.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 "dac.c"

;declaracion de variable global “SAMPLES\_NUM”

; es un int ⇒ size =4

.global SAMPLES\_NUM

.data

.align 2

.type SAMPLES\_NUM, %object

.size SAMPLES\_NUM, 4

SAMPLES\_NUM:

.word 80

.text

.align 2

;declaracion de funcion “Enable\_ENAIO”

;en C → void Enable\_ENAIO(){ SCU->ENAIO)[2]|=1; }

.global Enable\_ENAIO

.syntax unified

.arm

.fpu softvfp

.type Enable\_ENAIO, %function

Enable\_ENAIO:

@ Function supports interworking.

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

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

@ link register save eliminated.

str fp, [sp, #-4]!

add fp, sp, #0

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

;1074290688 ⇒ SCU

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

;1074290688 ⇒ SCU

ldr r3, [r3, #3216] ;cargo (r3 + 3216) en r3

;r3 → .L2 → 1074290688 → 40086000⇒ SCU

;1074290688+ 3216 = 4008 6C90

;3216 / 4=804 byes (es dividido 4 por ser unsigned int)

; 804 = SFSP[16][32] (512) + RESERVED0[256] (256) +

;+SFSCLK[4] (4) + SFSCLK[4] + RESERVED16[28] +

;SFSUSB

;+ SFSI2C0 (1) + ENAIO[1] (2)

;==> cargo SCU->ENAIO[2] en r3

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

str r3, [r2, #3216] ;idem usando r2 (tiene .L2) ==> cargo SCU->ENAIO[2] en r3

nop

add sp, fp, #0

@ sp needed

ldr fp, [sp], #4

bx lr

.L3:

.align 2

.L2:

.word 1074290688

.size Enable\_ENAIO, .-Enable\_ENAIO

;declaracion de funcion “Enable\_DMA”

;en c : void Values\_DAC(int value){ (DAC->CR)= ((value & 0x3FF)<< 6) | (1<<16); }

.align 2

.global Enable\_DMA

.syntax unified

.arm

.fpu softvfp

.type Enable\_DMA, %function

Enable\_DMA:

@ Function supports interworking.

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

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

@ link register save eliminated.

str fp, [sp, #-4]!

add fp, sp, #0

ldr r2, .L5

ldr r3, .L5

ldr r3, [r3, #4]

orr r3, r3, #15

str r3, [r2, #4]

nop

add sp, fp, #0

@ sp needed

ldr fp, [sp], #4

bx lr

.L6:

.align 2

.L5:

.word 1074663424

.size Enable\_DMA, .-Enable\_DMA

;declaración de funcion “Values\_DAC”

.align 2

.global Values\_DAC

.syntax unified

.arm

.fpu softvfp

.type Values\_DAC, %function

Values\_DAC:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 8

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

@ link register save eliminated.

str fp, [sp, #-4]!

add fp, sp, #0

sub sp, sp, #12

str r0, [fp, #-8]

ldr r2, .L8

ldr r3, [fp, #-8]

lsl r3, r3, #6

lsl r3, r3, #16

lsr r3, r3, #16

orr r3, r3, #65536

str r3, [r2]

nop

add sp, fp, #0

@ sp needed

ldr fp, [sp], #4

bx lr

.L9:

.align 2

.L8:

.word 1074663424

.size Values\_DAC, .-Values\_DAC

;declaración de funcion “cfg\_DAC”

.align 2

.global cfg\_DAC

.syntax unified

.arm

.fpu softvfp

.type cfg\_DAC, %function

cfg\_DAC:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 8

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

push {fp, lr}

add fp, sp, #4

sub sp, sp, #8

bl Enable\_ENAIO

ldr r3, .L11

str r3, [fp, #-8]

ldr r2, .L11+4

ldr r3, [fp, #-8]

lsl r3, r3, #16

lsr r3, r3, #16

str r3, [r2, #8]

nop

sub sp, fp, #4

@ sp needed

pop {fp, lr}

bx lr

.L12:

.align 2

.L11:

