H8SDK リファレンスマニュアル

rev. 33

2008 年 12 月 25 日

箕浦 賢嗣

K&J ソフトウェアデザイン

目次

1　はじめに

1.1　この文書について

この文書は、H8 Software Development Kit(以下H8SDK) のインタフェース 仕様書である。
文書は大きくデータ構造の解説セクションと、ファイルモジュール別のイ ンタフェース解説セクションに分かれており、それぞれのセクションは冒 頭で解説項目を概略説明付きで列挙した後、項目の詳細説明が続くという 構成になっている。

1.2　対象

H8SDK を使用してH8 ボード上アプリケーションの作成を行う開発者向けの文 書であり、H8 プラットフォームのハードウェア仕様を理解していることが 前提である。

1.3　名前空間

H8SDK は、以下の規則に基づいて名前空間が規定されている。

- '_' で始まる名前空間
- stddef.h、stdlib.h、stdio.h、assert.h 以外のモジュールに おいて、モジュール名として使われている 3 文字〜で始まる名前空間

注意

規則外のシンボルも一部存在する

2　バグ一覧

globalScope> メンバ MUSIC_setTempo (_UBYTE val)

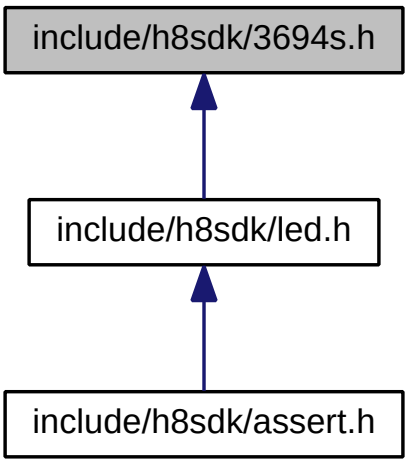
このAPI は未実装。呼び出すとハングアップする。

3　ファイル詳解

3.1　include/h8sdk/3694s.h ファイル

3694s I/O map

被依存関係図:



クラス

- struct _st_lvd
  LVD 構造マップ
- struct _st_iic2
  IIC2 構造マップ
- struct _st_tw
  TimerW 構造マップ
- struct _st_flash
- struct _st_tv
  TV 構造マップ
- struct _st_ta
  TimerA 構造マップ
- struct _st_sci3
  SCI3 構造マップ
- struct _st_ad
  A/D 構造マップ
- struct _st_wdt
  WDT 構造マップ
- struct _st_abrk
  ABRK 構造マップ
- struct _st_io
- union _un_syscr1
  SYSCR1 構造マップ
- union _un_syscr2
  SYSCR2 構造マップ
- union _un_iegr1
  IEGR1 構造マップ
- union _un_iegr2
- union _un_ienr1
  IENR1 構造マップ
- union _un_irr1
  < *IRR1* 構造マップ
- union _un_iwpr

IWPR **構造マップ**

- union _un_mstcr1
  MSTCR1 **構造マップ**
- union _st_lvd.CR
  LVDCR.
- struct _st_lvd.CR.BIT
- union _st_lvd.SR
  LVDSR.
- struct _st_lvd.SR.BIT
- union _st_iic2.ICCR1
  ICCR1.
- struct _st_iic2.ICCR1.BIT
- union _st_iic2.ICCR2
  ICCR2.
- struct _st_iic2.ICCR2.BIT
- union _st_iic2.ICMR
  ICMR.
- struct _st_iic2.ICMR.BIT
- union _st_iic2.ICIER
  ICIER.
- struct _st_iic2.ICIER.BIT
- union _st_iic2.ICSR
  ICSR.
- struct _st_iic2.ICSR.BIT
- union _st_iic2.SAR
  SAR.
- struct _st_iic2.SAR.BIT
- union _st_tw.TMRW
  TMRW.
- struct _st_tw.TMRW.BIT
- union _st_tw.TCRW
  TCRW.
- struct _st_tw.TCRW.BIT
- union _st_tw.TIERW
  TIERW.
- struct _st_tw.TIERW.BIT
- union _st_tw.TSRW
  TSRW.
- struct _st_tw.TSRW.BIT
- union _st_tw.TIOR0
  TIOR0.
- struct _st_tw.TIOR0.BIT
- union _st_tw.TIOR1
  TIOR1.
- struct _st_tw.TIOR1.BIT
- union _st_flash.FLMCR1
- struct _st_flash.FLMCR1.BIT
- union _st_flash.FLMCR2
- struct _st_flash.FLMCR2.BIT
- union _st_flash.FLPWCR
- struct _st_flash.FLPWCR.BIT
- union _st_flash.EBR1

- struct _st_flash.EBR1.BIT
- union _st_flash.FENR
- struct _st_flash.FENR.BIT
- union _st_tv.TCRV0
- struct _st_tv.TCRV0.BIT
- union _st_tv.TCSRV
- struct _st_tv.TCSRV.BIT
- union _st_tv.TCRV1
- struct _st_tv.TCRV1.BIT
- union _st_ta.TMA
- struct _st_ta.TMA.BIT
- union _st_sci3.SMR
- struct _st_sci3.SMR.BIT
- union _st_sci3.SCR3
- struct _st_sci3.SCR3.BIT
- union _st_sci3.SSR
- struct _st_sci3.SSR.BIT
- union _st_ad.ADCSR
- struct _st_ad.ADCSR.BIT
- union _st_ad.ADCR
- struct _st_ad.ADCR.BIT
- union _st_wdt.TCSRWD
- struct _st_wdt.TCSRWD.BIT
- union _st_wdt.TMWD
- struct _st_wdt.TMWD.BIT
- union _st_abrk.CR
- struct _st_abrk.CR.BIT
- union _st_abrk.SR
- struct _st_abrk.SR.BIT
- union _st_io.PUCR1
- struct _st_io.PUCR1.BIT
- union _st_io.PUCR5
- struct _st_io.PUCR5.BIT
- union _st_io.PDR1
- struct _st_io.PDR1.BIT
- union _st_io.PDR2
- struct _st_io.PDR2.BIT
- union _st_io.PDR5
- struct _st_io.PDR5.BIT
- union _st_io.PDR7
- struct _st_io.PDR7.BIT
- union _st_io.PDR8
- struct _st_io.PDR8.BIT
- union _st_io.PDRB
- struct _st_io.PDRB.BIT
- union _st_io.PMR1
- struct _st_io.PMR1.BIT
- union _st_io.PMR5
- struct _st_io.PMR5.BIT
- struct _un_syscr1.BIT
- struct _un_syscr2.BIT
- struct _un_iegr1.BIT

- struct _un_iegr2.BIT
- struct _un_ienr1.BIT
- struct _un_irr1.BIT
- struct _un_iwpr.BIT
- struct _un_mstcr1.BIT

マクロ定義

- #define _LVD (*(volatile struct _st_lvd *)0xF730)
  LVD Address.
- #define _IIC2 (*(volatile struct _st_iic2 *)0xF748)
  IIC2 Address.
- #define _TW (*(volatile struct _st_tw *)0xFF80)
  TW Address.
- #define _FLASH (*(volatile struct _st_flash *)0xFF90)
  FLASH Address.
- #define _TV (*(volatile struct _st_tv *)0xFFA0)
  TV Address.
- #define _TA (*(volatile struct _st_ta *)0xFFA6)
  TA Address.
- #define _SCI3 (*(volatile struct _st_sci3 *)0xFFA8)
  SCI3 Address.
- #define _AD (*(volatile struct _st_ad *)0xFFB0)
  A/D Address.
- #define _WDT (*(volatile struct _st_wdt *)0xFFC0)
  WDT Address.
- #define _ABRK (*(volatile struct _st_abrk *)0xFFC8)
  ABRK Address.
- #define _IO (*(volatile struct _st_io *)0xFFD0)
  IO Address.
- #define _SYSCR1 (*(volatile union _un_syscr1*)0xFFF0)
  SYSCR1 Address.
- #define _SYSCR2 (*(volatile union _un_syscr2*)0xFFF1)
  SYSCR2Address.
- #define _IEGR1 (*(volatile union _un_iegr1 *)0xFFF2)
  IEGR1 Address.
- #define _IEGR2 (*(volatile union _un_iegr2 *)0xFFF3)
  IEGR2 Address.
- #define _IENR1 (*(volatile union _un_ienr1 *)0xFFF4)
  IENR1 Address.
- #define _IRR1 (*(volatile union _un_irr1 *)0xFFF6)
  IRR1 Address.
- #define _IWPR (*(volatile union _un_iwpr *)0xFFF8)
  IWPR Address.
- #define _MSTCR1 (*(volatile union _un_mstcr1*)0xFFF9)
  MSTCR1 Address.

3.1.1 詳解

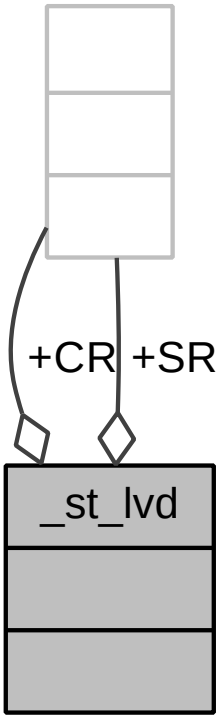
3694s I/O map

3.1.2 クラス詳解

■struct _st_lvd LVD 構造マップ

3694s.h の 23 行目に定義があります。

_st_lvd 連携図



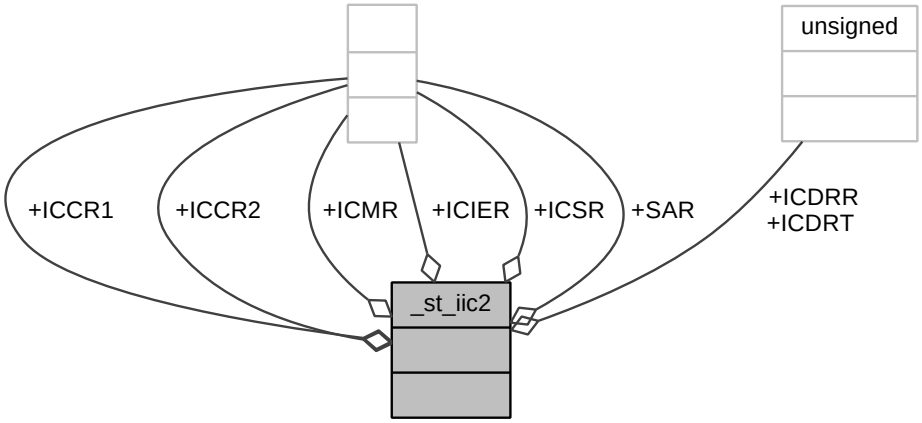
クラスメンバ

union _st_lvd	CR	LVDCR.
union _st_lvd	SR	LVDSR.

■struct _st_iic2 IIC2 構造マップ

3694s.h の 57 行目に定義があります。

_st_iic2 連携図



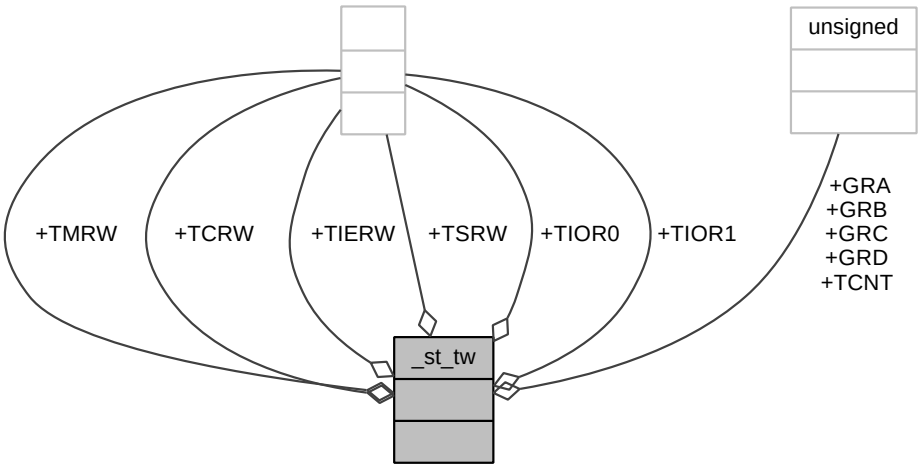
クラスメンバ

union _st_iic2	ICCR1	ICCR1.
union _st_iic2	ICCR2	ICCR2.
union _st_iic2	ICMR	ICMR.
union _st_iic2	ICIER	ICIER.
union _st_iic2	ICSR	ICSR.
union _st_iic2	SAR	SAR.
unsigned char	ICDRT	
unsigned char	ICDRR	

■struct _st_tw TimerW 構造マップ

3694s.h の 158 行目に定義があります。

_st_tw 連携図

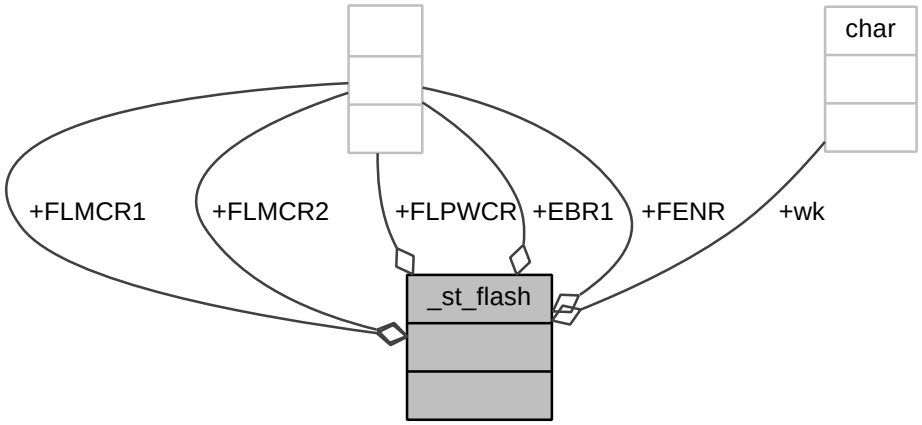


クラスメンバ

union _st_tw	TMRW	TMRW.
union _st_tw	TCRW	TCRW.
union _st_tw	TIERW	TIERW.
union _st_tw	TSRW	TSRW.
union _st_tw	TIOR0	TIOR0.
union _st_tw	TIOR1	TIOR1.
unsigned int	TCNT	
unsigned int	GRA	
unsigned int	GRB	
unsigned int	GRC	
unsigned int	GRD	

■struct _st_flash 3694s.h の 260 行目に定義があります。

_st_flash 連携図



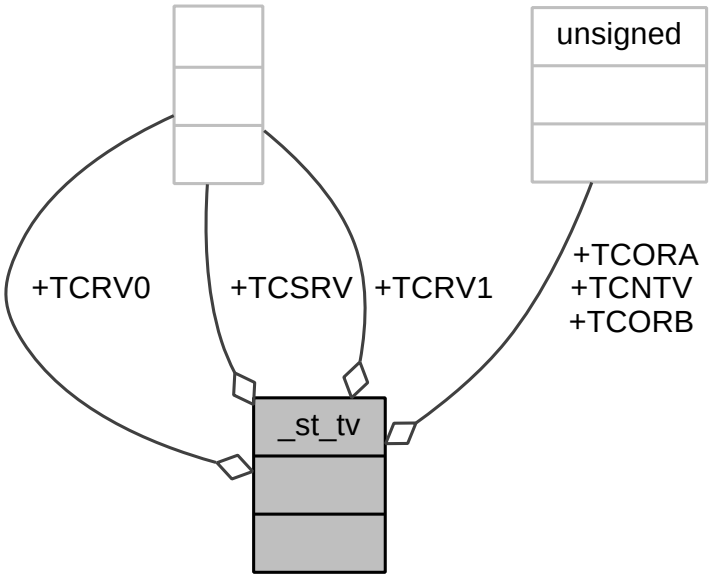
クラスメンバ

union _st_flash	FLMCR1	
union _st_flash	FLMCR2	
union _st_flash	FLPWCR	
union _st_flash	EBR1	
char	wk[7]	
union _st_flash	FENR	

■struct _st_tv TV 構造マップ

3694s.h の 327 行目に定義があります。

_st_tv 連携図



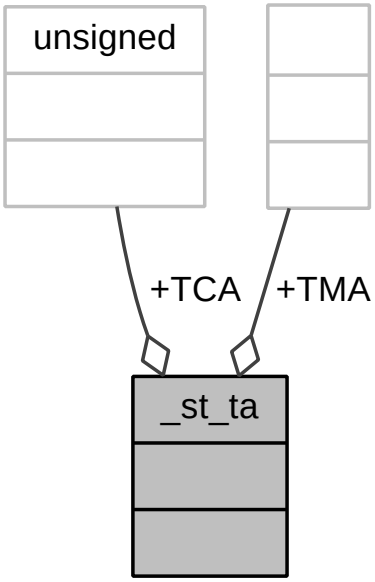
クラスメンバ

union _st_tv	TCRV0	
union _st_tv	TCSRV	
unsigned char	TCORA	
unsigned char	TCORB	
unsigned char	TCNTV	
union _st_tv	TCRV1	

■struct _st_ta TimerA 構造マップ

3694s.h の 376 行目に定義があります。

_st_ta 連携図



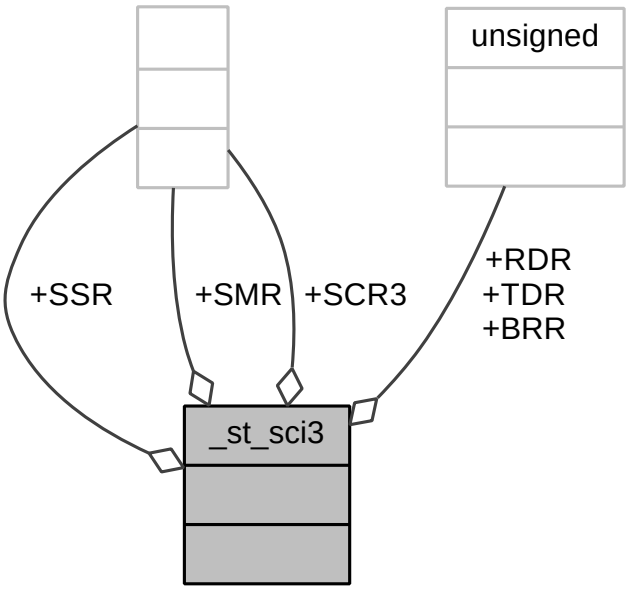
クラスメンバ

union _st_ta	TMA	
unsigned char	TCA	

■struct _st_sci3 SCI3 構造マップ

3694s.h の 393 行目に定義があります。

_st_sci3 連携図



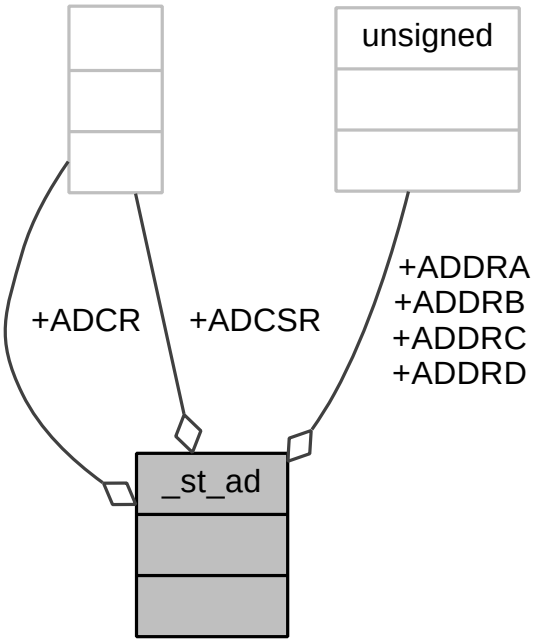
クラスメンバ

union _st_sci3	SMR	
unsigned char	BRR	
union _st_sci3	SCR3	
unsigned char	TDR	
union _st_sci3	SSR	
unsigned char	RDR	

■struct _st_ad A/D 構造マップ

3694s.h の 438 行目に定義があります。

_st_ad 連携図



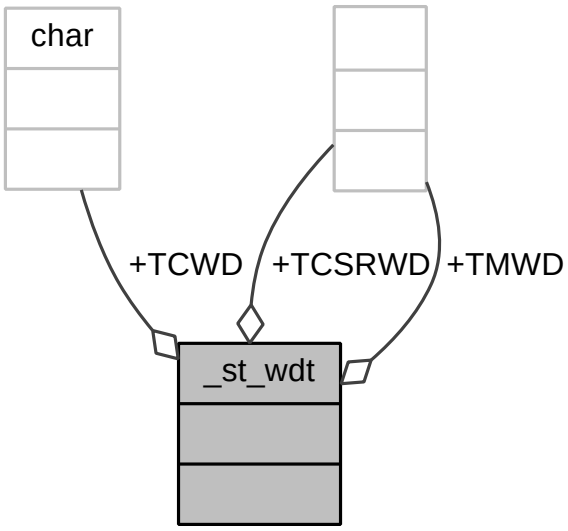
クラスメンバ

unsigned int	ADDRA	
unsigned int	ADDRB	
unsigned int	ADDRC	
unsigned int	ADDRD	
union _st_ad	ADCSR	
union _st_ad	ADCR	

■struct _st_wdt WDT 構造マップ

3694s.h の 464 行目に定義があります。

_st_wdt 連携図



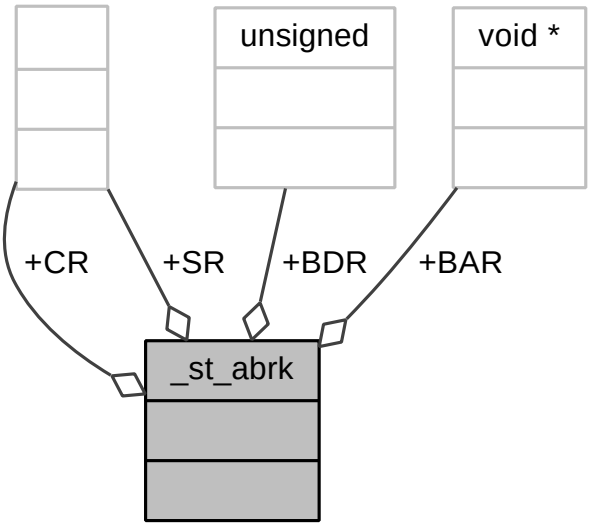
クラスメンバ

union _st_wdt	TCSRWD	
unsigned char	TCWD	
union _st_wdt	TMWD	

■struct _st_abrk ABRK 構造マップ

3694s.h の 492 行目に定義があります。

_st_abrk 連携図

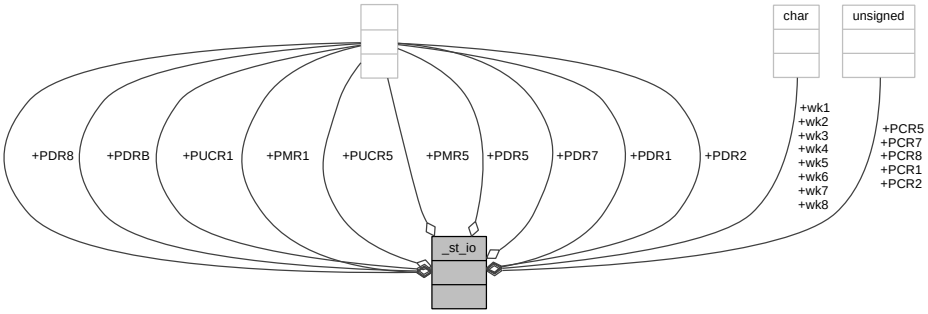


クラスメンバ

union _st_abrk	CR	
union _st_abrk	SR	
void *	BAR	
unsigned int	BDR	

■struct _st_io 3694s.h の 517 行目に定義があります。

_st_io 連携図



クラスメンバ

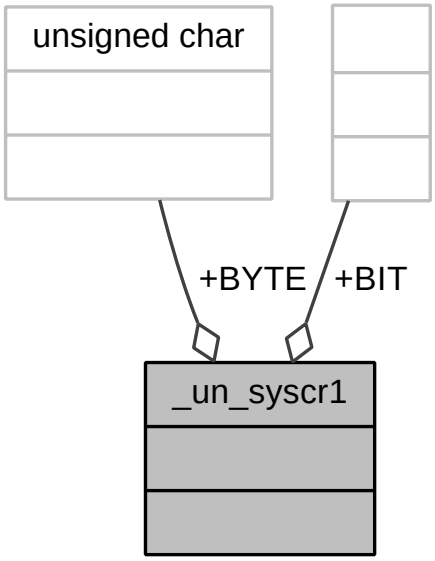
union _st_io	PUCR1	
union _st_io	PUCR5	
char	wk1[2]	
union _st_io	PDR1	
union _st_io	PDR2	
char	wk2[2]	
union _st_io	PDR5	
char	wk3	
union _st_io	PDR7	
union _st_io	PDR8	
char	wk4	
union _st_io	PDRB	
char	wk5[2]	
union _st_io	PMR1	
union _st_io	PMR5	
char	wk6[2]	

unsigned char	PCR1	
unsigned char	PCR2	
char	wk7[2]	
unsigned char	PCR5	
char	wk8	
unsigned char	PCR7	
unsigned char	PCR8	

■union _un_syscr1 SYSCR1 構造マップ

3694s.h の 673 行目に定義があります。

_un_syscr1 連携図



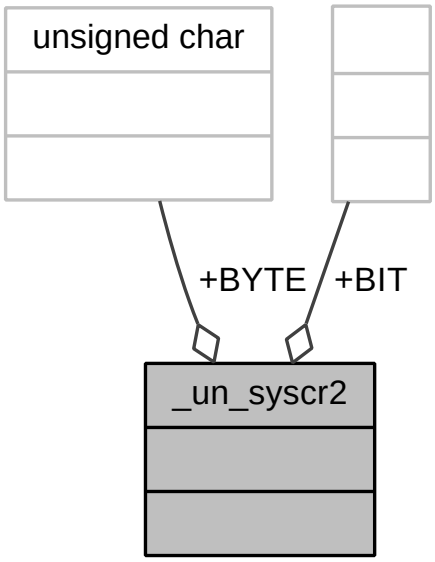
クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_syscr1	BIT	

■union _un_syscr2 SYSCR2 構造マップ

3694s.h の 684 行目に定義があります。

_un_syscr2 連携図



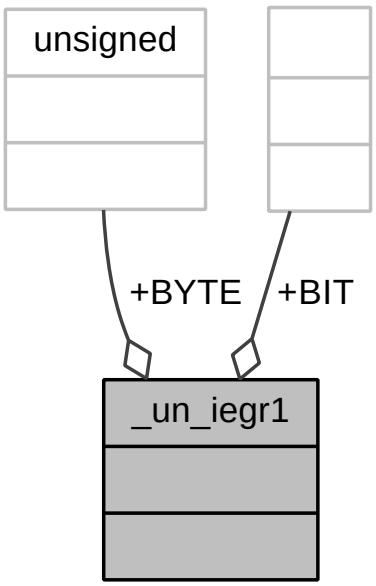
クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_syscr2	BIT	

