～目次～

[構成ディレクトリの説明]

- arch/cpu：CPU 依存部(ARM-Cortex-A8)
- arch/gcc：コンパイラ依存部(gcc4.x)
- c_lib　：簡単なＣ標準ライブラリ関数類(まだライブラリにはしていません)
- doc　　：ドキュメント類
- kernel　：kernel の機能(ターゲット非依存部)
- net　　：転送プロトコル(ログの送信に使用しています)
- target　：borad 依存部(beagle-borad-xM)
- tsk_lib　：簡単なサンプルタスク類(まだライブラリにはしていません)

[各ファイルの説明]

- arch/cpu/cpu_cntrl.h　：CPSR ののマクロを定義
- arch/cpu/intr.h　：IRQ 周り
- arch/cpu/intr_cntrl.c　：MIR 操作
- arch/cpu/intr_cntrl.h　：MIR 定義
- arch/cpu/intr_hadle.c　：割り込みハンドラ
- arch/cpu/intr_hadle.h　：割り込みハンドラインターフェース
- arch/cpu/main.h　：kernel main
- arch/cpu/startup.S　：startup

- arch/gcc/_divsi3.S　：乗算、除算関連
- arch/gcc/_udivsi3.S　：乗算、除算関連
- arch/gcc/div0.c　：乗算、除算関連

- c_lib/lib.c　：簡易Ｃ標準ライブラリ関数
- c_lib/lib.h　：簡易Ｃ標準ライブラリ関数のインターフェース

○ kernel/command.c ：OS を操作する基本コマンド
○ kernel/command.h ：OS を操作する基本コマンドインターフェース
○ kernel/defines.h ：型、エラーコードを定義
○ kernel/init_tsk.c ：kernel の init 後、起動するタスク
○ kernel/intr_manage.c ：割込み管理機能
○ kernel/intr_manage.h ：割込み管理機能インターフェース
○ kernel/kernel.c ：kernel の init、カーネルコア、カーネルコアメカニズム
○ kernel/kernel.h ：kernel.c のインターフェース
○ kernel/kernel.scr ：リンカ設定(linker script)、メモリマップ
○ kernel/memory.c ：heap 管理(動的メモリ)
○ kernel/memory.h ：heap 管理(動的メモリ)インターフェース
○ kernel/multi_timer.c ：タイママルチ管理
○ kernel/muleti_timer.h ：タイママルチ管理インターフェース
○ kernel/ready.c ：レディー状態管理
○ kernel/ready.h ：レディー状態管理インターフェース
○ kernel/scheduler.c ：スケジューリングポリシー
○ kernel/scheduler.h ：スケジューリングポリシーのインターフェース
○ kernel/syscall.c ：システムコール管理
○ kernel/syscall.h ：システムコール管理インターフェース
○ kernel/task.h ：タスクメカニズム、タスク周り操作マクロ定義
○ kernel/task_manage.c ：タスク管理
○ kernel/task_manage.h ：タスク管理インターフェース
○ kernel/task_sync.c ：タスク付属同期
○ kernel/task_sync.h ：タスク付属同期インターフェース

○ kernel_svc/log_manage.c　：ロギング
○ kernel_svc/log_manage.h　：ログ管理インターフェース

○ net/jis_ctrl_crd.h　：制御コード定義(JIS X 0211~C0 集合)
○ net/xmodem.c　：xmodem 転送プロトコル(送信)
○ net/xmodem.h　：xmodem 転送プロトコル(送信)インターフェース

○ target/driver/serial_driver.c　：UART ドライバ
○ target/driver/serial_driver.h　：UART ドライバのインターフェース
○ target/driver/timer_driver.c　：タイマドライバ
○ target/driver/timer_driver.h　：タイマドライバインターフェース

○ tsk_lib/tsk_set1　：サンプルタスクセット
○ tsk_lib/tsk_set2　：サンプルタスクセット
○ tsk_lib/tsk_set3　：サンプルタスクセット

[使用ツール]
使用しているツールを列挙します.
○ CodeSourcery
主に、以下を使用しています.
● as(アセンブリ)
● gcc4.5.2(コンパイルチェーン)
● ld(リンクユーティリティ)
以下があると便利です.
● ar(アーカイバ)
● nm(シンボル関連)
● objcopy(実効フォーマット変換)
● objdump(実効フォーマット操作)
● strip(シンボルテーブル最適化)
○ git(バージョン管理)
○ make(ソース，ビルド管理)
○ doxygen(コードドキュメント管理)
○ graphviz(コード可視化)
○ uboot(ブートローダ)
● mkimage(OS を圧縮して展開、実効する場合)
○ minicom や terateam 等のエミュレータ(動作確認用)

[OS ビルド関連]
　make を使用しています。具体的には、複数のディレクトリに分散したファイルを一度トップディレクトリにかき集める(include directive)を使用し、依存関係を解析し、各モジュールを生成し、一度 objs ディレクトリに集める。そして、モジュールのリンク順序を求め、kernel.scr の設定にしたがい、bin ディレクトリに実効フォーマットを生成します。
　OS のソースファイルの位置に従って、Makefile のパスを変更して下さい。また、uboot を任意の位置に置いて下さい(Makefile 参照)
○ OS のビルドと実効フォーマット(通常)
>% make　　//OS ビルド
>% make bin //OS 実効フォーマットを uboot 形式に変換

○ OS のビルドと実効フォーマット(OS 圧縮イメージ)
>% make　　// OS ビルド
>% make image　　// OS を圧縮(uboot の mkimage を使用する)

○ クリーン
>% make clean

[OS 実効関連]
　uboot のマニュアル参照して下さい．[使用ツール]で挙げたエミュレータ上で操作します．
(例 1) OS を転送プロトコルで転送
　ymodem 等の modem プロトコル，TFTP ブート等が使用できます．転送プロトコルによっては、拡張ユーティリティが必要になります．
(例 2) OS を実機の SD カードに置いてブート
　本 OS は、beagle-borad-xM を使用しています．このボードは、NAND が無く、SD カードがあります(注: beagle-borad だと NAND あり)
つまり、ROM 化はいりません。そのため、ライターも必要ありません。

[OS 起動]
　OS 起動後はコマンドで操作します.help コマンドで一覧を表示できます．

[ソースコードリーディング]
　ソースコードリーディングの際には,TAB 幅を 2 に設定して下さい．