1. システムインタフェースレイヤAPI仕様

1.1. システムインタフェースレイヤの概要

システムインタフェースレイヤ（この章では，SILと略記する）は，デバイスを直接操作するプログラムが用いるための機能である．ITRONデバイスドライバ設計ガイドラインの一部分として検討されたものをベースに，TOPPERSプロジェクトにおいて修正を加えて用いている．

SILの機能は，プロセッサの特権モードで実行されているプログラムが使用することを想定している【NGKI0801】．非特権モードで実行されているプログラムからSILの機能を呼び出した場合の動作は，次の例外を除いては保証されない【NGKI0802】．

- 微少時間待ちの機能を呼び出すこと
- エンディアンの取得のためのマクロを参照すること
- メモリ空間アクセス関数により，アクセスを許可されたメモリ領域にアクセスすること
- I/O空間アクセス関数により，アクセスを許可されたI/O領域にアクセスすること

1.2. SILヘッダファイル

SILを用いるために必要な定義は，SILヘッダファイル（sil.h）およびそこからインクルードされるファイルに含まれている【NGKI0803】．SILを用いる場合には，SILヘッダファイルをインクルードする【NGKI0804】．

1.3. 全割込みロック状態の制御

デバイスを扱うプログラムの中では，すべての割込み（NMIを除く，以下同じ）をマスクしたい場合がある．カーネルで制御できるCPUロック状態は，カーネル管理外の割込み（NMI以外にカーネル管理外の割込みがあるかはターゲット定義）をマスクしないため，このような場合に用いることはできない．

そこで，SILでは，すべての割込みをマスクする全割込みロック状態を制御するための以下の機能を用意している．

(1) SIL_PRE_LOC

全割込みロック状態の制御に必要な変数を宣言するマクロ【NGKI0805】．通常は，型と変数名を並べたもので，最後に";"を含まない．

このマクロは，SIL_LOC_INT，SIL_UNL_INTを用いる関数またはブロックの先頭の変数宣言部に記述しなければならない【NGKI0806】．SIL_LOC_INT，SIL_UNL_INTを1つの関数内でネストして用いることは可能であるが，その場合には，ネストレベル毎にブロックを作り，そのブロックの先頭の変数宣言部にSIL_PRE_LOCを記述しなければならない【NGKI0807】．そのように記述しなかっ

た場合の動作は保証されない【NGKI0808】.

(2) SIL_LOC_INT()

全割込みロックフラグをセットすることで，NMIを除くすべての割込みをマスクし，全割込みロック状態に遷移する【NGKI0809】.

(3) SIL_UNL_INT()

全割込みロックフラグを，対応するSIL_LOC_INTを実行する前の状態に戻す【NGKI0810】. SIL_LOC_INTを実行せずにSIL_UNL_INTを呼び出した場合の動作 は保証されない【NGKI0811】.

なお，全割込みロック状態で呼び出せるサービスコールなどの制限事項につい ては，「2.5.4 全割込みロック状態と全割込みロック解除状態」の節を参照す ること.

【補足説明】

全割込みロック状態の制御機能の使用例は次の通り.

```
{
    SIL_PRE_LOC;

    SIL_LOC_INT();
    // この間はNMIを除くすべての割込みがマスクされる.
    // この間にサービスコールを呼び出してはならない（一部例外あり）.
    SIL_UNL_INT();
}
```

1.4. SILスピンロック

マルチプロセッサシステムにおいて，カーネルの機能を用いずに，他のプロセッサとの間でも排他制御を実現したい場合がある．そこでSILでは，割込みのマスクとプロセッサ間ロックの取得により排他制御を行うためのスピンロックの機能を用意している．これを，カーネルのスピンロック機能と区別するために， SILスピンロックと呼ぶ.

プロセッサ間ロックを取得している間は，全割込みロック状態にすることです べての割込み（NMIを除く）がマスクされる【NGKI0812】．ロックが他のプロセッサに取得されている場合には，ロックが取得できるまでループによって待つ【NGKI0813】．ロックの取得を待つ間は，割込みはマスクされない（ロックの取得を試みる前にマスクしていた割込みは，マスク解除されない）【NGKI0814】．プロセッサ間ロックを取得し割込みをマスクすることを，SILスピンロックを取得するという．また，プロセッサ間ロックを返却し割込みをマ スク解除することを，SILスピンロックを返却するという.

SILで取得・返却するプロセッサ間ロックは，システムに唯一存在する 【NGKI0815】.

(1) SIL_PRE_LOC

全割込みロック状態の制御に必要な変数を宣言するマクロであるが，SILスピンロックの取得・解放にも兼用する【NGKI0816】.