■union _un_iegr1 IEGR1 構造マップ

3694s.h の 697 行目に定義があります。

_un_iegr1 連携図

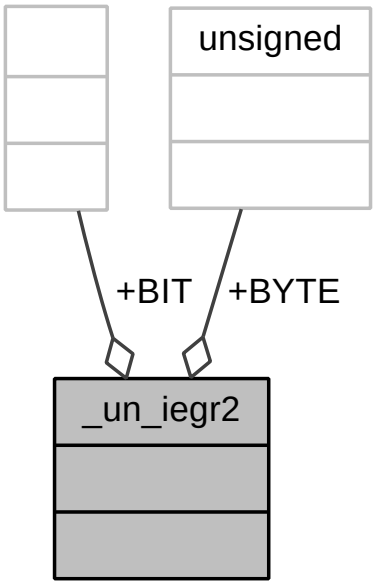


クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_iegr1	BIT	

■union _un_iegr2 3694s.h の 711 行目に定義があります。

_un_iegr2 連携図



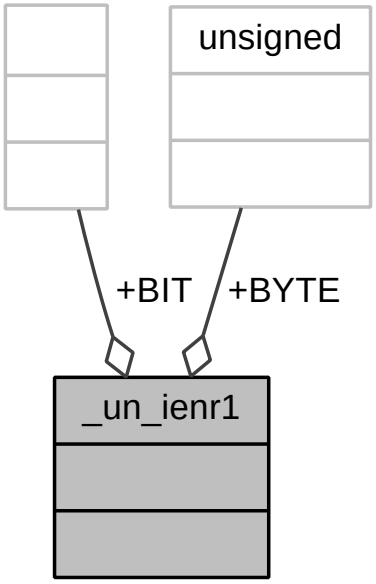
クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_iegr2	BIT	

■union _un_ienr1 IENR1 構造マップ

3694s.h の 726 行目に定義があります。

_un_ienr1 連携図



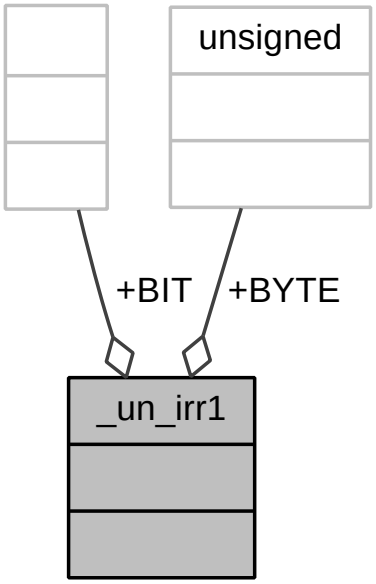
クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_ienr1	BIT	

■union _un_irr1 ＜ IRR1 構造マップ

3694s.h の 742 行目に定義があります。

_un_irr1 連携図



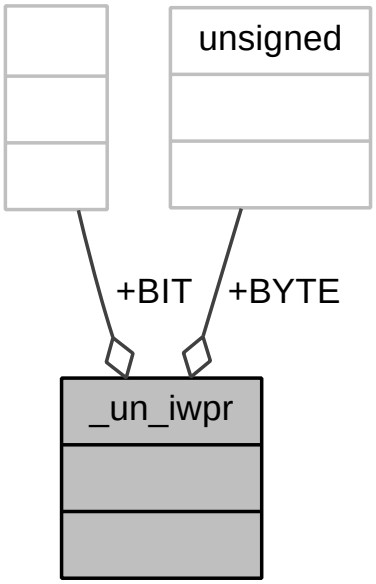
クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_irr1	BIT	

■union _un_iwpr IWPR 構造マップ

3694s.h の 757 行目に定義があります。

_un_iwpr 連携図



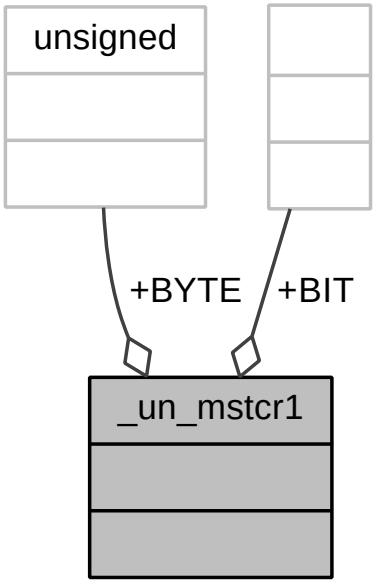
クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_iwpr	BIT	

■union _un_mstcr1 MSTCR1 構造マップ

3694s.h の 772 行目に定義があります。

_un_mstcr1 連携図



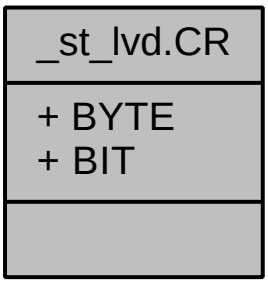
クラスメンバ

unsigned char	BYTE	Byte Access.
struct _un_mstcr1	BIT	

■union _st_lvd.CR LVDCR.

3694s.h の 26 行目に定義があります。

_st_lvd.CR 連携図

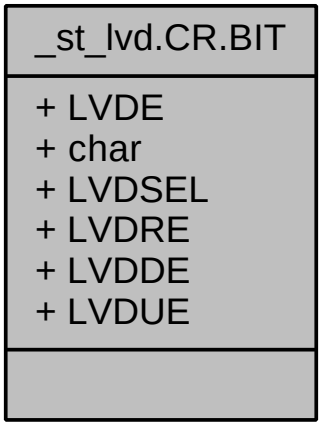


クラスメンバ

unsigned char	BYTE	Byte Access.
CR	BIT	Bit Access.

■struct _st_lvd.CR.BIT 3694s.h の 30 行目に定義があります。

_st_lvd.CR.BIT 連携図



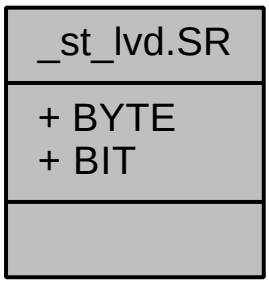
クラスメンバ

unsigned char	LVDE:1	
unsigned	char:3	
unsigned char	LVDSEL:1	
unsigned char	LVDRE:1	
unsigned char	LVDDE:1	
unsigned char	LVDUE:1	

■union _st_lvd.SR LVDSR.

3694s.h の 42 行目に定義があります。

_st_lvd.SR 連携図

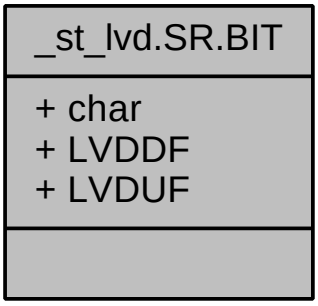


クラスメンバ

unsigned char	BYTE	Byte Access.
SR	BIT	Bit Access.

■struct _st_lvd.SR.BIT 3694s.h の 46 行目に定義があります。

_st_lvd.SR.BIT 連携図



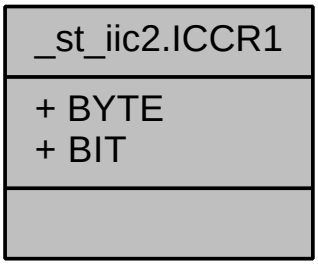
クラスメンバ

unsigned	char:6	
unsigned char	LVDDF:1	
unsigned char	LVDUF:1	

■union _st_iic2.ICCR1 ICCR1.

3694s.h の 59 行目に定義があります。

_st_iic2.ICCR1 連携図

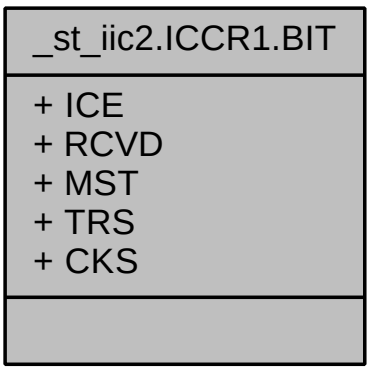


クラスメンバ

unsigned char	BYTE	Byte Access.
ICCR1	BIT	Bit Access.

■struct _st_iic2.ICCR1.BIT 3694s.h の 63 行目に定義があります。

_st_iic2.ICCR1.BIT 連携図



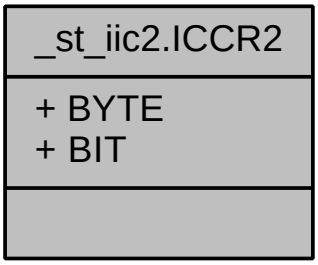
クラスメンバ

unsigned char	ICE:1	
unsigned char	RCVD:1	
unsigned char	MST:1	
unsigned char	TRS:1	
unsigned char	CKS:4	

■union _st_iic2.ICCR2 ICCR2.

3694s.h の 74 行目に定義があります。

_st_iic2.ICCR2 連携図

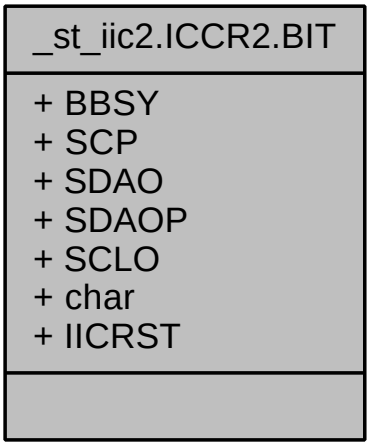


クラスメンバ

unsigned char	BYTE	Byte Access.
ICCR2	BIT	Bit Access.

■struct _st_iic2.ICCR2.BIT 3694s.h の 78 行目に定義があります。

_st_iic2.ICCR2.BIT 連携図



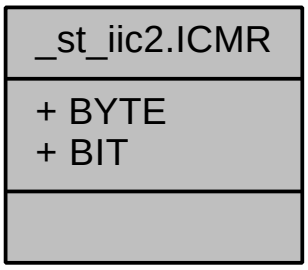
クラスメンバ

unsigned char	BBSY:1	
unsigned char	SCP:1	
unsigned char	SDAO:1	
unsigned char	SDAOP:1	
unsigned char	SCLO:1	
unsigned	char:1	
unsigned char	IICRST:1	

■union _st_iic2.ICMR ICMR.

3694s.h の 91 行目に定義があります。

_st_iic2.ICMR 連携図



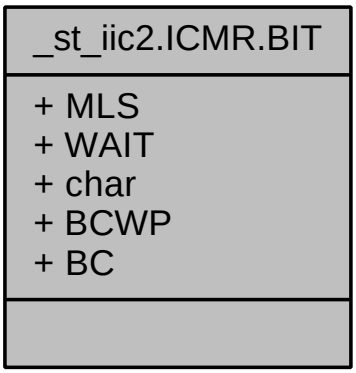
クラスメンバ

unsigned char	BYTE	Byte Access.

ICMR	BIT	Bit Access.

■struct _st_iic2.ICMR.BIT 3694s.h の 95 行目に定義があります。

_st_iic2.ICMR.BIT 連携図



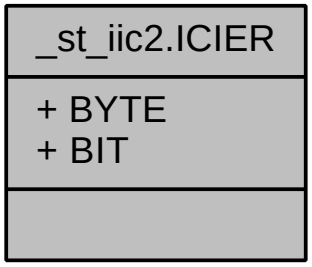
クラスメンバ

unsigned char	MLS:1	
unsigned char	WAIT:1	
unsigned	char:2	
unsigned char	BCWP:1	
unsigned char	BC:3	

■union _st_iic2.ICIER ICIER.

3694s.h の 106 行目に定義があります。

_st_iic2.ICIER 連携図



クラスメンバ

unsigned char	BYTE	Byte Access.

ICIER	BIT	Bit Access.

■struct _st_iic2.ICIER.BIT 3694s.h の 110 行目に定義があります。

_st_iic2.ICIER.BIT 連携図

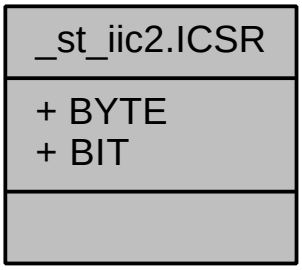
クラスメンバ

unsigned char	TIE:1	
unsigned char	TEIE:1	
unsigned char	RIE:1	
unsigned char	NAKIE:1	
unsigned char	STIE:1	
unsigned char	ACKE:1	
unsigned char	ACKBR:1	
unsigned char	ACKBT:1	

■union _st_iic2.ICSR ICSR.

3694s.h の 124 行目に定義があります。

_st_iic2.ICSR 連携図



クラスメンバ

unsigned char	BYTE	Byte Access.
ICSR	BIT	Bit Access.

■struct _st_iic2.ICSR.BIT 3694s.h の 128 行目に定義があります。

_st_iic2.ICSR.BIT 連携図

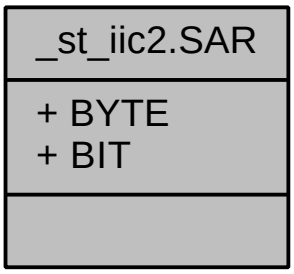
クラスメンバ

unsigned char	TDRE:1	
unsigned char	TEND:1	
unsigned char	RDRF:1	
unsigned char	NACKF:1	
unsigned char	STOP:1	
unsigned char	ALOVE:1	
unsigned char	AAS:1	
unsigned char	ADZ:1	

■union _st_iic2.SAR SAR.

3694s.h の 142 行目に定義があります。

_st_iic2.SAR 連携図

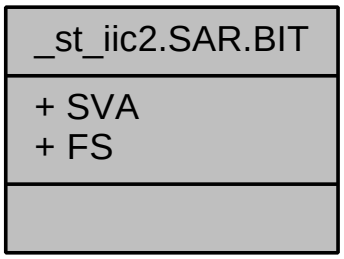


クラスメンバ

unsigned char	BYTE	Byte Access.
SAR	BIT	Bit Access.

■struct _st_iic2.SAR.BIT 3694s.h の 146 行目に定義があります。

_st_iic2.SAR.BIT 連携図



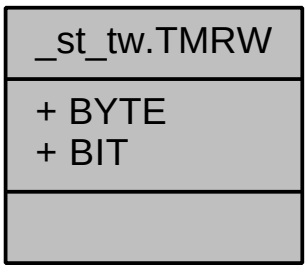
クラスメンバ

unsigned char	SVA:7	
unsigned char	FS:1	

■union _st_tw.TMRW TMRW.

3694s.h の 160 行目に定義があります。

_st_tw.TMRW 連携図

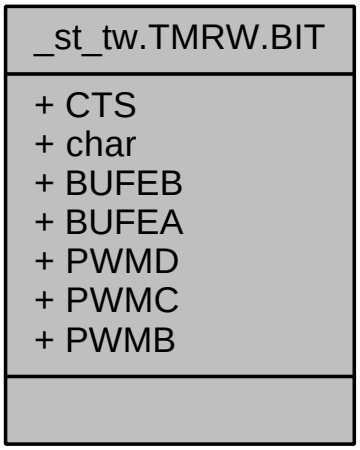


クラスメンバ

unsigned char	BYTE	Byte Access.
TMRW	BIT	Bit Access.

■struct _st_tw.TMRW.BIT 3694s.h の 164 行目に定義があります。

_st_tw.TMRW.BIT 連携図



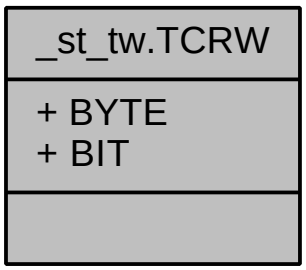
クラスメンバ

unsigned char	CTS:1	
unsigned	char:1	
unsigned char	BUFEB:1	
unsigned char	BUFEA:1	
unsigned char	PWMD:1	
unsigned char	PWMC:1	
unsigned char	PWMB:1	

■union _st_tw.TCRW TCRW.

3694s.h の 178 行目に定義があります。

_st_tw.TCRW 連携図



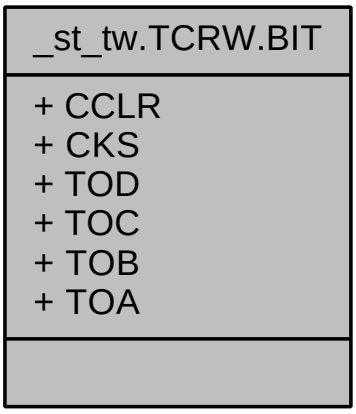
クラスメンバ

unsigned char	BYTE	Byte Access.

TCRW	BIT	Bit Access.

■struct _st_tw.TCRW.BIT 3694s.h の 182 行目に定義があります。

_st_tw.TCRW.BIT 連携図



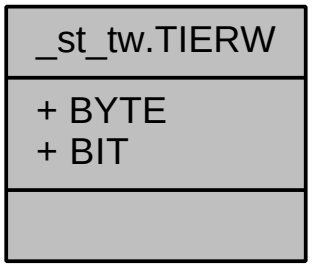
クラスメンバ

unsigned char	CCLR:1	
unsigned char	CKS:3	
unsigned char	TOD:1	
unsigned char	TOC:1	
unsigned char	TOB:1	
unsigned char	TOA:1	

■union _st_tw.TIERW TIERW.

3694s.h の 194 行目に定義があります。

_st_tw.TIERW 連携図

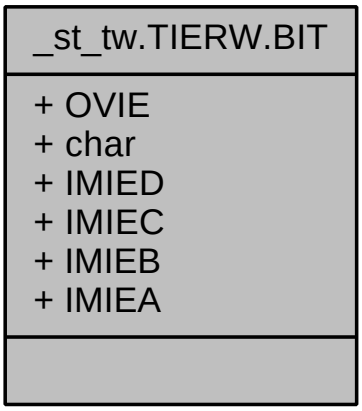


クラスメンバ

unsigned char	BYTE	Byte Access.
TIERW	BIT	Bit Access.

■struct _st_tw.TIERW.BIT 3694s.h の 198 行目に定義があります。

_st_tw.TIERW.BIT 連携図



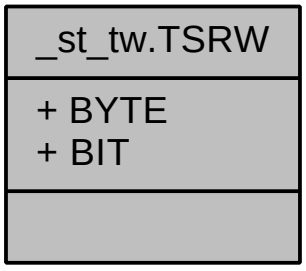
クラスメンバ

unsigned char	OVIE:1	
unsigned	char:3	
unsigned char	IMIED:1	
unsigned char	IMIEC:1	
unsigned char	IMIEB:1	
unsigned char	IMIEA:1	

■union _st_tw.TSRW TSRW.

3694s.h の 210 行目に定義があります。

_st_tw.TSRW 連携図

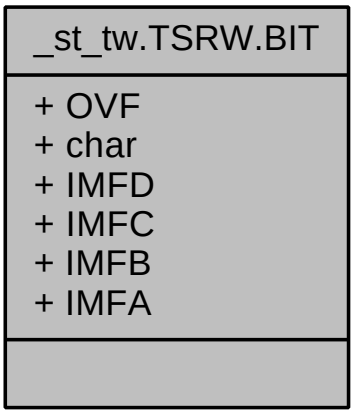


クラスメンバ

unsigned char	BYTE	Byte Access.
TSRW	BIT	Bit Access.

■struct _st_tw.TSRW.BIT 3694s.h の 214 行目に定義があります。

_st_tw.TSRW.BIT 連携図



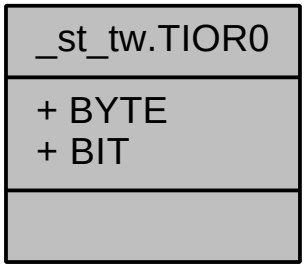
クラスメンバ

unsigned char	OVF:1	
unsigned	char:3	
unsigned char	IMFD:1	
unsigned char	IMFC:1	
unsigned char	IMFB:1	
unsigned char	IMFA:1	

■union _st_tw.TIOR0 TIOR0.

3694s.h の 226 行目に定義があります。

_st_tw.TIOR0 連携図

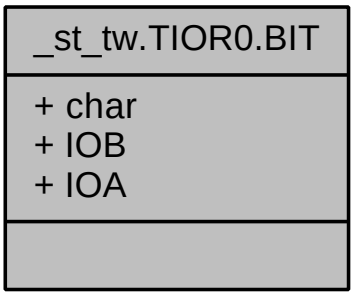


クラスメンバ

unsigned char	BYTE	Byte Access.
TIOR0	BIT	Bit Access.

■struct _st_tw.TIOR0.BIT 3694s.h の 230 行目に定義があります。

_st_tw.TIOR0.BIT 連携図



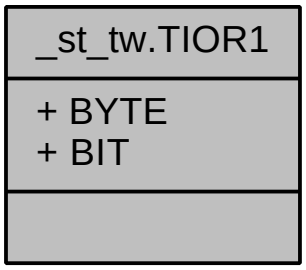
クラスメンバ

unsigned	char:1	
unsigned char	IOB:3	
unsigned char	IOA:3	

■union _st_tw.TIOR1 TIOR1.

3694s.h の 240 行目に定義があります。

_st_tw.TIOR1 連携図

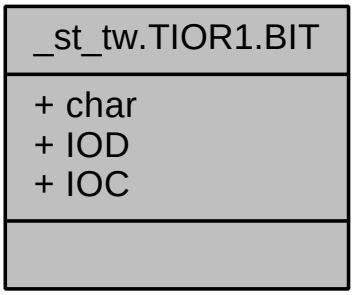


クラスメンバ

unsigned char	BYTE	Byte Access.
TIOR1	BIT	Bit Access.

■struct _st_tw.TIOR1.BIT 3694s.h の 244 行目に定義があります。

_st_tw.TIOR1.BIT 連携図

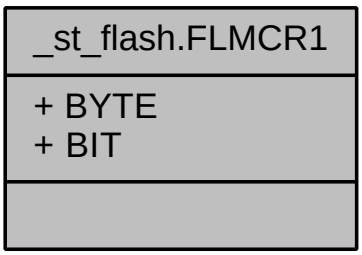


クラスメンバ

unsigned	char:1	
unsigned char	IOD:3	
unsigned char	IOC:3	

■union _st_flash.FLMCR1 3694s.h の 262 行目に定義があります。

_st_flash.FLMCR1 連携図

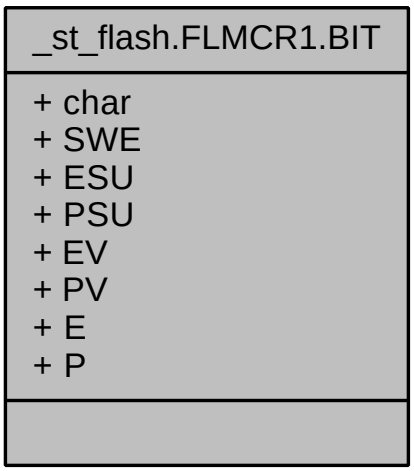


クラスメンバ

unsigned char	BYTE	Byte Access.
FLMCR1	BIT	Bit Access.

■struct _st_flash.FLMCR1.BIT 3694s.h の 266 行目に定義があります。

_st_flash.FLMCR1.BIT 連携図

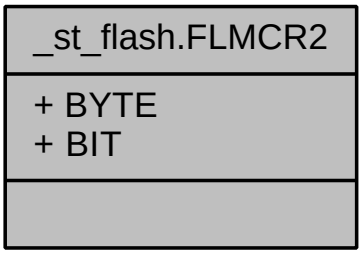


クラスメンバ

unsigned	char:1	
unsigned char	SWE:1	
unsigned char	ESU:1	
unsigned char	PSU:1	
unsigned char	EV:1	
unsigned char	PV:1	
unsigned char	E:1	
unsigned char	P:1	

■union _st_flash.FLMCR2 3694s.h の 279 行目に定義があります。

_st_flash.FLMCR2 連携図



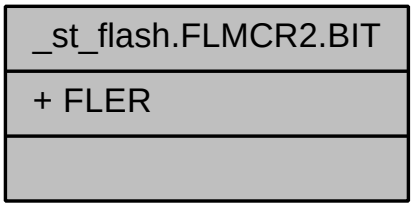
クラスメンバ

unsigned char	BYTE	Byte Access.

FLMCR2	BIT	Bit Access.

■struct _st_flash.FLMCR2.BIT 3694s.h の 283 行目に定義があります。

_st_flash.FLMCR2.BIT 連携図

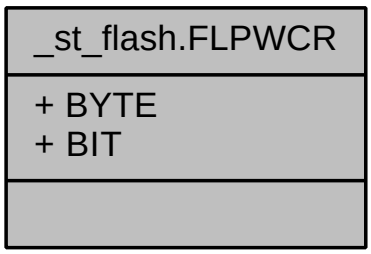


クラスメンバ

unsigned char	FLER:1	

■union _st_flash.FLPWCR 3694s.h の 289 行目に定義があります。

_st_flash.FLPWCR 連携図

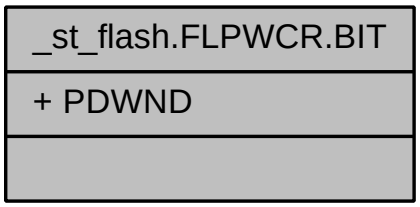


クラスメンバ

unsigned char	BYTE	Byte Access.
FLPWCR	BIT	Bit Access.

■struct _st_flash.FLPWCR.BIT 3694s.h の 293 行目に定義があります。

_st_flash.FLPWCR.BIT 連携図

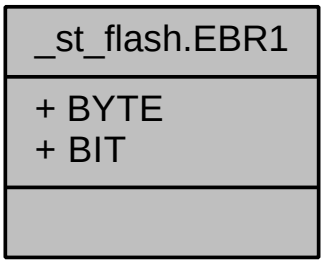


クラスメンバ

unsigned char	PDWND:1	

■union _st_flash.EBR1 3694s.h の 299 行目に定義があります。

_st_flash.EBR1 連携図

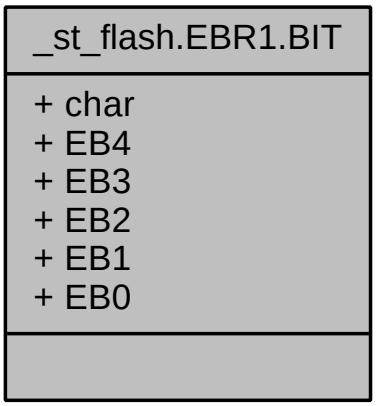


クラスメンバ

unsigned char	BYTE	Byte Access.
EBR1	BIT	Bit Access.

■struct _st_flash.EBR1.BIT 3694s.h の 303 行目に定義があります。

_st_flash.EBR1.BIT 連携図



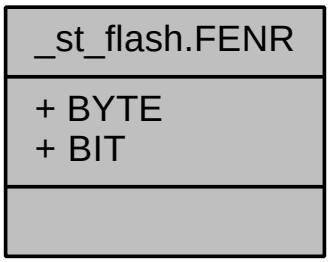
クラスメンバ

unsigned	char:3	
unsigned char	EB4:1	
unsigned char	EB3:1	
unsigned char	EB2:1	

unsigned char	EB1:1	
unsigned char	EB0:1	

■union _st_flash.FENR 3694s.h の 315 行目に定義があります。

_st_flash.FENR 連携図

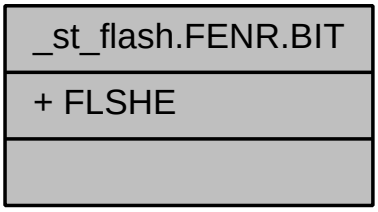


クラスメンバ

unsigned char	BYTE	Byte Access.
FENR	BIT	Bit Access.

