NuttX + MuPDF で
電子書籍リーダーの設計と制作
をしようとしてミスったあれこれ

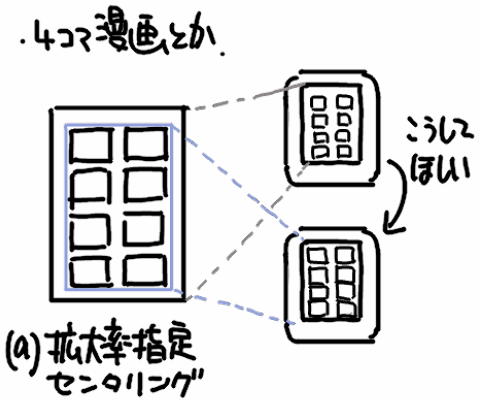
@kam1610
サークル 181 号

2023/02/19

背景

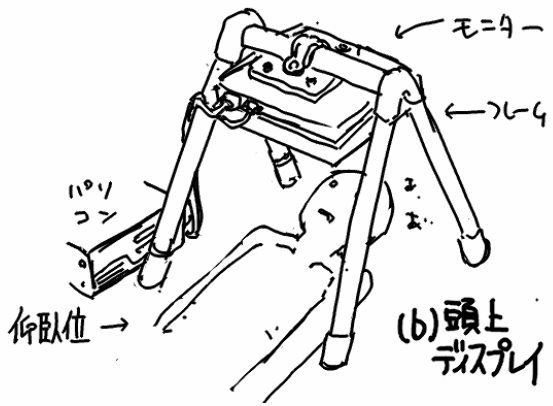
やりたいこと

- おそとで電子書籍よみたい
  イベントの待機列とか...[1]
- おふとんでよみたい
- 拡大率指定センタリング



候補

- EPD[2] : これが本命，みやすい．
- スマートフォン
- ノート PC
- 頭上ディスプレイ
- HUD



[1]なろう→ epub/mobi に変換するスクリプト (**narrow.rb**) をつくりました．すごい時間とける．このはなしもそのうち...

[2]Electronic paper display, いわゆる電子ペーパー

EPD の候補

- ベンダ系 (kindle,KOBO など):
  サービス固定がイヤ (脱獄もできそうだけど..)
- Android 系 (BOOX, MetaPad, など):
  BOOX は本物の (?)GooglePlay 使えるのがよさそう．あと A4 サイズあるのも良い．ただし高価 (¥129,800)．
- 自作系 (The Open Book など):

EPD の候補

- ベンダ系 (kindle,KOBO など):
  サービス固定がイヤ (脱獄もできそうだけど..)
- Android 系 (BOOX, MetaPad, など):
  BOOX は本物の (?)GooglePlay 使えるのがよさそう．あと A4 サイズあるのも良い．ただし高価 (¥129,800)．
- 自作系 (The Open Book など):
  口ゴが怖すぎる．

EPD の候補

- ベンダ系 (kindle,KOBO など):
  サービス固定がイヤ (脱獄もできそうだけど..)
- Android 系 (BOOX, MetaPad, など):
  BOOX は本物の (?)GooglePlay 使えるのがよさそう．あと A4 サイズあるのも良い．ただし高価 (¥129,800)．
- 自作系 (The Open Book など):
  ロゴが怖すぎる．

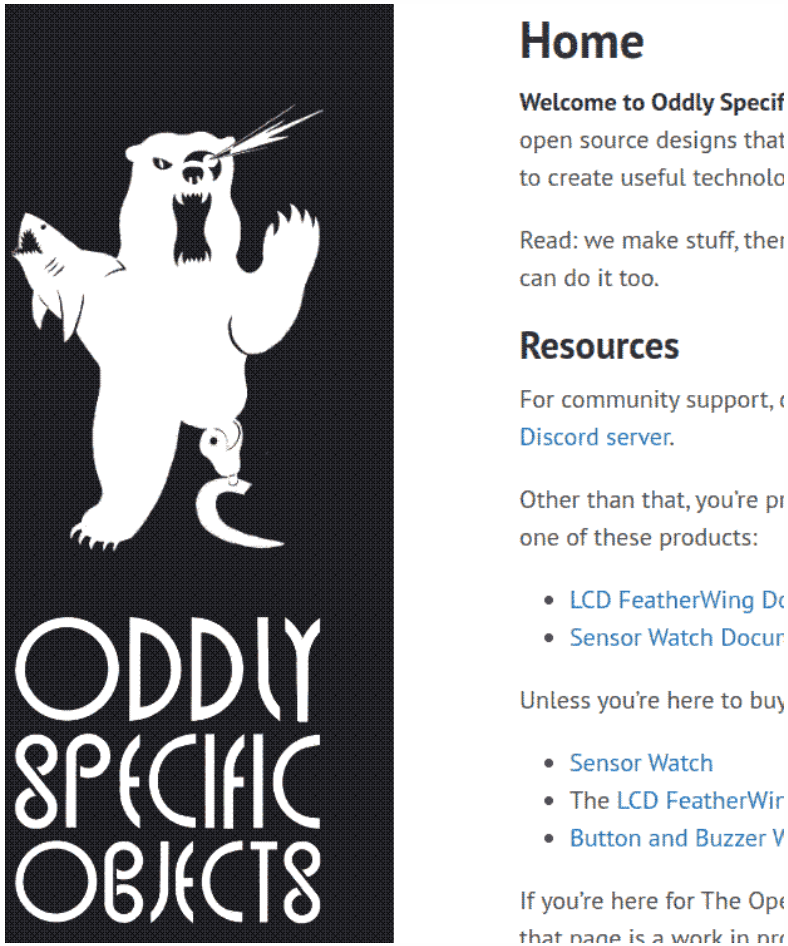
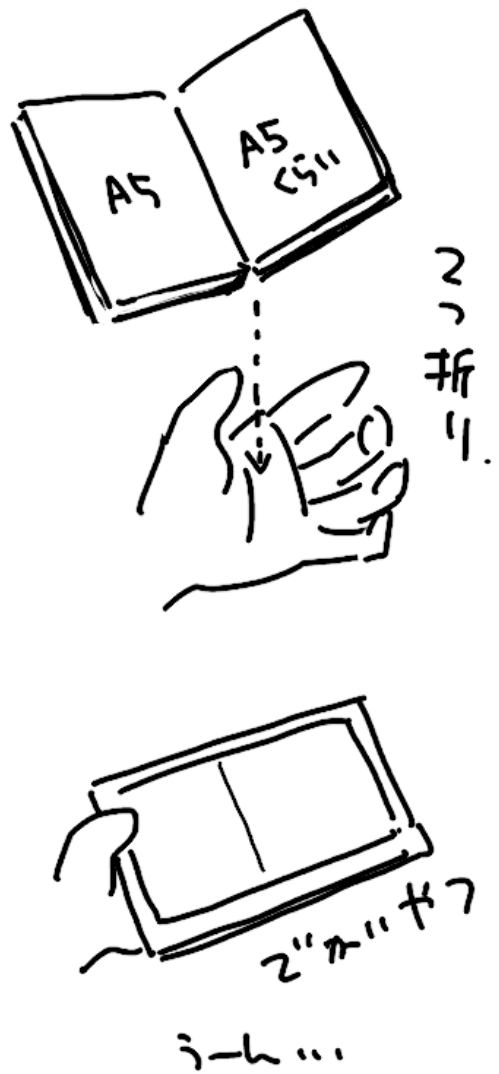


Figure 1: ロゴが怖い

要件は？

- 見開きが快適に読める
- 任意のフォーマットで読める
  EPD,PDF, 画像フォーマットなど
- 軽い (システムが)
  電子書籍リーダーに
  Android(Linux) はやりすぎな気
  がする..
- H/W スイッチ



結局

つくるか.

もちろん既製品もある

いわゆる，誰でも思いつきそうでナイやつです
“∑Book” とか．あと “Foldable E-Ink”，“Dual Screen Manga Reader” とか．
ただ，いまいち購入できそうなのがない...(需要/耐久性/コストなどなどでしょうか..)
あと，機能集約の観点だとタブレットかなとか思うのですが，EPD が良い,,,表が有機 EL，裏が EPD みたいなのがいいかも．
とりあえず，もう作りたくなっちゃったので気にしないことにします．

まずは材料の選定

アプリケーション	EPUB/PDFリーダー
ライブラリ	MuPDF, Ghostscript, Poppler(Xpdf), ...
OS	NuttX, Zephyr, VxWorks, LynxOS, QNX,...
ボード	結構悩んだので後述
チップ	
デバイス	WaveShare, PERVASIVE, Good Display

評価サイト[1]のベンチで最速，コンフィグしやすそう，フットプリント小さい2MByte(公称)

POSIX対応でフリーのRTOS

公式Wikiが充実してる，グレースケール対応でB6相当モジュールがある（他は小さいかモノクロ）
千石に為替調整前っぽい価格の在庫がある

[1] https://hub.alfresco.com/t5/alfresco-content-services-blog/pdf-rendering-engine-performance-and-fidelity-comparison/ba-p/287618

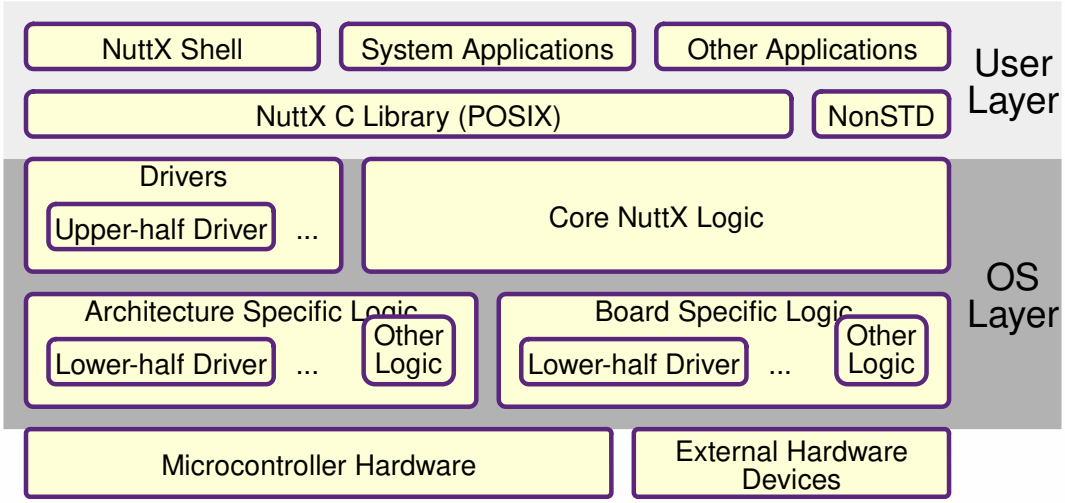
よりみち: NuttX について 1/4

沿革/特徴/採用例，アーキテクチャ，および簡単な導入手順 (1. カーネル，2. アプリ) を説明します．

- (沿革) 2007 初版リリース (Gregory Nutt 氏)，
  2019 年以降は Apache のインキュベーションプログラム
- (特徴)
  - (スケジュール) プリエンプティブ，固定優先度/ラウンドロビン，スポラディックスケジューリング
  - (API 体系) POSIX/ANSI 風の各種 API
  - (メモリ空間) フラット/MPU/MMU サポート
  - (モジュール) ローダブルモジュールサポート
  - (開発環境) GNU
- (採用事例) SONY の Spresense が採用したことで有名

よりみち: NuttX について 2/4

- アーキテクチャ



- ドライバが充実していて，階層スタイルが Linux とほぼ同じなので，Linux ドライバの経験があれば導入しやすそう．超語弊があるけど余計なものが動かない，Linux っぽいものとして使えそう (MuPDF はかなり簡単に移植できた)．
- 逆に，ITRON 系とかピュアな (?)RTOS からだととっつきにくいかも．

よりみち: NuttX について 3/4

簡単な導入手順 (カーネルビルド)

(1) ソース取得

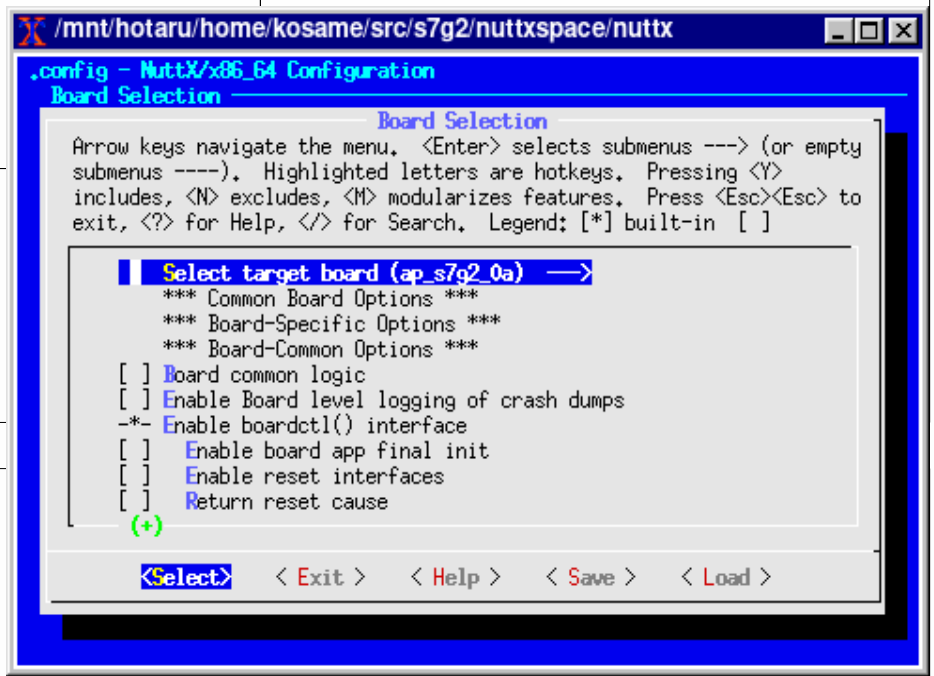
```
$ mkdir nuttxspace && cd nuttxspace
$ git clone https://${REPO}/incubator-nuttx.git nuttx
$ git clone https://${REPO}/incubator-nuttx-apps.git apps
$ tree -L 1 ../nuttxspace
../nuttxspace
|-- apps
`-- nuttx
```

