『RTOSとは？イベントフラグ編』

タスク間協調(同期)を利用した、
アプリケーション作成例とイベントフラグ
の動作について

イー・フォース株式会社　田村　聡

RTOSとは？ と題した、Webinarについて

一連のWebinarでは、RTOS(特に、μITRON)について解説をします

RTOSとは？

1.イベントフラグ編

タスク間協調(同期)を利用した、アプリケーション作成例とイベントフラグの動作について

イベントフラグの属性の中での、タスクの待ち行列について解説(FIFO順、タスク優先度順)

2.

X.

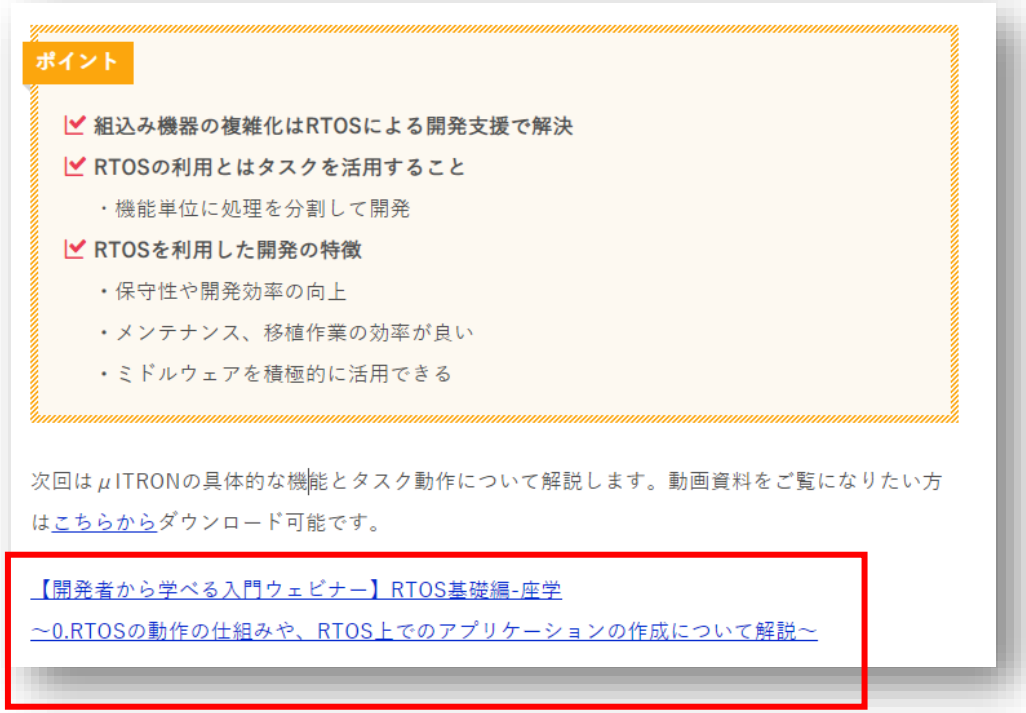
- タスクを連携させるためのサポート役
- タスク間協調処理の紹介

0. RTOSの動作の仕組みや、
RTOS上でのアプリケーションの作成について解説

- RTOSを利用した開発
- タスクの動作について解説

本ウェビナーについて

- **このWebinarを見ていて、もう少し導入編を聞いてみたい方や、復習をしたい方は、右のページにアクセスをお願いいたします。**



- **RTOS(リアルタイムOS)上でのアプリケーション作成の流れ**
- **RTOSの中で、μC3(弊社のRTOS製品)が採用しているμITRONの概要**
- **μITRONの動作の仕組み、特にタスクの動作について**

https://www.kumikomi.jp/rtos_beginner1/

https://www.kumikomi.jp/rtos_beginner2/

本ウェビナーについて

■ リアルタイムOS(RTOS)とは？は、こんな方にオススメです

- RTOSやμITRONの基礎について学んでみたい
- RTOS(特に、μITRON)を使ったSW開発を行う可能性があるので勉強しておきたい
- μC3(弊社のRTOS製品)がどんなものであるのか、概要について聴いてみたい

■ 今回のウェビナーで学んでほしいこと

- タスク間協調(同期)を利用したアプリケーション作成例
- タスク間協調処理のイベントフラグの動作について

1.本ウェビナーについて

2. μITRON4.0仕様のRTOS内での、イベントフラグの位置づけ

3. タスク間協調(同期)を利用したアプリケーション作成例

4. イベントフラグの紹介

ーシステムコールを利用した機能紹介と、属性の指定方法

ービットパターンを利用した基本動作の紹介

ーTA_WSGL又はTA_WMULの選択、TA_CLRの指定、イベントフラグの属性について

ータスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について

5. QA

µITRON4.0仕様準拠のµC3

	µC3 Compact	µC3 Standard	µC3 Standard+M	µC3 +Linux
仕様	**µITRON4.0** 自動車制御プロファイル	**µITRON4.0** スタンダードプロファイル		
追加仕様	-	-	マルチコア対応	OpenAMP対応
特長	極小フットプリント	高い割込み応答性能	AMP型のマルチコア対応	Linuxとの共存が可能
対応CPUコア	arm ・Cortex®-M0/M0+/M3/M4/M7/M33 Renesas ・RX intel ・Nios Ⅱ	arm ・Cortex®-M4/M7/M33, A5/A7/A8/A9/A15/A53/A73, R4/R5 Renesas ・Renesas RX 700	arm ・Arm Cortex®-A7/A9/A15/A53	arm ・Cortex®-A53/A53 ・Cortex®-A53/R5 ・Cortex®-A53/M4 ・Cortex®-A9/A9 ・Cortex®-A7/A7 ・Cortex®-A7/M4
主な 対応デバイス （詳しくは製品ガイドを参照下さい）	STMicro STM32 NXP Kinetis, LPCxxx Renesas RA, RX200/600/700 Intel Nios Ⅱ Cypress PSoC Sililabs EFR32 Xilinx Microblaze（planning）	Renesas RZ/A, RZ/T, RZ/N STMicro STM32 NXP i.MX RT, i.MX6/7/8 TI AM335x, 57x, 65x Intel SoC Xilinx Zynq-7000, MPSoC	Renesas RZ/G1E, G1N, G1M TI AM57x NXP i.MX6/7, LS1043A Xilixn Zynq-7000, MPSoC Intel SoC	Renesas RZ/G1E NXP i.MX6/7/8M Mini STMicro STM32MP157 TI AM65x Xilinx Zynq-7000, MPSoC Intel SoC

eForce

μITRON4.0仕様への対応状況

仕様

マイコン内蔵の小さなメモリだけで動作するように最適化

μITRON4.0のスタンダードプロファイルに準拠

	μC3/Standard	μC3/Compact
動的生成	○	×
タスク管理機能	○	○
タスク付属同期	○	○
タスク例外処理	×	×
同期・通信セマフォ	○	○
同期・通信イベントフラグ	○	○
同期・通信データキュー	○	○
同期・通信メールボックス	○	○
拡張同期・通信ミューテックス	○	×
拡張同期・通信メッセージバッファ	○	×
拡張同期・通信ランデブ	○	×
メモリプール管理固定長	○	○
メモリプール管理可変長	○	×
時間管理システム時刻管理	○	○
時間管理　周期ハンドラ	○	○
時間管理　アラームハンドラ	○	×
時間管理　オーバーランハンドラ	○	×
システム状態管理	○	○
割込み管理	○	○
サービスコール管理機能	×	×
システム構成管理機能	○	○
独自機能・デバイスドライバ管理機能	○	×

eForce

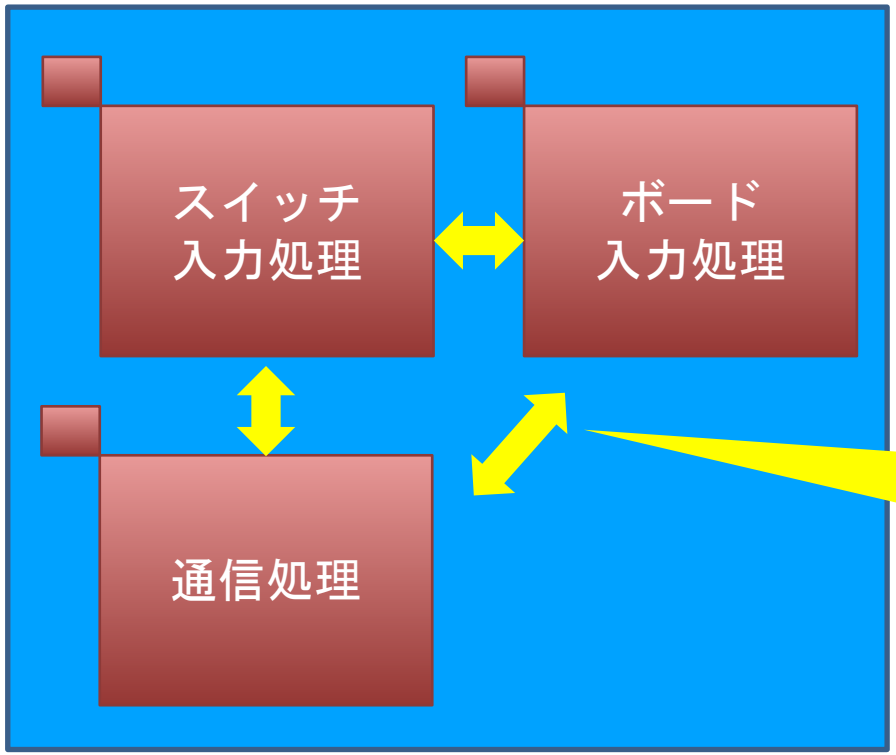
RTOSを利用して、機能毎に処理を分割

- タスクを利用して、機能毎に処理を分割
- タスク間を連携したシステム構築

- 機能のモジュール化
- 複数名での並列開発
- 機能単位での保守性・拡張性

RTOSなし

RTOSあり



タスク間協調処理

タスク間協調の紹介

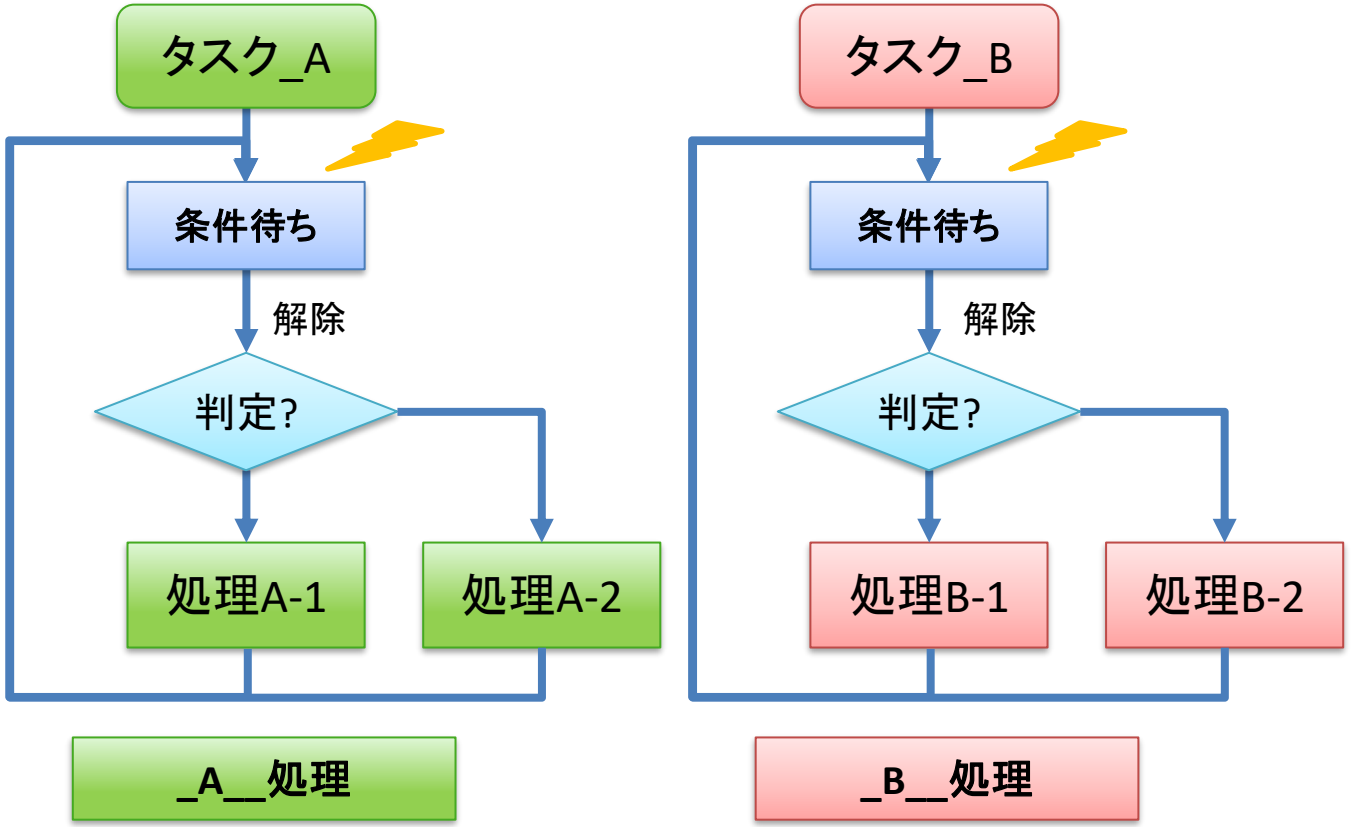
アプリケーション処理を構築するために、タスクを連携させるためのサポート役

- タスク間協調の紹介

種類	特徴	μITRONによる手段（例）
排他	資源待ち合わせ	セマフォ、(ミューテックス) など
同期	条件待ち合わせ	イベントフラグ
通信	タスク間でのデータ交換	データキュー、メールボックス、 (メッセージバッファ)