■struct _st_flash.FENR.BIT 3694s.h の 319 行目に定義があります。

_st_flash.FENR.BIT 連携図

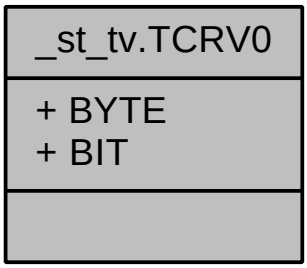


クラスメンバ

unsigned char	FLSHE:1	

■union _st_tv.TCRV0 3694s.h の 329 行目に定義があります。

_st_tv.TCRV0 連携図

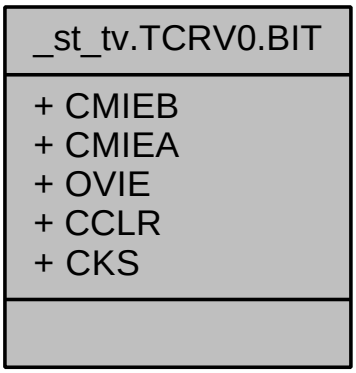


クラスメンバ

unsigned char	BYTE	Byte Access.
TCRV0	BIT	Bit Access.

■struct _st_tv.TCRV0.BIT 3694s.h の 333 行目に定義があります。

_st_tv.TCRV0.BIT 連携図

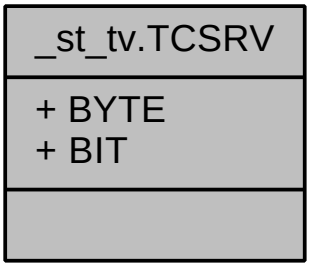


クラスメンバ

unsigned char	CMIEB:1	
unsigned char	CMIEA:1	
unsigned char	OVIE:1	
unsigned char	CCLR:2	
unsigned char	CKS:3	

■union _st_tv.TCSRV 3694s.h の 343 行目に定義があります。

_st_tv.TCSRV 連携図



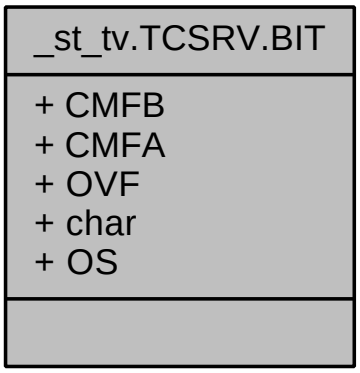
クラスメンバ

unsigned char	BYTE	Byte Access.

TCSRV	BIT	Bit Access.

■struct _st_tv.TCSRV.BIT 3694s.h の 347 行目に定義があります。

_st_tv.TCSRV.BIT 連携図

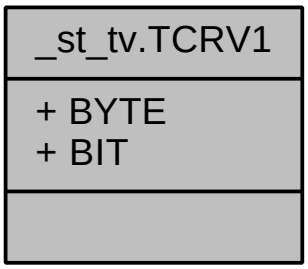


クラスメンバ

unsigned char	CMFB:1	
unsigned char	CMFA:1	
unsigned char	OVF:1	
unsigned	char:1	
unsigned char	OS:4	

■union _st_tv.TCRV1 3694s.h の 360 行目に定義があります。

_st_tv.TCRV1 連携図

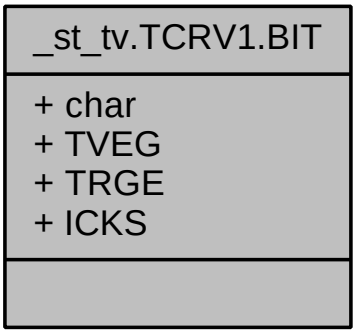


クラスメンバ

unsigned char	BYTE	Byte Access.
TCRV1	BIT	Bit Access.

■struct _st_tv.TCRV1.BIT 3694s.h の 364 行目に定義があります。

_st_tv.TCRV1.BIT 連携図

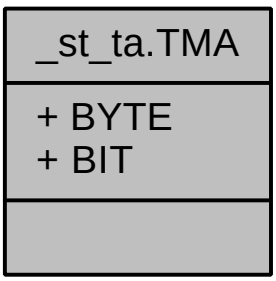


クラスメンバ

unsigned	char:3	
unsigned char	TVEG:2	
unsigned char	TRGE:1	
unsigned char	ICKS:1	

■union _st_ta.TMA 3694s.h の 378 行目に定義があります。

_st_ta.TMA 連携図

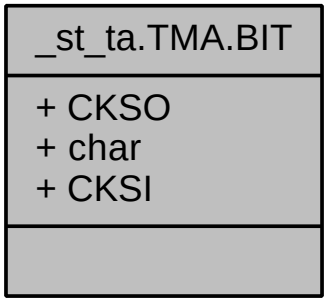


クラスメンバ

unsigned char	BYTE	Byte Access.
TMA	BIT	Bit Access.

■struct _st_ta.TMA.BIT 3694s.h の 382 行目に定義があります。

_st_ta.TMA.BIT 連携図

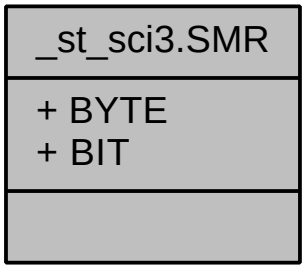


クラスメンバ

unsigned char	CKSO:3	
unsigned	char:1	
unsigned char	CKSI:4	

■union _st_sci3.SMR 3694s.h の 394 行目に定義があります。

_st_sci3.SMR 連携図

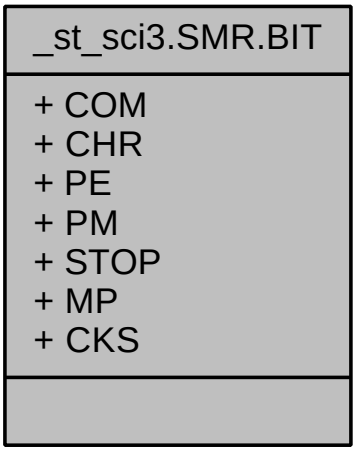


クラスメンバ

unsigned char	BYTE	Byte Access.
SMR	BIT	

■struct _st_sci3.SMR.BIT 3694s.h の 397 行目に定義があります。

_st_sci3.SMR.BIT 連携図

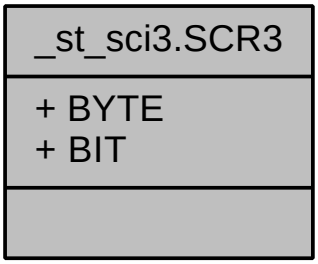


クラスメンバ

unsigned char	COM:1	
unsigned char	CHR:1	
unsigned char	PE:1	
unsigned char	PM:1	
unsigned char	STOP:1	
unsigned char	MP:1	
unsigned char	CKS:2	

■union _st_sci3.SCR3　3694s.h の 409 行目に定義があります。

_st_sci3.SCR3 連携図

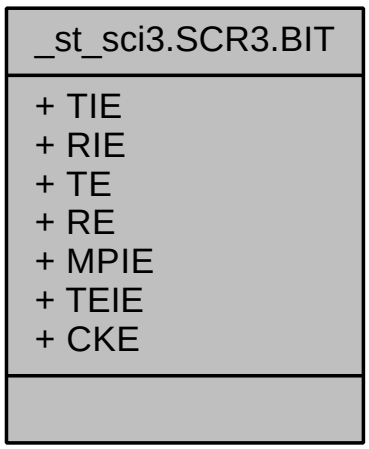


クラスメンバ

unsigned char	BYTE	
SCR3	BIT	

■struct _st_sci3.SCR3.BIT　3694s.h の 411 行目に定義があります。

_st_sci3.SCR3.BIT 連携図

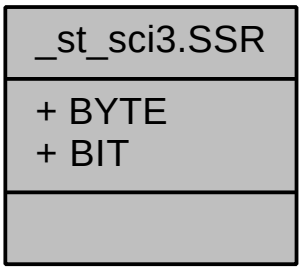


クラスメンバ

unsigned char	TIE:1	
unsigned char	RIE:1	
unsigned char	TE:1	
unsigned char	RE:1	
unsigned char	MPIE:1	
unsigned char	TEIE:1	
unsigned char	CKE:2	

■union _st_sci3.SSR 3694s.h の 422 行目に定義があります。

_st_sci3.SSR 連携図

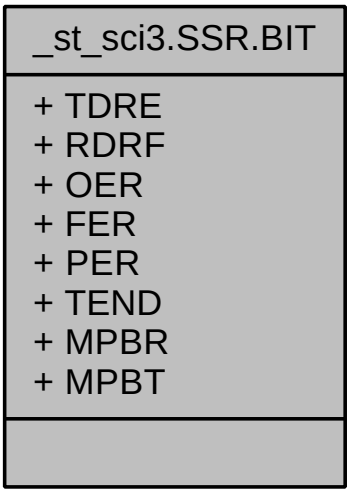


クラスメンバ

unsigned char	BYTE	
SSR	BIT	

■struct _st_sci3.SSR.BIT 3694s.h の 424 行目に定義があります。

_st_sci3.SSR.BIT 連携図

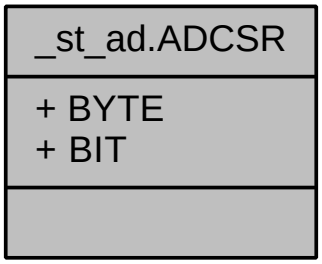


クラスメンバ

unsigned char	TDRE:1	
unsigned char	RDRF:1	
unsigned char	OER:1	
unsigned char	FER:1	
unsigned char	PER:1	
unsigned char	TEND:1	
unsigned char	MPBR:1	
unsigned char	MPBT:1	

■union _st_ad.ADCSR 3694s.h の 443 行目に定義があります。

_st_ad.ADCSR 連携図

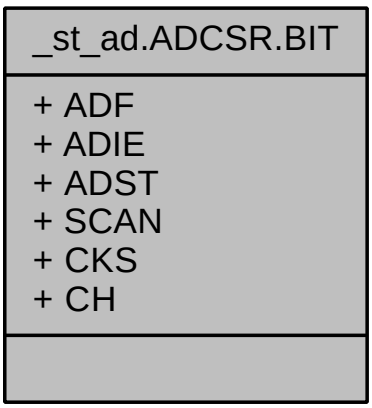


クラスメンバ

unsigned char	BYTE	Byte Access.
ADCSR	BIT	

■struct _st_ad.ADCSR.BIT 3694s.h の 446 行目に定義があります。

_st_ad.ADCSR.BIT 連携図

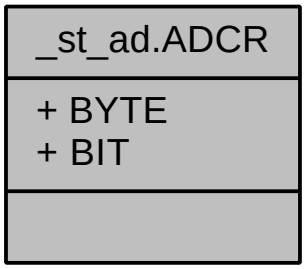


クラスメンバ

unsigned char	ADF:1	
unsigned char	ADIE:1	
unsigned char	ADST:1	
unsigned char	SCAN:1	
unsigned char	CKS:1	
unsigned char	CH:3	

■union _st_ad.ADCR 3694s.h の 456 行目に定義があります。

_st_ad.ADCR 連携図

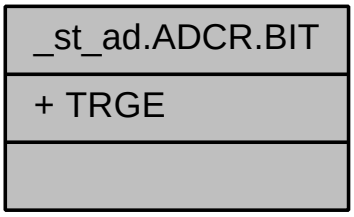


クラスメンバ

unsigned char	BYTE	
ADCR	BIT	

■struct _st_ad.ADCR.BIT 3694s.h の 458 行目に定義があります。

_st_ad.ADCR.BIT 連携図

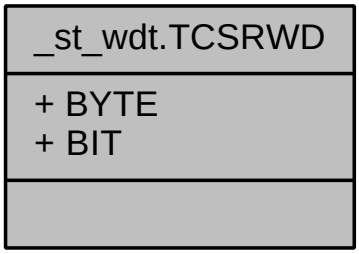


クラスメンバ

unsigned char	TRGE:1	

■union _st_wdt.TCSRWD 3694s.h の 465 行目に定義があります。

_st_wdt.TCSRWD 連携図

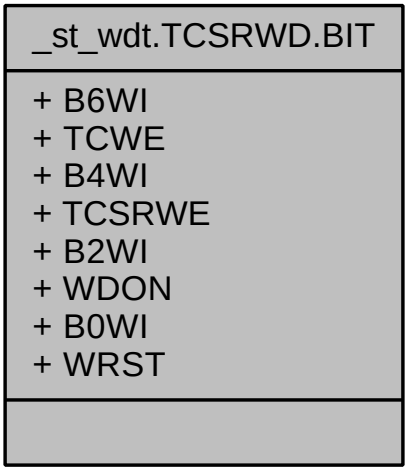


クラスメンバ

unsigned char	BYTE	Byte Access.
TCSRWD	BIT	

■struct _st_wdt.TCSRWD.BIT 3694s.h の 468 行目に定義があります。

_st_wdt.TCSRWD.BIT 連携図

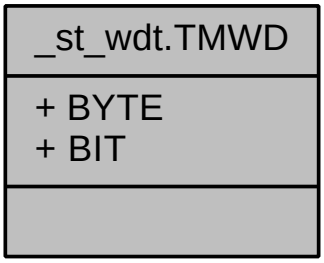


クラスメンバ

unsigned char	B6WI:1	
unsigned char	TCWE:1	
unsigned char	B4WI:1	
unsigned char	TCSRWE:1	
unsigned char	B2WI:1	
unsigned char	WDON:1	
unsigned char	B0WI:1	
unsigned char	WRST:1	

■union _st_wdt.TMWD 3694s.h の 481 行目に定義があります。

_st_wdt.TMWD 連携図

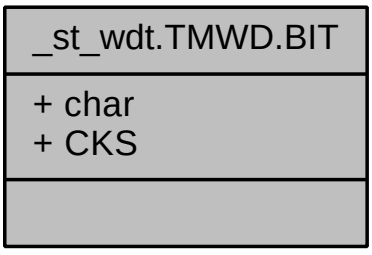


クラスメンバ

unsigned char	BYTE	Byte Access.
TMWD	BIT	

■struct _st_wdt.TMWD.BIT 3694s.h の 484 行目に定義があります。

_st_wdt.TMWD.BIT 連携図

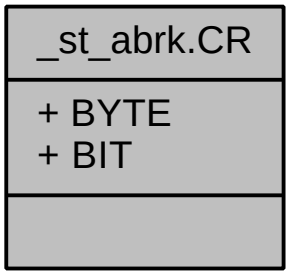


クラスメンバ

unsigned	char:4	
unsigned char	CKS:4	

■union _st_abrk.CR 3694s.h の 493 行目に定義があります。

_st_abrk.CR 連携図

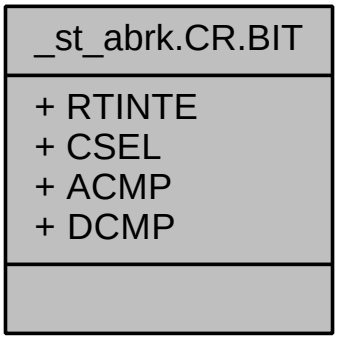


クラスメンバ

unsigned char	BYTE	Byte Access.
CR	BIT	

■struct _st_abrk.CR.BIT 3694s.h の 496 行目に定義があります。

_st_abrk.CR.BIT 連携図

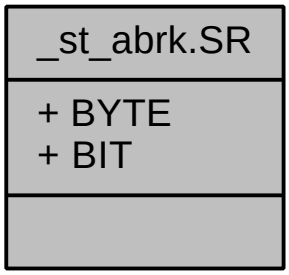


クラスメンバ

unsigned char	RTINTE:1	
unsigned char	CSEL:2	
unsigned char	ACMP:3	
unsigned char	DCMP:2	

■union _st_abrk.SR 3694s.h の 504 行目に定義があります。

_st_abrk.SR 連携図

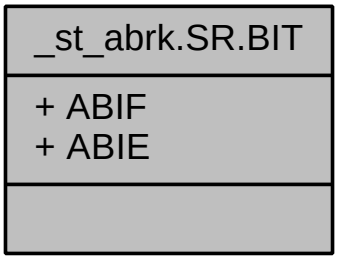


クラスメンバ

unsigned char	BYTE	Byte Access.
SR	BIT	

■struct _st_abrk.SR.BIT 3694s.h の 507 行目に定義があります。

_st_abrk.SR.BIT 連携図

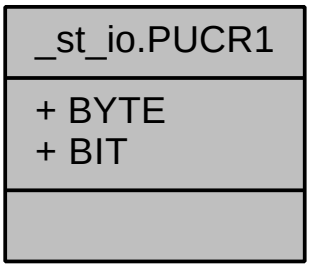


クラスメンバ

unsigned char	ABIF:1	
unsigned char	ABIE:1	

■union _st_io.PUCR1 3694s.h の 518 行目に定義があります。

_st_io.PUCR1 連携図

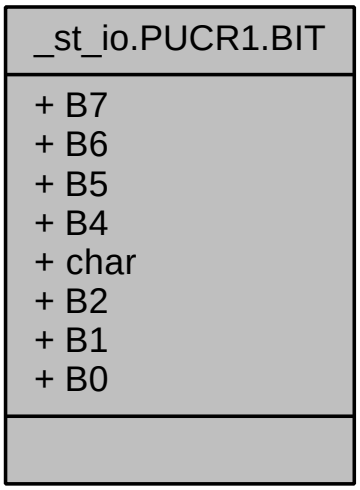


クラスメンバ

unsigned char	BYTE	Byte Access.
PUCR1	BIT	

■struct _st_io.PUCR1.BIT 3694s.h の 521 行目に定義があります。

_st_io.PUCR1.BIT 連携図

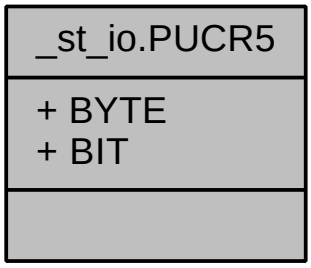


クラスメンバ

unsigned char	B7:1	
unsigned char	B6:1	
unsigned char	B5:1	
unsigned char	B4:1	
unsigned	char:1	
unsigned char	B2:1	
unsigned char	B1:1	
unsigned char	B0:1	

■union _st_io.PUCR5 3694s.h の 533 行目に定義があります。

_st_io.PUCR5 連携図

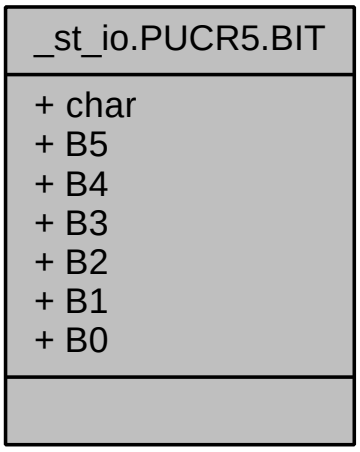


クラスメンバ

unsigned char	BYTE	Byte Access.
PUCR5	BIT	

■struct _st_io.PUCR5.BIT 3694s.h の 536 行目に定義があります。

_st_io.PUCR5.BIT 連携図

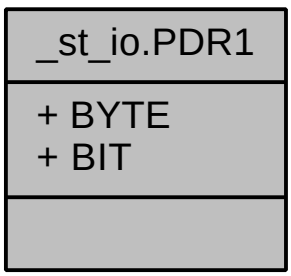


クラスメンバ

unsigned	char:2	
unsigned char	B5:1	
unsigned char	B4:1	
unsigned char	B3:1	
unsigned char	B2:1	
unsigned char	B1:1	
unsigned char	B0:1	

■union _st_io.PDR1 3694s.h の 548 行目に定義があります。

_st_io.PDR1 連携図

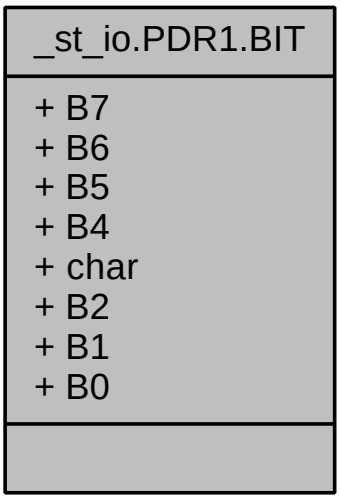


クラスメンバ

unsigned char	BYTE	Byte Access.
PDR1	BIT	

■struct _st_io.PDR1.BIT 3694s.h の 551 行目に定義があります。

_st_io.PDR1.BIT 連携図



クラスメンバ

unsigned char	B7:1	
unsigned char	B6:1	
unsigned char	B5:1	
unsigned char	B4:1	
unsigned	char:1	
unsigned char	B2:1	
unsigned char	B1:1	
unsigned char	B0:1	

■union _st_io.PDR2 3694s.h の 563 行目に定義があります。

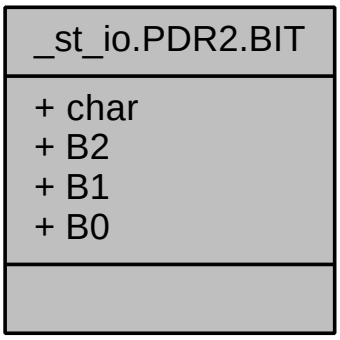
_st_io.PDR2 連携図

クラスメンバ

unsigned char	BYTE	Byte Access.
PDR2	BIT	

■struct _st_io.PDR2.BIT 3694s.h の 566 行目に定義があります。

_st_io.PDR2.BIT 連携図

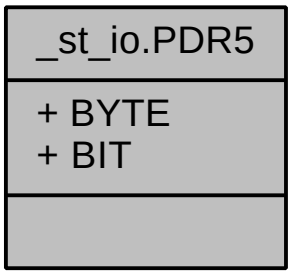


クラスメンバ

unsigned	char:5	
unsigned char	B2:1	
unsigned char	B1:1	
unsigned char	B0:1	

■union _st_io.PDR5 3694s.h の 575 行目に定義があります。

_st_io.PDR5 連携図

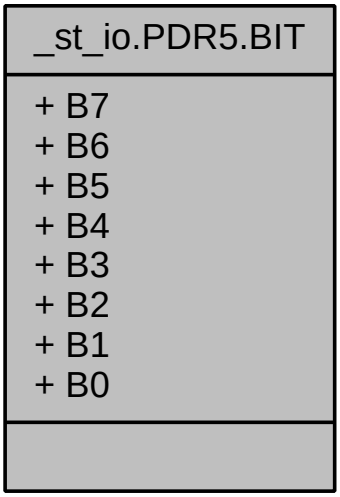


クラスメンバ

unsigned char	BYTE	Byte Access.
PDR5	BIT	

■struct _st_io.PDR5.BIT 3694s.h の 578 行目に定義があります。

_st_io.PDR5.BIT 連携図



クラスメンバ

unsigned char	B7:1	
unsigned char	B6:1	
unsigned char	B5:1	
unsigned char	B4:1	
unsigned char	B3:1	
unsigned char	B2:1	
unsigned char	B1:1	
unsigned char	B0:1	

■union _st_io.PDR7 3694s.h の 591 行目に定義があります。

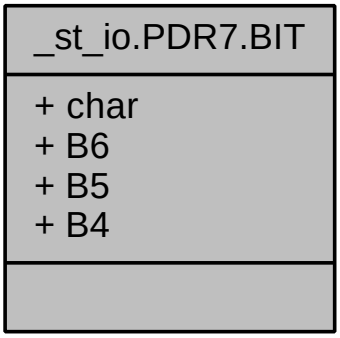
_st_io.PDR7 連携図

クラスメンバ

unsigned char	BYTE	Byte Access.
PDR7	BIT	

■struct _st_io.PDR7.BIT 3694s.h の 594 行目に定義があります。

_st_io.PDR7.BIT 連携図



クラスメンバ

unsigned	char:1	
unsigned char	B6:1	
unsigned char	B5:1	
unsigned char	B4:1	

■union _st_io.PDR8　3694s.h の 602 行目に定義があります。

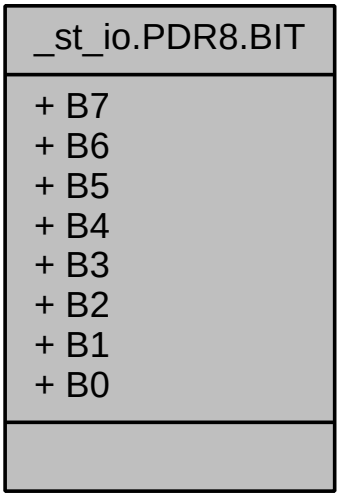
_st_io.PDR8 連携図

クラスメンバ

unsigned char	BYTE	Byte Access.
PDR8	BIT	

■struct _st_io.PDR8.BIT　3694s.h の 605 行目に定義があります。

_st_io.PDR8.BIT 連携図

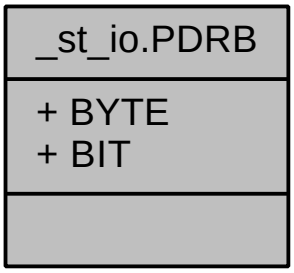


クラスメンバ

unsigned char	B7:1	
unsigned char	B6:1	
unsigned char	B5:1	
unsigned char	B4:1	
unsigned char	B3:1	
unsigned char	B2:1	
unsigned char	B1:1	
unsigned char	B0:1	

■union _st_io.PDRB 3694s.h の 618 行目に定義があります。

_st_io.PDRB 連携図

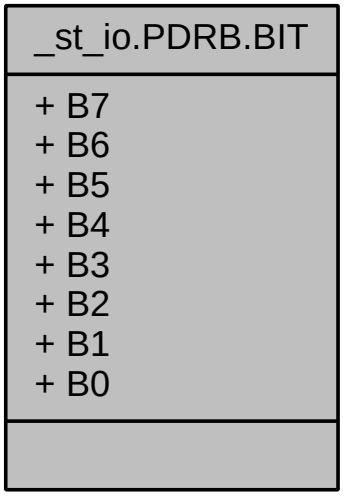


クラスメンバ

unsigned char	BYTE	Byte Access.
PDRB	BIT	

■struct _st_io.PDRB.BIT 3694s.h の 621 行目に定義があります。

_st_io.PDRB.BIT 連携図

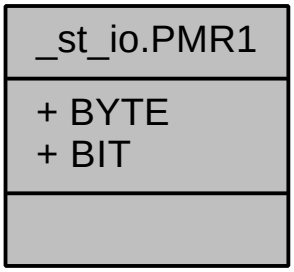


クラスメンバ

unsigned char	B7:1	
unsigned char	B6:1	
unsigned char	B5:1	
unsigned char	B4:1	
unsigned char	B3:1	
unsigned char	B2:1	
unsigned char	B1:1	
unsigned char	B0:1	

