

ARM Cortex-M 開発環境 CooCox - STM32編 -

2012.04.21 マイクロマウス東日本支部 高橋(ゼロソフト)

内容

- 最近のマイコン事情
- ルネサスの動向
- STM32
- STM32 開発環境 (GNU)
- CooCox
- 開発環境(ハードウェア)
- CoIDE (統合開発環境)
- CoIDEデモ - プロジェクト作成、クロック設定
- CoIDEデモ – printf (Semihosting)
- CoIDEデモ – GPIO LED点灯
- CoOS (RTOS)
- Atomic Write
- お知らせ

最近のマイコン事情

- 32bit化
- ARM 陣営 vs Renesas

ARM Cortex-M



nuvoTon



Renesas RX



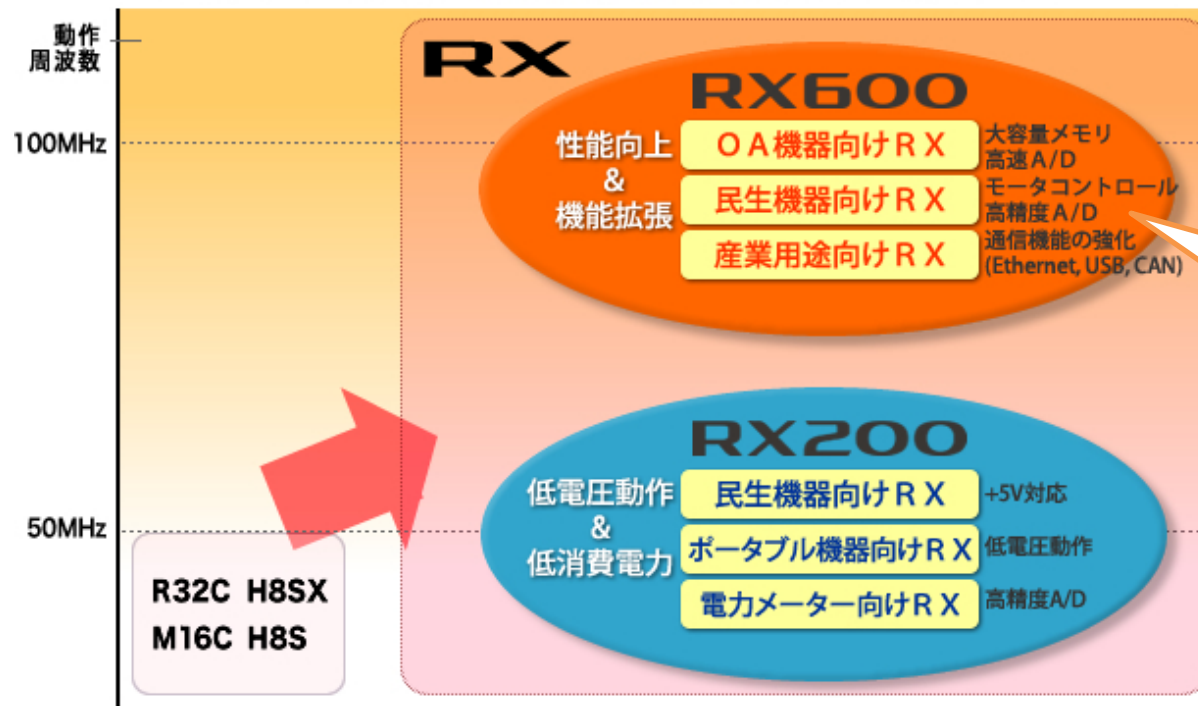
ルネサスの動向

Renesas High End MCU

High Performance
High Scalability
High Reliability



- RX600は既存製品の機能を継承し性能向上と機能拡張を実現
- RX200の最大の特長は、低電圧動作&低消費電力。RX600と互換を維持



R62Tがマウス
に良さそう
64pin LQFP
3.3V/5V

STM32

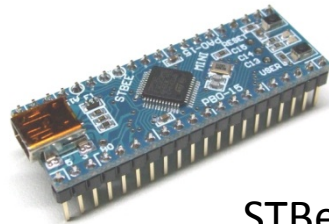
- ・最近、ボードや書籍がいろいろと出回ってきた。
- ・マウスでの採用事例も増えてきた様子。



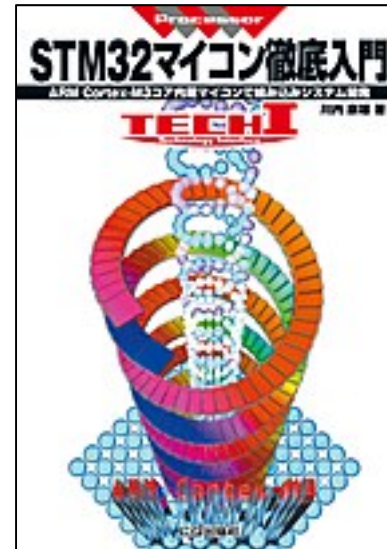
STM32VL
Discovery



STM32L
Discovery



STBee Mini



STM32 開発環境 (GNU)

ちょっと前まで



サイズ制限なし



コンパイラ
Code Sourcery G++



Coocox®

サイズ制限なし

現在



Lite版サイズ制限あり (コンパイラは自前ビルド)

Mentor
Graphics

Sourcery
CodeBench

Lite版 IDEなし

Coocox®

サイズ制限なし (コンパイラはLaunchPadより入手)

CooCox

- ARM Cortex-Mに特化した開発環境を提供 (Free)
- サポートしているCPUベンダー: Atmel, Energy Micro, Holtek, NXP, Nuvoton, ST, TI
- 発展途上。商用に比べて劣るところはあるが、アイデアは秀逸。ホビー用途には十分。
- ここ1年の更新で、STM32サポート状況が良くなった。

CoIDE
統合開発環境

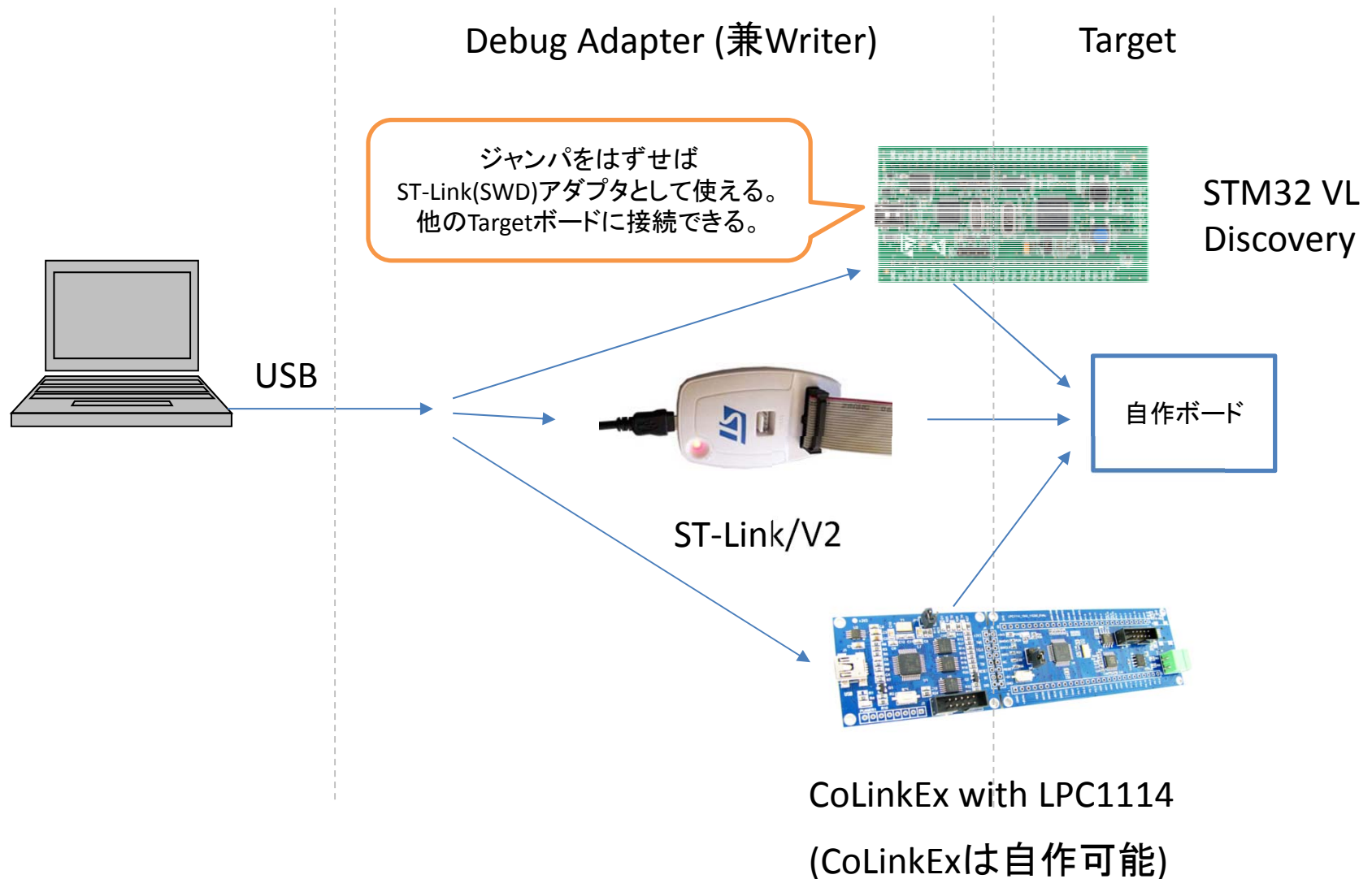
CoLinkEx
デバッグ用アダプタ
(SWD/JTAG)