タスク処理での、さまざまな待ち処理

RTOSあり




```
void Task_xxx(VP_INT exinf)
{
        for(;;) {
            条件待ち;
            if 判定
                処理_xxx;
            else
                処理_xxx;
        }
}
```

- 条件待ちの例
  - スイッチ入力(割込み) : 同期
  - アクセス権獲得(資源) : 排他
  - 時間経過待ち(タイムアウト)

プログラム

1.本ウェビナーについて

2. μITRON4.0仕様のRTOS内での、イベントフラグの位置づけ

3. タスク間協調(同期)を利用したアプリケーション作成例

4. イベントフラグの紹介

ーシステムコールを利用した機能紹介と、属性の指定方法

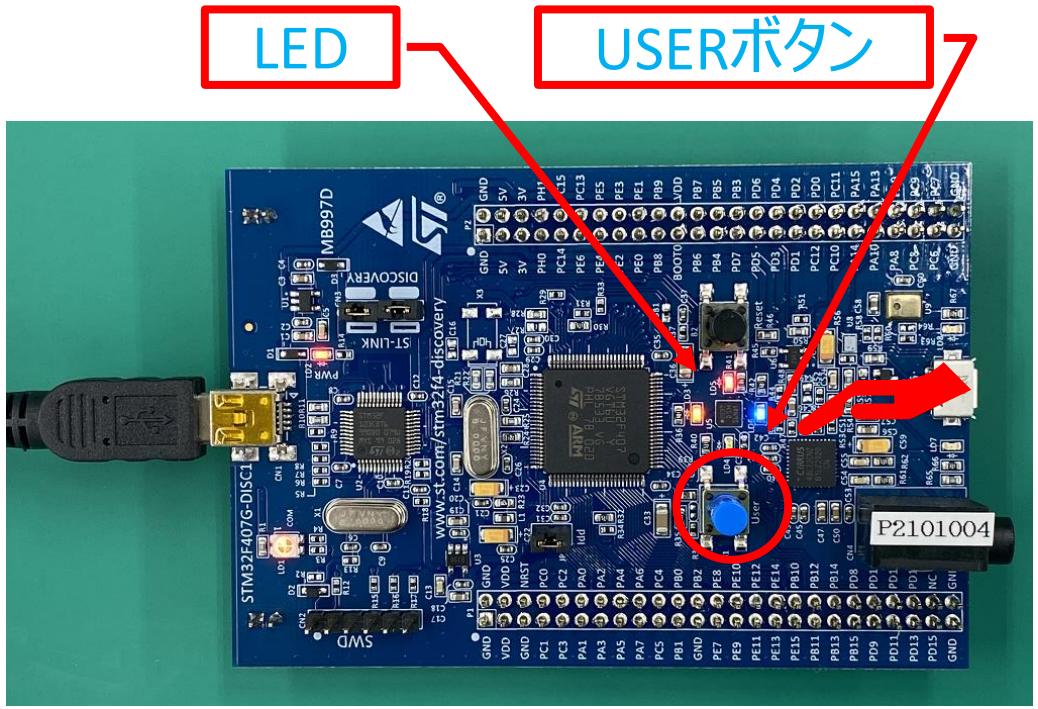
ービットパターンを利用した基本動作の紹介

ーTA_WSGL又はTA_WMULの選択、TA_CLRの指定、イベントフラグの属性について

ータスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について

5. QA

評価ボード上でのサンプル作成にRTOSを利用

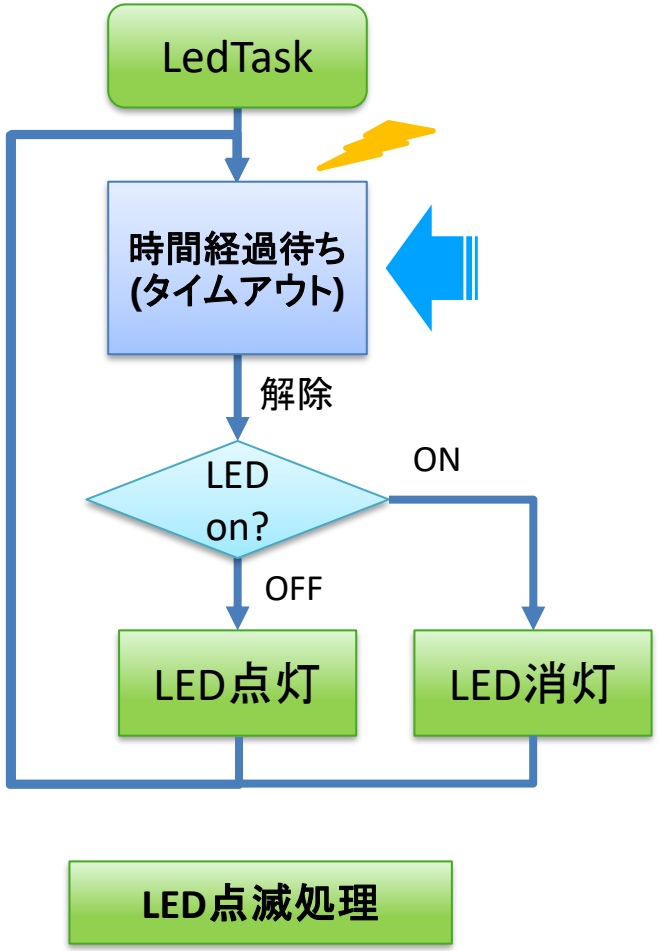


- USERボタンを利用したサンプル
  - ① LEDを点滅動作
    初期値: 1000msec間隔
  - ② USERボタン入力を受ける
  - ③ USERボタン入力で、LED点滅間隔を更新する
    1000msec -> 500msec -> 100msec -> 1000msecに戻る
- RTOSで表現する
  - a. 機能毎に、タスクで表現
  - b. ボタン入力の割り当て
  - c. 通知処理を割り当て

評価ボード上でのサンプル作成にRTOSを利用

LedTask
LED点滅処理

①

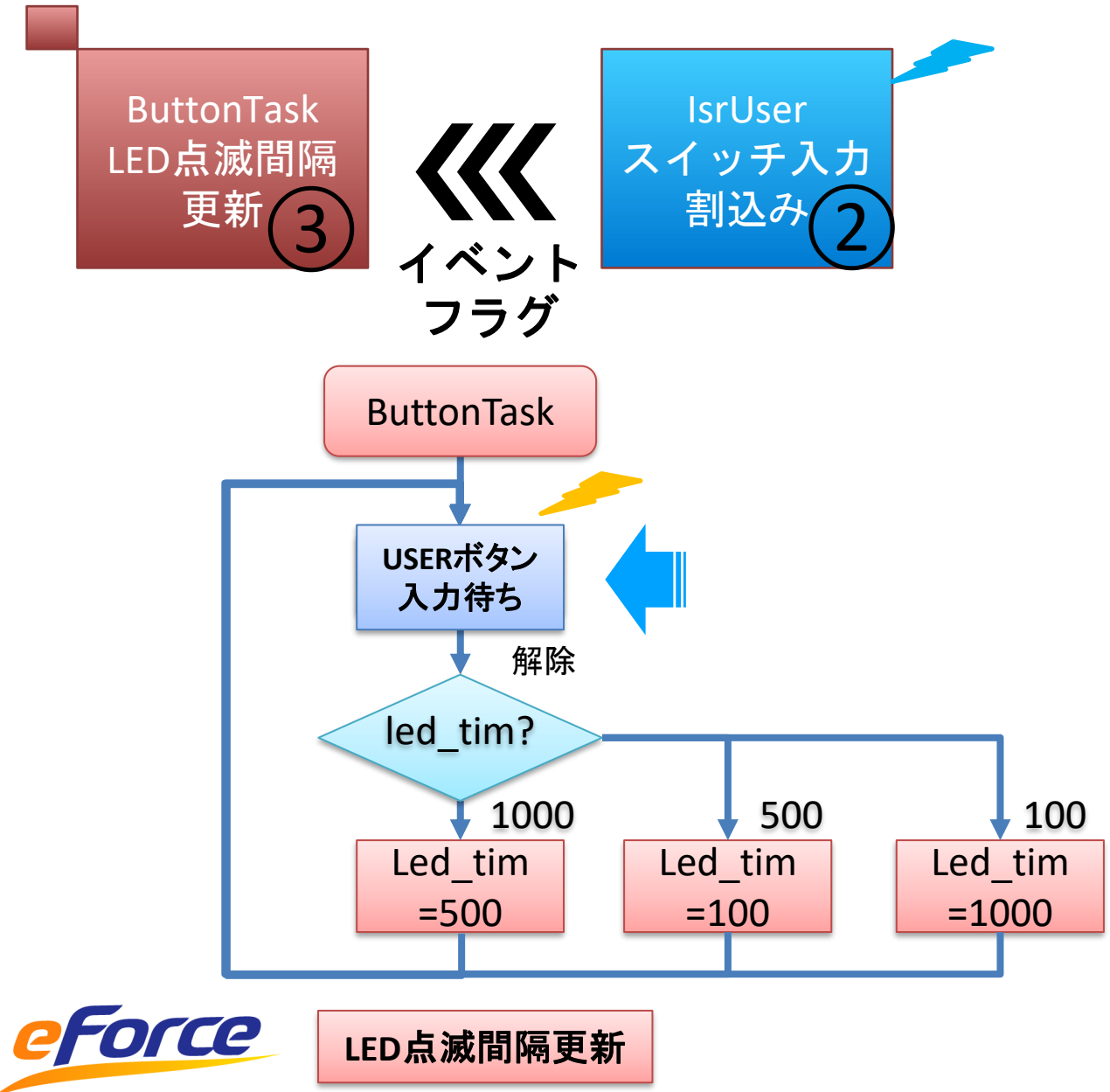


システムコール
アプリケーションから呼び出すAPI (RTOS機能)を示す

dly_tsk(dlytim): 自タスクの遅延
【解説】自タスクをシステムコールが呼び出された時刻から dlytim で指定される時間の間、時間経過待ち状態に遷移させます。

```
void LedTask(VP_INT exinf)
{
        led_tim = 1000; 初期値
        for(;;) {
            dly_tsk(led_tim); 時間経過待ち
            if (LED on?)
                LED消灯;
            else
                LED点灯;
        }
}
```

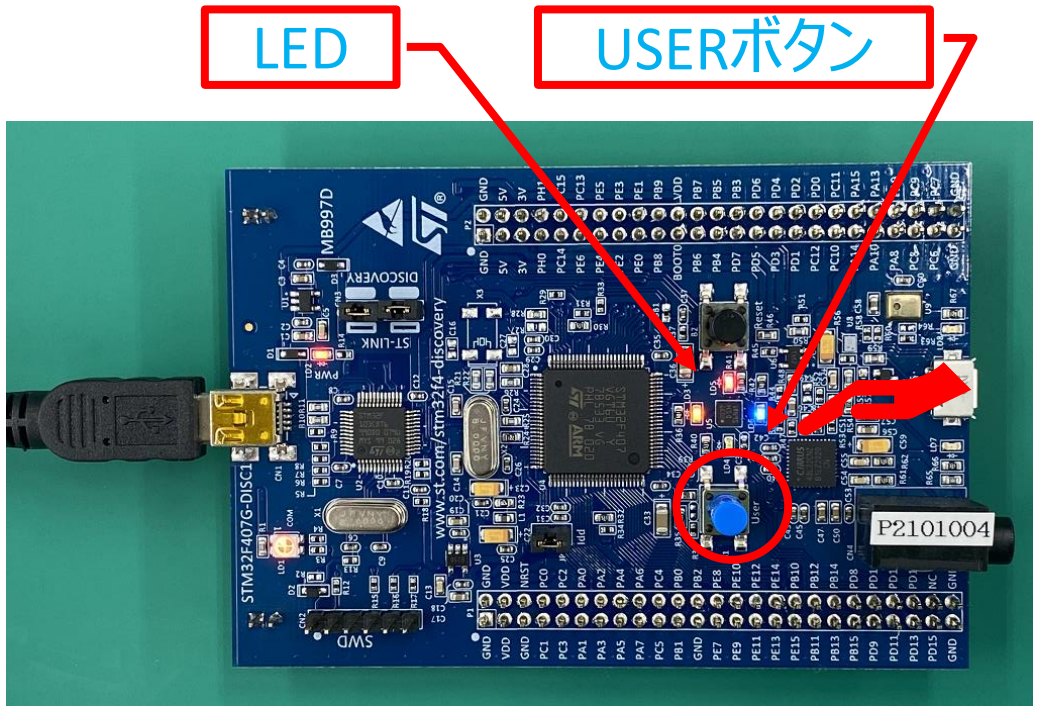
評価ボード上でのサンプル作成にRTOSを利用




```
void IsrUser(VP_INT exinf)
{
    iset_flg(1, 0x05);
}
```

```
void ButtonTask(VP_INT exinf)
{
    for(;;) {
        wai_flg(1, 0x05, TWF_ORW,&flgptn)
            同期待ち(スイッチ入力)
        switch (led_tim){
            case 1000;
                led_tim = 500;
                break;
            case 500;
                led_tim = 100;
                break;
            case 100;
                led_tim = 1000;
                break;
        }
    }
}
```

