SCHLOA v1.0.0

Compact Portable Spectrophotometer Developed by Kaiji Takeuchi 2019

1. 概要

SCHLOA (Fig.1.1) は分光分析法を採用した超小型ポータブル分光光度計です．コア・デバイスである分光器には Hamamatsu Photonics 製ミニ分光器に対応しています．本システムは持ち運べるサイズながら，反射，吸収，拡散及び散乱光の分光分析に対する，幅広い応用が可能です．コア・システムには Arm® Mbed™を採用しているため，ファームウェア開発も容易になります．

2. 特徴

・32 ビット Cortex-M4 コア・プロセッサ：STM32F446RE(180MHz)
・32 ビット Cortex-M4 デジタルフロントエンド・プロセッサ：STM32L432KC(80MHz)
・Arm® Mbed™ Mbed OS5 対応
・対応分光器波長感度
　Hamamatsu Photonics MS series
　　-C11708MA：640nm～1050nm
　　-C10988MA-01：340nm～750nm
　Hamamatsu Photonics micro series
　　-C12880MA：340nm～780nm
　　-C12666MA：340nm～850nm
・蓄積(露出)時間：～100µs
・ADC キャプチャ・レート：～1MHz
・高精度・高速セトリング アナログフロントエンド
・ADC 専用リファレンス電源内臓
・LED ドライバ内臓
　-PWM(電流フィードバック)
・リアルタイムクロック内臓
・各データ転送(記録)方式に対応
　-Bluetooth SPP (専用基板が必要)
　-USB I/F
　-SD Memory Card

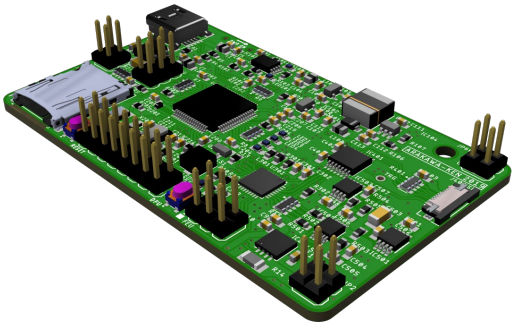

Fig. 1.1 SCHLOA Board

3. 目次

4. ブロック図と機能説明

■ メインボード

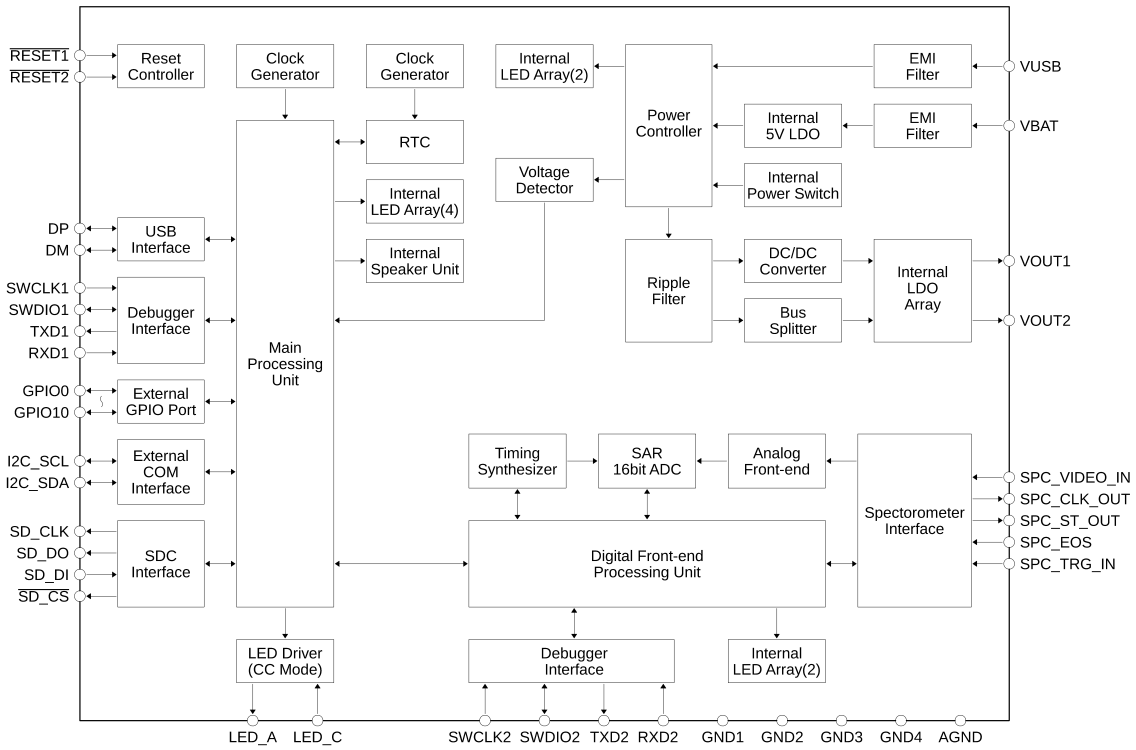

Fig. 4.1 Main Board's System Block Diagram

5. ピンマップと機能説明

■ ピンマップ

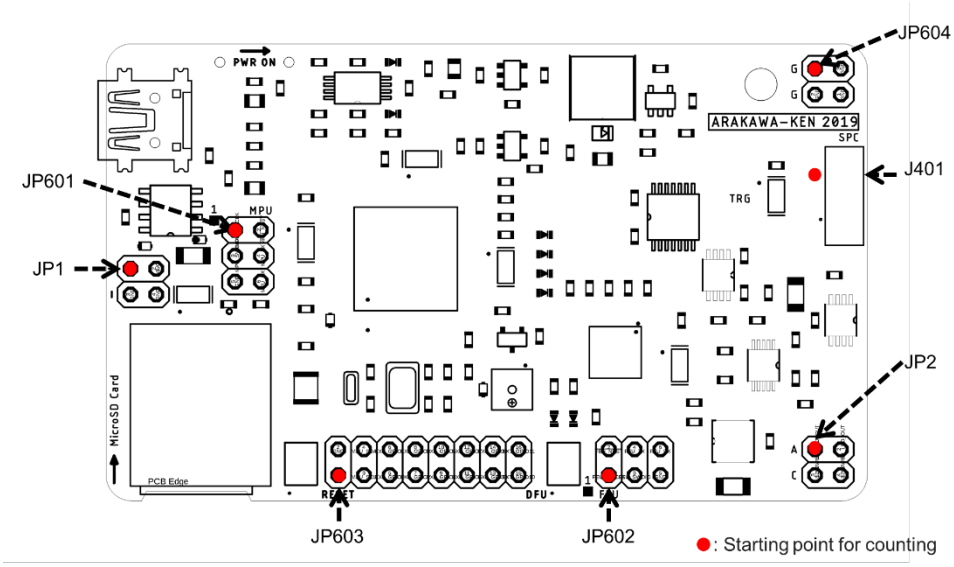

Fig. 5.1 Junction Ports Map

・Supply Port (JP1)

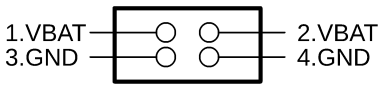

Fig. 5.2 JP1 Pins Map

・MPU Debug Port (JP601)

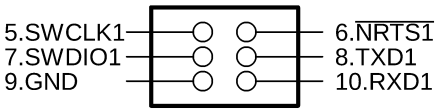

Fig. 5.3 JP601 Pins Map

・FEU Debug Port (JP602)

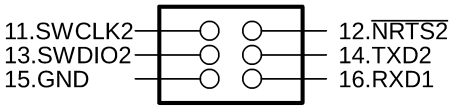

Fig. 5.4 JP602 Pins Map

▪ External Connection Port (JP603)

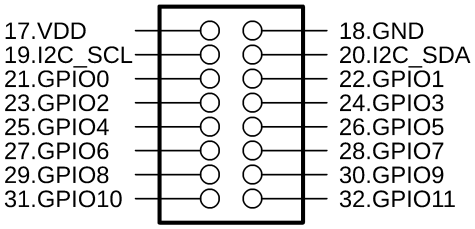

Fig. 5.5 JP603 Pins Map

▪ LED Output Port (JP2)

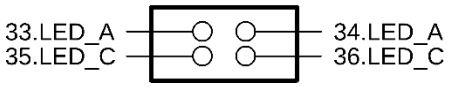

Fig. 5.6 JP2 Pins Map

▪ Spectrometer Connection Port (J401)

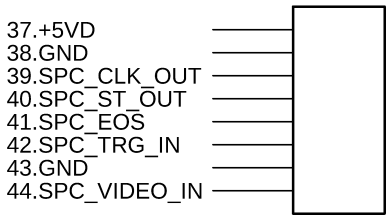

Fig. 5.7 J401 Pins Map

▪ Auxiliary Port (JP604)

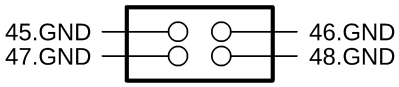

Fig. 5.8 JP604 Pins Map

■ 機能説明

Tabel 5.1 Pin Functions

NO.	Symbol	I/O	Tolerant	Function
1	VBAT	-	-	Power Supply Pin, 5.1~10V
2	VBAT	-	-	Power Supply Pin, 5.1~10V
3	GND	-	-	Ground Pin
4	GND	-	-	Ground Pin
5	SWCLK1	I	FT	MPU JTAG/SWD Clock Input Pin
6	NRST1	I	TT	MPU Reset Input Pin, Active Low
7	SWDIO1	I/O	FT	MPU JTAG/SWD Data I/O Pin
8	TXD1	O	FT	MPU UART Transmit Data Pin -Connect the RXD pin of the debugger.
9	GND	-	-	Ground Pin
10	RXD1	I	FT	MPU UART Receive Data Pin -Connect the RXD pin of the debugger.
11	SWCLK2	I	FT	FEU JTAG/SWD Clock Input Pin -Connect the SWCLK pin of the debugger.
12	NRST2	I	TT	FEU Reset Input Pin, Active Low -Connect the Reset pin of the debugger.
13	SWDIO2	I/O	FT	FEU JTAG/SWD Data I/O Pin -Connect the SWDIO pin of the debugger.
14	TXD2	O	FT	FEU UART Transmit Data Pin -Connect the RXD pin of the debugger.
15	GND	-	-	Ground Pin
16	RXD2	I	FT	FEU UART Receive Data Pin -Connect the RXD pin of the debugger
17	+5VD	-	-	Digital Power Output Pin, +5V Constant
18	GND	-	-	Ground Pin
19	I2C_SCL~	I/O	FT	I-squared-C Slave Clock Pin, Open Drain
20	I2C_SDA~	I/O	FT	I-squared-C Slave Data Pin, Open Drain
21	EXT_GPIO0	I/O	FT	External General-Purpose Input/Output Pin
22	EXT_GPIO1	I/O	FT	External General-Purpose Input/Output Pin -PWM is Available
23	EXT_GPIO2	I/O	FT	External General-Purpose Input/Output Pin -PWM is Available.

24	EXT_GPIO3	I/O	FT	External General-Purpose Input/Output Pin -PWM is Available.
25	EXT_GPIO4	I/O	FT	External General-Purpose Input/Output Pin -PWM is Available.
26	EXT_GPIO5	I/O	FT	External General-Purpose Input/Output Pin -PWM is Available.
27	EXT_GPIO6	I/O	FT	External General-Purpose Input/Output Pin
28	EXT_GPIO7	I/O	FT	External General-Purpose Input/Output Pin
29	EXT_GPIO8	I/O	FT	External General-Purpose Input/Output Pin
30	EXT_GPIO9	I/O	FT	External General-Purpose Input/Output Pin
31	EXT_GPIO10	I/O	FT	External General-Purpose Input/Output Pin
32	EXT_GPIO11	I/O	FT	External General-Purpose Input/Output Pin
33	LED_A	O	-	Power LED Anode Pin, MAX 500mA
34	LED_A	O	-	Power LED Anode Pin, MAX 500mA
35	LED_C	I	-	Power LED Cathode Pin, Current FB
36	LED_C	I	-	Power LED Cathode Pin, Current FB
37	+5VD	-	-	Digital Power Output Pin, +5V Constant
38	GND	-	-	Ground Pin
39	SPC_CLK_OUT*	O	FT	Spectrometer Clock Output Pin -Mounting R502, Auto Trigger Mode. -Mounting R501, Manual Trigger Mode.
40	SPC_ST_OUT*	O	FT	Spectrometer Start Pulse Output Pin
41	SPC_EOS*	I	FT	Spectrometer End of Scan Detect Pin
42	SPC_TRG_IN*	I	FT	Spectrometer Trigger Pulse Input Pin - No connection with controller
43	GND	-	-	Ground Pin
44	SPC_VIDEO_IN	I	-	Spectrometer Video Signal Input, 0~+5V
45	GND	-	-	Ground Pin
46	GND	-	-	Ground Pin
47	GND	-	-	Ground Pin
48	GND	-	-	Ground Pin

Note1 : FT = 5V Input Tolerant, TT = 3.3V Input Tolerant.

Note2 : チルダ表記(~)された入出力ピンはオープン・ドレインです.

Note3 : アスタリスク表記(*)された入出力ピンは内部でプル・ダウンされています.