■union _st_io.PMR1 3694s.h の 634 行目に定義があります。

_st_io.PMR1 連携図

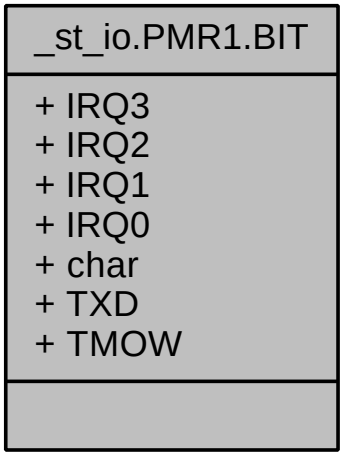


クラスメンバ

unsigned char	BYTE	Byte Access.
PMR1	BIT	

■struct _st_io.PMR1.BIT 3694s.h の 637 行目に定義があります。

_st_io.PMR1.BIT 連携図

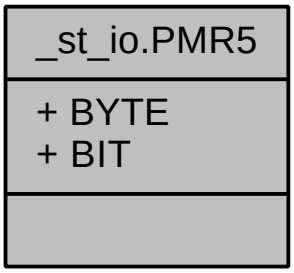


クラスメンバ

unsigned char	IRQ3:1	
unsigned char	IRQ2:1	
unsigned char	IRQ1:1	
unsigned char	IRQ0:1	
unsigned	char:2	
unsigned char	TXD:1	
unsigned char	TMOW:1	

■union _st_io.PMR5 3694s.h の 648 行目に定義があります。

_st_io.PMR5 連携図

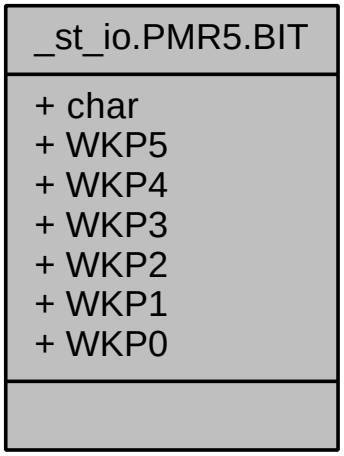


クラスメンバ

unsigned char	BYTE	Byte Access.
PMR5	BIT	

■struct _st_io.PMR5.BIT 3694s.h の 651 行目に定義があります。

_st_io.PMR5.BIT 連携図

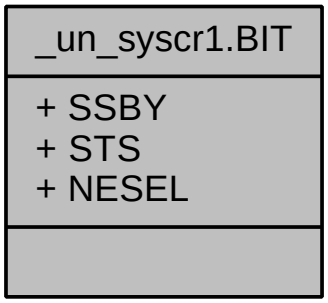


クラスメンバ

unsigned	char:2	
unsigned char	WKP5:1	
unsigned char	WKP4:1	
unsigned char	WKP3:1	
unsigned char	WKP2:1	
unsigned char	WKP1:1	
unsigned char	WKP0:1	

■struct _un_syscr1.BIT 3694s.h の 676 行目に定義があります。

_un_syscr1.BIT 連携図

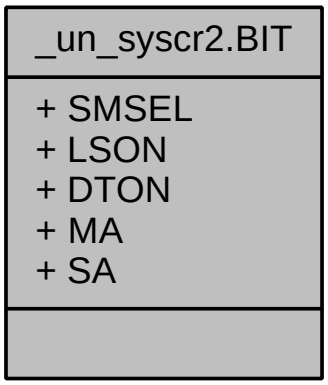


クラスメンバ

unsigned char	SSBY:1	
unsigned char	STS:3	
unsigned char	NESEL:1	

■struct _un_syscr2.BIT 3694s.h の 687 行目に定義があります。

_un_syscr2.BIT 連携図

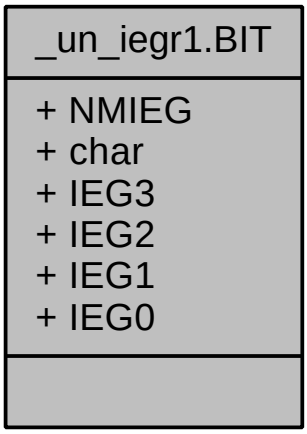


クラスメンバ

unsigned char	SMSEL:1	
unsigned char	LSON:1	
unsigned char	DTON:1	
unsigned char	MA:3	
unsigned char	SA:2	

■struct _un_iegr1.BIT 3694s.h の 700 行目に定義があります。

_un_iegr1.BIT 連携図

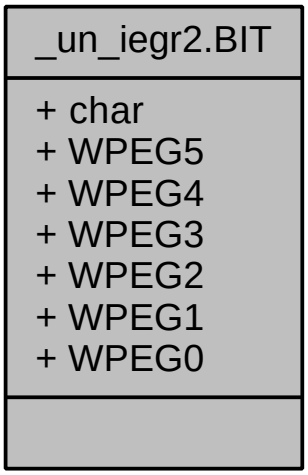


クラスメンバ

unsigned char	NMIEG:1	
unsigned	char:3	
unsigned char	IEG3:1	
unsigned char	IEG2:1	
unsigned char	IEG1:1	
unsigned char	IEG0:1	

■struct _un_iegr2.BIT 3694s.h の 714 行目に定義があります。

_un_iegr2.BIT 連携図

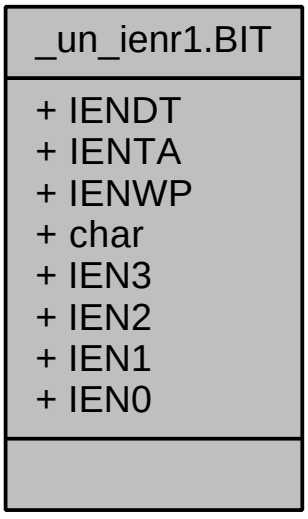


クラスメンバ

unsigned	char:2	
unsigned char	WPEG5:1	
unsigned char	WPEG4:1	
unsigned char	WPEG3:1	
unsigned char	WPEG2:1	
unsigned char	WPEG1:1	
unsigned char	WPEG0:1	

■struct _un_ienr1.BIT 3694s.h の 729 行目に定義があります。

_un_ienr1.BIT 連携図

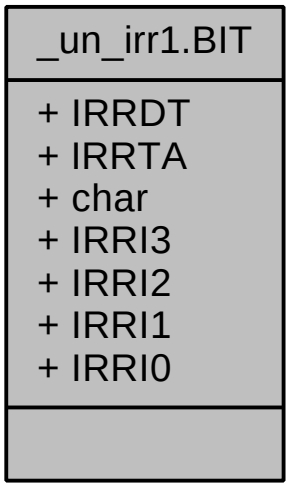


クラスメンバ

unsigned char	IENDT:1	
unsigned char	IENTA:1	
unsigned char	IENWP:1	
unsigned	char:1	
unsigned char	IEN3:1	
unsigned char	IEN2:1	
unsigned char	IEN1:1	
unsigned char	IEN0:1	

■struct _un_irr1.BIT 3694s.h の 745 行目に定義があります。

_un_irr1.BIT 連携図

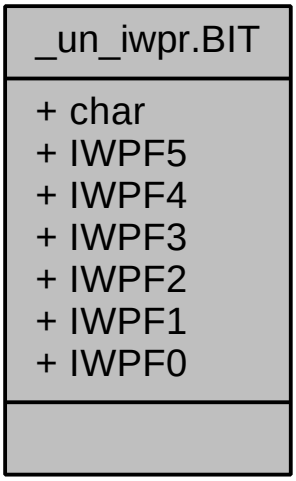


クラスメンバ

unsigned char	IRRDT:1	
unsigned char	IRRTA:1	
unsigned	char:2	
unsigned char	IRRI3:1	
unsigned char	IRRI2:1	
unsigned char	IRRI1:1	
unsigned char	IRRI0:1	

■struct _un_iwpr.BIT 3694s.h の 760 行目に定義があります。

_un_iwpr.BIT 連携図

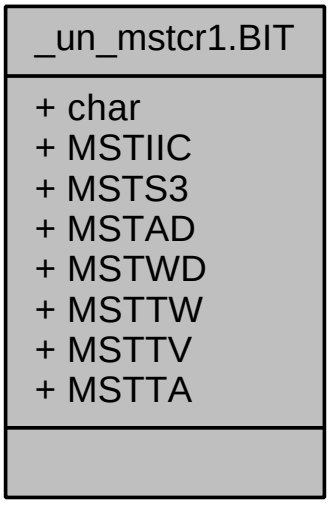


クラスメンバ

unsigned	char:2	
unsigned char	IWPF5:1	
unsigned char	IWPF4:1	
unsigned char	IWPF3:1	
unsigned char	IWPF2:1	
unsigned char	IWPF1:1	
unsigned char	IWPF0:1	

■struct _un_mstcr1.BIT 3694s.h の 775 行目に定義があります。

_un_mstcr1.BIT 連携図



クラスメンバ

unsigned	char:1	
unsigned char	MSTIIC:1	
unsigned char	MSTS3:1	
unsigned char	MSTAD:1	
unsigned char	MSTWD:1	
unsigned char	MSTTW:1	
unsigned char	MSTTV:1	
unsigned char	MSTTA:1	

3.1.3 マクロ定義詳解

■#define _LVD (∗(volatile struct **_st_lvd** ∗)0xF730) LVD Address.
3694s.h の 789 行目に定義があります。

■#define _IIC2 (∗(volatile struct **_st_iic2** ∗)0xF748) IIC2 Address.
3694s.h の 791 行目に定義があります。

■#define _TW (∗(volatile struct **_st_tw** ∗)0xFF80) TW Address.
3694s.h の 793 行目に定義があります。

■#define _FLASH (∗(volatile struct **_st_flash** ∗)0xFF90) FLASH Address.
3694s.h の 795 行目に定義があります。

■#define _TV (∗(volatile struct **_st_tv** ∗)0xFFA0) TV Address.
3694s.h の 797 行目に定義があります。

■#define _TA (∗(volatile struct **_st_ta** ∗)0xFFA6) TA Address.
3694s.h の 799 行目に定義があります。

■#define _SCI3 (∗(volatile struct **_st_sci3** ∗)0xFFA8) SCI3 Address.
3694s.h の 801 行目に定義があります。

■#define _AD (∗(volatile struct **_st_ad** ∗)0xFFB0) A/D Address.
3694s.h の 803 行目に定義があります。

■#define _WDT (∗(volatile struct **_st_wdt** ∗)0xFFC0) WDT Address.
3694s.h の 805 行目に定義があります。

■#define _ABRK (∗(volatile struct **_st_abrk** ∗)0xFFC8) ABRK Address.
3694s.h の 807 行目に定義があります。

■#define _IO (∗(volatile struct **_st_io** ∗)0xFFD0) IO Address.
3694s.h の 809 行目に定義があります。

■#define _SYSCR1 (∗(volatile union **_un_syscr1**∗)0xFFF0) SYSCR1 Address.
3694s.h の 811 行目に定義があります。

■#define _SYSCR2 (∗(volatile union **_un_syscr2**∗)0xFFF1) SYSCR2Address.

3694s.h の 813 行目に定義があります。

■#define _IEGR1 (*(volatile union _**un_iegr1** *)0xFFF2) IEGR1 Address.
3694s.h の 815 行目に定義があります。

■#define _IEGR2 (*(volatile union _**un_iegr2** *)0xFFF3) IEGR2 Address.
3694s.h の 817 行目に定義があります。

■#define _IENR1 (*(volatile union _**un_ienr1** *)0xFFF4) IENR1 Address.
3694s.h の 819 行目に定義があります。

■#define _IRR1 (*(volatile union _**un_irr1** *)0xFFF6) IRR1 Address.
3694s.h の 821 行目に定義があります。

■#define _IWPR (*(volatile union _**un_iwpr** *)0xFFF8) IWPR Address.
3694s.h の 823 行目に定義があります。

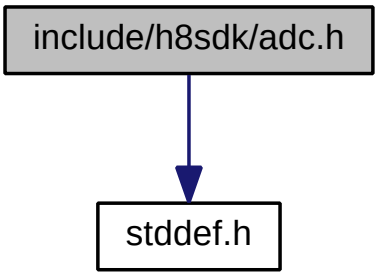
■#define _MSTCR1 (*(volatile union _**un_mstcr1***)0xFFF9) MSTCR1 Address.
3694s.h の 825 行目に定義があります。

3.2 include/h8sdk/adc.h ファイル

A/D コンバータドライバ

```
#include "stddef.h"
```

adc.h の依存先関係図:



列挙型

- enum ADC_Mode {
  ADC_NORMAL = 0,
  ADC_SCAN,
  ADC_NUM_OF_MODE }
  設定モード識別子
- enum ADC_Channel {
  ADC_AN0 = 0,
  ADC_AN1,
  ADC_AN2,
  ADC_AN3,
  ADC_NUM_OF_CHANNEL }

チャネル識別子

関数

- void ADC_init (ADC_Mode mode, _BOOL interrupt)
  A/D **コンバータ初期化**
- void ADC_enable (ADC_Channel ch)
  チャネル有効化
- void ADC_disable (ADC_Channel ch)
  チャネル無効化
- void ADC_start ()
  A/D **変換開始**
- void ADC_stop ()
  A/D **変換停止**
- _SDWORD ADC_get (ADC_Channel anx, _BOOL last)
  A/D **変換データを取得する**

3.2.1 詳解

A/D コンバータドライバ

このモジュールはA/D コンバータへのインタフェースを提供する。

初期化、使用チャネルの設定を行うことで変換データを取得できるように なる。

参照

H83694 グループ_ハードウェアマニュアル.pdf 16 章

3.2.2 列挙型詳解

■enum **ADC_Mode** 設定モード識別子

列挙値

ADC_NORMAL ノーマルモード

ADC_SCAN スキャンモード

ADC_NUM_OF_MODE モード数

adc.h の 34 行目に定義があります。

```
35 {
36     ADC_NORMAL = 0,
38     ADC_SCAN,
40     ADC_NUM_OF_MODE
42 } ADC_Mode;
```

■enum **ADC_Channel** チャネル識別子

列挙値

ADC_AN0 チャネルポートAN0

ADC_AN1 チャネルポートAN1

ADC_AN2 チャネルポートAN2

ADC_AN3 チャネルポートAN3

ADC_NUM_OF_CHANNEL チャネルポート数

adc.h の 47 行目に定義があります。

```
48 {
49     ADC_AN0 = 0,
51     ADC_AN1,
53     ADC_AN2,
55     ADC_AN3,
57     ADC_NUM_OF_CHANNEL
59 } ADC_Channel;
```

3.2.3 関数詳解

■void ADC_init (**ADC_Mode** *mode*, **_BOOL** *interrupt*) A/D コンバータ初期化

A/D コンバータ初期化。他API 使用前に必ず実行する。

引数

in	*mode*	使用するモードの種類
in	*interrupt*	_TRUE：割り込みを使う。 _FALSE：割り込みを使わない。

覚え書き

割り込みハンドラは外部で定義する。

■void ADC_enable (**ADC_Channel** *ch*) チャネル有効化

使用するチャネルを設定する。ノーマルモード時は指定チャネルのみ。ス キャンモード時は指定チャネルより若い番号のチャネルは全て使用可能に セットされる。例えば、ADC_AN2 を与えた場合 AN0〜AN2 のチャネルが有効 になる。

引数

in	*ch*	有効にするチャネル

参照

ADC_disable

■void ADC_disable (**ADC_Channel** *ch*) チャネル無効化

使用停止するチャネルを設定する。ノーマルモード時は指定チャネルのみ。 指定チャネルより若い番号のチャネルは全て使用不可にセットされる。例 えば、ADC_AN2 を与えた場合 AN0〜AN2 のチャネルが無効になる。

引数

in	*ch*	無効にするチャネル

参照

ADC_enable

■void ADC_start ( ) A/D 変換開始

A/D 変換が開始される。スキャンモード時は変換データが随時取得できるよ うになる。ノーマルモード時は一回のみ。

参照

ADC_stop

■void ADC_stop ( ) A/D 変換停止

A/D 変換が停止される。以降変換データは取得できない。

参照

ADC_start

■**_SDWORD** ADC_get (**ADC_Channel** *anx*, **_BOOL** *last*) A/D 変換データを取得する

A/D 変換データを取得する。読み込むチャネルが全て完了したときは必ず引 数 last に_TRUE をセットして実行する。

引数

in	*anx*	データを取得するチャネル
in	*last*	この取得で今回の変換データを全チャネルで破棄するかどうか

戻り値

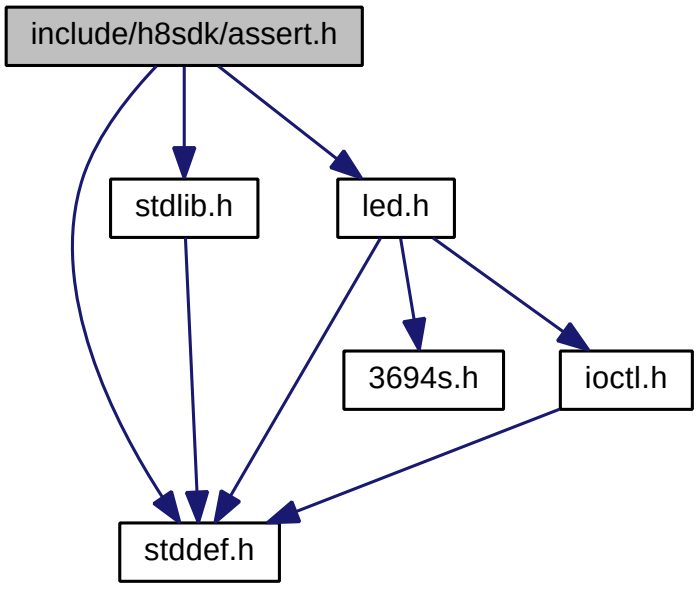
正の値	変換完了データ。有効値は 10bit
-1	変換処理が未完

3.3 include/h8sdk/assert.h ファイル

アサート

```
#include "stddef.h"
#include "stdlib.h"
#include "led.h"
```

assert.h の依存先関係図:



マクロ定義

- #define _assert(x)
  アサート

3.3.1 詳解

アサート

3.3.2 マクロ定義詳解

■#define _assert(x) **値:**

```
if (!(x))                                   \
    {                                           \
        LED_INIT;                               \
        while (_TRUE)                           \
        {                                       \
            LED_ON(LED_D5);                     \
            LED_OFF(LED_D6);                    \
            _msleep(500);                       \
            LED_OFF(LED_D5);                    \
            LED_ON(LED_D6);                     \
            _msleep(500);                       \
        }                                       \
    }
```

アサート

実装ミス以外ではありえないバグをチェックするための実行時エラーチェッ カ。デバッグ専用。リリースバイナリからは外される。

引数

in	x	評価するパラメータ。これが偽になると処理を緊急停止して LED 点滅で状態異常を知らせる。

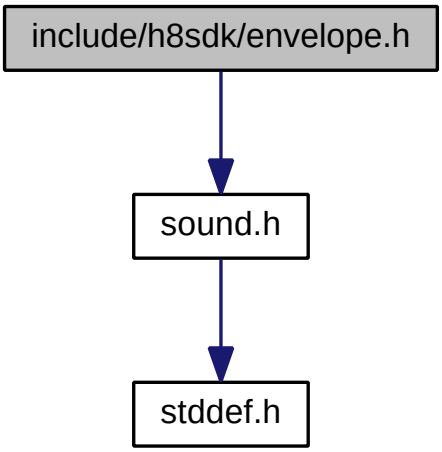
assert.h の 38 行目に定義があります。

3.4 include/h8sdk/envelope.h ファイル

エンベロープデータ定義

```
#include "sound.h"
```

envelope.h の依存先関係図:



変数

- const SOUND_Envelope ENVELOPE_na
  無音
- const SOUND_Envelope ENVELOPE_piano
  ピアノ
- const SOUND_Envelope ENVELOPE_piano_reverb
  リバーブ *(***ピアノ***)*
- const SOUND_Envelope ENVELOPE_flute
  フルート
- const SOUND_Envelope ENVELOPE_flute_reverb
  リバーブ *(***フルート***)*
- const SOUND_Envelope ENVELOPE_drum
  ドラム
- const SOUND_Envelope ENVELOPE_trumpet
  トランペット
- const SOUND_Envelope ENVELOPE_trumpet_reverb
  リバーブ *(***トランペット***)*
- const SOUND_Envelope ENVELOPE_harp
  ハープ
- const SOUND_Envelope ENVELOPE_harp_reverb
  リバーブ *(***ハープ***)*

3.4.1 詳解

エンベロープデータ定義

各種エンベロープの時系列変化データが定義されている。 192 個で 4 分音符分

3.4.2 変数詳解

■const **SOUND_Envelope** ENVELOPE_na 無音

■const **SOUND_Envelope** ENVELOPE_piano ピアノ

■const **SOUND_Envelope** ENVELOPE_piano_reverb リバーブ (ピアノ)

■const **SOUND_Envelope** ENVELOPE_flute フルート

■const **SOUND_Envelope** ENVELOPE_flute_reverb リバーブ (フルート)

■const **SOUND_Envelope** ENVELOPE_drum ドラム

■const **SOUND_Envelope** ENVELOPE_trumpet トランペット

■const **SOUND_Envelope** ENVELOPE_trumpet_reverb リバーブ (トランペット)

■const **SOUND_Envelope** ENVELOPE_harp ハープ

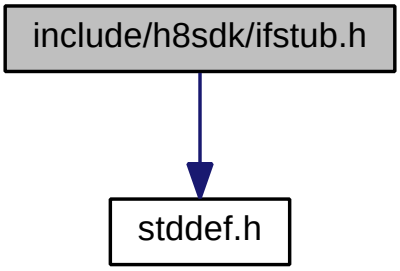
■const **SOUND_Envelope** ENVELOPE_harp_reverb リバーブ (ハープ)

3.5 include/h8sdk/ifstub.h ファイル

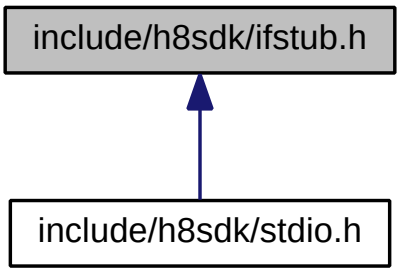
通信インタフェーススタブ

```
#include "stddef.h"
```

ifstub.h の依存先関係図:



被依存関係図:



クラス

- struct IFSTUB_Class
  スタブ本体型

型定義

- typedef _SINT(* IFSTUB_WriteStream) (const _UBYTE *data, _UBYTE size, _BOOL sync, _SINT tmo_ms)
  バイトストリーム送信
- typedef _SINT(* IFSTUB_ReadStream) (_UBYTE *buf, _UBYTE size, _BOOL sync, _SINT tmo_ms)
  バイトストリーム受信

列挙型

- enum IFSTUB_Type {
  IFSTUB_SCI = 0,
  IFSTUB_NUM_OF_TYPE }
  インタフェースタイプ型定義

関数

- const IFSTUB_Class * IFSTUB_getInstance (IFSTUB_Type t)

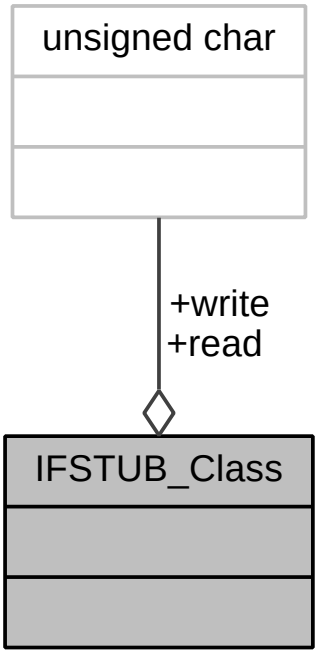
3.5.1 詳解

通信インタフェーススタブ
このモジュールは他基板と通信するときのインタフェースのみを定義した スタブである。

3.5.2 クラス詳解

■struct IFSTUB_Class スタブ本体型
ifstub.h の 79 行目に定義があります。

IFSTUB_Class 連携図



クラスメンバ

IFSTUB_↩ WriteStream	write	出力メソッド
IFSTUB_↩ ReadStream	read	入力メソッド

3.5.3 型定義詳解

■typedef _**SINT**(∗ IFSTUB_WriteStream) (const _**UBYTE** ∗data, _**UBYTE** size, _**BOOL** sync, _**SINT** tmo_ms) バイトストリーム送信

汎用バイトデータを送信予約する。同期/非同期送信の両方に対応。 指定データは内部バッファへ格納され、次回の送信可能時に処理が実行される。

引数

in	*data*	送信データの先頭へのポインタ
in	*size*	送信サイズ
in	*sync*	_TRUE: 同期型。size 分送信できるか、タイムアウトするまで ブロックする。_FALSE: 非同期型。タイムアウトは無効
in	*tmo_ms*	同期送信時のタイムアウト値。ミリセカンド。

戻り値

正の値	送信完了サイズ
負の値	送信エラー

ifstub.h の 56 行目に定義があります。

■typedef _**SINT**(∗ IFSTUB_ReadStream) (_**UBYTE** ∗buf, _**UBYTE** size, _**BOOL** sync, _**SINT** tmo↩_ms) バイトストリーム受信

内部バッファに溜まっているデータを取得する。同期/非同期受信の両方に対応。 内部バッファには非同期で随時受信データが溜まっていくため、定期的に このAPI を実行しないとバッファが溢れデータロスが発生する。

引数

in	*buf*	受信データを格納するバッファの先頭へのポインタ
in	*size*	受信サイズ
in	*sync*	_TRUE: 同期型。size 分受信できるか、タイムアウトするまで ブロックする。_FALSE: 非同期型。タイムアウトは無効
in	*tmo_ms*	同期受信時のタイムアウト値。ミリセカンド。

戻り値

正の値	受信完了サイズ
負の値	受信エラー

ifstub.h の 75 行目に定義があります。

3.5.4 列挙型詳解

■enum **IFSTUB_Type** インタフェースタイプ型定義

列挙値

IFSTUB_SCI シルアルポート

IFSTUB_NUM_OF_TYPE

ifstub.h の 29 行目に定義があります。

```
30 {
31     IFSTUB_SCI = 0,
33 #ifdef USE_LOOPBACK_SSRP

34     IFSTUB_SSRP_SKELETON,
36 #endif /* USE_LOOPBACK_SSRP */
37
38     IFSTUB_NUM_OF_TYPE
39 } IFSTUB_Type;
```

3.5.5 関数詳解

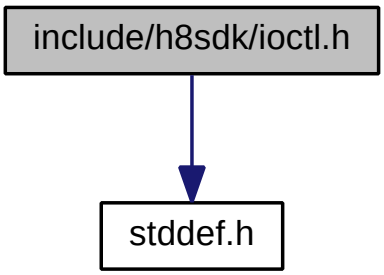
■const **IFSTUB_Class**∗ IFSTUB_getInstance (**IFSTUB_Type** *t*)

3.6 include/h8sdk/ioctl.h ファイル

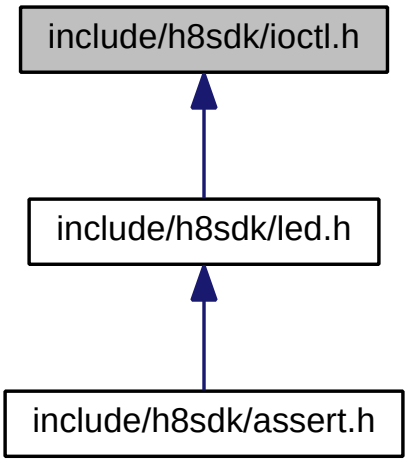
I/O ポートコントローラ

```
#include "stddef.h"
```

ioctl.h の依存先関係図:



被依存関係図:



列挙型

- enum IOCTL_Reg {
  IOCTL_REG_PCR1 = 0,
  IOCTL_REG_PCR2,
  IOCTL_REG_PCR5,
  IOCTL_REG_PCR7,
  IOCTL_REG_PCR8,
  IOCTL_NUM_OF_REG }
  PCR **レジスタリテラル定義型**