評価ボード上でのサンプル作成にRTOSを利用(振り返り)



- USERボタンを利用したサンプル
  - ① LEDを点滅動作
    初期値: 1000msec間隔
  - ② USERボタン入力を受ける
  - ③ USERボタン入力で、LED点滅間隔を更新する
    1000msec -> 500msec -> 100msec -> 1000msecに戻る
- RTOSで表現する
  a. 機能毎に、タスクで表現
  b. ボタン入力の割り当て
  c. 通知処理を割り当て

プログラム

1.本ウェビナーについて

2. μITRON4.0仕様のRTOS内での、イベントフラグの位置づけ

3. タスク間協調(同期)を利用したアプリケーション作成例

4. イベントフラグの紹介

ーシステムコールを利用した機能紹介と、属性の指定方法

ービットパターンを利用した基本動作の紹介

ーTA_WSGL又はTA_WMULの選択、TA_CLRの指定、イベントフラグの属性について

ータスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について

5. QA

システムコールを利用した機能紹介

- イベントフラグは、イベントの有無をビット毎のフラグで表現することにより、同期を行うためのオブジェクト

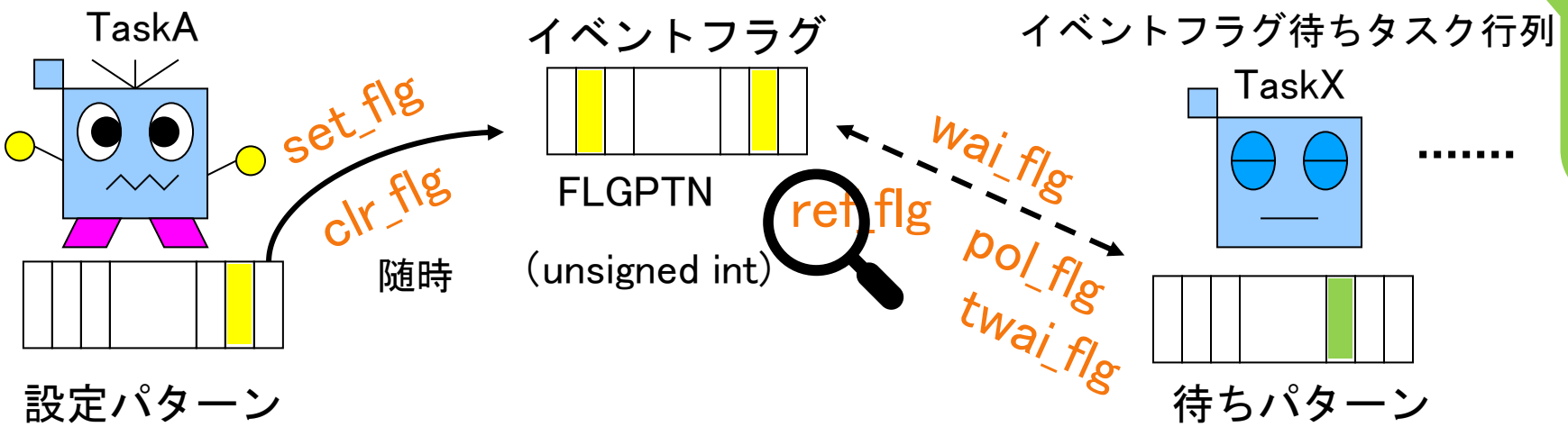
イベントフラグを生成する(uC3/Standard)	cre_flg, acre_flg
イベントフラグを削除する(uC3/Standard)	del_flg
イベントフラグをセットする	set_flg,iset_flg
イベントフラグをクリアする	clr_flg
イベントフラグで待つ	wai_flg,pol_flg,twai_flg
イベントフラグの状態を参照	ref_flg

ixxx_yyy

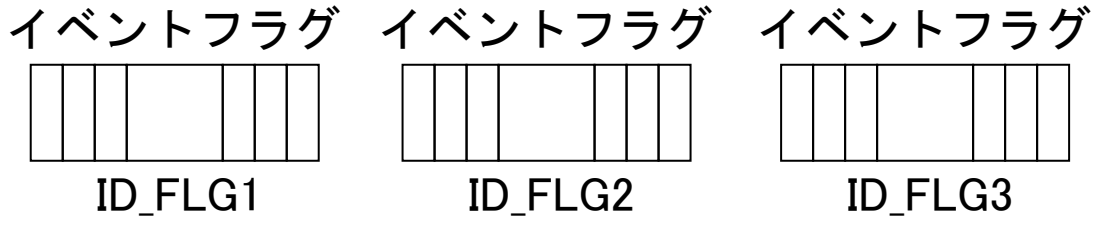
タスクコンテキスト以外から使用するシステムコール。

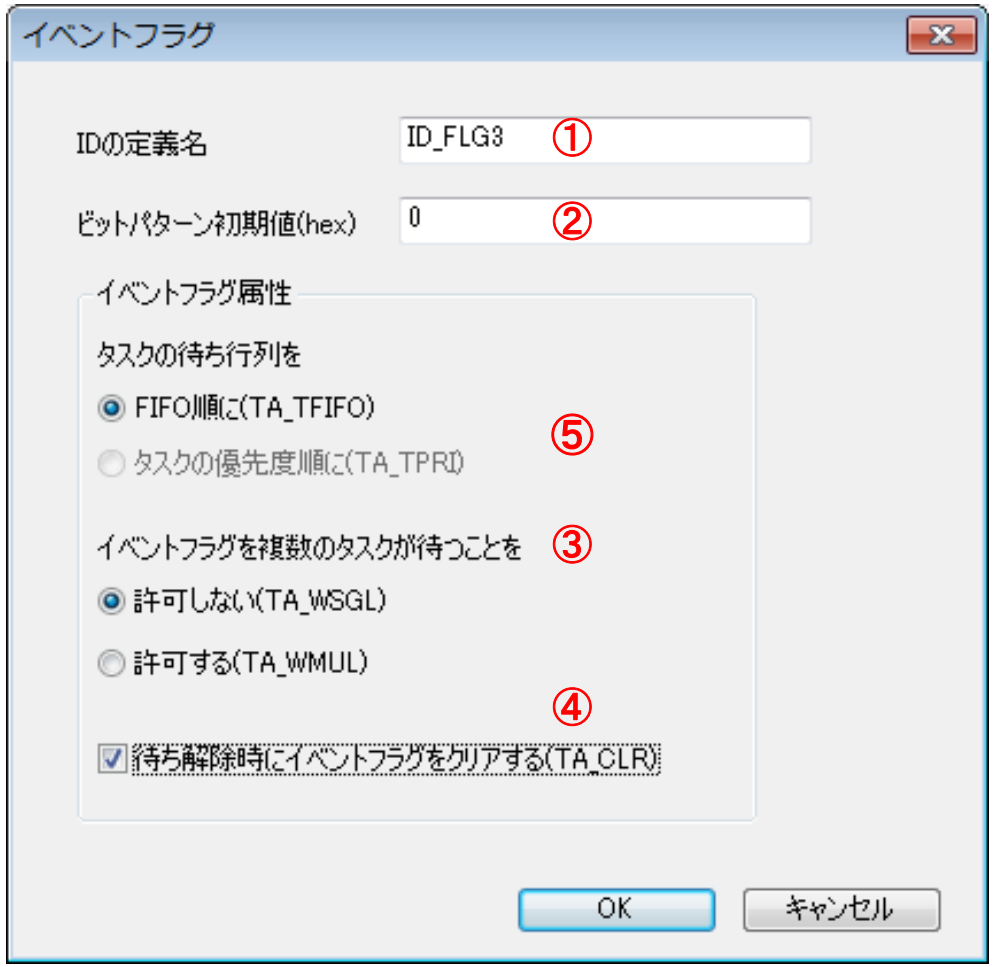
pxxx_yyy
ポーリング

txxx_yyy
タイムアウト



イベントフラグへの属性の指定方法





① イベントフラグのID番号を表す任意の定義名を指定します。この定義名は例として、kernel_id.x 内でマクロ定義されます。

② **イベントフラグの初期値**を16進数で指定します。

③ **複数のタスクを待ち状態に出来る**のかの属性を指定します。

④ **チェックするとTA_CLR属性がON**になり、タスクがイベントフラグ待ちから、条件成立により待ち解除される時に、ビットパターンの全てのビットをクリアします。