(2) プリセットコンフィグ

```
$ cd nuttx
$ ./tools/configure.sh -L | less
$ ./tools/configure.sh -l <board>:<confdir>
  ( ./tools/configure.sh -l sim:nsh とか )
```

(3) カーネルコンフィグ

```
$ make menuconfig
```



(4) make && ./nuttx

```
kosame@hikaru /dev/shm/nuttxspace/nuttx
$ ./nuttx

NuttShell (NSH) NuttX-12.0.0
Hello NuttX!
nsh>
```

よりみち: NuttX について 4/4

簡単な導入手順 (アプリケーション)

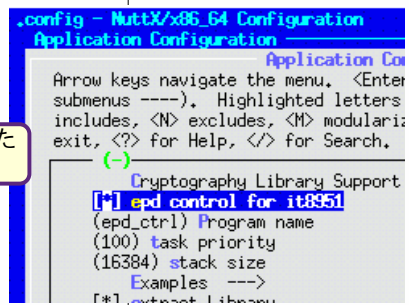
```
nuttxspace/
|-- apps
|    |-- epd_ctrl
|    |    |-- Kconfig
|    |    |-- Make.defs
|    |    |-- Makefile
|    |    `-- it8951.c
|    |-- other_apps
|
`-- nuttx
     `-- described_later
```

1. make menunconfig のエントリ

```
config EPD_CTRL_IT8951
       bool "epd control for it8951"
       default n
if EPD_CTRL_IT8951
config EPD_CTRL_IT8951_PROGNAME
       string "Program name"
       default "epd_ctrl"
config EPD_CTRL_IT8951_PRIORITY
       int "task priority"
       default 100
config EPD_CTRL_IT8951_STACKSIZE
       int "stack size"
       default DEFAULT_TASK_STACKSIZE
endif # EPD_CTRL_IT8951
```

コマンドにしたときの名前

タスク優先度



2. ビルド対象にしたいディレクトリと，その他フラグ設定

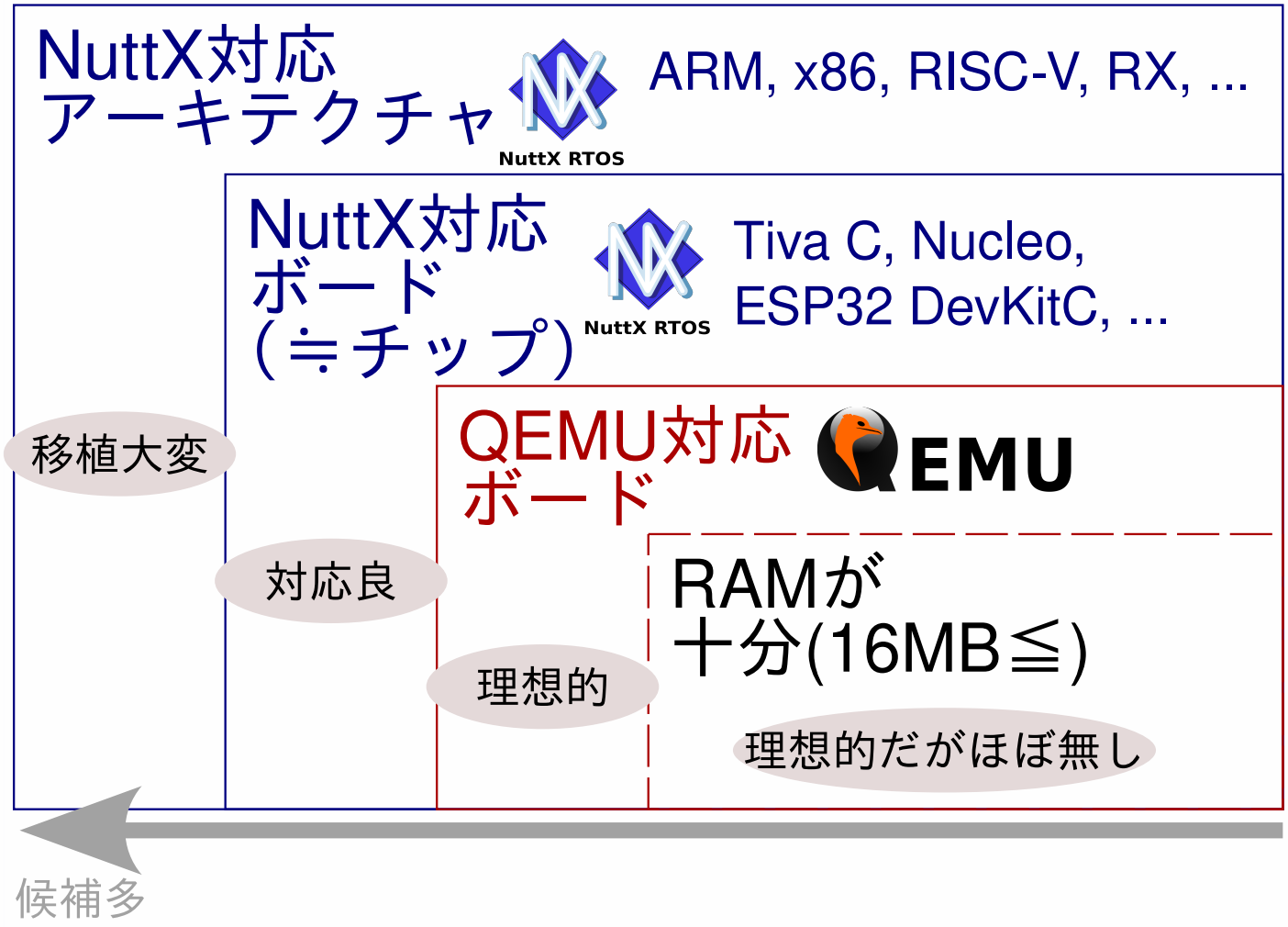
```
ifeq ($(CONFIG_EPD_CTRL_IT8951),y)
CONFIGURED_APPS += $(APPDIR)/epd_ctrl/
endif
CFLAGS += -I$(APPDIR)/epd_ctrl
CFLAGS += -DIT8951_Interface_SPI
```

3. ビルド対象ソースの定義

```
include $(APPDIR)/Make.defs
PROGNAME=$(CONFIG_EPD_CTRL_IT8951_PROGNAME)
PRIORITY=$(CONFIG_EPD_CTRL_IT8951_PRIORITY)
STACKSIZE=$(CONFIG_EPD_CTRL_IT8951_STACKSIZE)
MODULE=$(CONFIG_EPD_CTRL_IT8951)
CSRCS  = it8951.c
include $(APPDIR)/Application.mk
```