このマクロは，SIL_LOC_SPN，SIL_UNL_SPNを用いる関数またはブロックの先頭の変数宣言部に記述しなければならない【NGKI0817】. SIL_LOC_SPN，SIL_UNL_SPNを，同じ関数内のSIL_LOC_INT，SIL_UNL_INTとネストして用いることは可能であるが，その場合には，ネストレベル毎にブロックを作り，そのブロックの先頭の変数宣言部にSIL_PRE_LOCを記述しなければならない【NGKI0818】. そのように記述しなかった場合の動作は保証されない【NGKI0819】.

(2) SIL_LOC_SPN()

SILスピンロックが取得されていない状態である場合には，プロセッサ間ロックの取得を試みる【NGKI0820】. ロックが他のプロセッサに取得されている状態である場合や，他のプロセッサがロックの取得に成功した場合には，ロックが返却されるまでループによって待ち，返却されたらロックの取得を試みる【NGKI0821】. ロックの取得に成功した場合には，全割込みロックフラグをセットし，全割込みロック状態に遷移する【NGKI0822】.

(3) SIL_UNL_SPN()

プロセッサ間ロックを返却し，全割込みロックフラグを対応するSIL_LOC_SPNを実行する前の状態に戻す【NGKI0823】.

SILスピンロックを取得している状態でSIL_LOC_SPNを呼び出した場合の動作は保証されない【NGKI0824】. 逆に，SILスピンロックを取得していない状態でSIL_UNL_SPNを呼び出した場合の動作も保証されない【NGKI0825】.

なお，SILスピンロック取得中は全割込みロック状態となっているため，SILスピンロック取得中に呼び出せるサービスコールなどについては，「2.5.4 全割込みロック状態と全割込みロック解除状態」の節の制限事項が適用される.

なお，マルチプロセッサシステム以外では，SIL_LOC_SPNとSIL_UNL_SPNは用意されていない【NGKI0826】.

【使用上の注意】

全割込ロック状態やCPUロック状態でSIL_LOC_SPNを呼び出すことはできるが，割込みがマスクされている時間が長くなるために，そのような使い方は避ける べきである.

【補足説明】

SILスピンロック機能の使用例は次の通り.

```
{
    SIL_PRE_LOC;

    SIL_LOC_SPN();
    // この間はSILスピンロックを取得している．
    // この間はNMIを除くすべての割込みがマスクされる．
    // この間にサービスコールを呼び出してはならない（一部例外あり）．
    SIL_UNL_SPN();
}
```

1.5. 微少時間待ち

デバイスをアクセスする際に，微少な時間待ちを入れなければならない場合がある．そのような場合に，NOP命令をいくつか入れるなどの方法で対応すると，ポータビリティを損なうことになる．そこで，SILでは，微少な時間待ちを行うための以下の機能を用意している．

(1) void sil_dly_nse(ulong_t dlytim)

dlytimで指定された以上の時間（単位はナノ秒），ループなどによって待つ【NGKI0827】．指定した値によっては，指定した時間よりもかなり長く待つ場 合があるので注意すること．

1.6. エンディアンの取得

プロセッサのバイトエンディアンを取得するためのマクロとして，SILでは，以 下のマクロを定義している．

(1) SIL_ENDIAN_BIG，SIL_ENDIAN_LITTLE

ビッグエンディアンプロセッサではSIL_ENDIAN_BIGを，リトルエンディアンプロセッサではSIL_ENDIAL_LITTLEを，マクロ定義している【NGKI0828】．

1.7. メモリ空間アクセス関数

メモリ空間にマッピングされたデバイスレジスタや，デバイスとの共有メモリをアクセスするために，SILでは，以下の関数を用意している．

(1) uint8_t sil_reb_mem(const uint8_t *mem)

memで指定されるアドレスから8ビット単位で読み出した値を返す【NGKI0829】．

(2) void sil_wrb_mem(uint8_t *mem, uint8_t data)

memで指定されるアドレスにdataで指定される値を8ビット単位で書き込む【NGKI0830】．

(3) uint16_t sil_reh_mem(const uint16_t *mem)

memで指定されるアドレスから16ビット単位で読み出した値を返す【NGKI0831】.

(4) void sil_wrh_mem(uint16_t *mem, uint16_t data)

memで指定されるアドレスにdataで指定される値を16ビット単位で書き込む 【NGKI0832】.

(5) uint16_t sil_reh_lem(const uint16_t *mem)

memで指定されるアドレスから16ビット単位でリトルエンディアンで読み出した 値を返す【NGKI0833】. リトルエンディアンプロセッサでは，sil_reh_memと一致する. ビッグエンディアンプロセッサでは，sil_reh_memが返す値を，エンディ アン変換した値を返す.

(6) void sil_wrh_lem(uint16_t *mem, uint16_t data)

memで指定されるアドレスにdataで指定される値を16ビット単位でリトルエンディアンで書き込む【NGKI0834】. リトルエンディアンプロセッサでは，sil_wrh_memと一致する. ビッグエンディアンプロセッサでは，dataをエンディアン変換した値を，sil_wrh_memで書き込むのと同じ結果となる.

