.cpu arm7tdmi

.eabi\_attribute 20, 1

.eabi\_attribute 21, 1

.eabi\_attribute 23, 3

.eabi\_attribute 24, 1

.eabi\_attribute 25, 1

.eabi\_attribute 26, 1

.eabi\_attribute 30, 6

.eabi\_attribute 34, 0

.eabi\_attribute 18, 4

.file "regularizador\_main.c"

;declaracion de 5 variables globales booleanas

.global flag\_T1

.bss

.type flag\_T1, %object

.size flag\_T1, 1 ;tamaño de flag\_T1 es 1 byte

flag\_T1:

.space 1

.global flag\_T2

.type flag\_T2, %object

.size flag\_T2, 1 ;tamaño de flag\_T2 es 1 byte

flag\_T2:

.space 1

.global flag\_T3

.type flag\_T3, %object

.size flag\_T3, 1 ;tamaño de flag\_T3 es 1 byte

flag\_T3:

.space 1

.global flag\_T4

.type flag\_T4, %object

.size flag\_T4, 1 ;tamaño de flag\_T4 es 1 byte

flag\_T4:

.space 1

.global TEC\_flag

.type TEC\_flag, %object

.size TEC\_flag, 1 ;tamaño de TEC\_flag es 1 byte

TEC\_flag:

;declaracion de 5 funciones para el manejo de handlers de las interrupciones

.space 1

.text

.align 2

.global GPIO0\_IRQHandler

.syntax unified

.arm

.fpu softvfp

.type GPIO0\_IRQHandler, %function

GPIO0\_IRQHandler:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 0

@ frame\_needed = 1, uses\_anonymous\_args = 0

push {fp, lr} ;guarda el “frame pointer” y el “Link register” en el stack

add fp, sp, #4 ;le suma a stack pointer 4 y lo vuelca en el frame pointer

ldr r2, .L2 ;cargo la dirección L2 en el registro r2

ldr r3, .L2 ;cargo la dirección L2 en el registro r3

;en .L2 esta la word PIN\_INT

ldr r3, [r3, #36] ;le agrega 36 al registro r3

;esto lo hace para moverse por la estructura hasta IST

;se mueve 8 int ⇒ 8\* (4 bytes) = 36

orr r3, r3, #1 ;operación or entre r3 y 1 que se cargará en r3

str r3, [r2, #36] ;guarda el resultado de la operacion r3 en r2+36

;recordando que r2 tiene .L2, es decir, PIN\_INT

mov r0, #2 ;copia un 2 en r0, ya que L0B==2

bl LED\_toggle ;hace branch with link para usar la función LED\_toggle

ldr r3, .L2+4 ;ahora cargo la dirección L2+4 en r3

; .L2+4 es la segunda .word, es flag\_T1

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0 ;compara r3 contra 0 y segun el resultado modifica un flag

movne r3, #1 ;si el flag del cmp es “negative” ⇒ carga un 1 en r3

moveq r3, #0 ;si el flag del cmp es “equal”⇒ carga un 0 en r3

and r3, r3, #255 ;corresponde a la “norma de C para negación de un bool”

eor r3, r3, #1 ;corresponde a la “norma de C para negación de un bool”

and r3, r3, #255 ;corresponde a la “norma de C para negación de un bool”

and r3, r3, #1 ;corresponde a la “norma de C para negación de un bool”

and r2, r3, #255 ;el resultado de el toggle en r2 (usa and para hacer un ld)

ldr r3, .L2+4 ;ahora cargo la dirección L2+4 en r3

strb r2, [r3] ;guarda r2 (el resultado) en r3, es decir flag\_T1

ldr r3, .L2+8 ;ahora cargo la dirección L2+8 en r3

;L2+8 es TEC\_flag

mov r2, #1 ;un 1 en r2

strb r2, [r3] ;guarda lo que hay en r2 (un 1) en r3 (TEC\_flag)

nop

sub sp, fp, #4 ; resta 4 a frame pointer y lo guarda en stack pointer

@ sp needed

pop {fp, lr} ;saca el frame pointer y el lr del stack

bx lr ;??