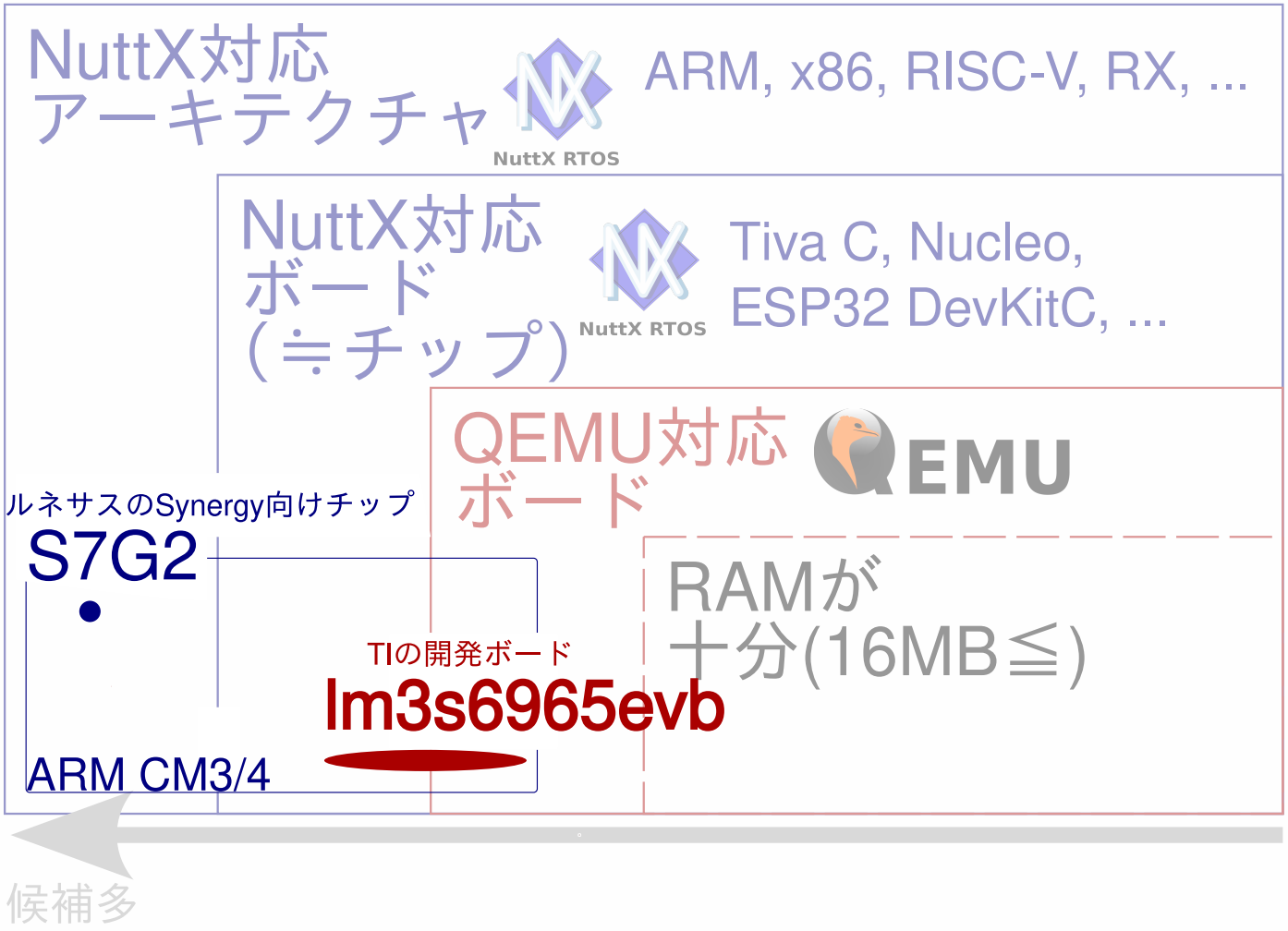
ビルド対象

ボード/チップの絞り込み



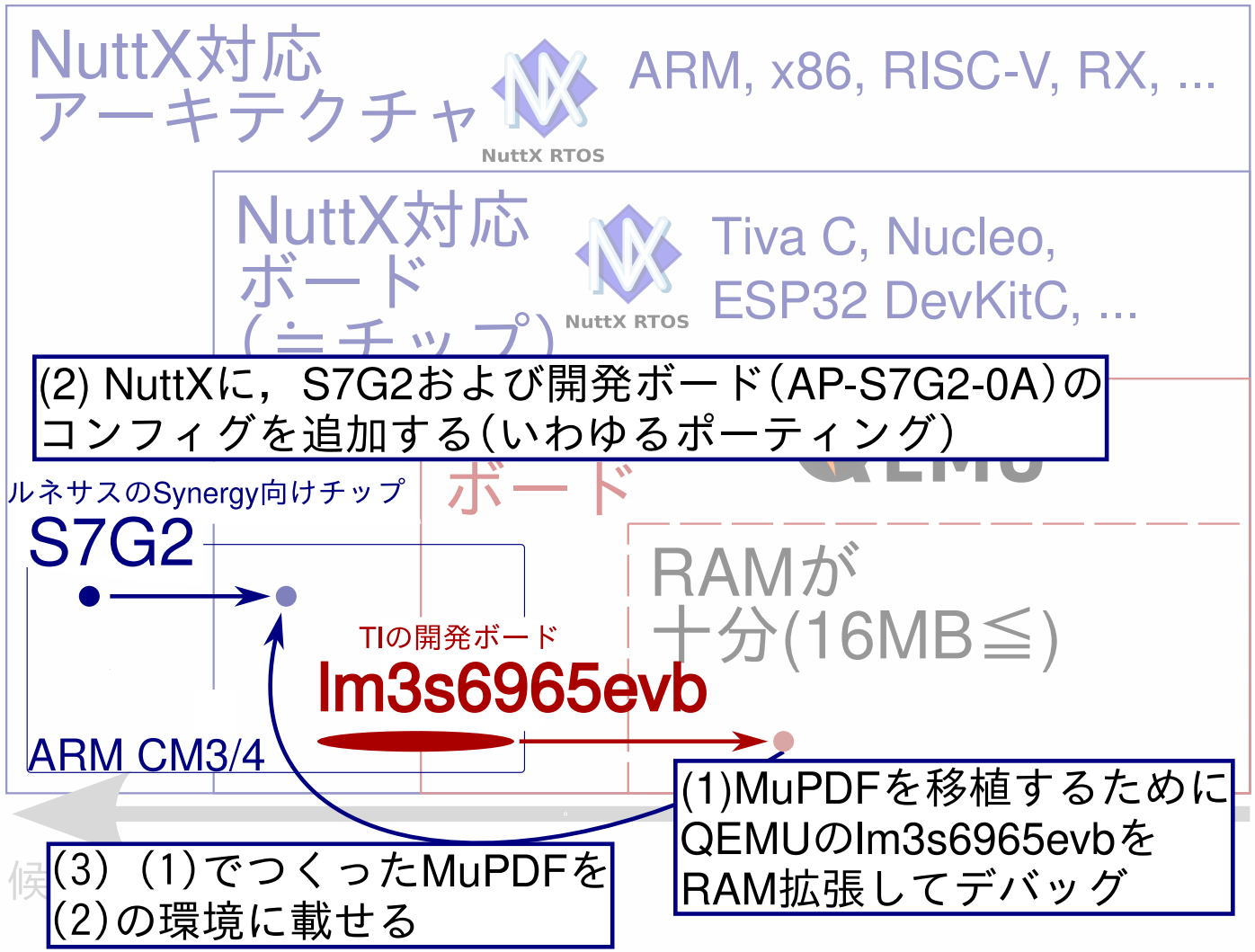
デバッグしやすそうな順で進めたい →QEMU ベースで

ボード/チップの絞り込み



丁度よいのがないので，入手性も考慮して
今回は S7G2 と lm3s6965 の組み合わせで開発

ボード/チップの絞り込み: こんな作戦で



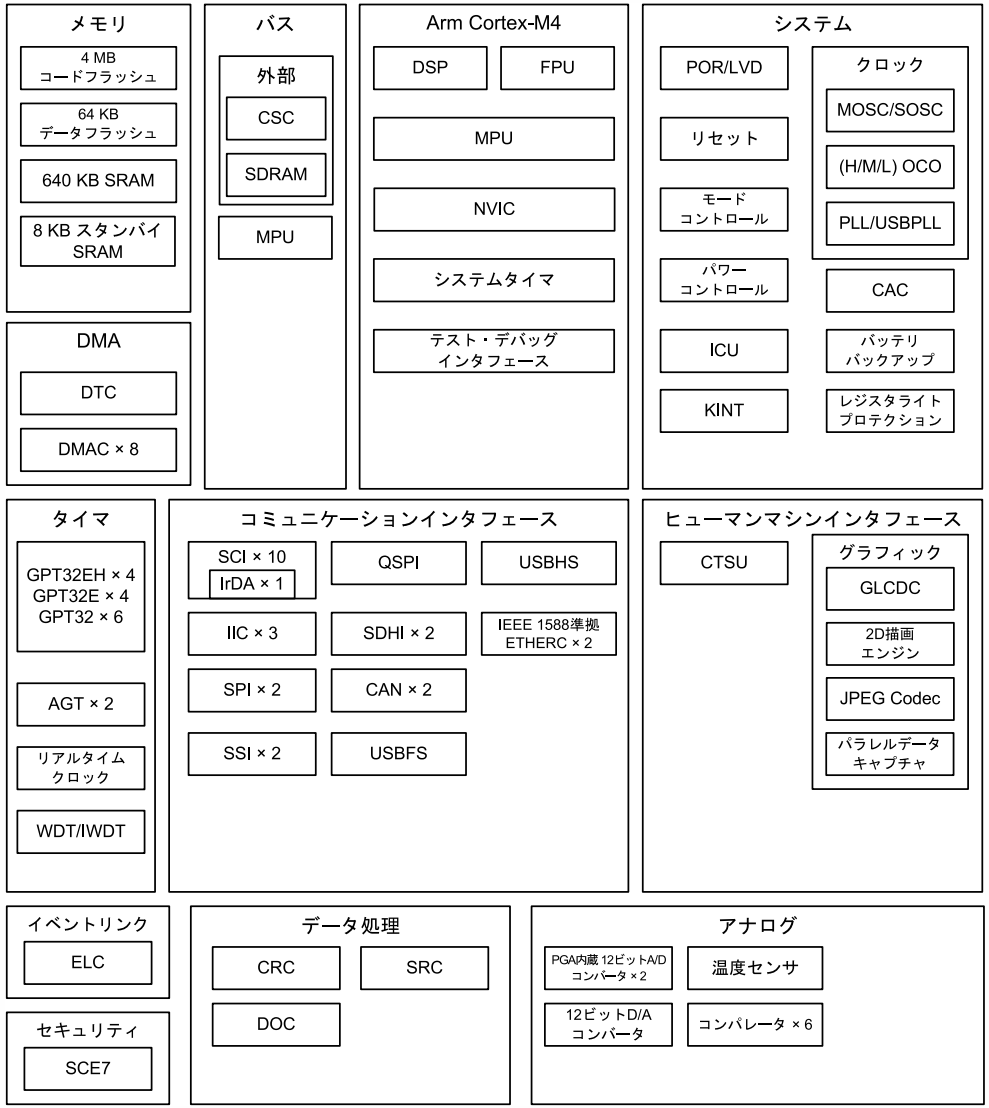
よりみち: S7G2 について

Renesas 社の Synergy[1] 用，ARM Cortex-M4 搭載 ASIC．

- 高価 (ほぼ最上位なので仕方ないけど 2000 円くらい)
- でかい (最大 26mm 角)
- 日本語版ドキュメントが参考資料扱い

ARM は RA シリーズがあるし，Synergy も自分のまわりだとあまり聞かなくてちょっと微妙な気がしますが，長期供給の対象で 2033/10 まで供給
→ 買える ARM！

アーキテクチャは，RX のコアを ARM におきかえたかんじ．



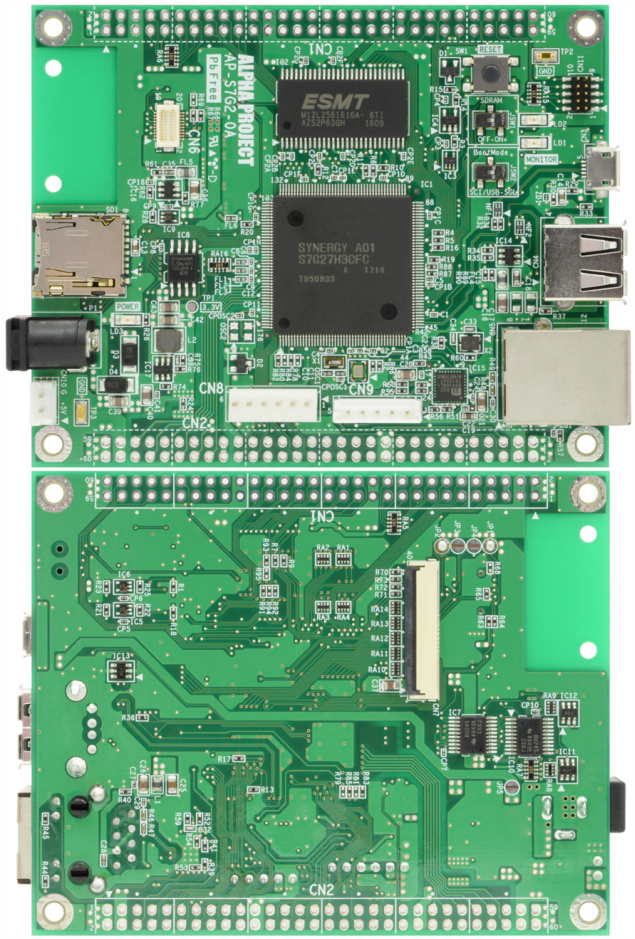
[1] Renesas 版の mbed みたいなやつ,,, という理解です．欧州がメインターゲット？

よりみち: S7G2 について

Alpha プロジェクト社の評価ボード (AP-S7G2-0A).
RAM がちょういっぱいある.

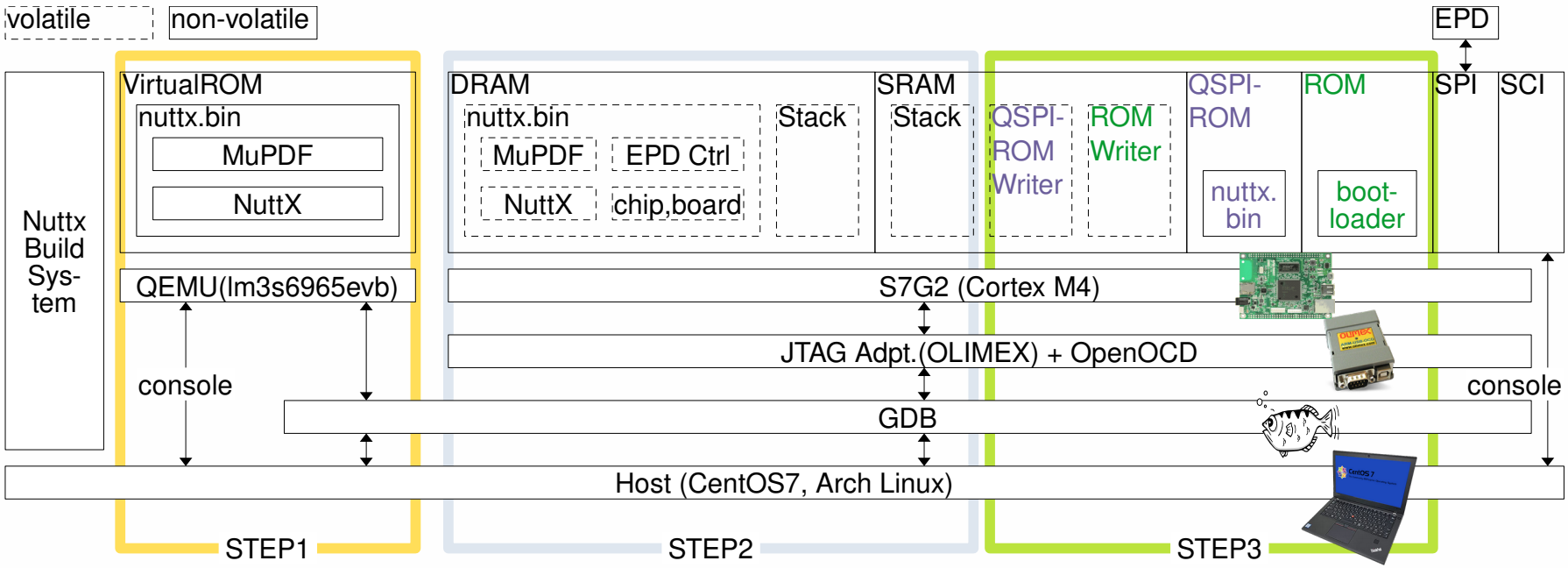
Core	CortexM4 240MHz
SRAM	640KB
ROM	4MB
DRAM	32MB
QSPI-ROM	16MB

素材も決まったので環境を整えます.



開発環境の構成

こんなかんじ．



作戦 (1)QEMU で MuPDF 移植，(2)DRAM でうごかず，(3) セルフブート
(4) 性能チューニング

QEMU/NuttX: MuPDF 移植

1. QEMU 調整 (ROM/RAM 増強)
2. MuPDF 機能選択
3. MuPDF 移植 (NuttX 向け調整)
4. ブート

QEMU(1/4): ROM/RAM 増強

まず，QEMU の仮想ボードだと ROM/RAM が不足しそうなので上限を引き上げます[1]

```
diff -u -U0 stellaris.c.orig stellaris.c
--- stellaris.c.orig
+++ stellaris.c
@@ -963,7 +963,8 @@
    { "LM3S6965EVB",
      0x10010002,
      0x1073402e,
-     0x00ff007f,
+     //0x00ff007f, /* dc0 */
+     0xffff3fff, /* dc0, ram 16M, flash 64M */
      0x001133ff,
```

上記で，`qemu-system-arm -M lm3s6965evb` したときに ROM/RAM: 64k/8k → 64M/16M になります．わぁい！[2]

[1] たぶん互換性提示用のレジスタ
[2] 妥当な方法ではないきがするけど...

QEMU(2/4): MuPDF 機能選択

MuPDF のサイズ削減するために，マクロを調整します．
mupdf/include/mupdf/fitz/config.h にて，
↓こんなかんじで [1]

```
/**
  Choose which plotters we need.
  By default we build all the plotters in.
  To avoid building plotters in that aren't
  needed, define the unwanted FZ_PLOTTERS_...
  define to 0.
*/
#define FZ_PLOTTERS_G     1
#define FZ_PLOTTERS_RGB   0 /* ←いらんトコを */
#define FZ_PLOTTERS_CMYK  0 /* けずっていく   */
#define FZ_PLOTTERS_N     0
```

[1] Makefile で-D とかも可

QEMU(3/4): MuPDF 移植 (NuttX 向け調整)

モトの Makefile の流用が (自分には) 難しかったので，「よりみち: NuttX について」のようにして自分で MuPDF 用の Makefile などをつくります．以下ポイントです．

- `CSRCS+=source/*.c`
- ただし，フォントディレクトリでは必要なものに厳選[1]．
- X11, GLUT, Tesseract は config.h でなく Makefile 側で無効化:`=no`

[1]それでも 20MB くらいあったので．結局 SourceHan のサブセットをつくっておきかえた．
[2]ライトオープン時に既存があるとエラーになるやつ

QEMU(3/4): MuPDF 移植 (NuttX 向け調整)

モトの Makefile の流用が (自分には) 難しかったので，「よりみち: NuttX について」のようにして自分で MuPDF 用の Makefile などをつくります．以下ポイントです．

- `CSRCS+=source/*.c`
- ただし，フォントディレクトリでは必要なものに厳選[1]．
- X11, GLUT, Tesseract は config.h でなく Makefile 側で無効化:`=no`

コードもちょっと修正

ハマり

fopen で'`x`'[2] オプションは非サポートなので外す

[1] それでも 20MB くらいあったので．結局 SourceHan のサブセットをつくっておきかえた．
[2] ライトオープン時に既存があるとエラーになるやつ

QEMU(3/4): MuPDF 移植 (NuttX 向け調整)

モトの Makefile の流用が (自分には) 難しかったので，「よりみち: NuttX について」のようにして自分で MuPDF 用の Makefile などをつくります．以下ポイントです．

- `CSRCS+=source/*.c`
- ただし，フォントディレクトリでは必要なものに厳選[1]．
- X11, GLUT, Tesseract は config.h でなく Makefile 側で無効化:`=no`