⑤ **タスクの待ち行列を**指定します。

プログラム

1.本ウェビナーについて

2. μITRON4.0仕様のRTOS内での、イベントフラグの位置づけ

3. タスク間協調(同期)を利用したアプリケーション作成例

4. イベントフラグの紹介

ーシステムコールを利用した機能紹介と、属性の指定方法

ービットパターンを利用した基本動作の紹介

ーTA_WSGL又はTA_WMULの選択、TA_CLRの指定、イベントフラグの属性について

ータスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について

5. QA

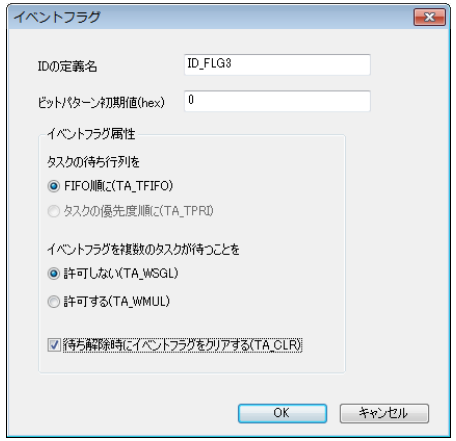
ビットパターンを利用した基本動作の紹介

- ビットパターンの利用

1. wai_flgでの、待ち状態に設定したタスクの解除条件

- OR条件の指定
- AND条件の指定

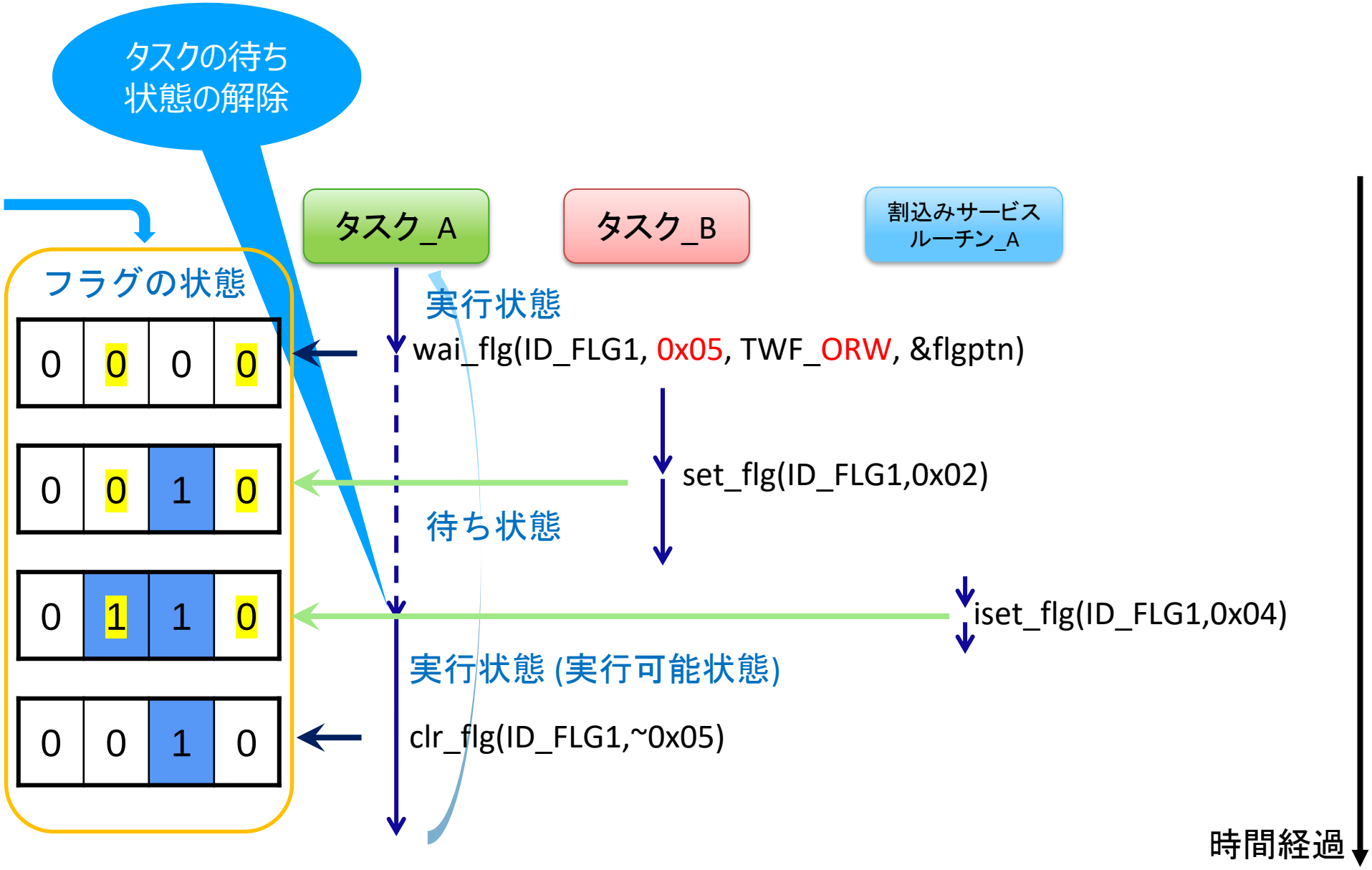
OR条件の指定



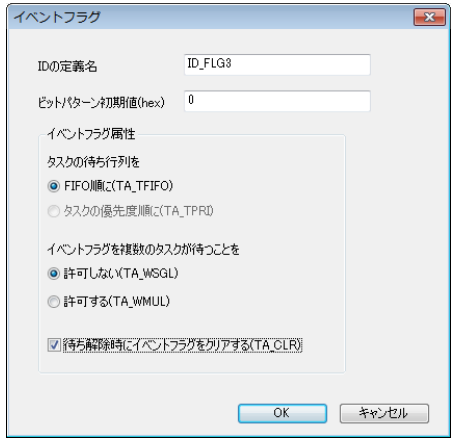
ID定義名: ID_FLG1

ビットパターン: 0x00

TA_CLR: 指定なし



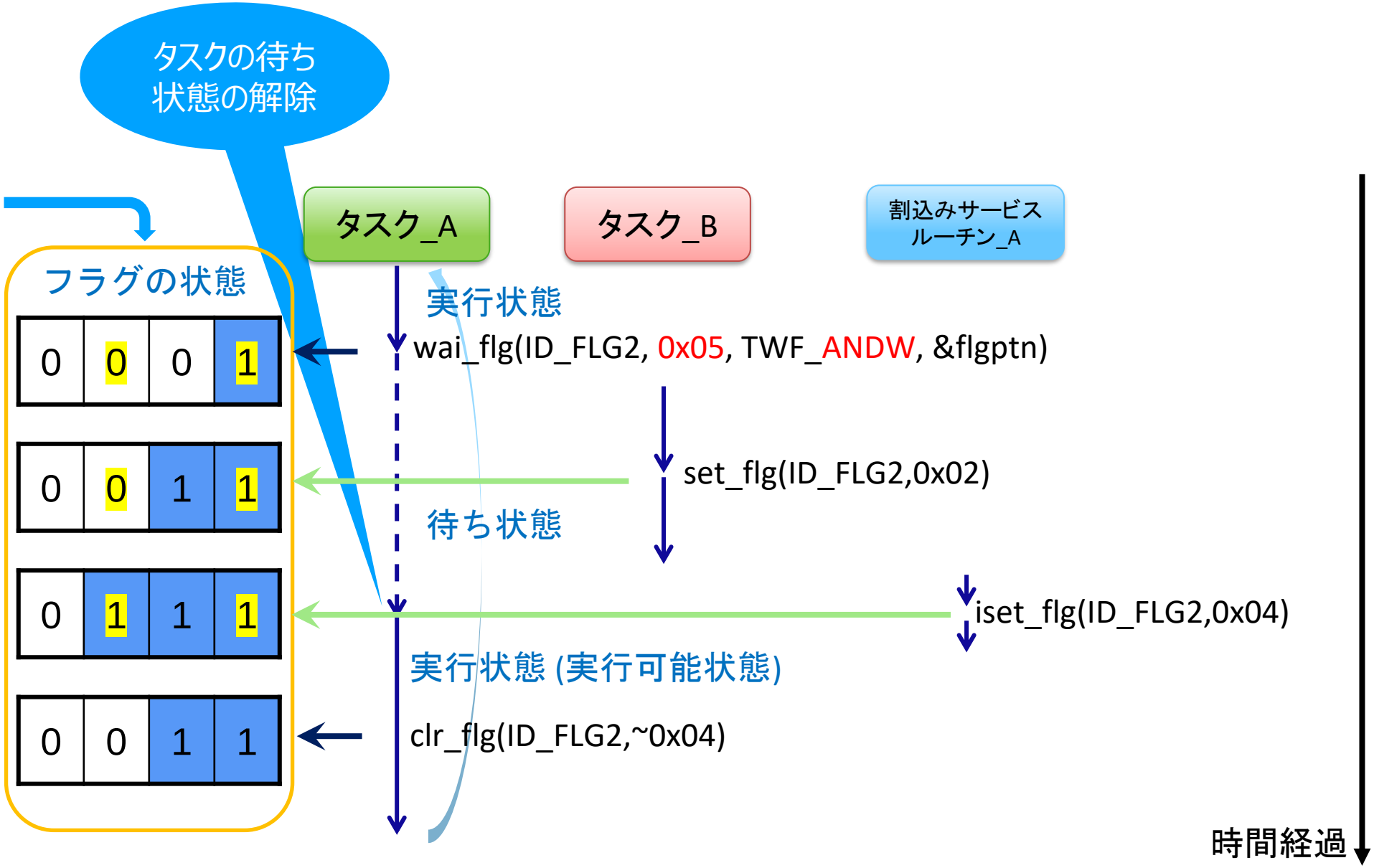
AND条件の指定



ID定義名: ID_FLG2

ビットパターン: 0x01

TA_CLR: 指定なし



プログラム

1.本ウェビナーについて

2. μITRON4.0仕様のRTOS内での、イベントフラグの位置づけ

3. タスク間協調(同期)を利用したアプリケーション作成例

4. イベントフラグの紹介

ーシステムコールを利用した機能紹介と、属性の指定方法

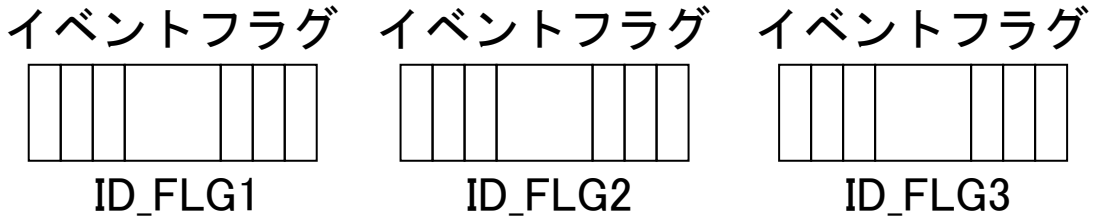
ービットパターンを利用した基本動作の紹介

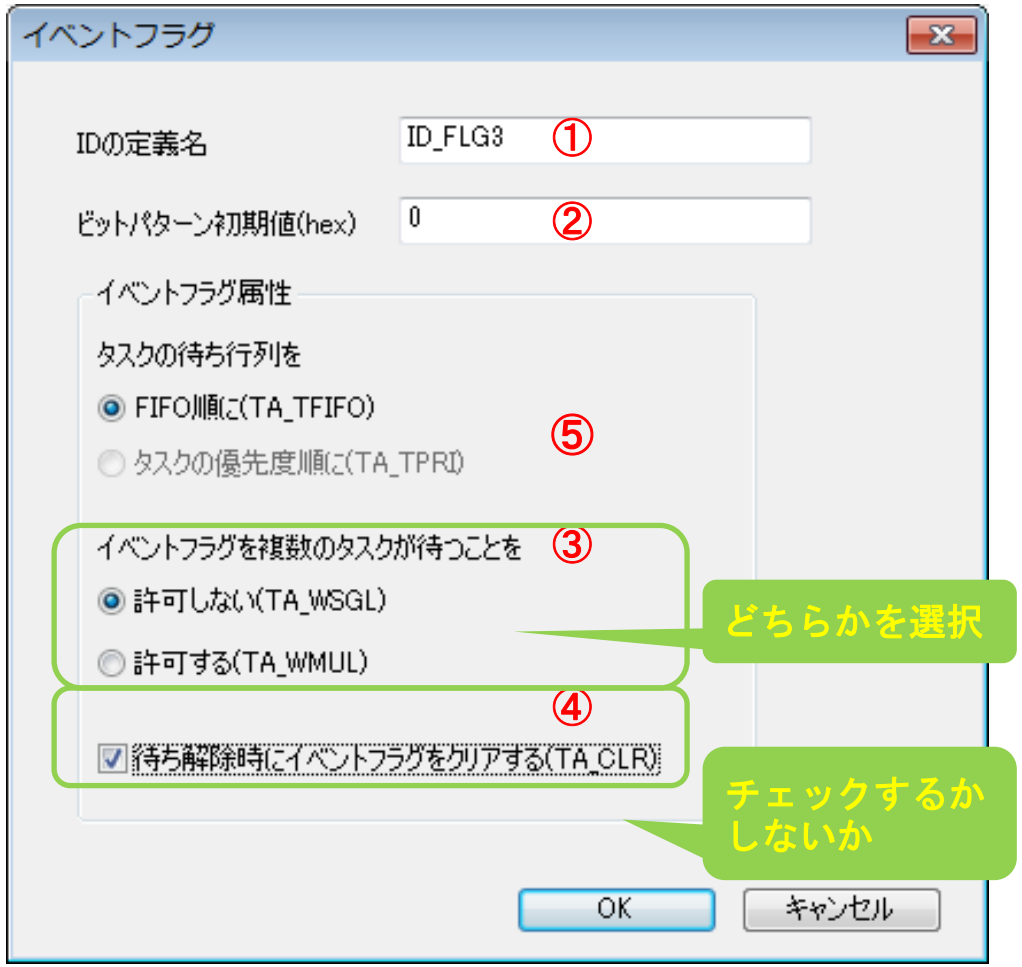
ーTA_WSGL又はTA_WMULの選択、TA_CLRの指定、イベントフラグの属性について

ータスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について(一回目QA後)