関数

- void IOCTL_init ()
  初期化
- void IOCTL_set (IOCTL_Reg reg, _UBYTE mask, _UBYTE val)
  レジスタへ値セット
- _UBYTE IOCTL_get (IOCTL_Reg reg)
  レジスタ値取得

3.6.1 詳解

I/O ポートコントローラ

このモジュールは各ポートコントロールレジスタ (PCR) への多重アクセスを 管理する。PCR は読み込みが不安定のため、複数の独立したモジュールから 設定する場合、前に設定された値が後のもので上書きされて失われてしま う。このモジュールはそれを防ぐためにPCR への多重アクセスの同期を取る 役割を担う。ただし、割り込みハンドラからの設定はサポートされない。

3.6.2 列挙型詳解

■enum **IOCTL_Reg** PCR レジスタリテラル定義型

列挙値

IOCTL_REG_PCR1 PCR1.
IOCTL_REG_PCR2 PCR2.
IOCTL_REG_PCR5 PCR5.
IOCTL_REG_PCR7 PCR7.
IOCTL_REG_PCR8 PCR8.
IOCTL_NUM_OF_REG

ioctl.h の 34 行目に定義があります。

```
35 {
36     IOCTL_REG_PCR1 = 0,
38     IOCTL_REG_PCR2,
40     IOCTL_REG_PCR5,
42     IOCTL_REG_PCR7,
44     IOCTL_REG_PCR8,
46
47     IOCTL_NUM_OF_REG
48 } IOCTL_Reg;
```

3.6.3 関数詳解

■void IOCTL_init ( ) 初期化

モジュールの初期化とPCR レジスタのゼロクリアを行う

■void IOCTL_set (**IOCTL_Reg** *reg*, **_UBYTE** *mask*, **_UBYTE** *val*) レジスタへ値セット

指定PCR レジスタへビットマスクを指定して値をセットする。 マスクビット以外の値は保持される。

引数

in	*reg*	セットするレジスタ
in	*mask*	セットする値の有効ビットマスク
in	*val*	セットする値

■**_UBYTE** IOCTL_get (**IOCTL_Reg** *reg*) レジスタ値取得

現在の指定PCR レジスタ値を取得する。内部で保持されている値が返り、実 際のI/O ポートへのアクセスは行わない。

引数

in	*reg*	値を取得するレジスタ

戻り値

レジスタ値

3.7 include/h8sdk/kbd_jp106.h ファイル

JP106 キーコード表

マクロ定義

- #define KBD_JP106_NA 0x00
- #define KBD_JP106_BREAK 0xF0
- #define KBD_JP106_ESC 0x76
- #define KBD_JP106_F1 0x05
- #define KBD_JP106_F2 0x06
- #define KBD_JP106_F3 0x04
- #define KBD_JP106_F4 0x0C
- #define KBD_JP106_F5 0x03
- #define KBD_JP106_F6 0x0B
- #define KBD_JP106_F7 0x83
- #define KBD_JP106_F8 0x0A
- #define KBD_JP106_F9 0x01
- #define KBD_JP106_F10 0x09
- #define KBD_JP106_F11 0x78
- #define KBD_JP106_F12 0x07
- #define KBD_JP106_HANKAKU 0x0E
- #define KBD_JP106_1 0x16
- #define KBD_JP106_2 0x1E
- #define KBD_JP106_3 0x26
- #define KBD_JP106_4 0x25
- #define KBD_JP106_5 0x2E
- #define KBD_JP106_6 0x36
- #define KBD_JP106_7 0x3D
- #define KBD_JP106_8 0x3E
- #define KBD_JP106_9 0x46
- #define KBD_JP106_0 0x45
- #define KBD_JP106_MINUS 0x4E
- #define KBD_JP106_HAT 0x55
- #define KBD_JP106_EN 0x6A
- #define KBD_JP106_BS 0x66
- #define KBD_JP106_TAB 0x0D
- #define KBD_JP106_Q 0x15
- #define KBD_JP106_W 0x1D
- #define KBD_JP106_E 0x24
- #define KBD_JP106_R 0x2D
- #define KBD_JP106_T 0x2C
- #define KBD_JP106_Y 0x35
- #define KBD_JP106_U 0x3C

- #define KBD_JP106_I 0x43
- #define KBD_JP106_O 0x44
- #define KBD_JP106_P 0x4D
- #define KBD_JP106_AT 0x54
- #define KBD_JP106_LBRACKET 0x5B
- #define KBD_JP106_ENTER 0x5A
- #define KBD_JP106_CAPSLOCK 0x58
- #define KBD_JP106_A 0x1C
- #define KBD_JP106_S 0x1B
- #define KBD_JP106_D 0x23
- #define KBD_JP106_F 0x2B
- #define KBD_JP106_G 0x34
- #define KBD_JP106_H 0x33
- #define KBD_JP106_J 0x3B
- #define KBD_JP106_K 0x42
- #define KBD_JP106_L 0x4B
- #define KBD_JP106_SEMICOLON 0x4C
- #define KBD_JP106_COLON 0x52
- #define KBD_JP106_RBRACKET 0x5D
- #define KBD_JP106_Q 0x15
- #define KBD_JP106_LSHIFT 0x12
- #define KBD_JP106_Z 0x1A
- #define KBD_JP106_X 0x22
- #define KBD_JP106_C 0x21
- #define KBD_JP106_V 0x2A
- #define KBD_JP106_B 0x32
- #define KBD_JP106_N 0x31
- #define KBD_JP106_M 0x3A
- #define KBD_JP106_COMMA 0x41
- #define KBD_JP106_PERIOD 0x49
- #define KBD_JP106_SLASH 0x4A
- #define KBD_JP106_BACKSLASH 0x51
- #define KBD_JP106_RSHIFT 0x59
- #define KBD_JP106_CTRL 0x14
- #define KBD_JP106_ALT 0x11
- #define KBD_JP106_MUHENKAN 0x67
- #define KBD_JP106_SPACE 0x29
- #define KBD_JP106_HENKAN 0x64
- #define KBD_JP106_KANA 0x13

3.7.1 詳解

JP106 キーコード表

JP106 vs. PS/2 スキャンコードセット 2 の対応表

3.7.2 マクロ定義詳解

■#define KBD_JP106_NA 0x00　kbd_jp106.h の 24 行目に定義があります。

■#define KBD_JP106_BREAK 0xF0　kbd_jp106.h の 25 行目に定義があります。

■#define KBD_JP106_ESC 0x76　kbd_jp106.h の 27 行目に定義があります。

■#define KBD_JP106_F1 0x05 kbd_jp106.h の 28 行目に定義があります。

■#define KBD_JP106_F2 0x06 kbd_jp106.h の 29 行目に定義があります。

■#define KBD_JP106_F3 0x04 kbd_jp106.h の 30 行目に定義があります。

■#define KBD_JP106_F4 0x0C kbd_jp106.h の 31 行目に定義があります。

■#define KBD_JP106_F5 0x03 kbd_jp106.h の 32 行目に定義があります。

■#define KBD_JP106_F6 0x0B kbd_jp106.h の 33 行目に定義があります。

■#define KBD_JP106_F7 0x83 kbd_jp106.h の 34 行目に定義があります。

■#define KBD_JP106_F8 0x0A kbd_jp106.h の 35 行目に定義があります。

■#define KBD_JP106_F9 0x01 kbd_jp106.h の 36 行目に定義があります。

■#define KBD_JP106_F10 0x09 kbd_jp106.h の 37 行目に定義があります。

■#define KBD_JP106_F11 0x78 kbd_jp106.h の 38 行目に定義があります。

■#define KBD_JP106_F12 0x07 kbd_jp106.h の 39 行目に定義があります。

■#define KBD_JP106_HANKAKU 0x0E kbd_jp106.h の 40 行目に定義があります。

■#define KBD_JP106_1 0x16 kbd_jp106.h の 41 行目に定義があります。

■#define KBD_JP106_2 0x1E kbd_jp106.h の 42 行目に定義があります。

■#define KBD_JP106_3 0x26 kbd_jp106.h の 43 行目に定義があります。

■#define KBD_JP106_4 0x25 kbd_jp106.h の 44 行目に定義があります。

■#define KBD_JP106_5 0x2E kbd_jp106.h の 45 行目に定義があります。

■#define KBD_JP106_6 0x36 kbd_jp106.h の 46 行目に定義があります。

■#define KBD_JP106_7 0x3D kbd_jp106.h の 47 行目に定義があります。

■#define KBD_JP106_8 0x3E kbd_jp106.h の 48 行目に定義があります。

■#define KBD_JP106_9 0x46 kbd_jp106.h の 49 行目に定義があります。

■#define KBD_JP106_0 0x45 kbd_jp106.h の 50 行目に定義があります。

■#define KBD_JP106_MINUS 0x4E kbd_jp106.h の 51 行目に定義があります。

■#define KBD_JP106_HAT 0x55 kbd_jp106.h の 52 行目に定義があります。

■#define KBD_JP106_EN 0x6A kbd_jp106.h の 53 行目に定義があります。

■#define KBD_JP106_BS 0x66 kbd_jp106.h の 54 行目に定義があります。

■#define KBD_JP106_TAB 0x0D kbd_jp106.h の 55 行目に定義があります。

■#define KBD_JP106_Q 0x15 kbd_jp106.h の 82 行目に定義があります。

■#define KBD_JP106_W 0x1D kbd_jp106.h の 57 行目に定義があります。

■#define KBD_JP106_E 0x24 kbd_jp106.h の 58 行目に定義があります。

■#define KBD_JP106_R 0x2D kbd_jp106.h の 59 行目に定義があります。

■#define KBD_JP106_T 0x2C kbd_jp106.h の 60 行目に定義があります。

■#define KBD_JP106_Y 0x35 kbd_jp106.h の 61 行目に定義があります。

■#define KBD_JP106_U 0x3C kbd_jp106.h の 62 行目に定義があります。

■#define KBD_JP106_I 0x43 kbd_jp106.h の 63 行目に定義があります。

■#define KBD_JP106_O 0x44 kbd_jp106.h の 64 行目に定義があります。

■#define KBD_JP106_P 0x4D kbd_jp106.h の 65 行目に定義があります。

■#define KBD_JP106_AT 0x54 kbd_jp106.h の 66 行目に定義があります。

■#define KBD_JP106_LBRACKET 0x5B kbd_jp106.h の 67 行目に定義があります。

■#define KBD_JP106_ENTER 0x5A kbd_jp106.h の 68 行目に定義があります。

■#define KBD_JP106_CAPSLOCK 0x58 kbd_jp106.h の 69 行目に定義があります。

■#define KBD_JP106_A 0x1C kbd_jp106.h の 70 行目に定義があります。

■#define KBD_JP106_S 0x1B kbd_jp106.h の 71 行目に定義があります。

■#define KBD_JP106_D 0x23 kbd_jp106.h の 72 行目に定義があります。

■#define KBD_JP106_F 0x2B kbd_jp106.h の 73 行目に定義があります。

■#define KBD_JP106_G 0x34 kbd_jp106.h の 74 行目に定義があります。

■#define KBD_JP106_H 0x33 kbd_jp106.h の 75 行目に定義があります。

■#define KBD_JP106_J 0x3B kbd_jp106.h の 76 行目に定義があります。

■#define KBD_JP106_K 0x42 kbd_jp106.h の 77 行目に定義があります。

■#define KBD_JP106_L 0x4B kbd_jp106.h の 78 行目に定義があります。

■#define KBD_JP106_SEMICOLON 0x4C kbd_jp106.h の 79 行目に定義があります。

■#define KBD_JP106_COLON 0x52 kbd_jp106.h の 80 行目に定義があります。

■#define KBD_JP106_RBRACKET 0x5D kbd_jp106.h の 81 行目に定義があります。

■#define KBD_JP106_Q 0x15 kbd_jp106.h の 82 行目に定義があります。

■#define KBD_JP106_LSHIFT 0x12 kbd_jp106.h の 83 行目に定義があります。

■#define KBD_JP106_Z 0x1A kbd_jp106.h の 84 行目に定義があります。

■#define KBD_JP106_X 0x22 kbd_jp106.h の 85 行目に定義があります。

■#define KBD_JP106_C 0x21 kbd_jp106.h の 86 行目に定義があります。

■#define KBD_JP106_V 0x2A kbd_jp106.h の 87 行目に定義があります。

■#define KBD_JP106_B 0x32 kbd_jp106.h の 88 行目に定義があります。

■#define KBD_JP106_N 0x31 kbd_jp106.h の 89 行目に定義があります。

■#define KBD_JP106_M 0x3A kbd_jp106.h の 90 行目に定義があります。

■#define KBD_JP106_COMMA 0x41 kbd_jp106.h の 91 行目に定義があります。

■#define KBD_JP106_PERIOD 0x49 kbd_jp106.h の 92 行目に定義があります。

■#define KBD_JP106_SLASH 0x4A kbd_jp106.h の 93 行目に定義があります。

■#define KBD_JP106_BACKSLASH 0x51 kbd_jp106.h の 94 行目に定義があります。

■#define KBD_JP106_RSHIFT 0x59 kbd_jp106.h の 95 行目に定義があります。

■#define KBD_JP106_CTRL 0x14 kbd_jp106.h の 96 行目に定義があります。

■#define KBD_JP106_ALT 0x11 kbd_jp106.h の 97 行目に定義があります。

■#define KBD_JP106_MUHENKAN 0x67 kbd_jp106.h の 98 行目に定義があります。

■#define KBD_JP106_SPACE 0x29 kbd_jp106.h の 99 行目に定義があります。

■#define KBD_JP106_HENKAN 0x64 kbd_jp106.h の 100 行目に定義があります。

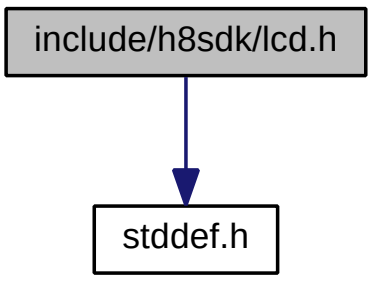
■#define KBD_JP106_KANA 0x13 kbd_jp106.h の 101 行目に定義があります。

3.8 include/h8sdk/lcd.h ファイル

LCD ドライバ

```
#include "stddef.h"
```

lcd.h の依存先関係図:



マクロ定義

- #define LCD_MAX_COLS 16

LCD **表示部の列数**

- #define LCD_MAX_ROWS 2
  LCD **表示部の行数**
- #define LCD_CG_COLS 5
  キャラクタ横ドット数
- #define LCD_CG_ROWS 8
  キャラクタ縦ドット数
- #define LCD_CG_MAXCHAR 8
  登録可能な最大キャラクタ数

関数

- void LCD_init ()
  LCD **デバイス初期化**
- _SINT LCD_puts (const _SBYTE *str)
  文字列出力
- _SINT LCD_write (const _UBYTE *data, _SINT size)
  バイトストリーム書き込み
- _SINT LCD_read (_UBYTE *buf, _SINT size)
  バイトストリーム読み出し
- void LCD_getCursor (_SINT *x, _SINT *y)
  カーソル位置取得
- void LCD_setCursor (_SINT x, _SINT y)
  カーソル移動
- void LCD_crlf ()
  改行出力
- void LCD_cls ()
  領域クリア
- void LCD_setChar (_UBYTE no, const _UBYTE *data)
  キャラクタ登録
- void LCD_setVisual (_BOOL all, _BOOL cur, _BOOL blink)
  表示モード変更

3.8.1 詳解

LCD ドライバ

このモジュールはLCD デバイスSC1602BS へのインタフェースを提供する。

3.8.2 マクロ定義詳解

■#define LCD_MAX_COLS 16 LCD 表示部の列数

lcd.h の 29 行目に定義があります。

■#define LCD_MAX_ROWS 2 LCD 表示部の行数

lcd.h の 34 行目に定義があります。

■#define LCD_CG_COLS 5 キャラクタ横ドット数

lcd.h の 39 行目に定義があります。

■#define LCD_CG_ROWS 8 キャラクタ縦ドット数

lcd.h の 44 行目に定義があります。

■#define LCD_CG_MAXCHAR 8 登録可能な最大キャラクタ数
lcd.h の 49 行目に定義があります。

3.8.3 関数詳解

■void LCD_init ( ) LCD デバイス初期化
デバイスを使用可能にする。全てのAPI の前にこれを実行しておく。初期設 定値は、

- キャラクタ長 8 ビット
- 表示 2 桁
- キャラクタサイズ 5x10 ドット
- カーソルOFF、ブリンクOFF
- カーソルインクリメントあり、表示シフトなし

■**_SINT** LCD_puts (const **_SBYTE** ∗ *str*) 文字列出力
文字列をLCD に出力する。文字列はNUL 終端されていなければならない。 特殊文字は '\n' のみ。改行文字と解釈される。
引数

in	*str*	文字配列の先頭へのポインタ

戻り値

出力された文字数

覚え書き

出力カーソルは表示部の領域をループする。

■**_SINT** LCD_write (const **_UBYTE** ∗ *data*, **_SINT** *size*) バイトストリーム書き込み
汎用データ出力。カーソル現在位置から指定サイズ分のバイトデータをLCD に出力する。
引数

in	*data*	データの先頭へのポインタ
in	*size*	出力サイズ

戻り値

出力されたバイト数

覚え書き

カーソルは表示部の領域をループする。

■**_SINT** LCD_read (**_UBYTE** ∗ *buf*, **_SINT** *size*) バイトストリーム読み出し
汎用データ入力。カーソル現在位置から指定バイト読み出す。

引数

in	*buf*	データを格納する領域の先頭へのポインタ
in	*size*	読み出しサイズ

戻り値

読み出されたバイト数

覚え書き

カーソルは表示部の領域をループする。

■void LCD_getCursor (_**SINT** ∗ *x*, _**SINT** ∗ *y*) カーソル位置取得

現在の出力位置を取得する。引数の位置指定は絶対値の 0 origin。原点は 左上隅、右下方向がプラスの領域。

引数

in	x	列方向の位置を格納するポインタ
in	y	行方向の位置を格納するポインタ

■void LCD_setCursor (_**SINT** *x*, _**SINT** *y*) カーソル移動

出力位置を移動させる。表示部の範囲でのみ移動可能。引数の位置指定は 絶対値の 0 origin。原点は左上隅、右下方向がプラスの領域。範囲を越え た値指定は無視される。

引数

in	x	列方向の位置を指定する
in	y	行方向の位置を指定する

■void LCD_crlf ( ) 改行出力

出力カーソルを一行下の最左列へ移動させる。

覚え書き

出力カーソルは表示部の領域をループする。

■void LCD_cls ( ) 領域クリア

LCD バッファの内容をクリアする

■void LCD_setChar (_**UBYTE** *no*, const _**UBYTE** ∗ *data*) キャラクタ登録

ユーザー作成キャラクタデータを新規にLCD デバイスへ登録する。 同一ア ドレスで登録済みのキャラクタは上書きされる。

引数

in	*no*	登録アドレス。0〜LCD_CG_MAXCHAR-1 までを指定する

in	*data*	登録キャラクタ配列先頭へのポインタ。LCD_CG_ROWS バイ トの配列をLCD_CG_COLS x LCD_CG_ROWS のビットパターン行列とみなしたデー タ列を指定する。

■void LCD_setVisual (_**BOOL** *all*, _**BOOL** *cur*, _**BOOL** *blink*) 表示モード変更

カーソル、点滅などの表示モードを変更する。初期状態は、表示は_TRUE、 カーソル、点滅は_FALSE。

引数

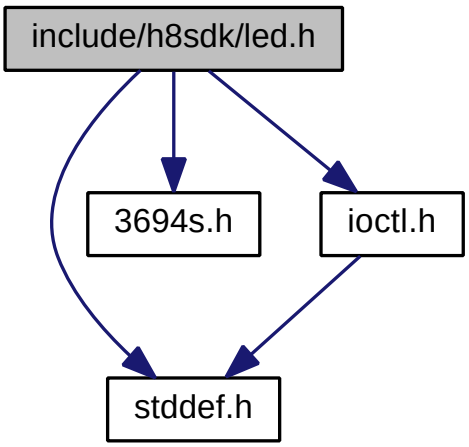
in	*all*	全表示のオン/オフ指定
in	*cur*	カーソルのオン/オフ指定
in	*blink*	カーソル位置のブリンクオン/オフ指定

3.9 include/h8sdk/led.h ファイル

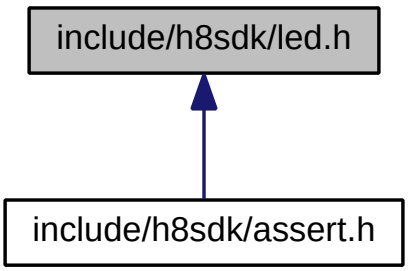
LED ドライバ

```
#include "stddef.h"
#include "3694s.h"
#include "ioctl.h"
```

led.h の依存先関係図:



被依存関係図:



マクロ定義

- #define LED_BASE (1U << 1)
  LED **が接続されている***GPIO* **レジスタのベース値**
- #define LED_INIT
  LED **初期化**
- #define LED_ON(t) (_IO.PDR8.BYTE &= ~(LED_BASE << (t)))
  LED **点灯**
- #define LED_OFF(t) (_IO.PDR8.BYTE |= LED_BASE << (t))
  LED **消灯**
- #define LED_TURN(t) (_IO.PDR8.BYTE ^= LED_BASE << (t))
  LED **表示反転**

列挙型

- enum LED_Type {
  LED_D5 = 0,
  LED_D6,
  LED_NUM_OF_TYPE }
  LED **タイプ識別子**

3.9.1 詳解

LED ドライバ

このモジュールはLED デバイスへのインタフェースを提供する。

3.9.2 マクロ定義詳解

■#define LED_BASE (1U << 1) LED が接続されているGPIO レジスタのベース値

led.h の 45 行目に定義があります。

■#define LED_INIT **値:**

```
do                                                          \
    {                                                       \
        IOCTL_set(IOCTL_REG_PCR8, 0x06, 0x06);              \
        _IO.PDR8.BYTE |= (LED_BASE << LED_D5 | LED_BASE << LED_D6);   \
    }                                                       \
    while (_FALSE)
```

LED 初期化

LED デバイスを使用可能に設定する。全てのAPI の前にこれを実行しておく。

led.h の 52 行目に定義があります。

■#define LED_ON(*t*) (_IO.PDR8.BYTE &= ~(**LED_BASE** << (t))) LED 点灯

LED をひとつ点灯させる。

引数

in	t	点灯させるLED の種類。LED_Type

led.h の 67 行目に定義があります。

■#define LED_OFF(t) (_IO.PDR8.BYTE |= **LED_BASE** << (t)) LED 消灯

LED をひとつ消灯させる。

引数

in	t	消灯させるLED の種類。LED_Type

led.h の 76 行目に定義があります。

■#define LED_TURN(t) (_IO.PDR8.BYTE ^= **LED_BASE** << (t)) LED 表示反転

指定されたLED の表示を現在のものと反転させる。

引数

in	t	反転させるLED の種類。LED_Type

led.h の 85 行目に定義があります。

3.9.3 列挙型詳解

■enum **LED_Type** LED タイプ識別子

列挙値

LED_D5 D5 番ポートのLED.

LED_D6 D6 番ポートのLED.