.L3:

.align 2

.L2:

.word 1074294784 ;es la direccion de memoria de PIN\_INT

.word flag\_T1

.word TEC\_flag

.size GPIO0\_IRQHandler, .-GPIO0\_IRQHandler

.align 2

.global GPIO1\_IRQHandler

.syntax unified

.arm

.fpu softvfp

.type GPIO1\_IRQHandler, %function

GPIO1\_IRQHandler:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 0

@ frame\_needed = 1, uses\_anonymous\_args = 0

push {fp, lr} ;guarda el “frame pointer” y el “Link register” en el stack

add fp, sp, #4 ;le suma a stack pointer 4 y lo vuelca en el frame pointer

;suma 4 porque es lo que ocupan las variables globales

ldr r2, .L5 ;cargo la dirección L5 en el registro r2

ldr r3, .L5 ;cargo la dirección L5 en el registro r3

;en .L5 esta la word PIN\_INT

ldr r3, [r3, #36] ;idem GPIO0\_IRQHandler

orr r3, r3, #2 ;idem GPIO0\_IRQHandler pero pone un 2

str r3, [r2, #36] ;idem GPIO0\_IRQHandler

mov r0, #3

bl LED\_toggle

ldr r3, .L5+4

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

movne r3, #1

moveq r3, #0

and r3, r3, #255

eor r3, r3, #1

and r3, r3, #255

and r3, r3, #1

and r2, r3, #255

ldr r3, .L5+4

strb r2, [r3]

ldr r3, .L5+8

mov r2, #1

strb r2, [r3]

nop

sub sp, fp, #4

@ sp needed

pop {fp, lr}

bx lr

.L6:

.align 2

.L5:

.word 1074294784

.word flag\_T2

.word TEC\_flag

.size GPIO1\_IRQHandler, .-GPIO1\_IRQHandler

.align 2

.global GPIO2\_IRQHandler

.syntax unified

.arm

.fpu softvfp

.type GPIO2\_IRQHandler, %function

GPIO2\_IRQHandler:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 0

@ frame\_needed = 1, uses\_anonymous\_args = 0

push {fp, lr}

add fp, sp, #4

ldr r2, .L8

ldr r3, .L8

ldr r3, [r3, #36]

orr r3, r3, #4

str r3, [r2, #36]

mov r0, #4

bl LED\_toggle

ldr r3, .L8+4

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

movne r3, #1

moveq r3, #0

and r3, r3, #255

eor r3, r3, #1

and r3, r3, #255

and r3, r3, #1

and r2, r3, #255

ldr r3, .L8+4

strb r2, [r3]

ldr r3, .L8+8

mov r2, #1

strb r2, [r3]

nop

sub sp, fp, #4

@ sp needed

pop {fp, lr}

bx lr

.L9:

.align 2

.L8:

.word 1074294784

.word flag\_T3

.word TEC\_flag

.size GPIO2\_IRQHandler, .-GPIO2\_IRQHandler

.align 2

.global GPIO3\_IRQHandler

.syntax unified

.arm

.fpu softvfp

.type GPIO3\_IRQHandler, %function

GPIO3\_IRQHandler:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 0

@ frame\_needed = 1, uses\_anonymous\_args = 0

push {fp, lr}

add fp, sp, #4

ldr r2, .L11

ldr r3, .L11

ldr r3, [r3, #36]

orr r3, r3, #8

str r3, [r2, #36]

mov r0, #5

bl LED\_toggle

ldr r3, .L11+4

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0

movne r3, #1

moveq r3, #0

and r3, r3, #255

eor r3, r3, #1

and r3, r3, #255

and r3, r3, #1

and r2, r3, #255

ldr r3, .L11+4

strb r2, [r3]

ldr r3, .L11+8

mov r2, #1

strb r2, [r3]

nop

sub sp, fp, #4

@ sp needed

pop {fp, lr}

bx lr

.L12:

.align 2

.L11:

.word 1074294784

.word flag\_T4

.word TEC\_flag

.size GPIO3\_IRQHandler, .-GPIO3\_IRQHandler

.align 2

.global DMA\_IRQHandler

.syntax unified

.arm

.fpu softvfp

.type DMA\_IRQHandler, %function

DMA\_IRQHandler:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 0

@ frame\_needed = 1, uses\_anonymous\_args = 0

@ link register save eliminated.