CoOS
Real Time OS

CoSmart
ピン割当 &
コード生成

CoX
ペリフェラル
共通ライブラリ

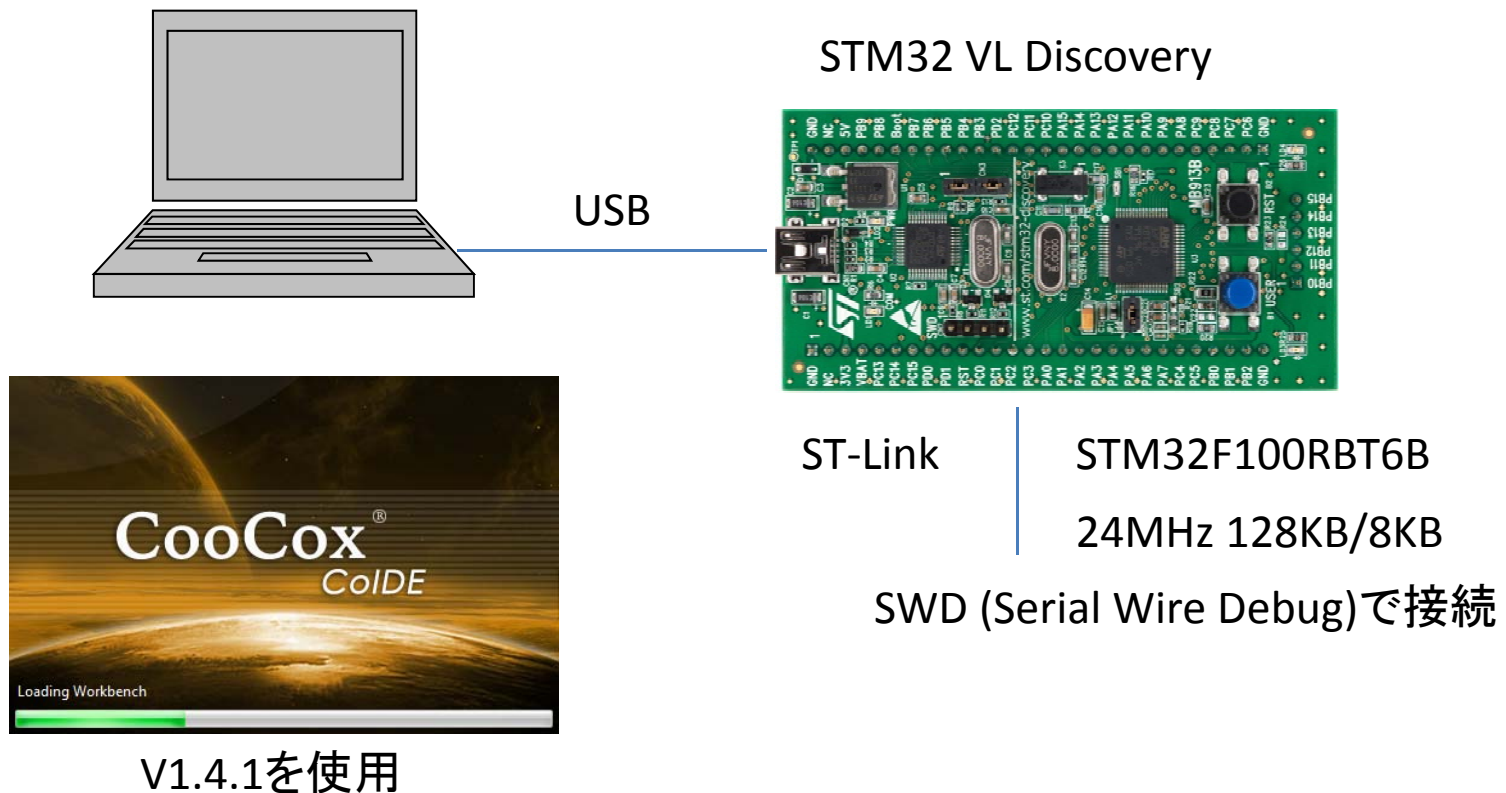
開発環境(ハードウェア)



CoIDE (統合開発環境)

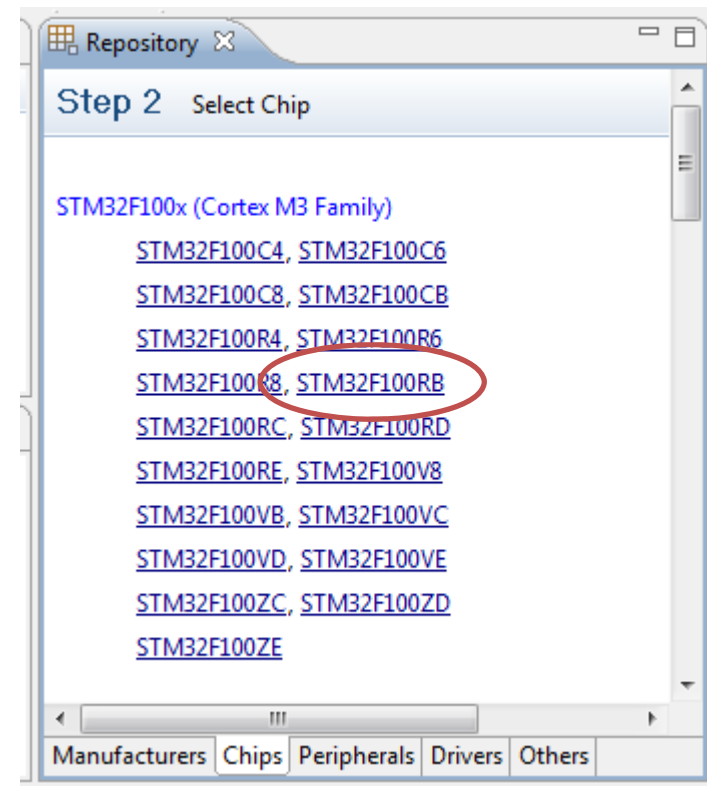
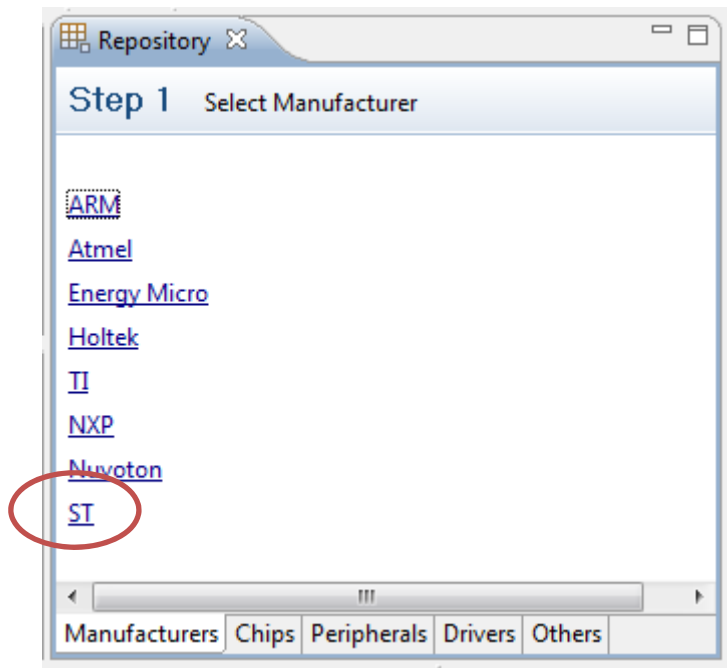
- CoIDEとコンパイラ(GCC ARM Embedded)をインストールする。
- CoIDEから、“Select Toolchain Path”でコンパイラのパスを指定する。

デモ環境:



