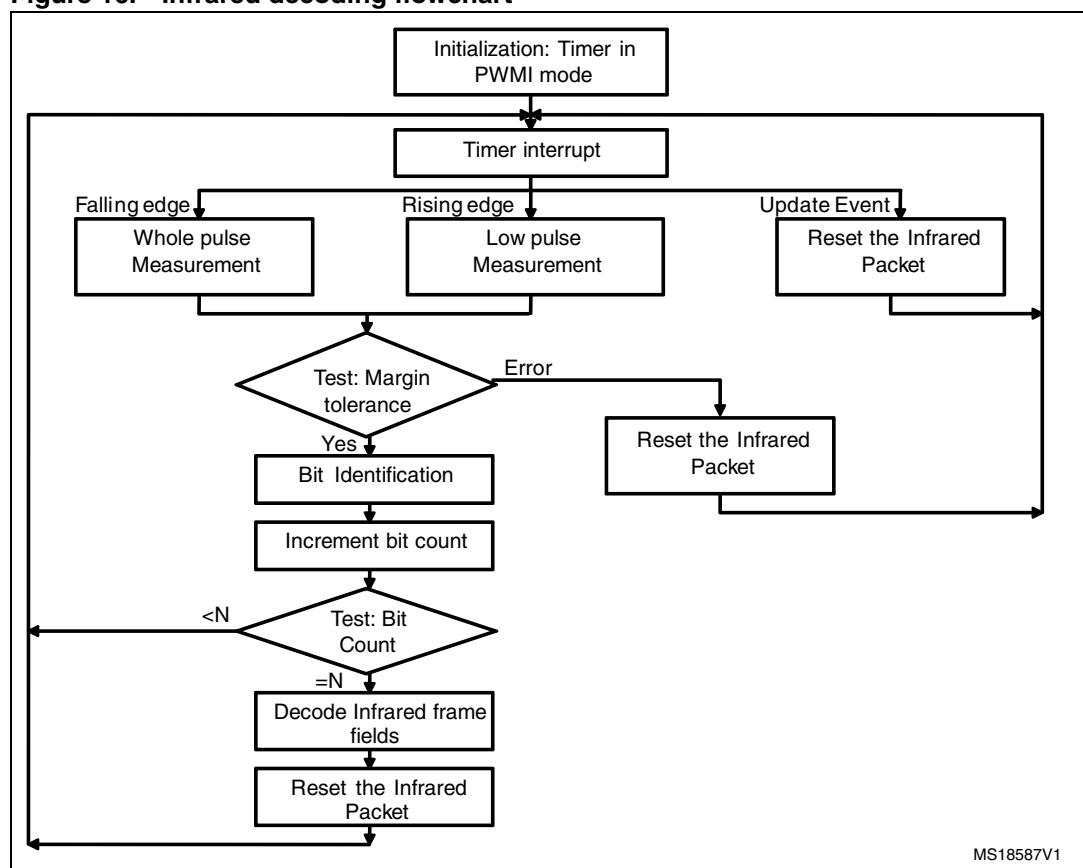
3.3 RC5 protocol solutions

3.3.1 RC5 frame decoding mechanism

Figure 17 shows how the RC5 frame is received. One of the 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 17. RC5 frame decoding mechanism
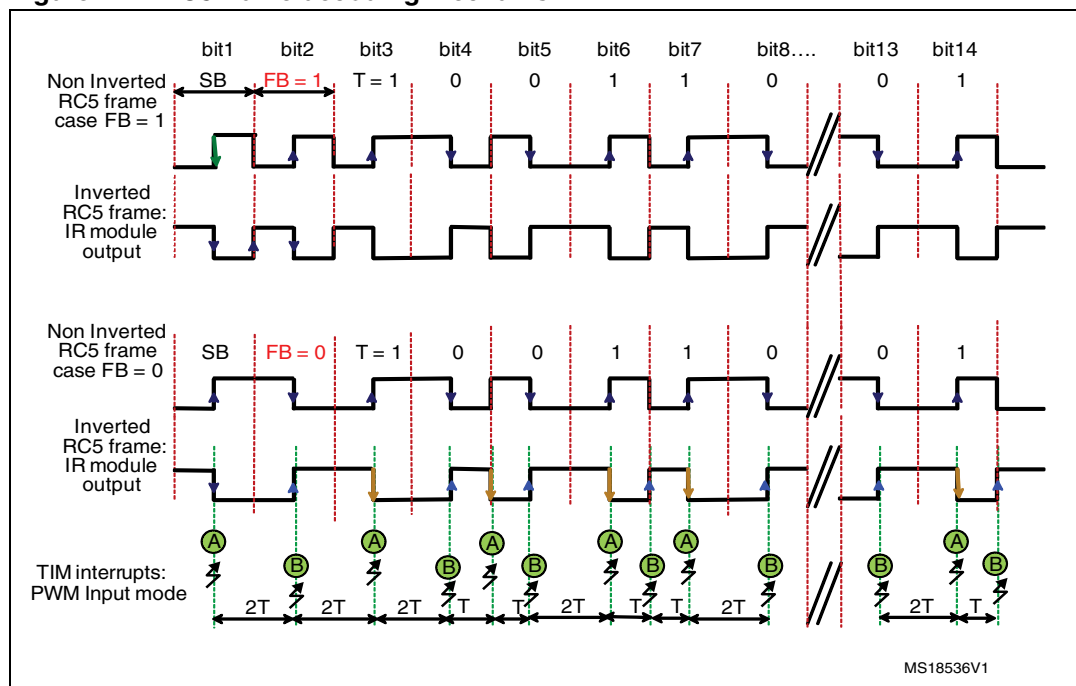
- **TIMER interrupt event:** falling edge