.word 2549

.word 1074663424

.size cfg\_DAC, .-cfg\_DAC

;declaración de funcion “cfg\_DMA”

.align 2

.global cfg\_DMA

.syntax unified

.arm

.fpu softvfp

.type cfg\_DMA, %function

cfg\_DMA:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 24

@ 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

sub sp, sp, #24 ;espacio para varibles locales

; en las siguientes 4 líneas usa str para guardar los parametros en los registros en c para ;llamar a la funcion enable\_DMA, los parametros son:

str r0, [fp, #-16] ;LLI\_T \*p\_LLI0

str r1, [fp, #-20] ;LLI\_T \*p\_LLI1

str r2, [fp, #-24] ;unsigned int\* src\_signal\_0

str r3, [fp, #-28] ;unsigned int\* src\_signal\_1

;llama a dos funciones del mismo archivo

bl Enable\_DMA ;branch & link de Enable DMA

bl cfg\_DAC ;branch & link de cfg\_DAC

ldr r3, .L14 ;carga SAMPLES\_NUM en r3

ldr r3, [r3] ;carga la direccion de SAMPLES\_NUM (guardada en r3) en r3

orr r3, r3, #-2046820352 ;hace un OR para poner 1 en posicion 25, 26 y 31

orr r3, r3, #4718592 ;hace un OR para poner 1 en posicion 19 y 22

str r3, [fp, #-8] ;guarda la mascara en DMA\_CONTROL\_MASK

;p\_LLI0->source=(unsigned int) src\_signal\_0;

ldr r2, [fp, #-24] ;carga la direccion src\_signal\_0 en r2

ldr r3, [fp, #-16] ;carga la direccion LLI\_T \*p\_LLI0 en r3

str r2, [r3] ;guarda p\_LLI0 en r2

;p\_LLI0->destination=(unsigned int) &(DAC->CR);

ldr r3, [fp, #-16] ;carga la direccion LLI\_T \*p\_LLI0 en r3

ldr r2, .L14+4 ;carga la direccion de DAC

str r2, [r3, #4] ;guarda r3+4 (p\_LLI0->source) en r2

;se mueve 4 porque es un unsigned int

;p\_LLI0->next= (unsigned int)p\_LLI1;

ldr r2, [fp, #-20] ;carga la direccion LLI\_T \*p\_LLI1 en r2

ldr r3, [fp, #-16] ;carga la direccion LLI\_T \*p\_LLI0 en r3

str r2, [r3, #8] ;guarda r3+8 (p\_LLI0->destination) en r2

;se mueve 8 porque es el miembro 3

;p\_LLI0->control= DMA\_CONTROL\_MASK;

ldr r3, [fp, #-16] ;carga la direccion LLI\_T \*p\_LLI0 en r3

ldr r2, [fp, #-8] ;carga DMA\_CONTROL\_MASK;

str r2, [r3, #12] ;guarda r3+12 (p\_LLI0->next) en r2

;se mueve 12 porque es el miembro 4

;p\_LLI1->source=(unsigned int) src\_signal\_1;

ldr r2, [fp, #-28] ;carga la direccion src\_signal\_1 en r2

ldr r3, [fp, #-20] ;carga la direccion LLI\_T \*p\_LLI1 en r3

str r2, [r3] ;guarda p\_LLI1 en r2

;p\_LLI1->destination=(unsigned int) &(DAC->CR);

ldr r3, [fp, #-20]

ldr r2, .L14+4 ;carga la direccion de DAC

str r2, [r3, #4]

;p\_LLI1->next=(unsigned int) p\_LLI0;

ldr r2, [fp, #-16]

ldr r3, [fp, #-20]

str r2, [r3, #8]

;p\_LLI1->control= DMA\_CONTROL\_MASK;

ldr r3, [fp, #-20]

ldr r2, [fp, #-8]

str r2, [r3, #12]

