User Guide
Commande climatiseur

RmDvTelecoIR.c/.h

Table des matières

1. Présentation du système

Cette partie « commande climatiseur » permet d'activer le climatiseur en plusieurs modes ainsi que de le stopper. Les trames émises en IR sont modulées à 38kHz en OOK. Les codes ont été hackés avec la télécommande d'origine en venant lire la suite de 0 et de 1 de l'enveloppe OOK.

La trame est longue, environ 115 bits. Chaque bit 0/1 est en fait codé sous la forme d'une impulsion :

La trame commence par un '1' long et un '0" long :

T correspond au timing le plus court, T # 430µs.

Si on considère qu'un bit correspond à cette durée T, on observera un « 1 » suivi d'une « 0 » ou bien un « 1 » suivi de deux « 0 ». Nous obtenons ainsi des trames de 37 bits. Voici un exemple :
FF 08 8A A2 A2 28 A8 8A 22 A2 A2 AA AA AA AA 8A 8A 8A AA A2 88 AA AA 28 AA AA AA AA AA AA AA AA A8 88 8A 22 80

La partie en gras correspond à un préambule commun à toutes les trames (19 octets). Un code complet sera donc découpé en deux :
- `Tab_CommonCode_Temp[19],`
- `TabCode[6][18],`

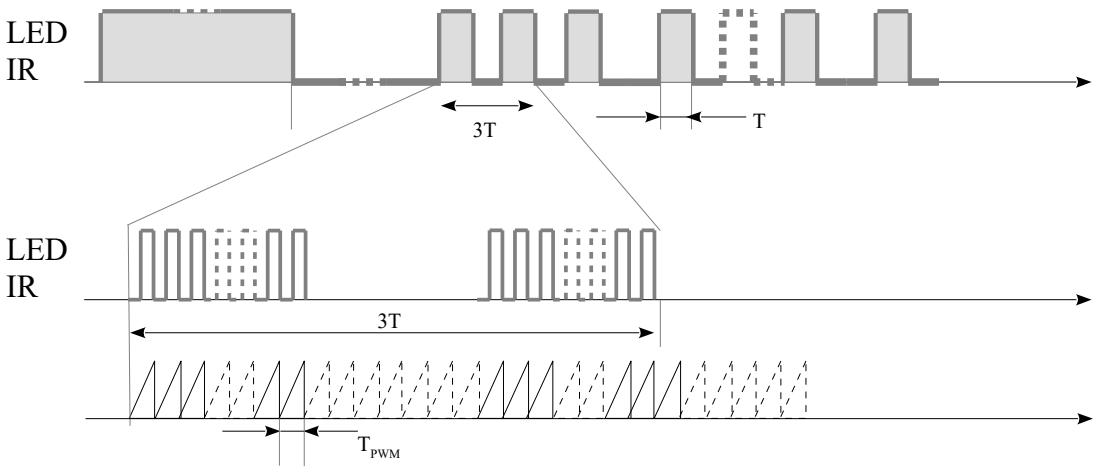
Cette seconde partie est découpée en 6 éléments de 18 octets. Cela correspond donc à 6 codes différents que l'on choisit pour s'enchaîner derrière le code commun. Dans ce cas, 6 codes sont possibles, on peut en ajouter d'autres.
Le code Stop est quand à lui original, et ne possède pas la partie commune :

```
TabOFF[36]=
{0xFF, 0x08, 0x8A, 0xA2, 0xA2, 0x28, 0xA8, 0x8A, 0x22, 0xA2, 0xA2, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0x2A, 0x22,
0xAA, 0xA8, 0x8A, 0xAA, 0x28, 0x8A, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xA8, 0x8A, 0x2A, 0x28};
```

2. Mise en œuvre

2.1. Elaboration du signal de sortie

Chaque bit est lu à une cadence de 430µs grâce à un timer en interruption. Lors de chaque décision (1 ou 0), on active ou non le canal d'un second timer qui aura été configuré en PWM 50% à 38kHz. En réalité, les deux timers sont un seul et unique qui gère tout, la PWM et l'interruption. Le schéma ci-dessous montre le principe :