5. QA

イベントフラグへの属性の指定方法と、その動作について





①
イベントフラグのID番号を表す任意の定義名を指定します。この定義名は例として、kernel_id.x 内でマクロ定義されます。

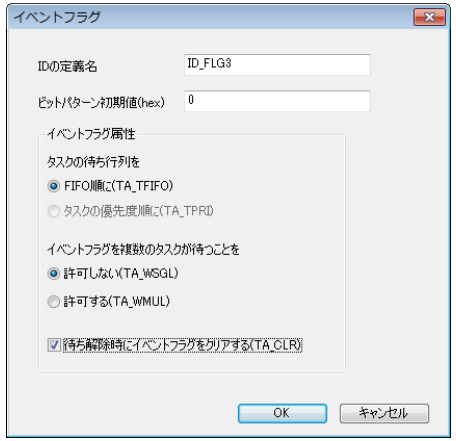
②
イベントフラグの初期値を16進数で指定します。

③
複数のタスクを待ち状態に出来るのかの属性を指定します。

④
チェックするとTA_CLR属性がONになり、タスクがイベントフラグ待ちから、条件成立により待ち解除される時に、ビットパターンの全てのビットをクリアします。

⑤
タスクの待ち行列を指定します。

属性に、TA_CLR指定あり

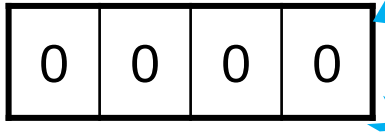


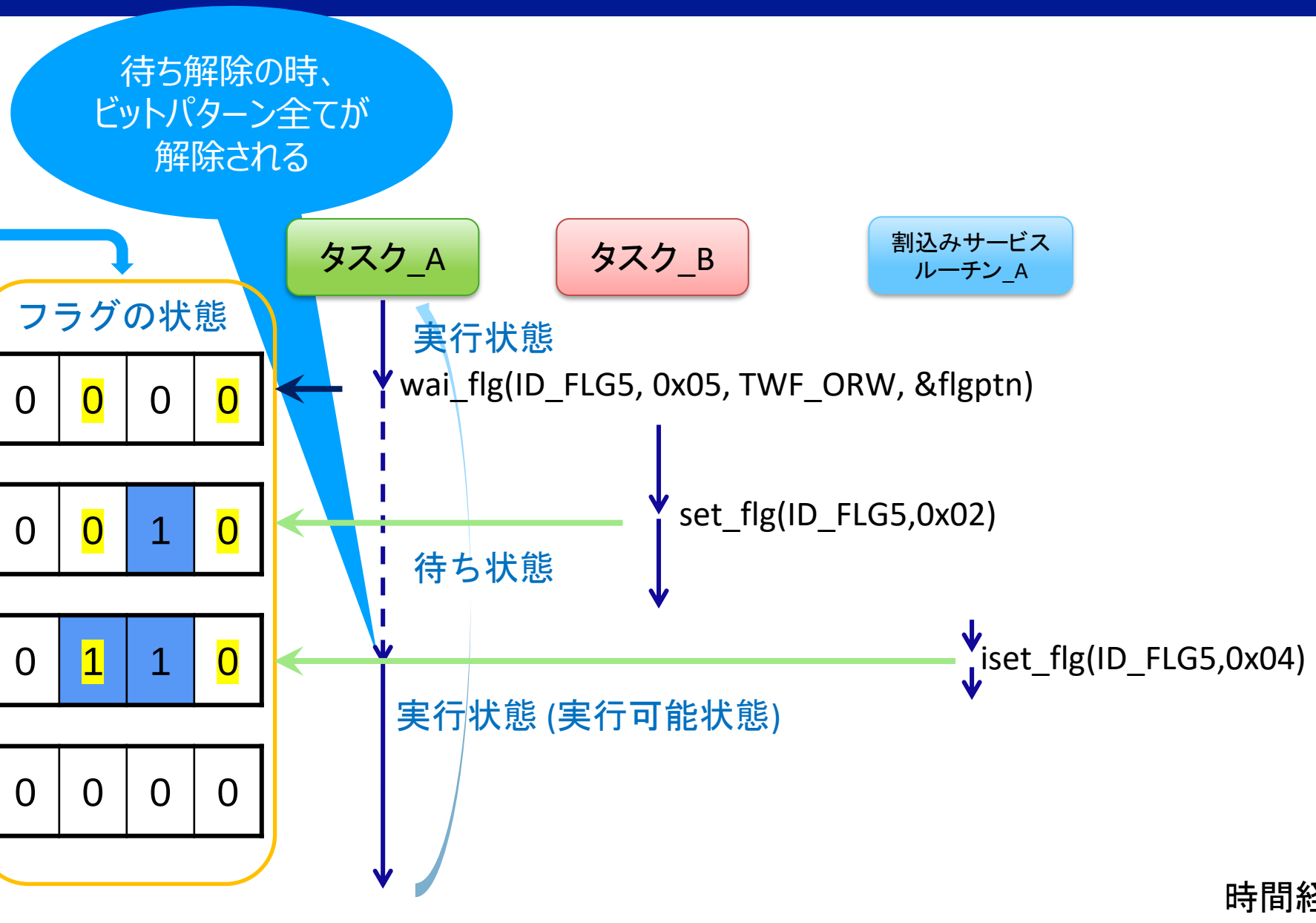
ID定義名: ID_FLG5

ビットパターン: 0x00

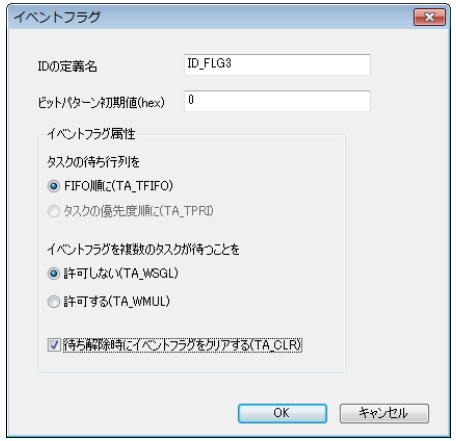
TA_WSGL指定

TA_CLR:指定あり





属性に、TA_WSGL条件の指定

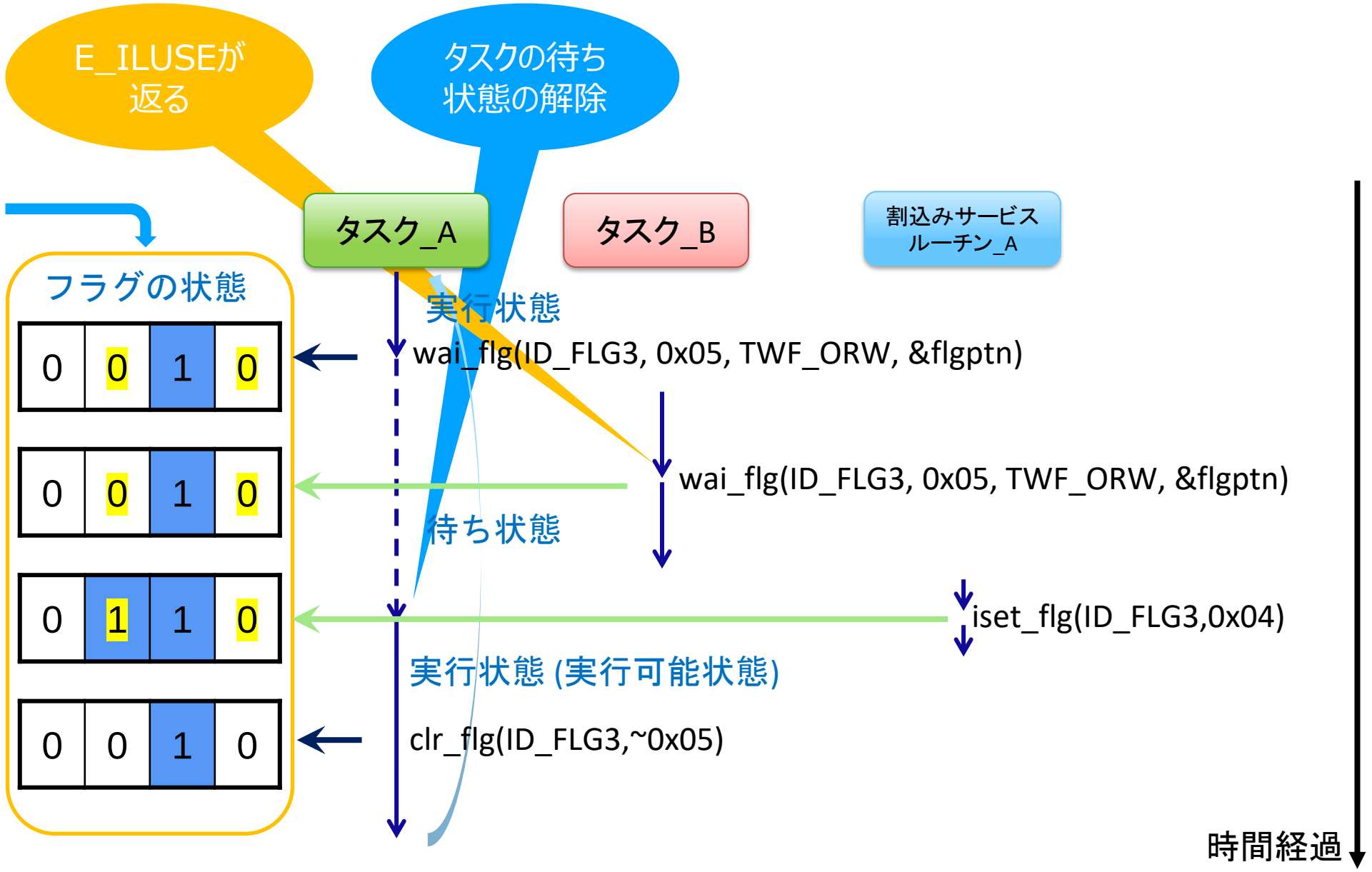


ID定義名: ID_FLG3

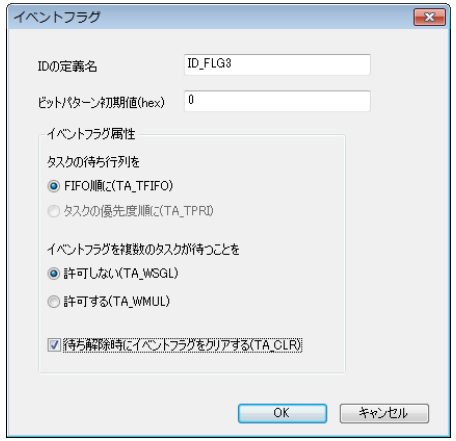
ビットパターン: 0x02

TA_WSGL指定

TA_CLR:指定なし



属性に、TA_WMUL条件の指定

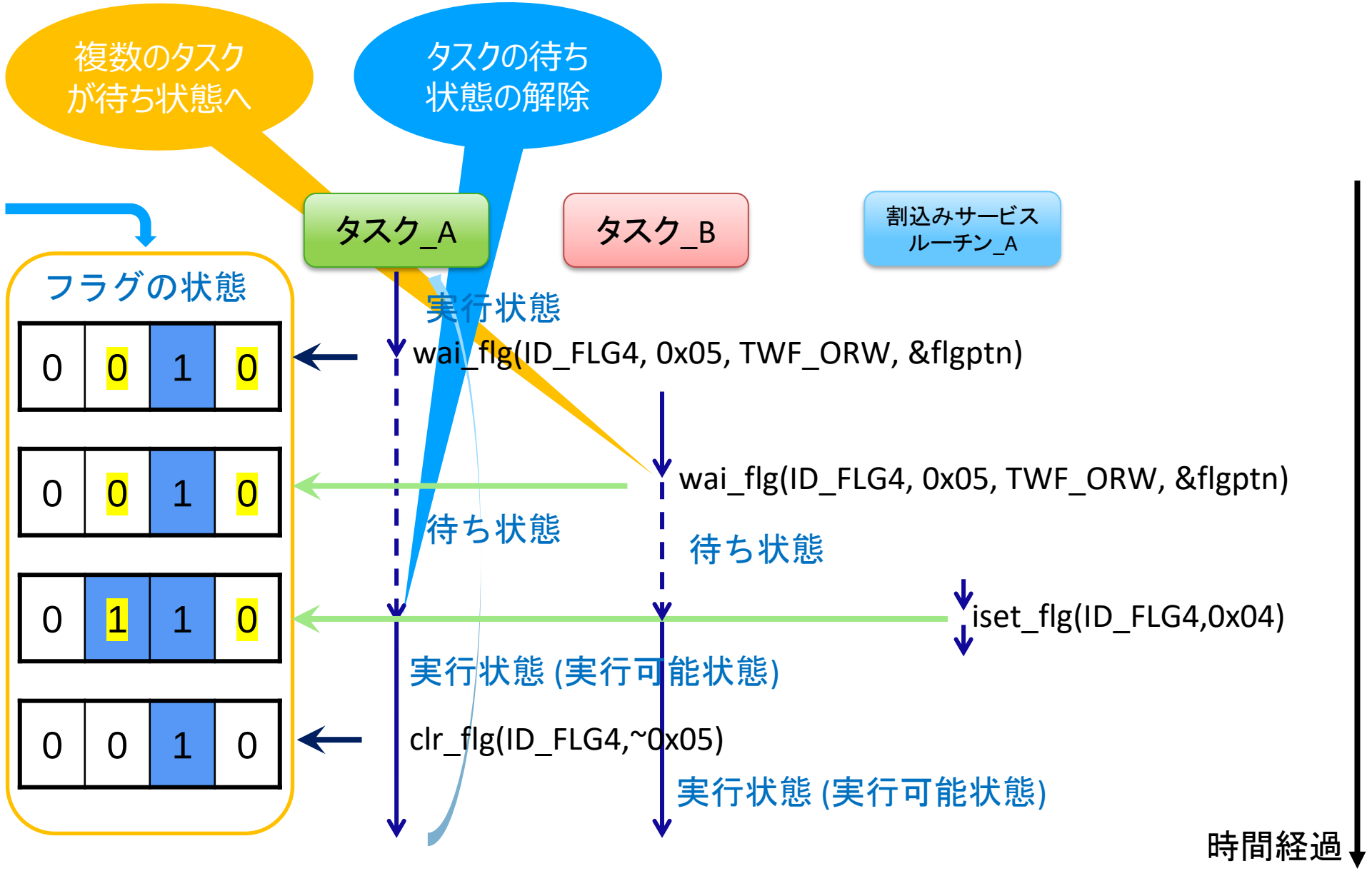


ID定義名: ID_FLG4

ビットパターン: 0x02

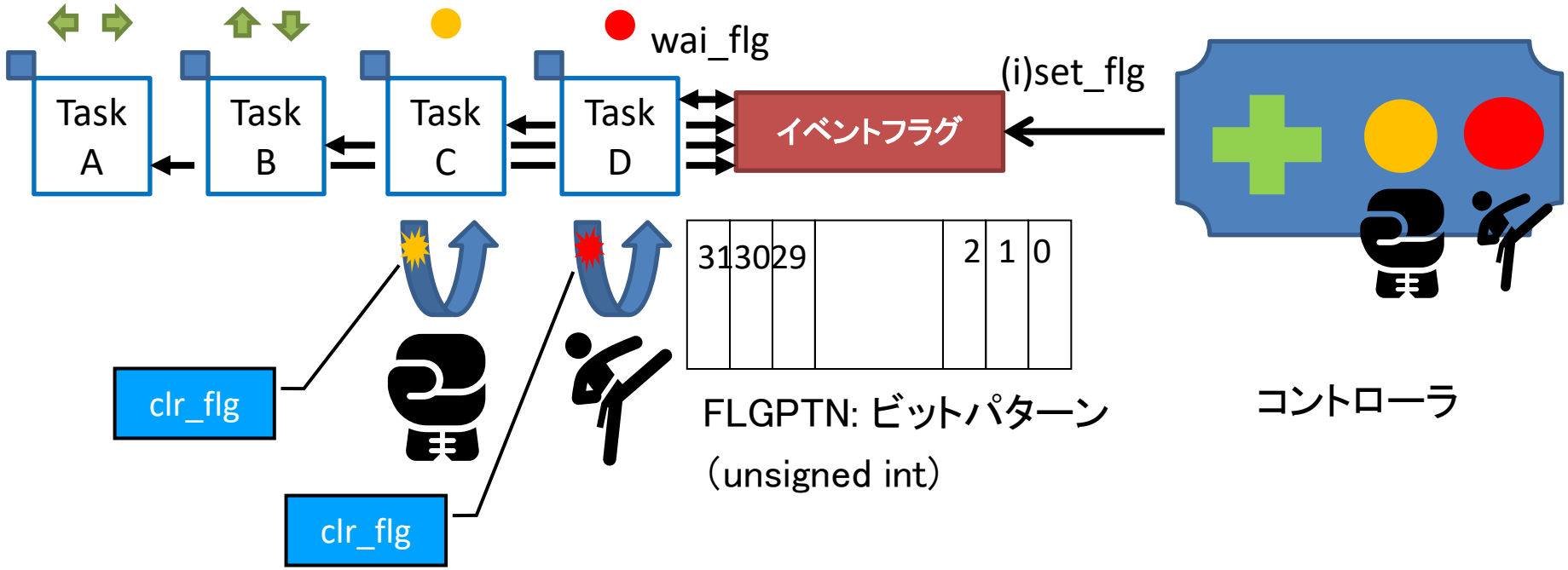
TA_WMUL指定

TA_CLR:指定なし



まとめ

イベントフラグを利用した、条件に応じた処理の実行



- イベントフラグ

- イベントフラグを利用して、コントローラのボタンの情報を通知
- 受け取り側で、ボタンに応じた処理の構築が可能

セミナーの振り返り

■ 評価ボード上でのサンプル作成にRTOSを利用

- タスク処理での、条件待ちを利用してみました
- イベントの発生を、イベントフラグを利用して、タスクに伝える処理を紹介

■ イベントフラグの紹介

- ビットパターンを利用した基本動作の紹介
- イベントフラグに設定できる属性や、属性による動作について
  - 複数のタスクが待つことの許可
  - 待ち解除時に、イベントフラグをクリアする
  - タスクの待ち行列の指定 (FIFO順・タスク優先度順)

プログラム

1.本ウェビナーについて

2. μITRON4.0仕様のRTOS内での、イベントフラグの位置づけ

3. タスク間協調(同期)を利用したアプリケーション作成例

4. イベントフラグの紹介

ーシステムコールを利用した機能紹介と、属性の指定方法

ービットパターンを利用した基本動作の紹介

ーTA_WSGL又はTA_WMULの選択、TA_CLRの指定、イベントフラグの属性について

ータスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について