6. 絶対定格

Tabel 6.1 Absolute Maximum Ratings

Note：V_{SS} = 0V

Parameter		min	max	Unit
V_{BAT}	Battery Input Voltage	-0.3	10	V
V_{USB}	USB Input Voltage	-0.3	5.0	V
V_{CNT}	Controller Input Voltage	-0.3	4.0	V
V_{xx}-V_{SS}	Delta Ground Voltage	-0.3	-	V
$V_{IOFT(PIN)}$	GPIO FT Pins input Voltage	V_{SS} - 0.3	V_{CNT} + 4.0	V
$V_{IOTT(PIN)}$	GPIO TT Pins input Voltage	V_{SS} - 0.3	4.0	V
$I_{IO(PIN)}$	GPIOs I/O Current	-25	25	mA
$\Sigma I_{IO(PIN)}$	GPIOs Total I/O Current	-80	80	mA
I_{LED}	LED Driver Output Current	-	600	mA
T_A	Operating Free-Air Temperature	5	50	℃
T_{stg}	Storage Temperature	-20	70	℃

7. 推奨動作範囲

Tabel 7.1 Recommended Operating Condition Ranges

Parameter		min	typ	max	Unit
V_{BAT}	Battery Input Voltage	5.1	9.0	10	V
V_{USB}	USB Input Voltage	4.8	5.0	5.5	V
V_{CNT}	Controller Input Voltage	3.0	3.3	4.0	V
T_A	Operating Free-Air Temperature	5	20	40	℃