コードもちょっと修正

ハマり

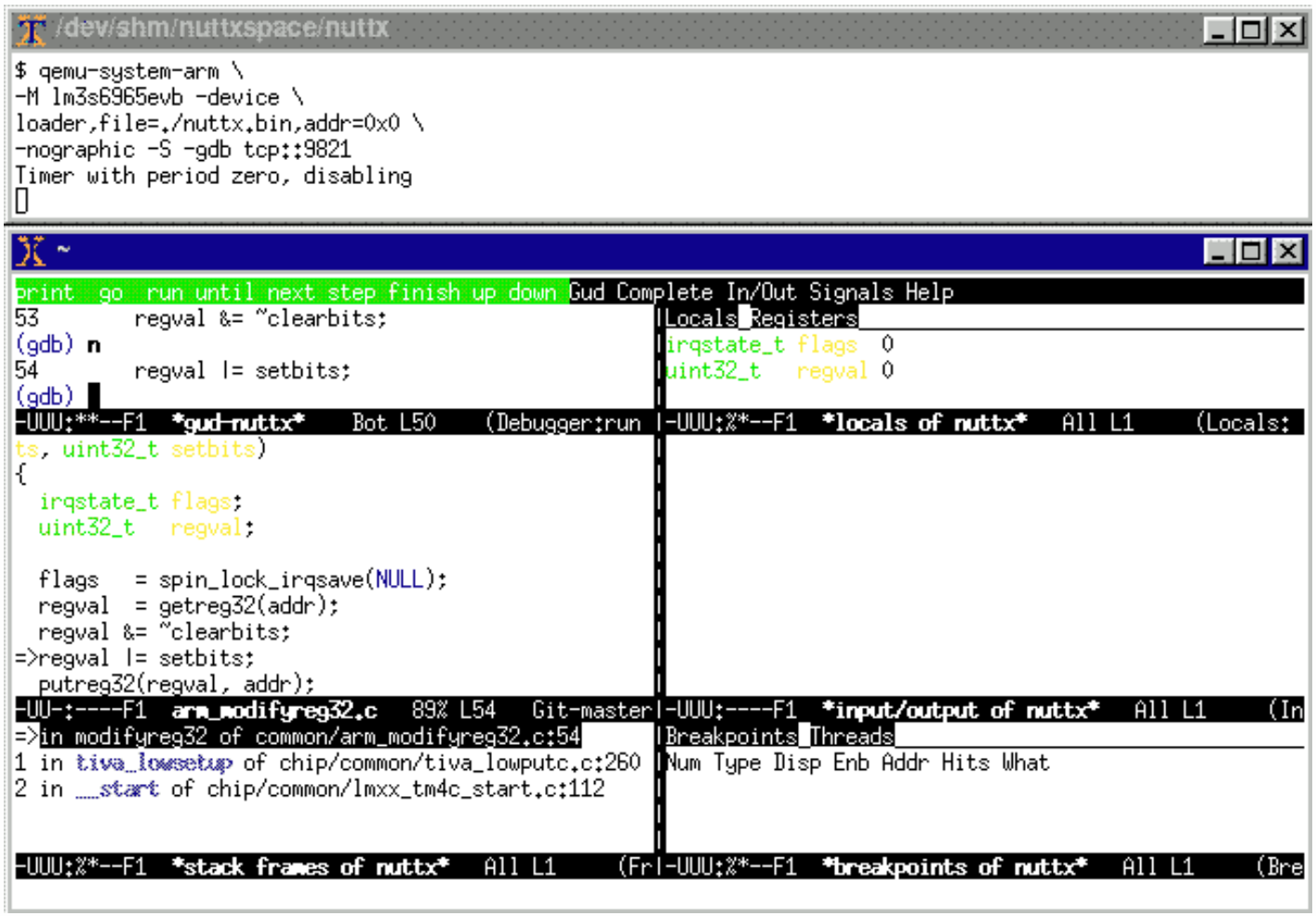
fopen で'x'[2] オプションは非サポートなので外す

これでだいたい，nuttx.bin が 30M → 11MB．ROM に入る！

[1] それでも 20MB くらいあったので．結局 SourceHan のサブセットをつくっておきかえた．
[2] ライトオープン時に既存があるとエラーになるやつ

QEMU(4/4): ブート

- `-device` でバイナリロード
- `-gdb` でデバッガ接続



QEMU: いったんまとめ

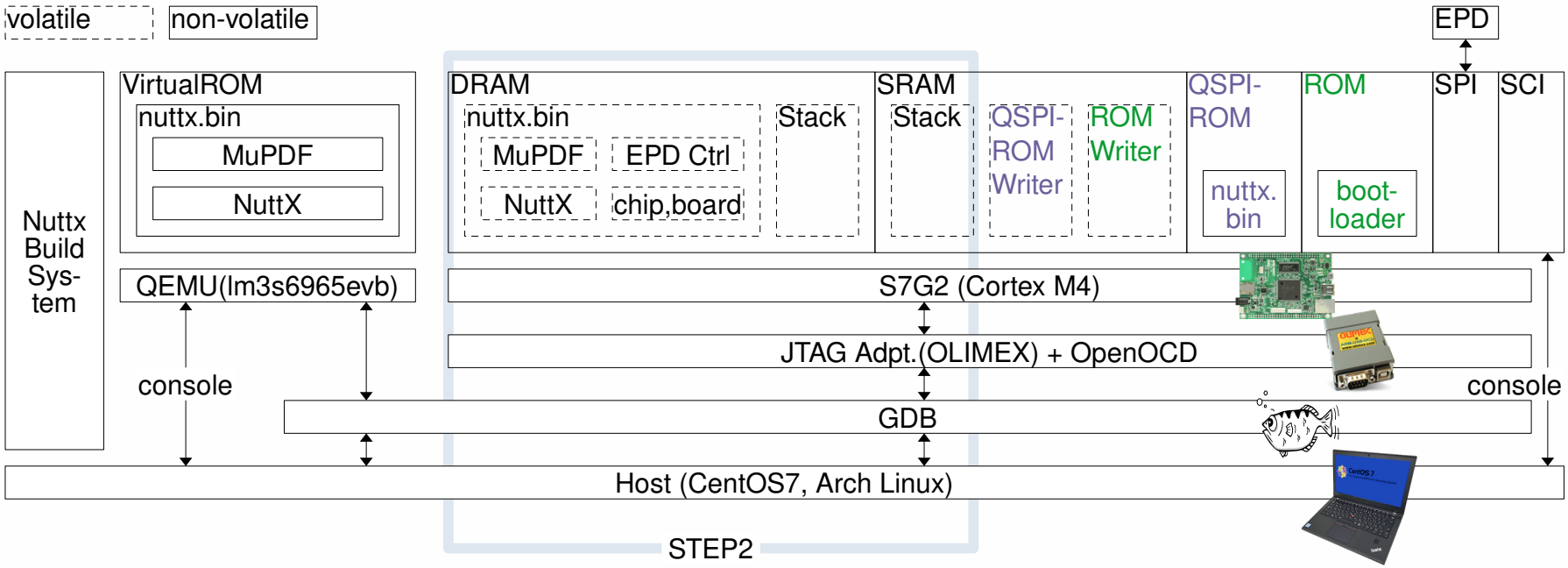
- ✓ QEMU 調整 (ROM/RAM 増強)
  `stellaris.c` のコードを調整して RAM16MB/ROM64MB に増強
- ✓ MuPDF 機能選択
  不要な機能を無効化
- ✓ MuPDF 移植 (NuttX 向け調整)
  フォント差し替えで 30MB → 11MB + 環境依存コードの調整
- ✓ ブート
  `-device` でバイナリロード

QEMU: いったんまとめ

- ✓ QEMU 調整 (ROM/RAM 増強)
  `stellaris.c` のコードを調整して RAM16MB/ROM64MB に増強
- ✓ MuPDF 機能選択
  不要な機能を無効化
- ✓ MuPDF 移植 (NuttX 向け調整)
  フォント差し替えで 30MB → 11MB + 環境依存コードの調整
- ✓ ブート
  `-device` でバイナリロード

次はいよいよ実機 (S7G2) に
NuttX をポーティングします.

S7G2 ポーティング


volatile
non-volatile
EPD
Nuttx Build Sys-tem
VirtualROM
nuttx.bin
MuPDF
NuttX
DRAM
nuttx.bin
MuPDF
EPD Ctrl
NuttX
chip,board
Stack
SRAM
Stack
QSPI-ROM Writer
ROM Writer
QSPI-ROM
nuttx. bin
ROM
boot-loader
SPI
SCI
QEMU(lm3s6965evb)
S7G2 (Cortex M4)
JTAG Adpt.(OLIMEX) + OpenOCD
console
GDB
console
Host (CentOS7, Arch Linux)
STEP2

S7G2 ポーティング

S7G2 向けの NuttX ポーティングでは，以下のふたつをつくります

- チップ (S7G2) 固有の設定
  各種デバイスドライバ (UART, GPIO, ...)，割込み関連，システムタイマ，など
- ボード (AP-S7G2-0A) 固有の設定
  チップを搭載したボード固有のコード．主にピンコンフィグなど．
  リンカスクリプトもここ．

S7G2 ポーティング (1/2): チップ

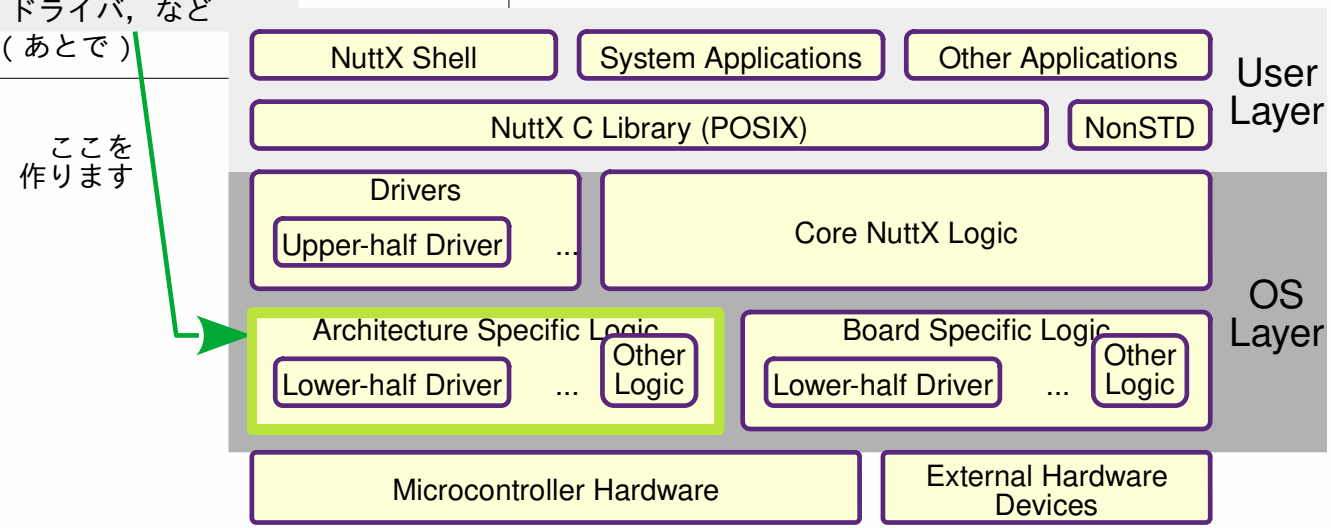
まずはチップ用のコンフィグをつくります．

```
nuttx/
|- arch/arm/
|   |- Kconfig -- ARCH_CHIP_S7G2 の config/source 設定
|   | 1          (ARCH_HAVE_MPU などアーキテクチャの属性と
|   |             置き場 (arch/arm/src/s7g2/) を定義 )
|   `- s7g2/ 2
|      |- Kconfig -- ARCH_CHIP_R7FS7G27H3A01CFC など
|      |
|      |- Meke.defs --  ビルド対象にする以下の C ファイルを
|      |    3           登録 , CHIP_CSRCS += s7g2_*.c
|      |
|      `- s7g2_*.c --  割込み，システムタイマ，
|       4              UART ドライバ，など
`- boards/ --  ボードの設定 ( あとで )
```

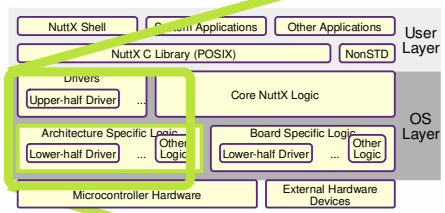
製品シリーズ，
RX65N とか STM32F7，くらいの粒度

チップのバリエーション
(ROM/RAM/ ピン数違いくらいの粒度)

ここを
作ります



S7G2 ポーティング (1/2): チップ



Drivers
Upper-half Driver

Architecture Specific Logic
Lower-half Driver
ARM 固有部分
S7G2 固有部分

1 Lower-half は，多くの場合アーキ / チップバリエーションくらいの 2 階層

ARM 固有部分
@arch/arm/src/common/arm_initialize.c

```
void up_initialize(){
  #ifdef USE_SERIALDRIVER
  arm_serialinit();
  #endif
}
```

2 ARM のシリアルでは，共通処理として Kconfig に応じてシリアル有効化するかどうかを判断

S7G2 固有部分
@arch/arm/src/s7g2/s7g2_serial.c

```
void arm_serialinit(){
  uart_register("/dev/ttyS0"
                &g_sciport);
  return;
}
```

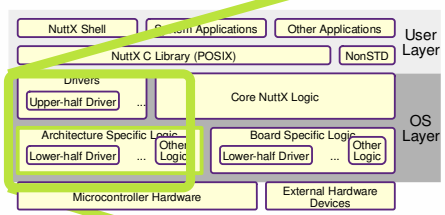
4 さいごに，Upper-half に登録 (Regist) する

```
struct uart_ops_s {
  int  (*attach      )(struct uart
  void (*detach      )(struct uart
  int  (*ioctl       )(struct file                    arg);
  int  (*receive     )(struct uart                    us);
  void (*rxint       )(struct uart
  void (*send        )(struct uart_dev_s *dev, int ch);
  void (*txint       )(struct uart_dev_s *dev, bool enable); };
```