  A: the TIMER interrupt is used to measure the period between two successive falling edges (the entire 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
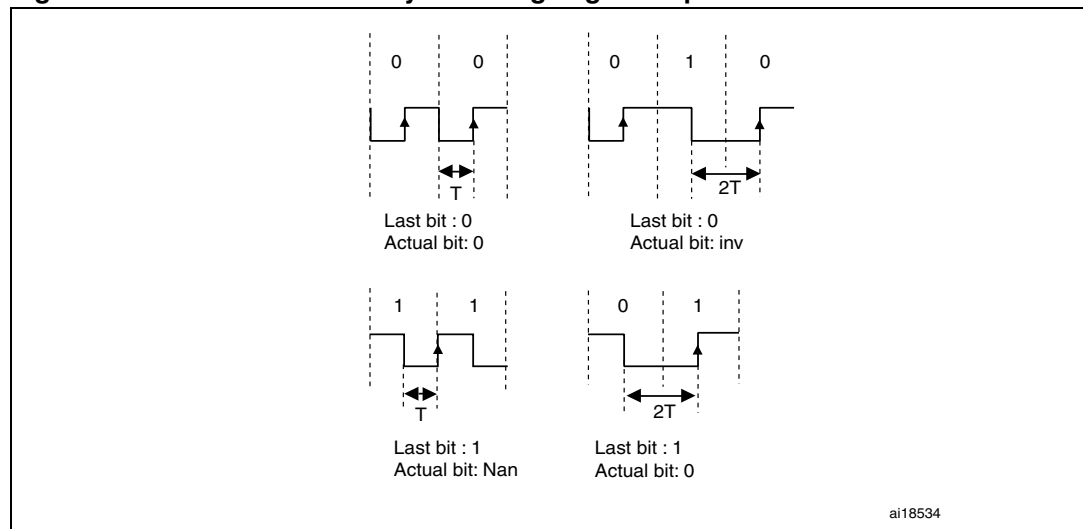
- If the low pulse duration is equal to T and the last bit determined is "0" => the actual bit is *logic0*.