LED_NUM_OF_TYPE LED タイプの数

led.h の 31 行目に定義があります。

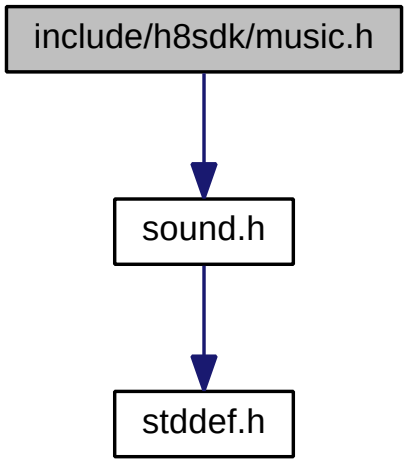
```
32 {
33     LED_D5 = 0,
35     LED_D6,
37
38     LED_NUM_OF_TYPE
40 } LED_Type;
```

3.10 include/h8sdk/music.h ファイル

楽曲演奏モジュール

```
#include "sound.h"
```

music.h の依存先関係図:



クラス

- struct MUSIC_Note
  音符定義型
- struct MUSIC_Part
  パート定義型
- struct MUSIC_Score
  楽譜定義型
- struct MUSIC_Position
  再生位置

マクロ定義

- #define MUSIC_MAX_PART SOUND_MAX_PRONOUNCE
  最大再生パート数
- #define MUSIC_L0 768
  全音符
- #define MUSIC_L2 384
  2 **分音符**
- #define MUSIC_L4 192
  4 **分音符**
- #define MUSIC_L8 96
  8 **分音符**
- #define MUSIC_L16 48
  16 **分音符**
- #define MUSIC_L32 24
  32 **分音符**
- #define MUSIC_L64 12
  64 **分音符**
- #define MUSIC_L128 6
  128 **分音符**
- #define MUSIC_L3C 64
  3 **連符**

- #define MUSIC_L6C 32
  6 **連符**

型定義

- typedef _BOOL(* MUSIC_Api) ()
  再生操作リクエスト*API* **の型**

列挙型

- enum MUSIC_State {
  MUSIC_ST_STOP = 0,
  MUSIC_ST_PLAY,
  MUSIC_ST_REVERSE,
  MUSIC_ST_PAUSE,
  MUSIC_NUM_OF_STATE }
  再生状態リテラル型

関数

- void MUSIC_init ()
  初期化
- MUSIC_State MUSIC_getState ()
  現在の再生状態取得
- void MUSIC_setTempo (_UBYTE val)
  テンポ設定
- void MUSIC_setLoop (_BOOL val)
  再生ループ設定
- void MUSIC_getPosition (MUSIC_Position *pos)
  再生位置取得
- void MUSIC_setPosition (const MUSIC_Position *pos)
  再生位置設定
- void MUSIC_setScore (const MUSIC_Score *score)
  楽曲登録
- const MUSIC_Score * MUSIC_getScore ()
  楽曲取得
- _BOOL MUSIC_play ()
  楽曲再生リクエスト
- _BOOL MUSIC_reverse ()
  楽曲逆再生リクエスト
- _BOOL MUSIC_pause ()
  楽曲一時停止リクエスト
- _BOOL MUSIC_stop ()
  楽曲停止リクエスト
- void MUSIC_render ()
  楽曲演奏/録音

変数

- const MUSIC_Api MUSIC_state_handler [MUSIC_NUM_OF_STATE]

状態ハンドラ配列

3.10.1 詳解

楽曲演奏モジュール

このモジュールは楽曲の再生/録音インタフェースを提供する。

サウンドドライバ必須のため、このモジュールを使うときは、外部でサウ ンドドライバの初期化をする必要はない。録音機能を有効にするときは USE_MUSIC_RECORD を define してコンパイルする。

覚え書き

> 録音機能を有効にするためのメモリ領域は、サウンドドライバの最大同時 発信音数 (SOUND_MAX_↩ PRONOUNCE) が 1 のときで最低 36 バイトのメモリ領域が 必要であり、以降増える度に 26 バイトの領域を必要とする。これで和音を 含む音符ひとつを録音可能である。
>
> さらに、録音可能な音符 (MUSIC_NOTE_OF_RECORD_PART) は、ひとつ増えるた びに 14 バイトの領域が必要となるが、最大同時発信音数を係数として増大 するため、録音音符サイズ×最大同時発信音数分の領域が必要になる。以 上より、録音時に必要なメモリサイズは、

```
10 + (10 + 14 * MUSIC_NOTE_OF_RECORD_PART) * SOUND_MAX_PRONOUNCE
```

> で表される。大体 3 和音の音符 5 つで 250 バイトほど必要になる。

参照

> sound.h

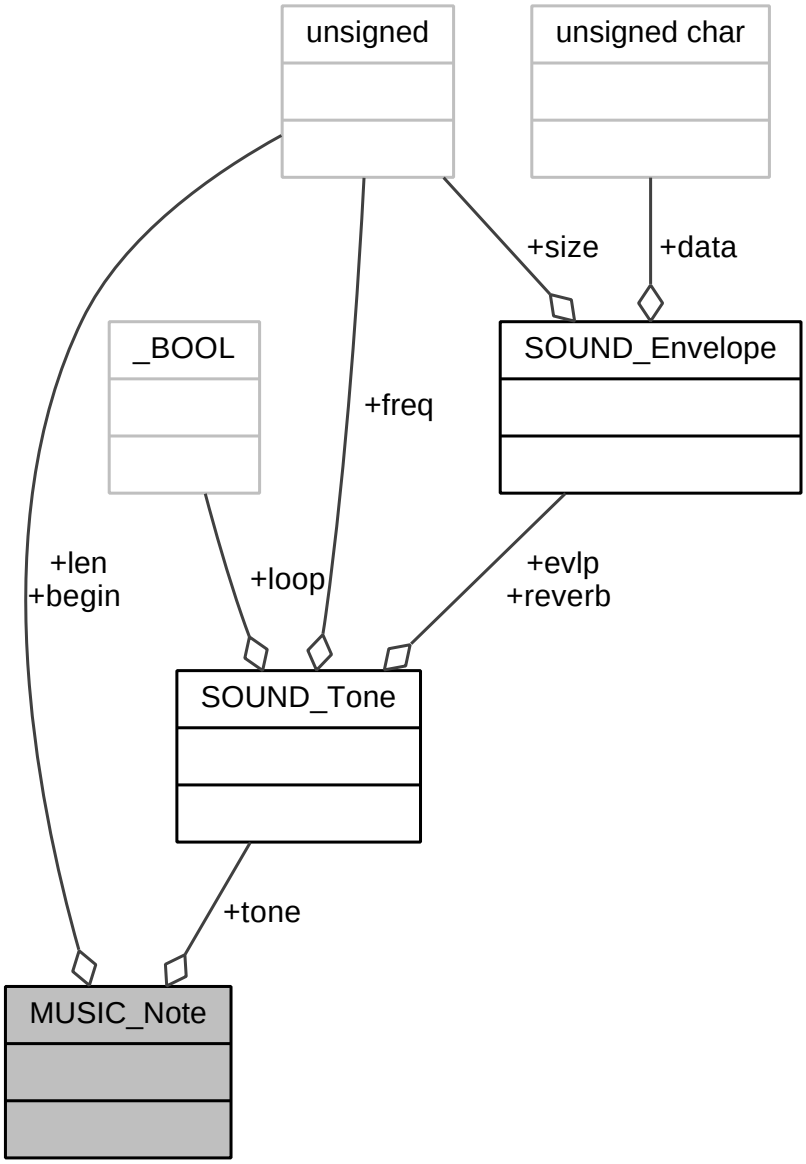
3.10.2 クラス詳解

■struct MUSIC_Note 音符定義型

単音の音階、音色、音長を定義する型。

music.h の 95 行目に定義があります。

MUSIC_Note 連携図



クラスメンバ

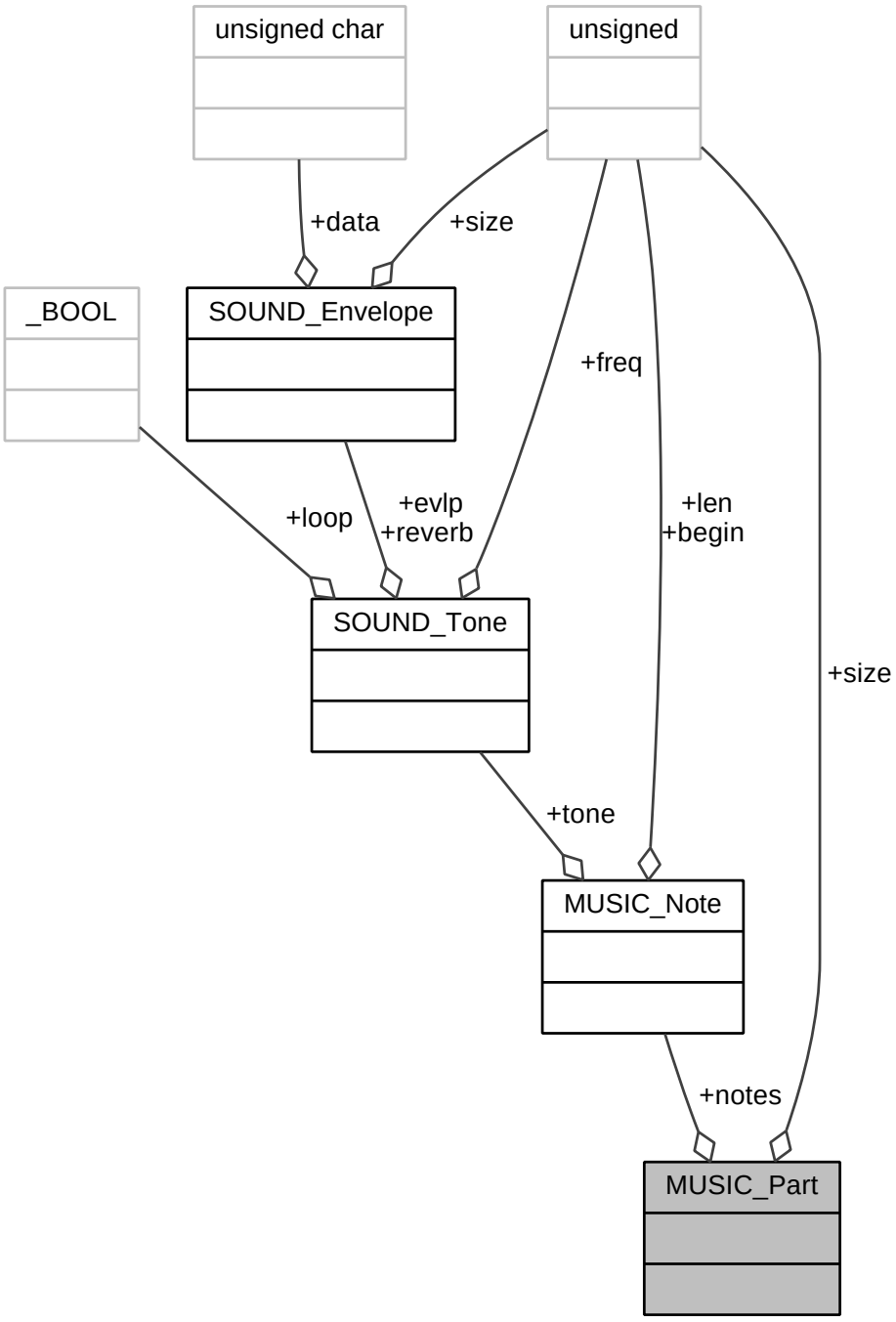
SOUND_Tone	tone	発生させる音データ (音階、音色)
_UDWORD	begin	発音を開始する楽曲カウント値 (絶対値)
_UWORD	len	音長。begin からの相対値

■struct MUSIC_Part パート定義型

複数の音符列から成る単パートを定義する型

music.h の 110 行目に定義があります。

MUSIC_Part 連携図



クラスメンバ

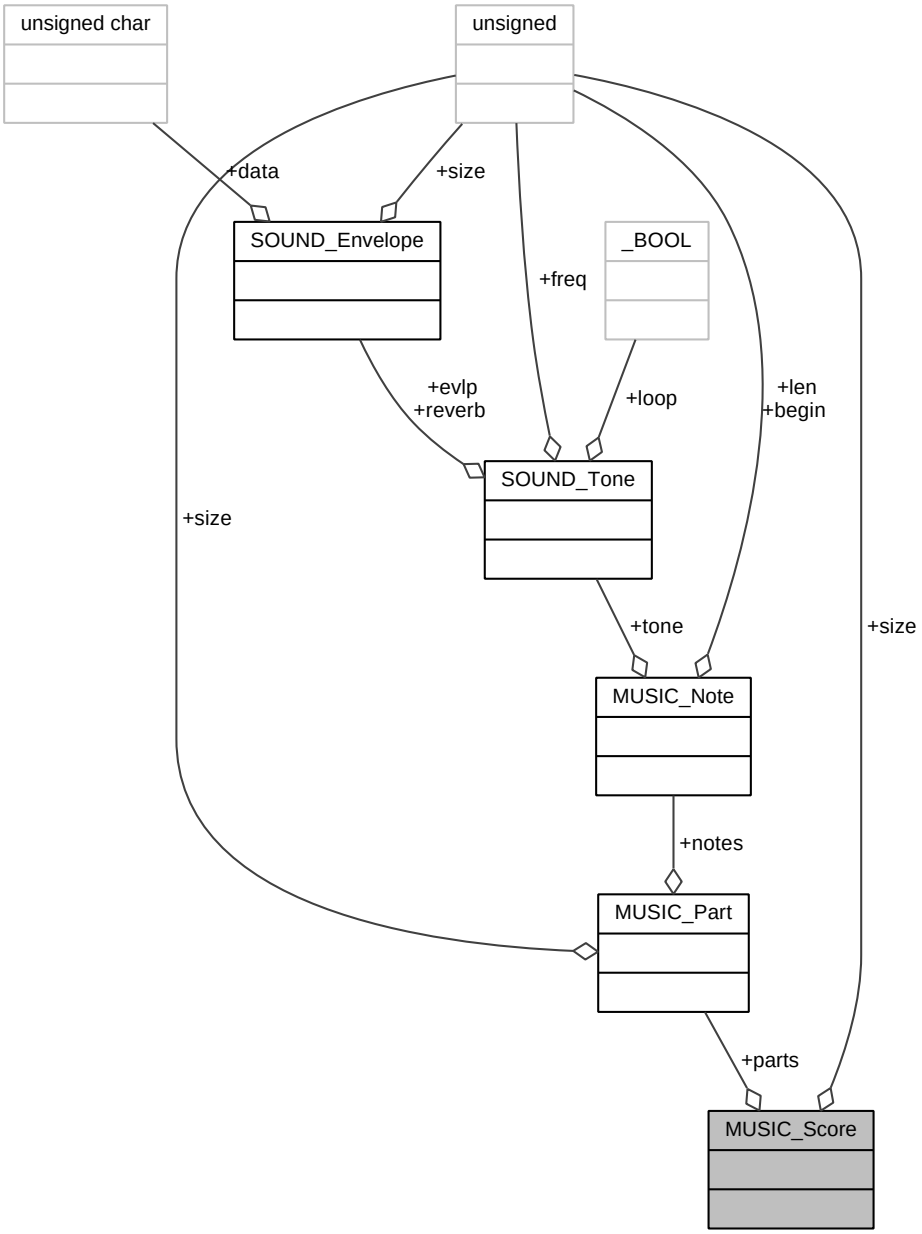
MUSIC_Note *	notes	
_UWORD	size	音符データ列

■struct MUSIC_Score 楽譜定義型

複数のパートから成る最終的な演奏楽曲を表す楽譜を定義する型

music.h の 123 行目に定義があります。

MUSIC_Score 連携図



クラスメンバ

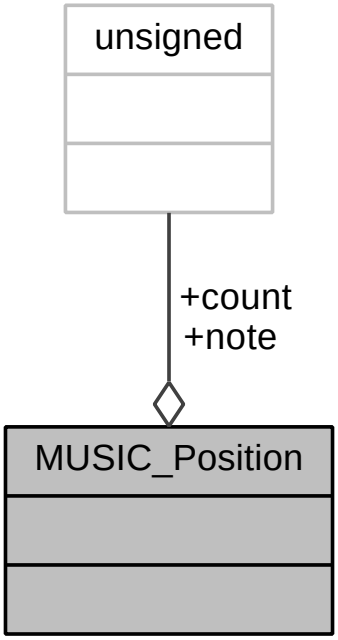
MUSIC_Part *	parts	
_UWORD	size	パートデータ列

■struct MUSIC_Position 再生位置

再生位置情報の定義

music.h の 136 行目に定義があります。

MUSIC_Position 連携図



クラスメンバ

_UWORD	note[MUSIC_↩ MAX_PART]	パート毎の音符データ列の位置
_UDWORD	count	楽曲の再生カウント

3.10.3 マクロ定義詳解

■#define MUSIC_MAX_PART **SOUND_MAX_PRONOUNCE** 最大再生パート数

music.h の 50 行目に定義があります。

■#define MUSIC_L0 768 全音符

music.h の 53 行目に定義があります。

■#define MUSIC_L2 384 2 分音符

music.h の 55 行目に定義があります。

■#define MUSIC_L4 192 4 分音符

music.h の 57 行目に定義があります。

■#define MUSIC_L8 96 8 分音符

music.h の 59 行目に定義があります。

■#define MUSIC_L16 48 16 分音符

music.h の 61 行目に定義があります。

■#define MUSIC_L32 24 32 分音符

music.h の 63 行目に定義があります。

■#define MUSIC_L64 12 64 分音符

music.h の 65 行目に定義があります。

■#define MUSIC_L128 6 128 分音符
music.h の 67 行目に定義があります。

■#define MUSIC_L3C 64 3 連符
music.h の 69 行目に定義があります。

■#define MUSIC_L6C 32 6 連符
music.h の 71 行目に定義があります。

3.10.4 型定義詳解

■typedef _**BOOL**(* MUSIC_Api) () 再生操作リクエストAPI の型
music.h の 147 行目に定義があります。

3.10.5 列挙型詳解

■enum **MUSIC_State** 再生状態リテラル型

列挙値

MUSIC_ST_STOP
MUSIC_ST_PLAY
MUSIC_ST_REVERSE
MUSIC_ST_PAUSE
MUSIC_NUM_OF_STATE

music.h の 77 行目に定義があります。

```
78 {
79     MUSIC_ST_STOP = 0,
80     MUSIC_ST_PLAY,
81     MUSIC_ST_REVERSE,
82     MUSIC_ST_PAUSE,
83 #ifdef USE_MUSIC_RECORD

84     MUSIC_ST_RECORD,
85 #endif /* USE_MUSIC_RECORD */
86
87     MUSIC_NUM_OF_STATE
88 } MUSIC_State;
```

3.10.6 関数詳解

■void MUSIC_init ( ) 初期化
サウンドドライバを初期化しモジュールを使用可能にする。 他API 使用前に必ず実行する。

■**MUSIC_State** MUSIC_getState ( ) 現在の再生状態取得

戻り値

再生状態リテラル

■void MUSIC_setTempo (_**UBYTE** *val*) テンポ設定
再生速度を設定する。0〜MUSIC_MAX_TEMPO いつでも設定可能。

引数

in	*val*	テンポ値

バグ このAPI は未実装。呼び出すとハングアップする。

■void MUSIC_setLoop (_**BOOL** *val*) 再生ループ設定

ループ再生するかどうかを設定する。デフォルトはオフ。 いつでも設定可能。

引数

in	*val*	TRUE:ループする。FALSE:ループしない

■void MUSIC_getPosition (**MUSIC_Position** ∗ *pos*) 再生位置取得

現在の再生位置を取得する。

引数

out	*pos*	再生中のポジション情報を格納する領域へのポインタ

■void MUSIC_setPosition (const **MUSIC_Position** ∗ *pos*) 再生位置設定

再生位置を指定値へ変更する。いつでも設定可能。

引数

in	*pos*	再生位置情報データへのポインタ

■void MUSIC_setScore (const **MUSIC_Score** ∗ *score*) 楽曲登録

再生する楽曲を登録する。

引数

in	*score*	楽曲データ

■const **MUSIC_Score**∗ MUSIC_getScore ( ) 楽曲取得

現在登録/録音されている楽曲データを取得する。

戻り値

楽曲データへのポインタ

■_**BOOL** MUSIC_play ( ) 楽曲再生リクエスト

登録/録音されている楽曲を順に再生させる。

戻り値

TRUE	リクエスト受諾
FALSE	リクエスト拒否

■_**BOOL** MUSIC_reverse ( ) 楽曲逆再生リクエスト

登録/録音されている楽曲を逆から再生させる。

戻り値

TRUE	リクエスト受諾
FALSE	リクエスト拒否

■**_BOOL** MUSIC_pause ( ) 楽曲一時停止リクエスト

再生中の楽曲を一時停止させる。

戻り値

TRUE	リクエスト受諾
FALSE	リクエスト拒否

■**_BOOL** MUSIC_stop ( ) 楽曲停止リクエスト

再生中の楽曲を停止させる。次回再生時は先頭からになる。

戻り値

TRUE	リクエスト受諾
FALSE	リクエスト拒否

■void MUSIC_render ( ) 楽曲演奏/録音

演奏状態 (MUSIC_play、MUSIC_reverse) もしくは録音状態 (MUSIC_record) に ある場合、このAPI を呼ぶことで楽曲の演奏/録音が実行される。内部では 呼び出し回数が音長 (MUSIC_Note::len) としてカウントされており、逐次楽 譜を読み進めながら音の再生を行っている。なので、通常はTimerA 割り込 みハンドラから呼ばれるようにしておけばよい。

3.10.7 変数詳解

■const **MUSIC_Api** MUSIC_state_handler[**MUSIC_NUM_OF_STATE**] 状態ハンドラ配列

この配列に再生状態リテラルをインデックスとして与えれば、リテラル値 に応じたハンドラが呼ばれる。

参照

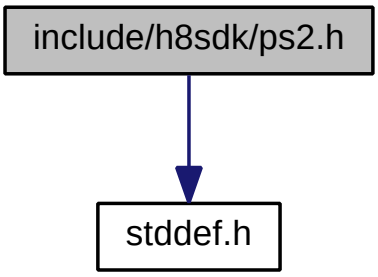
MUSIC_State

3.11 include/h8sdk/ps2.h ファイル

PS/2 ドライバ

```
#include "stddef.h"
```

ps2.h の依存先関係図:



マクロ定義

- #define PS2_MAX_RXBUF 8

列挙型

- enum PS2_Status {
  PS2_ERR_OK = 0,
  PS2_ERR_FRAMING,
  PS2_ERR_PARITY,
  PS2_NUM_OF_ERR }

関数

- void PS2_init ()
  PS2 **ポート初期化**
- void PS2_communicate ()
  送受信実行
- _SINT PS2_read (_UBYTE *buf, _UBYTE size, _BOOL sync, _SINT tmo_ms)
  バイトストリーム受信

3.11.1 詳解

PS/2 ドライバ

このモジュールはPS/2 ポートへのインタフェースを提供する。

3.11.2 マクロ定義詳解

■#define PS2_MAX_RXBUF 8 ps2.h の 26 行目に定義があります。

3.11.3 列挙型詳解

■enum **PS2_Status**

列挙値

PS2_ERR_OK
PS2_ERR_FRAMING
PS2_ERR_PARITY
PS2_NUM_OF_ERR

ps2.h の 28 行目に定義があります。

```
29 {
30     PS2_ERR_OK = 0,
31     PS2_ERR_FRAMING,
32     PS2_ERR_PARITY,
33
34     PS2_NUM_OF_ERR
35 } PS2_Status;
```

3.11.4 関数詳解

■void PS2_init ( ) PS2 ポート初期化

デバイスを使用可能にする。全てのAPI の前にこれを実行しておく。

■void PS2_communicate ( ) 送受信実行

PS/2 デバイスへ送受信処理を委託する。割り込みハンドラから呼ばれるの が前提。

■**_SINT** PS2_read (**_UBYTE** ∗ *buf*, **_UBYTE** *size*, **_BOOL** *sync*, **_SINT** *tmo_ms*) バイトストリーム受信

内部バッファに溜まっているデータを取得する。同期/非同期受信の両方に 対応。内部バッファには非同期で随時受信データが溜まっていくため、定 期的にこのAPI を実行しないとバッファが溢れデータロスが発生する。

引数

in	*buf*	受信データを格納するバッファの先頭へのポインタ
in	*size*	受信サイズ
in	*sync*	_TRUE: 同期型。size 分受信できるか、タイムアウトする までブロックする。 _FALSE: 非同期型。タイムアウトは無効
in	*tmo_ms*	同期受信時のタイムアウト値。ミリセカンド。

戻り値

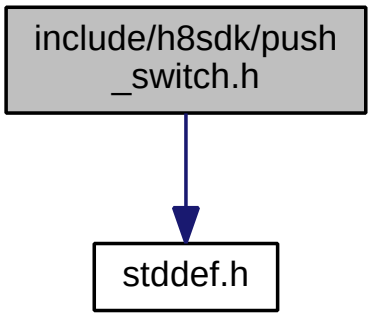
正の値	受信完了サイズ
負の値	受信エラー

3.12 include/h8sdk/push_switch.h ファイル

プッシュスイッチドライバ

```
#include "stddef.h"
```

push_switch.h の依存先関係図:



マクロ定義

- #define PSW_BORDER 0x800
  押下状態判定用のカウンタ

列挙型

- enum PSW_Type {
  PSW_SW1 = 0,
  PSW_SW2,
  PSW_NUM_OF_TYPE }
  スイッチタイプ識別子

関数

- void PSW_init (_BOOL interrupt)
  スイッチデバイス初期化。
- _BOOL PSW_get (PSW_Type t)
  スイッチ押下状態の取得
- _BOOL PSW_oneShot (PSW_Type t)
  スイッチ押下状態を一度だけ取得
- _BOOL PSW_snapShot (PSW_Type t)
  スイッチ押下状態スナップショットの取得

3.12.1 詳解

プッシュスイッチドライバ

このモジュールはプッシュスイッチへのインタフェースを提供する。

3.12.2 マクロ定義詳解

■#define PSW_BORDER 0x800 押下状態判定用のカウンタ

一度判定されたらこの値分カウントされるまで押下状態の判定は行わない

push_switch.h の 31 行目に定義があります。

3.12.3 列挙型詳解

■enum **PSW_Type** スイッチタイプ識別子

列挙値

PSW_SW1 S1 番ポートのスイッチ

PSW_SW2 S2 番ポートのスイッチ

PSW_NUM_OF_TYPE スイッチの数

push_switch.h の 36 行目に定義があります。

```
37 {
38     PSW_SW1 = 0,
40     PSW_SW2,
42
43     PSW_NUM_OF_TYPE
45 } PSW_Type;
```

3.12.4 関数詳解

■void PSW_init (**_BOOL** *interrupt*) スイッチデバイス初期化。

スイッチデバイスを使用可能に設定する。他API 使用前に必ず実行する。

引数

in	*interrupt*	_TRUE：割り込みを使う。 _FALSE：割り込みを使わない。

覚え書き

割り込みハンドラは外部で定義する。

■**_BOOL** PSW_get (**PSW_Type** *t*) スイッチ押下状態の取得

スイッチの押下状態を得る。一度押下と判定された場合、PSW_BORDER カウ ント分呼び出されるまで次の状態判定は行われない。なので、定期的に一 定回数このAPI を実行することで正確な状態を連続的に取得できるようにな る。ポーリング用。

引数

in	*t*	取得するスイッチのタイプ

戻り値

_TRUE	押されている
_FALSE	押されていない

覚え書き

割り込みを使用する場合はこのAPI を使う必要はない

■**_BOOL** PSW_oneShot (**PSW_Type** *t*) スイッチ押下状態を一度だけ取得

スイッチの押下状態を得る。一度押下と判定されたらスイッチがオフにな るまで押下判定は行われない

引数

in	t	取得するスイッチのタイプ

戻り値

$\_TRUE$	押されている
$\_FALSE$	押されていない

覚え書き

割り込みを使用する場合はこのAPI を使う必要はない

■ **_BOOL** PSW_snapShot (**PSW_Type** t) スイッチ押下状態スナップショットの取得

スイッチの押下状態を得る。取得されるのはその瞬間の状態。

引数

in	t	取得するスイッチのタイプ

戻り値

$\_TRUE$	押されている
$\_FALSE$	押されていない

覚え書き

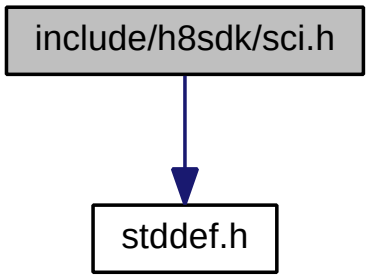
割り込みを使用する場合はこのAPI を使う必要はない

3.13 include/h8sdk/sci.h ファイル

シリアルポートドライバ

```
#include "stddef.h"
```

sci.h の依存先関係図:



マクロ定義

- #define SCI_MAX_TXBUF 128
  送信バッファサイズ
- #define SCI_MAX_RXBUF 128
  受信バッファサイズ
- #define SCI_ERR_PARITY (1U << 0)

パリティエラー識別子

- #define SCI_ERR_FRAMING (1U << 1)
  フレーミングエラー識別子
- #define SCI_ERR_OVERRUN (1U << 2)
  オーバーランエラー識別子

関数

- void SCI_init ()
  SCI **初期化**
- void SCI_communicate ()
  送受信実行
- _SINT SCI_puts (const _SBYTE *str)
  文字列送信
- _SINT SCI_write (const _UBYTE *data, _UBYTE size, _BOOL sync, _SINT tmo_ms)
  バイトストリーム送信
- _SINT SCI_read (_UBYTE *buf, _UBYTE size, _BOOL sync, _SINT tmo_ms)
  バイトストリーム受信
- _UBYTE SCI_getLastError ()
  エラー値取得

3.13.1 詳解

シリアルポートドライバ

このモジュールはシリアルポートデバイスSCI3 へのインタフェースを提供する

参照

H83694 グループ_ハードウェアマニュアル.pdf 14 章

3.13.2 マクロ定義詳解

■#define SCI_MAX_TXBUF 128 送信バッファサイズ

sci.h の 31 行目に定義があります。

■#define SCI_MAX_RXBUF 128 受信バッファサイズ

sci.h の 36 行目に定義があります。

■#define SCI_ERR_PARITY (1U << 0) パリティエラー識別子

sci.h の 41 行目に定義があります。

■#define SCI_ERR_FRAMING (1U << 1) フレーミングエラー識別子

sci.h の 46 行目に定義があります。

■#define SCI_ERR_OVERRUN (1U << 2) オーバーランエラー識別子

sci.h の 51 行目に定義があります。

3.13.3 関数詳解

■void SCI_init ( ) SCI 初期化

SCI を使用可能に設定する。他API 使用前に必ず実行しておく。 送受信は割り込みにより駆動される。

データ長 8 ビット、パリティなし、ストップビット 1、フロー制御なし、 ボーレート 19200bps

■void SCI_communicate ( ) 送受信実行

SCI デバイスへ送受信処理を委託する。割り込みハンドラから呼ばれるのが前提。

■**_SINT** SCI_puts (const **_SBYTE** ∗ *str*) 文字列送信

文字列を送信する。文字列はNUL 終端されていなければならない。 エラーが発生しない限り送信完了までブロックする。

引数

in	*str*	送信文字列の先頭へのポインタ

戻り値

正の値	送信完了文字数
負の値	送信エラー

■**_SINT** SCI_write (const **_UBYTE** ∗ *data*, **_UBYTE** *size*, **_BOOL** *sync*, **_SINT** *tmo_ms*) バイトストリーム送信