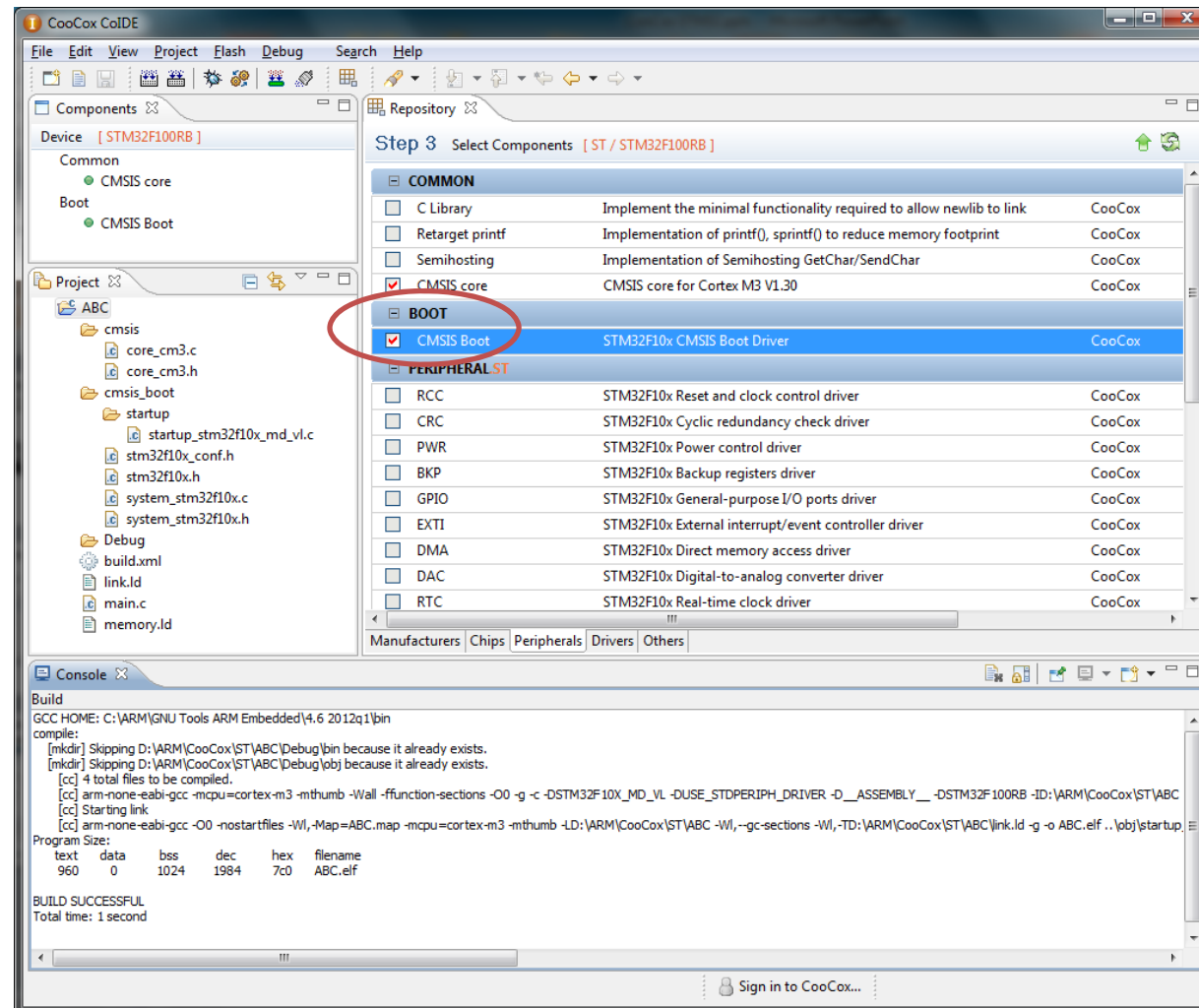
CoIDEデモ - プロジェクト作成 (1/3)

- New Project
- RepositoryからManufacturers, Chipsを選択する。

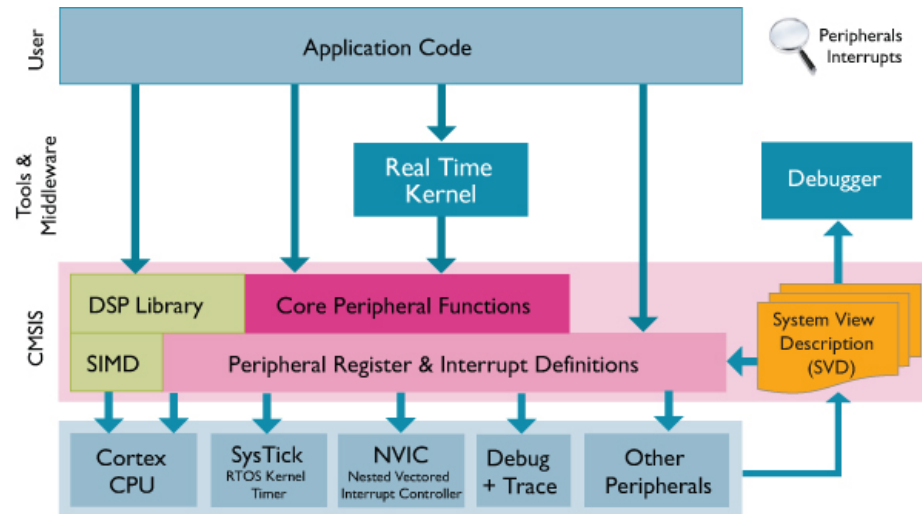


CoIDEデモ - プロジェクト作成 (2/3)

- PeripheralsからCMSIS Bootを選択する。



CoIDEデモ - プロジェクト作成 (3/3)



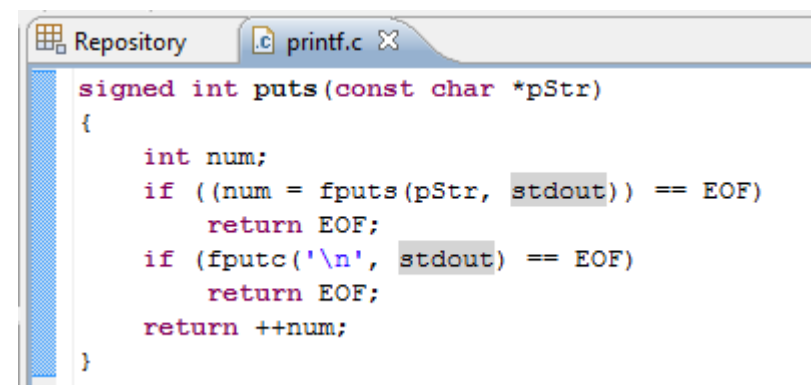
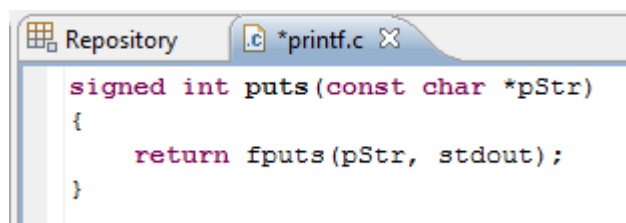
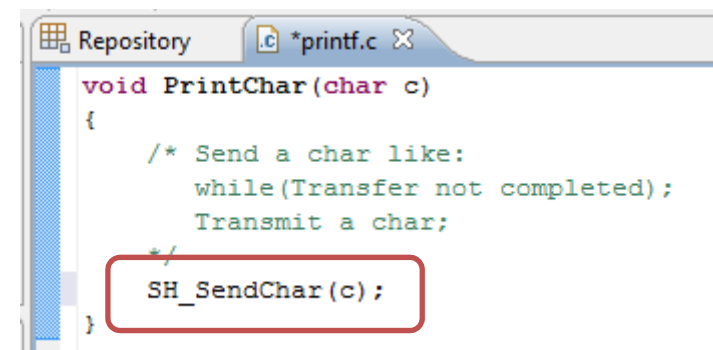
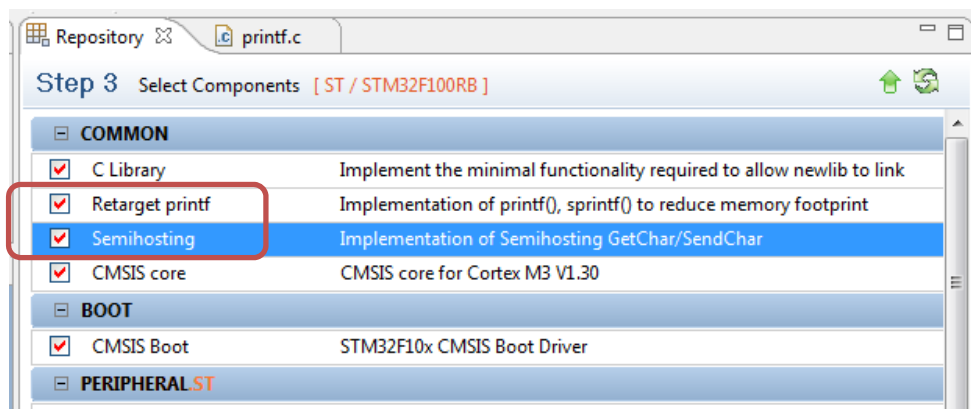
- CMSISはstartup, ISR(Interrupt Service Routine) 等が定義されている。
- ST Microが出している Firmware Libraryに入っているのと同じものがCooCoxで使われている。
Documentは”.../Libraries/CMSIS/Documentation/CMSIS_Core.htm”
- startupはCooCox独自 (.sではなく.cで書かれている)

CoIDEデモ – クロック設定

- CMSIS (boot)でクロック設定が行われている。
- CoIDEでは、CPUを選択した時に、CPUカテゴリをコンパイルオプションで指定している。
これに連動する形でCMSIS内のクロック設定が決まる。
- STM32 VL Discoveryでは設定変更は不要。(HSE 8MHz, SYSCLK 24MHz)
コンパイルオプション -DSTM32F10X_MD_VL が指定されると、
system_stm32f10x.cでSYSCLK_FREQ_24MHzが選択され、
PLL設定が行われる。
- CMSISは直接修正しないで、コンパイルオプションで制御した方が良い。
CooCoxでは簡単に出し入れができてしまうが、この時修正が消えてしまう。

CoIDEデモ – printf (Semihosting) (1/4)