T = 430µs
T_{PWM} = 1/38kHz ≈ 26.316 µs

Le timer, cadencé à 24MHz, nécessite 632 impulsions. Ce qui donne en réalité, 37.975 Khz (tout à fait correct).
Le nombre de périodes PWM, N, contenu dans les 430µs est donc 430µ/(632/24MHz) = 430*24/632 = 16,3. Si nous prenons 631 impulsions (et non 632), le résultat est sensiblement le même.

→ on va préférer viser 430µs et arrondir les 38kHz (filtre passe bande un peu large probable). Dans ces conditions, 430µs implique N=430*24 = 10320 précisément à 24MHz. La fréquence qui correspond est 1/430µs = 2.325581 kHz. Le ratio à 38kHz est donc 16.34 précisément. En prenant 16, la fréquence réelle obtenue pour la PWM sera de 37.2 kHz. Le facteur de qualité correspondant est 38/1.6 = 23. Il se peut que l'erreur sur le 38kHz implique une perte de portée à cause du passe bande de la clim.

→ Au final, on va utiliser deux timers.
L'un pour les 38kHz de PWM (N = 632, cad 37.97kHz), l'autre pour générer les 430µs (N=10392)

2.2. Algorithme

Deux fonctions sont utilisées :
`void RmDv_TelecoIR_Init(void)` : lance les initialisations nécessaires, à savoir, configurer les deux timers nécessaires.

`void RmDv_TelecoIR_SetCmde(RmDv_TelecoIR_Cmde Cmde)` : émet le code voulu. Le type RmDv_TelecoIR_Cmde est une énumération que l'on peut faire grossir au fur et à mesure. A cette fin, se reporter à la partie hacking (/HackingCodeClim) pour capter le code d'une télécommande.

2.2.1. Aspect temps réel

La fonction *RMDv_TelecoIR_SetCmde* est basée sur une boucle *while* qui s'arrête lorsque tous les octets à émettre son passés. Cette boucle attend systématiquement au début un flag qui indique que les 430µs sont passées. Ce flag est mis à '1' dans l'interruption associée au timer qui gère le « temps bit ». C'est tout ce qu'elle fait, elle est donc non bloquante.

2.2.2. Principe algorithmique

Extrait de code pour la trame stop (émission d'un coup, alors que les autres commandes se font en deux temps, voir code) :

```
while(RmDv_TelecoIR_IndexOctet<RmDv_TelecoIR_TabOffLen)
{
            while(RmDv_TelecoIR_BitStart==0); /* Attente 430µs, début du bit, tps réel*/
            RmDv_TelecoIR_BitStart=0;
            RmDv_TelecoIR_CurrentByte=RmDv_TelecoIR_TabOFF[RmDv_TelecoIR_IndexOctet];
            if (RmDv_TelecoIR_CurrentByte&RmDv_TelecoIR_IndexBit)==RmDv_TelecoIR_IndexBit)
            {
                        Timer_SetOutputMode(RmDv_TelecoIR_Timer_PWM,PWM);
            }
            else Timer_SetOutputMode(RmDv_TelecoIR_Timer_PWM,Clear);
            RmDv_TelecoIR_BitByteIncOverflow();
}
```

L'idée ici est de modifier le mode PWM. Si le bit vaut '1', on se place en PWM, sinon on se place en sortie forcée à '0'.

2.3. Les réglages .h

```
#define RmDv_TelecoIR_Timer_PWM TIM2
#define RmDv_TelecoIR_Timer_Bit TIM21
#define RmDv_TelecoIR_Prio_IT_Bit 0
```

TIM2 est obligatoire car la LED est connectée sur PA1 (canal 2 du Timer 2).

2.4. Diagramme de classes