- 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 19. Bit determination by the falling edge: high pulse
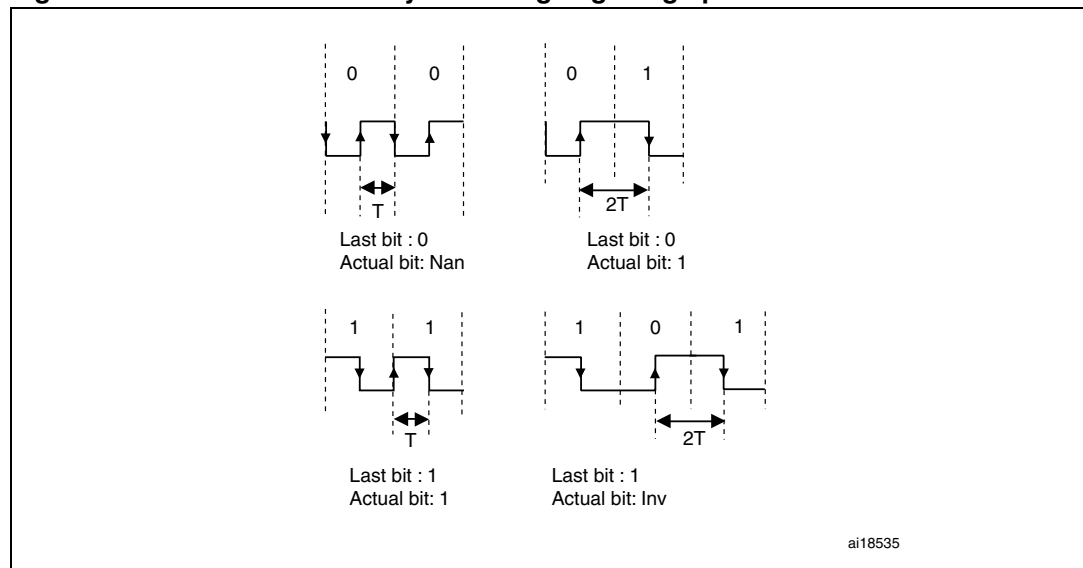
- If the high pulse duration is equal to T and the last bit determined is "0" => the actual bit is **Nan** (**no bit:** this bit is determined at the next rising edge).

- 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 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 TIM2_IRQHandler routine. It occurs each TIMER overflow to reset the RC5 packet.
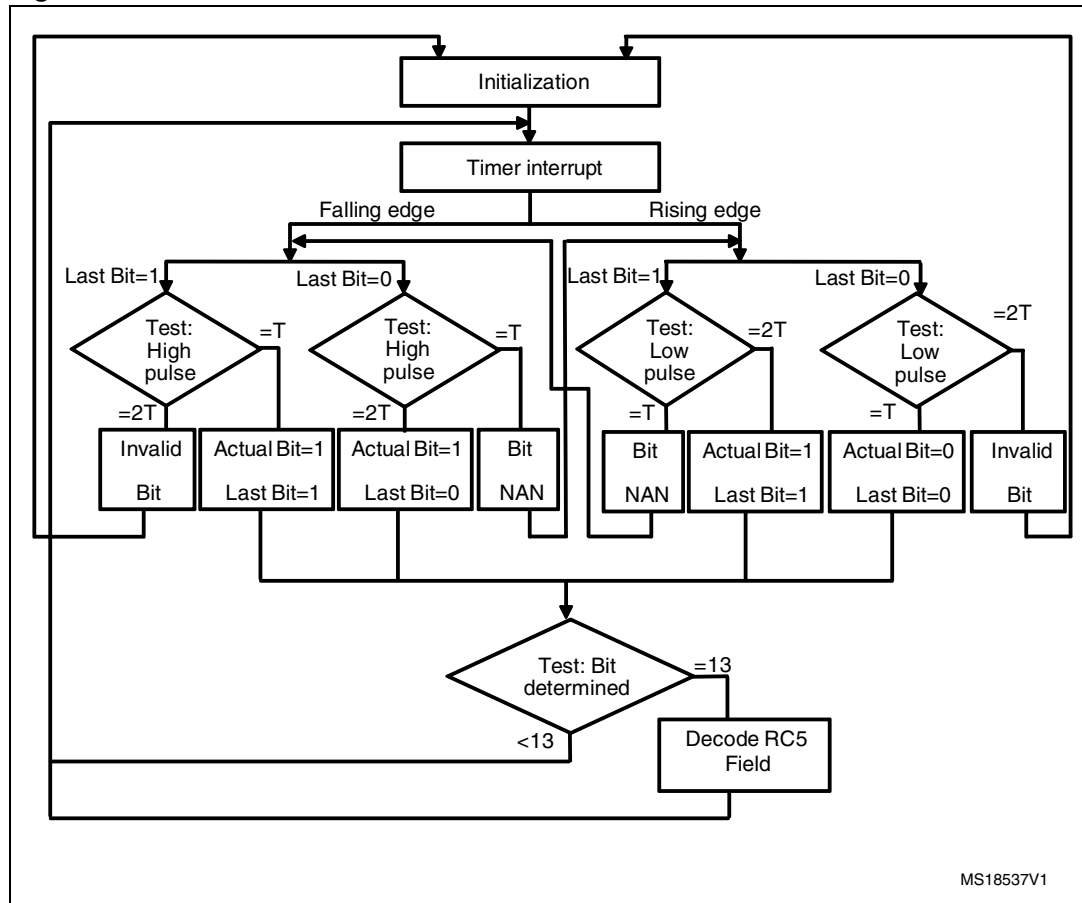
RC5_Decode(RC5_Frame_TypeDef *rc5_frame)