8. 内部回路とジャンクション

■ 内部回路

ピンラベルやワイヤラベルに若干の差異がありますが，内部回路は次のようになっています．ユニット毎に，部品番号に 100 刻みのオフセットが加わります．

・Main System

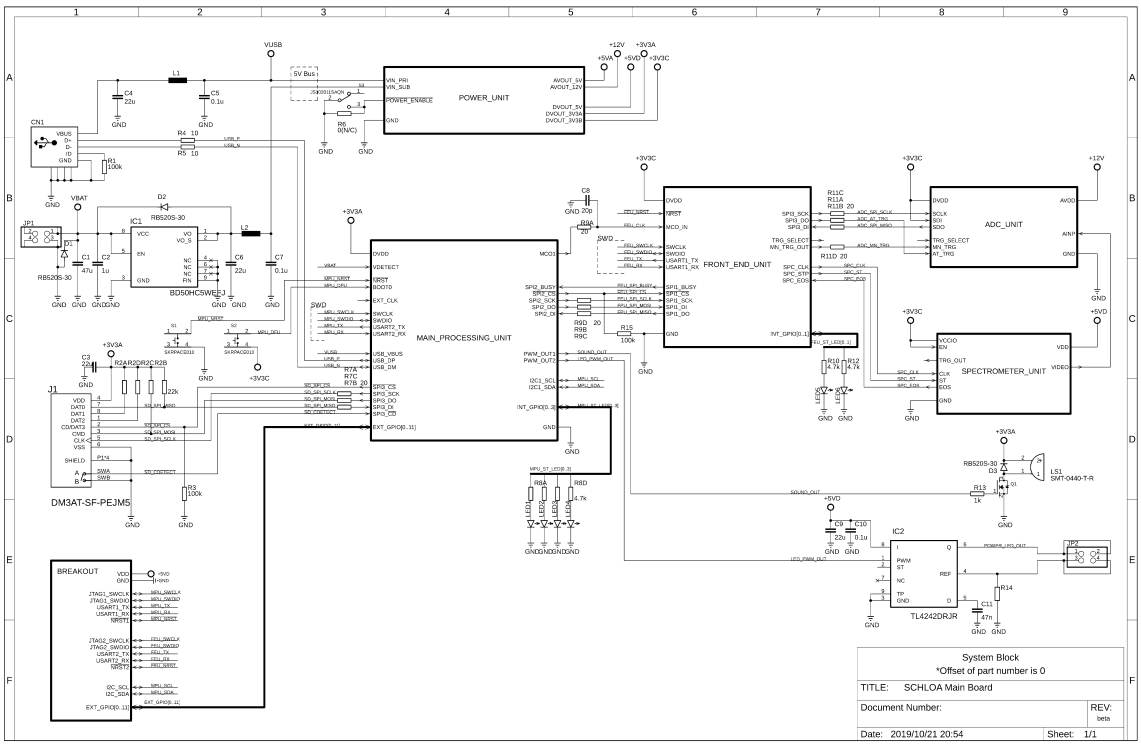

Fig. 8.1 Main Schematic

▪ Power Unit

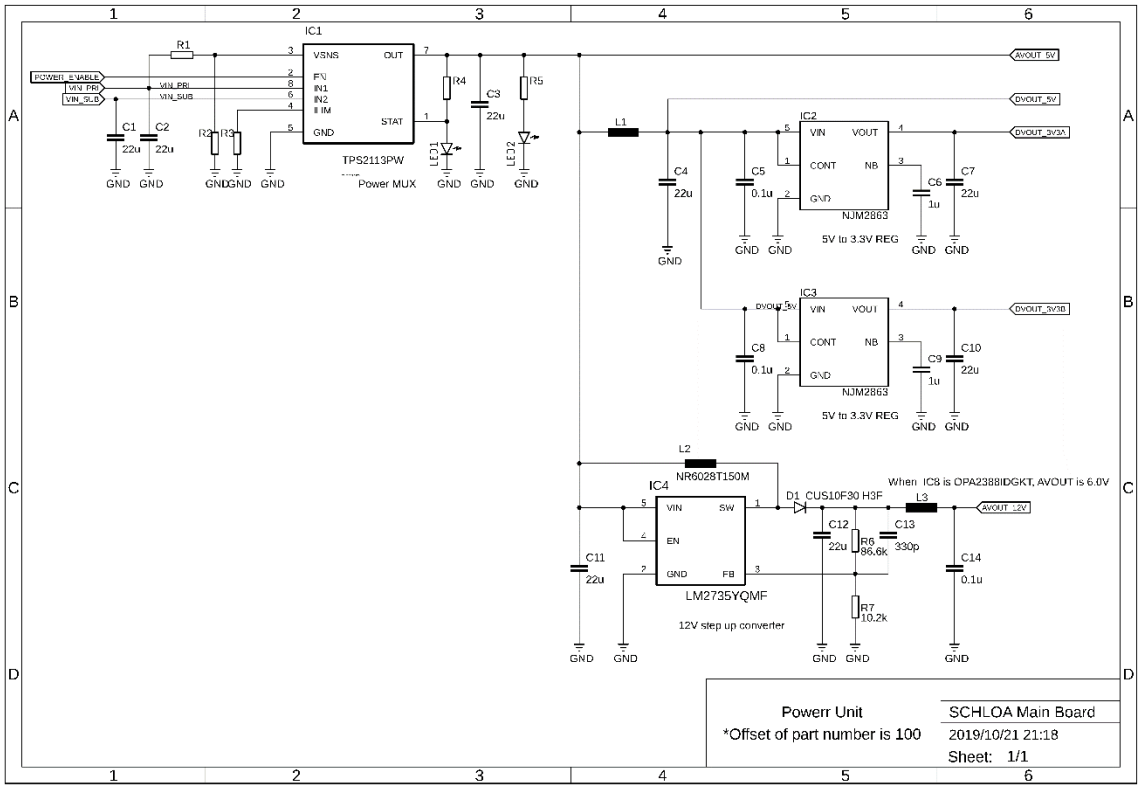

Fig. 8.2 Power Unit Schematic

▪ Main Processing Unit

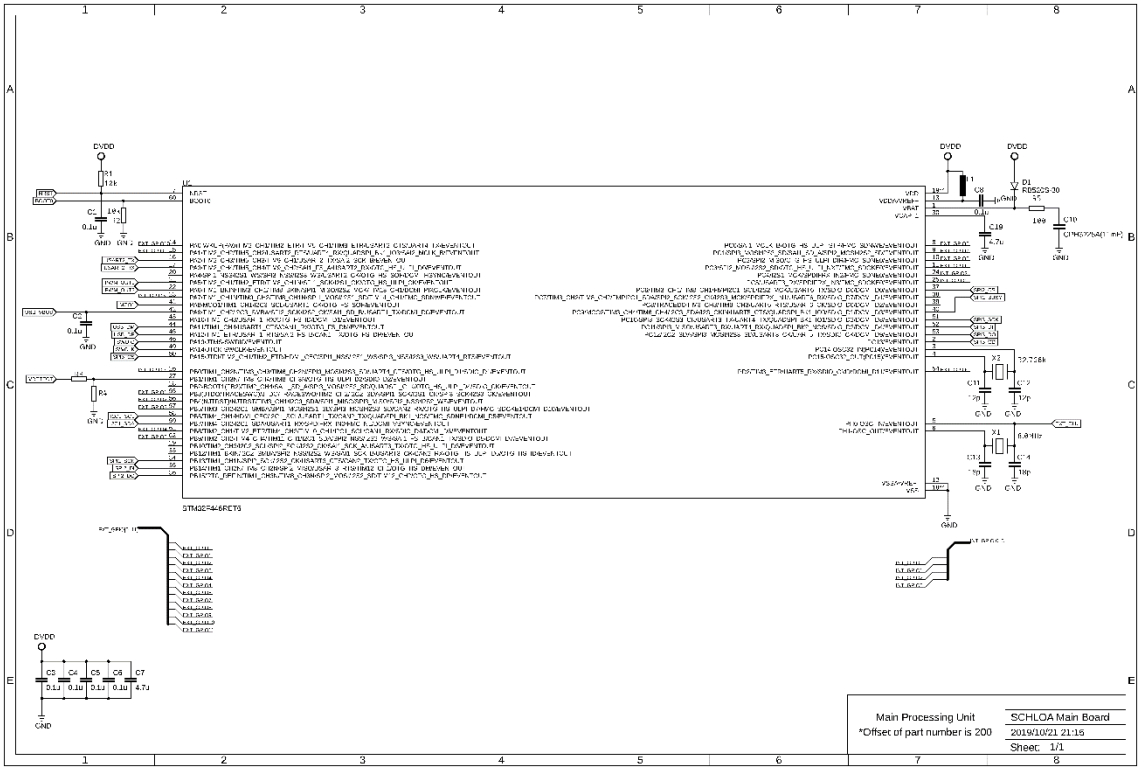

Fig. 8.3 Main Processing Unit Schematic

▪ Front End Processing Unit

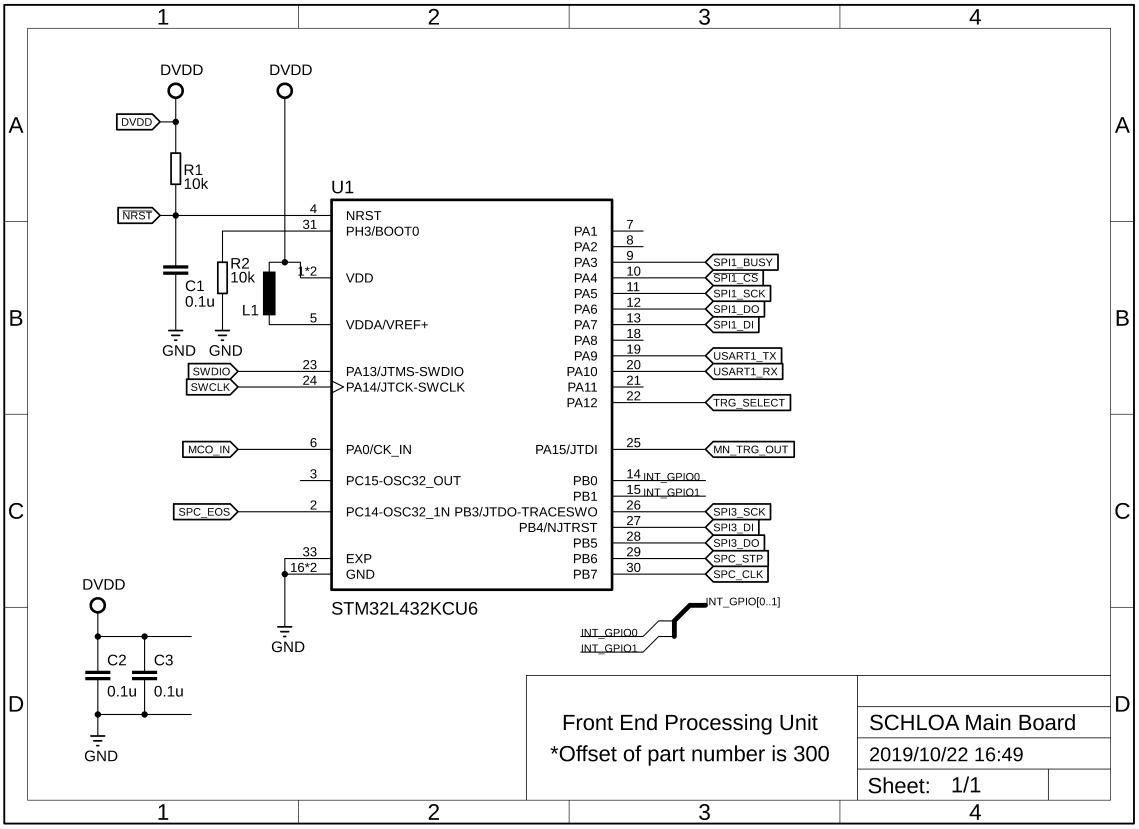

Fig. 8.4 Front End Processing Unit Schematic

▪ Spectrometer Unit

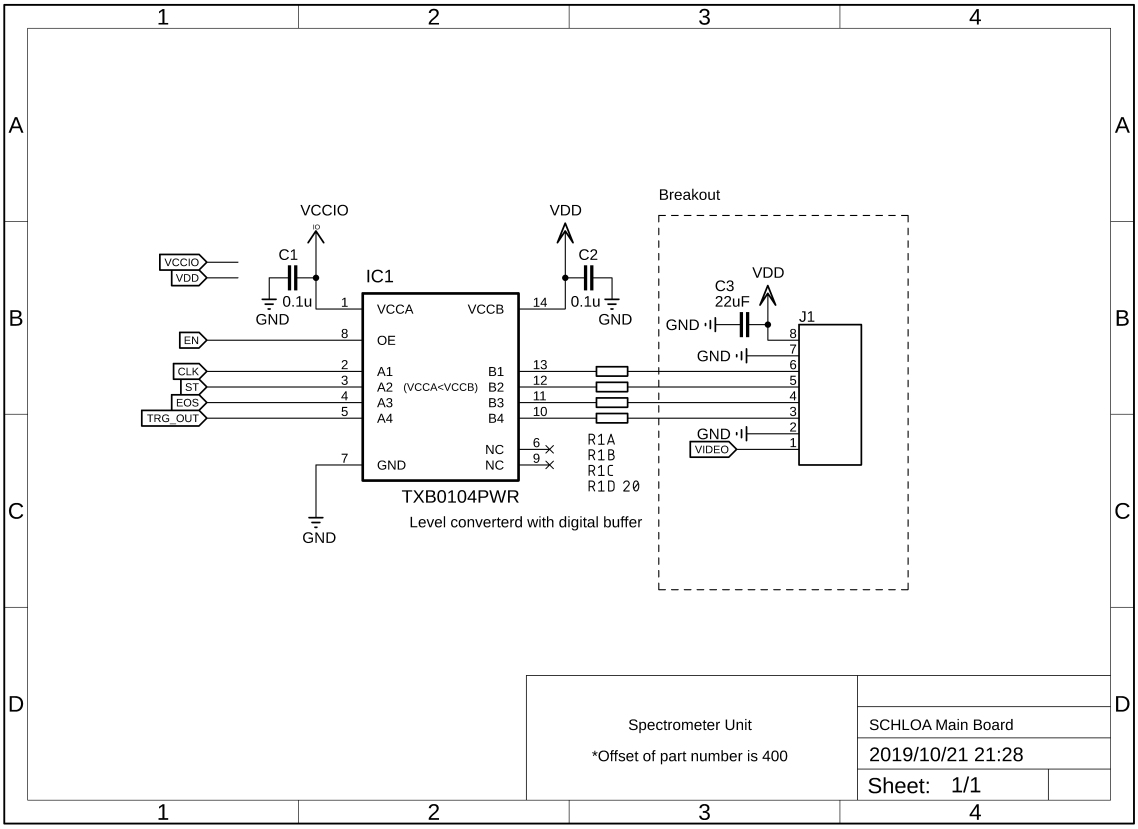

Fig. 8.6 Spectrometer Unit Schematic

▪ A/D Convert Unit

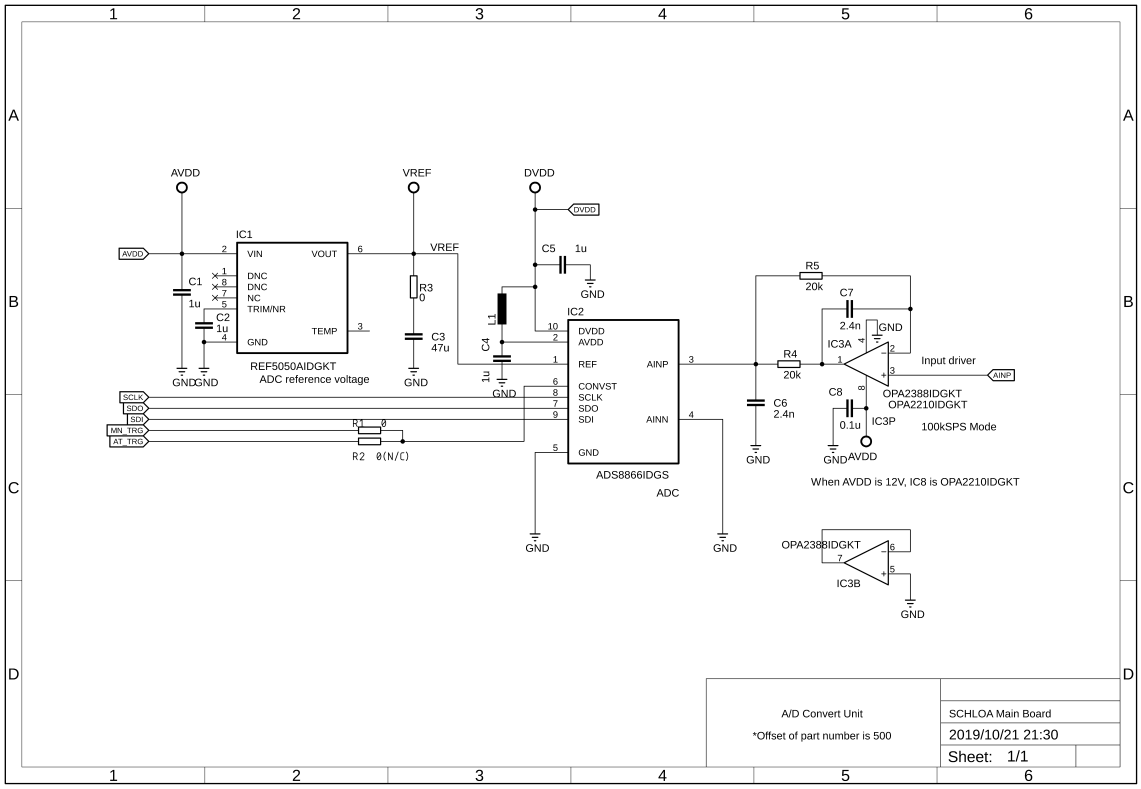

Fig. 8.7 A/D Convert Unit Schematic

▪ Breakeout Pins (External Access Port)

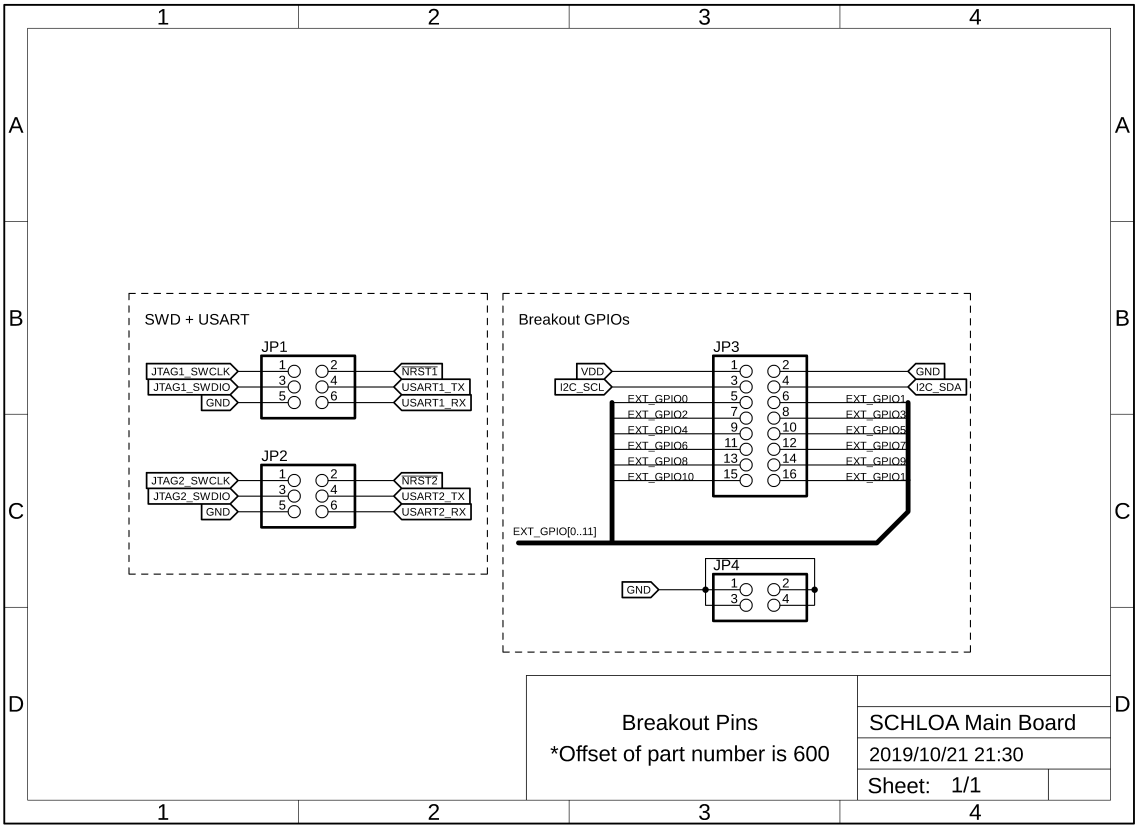

Fig. 8.8 Breakeout Pins Schematic

■ 内部ジャンクション

■ **Tabel 8.1 Internal Bus Junctions**

Net Symbol	Net Function	Junction	
		From	To
VDETECT	Voltage Detect	STM32F446RE PB1	Battery Power Bus
MPU_NRST#	Reset Signal Input	STM32F446RE NRST	S1, (6.NRST1)
MPU_DFU*	Boot Mode Select	STM32F446RE BOOT0	S2
MPU_SWCLK	SWD Clock Input	STM32F446RE PA14	(5.SWCLK1)
MPU_SWDIO	SWD Data I/O	STM32F446RE PA13	(7.SWDIO1)
MPU_TX	USART TX	STM32F446RE PA2	(8.TXD1)
MPU_RX	USART RX	STM32F446RE PA3	(10.RXD1)
USB_VBUS	USB Bus Input	STM32F446RE PA9	USB Power Bus
USB_DP	USB Data Pulse	STM32F446RE PA11	USB Data Bus
USB_DM	USB Data Minus	STM32F446RE PA12	USB Data Bus
SD_SPI_CS*	SDC SPI Select	STM32F446RE PC13	SDC CD
SD_SPI_SCLK#	SDC SPI Clock	STM32F446RE PC10	SDC CLK
SD_SPI_MOSI#	SDC SPI MOSI	STM32F446RE PC12	SDC DI
SD_SPI_MISO#	SDC SPI MISO	STM32F446RE PC11	SDC DO
EXT_GPIO0	External GPIO	STM32F446RE PD2	(21.GPIO0)
EXT_GPIO1	External GPIO	STM32F446RE PB3	(22.GPIO1)
EXT_GPIO2	External GPIO	STM32F446RE PB4	(23.GPIO2)
EXT_GPIO3	External GPIO	STM32F446RE PB5	(24.GPIO3)
EXT_GPIO4	External GPIO	STM32F446RE PB8	(25.GPIO4)
EXT_GPIO5	External GPIO	STM32F446RE PB9	(26.GPIO5)
EXT_GPIO6	External GPIO	STM32F446RE PC0	(27.GPIO6)
EXT_GPIO7	External GPIO	STM32F446RE PC1	(28.GPIO7)
EXT_GPIO8	External GPIO	STM32F446RE PC2	(29.GPIO8)
EXT_GPIO9	External GPIO	STM32F446RE PC3	(30.GPIO9)
EXT_GPIO10	External GPIO	STM32F446RE PA0	(31.GPIO10)
EXT_GPIO11	External GPIO	STM32F446RE PA1	(32.GPIO11)
MPU_ST_LED0	Status LED	STM32F446RE PB0	LED1
MPU_ST_LED1	Status LED	STM32F446RE PC5	LED2
MPU_ST_LED2	Status LED	STM32F446RE PC4	LED3
MPU_ST_LED3	Status LED	STM32F446RE PA7	LED4

MPU_SCL~	I2C Slave Clock	STM32F446RE PB6	(19.I2C_SCL)
MPU_SDA~	I2C Slave Data	STM32F446RE PB7	(20.I2C_SDA)
SOUND_OUT	Sound	STM32F446RE PA5	LS1
LED_PWM_OUT	LED Driver PWM	STM32F446RE PA6	TL4242 PWM
FEU_SPI_BUSY	FEU SPI Busy	STM32F446RE PC9	STM32L432KC PA3
EFU_SPI_CS*	FEU SPI Select	STM32F446RE PC6	STM32L432KC PA4
FEU_SPI_SCLK	FEU SPI Clock	STM32F446RE PB13	STM32L432KC PA5
FEU_SPI_MOSI	FEU SPI MOSI	STM32F446RE PB15	STM32L432KC PA7
FEU_SPI_MISO	FEU SPI MISO	STM32F446RE PB14	STM32L432KC PA6
FEU_MCLK	System Clock	STM32F446RE PA8	STM32L432KC PA0
FEU_NRST#	Reset Signal Input	STM32L432KC NRST	(12.NRST2)
FEU_SWCLK	SWD Clock Input	STM32L432KC PA14	(11.SWCLK2)
FEU_SWDIO	SWD Data I/O	STM32L432KC PA13	(13.SWDIO2)
FEU_TX	USART TX	STM32L432KC PA9	(14.TXD2)
FEU_RX	USART RX	STM32L432KC PA10	(16.RXD2)
ADC_ AT_TRG	ADC Auto Trigger	STM32L432KC PB5	ADS88xx CONVST
ADC _MN_TRG	ADC Manual Trigger	STM32L432KC PA15	ADS88xx CONVST
ADC_SPI_SCLK	ADC SPI Clock	STM32L432KC PB3	ADS88xx SCLK
ADC_SPI_MISO	ADC SPI MISO	STM32L432KC PB4	ADS88xx SDO
SPC_SCLK*	SPC Shift Clock	STM32L432KC PB7	(39.SPC_CLK_OUT)
SPC_ST*	SPC Start Pulse	STM32L432KC PB6	(40.SPC_ST_OUT)
SPC_EOS*	SPC End of Scan	STM32L432KC PC14	(41.SPC_EOS)
FEU_ST_LED0	Status LED	STM32L432KC PB0	LED5
FEU_ST_LED1	Status LED	STM32L432KC PB1	LED6

Note1 : チルダ(~) 表記されたネット・シンボルはオープン・ドレインです.

Note2 : アスタリスク(*) 表記されたネット・シンボル内部でプル・ダウンされています.

Note3 : シャープ(#) 表記されたネット・シンボルは内部でプル・アップされています.

Note4 : 小括弧で包まれた接続先は外部アクセスポートです.

9. 内部回路の動作と機能説明

本項で説明する殆どは．専用の API を用いることで無視することができます．API は汎用性が最大に考慮されているため，動作速度を意識したソフトウェア設計やより高効率化，良特性化を試みる際には熟読する必要があります．

■ パワーマルチプレクサの動作

電源マネジメントとして TEXAS INSTRUMENTS Auto-Switching Power Mux TPS2113PWR を実装しています．バッテリー使用時に USB ケーブルを接続し，PC あるいはスマートフォンと通信を行う際，自動的に電源供給を USB に変更します．

・電源供給の切り替え

パワーマルチプレクサは VSNS ピンに入力される電圧によって，USB 電源 (IN1 ピン) とバッテリー電源 (IN2 ピン) が切り替わります．IN2 選択時のみ STAT ピンに接続された LED101 が点灯します．閾値電圧は 0.8V です，また，EN ピンに接続されたスライドスイッチ SW101 を ON にすると電源供給を開始し，LED102 が点灯します．入出力の真理値表は次のようになります．

Tabel 9.1 Output Truth Table

SW101	$V_{I(VSNS)} > 0.8V$	$V_{I(IN2)} > V_{I(IN1)}$	LED101	LED102	OUT
ON	Yes	X	L	H	IN1
ON	No	No	L	H	IN1
ON	No	Yes	H	H	IN2
OFF	X	X	L	L	Hi-Z

Note1 : X = Don’t Carre, Hi-Z = High Impedance.