5. QA(1回目）

『RTOSとは？イベントフラグ編』

イベントフラグ属性の中での、タスクの待ち
行列について解説(FIFO順、タスク優先度順)

イー・フォース株式会社　田村　聡

RTOSとは？ と題した、Webinarについて

一連のWebinarでは、RTOS(特に、μITRON)について解説をします

RTOSとは？

1.イベントフラグ編

タスク間協調(同期)を利用した、アプリケーション作成例とイベントフラグの動作について

イベントフラグの属性の中での、タスクの待ち行列について解説(FIFO順、タスク優先度順)

2.

X.

- タスクを連携させるためのサポート役
- タスク間協調処理の紹介

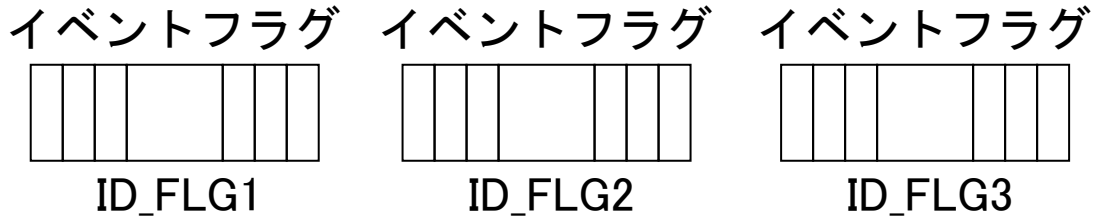
0. RTOSの動作の仕組みや、
RTOS上でのアプリケーションの作成について解説

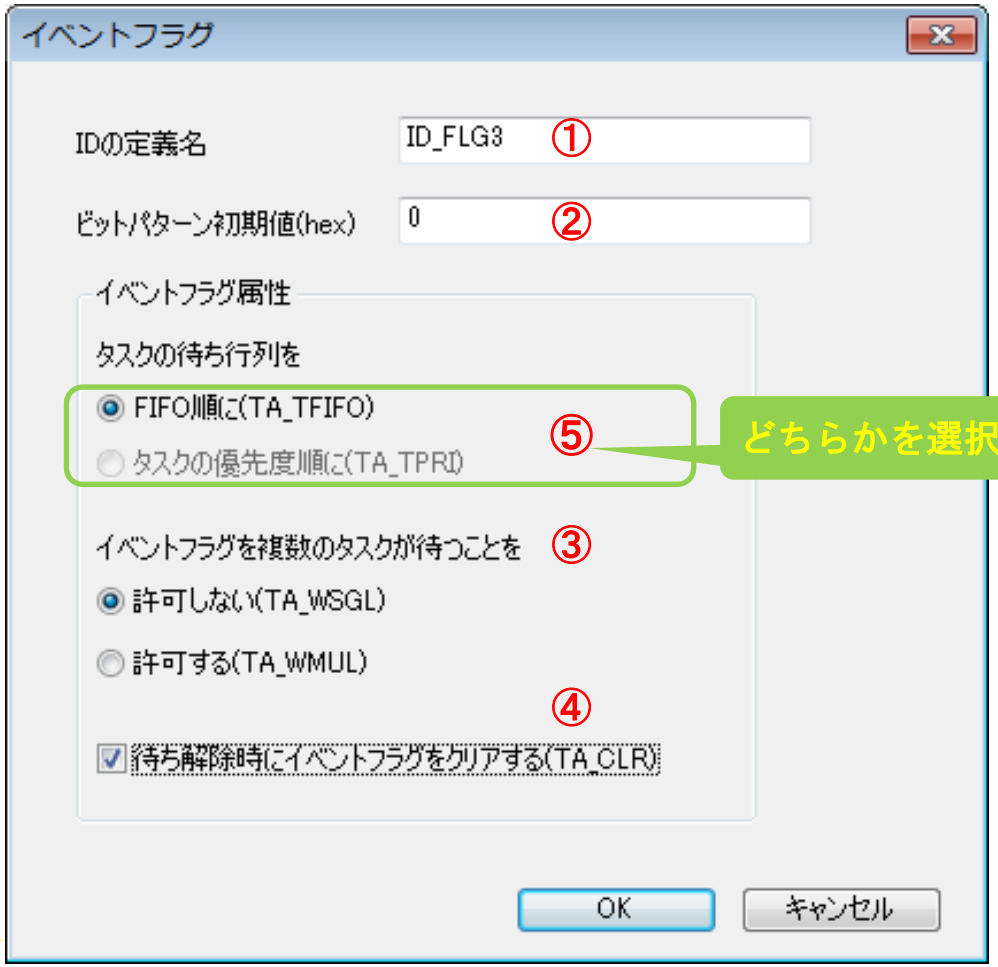
- RTOSを利用した開発
- タスクの動作について解説

プログラム

4. イベントフラグの紹介
   - ータスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について
   - ー補足
5. QA(2回目)

イベントフラグへの属性の指定方法、その動作について





① イベントフラグのID番号を表す任意の定義名を指定します。この定義名は例として、kernel_id.x 内でマクロ定義されます。

② **イベントフラグの初期値**を16進数で指定します。

③ **複数のタスクを待ち状態に出来る**のかの属性を指定します。

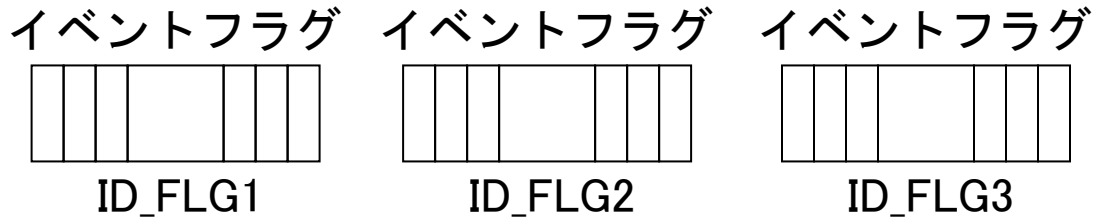
④ **チェックするとTA_CLR属性がON**になり、タスクがイベントフラグ待ちから、条件成立により待ち解除される時に、ビットパターンの全てのビットをクリアします。

⑤ **タスクの待ち行列を**指定します。

タスク待ち行列に、TA_TFIFO又はTA_TPRIの選択について

※. イベントフラグに、属性を指定する

->イベントフラグ毎に、性質を変化させる



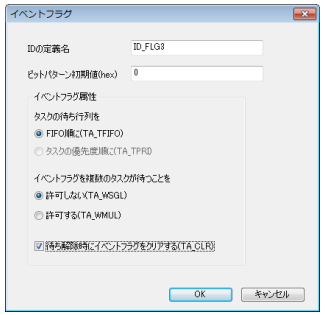
① タスクの待ち行列を、FIFO順に (TA_TFIFO)

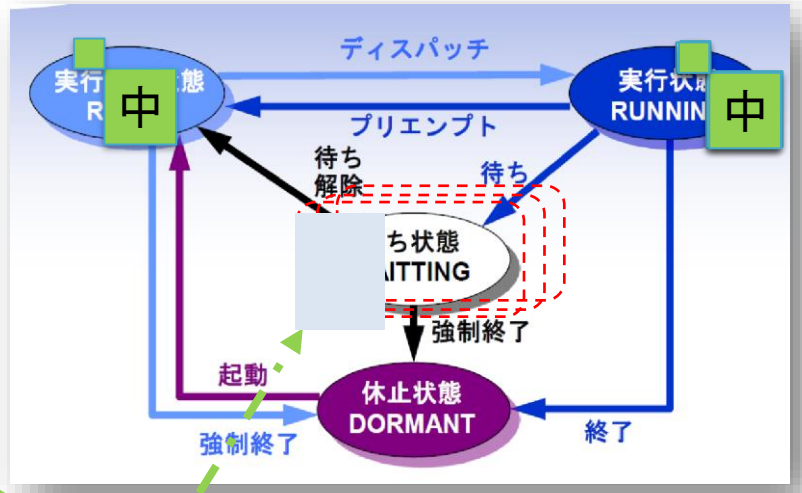
② タスクの待ち行列を、タスクの優先度順に (TA_TPRI) ※μC3/Standardで利用可

TA_CLRの
指定あり・なしで、
動作に違いが表れる

	TA_TFIFO	TA_TPRI
TA_CLR 指定あり	条件1	条件2
TA_CLR 指定なし	条件3	条件4

条件1 [TA_CLR指定あり、TA_TFIFO]

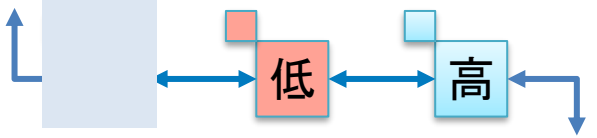




ID_FLG6に関して、
待ち状態の解除順

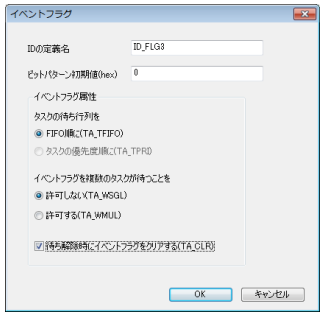
先 後

優先順位
高



優先順位
低

条件2 [TA_CLR指定あり、TA_TPRI]