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 ;
```

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

RC5_DeInit()

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

TIM2_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 entire 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 3.7 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, 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

```c
#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 *TIM2_IRQHandler ISRs are coded within the stm32f0xx_it.c.*

● 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. Refer to the following Table.*

Table 3.   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	TIM2
#define TIM_PRESCALER	TIM prescaler	This parameter is computed so as to have 1 us as time base. TIM frequency (MHz) / (prescaler + 1)	47
#define IR_TIM_CLK	APB clock of the used timer	Refer to stm32f0xx_rcc.c/.h driver	RCC_APB1Periph_TIM2
#define IR_TIM_IRQn	IR TIM IRQ	Refer to stm32f0xx_misc.c/.h driver	TIM2_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]	GPIOB
#define IR_GPIO_PORT_CLK	IR pin GPIO clock port	Refer to RCC FW driver	RCC_APB2Periph_GPIOB
#define IR_GPIO_PIN	Pin which IR is connected	Refer to your product pinout[1]	GPIO_Pin_3

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

3.4 SIRC infrared control solution

3.4.1 Software implementation

Figure 21. SIRC frame reception mechanism

MS18555V1
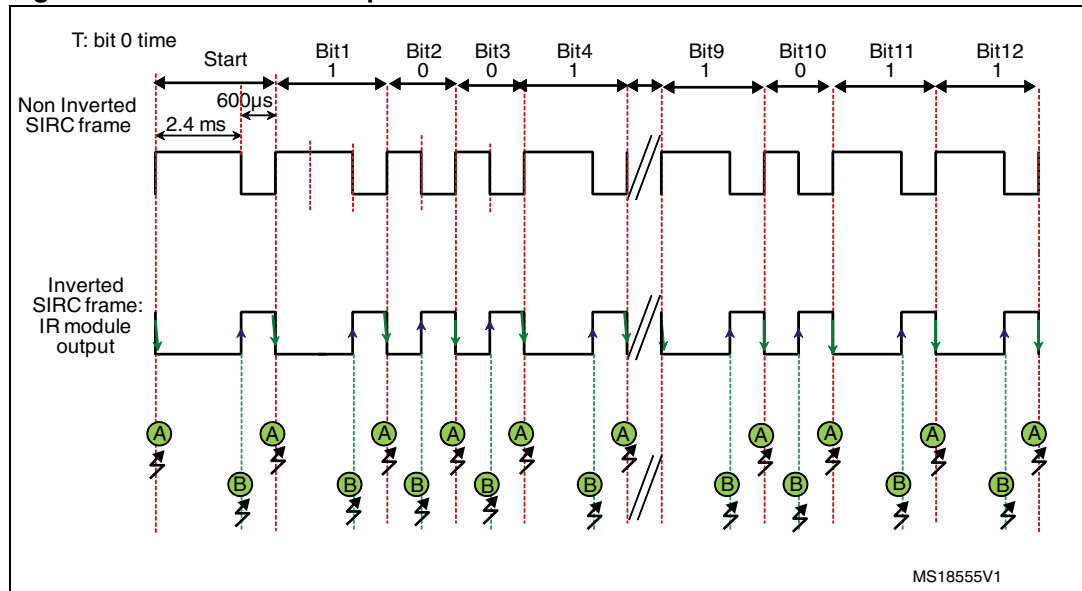
TIMER interrupt: in PWM input mode

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 22. SIRC solution flowchart