;(GPDMA->CH[0]).LLI=\*p\_LLI0;

ldr r3, .L14+8 ;carga la direccion de GPDMA

ldr r2, [fp, #-16] ;carga la direccion LLI\_T \*p\_LLI0 en r3

add ip, r3, #256 ;(ip scratch register se usa para pisar.)

mov r3, r2 ;Copia la direccion de r2 a r3.

ldm r3, {r0, r1, r2, r3}

stm ip, {r0, r1, r2, r3}

;(GPDMA->CONFIG)|=1;

ldr r2, .L14+8 ;carga la direccion de GPDMA

ldr r3, .L14+8 ;carga la direccion de GPDMA

ldr r3, [r3, #48]

orr r3, r3, #1 ;OR contra 1

str r3, [r2, #48]

;(GPDMA->CH[0]).CONFIG = 1 ….

ldr r3, .L14+8 ;carga la direccion de GPDMA

ldr r2, .L14+12 ;mascara del resultado del OR

str r2, [r3, #272] ;guarda la mascara en (GPDMA->CH[0]).CONFIG

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 ;branch a la direccion del lr que se habia guardado en la pila

.L15:

.align 2

.L14:

.word SAMPLES\_NUM

.word 1074663424 ;direccion de DAC

.word 1073750016 ;direccion de GPDMA

.word 35777 ;mascara

.size cfg\_DMA, .-cfg\_DMA

.global \_\_aeabi\_i2d

.global \_\_aeabi\_dmul

.global \_\_aeabi\_ddiv

.global \_\_aeabi\_dadd

.global \_\_aeabi\_d2iz

;declaracion de función signal fill

.align 2

.global signal\_fill

.syntax unified

.arm

.fpu softvfp

.type signal\_fill, %function

signal\_fill:

@ Function supports interworking.

@ args = 0, pretend = 0, frame = 24

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

push {r4, r5, r6, r7, r8, r9, r10, fp, lr} ;guarda: “frame pointer”, “Link register”

;y registros de 4 a 10 en el stack pointer

add fp, sp, #32 ;el fp=sp+32

sub sp, sp, #28 ;sp=sp-28

str r0, [fp, #-56] ;guarda src\_signal\_0 en fp-56

str r1, [fp, #-60] ;guarda src\_signal\_1 en fp-60

mov r2, sp

mov r10, r2 ;copia la direccion de sp en r10

;int i=0

mov r2, #0 ;carga un 0 en r2

str r2, [fp, #-40] ;store en fp-40 del 0 cargado en r2

;fp-40 va a representar int i

;int SAMPLES\_NUM = SAMPLE\_FREQ/MIN\_FREQ;

mov r2, #80 ;guarda 80 en r2

;80 es SAMPLES\_NUM

;= SAMPLE\_FREQ/MIN\_FREQ;

str r2, [fp, #-44] ;store en fp-40 del 80 cargado en r2

;fp-44 va a representar SAMPLES\_NUM

;double aux[SAMPLES\_NUM];

ldr r0, [fp, #-44] ;carga lo que esta en fp-44 a r0

;es decir carga SAMPLES\_NUM en r0

sub r2, r0, #1 ;en r2 carga r0-1

str r2, [fp, #-48] ;guarda en fp-48 de r2

;fp-48 va a representar el tamaño de aux

;Organiza registros para guardar el vector aux

mov r2, r0

mov r1, r2

mov r2, #0

lsl r6, r2, #6

orr r6, r6, r1, lsr #26

lsl r5, r1, #6

mov r2, r0

mov r1, r2

mov r2, #0

lsl r4, r2, #6

orr r4, r4, r1, lsr #26

lsl r3, r1, #6

mov r3, r0

lsl r3, r3, #3

add r3, r3, #7

add r3, r3, #7

lsr r3, r3, #3

lsl r3, r3, #3

sub sp, sp, r3

mov r3, sp

add r3, r3, #7

lsr r3, r3, #3

lsl r3, r3, #3

str r3, [fp, #-52] ;guarda en fp-52 de r3 la direccion de aux

mov r3, #0

;for(i=0;i<SAMPLES\_NUM;i++)