3 チップ固有部分では，Upper-half が定義している構造体に S7G2 固有の関数をセット
これらの関数をがんばって書く

```
sruct uart_opt_s g_sciport;

static void s7g2 固有 detach(){
  …
}

g_sciport.detach
  = s7g2 固有 detach();
```

S7G2 ポーティング (1/2): チップ



Drivers
Upper-half Driver

Architecture Specific Logic
Lower-half Driver
ARM 固有部分
S7G2 固有部分

1 Lower-half は，多くの場合アーキ / チップバリエーションくらいの 2 階層

```
ARM 固有部分
@arch/arm/src/common/arm_initialize.c
void up_initialize(){
  #ifdef USE_SERIALDRIVER
  arm_serialinit();
  #endif
}
```

2 ARM のシリアルでは，共通処理として Kconfig に応じてシリアル有効化するかどうかを判断

```
S7G2 固有部分
@arch/arm/src/s7g2/s7g2_serial.c
void arm_serialinit(){
  uart_register("/dev/ttyS0"
                &g_sciport);
  return;
}
```

4 さいごに，Upper-half に登録 (Regist) する

```
struct uart_ops_s {
  int  (*attach      )(struct uart
  void (*detach      )(struct uart
  int  (*ioctl       )(struct file                    arg);
  int  (*receive     )(struct uart                    us);
  void (*rxint       )(struct uart
  void (*send        )(struct uart_dev_s *dev, int ch);
  void (*txint       )(struct uart_dev_s *dev, bool enable); };
```

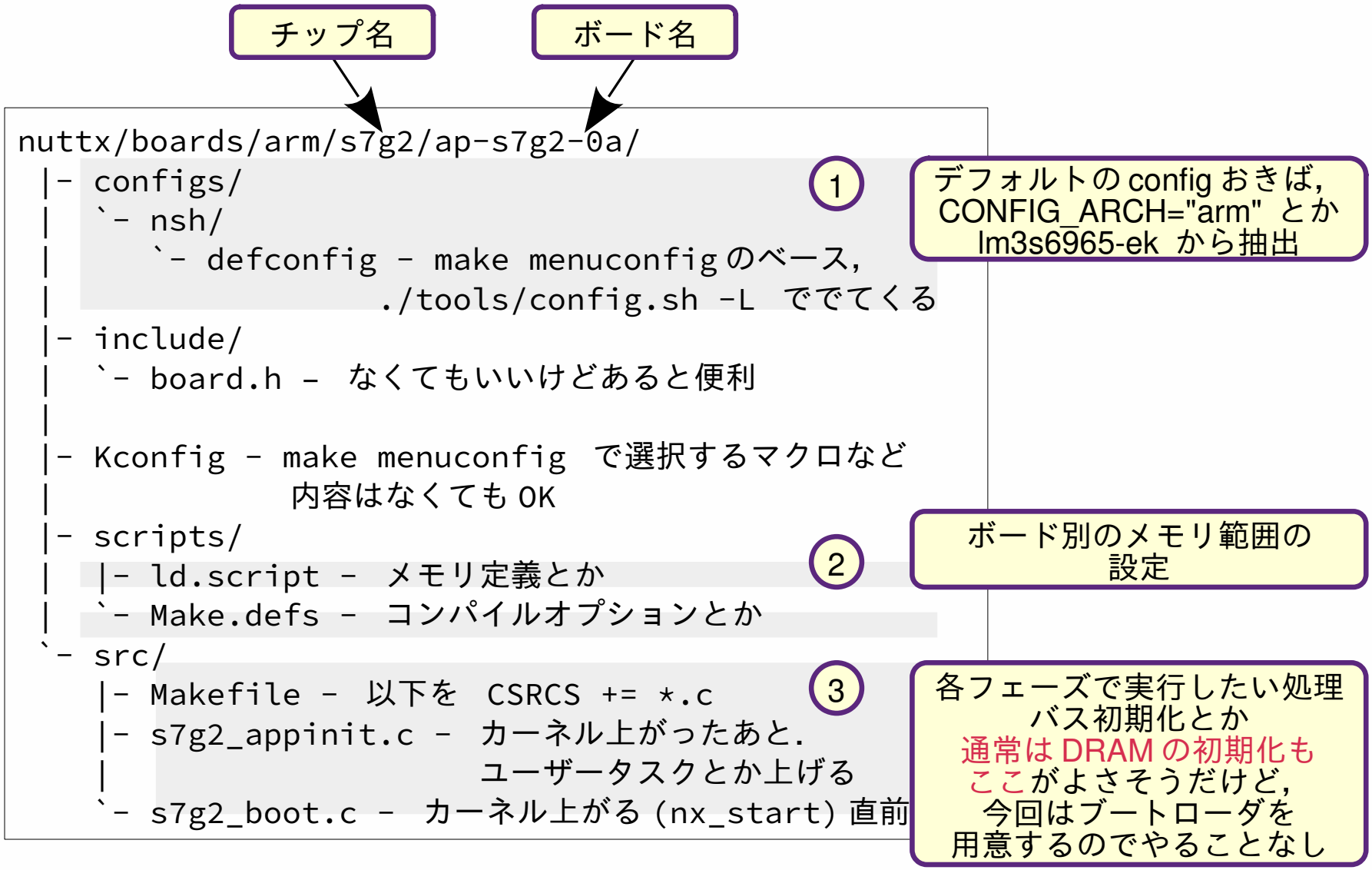
3 チップ固有部分では，Upper-half が定義している構造体に S7G2 固有の関数をセット
これらの関数をがんばって書く

```
sruct uart_opt_s g_sciport;

static void s7g2 固有 detach(){
  …
}

g_sciport.detach
  = s7g2 固有 detach();
```

チップができたので，次はボード．

S7G2 ポーティング (2/2): ボード

ボードのポーティング



S7G2 ポーティング (2/2): ボード

ここまででコンソールが,,，上がらない

S7G2 ポーティング (2/2): ボード

ここまででコンソールが,,，上がらない

- 現象 1: プロンプトの冒頭数文字が表示されたあと
  ハングアップしているように見える
- 現象 2: UART の送受信割込みが入ったままになっている

S7G2 ポーティング (2/2): ボード

ここまででコンソールが,,，上がらない

- 現象 1: プロンプトの冒頭数文字が表示されたあと
  ハングアップしているように見える
- 現象 2: UART の送受信割込みが入ったままになっている
- 確認 1: UART モジュール (SCI) を送受信無効，ステータスクリアしても
  上がったまま
- 確認 2: ARM 側割込みコントローラ (NVIC) で割込み無効にすると落ち
  るが，有効にすると直後に入る

S7G2 ポーティング (2/2): ボード

ここまででコンソールが,,，上がらない

- 現象 1: プロンプトの冒頭数文字が表示されたあと
  ハングアップしているように見える
- 現象 2: UART の送受信割込みが入ったままになっている
- 確認 1: UART モジュール (SCI) を送受信無効，ステータスクリアしても
  上がったまま
- 確認 2: ARM 側割込みコントローラ (NVIC) で割込み無効にすると落ち
  るが，有効にすると直後に入る
- 原因:

S7G2 ポーティング (2/2): ボード

ここまででコンソールが,,，上がらない

- 現象 1: プロンプトの冒頭数文字が表示されたあと
  ハングアップしているように見える
- 現象 2: UART の送受信割込みが入ったままになっている
- 確認 1: UART モジュール (SCI) を送受信無効，ステータスクリアしても
  上がったまま
- 確認 2: ARM 側割込みコントローラ (NVIC) で割込み無効にすると落ち
  るが，有効にすると直後に入る
- 原因:
  ICU(ルネサスの割込みコントーラ) がラッチしてた (SCI→ICU→NVIC，
  割込みコントローラが 2 段構成になってる..).

S7G2 ポーティング (2/2): ボード

ここまででコンソールが,,，上がらない

- 現象 1: プロンプトの冒頭数文字が表示されたあと
  ハングアップしているように見える
- 現象 2: UART の送受信割込みが入ったままになっている
- 確認 1: UART モジュール (SCI) を送受信無効，ステータスクリアしても
  上がったまま
- 確認 2: ARM 側割込みコントローラ (NVIC) で割込み無効にすると落ち
  るが，有効にすると直後に入る
- 原因:
  ICU(ルネサスの割込みコントーラ) がラッチしてた (SCI→ICU→NVIC，
  割込みコントローラが 2 段構成になってる..).

コンソール動いたので，QEMU で準備した MuPDF を持ってこよう！

MuPDF 移植

QEMU のがそのままビルドできた．わぁい．
さっそくデバッガで DRAM にロードして動作確認したいけど，，

MuPDF 移植

QEMU のがそのままビルドできた．わぁい．
さっそくデバッガで DRAM にロードして動作確認したいけど，，

- 現象 1: 意図しないタイミングで不正命令アボートとなる
  エラーのパターン
- 現象 2: 簡易メモリチェックをすると，
  特定のアドレス付近でベリファイエラーとなる．

MuPDF 移植

QEMU のがそのままビルドできた．わぁい．
さっそくデバッガで DRAM にロードして動作確認したいけど，，

- 現象 1: 意図しないタイミングで不正命令アボートとなる
  エラーのパターン
- 現象 2: 簡易メモリチェックをすると，
  特定のアドレス付近でベリファイエラーとなる．
- 試行 1: 16/32bit アクセスで頻度やアドレスは変化するけど
  ベリファイエラー発生は継続．
- 試行 2: クロックソースを内蔵 OSC →外部水晶にしたら多少改善．

MuPDF 移植

QEMU のがそのままビルドできた．わぁい．
さっそくデバッガで DRAM にロードして動作確認したいけど，，

- 現象 1: 意図しないタイミングで不正命令アボートとなる
  エラーのパターン
- 現象 2: 簡易メモリチェックをすると，
  特定のアドレス付近でベリファイエラーとなる．
- 試行 1: 16/32bit アクセスで頻度やアドレスは変化するけど
  ベリファイエラー発生は継続．
- 試行 2: クロックソースを内蔵 OSC →外部水晶にしたら多少改善．
- 確認:
  DRAM に対するクロックやウェイトの設定は Synergy 版と同じ．

MuPDF 移植

QEMU のがそのままビルドできた．わぁい．
さっそくデバッガで DRAM にロードして動作確認したいけど，，

- 現象 1: 意図しないタイミングで不正命令アボートとなる
  エラーのパターン
- 現象 2: 簡易メモリチェックをすると，
  特定のアドレス付近でベリファイエラーとなる．
- 試行 1: 16/32bit アクセスで頻度やアドレスは変化するけど
  ベリファイエラー発生は継続．
- 試行 2: クロックソースを内蔵 OSC →外部水晶にしたら多少改善．
- 確認:
  DRAM に対するクロックやウェイトの設定は Synergy 版と同じ．
- 原因:

MuPDF 移植

QEMU のがそのままビルドできた．わぁい．
さっそくデバッガで DRAM にロードして動作確認したいけど，，

- 現象 1: 意図しないタイミングで不正命令アボートとなる
  エラーのパターン
- 現象 2: 簡易メモリチェックをすると，
  特定のアドレス付近でベリファイエラーとなる．
- 試行 1: 16/32bit アクセスで頻度やアドレスは変化するけど
  ベリファイエラー発生は継続．
- 試行 2: クロックソースを内蔵 OSC →外部水晶にしたら多少改善．
- 確認:
  DRAM に対するクロックやウェイトの設定は Synergy 版と同じ．
- 原因:
  端子駆動能力の設定不足 (デフォルト: 低駆動 → 高駆動 で解消)．

MuPDF 移植

QEMU のがそのままビルドできた．わぁい．
さっそくデバッガで DRAM にロードして動作確認したいけど，，

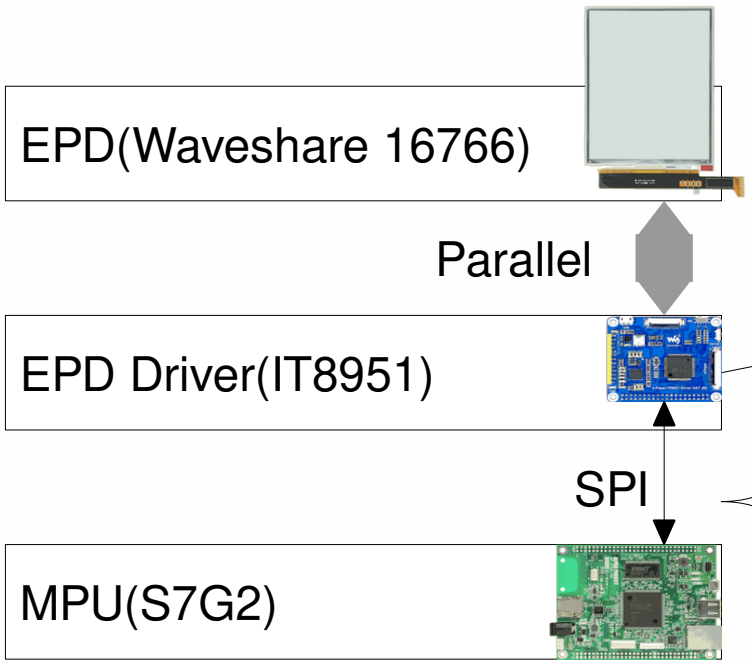
- 現象 1: 意図しないタイミングで不正命令アボートとなる
  エラーのパターン
- 現象 2: 簡易メモリチェックをすると，
  特定のアドレス付近でベリファイエラーとなる．
- 試行 1: 16/32bit アクセスで頻度やアドレスは変化するけど
  ベリファイエラー発生は継続．
- 試行 2: クロックソースを内蔵 OSC →外部水晶にしたら多少改善．
- 確認:
  DRAM に対するクロックやウェイトの設定は Synergy 版と同じ．
- 原因:
  端子駆動能力の設定不足 (デフォルト: 低駆動 → 高駆動 で解消)．