MS18556V1
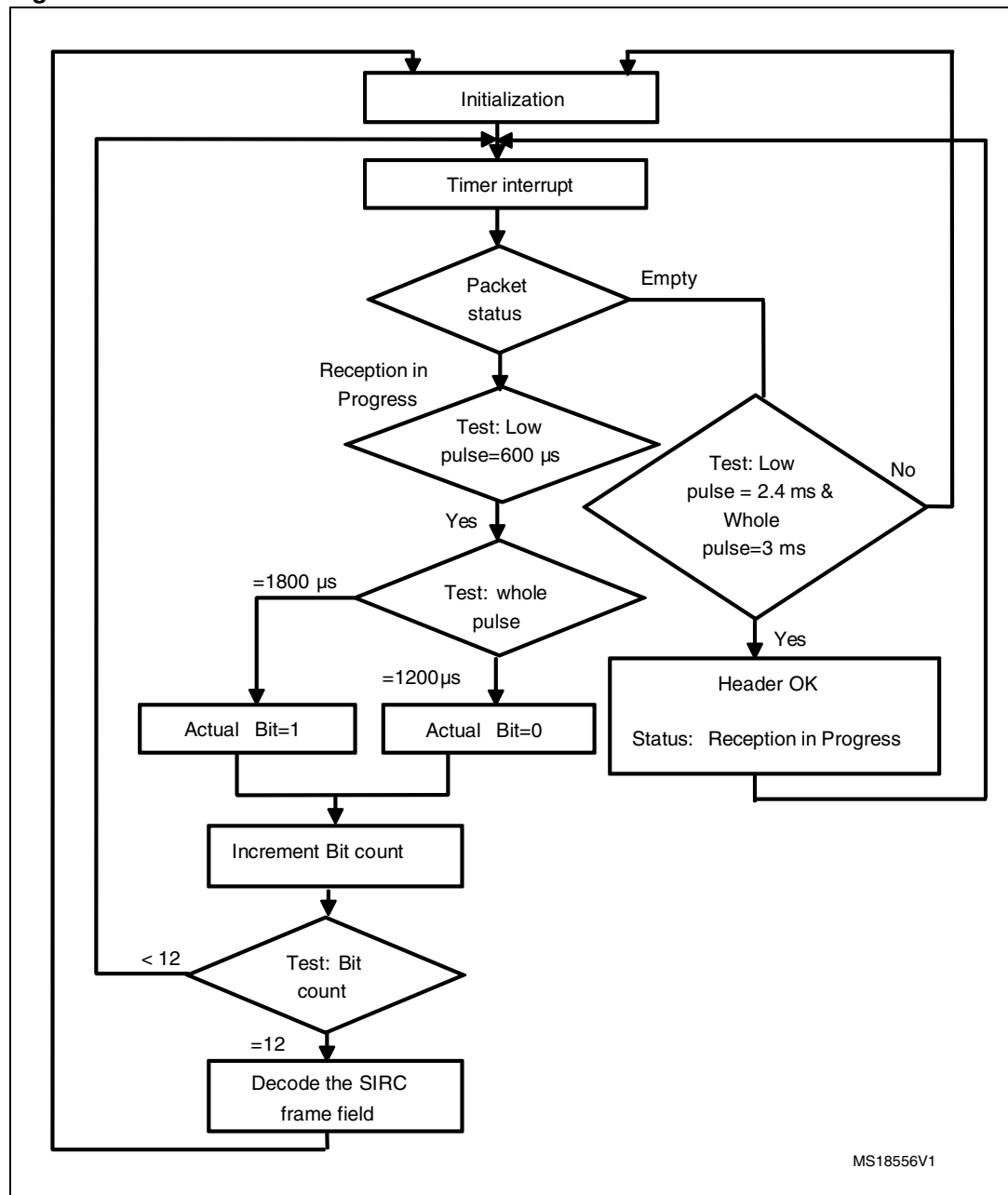
3.4.2 SIRC library

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 Address;          /* Address field    */
__IO uint8_t Command;          /* Command 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.

TIM2_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 (low pulse duration).
– **Update event (time-out event)**: this resets the RC5 packet. The timer overflow is set to 4 ms.

The bit value is determined from these two durations.