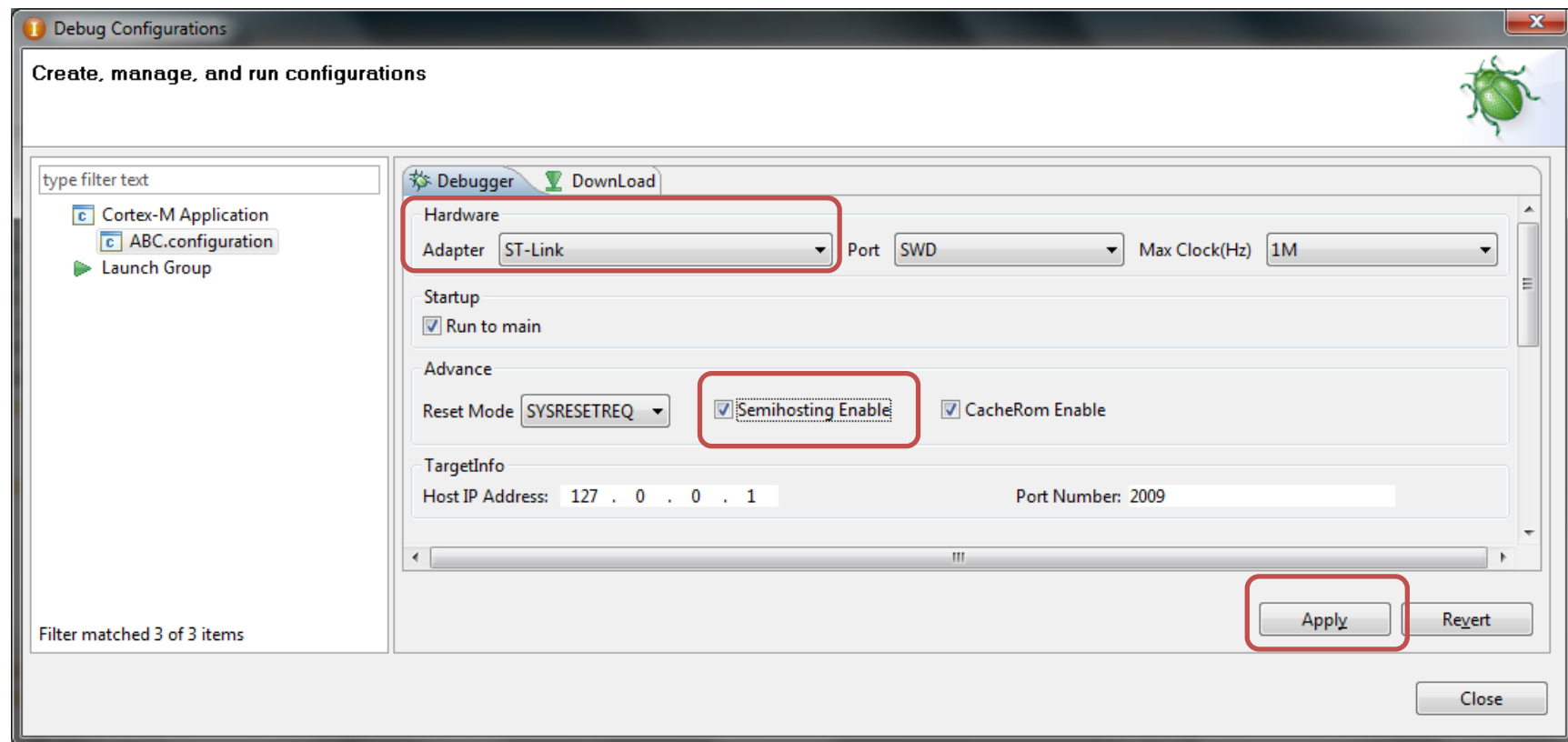
- PeripheralsからRetarget printf, Semihostingを選択する。
- printf.cを書き換える。
 - 1) #include “semihosting.h”
 - 2) PrintChar() Semihosting出力
 - 3) puts() バグ対応



- main.cにprintfを仕込んでビルドする。

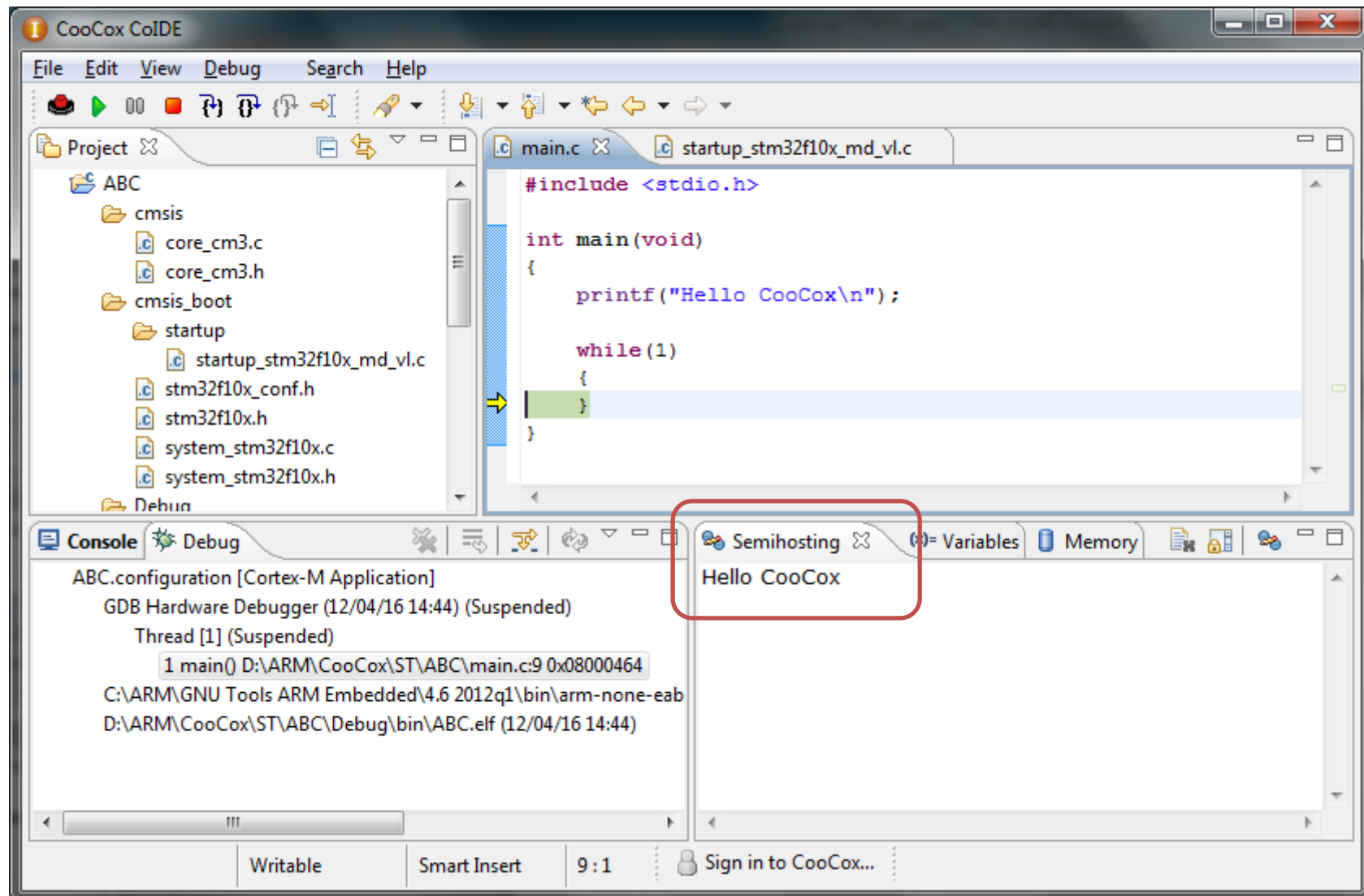
CoIDEデモ – printf (Semihosting) (2/4)