ID定義名: ID_FLG7
ビットパターン: 0x02
TA_WMUL指定
TA_CLR:指定あり
タスク待ち行列:TA_TPRI

TA_TPRI

待ち解除の時、
ビットパターン全てが
解除される

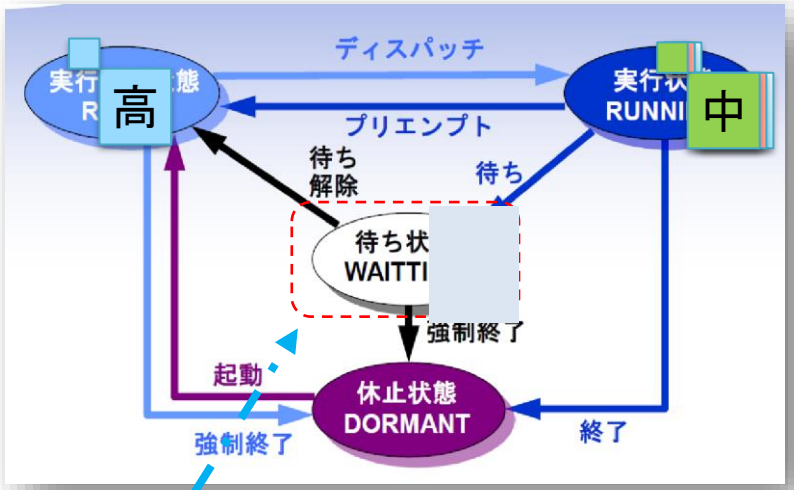
タスク_中 | タスク_低 | タスク_高 | 割込みサービスルーチン_A

フラグの状態

0	0	1	0
0	0	1	0
0	0	1	0
0	1	1	0

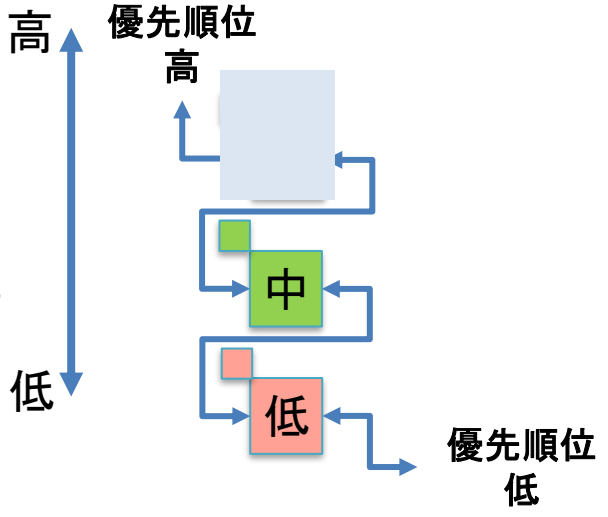
0	0	0	0

実行状態
wai_flg(ID_FLG7, 0x05, TWF_ORW, &flgptn)
待ち状態
実行状態
wai_flg(ID_FLG7, 0x05, TWF_ORW, &flgptn)
待ち状態
実行状態
wai_flg(ID_FLG7, 0x05, TWF_ORW, &flgptn)
待ち状態
iset_flg(ID_FLG7, 0x04)
待ち状態
実行状態
(実行可能状態)
時間経過

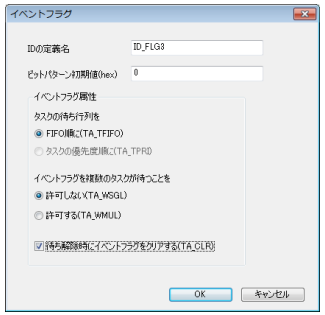


ID_FLG7に関して、
待ち状態の解除順

先 後



条件3 [TA_CLR指定なし、TA_TFIFO]



ID定義名: ID_FLG8
ビットパターン: 0x02
TA_WMUL指定
TA_CLR:指定なし
タスク待ち行列:TA_TFIFO

TA_TFIFO

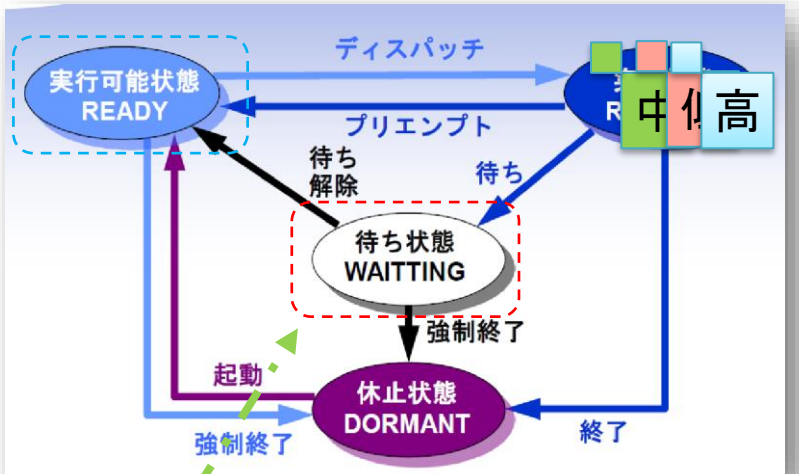
3つのタスクの待ち状態が解除される

タスク_中　タスク_低　タスク_高　割込みサービスルーチン_A

フラグの状態

0	0	1	0
0	0	1	0
0	0	1	0
0	1	1	0

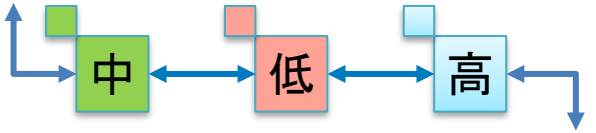
実行状態
wai_flg(ID_FLG8, 0x05, TWF_ORW, &flgptn)
待ち状態
実行状態
wai_flg(ID_FLG8, 0x05, TWF_ORW, &flgptn)
待ち状態
実行状態
wai_flg(ID_FLG8, 0x05, TWF_ORW, &flgptn)
待ち状態
iset_flg(ID_FLG8, 0x04)
時間経過



ID_FLG8に関して、
待ち状態の解除順

先　後

優先順位
高



優先順位
低

条件3 [TA_CLR指定なし、TA_TFIFO]

TA_CLR:指定なし
タスク待ち行列:TA_TFIFO

TA_TFIFO

3つのタスクの待ち状態が解除される

フラグの状態

0	1	1	0
0	0	1	0

タスク_中　タスク_低　タスク_高

割込みサービスルーチン_A

待ち状態

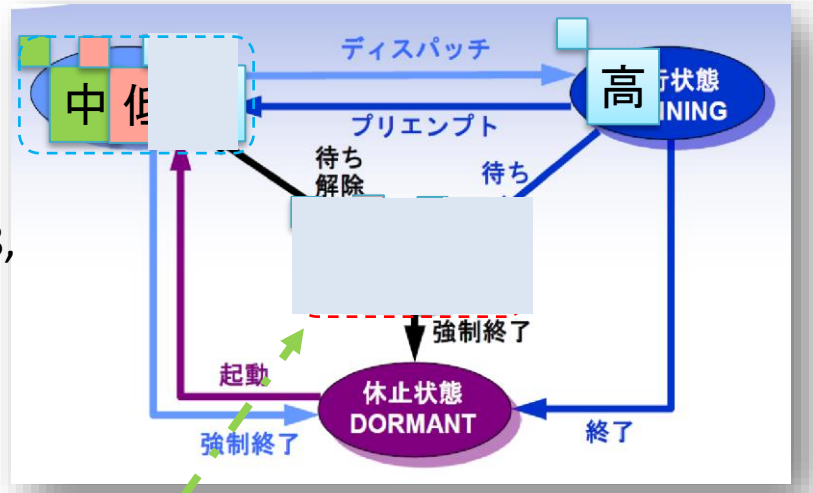
実行可能状態

実行状態

iset_flg(ID_FLG8, 0x04)

clr_flg(ID_FLG8,~0x05)

時間経過



実行できる状態になった順
レディーキュー

先　後

高　優先順位 高

中

低　優先順位 低

低

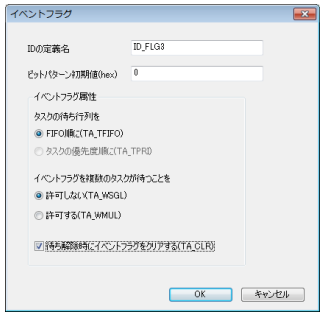
ID_FLG8に関して、
待ち状態の解除順

先　後

優先順位 高

優先順位 低

条件4 [TA_CLR指定なし、TA_TPRI]



TA_TPRI

3つのタスクの待ち状態が解除される

タスク_中　タスク_低　タスク_高　割込みサービスルーチン_A

フラグの状態

0	0	1	0
0	0	1	0
0	0	1	0
0	1	1	0

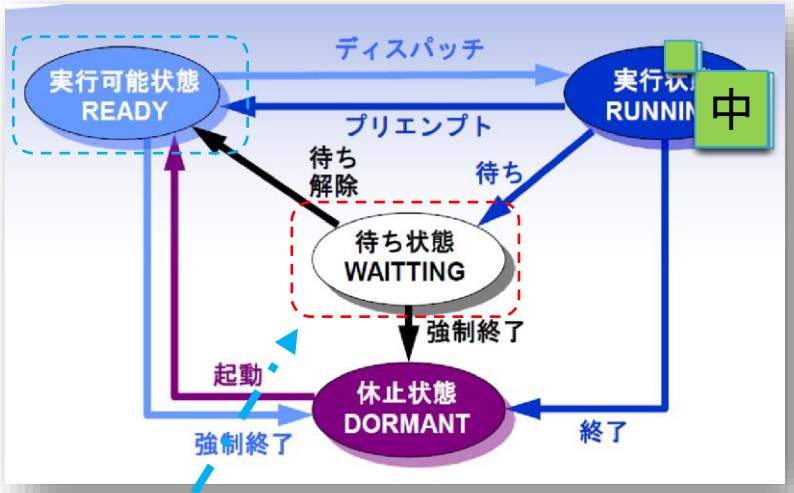
実行状態
wai_flg(ID_FLG9, 0x05, TWF_ORW, &flgptn)

待ち状態　実行状態
wai_flg(ID_FLG9, 0x05, TWF_ORW, &flgptn)

待ち状態　実行状態
wai_flg(ID_FLG9, 0x05, TWF_ORW, &flgptn)

待ち状態
iset_flg(ID_FLG9, 0x04)

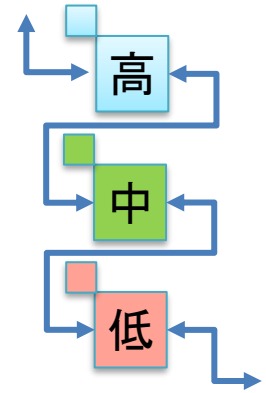
時間経過



ID_FLG9に関して、待ち状態の解除順

先　後

高　優先順位



低　優先順位

eForce

条件4 [TA_CLR指定なし、TA_TPRI]

TA_CLR:指定なし
タスク待ち行列:TA_TPRI

TA_TPRI

3つのタスクの待ち状態が解除される

フラグの状態

0	1	1	0
0	0	1	0

タスク_中　タスク_低　タスク_高

割込みサービスルーチン_A

待ち状態

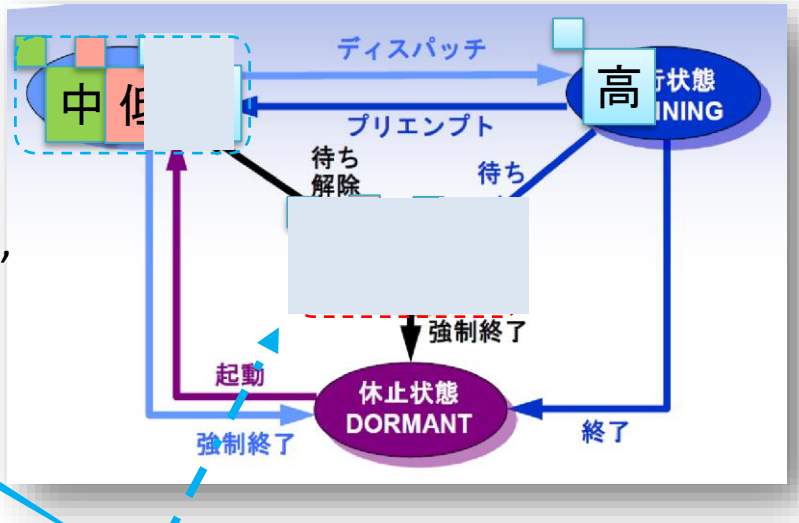
実行可能状態

実行状態

iset_flg(ID_FLG9, 0x04)

clr_flg(ID_FLG9,~0x05)