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.
– You 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*   *TIM2_IRQHandler ISRs are coded within the stm32f0xx_it.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 4.   **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	TIM2
#define TIM_PRESCALER	TIM prescaler	This parameter is computed so as to have 1 us as time base. TIM frequency (MHz) / (prescaler + 1)	47
#define IR_TIM_CLK	APB clock of the used timer	Refer to RCC FW driver	RCC_APB1Periph_TIM2
#define IR_TIM_IRQn	IR TIM IRQ	stm32f0xx_misc.c/.h driver	TIM2_IRQn
#define IR_TIM_Channel	IR TIM channel	TIM_Channel_1 or TIM_Channel_2	TIM_Channel_1

Define name	Description	Allowed values	Example
#define IR_GPIO_PORT	Port which IR output is connected	Refer to your product pinout[1]	GPIOB
#define IR_GPIO_PORT_CLK	IR pin GPIO clock port	Refer to stm32f0xx_rcc.c/.h driver	RCC_APB2Periph_GPIOB
#define IR_GPIO_PIN	Pin which IR is connected	Refer to your product pinout[1]	GPIO_Pin_3

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

4 Interface layer

Figure 23. Application layer architecture
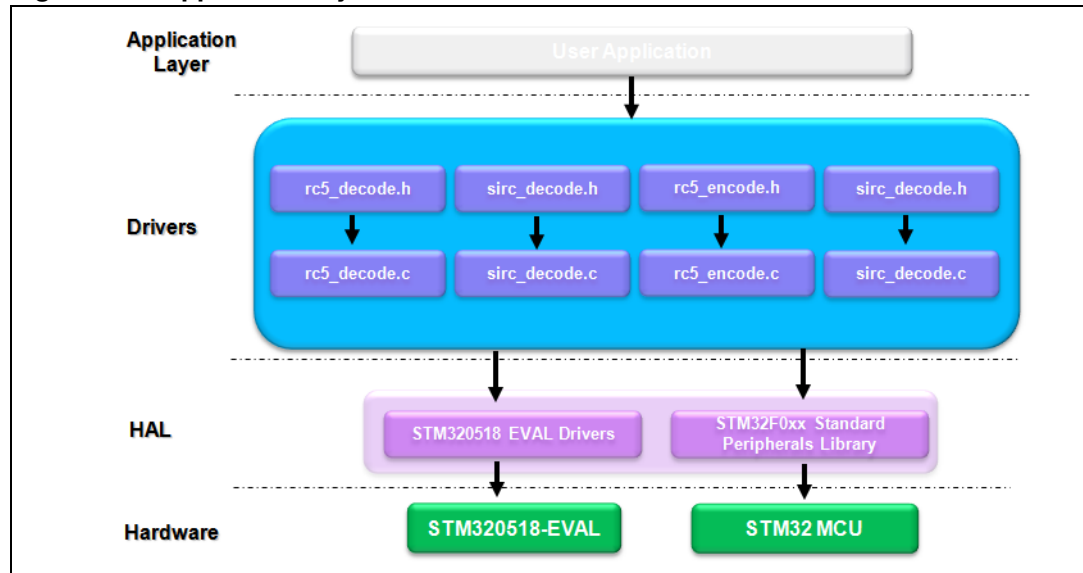
There are many similar infrared protocols that are differentiated by timing parameters such as the SIRC protocol. These protocols are handled by sirc_decode.c/sirc_encode.c functions. You only need to update the timing values.

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 within the STM320518-EVAL Demo Config2 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.