;i=0

str r3, [fp, #-40] ;carga de nuevo valor de i → 0

b .L17 ;salto incondicional a .L17

;este salto es incondicional, ya que

;representa la primera comparación del

;for --> i<SAMPLES\_NUM

.L18: ;en C representa el interior del for

;aux[i]= (flag\_T1\*sin(2\*PI\*i/SAMPLES\_NUM) + flag\_T2\*sin(2\*PI\*2\*i/SAMPLES\_NUM) + ;flag\_T3\*sin(2\*PI\*4\*i/SAMPLES\_NUM) + ;flag\_T4\*sin(2\*PI\*8\*i/SAMPLES\_NUM))/(flag\_T1+flag\_T2+flag\_T3+flag\_T4);

ldr r3, .L19+32 ;carga en r3 flag\_T1

;r3=flag\_T1

ldrb r3, [r3] @ zero\_extendqisi2

mov r0, r3 ;r0=flag\_T1

bl \_\_aeabi\_i2d ;conversion a double de r0

;r0 y r1 ⇒ flag\_T1 en double

mov r4, r0

mov r5, r1 ;r4 y r5 ⇒ flag\_T1 en double

ldr r0, [fp, #-40] ;carga i en r0

;r0=i

bl \_\_aeabi\_i2d ;conversión a double de r0

;r0 y r1 ⇒ i convertido a double

adr r3, .L19 ;carga .L19 en r3

;r3=2\*PI

ldmia r3, {r2-r3} ;carga PI enmúltiples registros

;r2 y r3 ⇒ 2\*PI en double

bl \_\_aeabi\_dmul ;(r0 , r1) \* (r2 , r3) =i \* 2PI

;r0 y r1 ⇒ 2PI\* i convertido a double

mov r2, r0

mov r3, r1

mov r6, r2

mov r7, r3 ;r6 y r7 ⇒ 2pi\*i (double)

ldr r0, [fp, #-44] ;en r0 carga SAMPLES\_NUM

bl \_\_aeabi\_i2d ;conversión a double de SAMPLES\_NUM

;r0 y r1 ⇒ SAMPLES\_NUM

mov r2, r0

mov r3, r1 ;r2 y r3 ⇒ SAMPLES\_NUM

mov r0, r6

mov r1, r7 ;r0 y r1⇒ i\*2pi

bl \_\_aeabi\_ddiv ;divide i\*2PI/SAMPLES\_NUM

;r0 y r1 ⇒ i\*2PI/SAMPLES\_NUM

mov r2, r0

mov r3, r1

mov r0, r2

mov r1, r3 ;r0 y r1 ⇒ i\*2PI/SAMPLES\_NUM

;sin(2\*PI\*i/SAMPLES\_NUM)

bl sin ;llamada a sin con i\*2PI/SAMPLES\_NUM

mov r2, r0

mov r3, r1 ;r2 y r3 ⇒ sin(2\*PI\*i/SAMPLES\_NUM)

;r4 y r5 ⇒ flag\_T1 en double

mov r0, r4

mov r1, r5

;r0 y r1 ⇒ flag\_T1 en double

bl \_\_aeabi\_dmul

mov r3, r0

mov r4, r1

mov r5, r4

mov r4, r3

;repite las mismas instrucciones para los demás términos de los senos

ldr r3, .L19+36

ldrb r3, [r3] @ zero\_extendqisi2

mov r0, r3

bl \_\_aeabi\_i2d

mov r6, r0

mov r7, r1

ldr r0, [fp, #-40]

bl \_\_aeabi\_i2d

adr r3, .L19+8

ldmia r3, {r2-r3}

bl \_\_aeabi\_dmul

mov r2, r0

mov r3, r1

mov r8, r2

mov r9, r3

ldr r0, [fp, #-44]

bl \_\_aeabi\_i2d

mov r2, r0

mov r3, r1

mov r0, r8

mov r1, r9

bl \_\_aeabi\_ddiv

mov r2, r0

mov r3, r1

mov r0, r2

mov r1, r3

;sin(2\*PI\*2\*i/SAMPLES\_NUM)