MuPDF も動いたので，最後は EPD ドライバ．

EPD ドライバ

1. 構成，動作
2. ちょい工夫

EPD ドライバ: 構成，動作



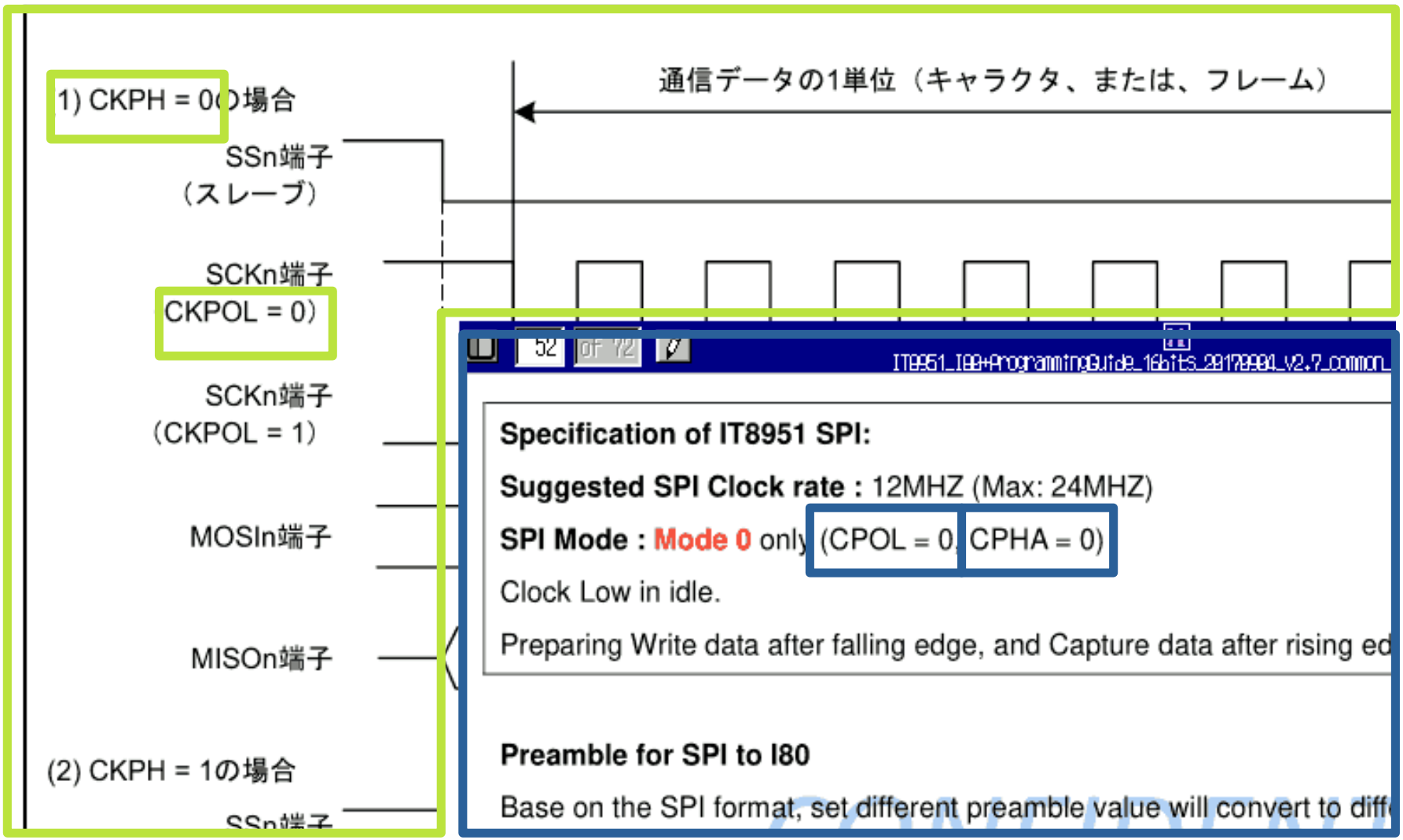
SDRAM 4/8MB を内蔵

一旦ここに書き込んでから，EPD に対する出力を指示する

- 画像バッファ転送
- 更新モード指定
- 指定領域の表示更新

EPD ドライバ: 構成，動作

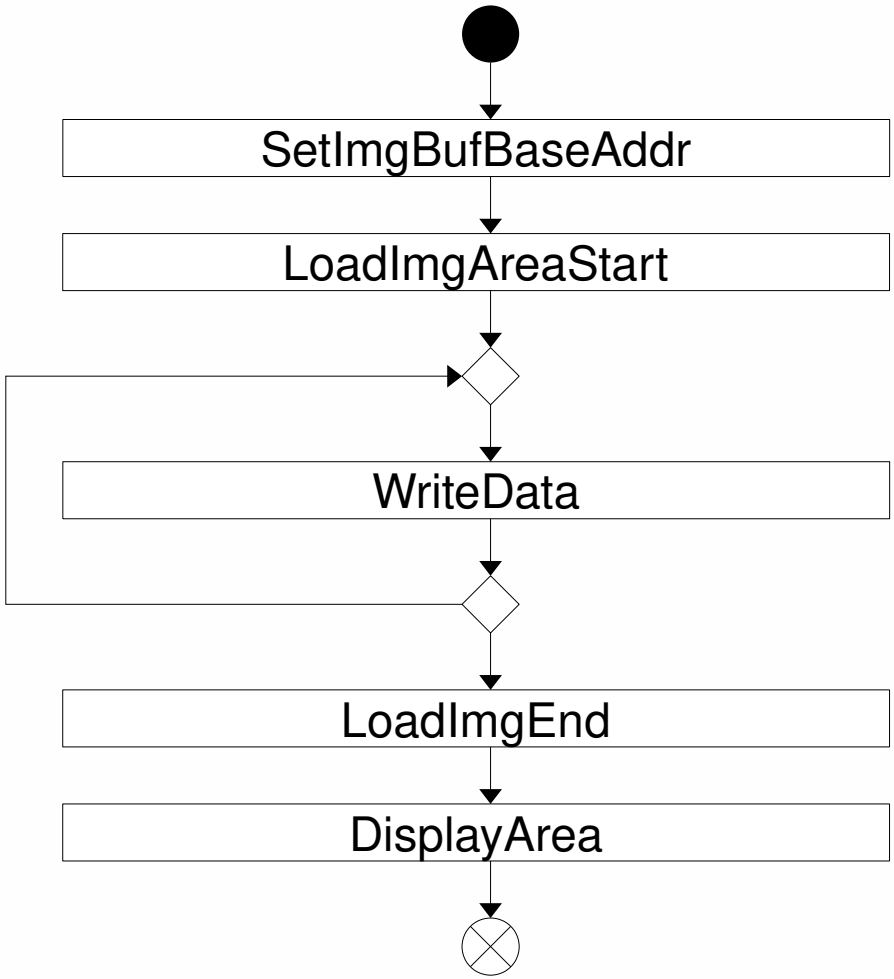
微妙にハマりポイント



S7G2 のドキュメント

IT8951 のドキュメント

EPD ドライバ: ちょい工夫



IT8951 側のバッファアドレスを指定
(省略してますが，初期化時に IT8951 をリードして確認します)

転送開始を示すコマンド

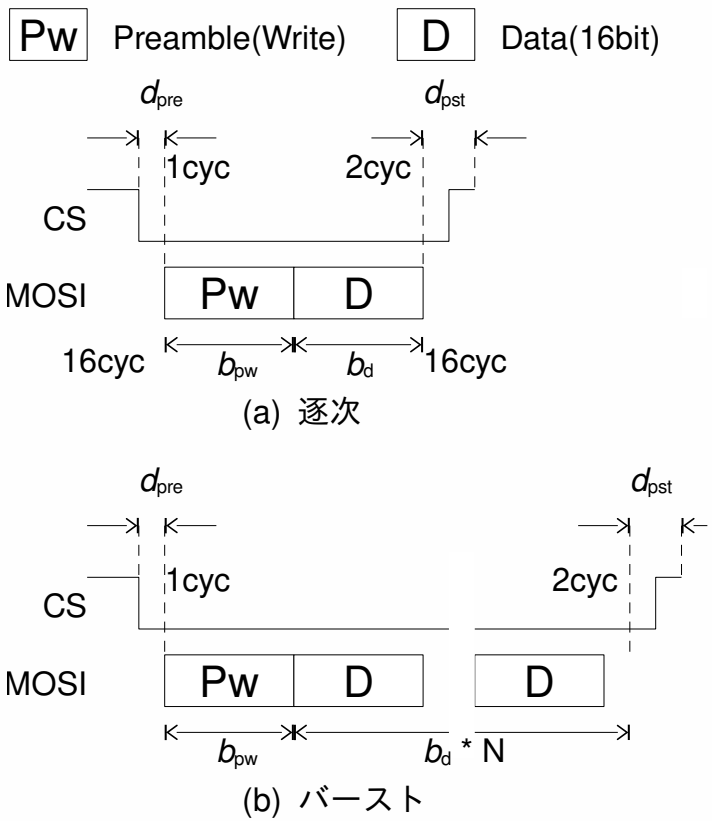
実データの送信

転送終了を示すコマンド

描画指示のコマンド
(部分的な更新も可能)

EPD ドライバ: ちょい工夫

サンプルプロジェクトでは 1 画素ごとに SPI の転送命令してておっそい．
バースト化+4bpp 化で 3s→0.7s くらいに．



逐次の場合の全画面更新時間は，

$$\frac{wh}{b_{\mathrm{oct}}}\left(b_{\mathrm{D}} + b_{\mathrm{pw}} + d_{\mathrm{pre}} + d_{\mathrm{pst}}\right)\frac{1}{f_{\mathrm{sck}}} \simeq 3.07\mathrm{sec}.$$

where f_{sck} = 15MHz, b_{pw} = 16bit, b_{D} = 16bit, d_{pre} = 1cyc, d_{pst} = 2cyc, w = 1872px, h = 1404, b_{oct} = 8bit, b_{qua}bit.

バーストすると以下 [1]．

$$\left(\frac{wh}{b_{\mathrm{oct}}}b_{\mathrm{D}} + b_{\mathrm{pw}} + d_{\mathrm{pre}} + d_{\mathrm{pst}}\right)\frac{1}{f_{\mathrm{sck}}} \simeq 1.40\mathrm{sec}.$$

さらに 1 画素あたり 8bit→4bit(b_{qua}) の設定にすると 0.70sec くらいまで短縮 [2]．投機的に転送しておけば気にならなさそう？

[1]実際は，ビットエラーの都合で 1 行ごとくらい
[2]1byte で 2 画素送れる．デバイスが 16 階調なので十分だけど DMA とは相性よくない

よりみち: EPD 表示制御

品質と更新時間のトレードオフ

Mode	Grad.	Dly[ms]	Ghost	Usage	Remarks
INIT	-	2000	なし	初期化	
GC16	16	450	とても少	高画質	
GLD16	16	450	少	白地のテキストおよび画像	全画面向け
GLR16	16	450	少	白地のテキスト	29,30 遷移○
GL16	16	450	中	白地のテキスト	29,30 遷移×
DU4	4	290	中	メニュー，ペン入力 (高画質)	
DU	2	260	少	メニュー，テキスト編集，ペン入力	
A2	2	120	中	高速ページめくり	

S7G2 ポーティング: いったんまとめ

- ✓ S7G2 ポーティング
  チップとボード用のコンフィグ作成，実機でブート確認
- ✓ MuPDF 移植
  QEMU コンフィグの流用確認，DRAM 初期化修正
- ✓ EPD ドライバ
  SPI 極性調整，転送バースト化，

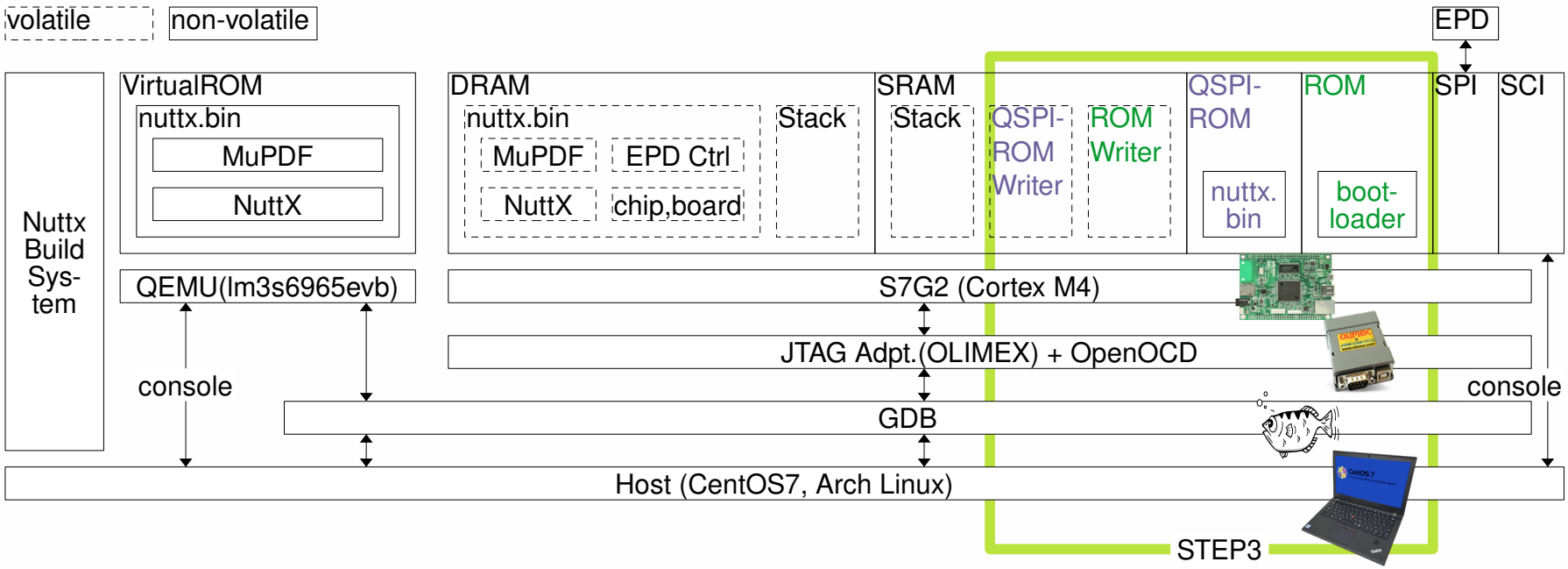
S7G2 ポーティング: いったんまとめ

- ✓ S7G2 ポーティング
  チップとボード用のコンフィグ作成，実機でブート確認
- ✓ MuPDF 移植
  QEMU コンフィグの流用確認，DRAM 初期化修正
- ✓ EPD ドライバ
  SPI 極性調整，転送バースト化，