標準では USB 電源による供給が最優先されるようになっていますが，$V_{I(IN2)}$ と $V_{I(IN1)}$との比較によって供給元を選択したい場合は R101 を取り外してください．

・電流制限

最大出力電流は 1.25 A ですが，制限抵抗 R_{LIM}(R103)を変更することで出力制限 I_{LIM}[A]が変更可能です．R_{LIM} は以下の式で決定されます．

$$R_{LIM} = \frac{500}{I_{LIM}} \ [\Omega]$$

■　リセット回路の動作

・パワーオン時のリセットシーケンス

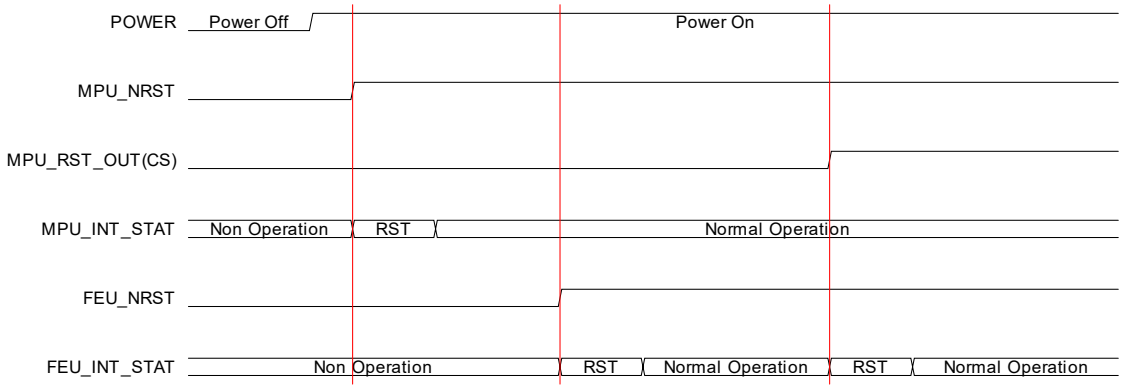

Fig. 9.1 Reset Sequence

各ユニットの NRST ピンはリセットシグナルを送らない限りはリセットされませんが，パワーオン時に限ってはその動作が異なります．NRST ピンには RC 回路による遅延回路が接続されており，パワーオン時の立ち上がりタイミングが遅延します．Fig. 9.2 にリセット回路の模式図を示します．

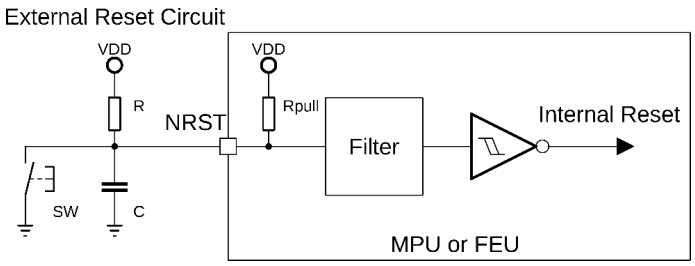

Fig. 9.2 Reset Circuit

NRST ピンのハイレベル・スレッショルド電圧V_{TH}は 2V であり，パワーオンから内部リセット信号が発生するまでの遅延時間t_dは次の式で算出できます．

$$t_d = \frac{CRR_{pull}}{R + R_{pull}} ln\left(\frac{V_{DD}}{V_{DD} - V_{TH}}\right) \text{ [s]}$$

ただし，R を実装しない場合は次のようになります．ただし，内部抵抗R_{pull}は 40 kΩです．

$$t_d = CR_{pull} ln\left(\frac{V_{DD}}{V_{DD} - V_{TH}}\right) \text{ [s]}$$

また，FEU は転送データの CONF Byte の 4 bit 目に 1 セットし，MPU から転送することでリセットします．詳しくは Tabel 9.5 を参照してください．

■ LED ドライバの動作

パワーLED ドライバとして，CC(Constant Current) Mode で動作する TEXAS INSTRUMENTS LED Driver TL4242 を実装しています．最大 500 mA での定電流ドライブが可能です．複数の LED を制御したい場合は専用のシールド基板を使用してください.

・出力電流の設定

リファレンス抵抗 R_{REF} (R14)を変更することで出力電流 I_Q [mA]を変更することができます．R_{REF} は以下の式で決定されます.

$$R_{REF} = \frac{V_{REF}}{I_Q}\ [\Omega]$$

V_{REF}は 0.177 mV と一定であり，R_{REF} は 0.39～10 Ωであることが求められます．例えばを R14 に 0.39Ωを実装した場合の I_Q は 454 mA になります.

・輝度調整

PWM 信号を入力することで輝度調整を行えます．PWM 信号の High 時間のみドライバが動作します．外部コンデンサ C_D(C11) [F]を実装したときの，立下り遅延時間 $T_{STH(LOW)}$ [ms]は C_Dに比例し，次の式で決定されます.

$$T_{STH(LOW)} = \frac{C_D}{C_{INT}} \times 10\ [\mathrm{ms}]$$

C_{INT} は 47 nF です.

・保護回路の動作

出力が開放及び短絡された際，極性を逆に接続した際，ドライバがオーバーヒートした際に保護回路が働き ST ピンの出力が Low に設定されます．この時の出力遷移時間は$T_{STH(LOW)}$ になります.

■ 音声出力機能

音声トランスデューサーとして PUI Audio Transducers SMT-0440-T-R を実装しています．メインコントロール・ユニットの PWM 出力ピンから信号を入力し，Nch MOS で構成されたローサイド・ドライバを介して出力します．共振周波数は 4000 ±500 Hz です．十分な音圧を得たい場合，この帯域での使用が推奨されます．矩形波以外の音声出力を行いたい場合は PWM 方式あるいは PDM 方式で音声を変調してください．変調周波数が可聴域のため，場合によっては RC ローパスフィルタを実装してください.

■　分光器の蓄積（露出）動作

分光器にはハードウェア的に互換が保てる Hamamatsu Photonics MS series 及び micro series が使用できます．以下は Hamamatsu Photonics micro series C12880MA を使用した場合になります．他の分光器を使用する場合は変換時の総クロック数が異なります．

・タイミングチャート

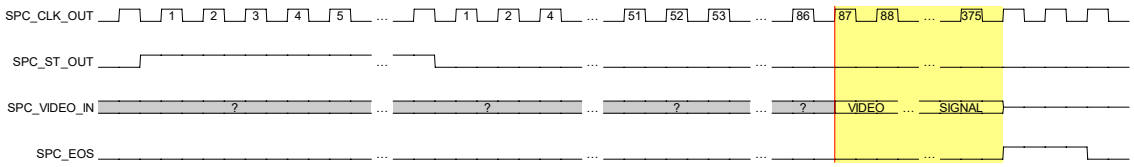

Fig. 9.3 Spectrometer Timing Chart

蓄積(露出)時間は，SPC_ST_OUT が立ち上がり，4 クロック目の SPC_CLK_OUT の立ち上がりから，SPC_ST_OUT の立ち下がりから数えて，SPC_CLK_OUT の 52 クロック目の立ち上がりまでの期間です．

・推奨動作範囲

Tabel 9.2 Recommended Operating Condition Ranges

Parameter		min	Typ	max	Unit
F_{CLK}	Clock Pulse Frequency	-	-	8.0	MHz
D_{CLK}	Clock Pulse Duty Ratio	45	50	55	%
T_{ST}	Start Pulse Cycle	381/ F_{CLK}	-	-	s
$T_{ST(High)}$	Start Pulse High Time	6/ F_{CLK}	-	-	s
$T_{ST(Low)}$	Start Pulse LowTime	375/ F_{CLK}	-	-	s

・最小蓄積（露出）時間の算出法

クロックパルス周波数 F_{CLK} を 500 kHz とした場合．

-スタートパルス High 時間 [$T_{ST(Low)}$]

$$T_{ST(High)} = 6/F_{CLK}$$
$$= 12\ \mu s$$

-最小蓄積時間 [$T_{ACM(min)}$]

$$T_{ACM(min)} = T_{ST(High)} + 48/F_{CLK}$$
$$= 108\ \mu s$$

■　ビデオ信号のキャプチャ動作

ビデオ信号のキャプチャには TEXAS INSTRUMENTS microPOWER™ ADS886x family に対応しています,

Tabel 9.2 Compatible Devices

Thoroughput	Device Names	Digikey PN
100kSPS	ADS8866IDGSR	296-39885-1-ND
400kSPS	ADS8864IDGSR	296-39881-1-ND
680kSPS	ADS8862IDGSR	296-39877-1-ND
1MSPS	ADS8860IDGSR	296-50909-1-ND

本ボードでは低消費電力化のため，AD コンバータのキャプチャ・レートを 100 µs に設定しています．キャプチャ・レートを変更する場合，あるいは低歪み化する場合はフロントエンド回路の RC キックバック・フィルタ回路を変更する必要があります．AD コンバータの入力信号はステップ信号であるため，設計はそれほど難しくありません．入力信号の周波数はキャプチャ・レートと等しくなります.

・トリガ・モード

変換開始トリガの作成モードは，SPI ペリフェラルを用いてハードウェア的にトリガ信号を作成するオートトリガ・モードとソフトウェア的にトリガ信号を作成するマニュアルトリガ・モードの 2 つがあります．どちらのモードを使用するにしても，フロントエンド・アンプの RC キックバック・フィルタ回路の帯域を考慮する必要がります.

-オートトリガ・モード

オートトリガ・モードはデジタルフロントエンド・ユニットの SPI ペリフェラルを利用し，トリガ信号を作成するトリガ・モードです．変換時間は自由に変更できませんが，キャプチャ時に自動的にトリガを作成するため，スムーズに変換結果を読み込むことができます.
このモードを有効にするには R502 を実装し，R501 を取り外す必要があります.

-マニュアルトリガ・モード

マニュアルトリガ・モードはデジタルフロントエンド・ユニットの GPIO ペリフェラルを利用し，任意のタイミングでトリガ信号を作成させるトリガ・モードです．任意のタイミングでトリガを作成することができ，変換時間を自由に設定できる反面，読み込みのタイミングを考慮してトリガを作成する必要があります.
このモードを有効にするには R501 を実装し，R502 を取り外す必要があります.

・タイミングチャート

-オートトリガ・モード

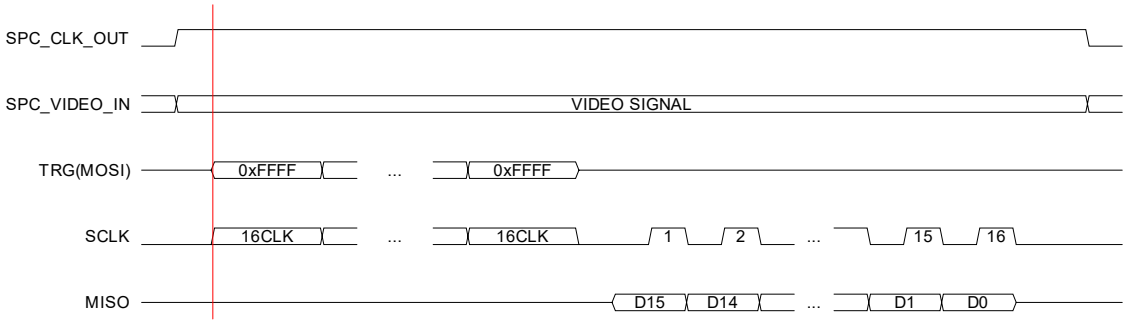

Fig. 9.4 Auto Trigger Mode Timing Chart

総転送時間が AD コンバータの最小変換時間から最大変換時間になるようデジタルフロントエンド・ユニットの MOSI から 0xFFFF をバースト転送してください.

-マニュアルトリガ・モード

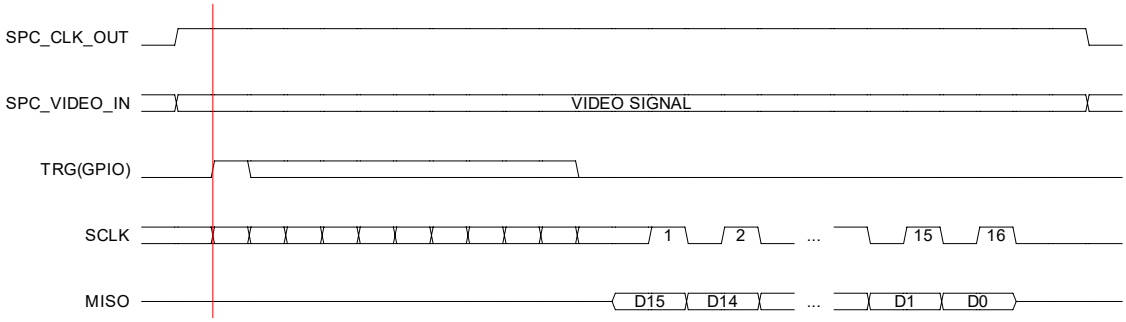

Fig. 9.5 Manual Trigger Mode Timing Chart

作成するトリガ信号の High 時間は AD コンバータの最小変換時間から最大変換時間に収まっている必要があります.

・AD コンバータのリファレンス電源

AD コンバータの基準電圧源には TEXAS INSTRUMENTS REF50xx に対応しています．分光器のビデオ信号のピーク値が 5.0V であるため，REF5050 を標準で実装していますが，基準電圧を変更したい場合には以下のデバイスを利用することができます．グレードにより許容誤差，温度係数，ノイズが異なります．

Tabel 9.3 Compatible Devices

Output Voltage	Device Names	Digikey PN
1.0V	REF5010	-
2.0V	REF5020	-
2.5V	REF5025	-
3.0V	REF5030	-
4.0V	REF5040	-
4.5V	REF5045	-
5.0V	REF5050	-

■　メインコントロール・ユニットとデジタルフロントエンド・ユニットの通信

マスターである MPU とスレーブである FEU は 4-wire SPI + Busy 検出で通信を行います.