- Debug ConfigurationでST-Linkを指定
- Semihosting Enableを指定



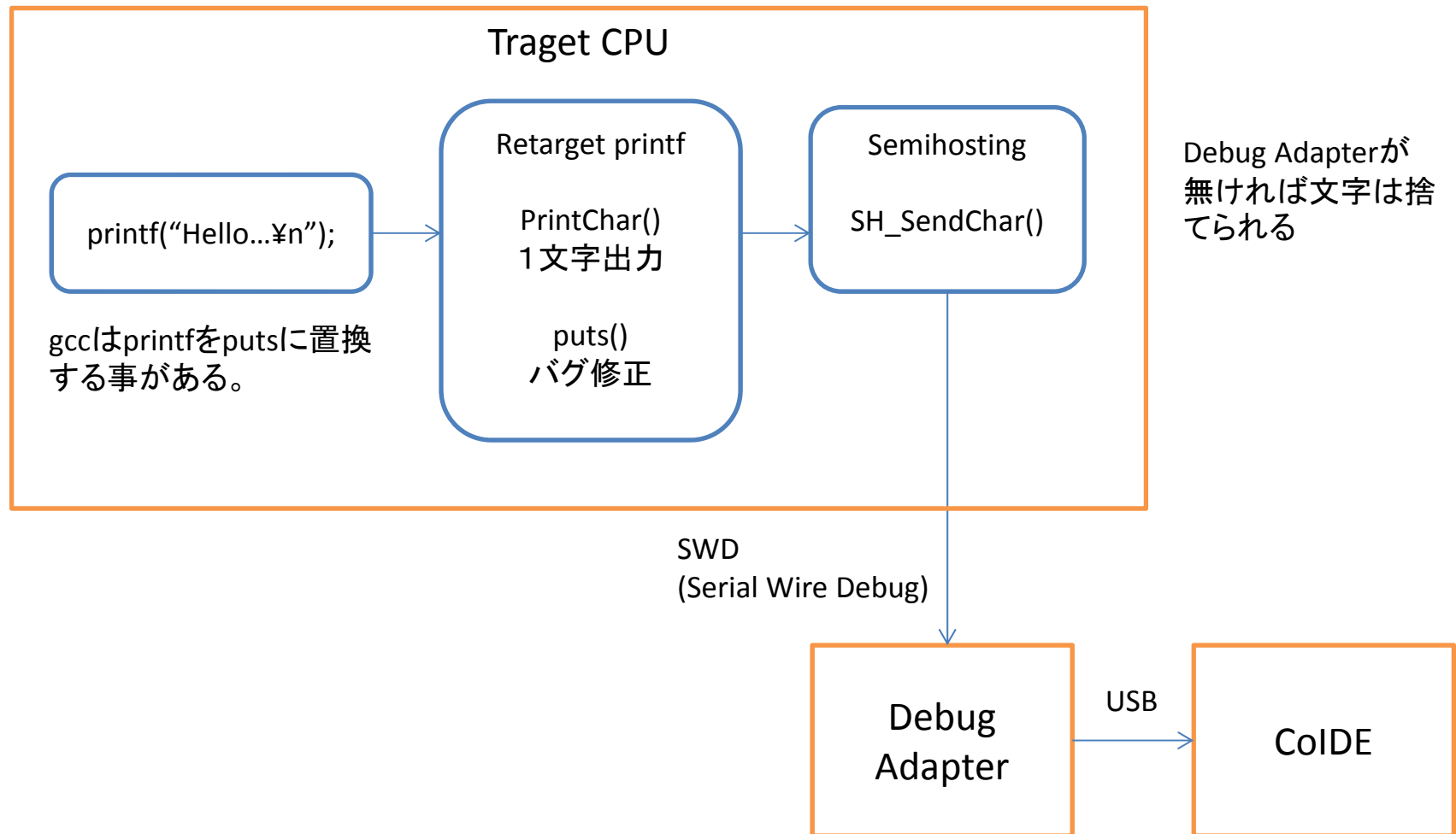
CoIDEデモ – printf (Semihosting) (3/4)

- Debug開始



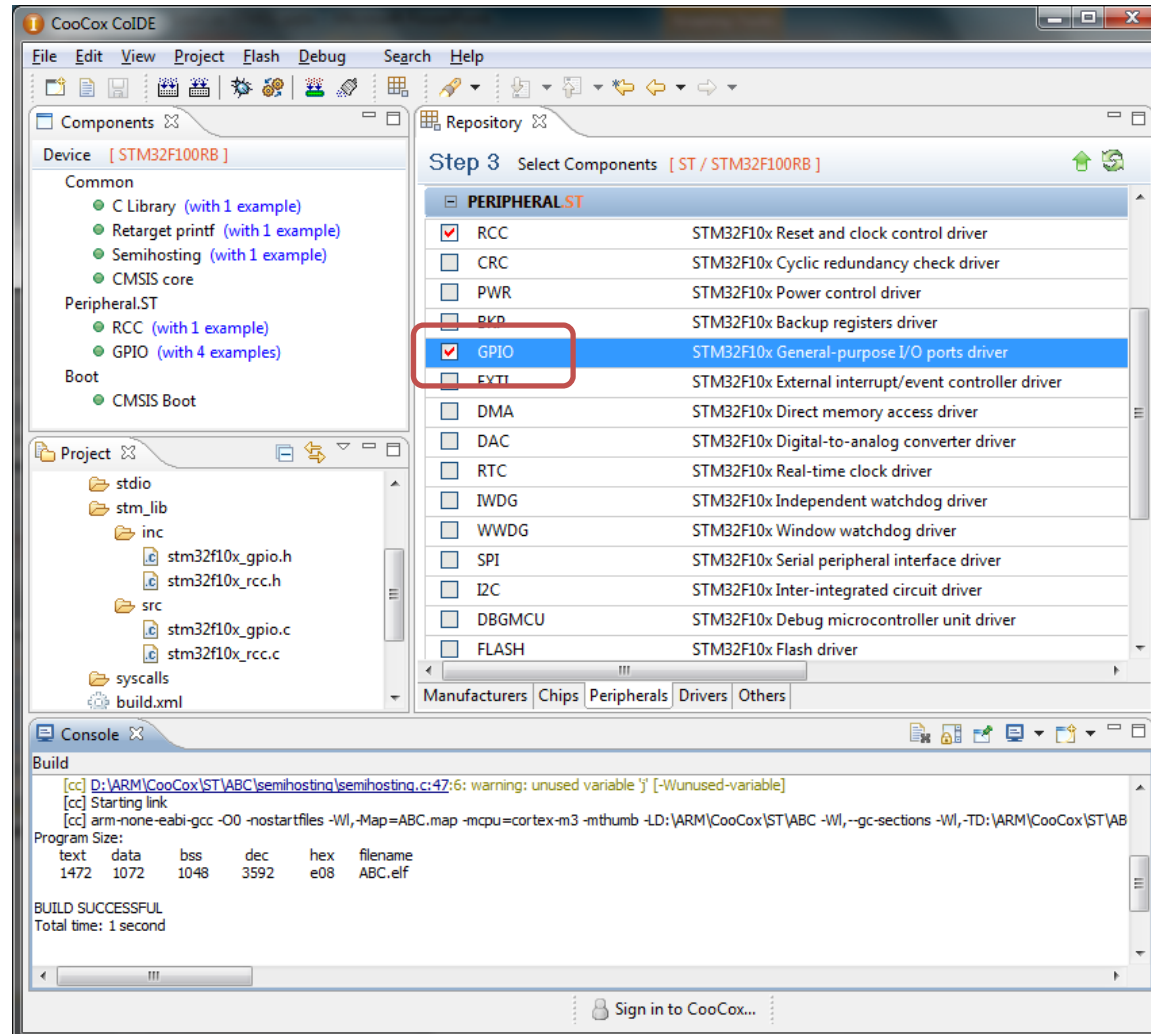
CoIDEデモ – printf (Semihosting) (4/4)

・まとめ



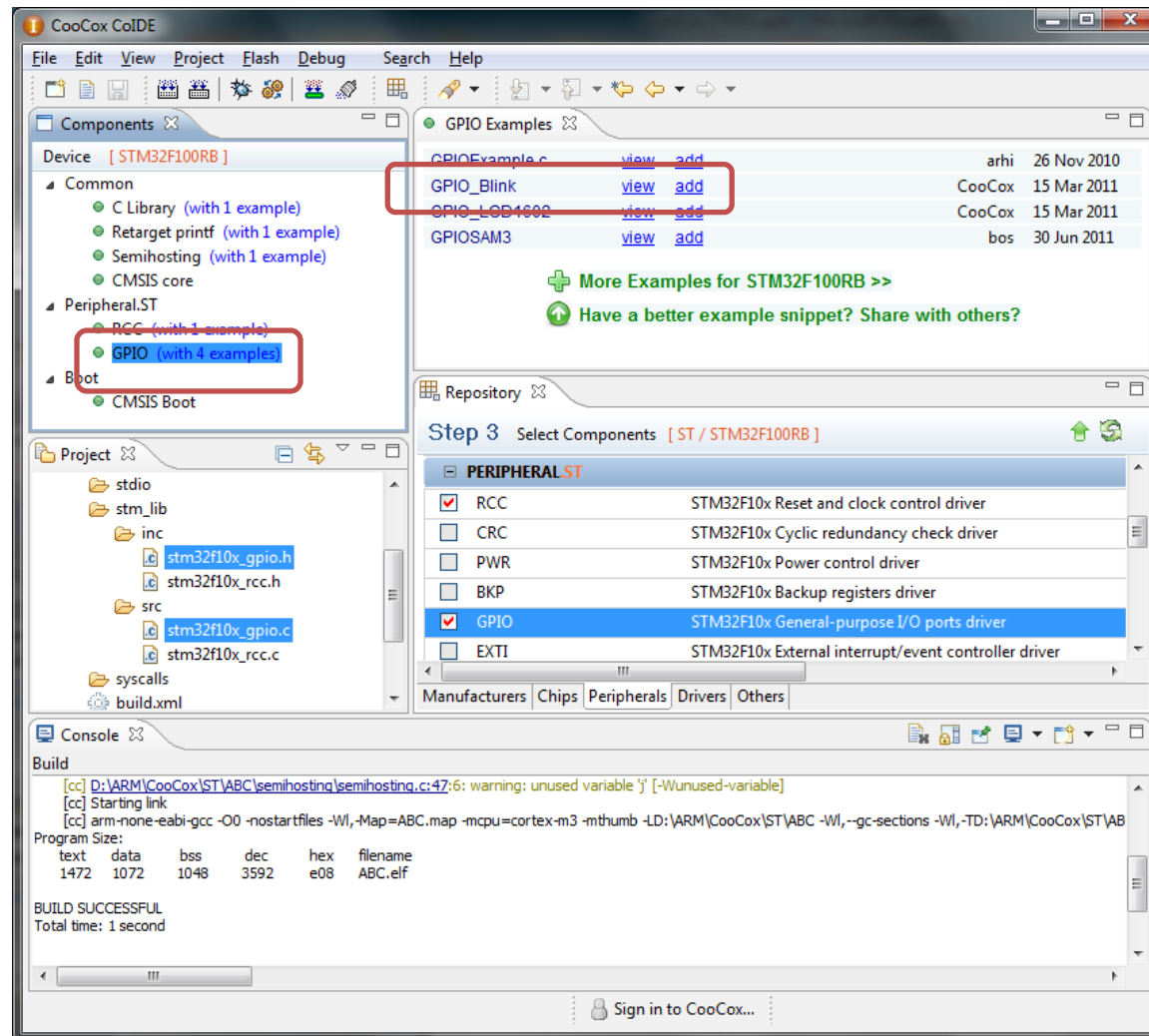
CoIDEデモ – GPIO LED点灯 (1/7)