汎用バイトデータを送信予約する。同期/非同期送信の両方に対応。 指定データは内部バッファへ格納され、次回の送信可能時に処理が実行される。

引数

in	*data*	送信データの先頭へのポインタ
in	*size*	送信サイズ
in	*sync*	_TRUE: 同期型。size 分送信できるか、タイムアウトするまで ブロックする。_FALSE: 非同期型。タイムアウトは無効
in	*tmo_ms*	同期送信時のタイムアウト値。ミリセカンド。

戻り値

正の値	送信完了サイズ
負の値	送信エラー

■**_SINT** SCI_read (**_UBYTE** ∗ *buf*, **_UBYTE** *size*, **_BOOL** *sync*, **_SINT** *tmo_ms*) バイトストリーム受信

内部バッファに溜まっているデータを取得する。同期/非同期受信の両方に対応。 内部バッファには非同期で随時受信データが溜まっていくため、定期的に このAPI を実行しないとバッファが溢れデータロスが発生する。

引数

in	*buf*	受信データを格納するバッファの先頭へのポインタ
in	*size*	受信サイズ

in	*sync*	_TRUE: 同期型。size 分受信できるか、タイムアウトするまで ブロックする。_FALSE: 非同期型。タイムアウトは無効
in	*tmo_ms*	同期受信時のタイムアウト値。ミリセカンド。

戻り値

正の値	受信完了サイズ
負の値	受信エラー

■**_UBYTE** SCI_getLastError ( ) エラー値取得

最後に発生したエラー番号を取得する。

戻り値

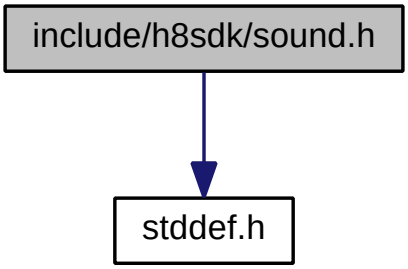
エラー番号。SCI_ERR_* で定義されているコード。

3.14 include/h8sdk/sound.h ファイル

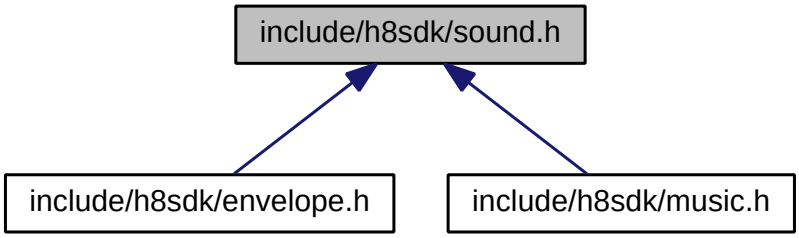
サウンドドライバ

```
#include "stddef.h"
```

sound.h の依存先関係図:



被依存関係図:



クラス

- struct SOUND_Envelope
  エンベロープ定義型
- struct SOUND_Tone
  音定義型

マクロ定義

- #define SOUND_MAX_PRONOUNCE 3
  最大同時発信音数
- #define SOUND_MAX_VOLUME 255
  ボリュームの最大値
- #define SOUND_PRONOUNCE_LEN 2
- #define SOUND_NA 0
  無音
- #define SOUND_C0 255
  オクターブ *0* ド
- #define SOUND_Cb0 241
  オクターブ *0* ド#
- #define SOUND_D0 228
  オクターブ *0* レ
- #define SOUND_Db0 214
  オクターブ *0* レ#
- #define SOUND_E0 203
  オクターブ *0* ミ
- #define SOUND_F0 192
  オクターブ *0* ファ
- #define SOUND_Fb0 180
  オクターブ *0* ファ#
- #define SOUND_G0 171
  オクターブ *0* ソ
- #define SOUND_Gb0 161
  オクターブ *0* ソ#
- #define SOUND_A0 152
  オクターブ *0* ラ
- #define SOUND_Ab0 143
  オクターブ *0* ラ#
- #define SOUND_H0 136
  オクターブ *0* シ
- #define SOUND_C1 128
  オクターブ *1* ド
- #define SOUND_Cb1 120
  オクターブ *1* ド#
- #define SOUND_D1 114
  オクターブ *1* レ
- #define SOUND_Db1 107
  オクターブ *1* レ#
- #define SOUND_E1 101
  オクターブ *1* ミ
- #define SOUND_F1 96
  オクターブ *1* ファ
- #define SOUND_Fb1 90
  オクターブ *1* ファ#
- #define SOUND_G1 85
  オクターブ *1* ソ
- #define SOUND_Gb1 80
  オクターブ *1* ソ#
- #define SOUND_A1 76

オクターブ 1 ラ
- #define SOUND_Ab1 72

オクターブ 1 ラ#
- #define SOUND_H1 68

オクターブ 1 シ
- #define SOUND_C2 64

オクターブ 2 ド
- #define SOUND_Cb2 60

オクターブ 2 ド#
- #define SOUND_D2 57

オクターブ 2 レ
- #define SOUND_Db2 54

オクターブ 2 レ#
- #define SOUND_E2 51

オクターブ 2 ミ
- #define SOUND_F2 48

オクターブ 2 ファ
- #define SOUND_Fb2 45

オクターブ 2 ファ#
- #define SOUND_G2 43

オクターブ 2 ソ
- #define SOUND_Gb2 40

オクターブ 2 ソ#
- #define SOUND_A2 38

オクターブ 2 ラ
- #define SOUND_Ab2 36

オクターブ 2 ラ#
- #define SOUND_H2 34

オクターブ 2 シ
- #define SOUND_C3 32

オクターブ 3 ド
- #define SOUND_Cb3 30

オクターブ 3 ド#
- #define SOUND_D3 29

オクターブ 3 レ
- #define SOUND_Db3 27

オクターブ 3 レ#
- #define SOUND_E3 25

オクターブ 3 ミ
- #define SOUND_F3 24

オクターブ 3 ファ
- #define SOUND_Fb3 23

オクターブ 3 ファ#
- #define SOUND_G3 21

オクターブ 3 ソ
- #define SOUND_Gb3 20

オクターブ 3 ソ#
- #define SOUND_A3 19

オクターブ 3 ラ
- #define SOUND_Ab3 18

オクターブ 3 ラ#
- #define SOUND_H3 17

オクターブ 3 シ

型定義

- typedef _BOOL(* SOUND_Api) ()
  再生操作リクエスト API の型

列挙型

- enum SOUND_State {
  SOUND_ST_STOP = 0,
  SOUND_ST_PLAY,
  SOUND_ST_REVERSE,
  SOUND_NUM_OF_STATE }
  再生状態リテラル型

関数

- void SOUND_init ()
  サウンドドライバ初期化
- SOUND_State SOUND_getState ()
  現在の再生状態取得
- void SOUND_setVolume (_UBYTE val)
  ボリューム設定
- void SOUND_setEvlpCycle (_UWORD val)
  エンベロープ周期設定
- void SOUND_setTone (_UBYTE idx, const SOUND_Tone *tone)
  音登録
- void SOUND_removeTone (_UBYTE idx)
  音削除
- _BOOL SOUND_play ()
  音再生リクエスト
- _BOOL SOUND_reverse ()
  音逆再生リクエスト
- _BOOL SOUND_stop ()
  停止リクエスト
- void SOUND_makePulse ()
  波形生成
- void SOUND_pronounce ()
  音データ発音

変数

- const SOUND_Api SOUND_state_handler [SOUND_NUM_OF_STATE]
  状態ハンドラ配列

3.14.1 詳解

サウンドドライバ

このモジュールは音を再生させるためのインタフェースを提供する。

H8-BASE2 で使用できる 3 つのタイマを使い、音程、音長、強弱を表わす矩形波を スピーカポートへ出力する。

注意

TimerA, TimerV, TimerW を他のモジュールで使用してはならない。

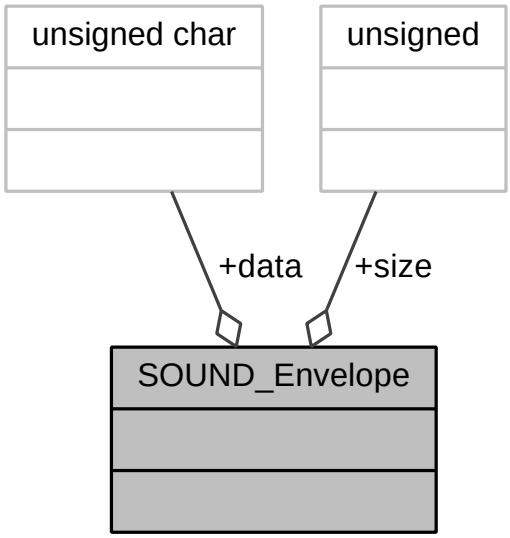
3.14.2 クラス詳解

■struct SOUND_Envelope エンベロープ定義型

音発生時の強弱レベルを定義する型

sound.h の 154 行目に定義があります。

SOUND_Envelope 連携図



クラスメンバ

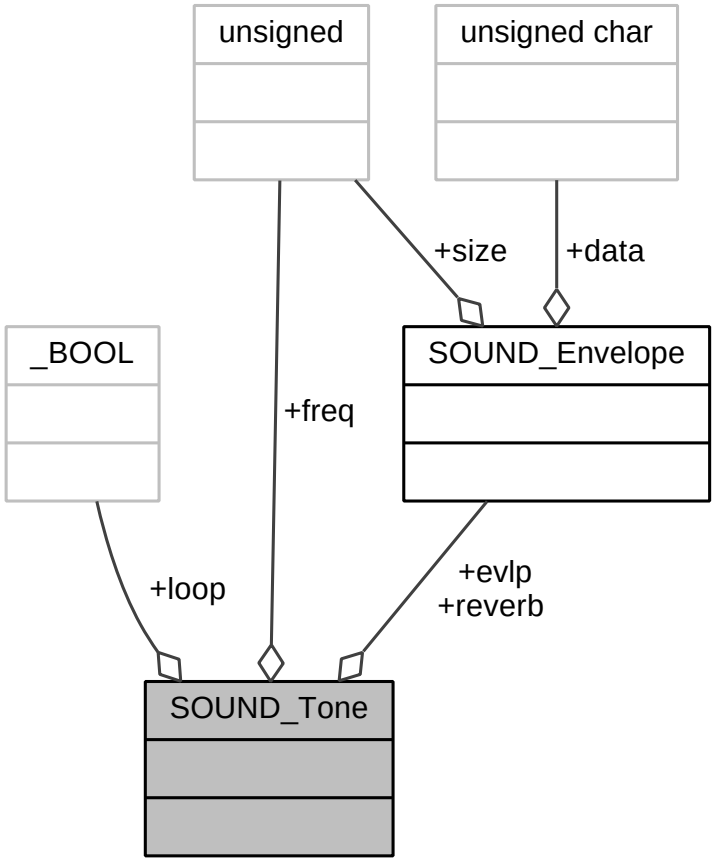
_UBYTE *	data	エンベロープデータ列の先頭へのポインタ
_UWORD	size	エンベロープデータ列のサイズ

■struct SOUND_Tone 音定義型

あるタイミングでの発生音を定義する型

sound.h の 167 行目に定義があります。

SOUND_Tone 連携図



クラスメンバ

_UBYTE	freq	音の周波数
SOUND_↩ Envelope *	evlp	この音を発生させるときのエンベロープ
SOUND_↩ Envelope *	reverb	再生後の残響効果。リバーブ
_BOOL	loop	リバーブのループフラグ

3.14.3 マクロ定義詳解

■#define SOUND_MAX_PRONOUNCE 3 最大同時発信音数

sound.h の 31 行目に定義があります。

■#define SOUND_MAX_VOLUME 255 ボリュームの最大値

sound.h の 33 行目に定義があります。

■#define SOUND_PRONOUNCE_LEN 2 sound.h の 35 行目に定義があります。

■#define SOUND_NA 0 無音

sound.h の 38 行目に定義があります。

■#define SOUND_C0 255 オクターブ 0 ド
sound.h の 40 行目に定義があります。

■#define SOUND_Cb0 241 オクターブ 0 ド#
sound.h の 42 行目に定義があります。

■#define SOUND_D0 228 オクターブ 0 レ
sound.h の 44 行目に定義があります。

■#define SOUND_Db0 214 オクターブ 0 レ#
sound.h の 46 行目に定義があります。

■#define SOUND_E0 203 オクターブ 0 ミ
sound.h の 48 行目に定義があります。

■#define SOUND_F0 192 オクターブ 0 ファ
sound.h の 50 行目に定義があります。

■#define SOUND_Fb0 180 オクターブ 0 ファ#
sound.h の 52 行目に定義があります。

■#define SOUND_G0 171 オクターブ 0 ソ
sound.h の 54 行目に定義があります。

■#define SOUND_Gb0 161 オクターブ 0 ソ#
sound.h の 56 行目に定義があります。

■#define SOUND_A0 152 オクターブ 0 ラ
sound.h の 58 行目に定義があります。

■#define SOUND_Ab0 143 オクターブ 0 ラ#
sound.h の 60 行目に定義があります。

■#define SOUND_H0 136 オクターブ 0 シ
sound.h の 62 行目に定義があります。

■#define SOUND_C1 128 オクターブ 1 ド
sound.h の 64 行目に定義があります。

■#define SOUND_Cb1 120 オクターブ 1 ド#
sound.h の 66 行目に定義があります。

■#define SOUND_D1 114 オクターブ 1 レ
sound.h の 68 行目に定義があります。

■#define SOUND_Db1 107 オクターブ 1 レ#
sound.h の 70 行目に定義があります。

■#define SOUND_E1 101 オクターブ 1 ミ
sound.h の 72 行目に定義があります。

■#define SOUND_F1 96 オクターブ 1 ファ
sound.h の 74 行目に定義があります。

■#define SOUND_Fb1 90 オクターブ 1 ファ#
sound.h の 76 行目に定義があります。

■#define SOUND_G1 85 オクターブ 1 ソ
sound.h の 78 行目に定義があります。

■#define SOUND_Gb1 80 オクターブ 1 ソ#
sound.h の 80 行目に定義があります。

■#define SOUND_A1 76 オクターブ 1 ラ
sound.h の 82 行目に定義があります。

■#define SOUND_Ab1 72 オクターブ 1 ラ#
sound.h の 84 行目に定義があります。

■#define SOUND_H1 68 オクターブ 1 シ
sound.h の 86 行目に定義があります。

■#define SOUND_C2 64 オクターブ 2 ド
sound.h の 88 行目に定義があります。

■#define SOUND_Cb2 60 オクターブ 2 ド#
sound.h の 90 行目に定義があります。

■#define SOUND_D2 57 オクターブ 2 レ
sound.h の 92 行目に定義があります。

■#define SOUND_Db2 54 オクターブ 2 レ#
sound.h の 94 行目に定義があります。

■#define SOUND_E2 51 オクターブ 2 ミ
sound.h の 96 行目に定義があります。

■#define SOUND_F2 48 オクターブ 2 ファ
sound.h の 98 行目に定義があります。

■#define SOUND_Fb2 45 オクターブ 2 ファ#
sound.h の 100 行目に定義があります。

■#define SOUND_G2 43 オクターブ 2 ソ
sound.h の 102 行目に定義があります。

■#define SOUND_Gb2 40 オクターブ 2 ソ#
sound.h の 104 行目に定義があります。

■#define SOUND_A2 38 オクターブ 2 ラ
sound.h の 106 行目に定義があります。

■#define SOUND_Ab2 36 オクターブ 2 ラ#
sound.h の 108 行目に定義があります。

■#define SOUND_H2 34 オクターブ 2 シ
sound.h の 110 行目に定義があります。

■#define SOUND_C3 32 オクターブ 3 ド
sound.h の 112 行目に定義があります。

■#define SOUND_Cb3 30 オクターブ 3 ド#
sound.h の 114 行目に定義があります。

■#define SOUND_D3 29 オクターブ 3 レ
sound.h の 116 行目に定義があります。

■#define SOUND_Db3 27 オクターブ 3 レ#
sound.h の 118 行目に定義があります。

■#define SOUND_E3 25 オクターブ 3 ミ
sound.h の 120 行目に定義があります。

■#define SOUND_F3 24 オクターブ 3 ファ
sound.h の 122 行目に定義があります。

■#define SOUND_Fb3 23 オクターブ 3 ファ#
sound.h の 124 行目に定義があります。

■#define SOUND_G3 21 オクターブ 3 ソ
sound.h の 126 行目に定義があります。

■#define SOUND_Gb3 20 オクターブ 3 ソ#
sound.h の 128 行目に定義があります。

■#define SOUND_A3 19 オクターブ 3 ラ
sound.h の 130 行目に定義があります。

■#define SOUND_Ab3 18 オクターブ 3 ラ#
sound.h の 132 行目に定義があります。

■#define SOUND_H3 17 オクターブ 3 シ
sound.h の 134 行目に定義があります。

3.14.4 型定義詳解

■typedef **_BOOL**(* SOUND_Api) () 再生操作リクエストAPI の型
sound.h の 182 行目に定義があります。

3.14.5 列挙型詳解

■enum **SOUND_State** 再生状態リテラル型

列挙値

SOUND_ST_STOP
SOUND_ST_PLAY
SOUND_ST_REVERSE
SOUND_NUM_OF_STATE

sound.h の 140 行目に定義があります。

```
141 {
142     SOUND_ST_STOP = 0,
143     SOUND_ST_PLAY,
144     SOUND_ST_REVERSE,
145
146     SOUND_NUM_OF_STATE
147 } SOUND_State;
```

3.14.6 関数詳解

■void SOUND_init ( ) サウンドドライバ初期化

タイマとポートの初期化を行う。他API 使用前に必ず実行する。

■**SOUND_State** SOUND_getState ( ) 現在の再生状態取得

戻り値

再生状態リテラル

■void SOUND_setVolume (_**UBYTE** *val*) ボリューム設定

再生音量を設定する。0〜SOUND_MAX_VOLUME いつでも設定可能。

引数

in	*val*	ボリューム値

■void SOUND_setEvlpCycle (_**UWORD** *val*) エンベロープ周期設定

エンベロープをセットする周期を設定する。 この設定値は再生音質とボリューム調節の細かさに影響し、両者はトレードオフの 関係にある。初期値は 256。

引数

in	*val*	設定値

■void SOUND_setTone (_**UBYTE** *idx*, const **SOUND_Tone** ∗ *tone*) 音登録

音を登録する。SOUND_MAX_PRONOUNCE 個のバンクに同時にセットすることができる。 いつでも設定可能。

引数

in	*idx*	バンクのインデックス。0〜SOUND_MAX_PRONOUNCE-1

in	*tone*	設定するSOUND_Tone へのポインタ 無音 (SOUND_Tone::freq == 0)、またはSOUND_Tone::evlp がNULL のデータを 登録することはできない。

■void SOUND_removeTone (**_UBYTE** *idx*) 音削除

バンクを指定して設定された音を削除する。 いつでも設定可能。

引数

in	*idx*	削除するバンクのインデックス。0〜SOUND_MAX_PRONOUNCE-1

■**_BOOL** SOUND_play ( ) 音再生リクエスト

音の再生。エンベロープは順方向。

戻り値

_TRUE	リクエスト受諾
_FALSE	リクエスト拒否

■**_BOOL** SOUND_reverse ( ) 音逆再生リクエスト

音の再生。エンベロープは逆方向。

戻り値

_TRUE	リクエスト受諾
_FALSE	リクエスト拒否

■**_BOOL** SOUND_stop ( ) 停止リクエスト

再生中の音を停止させる。

戻り値

_TRUE	リクエスト受諾
_FALSE	リクエスト拒否

■void SOUND_makePulse ( ) 波形生成

SOUND_pronounce により解析された音情報により実際に矩形波を生成させる。 TimerV 割り込みハンドラからこのAPI を呼ぶ必要がある。割り込みフラグは内部で クリアしている。

参照

■void SOUND_pronounce ( ) 音データ発音

波形生成のためのタイマカウント情報などを設定する。 TimerA 割り込みハンドラからこのAPI を呼ぶ必要がある。割り込みフラグはクリア されない。

3.14.7 変数詳解

■const **SOUND_Api** SOUND_state_handler[**SOUND_NUM_OF_STATE**] 状態ハンドラ配列

この配列に再生状態リテラルをインデックスとして与えれば、リテラルに応じた ハンドラが呼ばれる。

参照

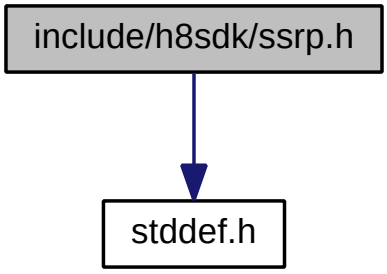
SOUND_State

3.15 include/h8sdk/ssrp.h ファイル

SSRP 通信プロトコル

```
#include "stddef.h"
```

ssrp.h の依存先関係図:



クラス

- struct SSRP_Header
  SSRP パケットヘッダ

マクロ定義

- #define SSRP_MAX_NODE 16
  接続ノード最大数
- #define SSRP_MAX_DATA 255
  最大送受信データサイズ
- #define SSRP_ADDR_INVALID 0
  無効なアドレス
- #define SSRP_ADDR_BROADCAST 0xff
  ブロードキャストアドレス
- #define SSRP_TMO_RECV 255
  受信タイムアウト値
- #define SSRP_TMO_SEND -1
  送信タイムアウト値
- #define SSRP_CMD_LOOP 0x00
  LOOP コマンド値
- #define SSRP_CMD_JOIN 0x01
  JOIN コマンド値
- #define SSRP_CMD_LEAVE 0x02
  LEAVE コマンド値
- #define SSRP_VAL_PREAMBLE 0xaa
  プリアンブル値
- #define SSRP_PACKET_SPC (1U << 8)
  定期処理の送信間隔
- #define SSRP_CMD_EX 0x80

外部モジュール用コマンドマスク

- #define SSRP_CMD_INVALID 0xff
  無効なコマンド値
- #define SSRP_FLG_EVT 0x00
  イベントフラグ値
- #define SSRP_FLG_REQ 0x01
  リクエストフラグ値
- #define SSRP_FLG_RES 0x02
  リクエストフラグ値
- #define SSRP_FLG_INVALID 0xff
  無効なフラグ値
- #define SSRP_TRXID_INVALID ((1U << (sizeof(SSRP_TransactionId) << 3)) - 1)
  無効なトランザクション*ID*
- #define SSRP_COMMAND_EQ(hp, com, flg) (((hp)->command == (com)) && ((hp)->flag == (flg)))
  コマンド/フラグ一致判定
- #define SSRP_INVALIDATE_PACKET(hp) ((hp)->from = SSRP_ADDR_INVALID, (hp)->to = SSRP_ADDR_INVALID)
  パケット無効化
- #define SSRP_IS_INVALID_PACKET(hp) (((hp)->from == SSRP_ADDR_INVALID) || ((hp)->to == SSRP_ADDR_INVALID))
  パケット無効判定

型定義

- typedef _UBYTE SSRP_Address
  アドレス型定義
- typedef _UBYTE SSRP_Command
  コマンド型定義
- typedef _UBYTE SSRP_Flag
  フラグ定義型
- typedef _UWORD SSRP_TransactionId
  トランザクション*ID* の定義型

列挙型

- enum SSRP_Transaction {
  SSRP_TRX_RECV = 0,
  SSRP_TRX_SEND,
  SSRP_NUM_OF_TRX }
  トランザクションタイプ識別子
- enum SSRP_Connection {
  SSRP_CON_FRONT = 0,
  SSRP_CON_BACK,
  SSRP_NUM_OF_CON }
  コネクション種別定義型

関数

- void SSRP_init (SSRP_Address myaddr)
  SSRP **初期化**
- void SSRP_start ()
  接続開始
- void SSRP_end ()
  接続終了
- _SINT SSRP_send (SSRP_Header *header, const _UBYTE *data, _UBYTE len, _BOOL sync)
  汎用パケット送信
- _SINT SSRP_sendto (SSRP_Address to, SSRP_Command cmd, SSRP_Flag flg, const _UBYTE *data, _UBYTE len, SSRP_TransactionId *trx_id, _BOOL sync)
  パケット送信
- _SINT SSRP_recvfrom (SSRP_Header **header_p, _UBYTE **data_p)
  パケット受信
- void SSRP_shutdown (SSRP_Transaction tr)
  送受信トランザクションをリセット
- SSRP_Address SSRP_getNode (_UBYTE idx)
  ノードアドレス取得
- SSRP_Address SSRP_getPear (SSRP_Connection con)
  接続先ノードアドレス取得
- _UBYTE SSRP_getTotalPear ()
  ネットワークのノード数を取得する
- _BOOL SSRP_ready ()
  送受信可能状態の判定
- void SSRP_exec ()
  定期処理の実行

変数

- _SBYTE SSRP_my_addr []
  自ノードアドレス文字列

3.15.1 詳解

SSRP 通信プロトコル

このモジュールは、一方通行のリング型ネットワーク上での通信プロトコル SSRP(Simple Single Ring Protocol) のシンプルな実装パターンである。

参照

ifstub.h

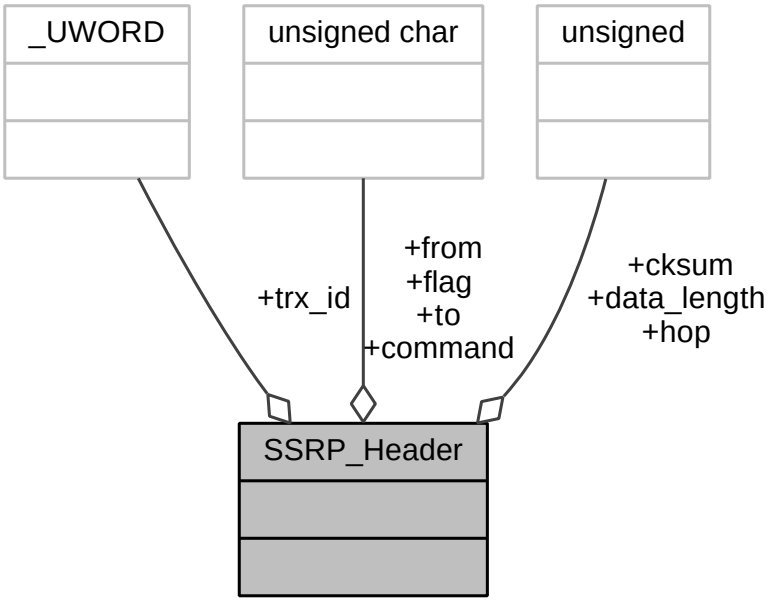
3.15.2 クラス詳解

■struct SSRP_Header SSRP パケットヘッダ

SSRP パケットの情報を表すヘッダ。パケットはこのヘッダとデータのペイロード から構成される。ヘッダのチェックサムは毎回SSRP 内部でチェックされ、不一致 のパケットは受信しても破棄されるが、ペイロード部分の有効性に関しては SSRP 内部では関知しない。

ssrp.h の 185 行目に定義があります。

SSRP_Header 連携図



クラスメンバ

SSRP_Address	from	送信先アドレス
SSRP_Address	to	送信元アドレス
SSRP_↩ Command	command	コマンド
SSRP_Flag	flag	パケットフラグ
_UBYTE	hop	ホップ数 各ノードによりパケットがリレーされるとこの値がインクリメントされる
_UBYTE	data_length	データペイロード長
SSRP_↩ TransactionId	trx_id	トランザクションID
_UWORD	cksum	ヘッダのチェックサム