bl sin

mov r2, r0

mov r3, r1

mov r0, r6

mov r1, r7

bl \_\_aeabi\_dmul

mov r2, r0

mov r3, r1

mov r0, r4

mov r1, r5

bl \_\_aeabi\_dadd

mov r3, r0

mov r4, r1

mov r5, r4

mov r4, r3

ldr r3, .L19+40

ldrb r3, [r3] @ zero\_extendqisi2

mov r0, r3

bl \_\_aeabi\_i2d

mov r6, r0

mov r7, r1

ldr r0, [fp, #-40]

bl \_\_aeabi\_i2d

adr r3, .L19+16

ldmia r3, {r2-r3}

bl \_\_aeabi\_dmul

mov r2, r0

mov r3, r1

mov r8, r2

mov r9, r3

ldr r0, [fp, #-44]

bl \_\_aeabi\_i2d

mov r2, r0

mov r3, r1

mov r0, r8

mov r1, r9

bl \_\_aeabi\_ddiv

mov r2, r0

mov r3, r1

mov r0, r2

mov r1, r3}

;sin(2\*PI\*4\*i/SAMPLES\_NUM)

bl sin

mov r2, r0

mov r3, r1

mov r0, r6

mov r1, r7

bl \_\_aeabi\_dmul

mov r2, r0

mov r3, r1

mov r0, r4

mov r1, r5

bl \_\_aeabi\_dadd

mov r3, r0

mov r4, r1

mov r5, r4

mov r4, r3

ldr r3, .L19+44

ldrb r3, [r3] @ zero\_extendqisi2

mov r0, r3

bl \_\_aeabi\_i2d

mov r6, r0

mov r7, r1

ldr r0, [fp, #-40]

bl \_\_aeabi\_i2d

adr r3, .L19+24

ldmia r3, {r2-r3}

bl \_\_aeabi\_dmul

mov r2, r0

mov r3, r1

mov r8, r2

mov r9, r3

ldr r0, [fp, #-44]

bl \_\_aeabi\_i2d

mov r2, r0

mov r3, r1

mov r0, r8

mov r1, r9

bl \_\_aeabi\_ddiv

mov r2, r0

mov r3, r1

mov r0, r2

mov r1, r3

;sin(2\*PI\*8\*i/SAMPLES\_NUM))

bl sin

mov r2, r0

mov r3, r1

mov r0, r6

mov r1, r7

bl \_\_aeabi\_dmul

mov r2, r0

mov r3, r1

mov r0, r4

mov r1, r5

bl \_\_aeabi\_dadd

mov r3, r0

mov r4, r1

mov r5, r4

mov r4, r3

;sumatoria de flags → (flag\_T1+flag\_T2+flag\_T3+flag\_T4);

ldr r3, .L19+32 ;carga flag\_T1

ldrb r3, [r3] @ zero\_extendqisi2

mov r2, r3 ;r2⇒ flag\_T1

ldr r3, .L19+36 ;carga flag\_T2

ldrb r3, [r3] @ zero\_extendqisi2

add r3, r2, r3 ;r3 = flag\_T1+flag\_T2

ldr r2, .L19+40 ;carga flag\_T3

ldrb r2, [r2] @ zero\_extendqisi2

add r3, r3, r2 ;r3 = flag\_T1+flag\_T2+flag\_T3

ldr r2, .L19+44 ;carga flag\_T4

ldrb r2, [r2] @ zero\_extendqisi2

add r3, r3, r2 ;r3 = flag\_T1+flag\_T2+flag\_T3+flag\_T4

mov r0, r3

bl \_\_aeabi\_i2d

mov r2, r0

mov r3, r1 ;r2 y r3 ⇒ flag\_T1+flag\_T2+flag\_T3+flag\_T4

mov r0, r4

mov r1, r5 ;r0 y r1 ⇒ resultado sumatoria de senos.