・タイミングチャート

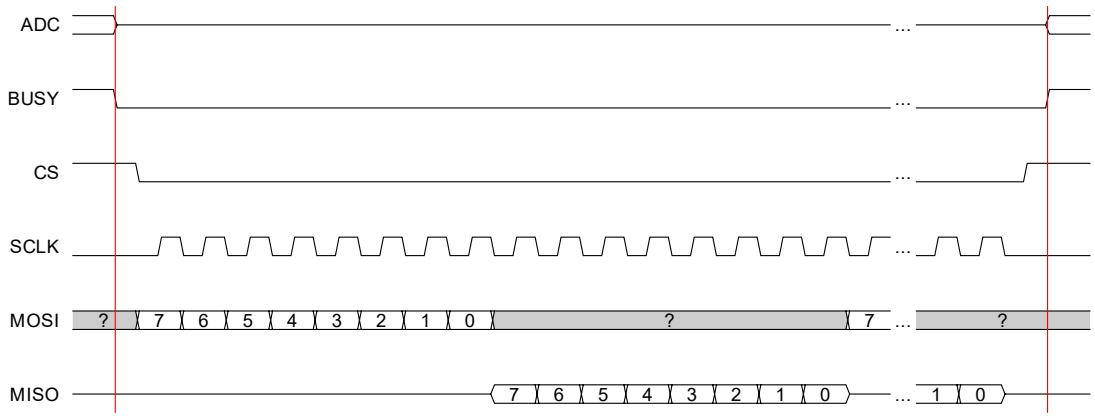

Fig. 9.6 4-wire SPI Communication Timing Chart

FEU が蓄積(露出)を行っている際, あるいは ADC 読み込みを行っている際 (BUSY 状態) は SPI 通信に対する応答が無効になります. FEU がフリーな状態では BUSY ピンの出力が Low になり，SPI 通信が有効になります．但し，DMA を使う際にはその限りではありません. SPI の動作モードはモード 0 (正極パルス，立ち上がりラッチ，立下りシフト)，MSB First です. 転送バイトオーダは基本リトルエンディアンで行い，データブロック長は 578 Byte です.

■　データフォーマット

-MPU→FEU の場合

データフォーマットは以下の様になります.

Tabel 9.4 MPU to FEU Transmit Data Formats

Byte	B0	B1	B2	B4	B5	B6	…	B576	B577
Data	CONF	ACM_TIME[3…0]				0x00	…	0x00	CSUM

CONF : Config Byte

Tabel 9.5 CONF Byte Data Formats

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Data	AEN	DEN	AMOD	RST	AVERAGEN[3…0]			
Default	0	0	0	0	0	0	0	0

AEN : Accumulation Enable

0 : Disable (default)

1 : Enable

DEN : Dark Correction Enable

0 : Disable (default)

1 : Enable

AMOD : Accumulation Mode

0 : Oneshot Mode (default)

1 : Continuous Mode

RST：FEU Reset

0 : Normal Operation (default)

1 : Reset

AVERAGEN[3…0] : Average Number (MSB Fisrt)

$$\overline{\text{pixel\_data}_i} = \frac{1}{N}\sum_{j=1}^{N} \text{pixel\_data}_{i,j}$$

0b0000 : N = 1 (default, NonAverage)

0b0001 : N = 2 ～ 0b111 : N = 9 ～ 0b1111 : N = 17

ACM_TIME[3…0] : Accumulation Time [μs] (32bit unsigned integer)

例) ACM_TIME = 114514 μs

ACM_TIME を 10 進数から 16 進数に変換する.

$$\text{ACM\_TIME} = (114514)_{10} = (44417\text{A9F})_{16}$$

バイトオーダをビッグエンディアンにする.

0x44	0x41	0x7A	0x9F

C_SUM : Check Sum (XOR)

-MPU←FEU の場合

Tabel 9.6 FEU to MPU Transmit Data Formats

Byte	B0	B1	…	B575	B576	B577
Data	PIX_DATA1[0…1]		…	PIX_DATA288[0…1]		CSUM

PIX_DATAx[0…1] : ADC Code for x th Pixel (16bit unsigned integer)

例) PIX_DATA16 = 810

PIX_DATA16 を 10 進数から 16 進数に変換する.

$$\text{PIX\_DATA16} = (810)_{10} = (32\text{A})_{16}$$

バイトオーダをリトルエンディアンにする.

0x2A	0x03

C_SUM : Check Sum (XOR)

・Check Sum (XOR)の算出法

送信データブロック長を N としたとき，1 Byte のデータ並びを d_{T1}, d_{T1}, … , d_{TN} とすると Check Sum $S_{TCH}(N)$ は 8bit の排他的論理和(XOR)による総和で表されます.

$$S_{TCH}(N) = d_{T1} \oplus d_{T2} \oplus \cdots \oplus d_{TN}$$

送信の際, $S_{TCH}(N)$をデータブロックの末尾に付加します. 受信の際, 受信データ$d_{R1}, d_{R2}, \cdots$, d_{RN} と末尾の $S_{TCH}(N)$との Check Sum $S_{RCH}(N+1)$を計算します.

$$S_{RCH}(N+1) = d_{R1} \oplus d_{R2} \oplus \cdots \oplus d_{RN} \oplus S_{TCH}(N)$$

この時，正常に送受信が行われていれば，受信データのみの Check Sum と$S_{TCH}(N)$は等しくなります.

$$S_{TCH}(N) = d_{R1} \oplus d_{R2} \oplus \cdots \oplus d_{RN}$$

自分自身との XOR をとると 0 となるので

$$S_{RCH}(N+1) = S_{TCH}(N) \oplus S_{TCH}(N)$$
$$= 0x00$$

となり， 0x00以外の値が得られた場合はエラーと判定できます.

Note1 : Check Sum はデータの完全性を保証するものではありません.

ファームウェア書き込みとシリアルデバッグ

・DAP または Writer を使用する場合

ARM 用の多くの DAP (Debug Adapter) が使用可能です. (j-Link, CMSIS-DAP, ST-Link など)，ボード上のデバッグポートと DAP を次のように接続してください.

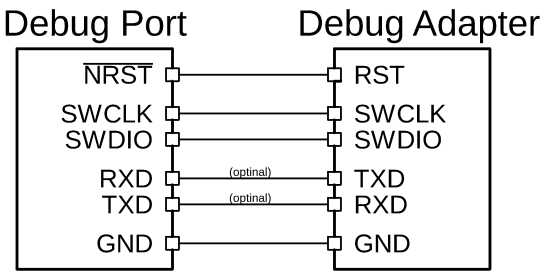

Fig. 9.7 DAP and Debug Port's Connection Diagram

ファームウェア書き込み時には必要ありませんが，シリアルポートが用意されているため簡易なデバッグ作業が行えます．本格的なデバッグの際には JTAG/SWD (SWCLK SWDIO) を使用してデバッグを行ってください.

・DFU Bootloader を使用する場合

MPU に使用している STM32F446RE には，DFU Bootloader がシステムメモリ領域に組み込まれています．そのため，専用の DAP や Writer を使用せずにファームウェアの書き込みが可能です．書き込み用のアプリケーションとして STMicroelectronics DfuSe あるいは CubeProg 等が必要です．Bootloader に入るには，ボード上の S2 を押した状態でパワーオンあるいはリセットシグナルを送る必要があります．USB ケーブルが検出されますと DFU Mode に入りファームウェアが書き込める状態になります．詳しくは AN2606 STM32 microcontroller system memory boot mode を参照してください.

10. ソフトウェアの機能説明

■　システムクロックの設定

Mbed OS を使用する場合，STM32F446RE ではシステムクロック周波数は 180MHz で動作します．しかし，ハードウェア的制限を受けるため，初期状態では HSI　(内部高速オシレータ)を用いてシステムクロックが作成されます．この時，システムクロック周波数は 160MHz に設定されるため．時間指定系の API では動作が 0.1%ほど遅れます．また，PLL の設定によっては誤差が大きくなります．これを防ぐために，ボード上の HSE XTAL (外部オシレータ) をクロックソースとして用いる必要があります．STM32L432KC の場合は初期では MSI (内部中速オシレータ)に設定されており HSE EXTC に設定します．

・JSON を変更し HSE XTAL の使用を有効にする

Mbed OS ではクロック周辺の設定を system_clock.c 上に記述しており，設定の指示は targets.json で行っています．以下は targets.json に記述されている，STM32F446RE の設定箇所です．

```
"NUCLEO_F446RE": {
        "inherits": ["FAMILY_STM32"],
        "supported_form_factors": ["ARDUINO", "MORPHO"],
        "core": "Cortex-M4F",
        "extra_labels_add": ["STM32F4", "STM32F446xE", "STM32F446RE"],
        "config": {
            "clock_source": {
                "help": "Mask value : USE_PLL_HSE_EXTC | USE_PLL_HSE_XTAL (need HW patch) |
                                                                        USE_PLL_HSI",
                "value": "USE_PLL_HSE_EXTC|USE_PLL_HSI",
                "macro_name": "CLOCK_SOURCE"
            }
        },
        "detect_code": ["0777"],
        "macros_add": ["USB_STM_HAL", "USBHOST_OTHER"],
        "device_has_add": [
            "ANALOGOUT",
            "CAN",
            "SERIAL_ASYNCH",
            "FLASH",
            "MPU"
        ],
        "release_versions": ["2", "5"],
                    "device_name": "STM32F446RE",
                    "bootloader_supported": true
 },
```

赤字表記箇所

`value": "USE_PLL_HSE_EXTC|USE_PLL_HSI"`

でクロックソースを指定しており，元の設定では ST-Link による MCO 入力(HSE)，あるいは HSI での動作になります．そのため当該箇所を

`"value": "USE_PLL_HSE_XTAL"`

に変更してください．　STM32L432KC の場合は該当箇所を

`value": "USE_PLL_HSE_EXTC"`

に変更します．

・クロック周辺の設定を自身で行い HSE XTAL **の使用を有効にする**

HAL を用いることで任意のシステムクロック周波数に変更することができますが．場合によっては Mbed OS との相互干渉により，システムが破城する可能性があるため，非推奨となります．PLL 係数等の設定は STMicroelectronics STM32CubeMX を用いることを推奨します，以下にシステムクロック周波数を 180MHz としたときの一例を示します.

```
int main (void) {

   /* Put user startup code here */

    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /* Configure the main internal regulator output voltage */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /* Initializes the CPU, AHB and APB busses clocks */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 180;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 2;
    RCC_OscInitStruct.PLL.PLLR = 2;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
        Error_Handler();
    }

    /* Activate the Over-Drive mode */
    if (HAL_PWREx_EnableOverDrive() != HAL_OK) {
        Error_Handler();
    }

    /* Initializes the CPU, AHB and APB busses clocks */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK) {
        Error_Handler();
    }

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOH_CLK_ENABLE();

   /* Put user startup code here */

   While(1) {

   /*Put user main code here */

   }
}
```

■ MCO の設定

FEU (STM32L432KC) は外部からの入力クロックを基に，システムクロックを作成します．そのため MPU (STM32F446RE) 側からのクロック出力機能 MCO (Microcontroller Clock Output) を有効にする必要があります．

出力は FEU_MCLK バスであり，MPU の PA8 から出力されます．詳しくは Fig. 8.3 と Table 8.1 を参照してください．

MCO の有効化は Mbed OS からは不可能であり，HAL を用いて行います．MPU が HSE 動作時は次の関数を追加してください．

```
void MCO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* Configure the MCO1 */
    HAL_RCC_MCOConfig(RCC_MCO1, RCC_MCO1SOURCE_HSE, RCC_MCODIV_1);

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOA_CLK_ENABLE();

    /*Configure GPIO pin : PA8 */
    GPIO_InitStruct.Pin = GPIO_PIN_8;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF0_MCO;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

HSI 動作時は

```
HAL_RCC_MCOConfig(RCC_MCO1, RCC_MCO1SOURCE_HSE, RCC_MCODIV_1)
```

と赤字表記された行を

```
HAL_RCC_MCOConfig(RCC_MCO1, RCC_MCO1SOURCE_HSI, RCC_MCODIV_2)
```

と変更してください．(内部オシレータが 16MHz の為) どちらの場合でも 8MHz のクロックパルスが PA8 から出力されます．HSI 動作時は，HSI の発振誤差と PLL の影響で 8±0.1MHz 程度になります．

Mbed **プラットフォームの変更箇所**

・C++11 **でプログラミングを行う**

Mbed では様々なコンパイラで動作させるために C++のバージョンが C++98 になっています．その為，ラムダ式等の機能を使用することができません．ここでは C++11 でコンパイルする方法について解説します．C++のバージョンを変更するには，{project_name}¥mbed-os¥tools¥profiles¥{profile_name}.jsom を書き換える必要があります．

例えば Build profile の一つ Relese では release.json を書き換えます．例えば ARMC6 コンパイラでは

```
"ARMC6": {
    "common": ["-c", "--target=arm-arm-none-eabi", "-mthumb", "-Oz",
                "-Wno-armcc-pragma-push-pop", "-Wno-armcc-pragma-anon-unions",
                "-DMULADDC_CANNOT_USE_R7", "-fdata-sections",
                "-fno-exceptions", "-MMD", "-D_LIBCPP_EXTERN_TEMPLATE(...)=",
                "-fshort-enums", "-fshort-wchar", "-DNDEBUG"],
    "asm": [],
    "c": ["-D__ASSERT_MSG", "-std=gnu99"],
    "cxx": ["-fno-rtti", "-std=gnu++98"],
    "ld": ["--show_full_path", "--legacyalign", "--keep=os_cb_sections"]
},
```

の赤字部

"cxx": ["-fno-rtti", "-std=gnu++98"]

から

"cxx": ["-fno-rtti", "-std=gnu++11"]

に書き換えます．

・SPI **関係の** API **の改良**

SPI 関係の API はバグにより，異なるチャネルであってもマイコン内で複数使用することができません（例えば SPI1 は Master で SPI2 では Slave の様な設定）．そのため次のようなメンバ関数を各クラスに追加してください．

-SPI Class

```
class SPI : private NonCopyable<SPI> {
public:
/* 省略 */
void free(void) {
    lock();
    spi_free(&_peripheral->spi);
    unlock();
}
/* 省略 */
};
```

-SPISlave Class