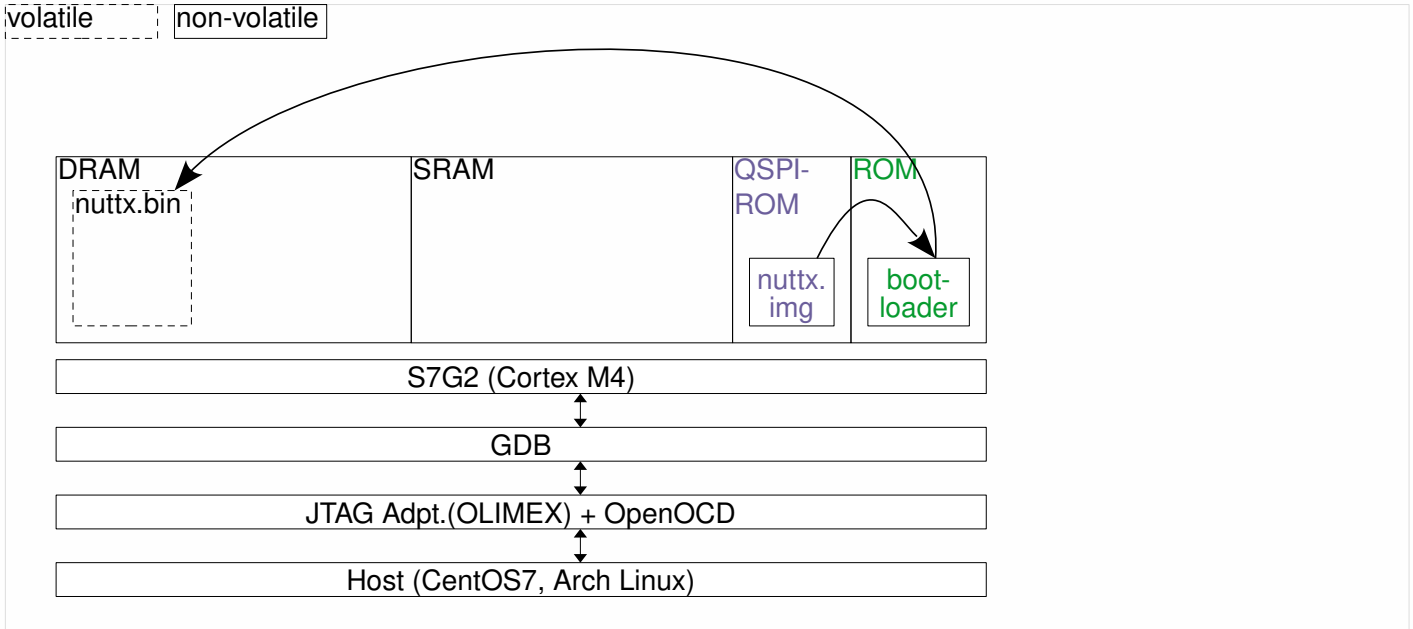
最後に，デバッガレスで起動できるように
ブートローダを作成します

S7G2 ポーティング

STEP3，自立起動をめざす



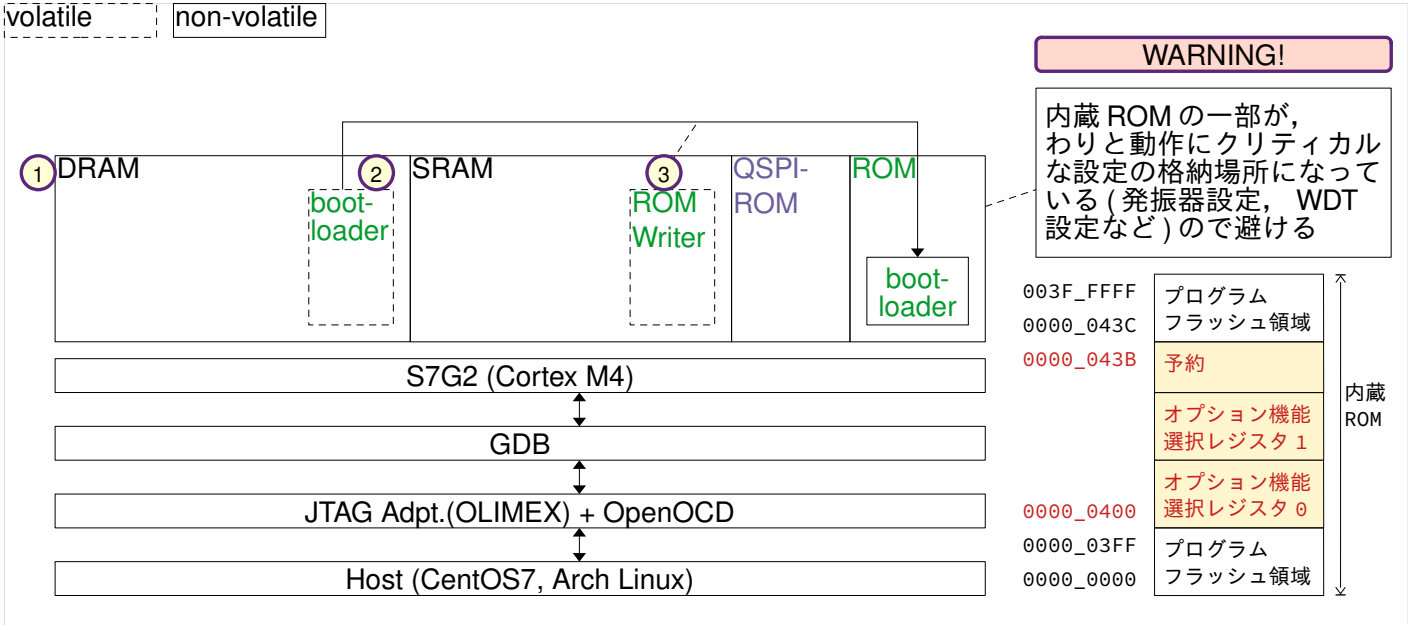
ブートローダ: 構成と動作



まず，最終的にこんなかんじにしたいな，のイメージ

- 内蔵 ROM(4MB) に格納された `bootloader` が，
  QSPI-ROM(16MB) の `nuttx.img` を DRAM にコピーしたあと，
  DRAM にジャンプ

ブートローダ: 構成と動作

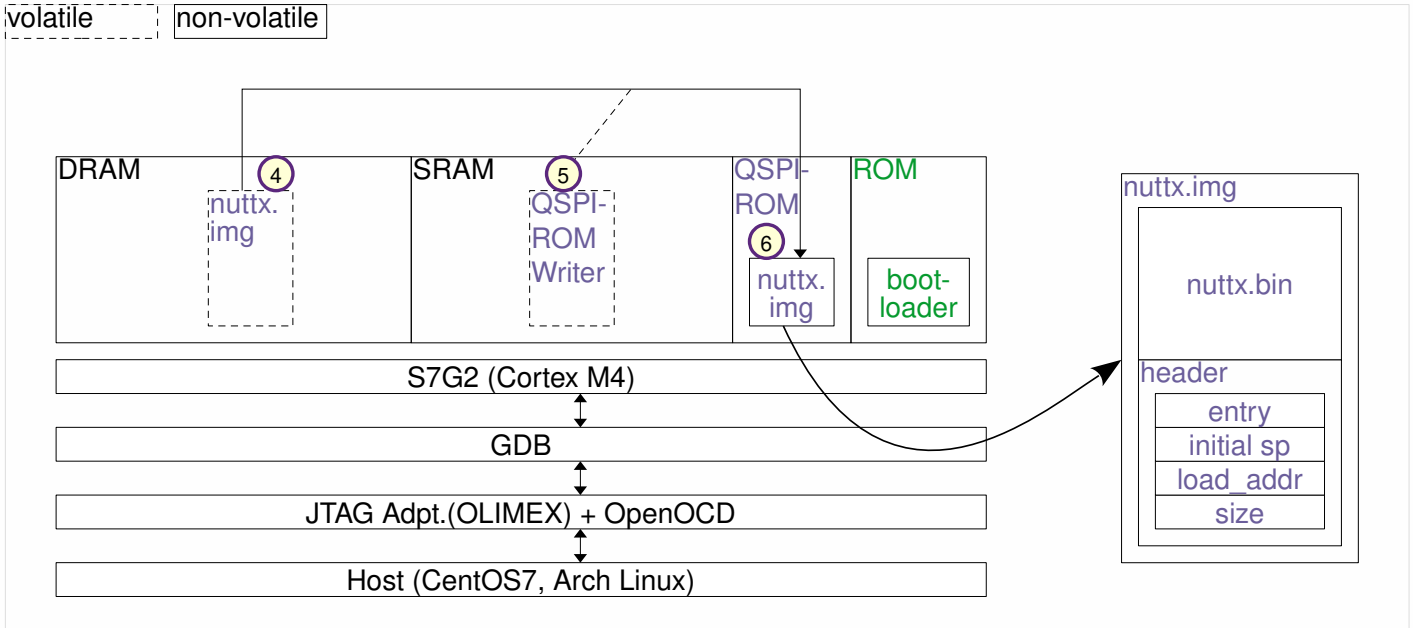


まず，内蔵 ROM に書込む準備をします．以下はデバッガの操作．

1. DRAM 初期化
2. `bootloader` のバイナリを DRAM に配置
3. 内蔵 ROM ライタを SRAM に配置・実行して DRAM→ 内蔵 ROM[1] にコピー

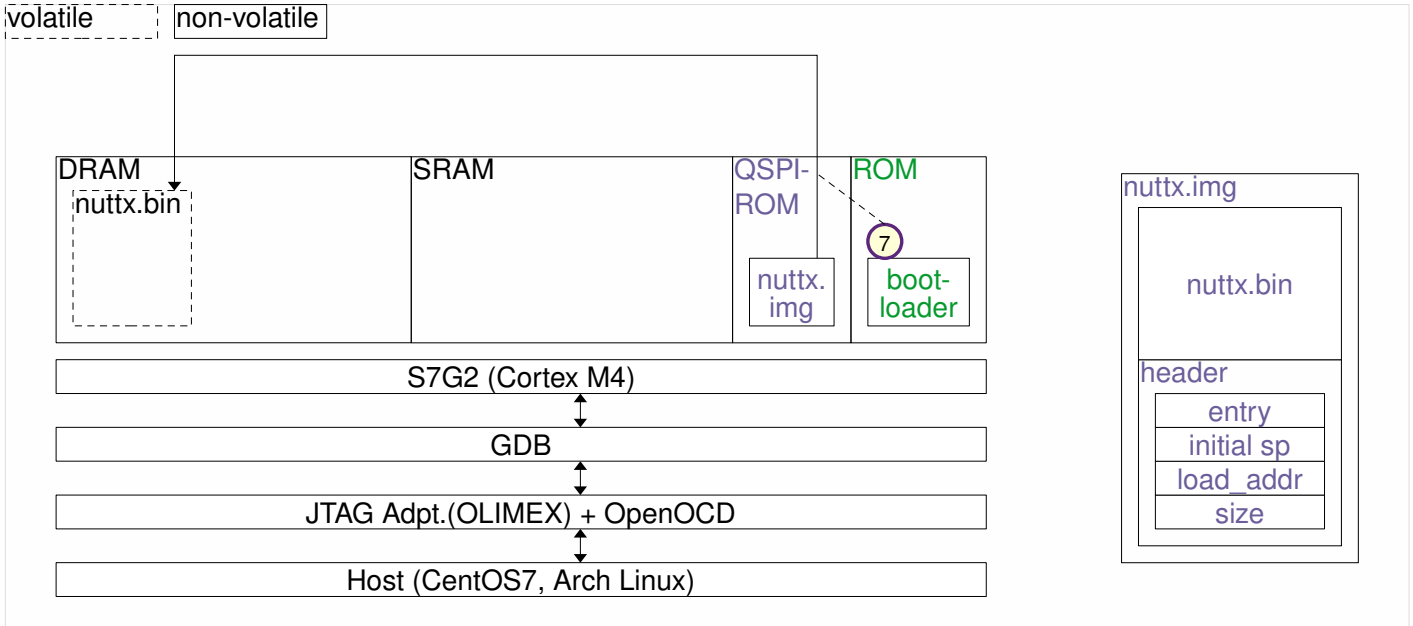
[1]ちなみに S7G2 内蔵フラッシュコントローラのドキュメントがルネサスの日本語/英語サイト探した限り見付からなかったのですが，RX 用の見ながらなんとなくつくったらうごいた．