str fp, [sp, #-4]! ; es lo mismo que hacer el push pero el optimizador lo hace asi

add fp, sp, #0

ldr r2, .L14

ldr r3, .L14 ;cargo la dirección L14 en el registro r3

;en .L5 esta la word GPDMA

ldr r3, [r3, #8] ;le agrega 8 al registro r3

;para moverse por la estructura hasta INTTCCLEAR

;se mueve 8 int ⇒ 2\* (4 bytes) = 8

orr r3, r3, #1 ;OR entre r3 y 1, luego lo carga en r3

str r3, [r2, #8] ;store de r3 (resultado) en r2+8 (GPDMA->INTTCCLEAR)

nop

add sp, fp, #0

@ sp needed

ldr fp, [sp], #4

bx lr

.L15:

.align 2

.L14:

.word 1073750016

.size DMA\_IRQHandler, .-DMA\_IRQHandler

.align 2

;codigo del main

.global main

.syntax unified

.arm

.fpu softvfp

.type main, %function

main:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 672

@ frame\_needed = 1, uses\_anonymous\_args = 0

push {fp, lr}

add fp, sp, #4

sub sp, sp, #672

bl TEC\_ALL ;hace branch and link a la funcion TEC\_ALL

bl LED\_ALL ;hace branch and link a la funcion LED\_ALL

mov r0, #0 ;pone un 0 en r0 para luego llamar a la funcion con parametro

bl Enable\_PIN\_INT ;hace branch and link a la funcion

mov r0, #1

bl Enable\_PIN\_INT

mov r0, #2

bl Enable\_PIN\_INT

mov r0, #3

bl Enable\_PIN\_INT

mov r2, #0 ;pone un 0,4,0 en r0,r1,r2 para poder hacer branch and link de

mov r1, #4 ;la funcion con 3 parametros

mov r0, #0

bl Select\_GPIO\_interrupt

mov r2, #1

mov r1, #8

mov r0, #0

bl Select\_GPIO\_interrupt

mov r2, #2

mov r1, #9

mov r0, #0

bl Select\_GPIO\_interrupt

mov r2, #3

mov r1, #9

mov r0, #1

bl Select\_GPIO\_interrupt

mov r0, #32 ;pone un 32 en r0 que corresponde a la interrupcion PIN\_INT0\_IRQn

bl Enable\_NVIC

mov r0, #33

bl Enable\_NVIC

mov r0, #34

bl Enable\_NVIC

mov r0, #35

bl Enable\_NVIC

sub r3, fp, #644 ;busca la direccion de src\_signal\_1 y lo carga en r3

sub r2, fp, #324 ;busca la direccion de src\_signal\_0 y lo carga en r2

sub r1, fp, #676 ;busca la direccion de LLI1 y lo carga en r1

sub r0, fp, #660 ;busca la direccion de LLI0 y lo carga en r0

bl cfg\_DMA ;branch and link de la funcion con los parametros cargados

mov r0, #2 ;pone un 2 en r0 que corresponde a DMA\_IRQn

bl Enable\_NVIC

.L18:

ldr r3, .L19 ;carga TEC\_flag

ldrb r3, [r3] @ zero\_extendqisi2

cmp r3, #0 ;compara r3 contra 0

;notar que en el if del código c el flag se compara contra 1

;pero es mas optimo comparar contra 0 e invertir la lógica

beq .L18 ;branch equal → si el resultado de la comparación dio “equal”

;entonces voy a .L18, es decir loopeo

sub r2, fp, #644 ;carga la direccion de src\_signal\_1

sub r3, fp, #324 ;carga la direccion de src\_signal\_0

mov r1, r2 ;copia r2 en r1 => r1= src\_signal\_1

mov r0, r3 ;copia r3 en r0 => r0=src\_signal\_0

bl signal\_fill ;branch and link con r0 y r1.

ldr r3, .L19 ;carga TEC\_flag

mov r2, #0 ;pone un 0 en r2

strb r2, [r3] ;guarda un 0 en r3 (la direccion de TEC\_flag)

b .L18 ;loop del while.

;no hace falta la comparación porque está optimizado

; y siempre va a dar falso

.L20:

.align 2

.L19:

.word TEC\_flag

.size main, .-main

.ident "GCC: (15:6.3.1+svn253039-1build1) 6.3.1 20170620"