(7) uint16_t sil_reh_bem(const uint16_t *mem)

memで指定されるアドレスから16ビット単位でビッグエンディアンで読み出した 値を返す【NGKI0835】. ビッグエンディアンプロセッサでは，sil_reh_memと一致する. リトルエンディアンプロセッサでは，sil_reh_memが返す値を，エンディ アン変換した値を返す.

(8) void sil_wrh_bem(uint16_t *mem, uint16_t data)

memで指定されるアドレスにdataで指定される値を16ビット単位でビッグエンディアンで書き込む【NGKI0836】. ビッグエンディアンプロセッサでは，sil_wrh_memと一致する. リトルエンディアンプロセッサでは，dataをエンディアン変換した値を，sil_wrh_memで書き込むのと同じ結果となる.

(9) uint32_t sil_rew_mem(const uint32_t *mem)

memで指定されるアドレスから32ビット単位で読み出した値を返す【NGKI0837】.

(10) void sil_wrw_mem(uint32_t *mem, uint32_t data)

memで指定されるアドレスにdataで指定される値を32ビット単位で書き込む 【NGKI0838】.

(11) uint32_t sil_rew_lem(const uint32_t *mem)

memで指定されるアドレスから32ビット単位でリトルエンディアンで読み出した 値を返す【NGKI0839】. リトルエンディアンプロセッサでは，sil_rew_memと一致する. ビッグエンディアンプロセッサでは，sil_rew_memが返す値を，エンディ アン変換した値を返す.

(12) void sil_wrw_lem(uint32_t *mem, uint32_t data)

memで指定されるアドレスにdataで指定される値を32ビット単位でリトルエンディ

アンで書き込む【NGKI0840】．リトルエンディアンプロセッサでは，
sil_wrw_memと一致する．ビッグエンディアンプロセッサでは，dataをエンディ
アン変換した値を，sil_wrw_memで書き込むのと同じ結果となる．

(13) uint32_t sil_rew_bem(const uint32_t *mem)

memで指定されるアドレスから32ビット単位でビッグエンディアンで読み出した　　値を返す【
NGKI0841】．ビッグエンディアンプロセッサでは，sil_rew_memと一
致する．リトルエンディアンプロセッサでは，sil_rew_memが返す値を，エンディ アン変換した値を返す．

(14) void sil_wrw_bem(uint32_t *mem, uint32_t data)

memで指定されるアドレスにdataで指定される値を32ビット単位でビッグエンディ
アンで書き込む【NGKI0842】．ビッグエンディアンプロセッサでは，
sil_wrw_memと一致する．リトルエンディアンプロセッサでは，dataをエンディ
アン変換した値を，sil_wrw_memで書き込むのと同じ結果となる．

1.8. I/O空間アクセス関数

メモリ空間とは別にI/O空間を持つプロセッサでは，I/O空間にあるデバイスレ
ジスタをアクセスするために，メモリ空間アクセス関数と同等の以下の関数を 用意している【NGKI0843】．

(1) uint8_t sil_reb_iop(const uint8_t *iop)
(2) void sil_wrb_iop(uint8_t *iop, uint8_t data)
(3) uint16_t sil_reh_iop(const uint16_t *iop)
(4) void sil_wrh_iop(uint16_t *iop, uint16_t data)
(5) uint16_t sil_reh_lep(const uint16_t *iop)
(6) void sil_wrh_lep(uint16_t *iop, uint16_t data)
(7) uint16_t sil_reh_bep(const uint16_t *iop)
(8) void sil_wrh_bep(uint16_t *iop, uint16_t data)
(9) uint32_t sil_rew_iop(const uint32_t *iop)
(10) void sil_wrw_iop(uint32_t *iop, uint32_t data)
(11) uint32_t sil_rew_lep(const uint32_t *iop)
(12) void sil_wrw_lep(uint32_t *iop, uint32_t data)
(13) uint32_t sil_rew_bep(const uint32_t *iop)
(14) void sil_wrw_bep(uint32_t *iop, uint32_t data)

1.9. プロセッサIDの参照

マルチプロセッサシステムにおいては，プログラムがどのプロセッサで実行さ
れているかを参照するために，以下の関数を用意している．

(1) void sil_get_pid(ID *p_prcid)

この関数を呼び出したプログラムを実行しているプロセッサのID番号を参照し，
p_prcidで指定したメモリ領域に返す【NGKI0844】．

【使用上の注意】

タスクは，sil_get_pidを用いて，自タスクを実行しているプロセッサを正しく参照できるとは限らない．これは，sil_get_pidを呼び出し，自タスクを実行しているプロセッサのID番号を参照した直後に割込みが発生した場合，sil_get_pidから戻ってきた時には自タスクを実行しているプロセッサが変化している可能性があるためである．