ブートローダ: 構成と動作



4. `nuttx.img` を DRAM に配置
5. QSPI-ROM ライタを SRAM に配置・実行して
6. DRAM→QSPI-ROM にコピー

ブートローダ: 構成と動作



7 リセットすると，`bootloader` が以下を実施

- DRAM 初期化
- QSPI 初期化
- `nuttx.img` の `header` に従って，`nuttx.bin` を `size` だけ，`load_addr` にコピーしたあと，スタックポインタに `initial_sp` をセットして `entry` にジャンプ

ブートローダ: 構成と動作

ついに独立動作,,, と思ったら,,,

- 現象 1: リセットして内蔵 ROM から走らせると,
  意図しないアドレスにジャンプして未定義命令違反になる

ブートローダ: 構成と動作

ついに独立動作,,, と思ったら,,,

- 現象 1: リセットして内蔵 ROM から走らせると,
  意図しないアドレスにジャンプして未定義命令違反になる
- 確認 1: ジャンプするタイミングはクロック設定のあと 1，2 命令後
  (LED で確認)

ブートローダ: 構成と動作

ついに独立動作,,, と思ったら,,,

- 現象 1: リセットして内蔵 ROM から走らせると，
  意図しないアドレスにジャンプして未定義命令違反になる
- 確認 1: ジャンプするタイミングはクロック設定のあと 1，2 命令後
  (LED で確認)
- 確認 2: 当該アドレスをデバッガでダンプすると初回はオールゼロ，2
  回目のアクセスで期待値を読む

ブートローダ: 構成と動作

ついに独立動作,,, と思ったら,,,

- 現象 1: リセットして内蔵 ROM から走らせると，
  意図しないアドレスにジャンプして未定義命令違反になる
- 確認 1: ジャンプするタイミングはクロック設定のあと 1，2 命令後
  (LED で確認)
- 確認 2: 当該アドレスをデバッガでダンプすると初回はオールゼロ，2
  回目のアクセスで期待値を読む
- 原因:

ブートローダ: 構成と動作

ついに独立動作,,, と思ったら,,,

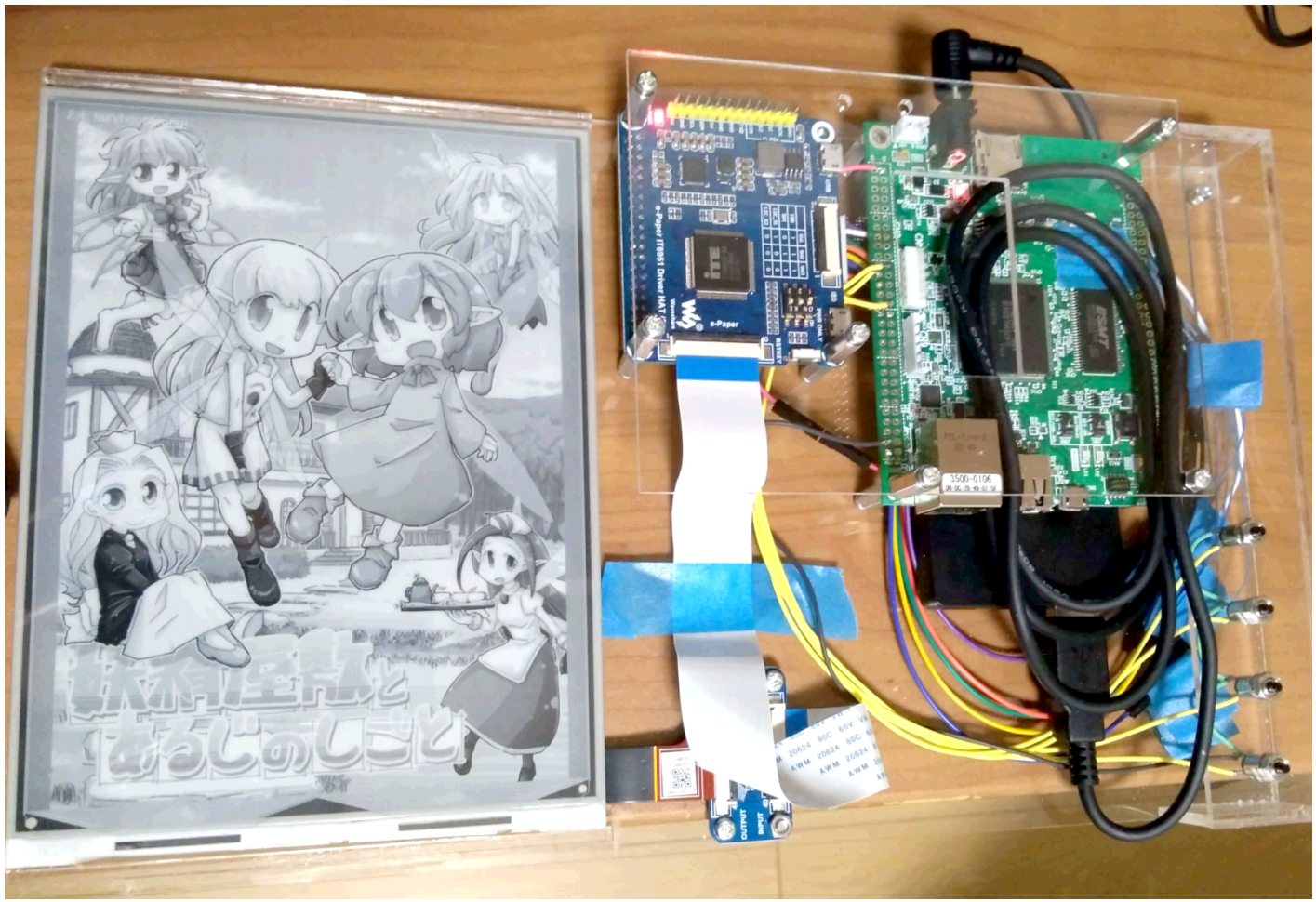
- 現象 1: リセットして内蔵 ROM から走らせると，
  意図しないアドレスにジャンプして未定義命令違反になる
- 確認 1: ジャンプするタイミングはクロック設定のあと 1，2 命令後
  (LED で確認)
- 確認 2: 当該アドレスをデバッガでダンプすると初回はオールゼロ，2 回目のアクセスで期待値を読む
- 原因:
  クロック設定変更前 (高速化前) の，FlashROM ウェイト数調整漏れ．
  80MHz 以下 0 ウェイト，80MHz–160MHz 1 ウェイト，
  160MHz–240MHz 2 ウェイト必要．
  (ウェイトなし状態で高速クロックに切り替えると，
  FlashROM から不定値を読んですっとぶ)

参考: WaveShare 社のサンプルプロジェクト

デモ,,, のまえに,
WaveShare 社のサンプルプロジェクト.

デモ

デモー.



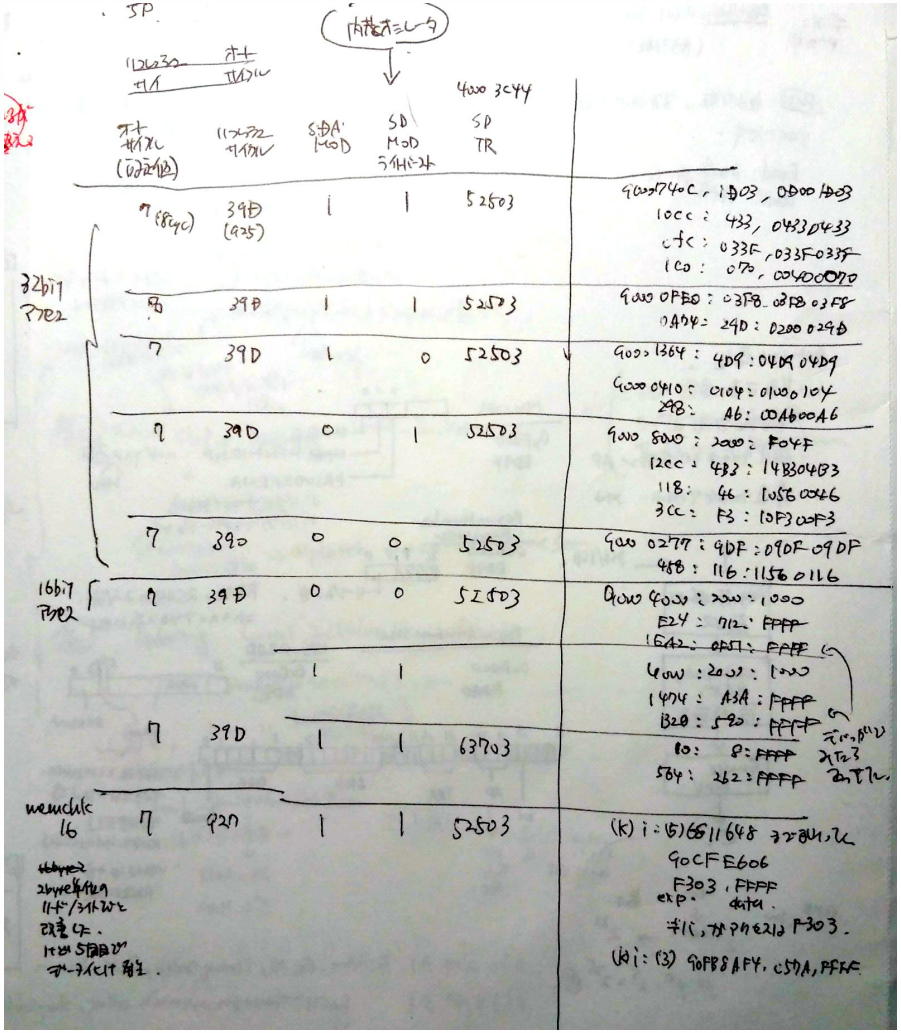
今後の課題

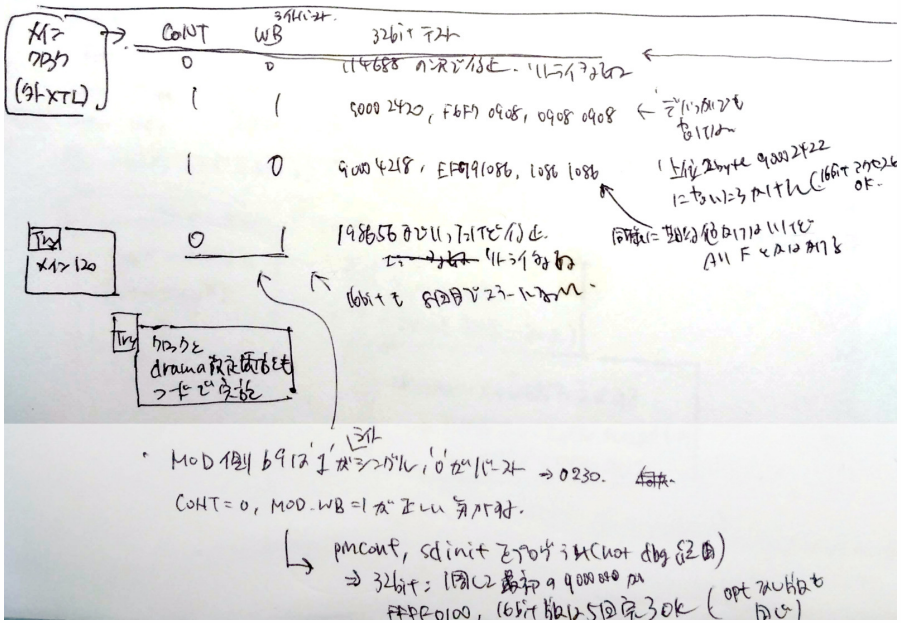
- 何はともあれ高速化
  FreeType,，HarfBuzz あたり,,
  キャッシュ/DMA で隠蔽．あきらめてプリレンダ．
  Cortex-A 採用とか..
- 2 画面化
- SD カード対応
- 拡大率指定センタリング
- QEMU と実機を行き来するときに
  毎回全体リビルドかからないようにしたい
- もっと低解像度でいい (現状 1.8xK1.4K)
- ナビ用に小さい OLED 併用とか

endro-

- 背景:
  - 電子書籍リーダーをつくってみたくなりました
- 設計:
  - 構成部品の選定方針を説明しました
  - POSIX 風オープンソース RTOS のひとつである，NuttX を紹介しました
- 開発
  - ルネサスの ARM チップ (S7G2) を NuttX にポーティングする手順を説明しました
  - QEMU ベースでデバッグしつつ，実機へ移行する手順を説明しました
- いろいろハマったことをご紹介しました
- 試作品をデモしました

ご静聴どうもありがとうございました





もどる

model	inch	px	dim[mm]	ref[s]	pwr[W]	I/F	direct($/¥)	sengoku(¥)
13504	7.5’	800x480	163x98	5	0.04/0.02m	SPI	56.99/8186	8450
15195	9.7’	1200x825	203x140	1	0.6/0.3		176.99/25417	20350
16766	7.8’	1872x1404	158x119	0.45	1.2/0.1	SPI/I80	150.99/21691	19150
16712	10.3’	1872x1404	210x157	0.45	1.2/0.1		247.99/35619	30000
20129	7.5’	800x480	163x98	16	0.05/0.01(uA?)		57.99	8470

Table 1: Waveshare 社の製品ラインナップと価格

S7G2	
CKPH	位相遅延 (0: 遅延あり，1: 遅延なし)
CKPOL	極性反転
IT8951	
CPHA	位相遅延 (0: 遅延なし，1: 遅延あり)
CPOL	アイドル時クロック極性

Table 2: 略語の意味

CKPH	CKPOL	取込極性	定常	有効エッジ位相
0	0	↑	H	even
0	1	↓	L	even
1	0	↑	L	odd
1	1	↓	H	odd
CPHA	CPOL	取込極性	定常	有効エッジ位相
0	0	↑	L	odd
0	1	↓	H	odd
1	0	↓	L	even
1	1	↑	H	even

Table 3: 設定値と動作の対応関係