3.15.3 マクロ定義詳解

■#define SSRP_MAX_NODE 16 接続ノード最大数

ssrp.h の 32 行目に定義があります。

■#define SSRP_MAX_DATA 255 最大送受信データサイズ

パケットのペイロード最大値

ssrp.h の 39 行目に定義があります。

■#define SSRP_ADDR_INVALID 0 無効なアドレス

ssrp.h の 43 行目に定義があります。

■#define SSRP_ADDR_BROADCAST 0xff ブロードキャストアドレス

ssrp.h の 47 行目に定義があります。

■#define SSRP_TMO_RECV 255 受信タイムアウト値
ssrp.h の 51 行目に定義があります。

■#define SSRP_TMO_SEND -1 送信タイムアウト値
ssrp.h の 55 行目に定義があります。

■#define SSRP_CMD_LOOP 0x00 LOOP コマンド値
ネットワーク接続ノードをカウントするコマンド
ssrp.h の 62 行目に定義があります。

■#define SSRP_CMD_JOIN 0x01 JOIN コマンド値
ネットワークへ参加するコマンド
ssrp.h の 69 行目に定義があります。

■#define SSRP_CMD_LEAVE 0x02 LEAVE コマンド値
ネットワークから離脱するコマンド
ssrp.h の 76 行目に定義があります。

■#define SSRP_VAL_PREAMBLE 0xaa プリアンブル値
パケット送信前に付加するプリアンブル値
ssrp.h の 83 行目に定義があります。

■#define SSRP_PACKET_SPC (1U << 8) 定期処理の送信間隔
JOIN、LOOP など、定期処理時の送信間隔
ssrp.h の 90 行目に定義があります。

■#define SSRP_CMD_EX 0x80 外部モジュール用コマンドマスク
ssrp.h の 115 行目に定義があります。

■#define SSRP_CMD_INVALID 0xff 無効なコマンド値
ssrp.h の 119 行目に定義があります。

■#define SSRP_FLG_EVT 0x00 イベントフラグ値
このフラグが設定されたパケットは通知のみのイベントである。
ssrp.h の 133 行目に定義があります。

■#define SSRP_FLG_REQ 0x01 リクエストフラグ値
このフラグが設定されたパケットはレスポンスが必要なリクエストである。
ssrp.h の 139 行目に定義があります。

■#define SSRP_FLG_RES 0x02 リクエストフラグ値
このフラグが設定されたパケットはリクエストに対するレスポンスである。
ssrp.h の 145 行目に定義があります。

■#define SSRP_FLG_INVALID 0xff 無効なフラグ値
ssrp.h の 149 行目に定義があります。

■#define SSRP_TRXID_INVALID ((1U << (sizeof(SSRP_TransactionId) << 3)) - 1) 無効なトランザクションID

ssrp.h の 175 行目に定義があります。

■#define SSRP_COMMAND_EQ(*hp*, *com*, *flg*) (((hp)->command == (com)) && ((hp)->flag == (flg))) コマンド/フラグ一致判定

ヘッダのコマンドエリアとフラグエリアが指定されたものと同じかどうかの 判定を行う

引数

in	*hp*	比較元ヘッダへのポインタ
in	*com*	判定するコマンド値
in	*flg*	判定するフラグ値

戻り値

コマンド/フラグ両方一致していれば正の値。そうでなければ 0

ssrp.h の 237 行目に定義があります。

■#define SSRP_INVALIDATE_PACKET(*hp*) ((hp)->from = **SSRP_ADDR_INVALID**, (hp)->to = **SSRP_ADDR_INVALID**) パケット無効化

指定されたヘッダのパケットを初期化して無効とする。

引数

in	*hp*	無効化するヘッダへのポインタ

ssrp.h の 247 行目に定義があります。

■#define SSRP_IS_INVALID_PACKET(*hp*) (((hp)->from == **SSRP_ADDR_INVALID**) || ((hp)->to == **SSRP_ADDR_INVALID**)) パケット無効判定

指定されたヘッダのパケットが無効なものかどうかを判定する

引数

in	*hp*	判定するヘッダへのポインタ

戻り値

無効化されたパケットならば正の値。そうでなければ 0

ssrp.h の 259 行目に定義があります。

3.15.4 型定義詳解

■typedef _**UBYTE SSRP_Address** アドレス型定義

ssrp.h の 95 行目に定義があります。

■typedef _**UBYTE SSRP_Command** コマンド型定義

このコマンドによって他ノードとデータの連携を取る。 0x00〜0x7f までは内部用に予約されている。外部モジュールから送信するパケット のコマンド値はSSRP_CMD_EX とOR を取らなければならない。

ssrp.h の 111 行目に定義があります。

■typedef _**UBYTE SSRP_Flag** フラグ定義型

パケットの属性を表すフラグ型。このフラグにより、リクエスト/レスポンス/ イベントかどうかの判定を行う。

ssrp.h の 127 行目に定義があります。

■typedef _**UWORD SSRP_TransactionId** トランザクションID の定義型

トランザクションを識別するためのID。自ノードからパケットを発信する毎に インクリメントされる送信カウンタ。パケットヘッダに毎回設定される。

ssrp.h の 171 行目に定義があります。

3.15.5 列挙型詳解

■enum **SSRP_Transaction** トランザクションタイプ識別子

列挙値

SSRP_TRX_RECV 受信タイプ

SSRP_TRX_SEND 送信タイプ

SSRP_NUM_OF_TRX トランザクションタイプの数

ssrp.h の 154 行目に定義があります。

```
155 {
156     SSRP_TRX_RECV = 0,
158     SSRP_TRX_SEND,
160
161     SSRP_NUM_OF_TRX
163 } SSRP_Transaction;
```

■enum **SSRP_Connection** コネクション種別定義型

SSRP はリングプロトコルなので接続は前と後の二つ。

列挙値

SSRP_CON_FRONT 前への接続

SSRP_CON_BACK 後への接続

SSRP_NUM_OF_CON 接続種別の数

ssrp.h の 214 行目に定義があります。

```
215 {
216     SSRP_CON_FRONT = 0,
218     SSRP_CON_BACK,
220
221     SSRP_NUM_OF_CON
223 } SSRP_Connection;
```

3.15.6 関数詳解

■void SSRP_init (**SSRP_Address** *myaddr*) SSRP 初期化

SSRP モジュールを初期化する。他API 使用前に必ず実行する。

■void SSRP_start ( ) 接続開始

自ノードをSSRP ネットワークに参加させる。以降SSRP_exec により自律接続処理が 行われるようになる。

■void SSRP_end ( ) 接続終了

自ノードをSSRP ネットワークから脱退させる。以降のSSRP_exec はパケット ルーティング処理のみとなる。

■ **_SINT** SSRP_send (**SSRP_Header** ∗ *header*, const **_UBYTE** ∗ *data*, **_UBYTE** *len*, **_BOOL** *sync*) 汎用パケット送信

ヘッダを指定してパケットを送信する。非同期送信の場合、ペイロードを指定サイズ 分送り切れないことがあるが、この場合次以降もトランザクションは継続しており、 指定する data ポインタの位置を変えれば続きからデータを送ることができる。 送信を諦めてトランザクションをリセットする場合はSSRP_shutdown を使用する。

引数

in	*header*	送信するヘッダへのポインタ
in	*data*	送信するデータペイロードへのポインタ
in	*len*	送信するデータのサイズ
in	*sync*	_TRUE:同期送信 _FALSE:非同期送信

戻り値

正の値	送信完了したデータサイズ
負の値	送信エラー

■ **_SINT** SSRP_sendto (**SSRP_Address** *to*, **SSRP_Command** *cmd*, **SSRP_Flag** *flg*, const **_UBYTE** ∗ *data*, **_UBYTE** *len*, **SSRP_TransactionId** ∗ *trx_id*, **_BOOL** *sync*) パケット送信

パラメータを指定してパケットを送信する。自動的に自ノードからの送信パケット となる。それ以外の特徴はSSRP_send と同じ。

引数

in	*to*	送信先アドレス
in	*cmd*	コマンド
in	*flg*	フラグ
in	*data*	データペイロードの先頭へのポインタ
in	*len*	送信するデータサイズ
out	*trx_id*	トランザクションID を保存する領域へのポインタ。NULL なら 無視される
in	*sync*	_TRUE:同期送信 _FALSE:非同期送信

戻り値

正の値	送信完了したデータサイズ
負の値	送信エラー

■ **_SINT** SSRP_recvfrom (**SSRP_Header** ∗∗ *header_p*, **_UBYTE** ∗∗ *data_p*) パケット受信

現在内部で保持されているパケットデータを取得する。実際のパケット受信は SSRP_exec により実行され 1 パケット分のヘッダとデータが保持されているため、 これはそれらを取得するだけのAPI である。よってブロックはしない。 また、取得できるのは自ノード宛かブロードキャストパケットのみである。

引数

out	*header_p*	受信パケットのヘッダポインタを格納するポインタ
out	*data_p*	受信パケットのデータペイロードポインタを格納するポインタ

戻り値

正の値	受信パケットのデータペイロードサイズ
負の値	受信パケット無し

■void SSRP_shutdown (**SSRP_Transaction** *tr*) 送受信トランザクションをリセット

現在処理中のトランザクションをリセットして未送信/未受信状態へ戻す。

引数

in	*tr*	リセットするトランザクションの種別

■**SSRP_Address** SSRP_getNode (**_UBYTE** *idx*) ノードアドレス取得

ネットワークに参加しているノードのアドレスを取得する。

引数

in	*idx*	取得するノードのインデックス

戻り値

ノードのアドレス

■**SSRP_Address** SSRP_getPear (**SSRP_Connection** *con*) 接続先ノードアドレス取得

接続されているノードのアドレスを接続種別に取得する。

引数

in	*con*	取得するノードの接続種別

戻り値

ノードのアドレス

■**_UBYTE** SSRP_getTotalPear ( ) ネットワークのノード数を取得する

SSRP ネットワークに参加しているノードの総数を取得する。自ノードも含む。

戻り値

ノード総数

■**_BOOL** SSRP_ready ( ) 送受信可能状態の判定

SSRP は最初に他ノードとのコネクションを張るが、この処理が完了しないと 送受信処理は行えないため、その状態判定に使用する。

戻り値

_TRUE	接続済み。送受信可能
_FALSE	未接続。送受信不可

■void SSRP_exec ( ) 定期処理の実行

接続処理やパケットリレー、パケット受信などを行う。このAPI を定期的に実行 することでSSRP が駆動する。SSRP_start が未実行の場合はパケットリレーのみが 行われる。

3.15.7 変数詳解

■**_SBYTE** SSRP_my_addr[] 自ノードアドレス文字列

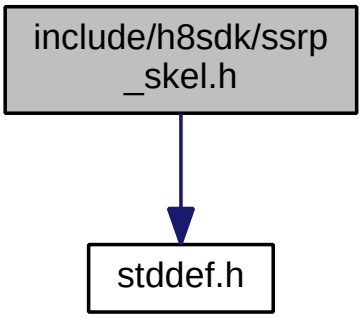
SSRP_Addr の値が 2 桁の 16 進数字列で設定される。

3.16 include/h8sdk/ssrp_skel.h ファイル

SSRP スケルトン

```
#include "stddef.h"
```

ssrp_skel.h の依存先関係図:



マクロ定義

- #define SSRP_SKEL_LOOPBACK_ADDR 0xfe

関数

- _SINT SSRP_SKEL_write (const _UBYTE *data, _UBYTE size, _BOOL sync, _SINT tmo_ms)
  バイトストリーム送信
- _SINT SSRP_SKEL_read (_UBYTE *buf, _UBYTE size, _BOOL sync, _SINT tmo_ms)
  バイトストリーム受信

3.16.1 詳解

SSRP スケルトン

このモジュールはSSRP 通信をループバックで行うためのスケルトンである。

参照

ssrp.h

3.16.2 マクロ定義詳解

■#define SSRP_SKEL_LOOPBACK_ADDR 0xfe ssrp_skel.h の 28 行目に定義があります。

3.16.3 関数詳解

■**_SINT** SSRP_SKEL_write (const **_UBYTE** ∗ *data*, **_UBYTE** *size*, **_BOOL** *sync*, **_SINT** *tmo_ms*) バイトストリーム送信

常に成功する送信API。送信データは破棄される。 その他仕様は ifstub.h に準拠。

参照

ifstub.h

■**_SINT** SSRP_SKEL_read (**_UBYTE** ∗ *buf*, **_UBYTE** *size*, **_BOOL** *sync*, **_SINT** *tmo_ms*) バイトストリーム受信

SSRP の内部状態に応じて常に適切な受信データを返す。 その他仕様は ifstub.h に準拠。

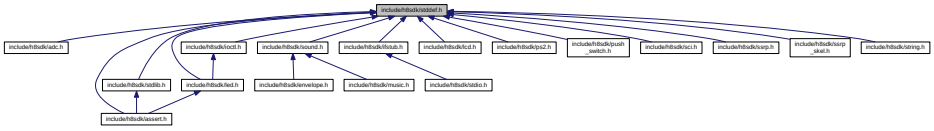
参照

ifstub.h

3.17 include/h8sdk/stddef.h ファイル

共通型、リテラル定義

被依存関係図:



マクロ定義

- #define _sizeof_array(x) (sizeof(x)/sizeof(x[0]))
  配列の要素数取得
- #define _offsetof(type, member) ((_UWORD)&((type∗)0)->(member))
  構造体メンバの要素へのオフセットを取得

型定義

- typedef signed char _SBYTE
  符号あり *8* ビット整数型
- typedef unsigned char _UBYTE
  符号なし *8* ビット整数型
- typedef signed short _SWORD

符号あり *16* ビット整数型

- typedef unsigned short _UWORD
  符号なし *16* ビット整数型
- typedef signed int _SINT
  符号あり *16* ビット整数型
- typedef unsigned int _UINT
  符号なし *16* ビット整数型
- typedef signed long _SDWORD
  符号あり *32* ビット整数型
- typedef unsigned long _UDWORD
  符号なし *32* ビット整数型

列挙型

- enum _BOOL {
  _FALSE = 0,
  _TRUE = ~_FALSE }
  ブール型

3.17.1 詳解

共通型、リテラル定義

3.17.2 マクロ定義詳解

■#define _sizeof_array(*x*) (sizeof(x)/sizeof(x[0])) 配列の要素数取得

引数

in	*x*	取得する配列名

戻り値

配列の要素数

stddef.h の 84 行目に定義があります。

■#define _offsetof(*type*, *member*) ((_**UWORD**)&((type*)0)->(member)) 構造体メンバの要素へのオフセットを取得

引数

in	*type*	member の構造体型タイプ
in	*member*	オフセットを取得するメンバ名

戻り値

member までのオフセット

stddef.h の 94 行目に定義があります。

3.17.3 型定義詳解

■typedef signed char **_SBYTE** 符号あり 8 ビット整数型

stddef.h の 48 行目に定義があります。

■typedef unsigned char **_UBYTE** 符号なし 8 ビット整数型

stddef.h の 50 行目に定義があります。

■typedef signed short **_SWORD** 符号あり 16 ビット整数型

stddef.h の 52 行目に定義があります。

■typedef unsigned short **_UWORD** 符号なし 16 ビット整数型

stddef.h の 54 行目に定義があります。

■typedef signed int **_SINT** 符号あり 16 ビット整数型

stddef.h の 56 行目に定義があります。

■typedef unsigned int **_UINT** 符号なし 16 ビット整数型

stddef.h の 58 行目に定義があります。

■typedef signed long **_SDWORD** 符号あり 32 ビット整数型

stddef.h の 60 行目に定義があります。

■typedef unsigned long **_UDWORD** 符号なし 32 ビット整数型

stddef.h の 62 行目に定義があります。

3.17.4 列挙型詳解

■enum **_BOOL** ブール型

2 値判定用ブール型。この型同士での評価以外には使用してはならない。

列挙値

_FALSE 偽

_TRUE 真

stddef.h の 69 行目に定義があります。

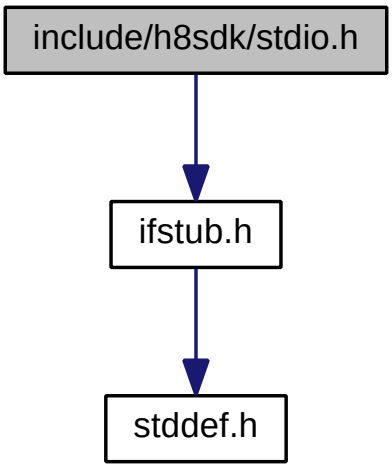
```
70 {
71     _FALSE = 0,
73     _TRUE = ~_FALSE
75 } _BOOL;
```

3.18 include/h8sdk/stdio.h ファイル

I/O ユーティリティライブラリ

```
#include "ifstub.h"
```

stdio.h の依存先関係図:



マクロ定義

- #define _dprintf(x) (_printf x)
  デバッグ用文字列出力

関数

- void _printf (const _SBYTE *format,...)
  書式付き文字出力

変数

- IFSTUB_Type _STDOUT
  標準出力ポート。デフォルトはシリアルポート
- IFSTUB_Type _STDIN
  標準入力ポート。デフォルトはシリアルポート
- IFSTUB_Type _STDERR
  標準エラー出力ポート。デフォルトはシリアルポート

3.18.1 詳解

I/O ユーティリティライブラリ

3.18.2 マクロ定義詳解

■#define _dprintf(x) (**_printf** x) デバッグ用文字列出力

引数

in	*x*	出力書式とパラメータ。必ず括弧付きで呼び出すこと。 e.g. _dprintf((”aaa = %d\n”, aaa));

stdio.h の 59 行目に定義があります。

3.18.3 関数詳解

■void _printf (const _**SBYTE** ∗ *format*, ...) 書式付き文字出力

指定フォーマットのパターンに従って文字列を 呼び出し前にシリアルドライバモジュールが初期化されている必要がある。

引数

in	*format*	出力書式。使用できる変換形式は ”%d”(符号あり 10 進 数値), ”%u”(符号なし 10 進数値), ”%x”(16 進数値), ”%o”(8 進数値), ”%s”(文字列) のみ。”%”を出力するときは ”%%” と記述する。
in	...	出力パラメータ列

参照

sci.h

3.18.4 変数詳解

■**IFSTUB_Type** _STDOUT 標準出力ポート。デフォルトはシリアルポート

■**IFSTUB_Type** _STDIN 標準入力ポート。デフォルトはシリアルポート

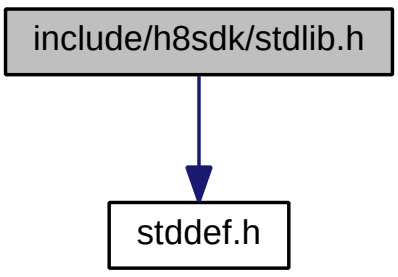
■**IFSTUB_Type** _STDERR 標準エラー出力ポート。デフォルトはシリアルポート

3.19 include/h8sdk/stdlib.h ファイル

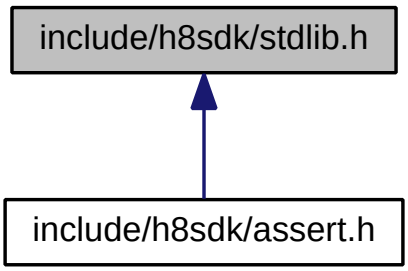
ユーティリティライブラリ

```
#include "stddef.h"
```

stdlib.h の依存先関係図:



被依存関係図:



マクロ定義

- #define NULL ((void∗)0)
  NULL **ポインタ識別子**
- #define _usleep(us)
  マイクロ秒スリープ
- #define _msleep(ms)
  ミリ秒スリープ
- #define _next_ring(now, last) ((now) == (last) ? 0: (now) + 1)
  リングバッファの次のインデックスを取得
- #define _prev_ring(now, last) ((now) == 0 ? (last): (now) - 1)
  リングバッファの前のインデックスを取得
- #define _hash(x, size) ((x) % (size))
  ハッシュ関数

関数

- _UWORD _check_sum (void ∗ary, _UWORD sz)
  チェックサム計算
- _UINT _itoa (_UINT in, _SINT base, _SBYTE ∗out, _UINT size)
  数値文字列変換
- _SDWORD _rand ()
  乱数取得
- void _srand (_SDWORD s)
  乱数の種設定

3.19.1 詳解

ユーティリティライブラリ

3.19.2 マクロ定義詳解

■#define NULL ((void∗)0) NULL ポインタ識別子

無効なポインタを表すリテラル

覚え書き

このリテラルのみH8SDK 命名規則の例外とする

stdlib.h の 31 行目に定義があります。

■#define _usleep(us) **値:**

```
do                                          \
    {                                           \
        _UWORD i;                               \
        const _UWORD border = (us) * 3;         \
        for (i = 0; i < border; i++);           \
    }                                           \
    while (_FALSE)
```

マイクロ秒スリープ

スリープ関数。ただのビジーループのため精度は環境によって大きく変化する

引数

in	*us*	スリープする時間。マイクロ秒

stdlib.h の 40 行目に定義があります。

■#define _msleep(ms) **値:**

```
do                                          \
    {                                           \
        _UWORD i;                               \
        const _UWORD border = (ms);             \
        for (i = 0; i < border; i++)            \
        {                                       \
            _usleep(1000);                      \
        }                                       \
    }                                           \
    while (_FALSE)
```

ミリ秒スリープ

スリープ関数。ただのビジーループのため精度は環境によって大きく変化する

引数

in	*ms*	ウェイトする時間。ミリ秒

stdlib.h の 56 行目に定義があります。

■#define _next_ring(now, last) ((now) == (last) ? 0: (now) + 1) リングバッファの次のインデックスを取得

引数

in	*now*	現在のインデックス値
in	*last*	最後のインデックス値

戻り値

次のインデックス値

stdlib.h の 76 行目に定義があります。

■#define _prev_ring(now, last) ((now) == 0 ? (last): (now) - 1) リングバッファの前のインデックスを取得

引数

in	*now*	現在のインデックス値
in	*last*	最後のインデックス値

戻り値

前のインデックス値

stdlib.h の 85 行目に定義があります。

■#define _hash(*x*, *size*) ((x) % (size)) ハッシュ関数
引数

in	*x*	ハッシュ計算対象
in	*size*	ハッシュテーブルのサイズ

戻り値

x のハッシュ値

stdlib.h の 95 行目に定義があります。

3.19.3 関数詳解

■**_UWORD** _check_sum (void * *ary*, **_UWORD** *sz*) チェックサム計算

16 ビット単位で指定データの総和を取る
引数

in	*ary*	データ配列先頭へのポインタ
in	*sz*	データサイズ

return チェックサム値

■**_UINT** _itoa (**_UINT** *in*, **_SINT** *base*, **_SBYTE** * *out*, **_UINT** *size*) 数値文字列変換

数値を指定進数表記の数字列に変換する。NUL 終端はしない。 10 進数を越える数字列の場合は 10 以降アルファベット表記となる。
引数

in	*in*	変換する数値。符号無し整数。
in	*base*	進数
out	*out*	結果を格納する文字配列先頭へのポインタ
in	*size*	out のサイズ

戻り値

変換桁数

■**_SDWORD** _rand ( ) 乱数取得

乱数値を取得する。精度は 15bit。

戻り値

乱数値

■void _srand (_SDWORD *s*) 乱数の種設定

乱数の種をセットする。

引数

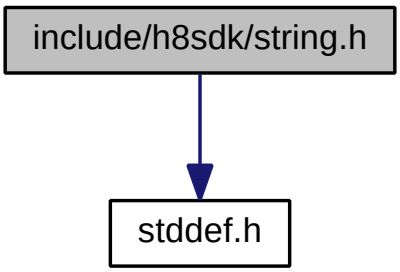
in	*s*	乱数の種

3.20 include/h8sdk/string.h ファイル

文字列操作ユーティリティライブラリ

```
#include "stddef.h"
```

string.h の依存先関係図:



関数

- _UWORD _strlen (const _SBYTE *str)
  文字列長取得
- void * _memcpy (void *dst, const void *src, _UWORD size)
  メモリ領域コピー
- void * _memset (void *dst, _SINT c, _UWORD size)
  メモリ領域セット

3.20.1 詳解

文字列操作ユーティリティライブラリ

3.20.2 関数詳解

■**_UWORD** _strlen (const **_SBYTE** * *str*) 文字列長取得

指定文字列先頭からNUL'\0' 文字までの数を返す。NUL は含まない。

引数

in	*str*	長さを取得する文字列先頭へのポインタ

戻り値

文字列長

■void* _memcpy (void * *dst*, const void * *src*, _**UWORD** *size*) メモリ領域コピー

メモリ領域を指定バイト分コピーする。領域は重なってはならない。

引数

in	*dst*	コピー先へのポインタ
in	*src*	コピー元へのポインタ
in	*size*	コピーバイト数

戻り値

dst へのポインタ

■void* _memset (void * *dst*, _**SINT** *c*, _**UWORD** *size*) メモリ領域セット

メモリ領域に指定値をセットする。

引数

in	*dst*	セット先へのポインタ
in	*c*	セットする値
in	*size*	セットするバイト数

戻り値

dst へのポインタ

3.21 include/h8sdk/unit_test.h ファイル

ユニットテストモジュール

マクロ定義

- #define UTEST_assert(message, test) do { if (!(test)) return message; } while (0)
  条件評価
- #define UTEST_run(test)
  テスト関数実行

型定義

- typedef char *(* UTEST_Func) ()
  テスト関数型

変数

- int UTEST_run_count

テスト回数パラメータ

3.21.1 詳解

ユニットテストモジュール

3.21.2 マクロ定義詳解

■#define UTEST_assert(*message*, *test*) do { if (!(test)) return message; } while (0) 条件評価

指定された条件を評価し、偽ならば指定文字列を返して呼び出し元関数を 抜ける。

引数

in	*message*	評価が偽のときのメッセージ
in	*test*	評価式

unit_test.h の 46 行目に定義があります。

■#define UTEST_run(*test*) **値:**

```
do {                                        \
        char *message = (test)();               \
        UTEST_run_count++;                      \
        if (message) return message;            \
    } while (0)
```

テスト関数実行

指定されたテスト関数を実行し、失敗した場合はメッセージを返して呼び 出し元関数を抜ける。

引数

in	*test*	実行するテスト関数へのポインタ

unit_test.h の 57 行目に定義があります。

3.21.3 型定義詳解

■typedef char*(* UTEST_Func) () テスト関数型

UTEST_run に与えるテスト関数の型

unit_test.h の 27 行目に定義があります。

3.21.4 変数詳解

■int UTEST_run_count テスト回数パラメータ

UTEST_run を実行する度にカウントアップする。テストモジュール本体で実 装しておく。

索引