- PeripheralsからGPIOを選択する。



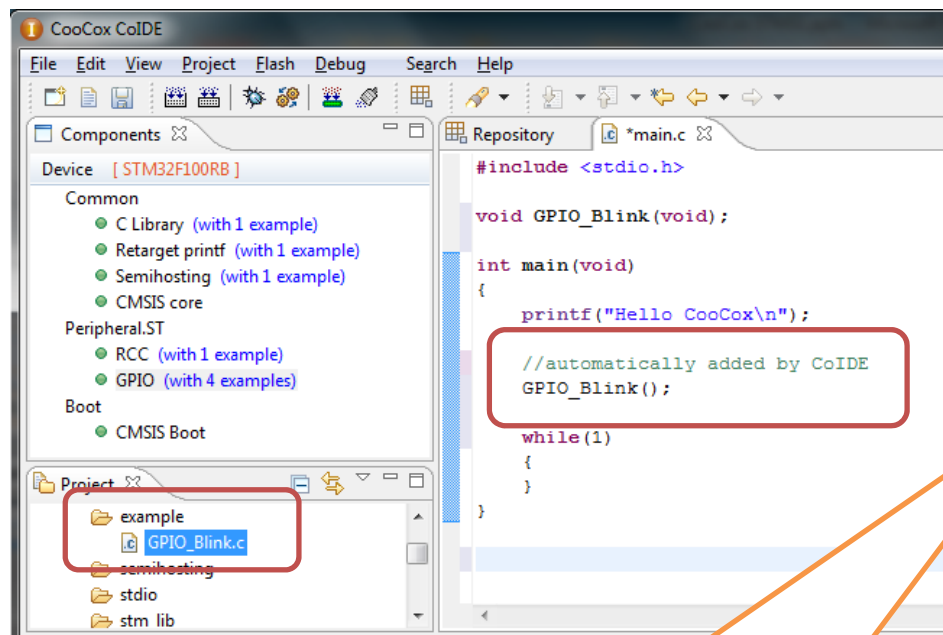
CoIDEデモ – GPIO LED点灯 (2/7)

- GPIOからexampleを選択する。(addで追加)

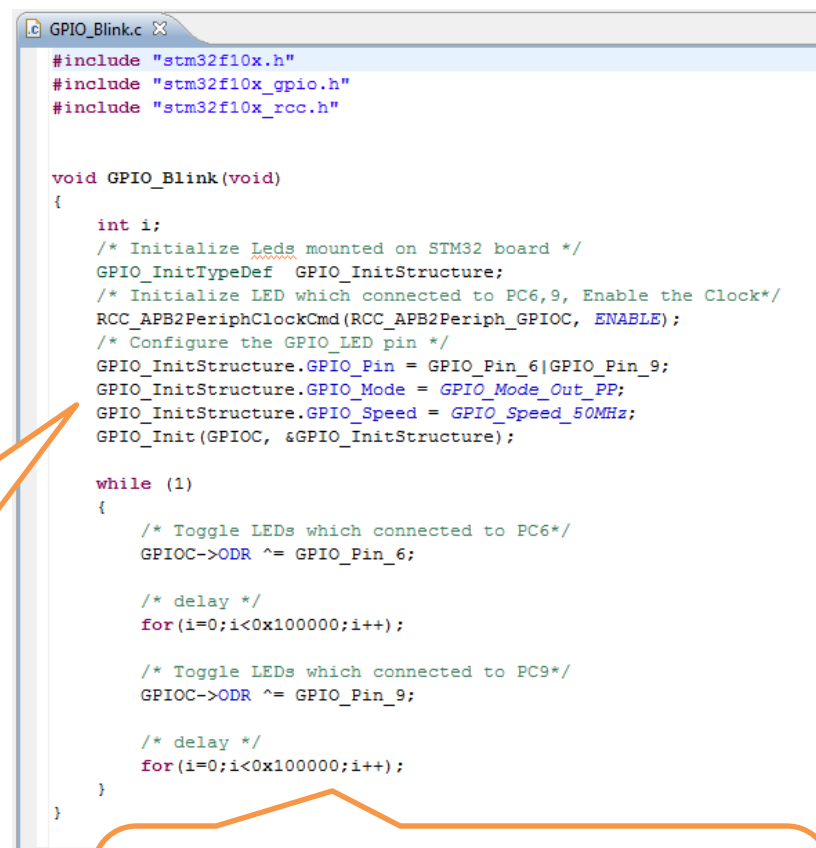


CoIDEデモ – GPIO LED点灯 (3/7)

- exampleを書き換える。



RCC (Reset and Clock Control)でGPIOにClockを供給
GPIOのピンを出力に設定



PC6,9を点滅させている。
STM32VL DiscoveryではPC8,9にLEDが割り
当てられているので、番号を書き換える。

CoIDEデモ – GPIO LED点灯 (4/7)

- ・問題点：時間待ちが単なるループ（最適化で消えてしまう）
- ・解決策：SysTickハンドラを使う。（分解能 1ms）

```
GPIO_Blink.c
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

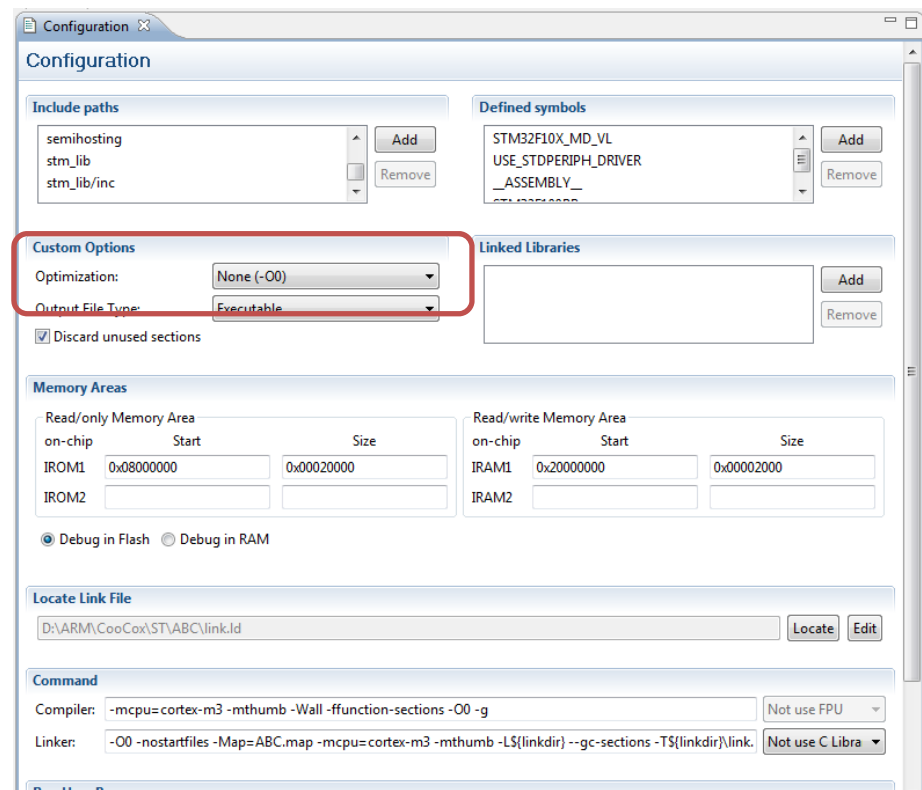
void GPIO_Blink(void)
{
    int i;
    /* Initialize Leds mounted on STM32 board */
    GPIO_InitTypeDef GPIO_InitStructure;
    /* Initialize LED which connected to PC6,9, Enable the Clock*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    /* Configure the GPIO_LED pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    while (1)
    {
        /* Toggle LEDs which connected to PC6*/
        GPIOC->ODR ^= GPIO_Pin_6;

        /* delay */
        for(i=0;i<0x100000;i++);

        /* Toggle LEDs which connected to PC9*/
        GPIOC->ODR ^= GPIO_Pin_9;

        /* delay */
        for(i=0;i<0x100000;i++);
    }
}
```



-Os (サイズ優先)がお勧め。コードサイズが約半分になり、速度2倍。

CoIDEデモ – GPIO LED点灯 (5/7)