```
class SPISlave : private NonCopyable<SPISlave> {
public:
/* 省略 */
void free() {
    spi_free(&_spi);
}
/* 省略 */
};
```

追加した free()関数はオブジェクトで行った設定を全て開放する関数です．ピン設定や割り込みハンドラ等も開放されます．ただし，動的確保したメモリは開放されない為，メモリリークの危険性があることに注意してください．下に示すのは，修正した API を使ったサンプルプログラムです．

```
#include “mbed.h”
SPI *spi_master;
SPISlave *spi_slvae;

enum SPIMux { SPI_MASTER, SPI_SLAVE };
void spi_switch(SPIMux mux) {
  spi_master->free();
  spi_slave->free();

  switch (mux) {
  case SPI_MASTER:
    delete spi_master;
    spi_master = new SPISlave(SPI1_MOSI, SPI1_MISO, SPI1_SCLK, SPI1_CS);
    break;

  case SPI_SLAVE:
    delete spi_slave;
    spi_slave = new SPI(SPI2_MOSI, SPI2_MISO, SPI2_SCLK, SPI2_CS);
    break;

  default:
    break;
  }
}

Int main() {
  While(1) {
    spi_switch(SPI_MASTER);
    /* SPI Master の送受信処理 */

    spi_switch(SPI_SLAVE);
    /* SPI Slave の送受信処理 */
  }
}
```

■ **専用** API (schloa-api) **の概要**

より開発を容易にするため，Mbed 用 API パッケージ schloa-api を用意しました．使用する際はヘッダファイル schloa.h をインクルードし schloa_init をコールしてください

■ ADC API

ADC である ADS88xx ファミリを容易に動作される API です.

・ADC Class

Constructor & Destructor Documentation

ADC (SPI *_spi, PinName _conv)
指定された SPI マスターとコンバート出力ピンの設定を行います. Parameters:

_spi	SPI オブジェクトのポインタ
_cs	CS 出力ピン

~ADC ()
コンバート出力ピンを開放します.

Member Function Documentation

void setReference (float ref)
ADC に接続されているリファレンス電圧を指定します. Parameters:

ref	リファレンス電圧 [V]

uint16_t read ()
ワンショットトリガで A/D 変換を開始し，変換結果をバイナリデータで返します. Returns: Raw ADC Code (0x0000 ~ 0xFFFF)

uint16_t readVoltage ()
ワンショットトリガで A/D 変換を開始し，変換結果を電圧値で返します. Returns: Voltage [V]

operator uint16_t ()
ADC::read()の省略形

■ Backup API

バックアップ API は RTC のバックアップ EPPROM 領域を利用して，データのバックアップを行う API です

・Backup Class

Constructor & Destructor Documentation

Backup()
RTC の初期化を行い，バックアップ機能を有効にします.

Member Function Documentation

uint32_t read (uint32_t bkp_num)
バックアップメモリからデータを取得します. Note: 選択できる番地は RTC_BKP_DR0 が基準になります. Parameters: bkp_num：バックアップメモリの相対番地 Returns: 読み込んだ番地のデータ

void write(uint32_t bkp_num, uint32_t data)
バックアップメモリにデータを書き込みます Parameters: bkp_num：バックアップメモリの相対番地 data：書き込むデータ

■ Communication API

Communication API は MPU と FEU 間のデータ転送を行う API です．通信プロトコルについては Table9.4 ～ Table9.6 を参照してください. 主に使用するのは CommunicationMaster Class と CommunicationSlave Class になるでしょう．

・CommunicationMaster Class

CommunicationMaster Class は MPU が FEU と通信するための API です.

Constructor & Destructor Documentation

CommunicationMaster (SPI *_spi, PinName _cs, PinName _busy, MPU_TransDataType_t *rx_data)
指定された SPI マスターと使用する受信データのアドレスを設定します. Note: このコンストラクタは CommunicationMaster オブジェクトを使用して, CS ピンと BUSY ピンの GPIO 入出力を操作します. Parameters: _spi: SPI オブジェクトのポインタ _cs: CS 出力ピン _busy: BUSY 入力ピン rx_data: 受信データのポインタ

CommunicationMaster (SPI *_spi, PinName _cs, PinName _busy,)
指定された SPI マスターを設定します. Note: このコンストラクタは CommunicationMaster オブジェクトを使用して, CS ピンと BUSY ピンの GPIO 入出力を操作します. Parameters: _spi: SPI オブジェクトのポインタ _cs: CS 出力ピン _busy: BUSY 入力ピン

Member Function Documentation

bool transfer(MPU_TransDataType_t *rx_data)
データを転送し，受信が成功した場合は解析して書き出します. Note: 転送が終了するまで wait 処理が入ります. Parameters: rx_data: 受信データの構造体ポインタ Returns: 転送の成功の有無 true：成功， false：失敗

bool transfer()
データを転送し，受信が成功した場合は解析し，受信データをコンストラクタで設定した受信データのアドレスに代入します. Returns: 転送の成功の有無 true：成功， false：失敗

bool available ()
受信データが更新されたか確認します. returns 更新の有無 true：更新, false：未更新

void attachTransferHandler (Callback<void()> handle)
転送完了の際に呼び出されるハンドラをアタッチします. Parameters: handle　コールされる関数のポインタ

void attachTransferHandler (T *obj, M method)
転送完了の際に呼び出されるハンドラをアタッチします.

Parameters:

obj	メンバ関数がコールされるオブジェクトのポインタ
method	コールされるメンバ関数のポインタ

void attachTransferErrorHandler (Callback<void()> handle)
転送失敗の際に呼び出されるハンドラをアタッチします.

Parameters:

handle	コールされる関数のポインタ

void attachTransferErrorHandler (T *obj, M method)
転送失敗の際に呼び出されるハンドラをアタッチします.

Parameters:

obj	メンバ関数がコールされるオブジェクトのポインタ
method	コールされるメンバ関数のポインタ

void detachTransferHandler ()
転送完了の際に呼び出されるハンドラをデタッチします.

void detachTransferErrorHandler ()
転送失敗の際に呼び出されるハンドラをデタッチします.

・CommunicationSlave Class

CommunicationSlave Class は FEU が MPU と通信するための API です.

Constructor & Destructor Documentation

CommunicationMaster (SPISlave *_spi, PinName _busy, FEU_TransDataType_t *rx_data)
指定された SPI マスターと使用する受信データのアドレスを設定します. Note: このコンストラクタは CommunicationSlave オブジェクトを使用して，CS ピンと BUSY ピンの GPIO 入出力を操作します. Parameters: _spi: SPI オブジェクトのポインタ _busy: BUSY 入力ピン rx_data: 受信データのポインタ

CommunicationMaster (SPISlave *_spi, PinName _busy,)
指定された SPI マスターを設定します. Note: このコンストラクタは CommunicationSlave オブジェクトを使用して，CS ピンと BUSY ピンの GPIO 入出力を操作します. Parameters: _spi: SPI オブジェクトのポインタ _busy: BUSY 入力ピン

Member Function Documentation

bool transfer (FEU_TransDataType_t *rx_data)
データを転送し，受信が成功した場合は解析して書き出します. Note: 転送が終了するまで wait 処理が入ります. Parameters: rx_data | 受信データの構造体ポインタ Returns: 転送の成功の有無 true：成功， false：失敗

bool transfer ()
データを転送し，受信が成功した場合は，解析し，受信データをコンストラクタで設定した受信データのアドレスに代入します. Note: 転送が終了するまで wait 処理が入ります. Returns: 転送の成功の有無 true：成功， false：失敗

bool available ()
受信データが更新されたか確認します. returns 更新の有無 true：更新， false：未更新

void attachTransferHandler (Callback<void()> handle)
転送完了の際に呼び出されるハンドラをアタッチします. Parameters: handle | コールされる関数のポインタ

void attachTransferHandler (T *obj, M method)
転送完了の際に呼び出されるハンドラをアタッチします.

Parameters:

obj	メンバ関数がコールされるオブジェクトのポインタ
method	コールされるメンバ関数のポインタ

void attachTransferErrorHandler (Callback<void()> handle)
転送失敗の際に呼び出されるハンドラをアタッチします.

Parameters:

handle	コールされる関数のポインタ

void attachTransferErrorHandler (T *obj, M method)
転送失敗の際に呼び出されるハンドラをアタッチします.

Parameters:

obj	メンバ関数がコールされるオブジェクトのポインタ
method	コールされるメンバ関数のポインタ

void detachTransferHandler ()
転送完了の際に呼び出されるハンドラをデタッチします.

void detachTransferErrorHandler ()
転送失敗の際に呼び出されるハンドラをデタッチします.

・MPUToFEUFitting Class

MPUToFEUFitting Class は FEU 用のデータフォーマットを基に，送信データの作成，そして，受信データの解析を行う API です.

Constructor & Destructor Documentation

MPUToFEUFitting (FEU_TransDataType_t *data)
データ解析結果を代入する構造体ポインタを指定します. Parameters: data　解析結果を代入する構造体ポインタ

MPUToFEUFitting ()
データフィッティングの際に使用するバッファを初期化します.

Member Function Documentation

void fitting (uint8_t *msg)
設定値をメッセージとしてフィッティングし書き出します. Parameters: msg　書き出されるメッセージの配列ポインタ

void fitting ()
設定値をメッセージとしてフィッティングします.

void setAccumulationBit (bool en)
Config Byte の AccumulationBit の設定を行います. Parameters: en　true:有効, false:無効

void accumulationEnable ()
Config Byte の AccumulationBit を有効に設定します.

void accumulationDisable ()
Config Byte の AccumulationBit を無効に設定します.

bool isAccumulationEnable (FEU_TransDataType_t *data)
受信データの構造体から AccumulationBit のみを取り出します.

Parameters:

data	受信データの構造体ポインタ

bool isAccumulationEnable ()
コンストラクタで設定した受信データから AccumulationBit のみを取り出します.

Returns::

AccumulationBit true:有効 false:無効

void setDrakCorrectionBit (bool en)
Config Byte の DrakCorrectionBit の設定を行います.

Parameters:

en	true:有効, false:無効

void drakCorrectionEnable ()
Config Byte の DrakCorrectionBit を有効に設定します.

void drakCorrectionDisable ()
Config Byte の DrakCorrectionBit を無効に設定します.

bool isDrakCorrectionEnable (FEU_TransDataType_t *data)
受信データの構造体から DrakCorrectionBit のみを取り出します.

Parameters:

data	受信データの構造体ポインタ

bool isDrakCorrectionEnable ()
コンストラクタで設定した受信データから DrakCorrectionBit のみを取り出します.

Returns::

DrakCorrectionBit true:有効 false:無効

void setAccumulationModeBit (AccumulationMode mode)
Config Byte の AccumulationModeBit の設定を行います. Parameters: mode: 蓄積モード

void oneshotMode ()
Config Byte の AccumulationModeBit を Oneshot に設定します.

void continuousMode ()
Config Byte の AccumulationModeBit を Continuous に設定します.

AccumulationMode getAccumulationMode (FEU_TransDataType_t *data)
受信データの構造体から AccumulationModeBit のみを取り出します. Parameters: data: 受信データの構造体ポインタ Returns:: AccumulationModeBit

AccumulationMode getAccumulationMode ()
コンストラクタで設定した受信データから AccumulationModeBit のみを取り出します Returns: AccumulationModeBit

void setResetBit (OperationMode mode)
Config Byte の ResetBitt の設定を行います. Parameters: mode: 動作モード

void resetOperationMode ()
Config Byte の ResetBitt を ResetOperation に設定します.

void normalOperationMode ()
Config Byte の ResetBit を NormalOperation に設定します.

OperationMode getOperationMode (FEU_TransDataType_t *data)
受信データの構造体から ResetBit のみを取り出します. Parameters: data: 受信データの構造体ポインタ Returns:: ResetBit

OperationMode getOperationMode ()
コンストラクタで設定した受信データから ResetBit のみを取り出します Returns: ResetBit

void setAverageNumber (uint8_t num)
算術平均処理する回数を設定します. Note: 0 を指定すると算術平均処理を行いません. Parameters: num: 平均回数

uint8_t getAverageNumber(FEU_TransDataType_t *data)
受信データの構造体から平均回数のみを取り出します. Parameters: data: 受信データの構造体ポインタ Returns:: 平均回数

uint8_t getAverageNumber()
コンストラクタで設定した受信データから平均回数のみを取り出します. Returns: 平均回数

void setConfigByte (FEU_ConfigDataType_t conf)
Config Byte を書き込みます. Parameters: conf: ConfigByte の構造体

void setConfigByte();
Config Byte を書き込みます. Note: ConfigByte の各 bit の設定後にコールしてください．コールしない場合，設定が反映されずにメッセージが書き出されます

FEU_ConfigDataType_t getConfigByte (FEU_TransDataType_t *data)

受信データの構造体から ConfigByte のみを取り出します.

Parameters:

data	受信データの構造体ポインタ

Returns::

ConfigByte

FEU_ConfigDataType_t getConfigByte()

コンストラクタで設定した受信データから ConfigByte のみを取り出します

Returns:

ConfigByte

void setAccumulationTime (uint32_t us)

蓄積時間を設定します.

Parameters:

us	蓄積時間[us]

uint32_t getAccumulationTime(FEU_TransDataType_t *data)

受信データの構造体から蓄積時間のみを取り出します.

Parameters:

data	受信データの構造体ポインタ

Returns::

蓄積時間[us]

uint32_t getAccumulationTime ()

コンストラクタで設定した受信データから蓄積時間のみを取り出します.

Returns:

蓄積時間[us]

void setTransData(FEU_TransDataType_t *data)
構造体を代入し全ての設定を行います. Parameters: data　送信データの構造体ポインタ

bool messageAnalyze (uint8_t *msg, FEU_TransDataType_t *data)
チェックサムを調べ，メッセージが有効であれば，メッセージを解析し，解析したデータを書き出します. Parameters: msg　受信したメッセージの配列ポインタ data　書き出される解析結果の構造体ポインタ Successful reception (true : success , false : failure)

・FEUToMPUFitting Class

FEUToMPUFitting Class は MPU 用のデータフォーマットを基に，送信データの作成，そして，受信データの解析を行う API です.

Constructor & Destructor Documentation

FEUToMPUFitting (MPU_TransDataType_t *data)
データ解析結果を代入する構造体ポインタを指定します. Parameters: data　解析結果を代入する構造体ポインタ

FEUToMPUFitting ()
データフィッティングの際に使用するバッファを初期化します.

Member Function Documentation

void fitting (uint8_t *msg)
設定値をメッセージとしてフィッティングし書き出します. Parameters: msg　書き出されるメッセージの配列ポインタ

void fitting ()
設定値をメッセージとしてフィッティングします.

void setPixData (uint16_t *data)
送信するピクセルデータを設定します. Note： 値の代入は行わず，先頭アドレスのみを登録します．そのため fitting をコールするまでピクセルデータの変更は行わないでください. Parameters: data　ピクセルデータの配列ポインタ

void getPixData (MPU_TransDataType_t *data, uint16_t *pix_buf)
受信データの構造体からピクセルデータのみを取り出します. Parameters: data　受信データの構造体ポインタ pix_buf　ピクセルデータの配列ポインタ

void getPixData (uint16_t *pix_buf)
コンストラクタで設定した受信データからピクセルデータのみを取り出します. Parameters: pix_buf　ピクセルデータの配列ポインタ

void setTransData(MPU_TransDataType_t *data)
構造体を代入し全ての設定を行います. Parameters: data　送信データの構造体ポインタ

bool messageAnalyze(uint8_t *msg, MPU_TransDataType_t *data)
チェックサムを調べ，メッセージが有効であれば，メッセージを解析し，解析したデータを書き出します. Parameters: msg　受信したメッセージの配列ポインタ data　書き出される解析結果の構造体ポインタ Returns: Successful reception (true : success , false : failure)

・Fitting Class

Fitting Class はデータフィッティングの際に使用するバッファの管理を容易にする API です.
Template Class の為，型指定が必要です.

Constructor & Destructor Documentation

template<class T> Fitting(uint32_t msg_size)
指定したデータ長 (型依存) の T 型の動的配列を作成します. Parameters: msg_size: 配列の要素数

template<class T> ~Fitting()
確保したメモリを開放します. Parameters: msg_size: 配列の要素数

Member Function Documentation

template<class T> T checkSum(T *msg, uint32_t length)
指定した要素までの Check Sum を計算します Parameters: msg: チェックする配列ポインタ length: チェックする要素数 Returns Check Sum (0:Non Error, other:Error)

template<class T> T checkSum()
作成した動的配列の最後の要素を抜いた Check Sum を計算します. Returns Check Sum (0:Non Error, Other:Error)

template<class T> uint32_t size()
作成した動的配列の要素数を返します. Note: 確保されるバイト長は T 型のバイト長を乗算する必要があります. Returns Byte Length

template<class T> T front()
作成した動的配列の先頭要素を参照します. Returns 先頭要素

template<class T> T back()
作成した動的配列の最後尾要素を参照します. Returns 最後尾要素

template<class T> T *data()
作成した動的配列の先頭要素のポインタを参照します. Returns 先頭要素のポインタ

template<class T> T & operator [](uint32_t n)

作成した動的配列の指定した要素を参照します．(オペレータのオーバーロード)
 	
Exsample Code:
`int main() {`
`  Fitting<int> obj(3);`
`  obj[2] = 114514;`
`  printf(“obj[2] = %d¥n¥r” obj[2]); //Result : obj[3] = 114514`
`}`

■ DebugPort API

Cortex コアが持つ ITM 機能によって printf 形式のデバッグを提供する API です．USART を用いたデバッグより高速に行えます.

・DebugPort Class

Constructor & Destructor Documentation

DebugPort(const char *name)
ITM デバッグを有効にします. Note: JTAG デバッガとマイコンの SWCLK SWDIO SWO を接続する必要があります. Parameters: name: ポート名

Member Function Documentation

int putc(int c)
文字出力を行います. Parameters: c: 文字 Returns: エラー

int printf(const char* format, ...)
フォーマットに従い，文字列を出力します. Note: JTAG デバッガとマイコンの SWCLK SWDIO SWO を接続する必要があります. Parameters: printf 準拠 Returns: 出力した文字数

■ DrawSpectrum API

専用 GUI DrawSpectrum との相互通信機能を提供する API です.

・DrawSpectrum Class

Constructor & Destructor Documentation

DrawSpectrum(USBSerial *obj, DrawSpectrum_TransDataType_t *rx_data)
初期設定を行います. Note: Mbed OS2 版の USBSerial API が必要になります. Parameters: obj: USBSerial のオブジェクトポインタ rx_data: 受信データの構造体ポインタ

DrawSpectrum(USBSerial *obj)
初期設定を行います. 受信バッファは動的確保されます. Note: Mbed OS2 版の USBSerial API が必要になります. Parameters: obj: USBSerial のオブジェクトポインタ

Member Function Documentation

void send(DrawSpectrum_TransDataType_t *tx_data)
再設定しデータをパケット転送します. Parameters: tx_data: 出力データの構造体ポインタ

void send()
設定済みのデータをパケット転送します

DrawSpectrum_DataStatusType read(DrawSpectrum_TransDataType_t *rx_data)
受信したパケットデータを解析し，各データに分別します. Parameters: rx_data: 受信データの構造体ポインタ Returns: データステータス

DrawSpectrum_DataStatusType read(DrawSpectrum_TransDataType_t *rx_data)
受信したパケットデータを解析し，各データに分別します. 受信データはコンストラクタで指定したアドレスへコピーされます. Returns: データステータス

void attachPacketReceiveHandler(Callback<void()> handle)
データパケット受信時にコールされるハンドラを指定します. Parameters: handle: コールされる関数のポインタ

template<typename T, typename M> void attachPacketReceiveHandler(T *obj, M method)
データパケット受信時にコールされるハンドラを指定します. Parameters: obj: メンバ関数がコールされるオブジェクトのポインタ method: コールされるメンバ関数のポインタ

・DrawSpectrumAnalyze Class

Constructor & Destructor Documentation

DrawSpectrumAnalyze(DrawSpectrum_TransDataType_t *rx_data)
初期設定を行い，バッファを初期化します.

Parameters:

rx_data	受信データの構造体ポインタ

DrawSpectrumAnalyze()
初期設定を行い，バッファを初期化します.

Member Function Documentation

殆どが Communication API と同じ形をとっています．以下は追加されたメンバです.

void setPixDataOffset(uint16_t *data);
ピクセルデータのオフセット値を設定します.

Parameters:

data	オフセットデータ配列のポインタ

virtual uint32_t pixDataNormalizeation(uint32_t ch);
ピクセルデータを正規化後，転送データに変換します.

Note:

正規化処理はダーク減算後，指定ピクセルでの理論上の最大感度に対する相対値の百分率に変換します．一般的な相対感度とは異なることに注意してください.

転送データは正規化値の 100 倍の値になります.

$$\widetilde{D} = \frac{D \times 10000}{ADC\_MAX\_CODE - OFFSET}$$

Parameters:

data	オフセットデータ配列のポインタ

void getLEDBrightness(DrawSpectrum_TransDataType_t *data, float *brightness)
受信データから各 LED の明度情報を取得します.

Parameters:

data	受信データの構造体ポインタ
brightness	LED のチャネル分の明度データのポインタ

void getLEDBrightness(float *brightness)

受信データから各 LED の明度情報を取得します.

Parameters:

brightness	LED のチャネル分の明度データのポインタ

std::time_t getTime(DrawSpectrum_TransDataType_t *data)

受信データから時刻情報を取得します.

Parameters:

data	受信データの構造体ポインタ

Returns:

UNIX 時間

std::time_t getTime()

受信データから時刻情報を取得します.

Returns:

UNIX 時間

■ FastOut API

FastOut Class は直接的にレジスタを操作することで GPIO の出力の高速化を図った API です．ハードウェアの依存性が高いため汎用的な使用ができません.

・FastOut Class

Constructor & Destructor Documentation

FastOut (PinName _pin)
指定した GPIO ピンの初期化を行い，出力ポートに設定します. Parameters: _pin　出力ピン

~FastOut()
出力に設定したピンを開放します

Member Function Documentation

void write (uint32_t value)
GPIO ピンの出力値を設定します. Parameters: value　0 : Low, Other : High

void toggle ()
GPIO ピンの出力値をトグルします.

uint32_t read ()
GPIO ピンの出力状態を返します. Returns: Pin Status 0 : Low, Other : High

FastOut &operator= (uint32_t value)

FastOut::write()の省略形
 	
Ex)
`Int main() {`
`  FastOut out (LED1);`
`  out = 1; //LED1 High`
`}`

FastOut &operator= (FastOut& rhs)
オブジェクトから出力状態をコピーし出力状態を変更する FastOut::write()の省略形

operator uint32_t()
FastOut::read ()の省略形

■ LEDDrive API

LED 制御用の API です．PWM によって調光します.

・LEDDrive Class

Constructor & Destructor Documentation

LEDDrive(uint32_t pin_num, ...)
出力ピンの設定を行います. Parameters: pin_num：出力ピンの総数 …：(PinName 型)出力ピン番号(可変長)

virtual ~LEDDrive()
出力ピンを開放します.

Member Function Documentation

uint32_t size()
指定した出力ピンの総数を取得します. Returns: ピンの総数

void write(uint32_t pin_num, float val)
指定したピンの Duty 比を設定します Parameters: pin_num：ピン番号 val：Duty 比
void write(float *val)
全てのピンの Duty 比を設定します Parameters: val：Duty 比配列のポインタ

void setFrequency(uint32_t pin_num, float frequency)

指定したピンの周波数を変更します.

Parameters:

pin_num	ピン番号
frequency	周波数

void setFrequency(float *frequency)

全てのピンの周波数を変更します.

Parameters:

frequency	周波数配列のポインタ

float read(uint32_t pin_num)

指定したピンの Duty 比を取得します.

Parameters:

pin_num	ピン番号

float & operator [](int n)

LEDDrive::write と LEDDrive::read の省略形

Parameters:

n	ピン番号

Ex)