Figure 24. IR transmitter demo
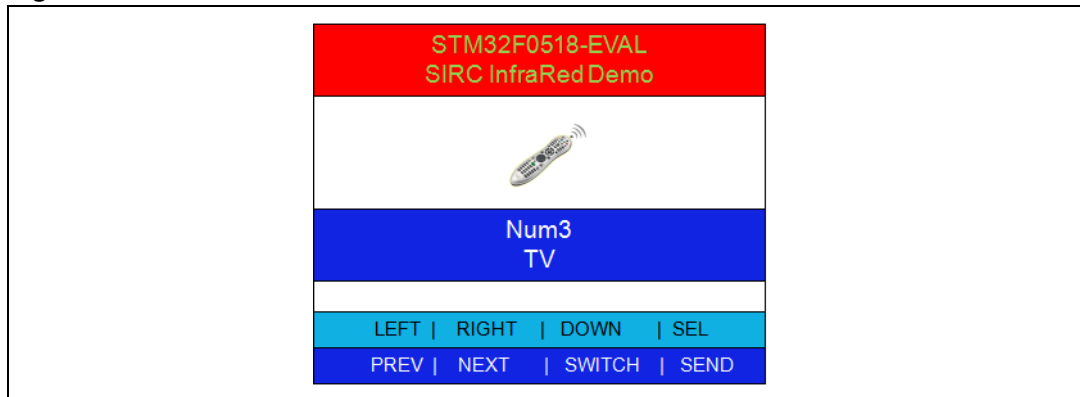
4.1.2 Receiver demonstration using GP-Timer configured in PWM mode
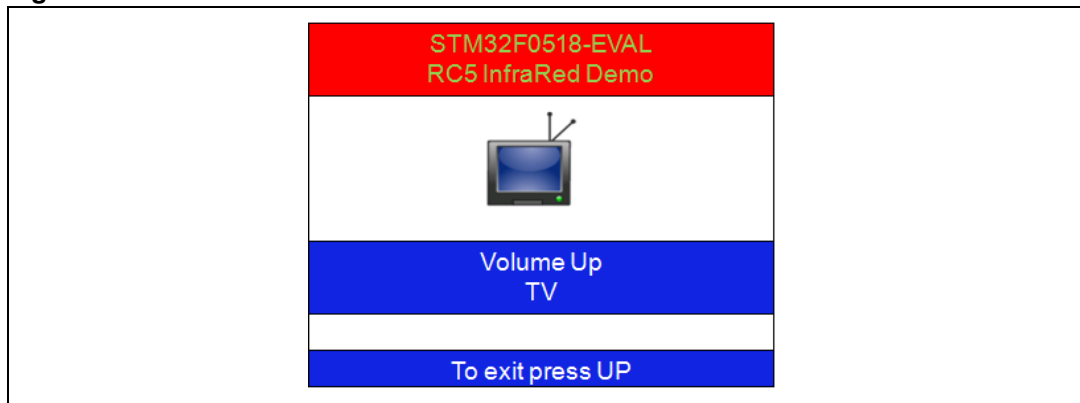
This demonstration consists in receiving IR messages and sending them to the LCD.

Each IR message is displayed in two parts.

● The device that transmitted the IR frame.

● The command to be executed.

The following figure shows the RC5 decoder using the PWMI method (refer to *Section 3.2 on page 15*).

Figure 25. RC5 received frame shown in the LCD

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, you must:

1. Add the header file of the appropriate IR protocol to your project.
   Example: rc5_decode.h.
2. Add the file.c corresponding to the IR protocol to your 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 .

Example:

```
void TIM2_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 */
    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);
        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);
    RC5_ResetPacket();
    }
}
```

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

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 5.   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	4050
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 - 60)
IR_ONTIME_MAX_US	Max low pulse in µs	(1200 + 60)
IR_HEADER_LOW_MIN_US	Min header low pulse in µs	(2400 - 150)
IR_HEADER_LOW_MAX_US	Max header low pulse in µs	(2400 + 150)
IR_HEADER_WHOLE_MIN_US	Min header whole duration in µs	(2400 + 600 - 60)
IR_HEADER_WHOLE_MAX_US	Max header whole duration in µs	(2400 + 600 + 60)
IR_VALUE_STEP_US	Step value between bit0 and bit1 in µs	600

Table 5.   List of defines in the header file for the IR protocol parameters

Defines	Meaning	Example settings for SIRC protocol
IR_VALUE_MARGIN_US	Margin in µs	100
IR_VALUE_00_US	Bit0 duration in µs	1200

Note: *IR refers to the name of the IR protocol. For example, SIRC_HEADER_LOW_MIN_US.*

–   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 you must:

1.   Add the header file of the appropriate IR protocol to your project.
    Example: rc5_encode.h.
2.   Add the file.c corresponding to the IR protocol to your 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 .

Example:

```
void TIM16_IRQHandler(void)
{
   RC5_Encode_SignalGenerate(RC5_FrameManchestarFormat);

  /* Clear TIM16 update interrupt */
  TIM_ClearITPendingBit(TIM16, TIM_IT_Update);
}
```

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 implementing an IR transmitter/receiver in software using a general-purpose timer.

The IR encoding application uses the STM32F0xx microcontroller and contains a powerful 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 of the various audiovisual products in a given environment.

6 Revision history

Table 6.   Document revision history

Date	Revision	Changes
02-May-2012	1	Initial release

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