- SysTick.c, SysTick.hを導入

```

//
// SysTick.c
//
// SysTick Delay (ms)
//

#pragma GCC optimize ("-Os")

#include "stm32f10x.h"
#include "SysTick.h"

static volatile uint32_t TimingDelay;

void SysTick_Handler(void)
{
    if (TimingDelay != 0) {
        TimingDelay--;
    }
}

void SysTick_Init(void)
{
    // Setup SysTick Timer for 1 msec interrupts
    SysTick_Config(SystemCoreClock / 1000);
}

void SysTick_DelayMs(uint32_t nTime)
{
    TimingDelay = nTime;
    while(TimingDelay != 0);
}

```

このソースを常に
最適化(-Os)する。
効率重視

SysTick_Handlerはsystem側でweak属性で定義されている。
ここで再定義すると、オーバーライドされる。
1ms 毎に呼び出される。

SystemCoreClockはsystem側で定義されている。(単位Hz)
SysTick_Config()に1ms分のクロック数を指定する。

パラメータに待ち時間(ms単位)を指定する。
SysTick_Handler()のカウントダウンで指定時間に達したら終了

```

//
// SysTick.h
//

#ifndef SYSTICK_H
#define SYSTICK_H

#include <stdint.h>

void SysTick_Init(void);
void SysTick_DelayMs(uint32_t dlyTicks);

#endif // SYSTICK_H

```

CoIDEデモ – GPIO LED点灯 (6/7)

- SysTick_Init(), SysTick_DelayMs()を呼び出す。

```
GPIO_Blink.c
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "SysTick.h"

void GPIO_Blink(void)
{
    SysTick_Init();

    /* Initialize Leds mounted on STM32 board */
    GPIO_InitTypeDef GPIO_InitStructure;
    /* Initialize LED which connected to PC8,9, Enable the Clock*/
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    /* Configure the GPIO_LED pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    while (1)
    {
        /* Toggle LEDs which connected to PC8*/
        GPIOC->ODR ^= GPIO_Pin_8;

        SysTick_DelayMs(500);

        /* Toggle LEDs which connected to PC9*/
        GPIOC->ODR ^= GPIO_Pin_9;

        SysTick_DelayMs(500);
    }
}
```

SysTickを初期化する。mainの頭で一度呼び出せば良い。
ここでは説明の都合上、GPIO_Blink()の頭で呼び出している。

ループをSysTick_Delay()に置き換えた。

CoIDEデモ – GPIO LED点灯 (7/7)

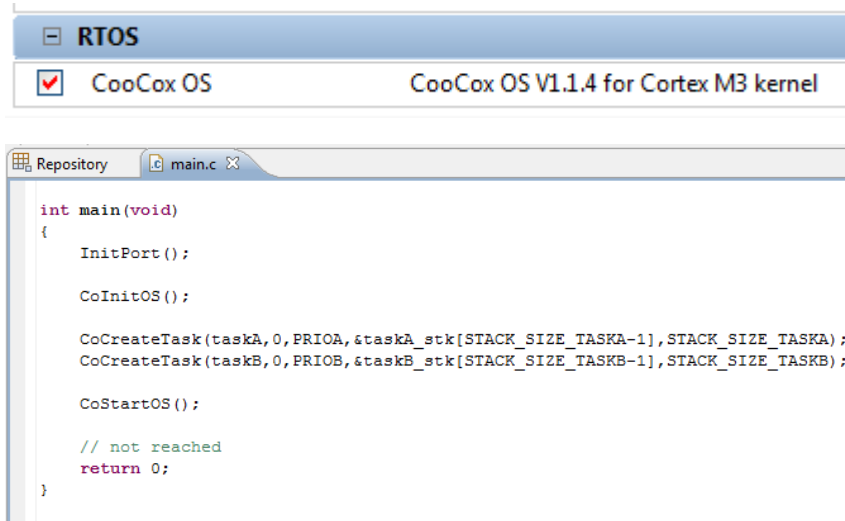
- ・PORTBにLEDを16個付けてみた。PB3,4が使えない。
- ・DefaultでJTAGに割り当てられている。AFIOでRemapする。

LQFP64の場合、ポートは51本使えるが...

<u>GPIOA</u>	<u>GPIOB</u>	<u>GPIOC</u>	<u>GIOD</u>
PA0	PB0	PC0	PD0: OSC_IN (X-tal)
PA1	PB1	PC1	PD1: OSC_OUT (X-tal)
PA2	PB2	PC2	PD2
PA3	PB3: JTDO (Remap可能)	PC3	
PA4	PB4: NJTRST (Remap可能)	PC4	
PA5	PB5	PC5	
PA6	PB6	PC6	
PA7	PB7	PC7	
PA8	PB8	PC8	
PA9	PB9	PC9	
PA10	PB10	PC10	
PA11	PB11	PC11	
PA12	PB12	PC12	
PA13: SWDIO	PB13	PC13: 制限有(※1)	※1 Sinkのみ, 3mA, 2MHz
PA14: SWCLK	PB14	PC14: 制限有(※1,2)	※2 32.768kHz X-tal 接続
PA15: JTDI (Remap可能)	PB15	PC15: 制限有(※1,2)	

CoOS (RTOS)

- 様々なI/Oを扱うようになってくると、Real Time OSが必要になってくる。
- 処理を複数のタスクに分割し、優先度を設定する。
- タスクスケジューリングは定期的にRTOSが行う。
- プリエンプティブなタスクスケジューリング (SysTick割り込みを利用)
- I/O割り込みにより、優先度の高い処理を行う場合、OS経由でタスクに指示を出す事で、すぐにタスクスケジューリングを起こさせる。
- CoOSは、OsConfig.hでカスタマイズする。
- CFG_CPU_FREQを24MHzに修正
- CFG_SYSTICK_FREQは100Hz (1000Hzまで可)



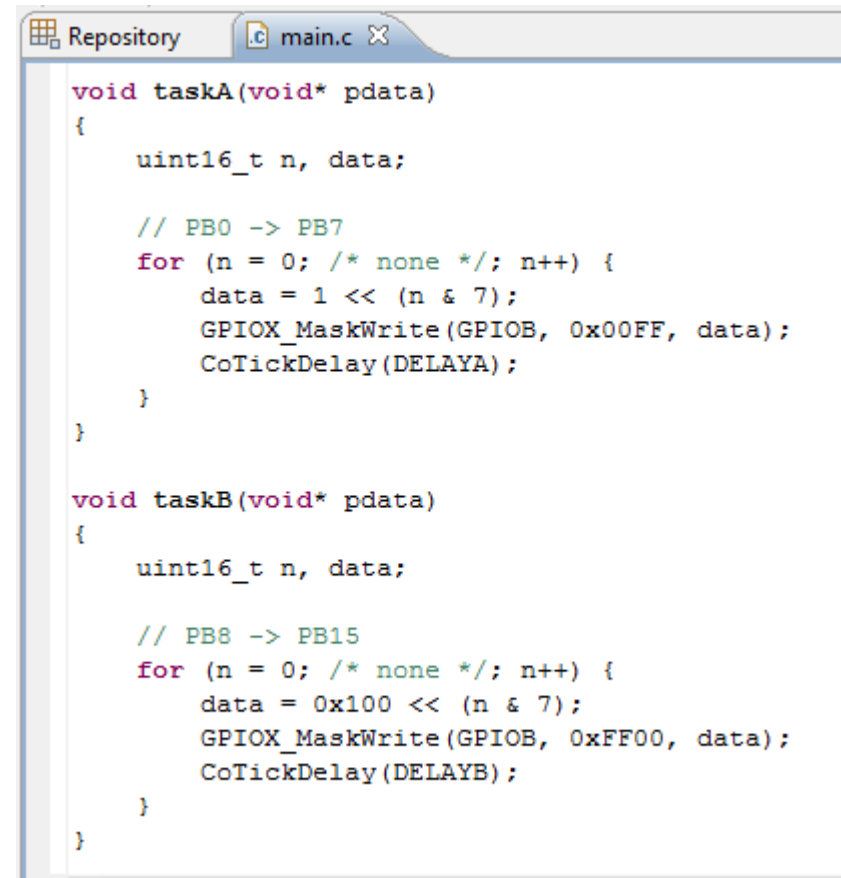
```
int main(void)
{
    InitPort();

    CoInitOS();

    CoCreateTask(taskA, 0, PRIOA, &taskA_stk[STACK_SIZE_TASKA-1], STACK_SIZE_TASKA);
    CoCreateTask(taskB, 0, PRIOB, &taskB_stk[STACK_SIZE_TASKB-1], STACK_SIZE_TASKB);

    CoStartOS();

    // not reached
    return 0;
}
```



```
void taskA(void* pdata)
{
    uint16_t n, data;

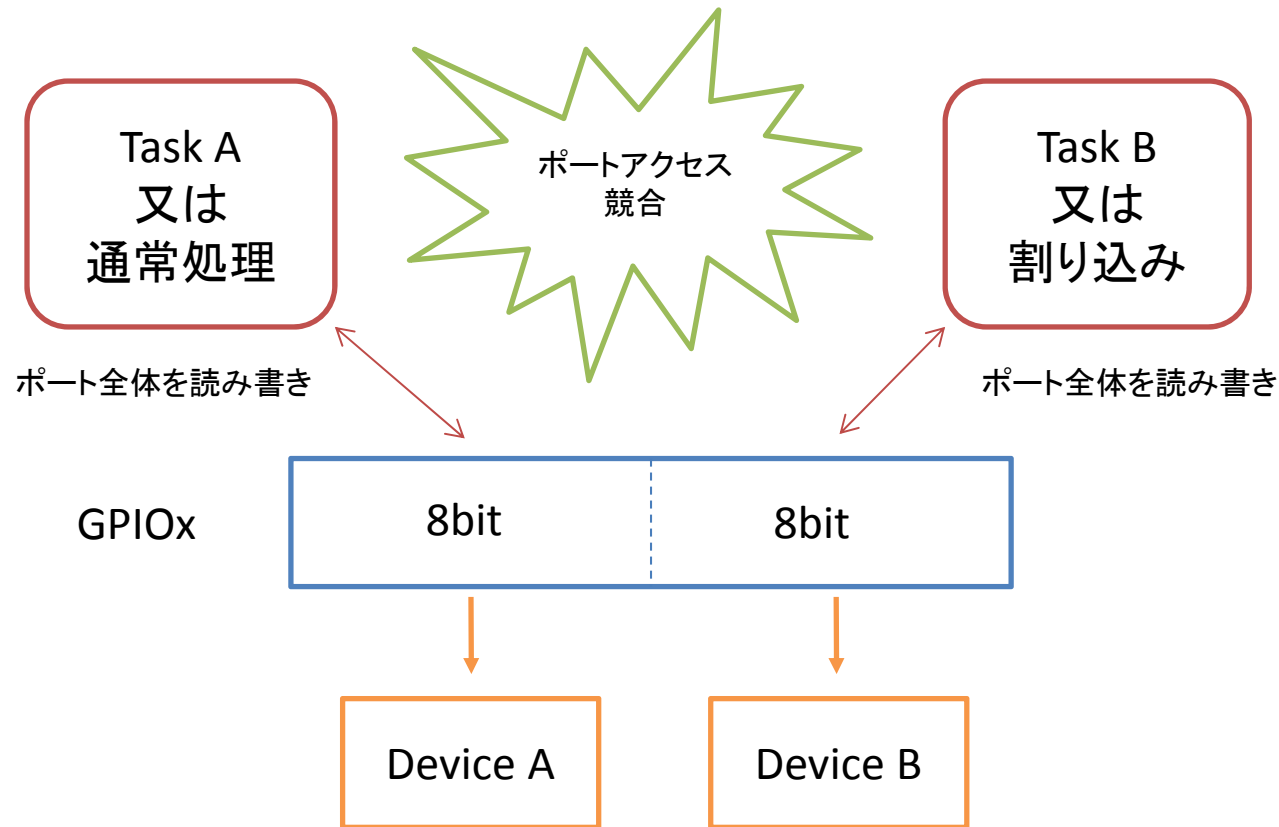
    // PB0 -> PB7
    for (n = 0; /* none */; n++) {
        data = 1 << (n & 7);
        GPIOX_MaskWrite(GPIOB, 0x00FF, data);
        CoTickDelay(DELAYA);
    }
}

void taskB(void* pdata)
{
    uint16_t n, data;

    // PB8 -> PB15
    for (n = 0; /* none */; n++) {
        data = 0x100 << (n & 7);
        GPIOX_MaskWrite(GPIOB, 0xFF00, data);
        CoTickDelay(DELAYB);
    }
}
```