división → (sumatoria senos)/(flag\_T1+flag\_T2+flag\_T3+flag\_T4);

bl \_\_aeabi\_ddiv

mov r3, r0

mov r4, r1 ;r3 y r4 ⇒ resultado de la división

;aux[i]=resultado de la division

ldr r2, [fp, #-40] ;carga i en r2

lsl r2, r2, #3 ;desplaza 3 lugares r2 → i<<3

add r2, r1, r2 ;r2=r1+r2 ⇒ r2 es la dirección de aux[i]

stm r2, {r3-r4} ; guarda en aux[i] el resultado de la división

;src\_signal\_0[i]= ((int) (511\*aux[i]+512)<< 6) | (1<<16);

ldr r3, [fp, #-40] ;carga i en r3

lsl r3, r3, #2 ;desplaza 2 lugares r3 → i<<2

ldr r2, [fp, #-56] ;carga src\_signal\_0 en r2

add r4, r2, r3

ldr r2, [fp, #-52] ;carga aux[i]

ldr r3, [fp, #-40] ;carga i en r3

lsl r3, r3, #3 ;desplaza 3 lugares r3→ i<<3

add r3, r2, r3 ;r2=r1+r2 ⇒ r2 es la dirección de aux[i]

ldmia r3, {r0-r1} ; r0y r1 ⇒ aux[i]

mov r2, #0

ldr r3, .L19+48 ;carga 511 (double)

;multiplicacion → 511\*aux[i]

bl \_\_aeabi\_dmul

mov r2, r0

mov r3, r1

mov r0, r2

mov r1, r3

mov r2, #0

ldr r3, .L19+52 ;carga 512 (double)

;suma → (511\*aux[i]+512)

bl \_\_aeabi\_dadd

mov r2, r0

mov r3, r1

mov r0, r2

mov r1, r3

bl \_\_aeabi\_d2iz ;casteo a int de (511\*aux[i]+512)

mov r3, r0

(511\*aux[i]+512)<< 6)

lsl r3, r3, #6

; or y desplazamiento→ | (1<<16);

orr r3, r3, #65536 ;#65536 ⇒ (1<<16)

str r3, [r4] ;guarda en la dirección de src\_signal\_0[i]

ldr r3, [fp, #-40] ;carga i en r3

lsl r3, r3, #2 ;r3=r3<<2

;src\_signal\_1[i]=src\_signal\_0[i];

ldr r2, [fp, #-60] ;load src\_signal\_1

add r3, r2, r3 ;src\_signal\_1[i]

ldr r2, [fp, #-40] ;carga i en r2

lsl r2, r2, #2 ;r2=r2<<2

ldr r1, [fp, #-56] ;src\_signal\_0

add r2, r1, r2 ;src\_signal\_0[i]

ldr r2, [r2] ;carga del valor en src\_signal\_0[i]

str r2, [r3] ;src\_signal\_1[i]=src\_signal\_0[i];

;incrementación de i → i++

ldr r3, [fp, #-40] ;carga i en r2

add r3, r3, #1 ;r3=r3+1 ⇒ i++

str r3, [fp, #-40] ;guarda el i ya incrementado

.L17:

;i<SAMPLES\_NUM

ldr r2, [fp, #-40] ;carga en r2 i

ldr r3, [fp, #-44] ;carga en r3 SAMPLES\_NUM

cmp r2, r3 ;comparación entre i y SAMPLES\_NUM

blt .L18 ;si la comparación es verdadera sigue a .L18

;es el condicional del for

mov sp, r10

nop

sub sp, fp, #32

@ sp needed

pop {r4, r5, r6, r7, r8, r9, r10, fp, lr}

bx lr

.L20:

.align 3

.L19:

.word 603906762

.word 1075388883

.word 603906762

.word 1076437459

.word 603906762

.word 1077486035

.word 603906762

.word 1078534611

.word flag\_T1 ;+32

.word flag\_T2 ;+36

.word flag\_T3 ;+40

.word flag\_T4 ;+44

.word 1082126336 ;+48

.word 1082130432 ;+52

.size signal\_fill, .-signal\_fill

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