```
Int main() {
    LEDDrive out (2, PB_0, PB 1);
    Out[1] = Out[0]; //out.write(1, out.read(0));
}
```

■ LiquidCrystal API

printf 形式で LCD に文字列表示機能を提供する API です．ST7032 互換のドライバ IC にのみ対応します.

・LiquidCrystal Class

Constructor & Destructor Documentation

LiquidCrystal(I2C *i2c, uint8_t addr)
指定された I2C マスターと使用するスレーブアドレスを設定します. Parameters: i2c：I2C のオブジェクトポインタ addr：スレーブアドレス

Member Function Documentation

void init()
LCD の初期化を行います.

void clear()
文字の消去を行います.

void home()
カーソルをホーム地点に移動させます.

void setCursor(uint32_t col, uint32_t line)
カーソルを指定の位置に移動させます. Parameters: col：移動する列 line：移動する行

void shift(int32_t shift)
指定した文字分表示を左右にシフトさせます. Parameters: shift：シフトさせる列数 (+：右 -：左)

void setContrast(uint8_t val)
コントラストを設定します Parameters: val: コントラスト(7bit)

int putc(int c)
カーソル地点に文字出力を行います. Parameters: c: 文字 Returns: エラー

int printf(const char* format, ...)
フォーマットに従い，カーソル地点に文字列を出力します. Parameters: printf 準拠 Returns: 出力した文字数

■ SoftPWMOut API

ソフトウェアによって PWM 機能を提供する API です．ハードウェアに依存せず PWM 出力が可能です.

・SoftPWMOut Class

Constructor & Destructor Documentation

SoftPWMOut(PinName pin)
出力ピンの設定を行います. Parameters: pin：ピン番号

Member Function Documentation

void start()
タイマ割り込みを開始し，出力を有効にします.

void free()
タイマ割り込みを無効にし，ピンを開放します.

void write(float value)
Duty 比を設定します. Parameters: value：Duty 比

float read()
現在の Duty 比を取得します. Returns: Duty 比

void period(float seconds)
パルス周期を設定します Parameters: seconds | パルス周期[s]

void period(float ms)
パルス周期を ms 単位で設定します Parameters: ms | パルス周期[ms]

void period(float us)
パルス周期を μs 単位で設定します Parameters: us | パルス周期[μs]

void pulsewidth(float seconds)
パルス幅を設定します. Parameters: seconds | パルス幅[s]

void pulsewidth(float ms)
パルス幅を ms 単位で設定します. Parameters: ms | パルス幅[ms]

void pulsewidth(float us)
パルス幅を μs 単位で設定します. Parameters: us | パルス幅[μs]

SoftPWMOut &operator=(float value)

SoftPWMOut::write の省略形

Parameters:

value	Duty 比

Ex)