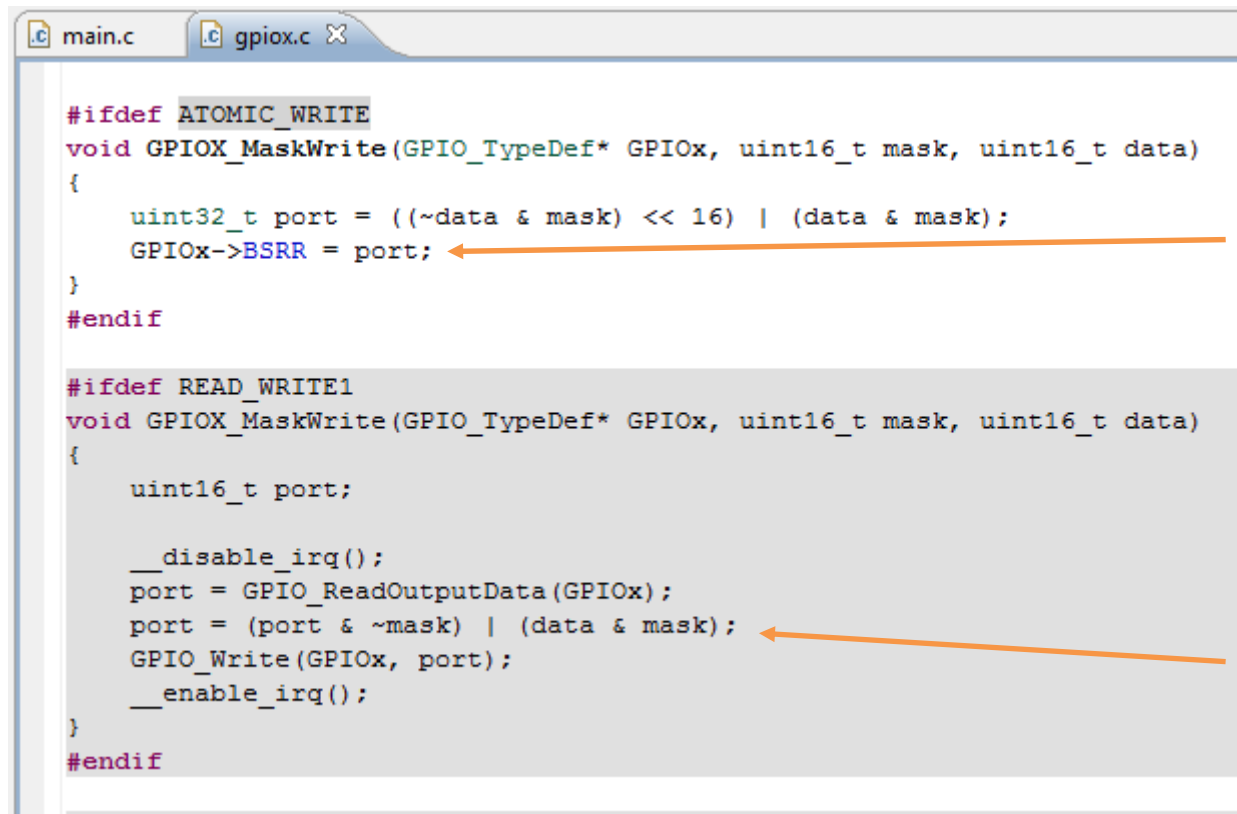
Atomic Write (1/3)

- マルチタスクで、ポート書き込みが競合する。
(シングルタスクでも通常処理と割り込みでポート書き込みを行うケースは同様)



Atomic Write (2/3)

- STM32ではGPIOのBSRRを使うと、簡単にAtomic Writeできる。
- 通常はGPIOをRead - Modify - Writeする場合、割り込みを禁止する。



```
main.c  gpiox.c X
#ifdef ATOMIC_WRITE
void GPIOX_MaskWrite(GPIO_TypeDef* GPIOx, uint16_t mask, uint16_t data)
{
    uint32_t port = ((~data & mask) << 16) | (data & mask);
    GPIOx->BSRR = port;
}
#endif

#ifdef READ_WRITE1
void GPIOX_MaskWrite(GPIO_TypeDef* GPIOx, uint16_t mask, uint16_t data)
{
    uint16_t port;

    __disable_irq();
    port = GPIO_ReadOutputData(GPIOx);
    port = (port & ~mask) | (data & mask);
    GPIO_Write(GPIOx, port);
    __enable_irq();
}
#endif
```

BSRRレジスタにResetとSetする値を一度に書き込む。レジスタアクセスはアセンブラ1命令で行われるので、割り込みに邪魔されない。Atomicである。

書籍で良く見かける事例
Read - Modify - Write
Read後に割り込みが発生してポート状態が変化してしまうと、割り込みで変更した状態が反映されていない値がWriteされてしまう。これを抑止するには、前後で割り込みを禁止する。

Atomic Write (3/3)

・GPIO 出力用レジスタはODR

・BSRRはODRの特定ビットをAtomicに変更できる。

・すべての書き出しをBSRR経由にする事で、排他制御が不要となる。

7.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy[15:0]**: Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. E).

7.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy**: Port x Reset bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Reset the corresponding ODRx bit

Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy**: Port x Set bit y (y= 0 .. 15)

These bits are write-only and can be accessed in Word mode only.

0: No action on the corresponding ODRx bit

1: Set the corresponding ODRx bit

お知らせ



2012 Japan IT Week 春 ビッグサイト 内
第15回 **組み込みシステム 開発技術展**
会期: 2012年5月9日(水)~11日(金) 会場: 東京ビッグサイト 主催: リー

通称ESEC。入場券持ってきました。
日本最大。すごい人手です。
初日が一番空いてます。名刺があると便利。



トランススタ技術 Interface Tech Village
秋葉原でARMマイコンを体感し、楽しく学ぶ1日
**ARM Cortex-M
マイコン・ワークショップ 2012**
2012/5/25 金 開催
ARM Cortex-M.
MCU Workshop 2012

4/23(月) 登録開始