```
Int main() {
    SoftPWMOut out (LED1);
    out.start();
    out = 0.5f; //out.write(0.5f);
}
```

SoftPWMOut &operator=(SoftPWMOut &rhs)

SoftPWMOut::write の省略形

Parameters:

rhs	SoftPWMOut のオブジェクトポインタ

■ SystemTicker API

SystemTicker API は System Timer を利用してタイマ割り込みを発生される API です，
System Timer を使用する都合上，時間計の API(wait, timer 等)は使用不可になります.

・System Ticker Class

Constructor & Destructor Documentation

SystemTicker()
カウンタを初期化します

~SystemTicker()
タイマを開放し，割り込みハンドラをデタッチします.

Member Function Documentation

void attach (Callback<void()> handle, float s)
タイマ割り込みの際にコールされるハンドラをアッタッチし，インターバル時間を[s]単位で指定します. Parameters:

handle	コールされる関数ポインタ
s	インターバル時間[s]

void attach (T obj, M *method, float s)
タイマ割り込みの際にコールされるハンドラをアッタッチし，インターバル時間を[s]単位で指定します. Parameters:

obj	メンバ関数がコールされるオブジェクトのポインタ
method	コールされるメンバ関数のポインタ
s	インターバル時間

void attach_ms (Callback<void()> handle, float ms)
タイマ割り込みの際にコールされるハンドラをアッタッチし, インターバル時間を[ms]単位で指定します.

Parameters:

handle	コールされる関数ポインタ
ms	インターバル時間[s]

void attach_ms (T obj, M *method, float ms)
タイマ割り込みの際にコールされるハンドラをアッタッチし, インターバル時間を[ms]単位で指定します.

Parameters:

obj	メンバ関数がコールされるオブジェクトのポインタ
method	コールされるメンバ関数のポインタ
m**s**	インターバル時間

void attach_us (Callback<void()> handle, float us)
タイマ割り込みの際にコールされるハンドラをアッタッチし，インターバル時間を[us]単位で指定します. .

Parameters:

handle	コールされる関数ポインタ
ms	インターバル時間[us]

void attach_us (T obj, M *method, float us)
タイマ割り込みの際にコールされるハンドラをアッタッチし，インターバル時間を[us]単位で指定します.

Parameters:

obj	メンバ関数がコールされるオブジェクトのポインタ
method	コールされるメンバ関数のポインタ
m**s**	インターバル時間[us]

void detach()
タイマ割り込みの割り込みハンドラをデタッチします.

static void init (uint32_t cnt)
タイマの最大カウント数を設定し，タイマを初期化します. Parameters: cnt　タイマカウンタの最大値

Temperature API

温度センサの制御を提供する API です．ADT7410 にのみ対応します.

■ Sound API

音声出力機能を提供する API です．現在は矩形信号にのみ対応します．

・Sound Class

Constructor & Destructor Documentation

Sound(PinName _pin)
音声出力ピンの初期化を行います． Note: SoftPWMOut API により音声作成を行うため，ピンによる制限がありません． Parameters: _pin　ピン番号

Member Function Documentation

void toneWrite(uint8_t note, uint8_t verocity)
指定したノート番号の周波数とヴェロシティの強さの矩形波を出力します． Parameters: note　ノート番号(7bit) verocity　ヴェロシティ(7bit)

void set(float frq, float level)
指定した周波数と Duty 比の矩形波を出力します． Parameters: frq　周波数 level　Duty 比

void free()
ピンを開放します．

Sound &operator=(float frq)
Sound::set の半省略形 Note: Duty 比は 50%固定出力になります. Parameters: frq　周波数

■ Spectoro API

Spectoro API はマイクロ分光器を容易に動作させる API です．システムタイマを使うことで不具合が生じる場合は NOT_USING_SYSTICK を定義してください.

・Spectro Class

Constructor & Destructor Documentation

Spectro (PinName clk_pin, PinName start_pin)
分光器に対する入出力設定と設定の初期化を行います. Note: DigitalOut オブジェクトあるいは FastOut オブジェクトを使用して CLK ピンと ST ピンの入出力設定を行います. Parameters: clk_pin　クロックパルスピン Start_pin　スタートパルスピン

~Spectro ()
分光器に対する入出力設定を開放します．また，読み込みハンドラをデタッチします. Note: DigitalOut オブジェクトあるいは FastOut オブジェクトを使用して CLK ピンと ST ピンの入出力の開放を行います.

Member Function Documentation

void startRead ();
読み込み処理等の割り込みを許可し，蓄積を開始します. Note: SystemTicker オブジェクトあるいは Ticker オブジェクトでタイマ割り込みを有効にします．また読み込み終了まで何度コールしても問題ありません.

bool notifyReadComplete ()
全ピクセルの読み込み完了を通知します. Returns: 読み込みの完了 true:完了 false:未了 (読み込みを行っていない場合も有)

bool setClockFrequency (float clk_frq)

クロック周波数を設定します.

Note:
蓄積時間はクロック周波数に依存するため，出来るだけクロック周波数を大きくしてください. クロック周波数の最大値はハードウェア依存であり，自身で最大値を計測する必要があります．設定の可不可はあくまでも理論値で判定しています.

Parameters:

clk_frq	クロック周波数[Hz]

Returns:
設定の可不可 true:設定可能 false:設定不可能

bool setAccumuTime (float us)

蓄積時間をマイクロ秒単位で設定します.

Note:
setFrequency でクロック周波数を設定してからコールしてください

Parameters:

us	蓄積時間[us]

Returns:
設定の可不可 true:設定可能 false:設定不可能

void setCalibCoeff (float *coeff)

5 次近似の波長補正係数を設定します.

Note:
詳しくは分光器の補正係数リストを参照してください.

Parameters:

coeff	波長補正係数の配列ポインタ

float waveLengthConvert (uint32_t pix_num)
設定した波長補正係数を基に，ピクセル番号から波長を算出します. Note: setCalibCoeff で補正係数を設定してからコールしてください. Parameters: pix_num | ピクセル番号 (0, 1, 2, … ,287) Returns: 設定の可不可 true:設定可能 false:設定不可能

void attachReadHandler (Callback<void()> handle)
ピクセルデータが分光器から出力される際にコールされるハンドラをアッタッチします. Note: このハンドラ内でピクセルの読み込みを行ってください. Parameters: handle | コールされる関数のポインタ

void attachReadHandler (T *obj, M method)
ピクセルデータが分光器から出力される際にコールされるハンドラをアッタッチします. Note: このハンドラ内でピクセルの読み込みを行ってください. Parameters: obj | メンバ関数がコールされるオブジェクトのポインタ method | コールされるメンバ関数のポインタ

■ SwitchStatus API

スイッチ入力のチャタリング除去機能や長押し検出機能を提供する API です.

・SwitchStatus Class

Constructor & Destructor Documentation

SwitchStatus(uint32_t pin_num, ...)
スイッチ入力ピンの初期化を行います. Parameters: pin_num: スイッチ入力ピンの総数 ...: (PinName 型)ピン番号(可変長)

virtual ~SwitchStatus()
全てのスイッチ入力ピンを開放します.

Member Function Documentation

void mode(PinMode pull_mode = PullDefault)
スイッチ入力ピンのプルモードを設定します. Note: プルモードは read 時の論理モードに直結します. Parameters: pull_mode: プルモード

void startSampring(uint32_t us = 30000)
サンプリング周期を μs 単位で設定します. Parameters: us: サンプリング周期[μs]

bool read(uint32_t sw_num)

スイッチの状態を取得します.

Parameters:

sw_num	スイッチ番号

Returns:

スイッチの状態(押しているとき:True)

bool longPressRead(uint32_t sw_num, uint32_t th)

スイッチの長押し状態を取得します.

Parameters:

sw_num	スイッチ番号
th	閾値時間[μs]

Returns:

スイッチの状態(長押ししている状態:True)

■　スペクトル表示アプリケーション drawSpectrum

Windows 用にスペクトル表示アプリケーションを用意しました．GUI 操作でスペクトルの表示やスペクトルの保存，露出設定が可能です．

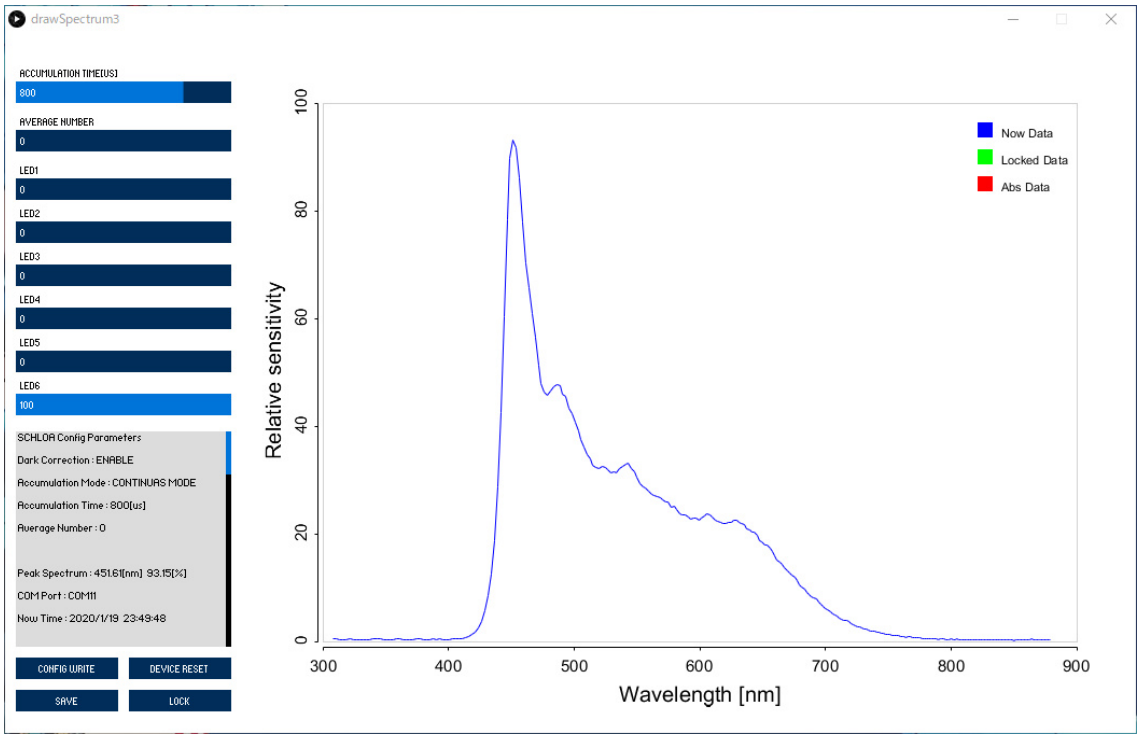

Fig. 10.1 Original GUI “DrawSpectrum”

言語は Java で開発されています．仕様上，相対感度と表示されていますが，これは各ピクセルにおける最大感度に対する相対値を取っています．修正が必要になります．

・データフォーマット

データブロック長は可変長です．シリアル通信を使って文字列で送信してください．開始文字は'$'(ドル記号)，区切り文字は ','(カンマ)，終了文字は'¥n'(LF 文字)です．文字形式は ASCII です．

・送信データフォーマット

Tabel 10.1 drawSpectrum Send Data Formats

String	S0	S1	S3	S4	…	S291	S293
Data	'$'	CNF_C	ACM_T	PIX0	…	PIX287	"¥n"

"$" : Start Character

CNF：Config Characters

Tabel 10.2 CONF Character Data Formats

Character	C0	C1	C2	C3	C4	C5
Data	AEN	DEN	AMOD	RSV	AVERAGEN	

AEN : Accumulation Enable

'0' : Disable

'1' : Enable

DEN : Dark Correction Enable

'0' : Disable

'1' : Enable

AMOD : Accumulation Mode

'0' : Oneshot Mode

'1' : Continuous Mode

RSV：Reserved

AVERAGEN : Average Number

"01" ~ "17"

ACM_T：Accumulation Time

PIXn : Pixel Datas (Number of n)

'¥0'：End Character

','： Delimiter

・受信データフォーマット

Table 10.2 drawSpectrum Receive Data Formats

String	S0	S1	S2	S3	S4	…	S9	S10	11
Data	'$'	RST	AVERAGEN	ACM_T	LED0	…	LED5	TIME	"¥n"

"$" : Start Character

RST：Reset Flag

'0' : Normal Operation

'1' : Reset Operation

AVERAGEN : Average Number

"01" ~ "17"

ACM_T：Accumulation Times

LEDn : LED brightness(Number of n)

'¥0'：End Character

','： Delimiter

・操作方法

・各部名称

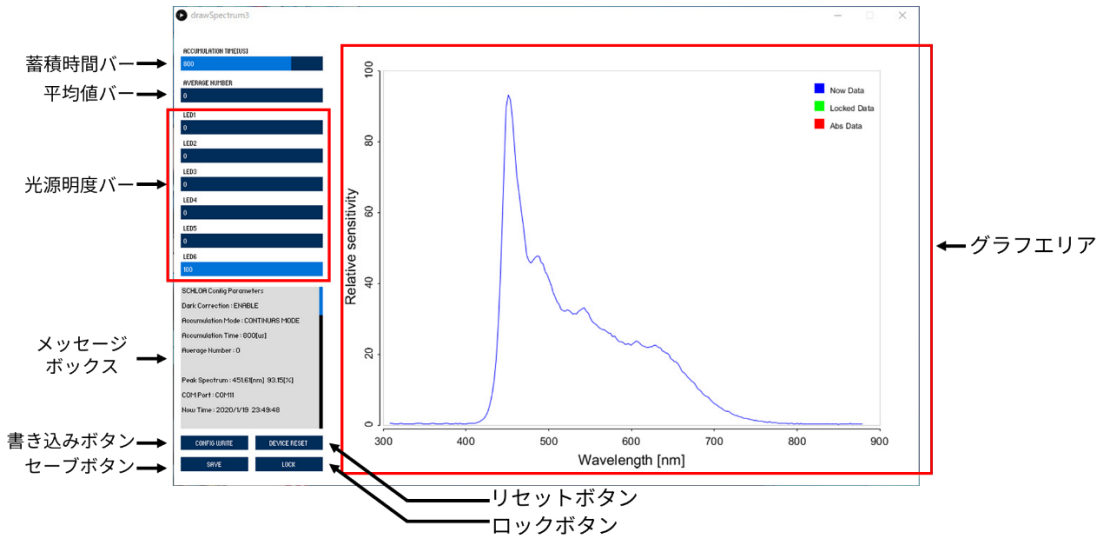

Fig. 10.2 Controller Names

蓄積時間バー：蓄積時間の設定値の操作

平均値バー：平均数の設定値の操作

光源明度バー：各光源の明度の操作

メッセージボックス：測定条件の表示

書き込みボタン：設定値の送信

リセットボタン：リセット命令の送信

セーブボタン：csv データと png データの保存

ロックボタン：スペクトルを固定表示した状態で現在のスペクトルを表示

・キャリブレーション

キャリブレーションは波長にのみ対応しています.

data ファイル内の wavelength_calibration_coefficient.csv 内の 5 つの補正係数を変更することで波長補正が可能です．補正係数はマイクロ分光器購入時の付属資料に記載されています.

・吸収スペクトル測定

ロックボタンを押すと押す前のスペクトルを入射スペクトル，押した後のスペクトルを透過(反射)スペクトルとして自動的に吸収スペクトルを算出します．また，蓄積時間の差異も自動的に補正されます.

・測定データの保存

セーブボタンを押すと測定値を CSV 形式でグラフエリアを PNG 形式で保存することが可能です．ファイル名は SPC 年月日時分秒.拡張子でドキュメント内の DrawSpectrum に保存されます.