時間経過



実行できる状態になった順
レディーキュー

先　後

高　優先順位 高

中

低

低　優先順位 低

ID_FLG9に関して、
待ち状態の解除順

先　後

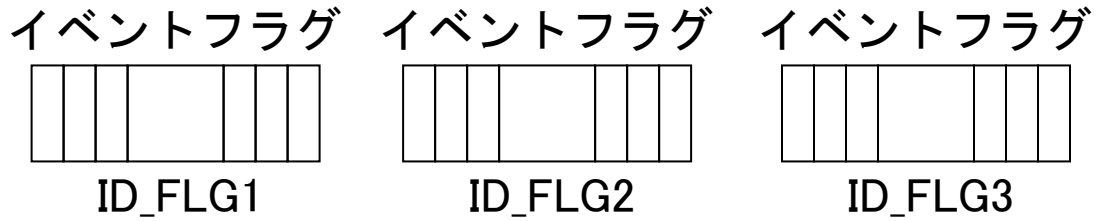
高　優先順位 高

低　優先順位 低

タスク待ち行列に、TA_TFIFO又はTA_TPRIの選択について(振り返り)

※. イベントフラグに、属性を指定する

->イベントフラグ毎に、性質を変化させる



① タスクの待ち行列を、FIFO順に (TA_TFIFO)

② タスクの待ち行列を、タスクの優先度順に (TA_TPRI) ※μC3/Standardで利用可

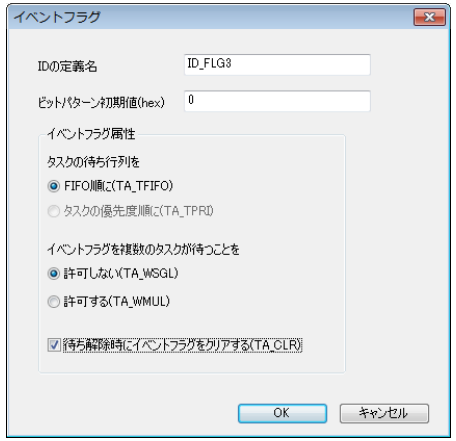
どのタスクが、待ち解除されるか

	TA_TFIFO	TA_TPRI
TA_CLR 指定あり	条件1	条件2
TA_CLR 指定なし	条件3	条件4

プログラム

4. イベントフラグの紹介
   - タスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について
   - 補足
5. QA(2回目)

属性に、TA_WSGL条件の指定の時、TA_TPRI指定は不要



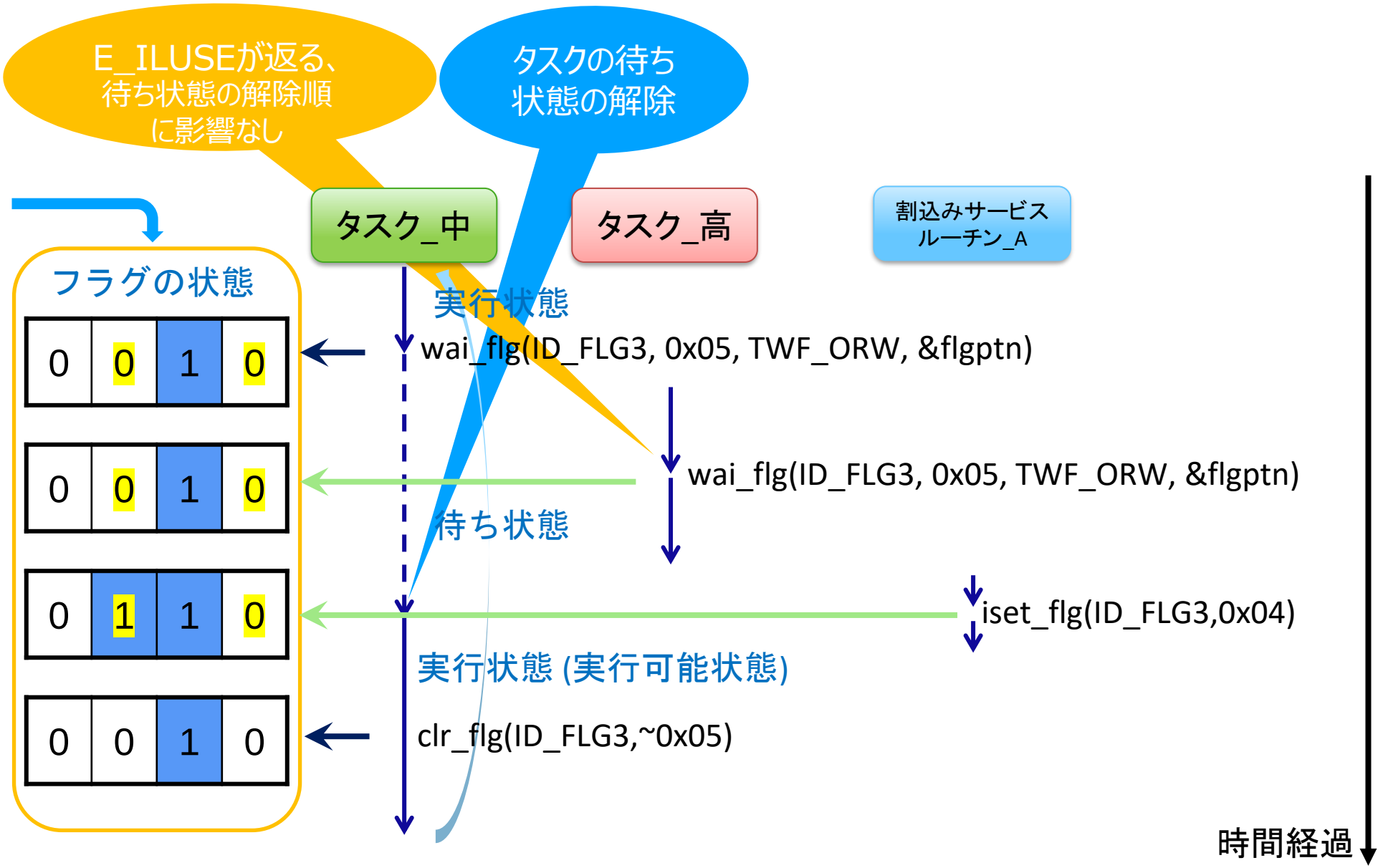
ID定義名: ID_FLG3

ビットパターン: 0x02

TA_WSGL指定

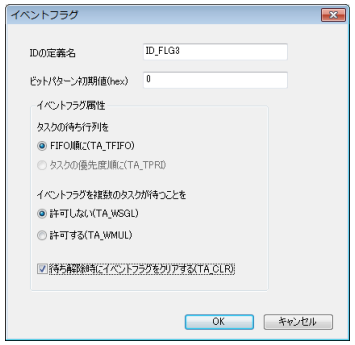
TA_CLR:指定なし

タスク待ち行列指定
→TA_TFIFOで

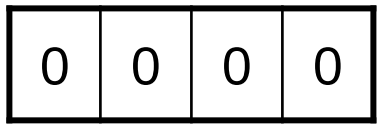


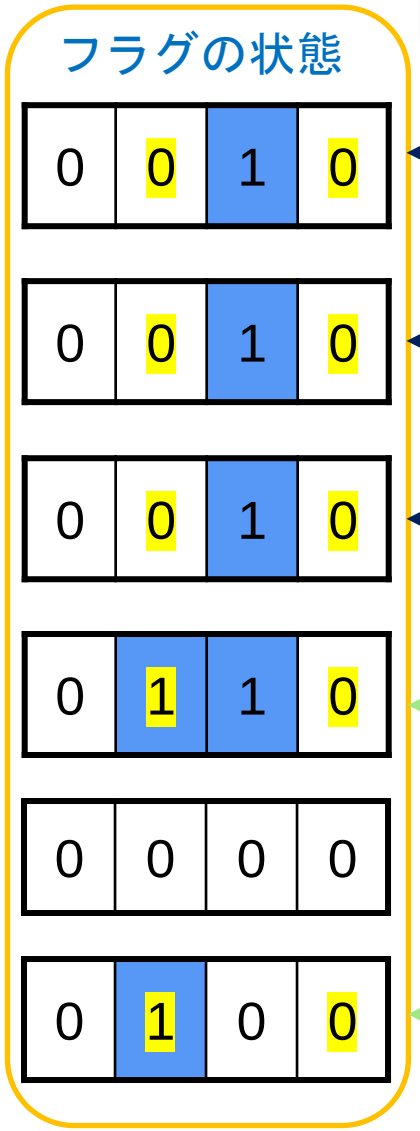
TA_CLR指定を利用してみる。

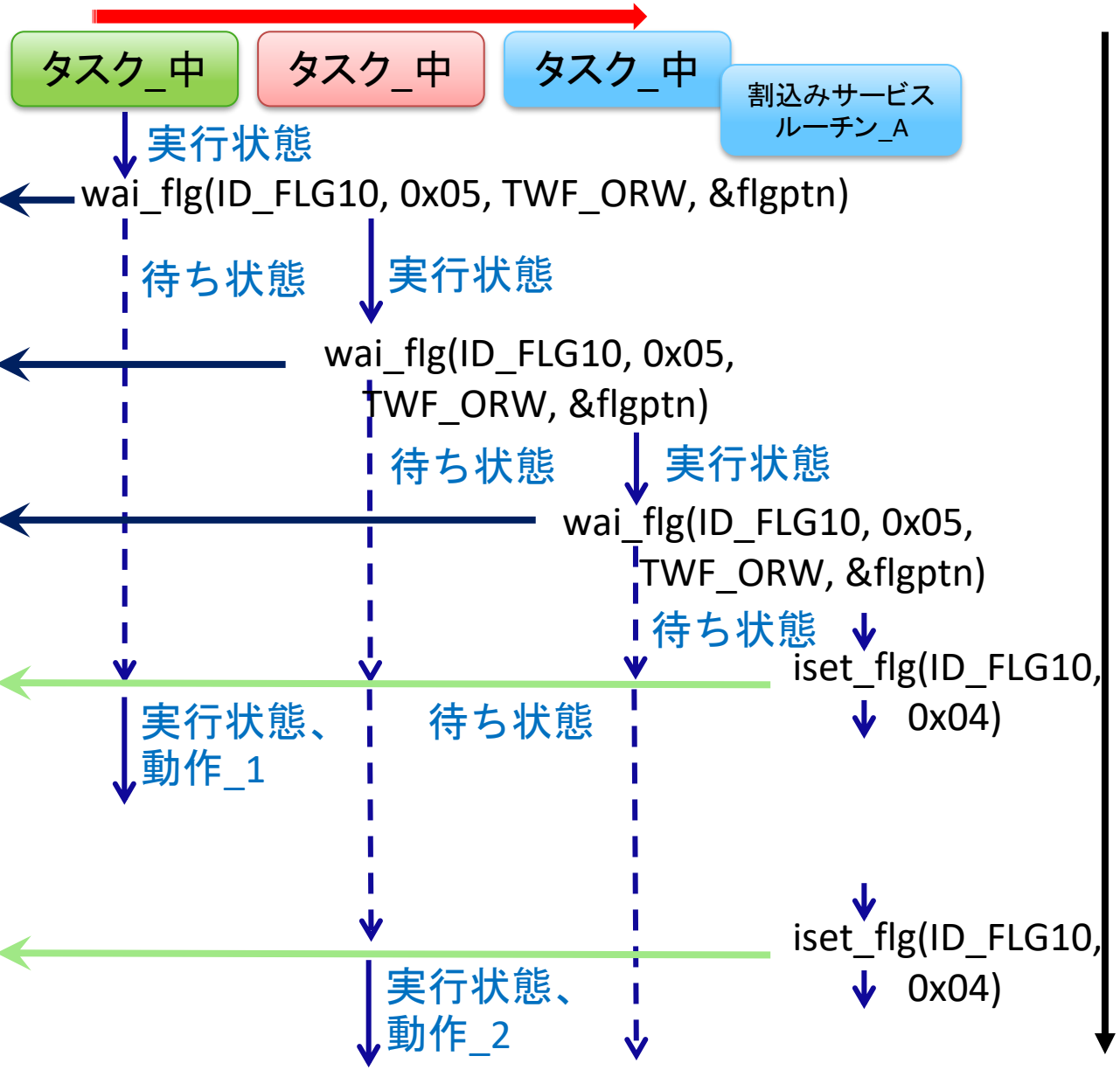
各タスクが、バッファに入り、条件成立で、動作する



ID定義名: ID_FLG10
ビットパターン: 0x02
TA_WMUL指定
TA_CLR:指定あり
タスク待ち行列:TA_TFIFO







セミナー振り返り

■ 評価ボード上でのサンプル作成にRTOSを利用

- タスク処理での、条件待ちを利用してみました
- イベントの発生を、イベントフラグを利用して、タスクに伝える処理を紹介

■ イベントフラグの紹介

- ビットパターンを利用した基本動作の紹介
- イベントフラグに設定できる属性や、属性による動作について
  - 複数のタスクが待つことの許可
  - 待ち解除時に、イベントフラグをクリアする
  - タスクの待ち行列の指定 (FIFO順・タスク優先度順)

ご清聴ありがとうございました

プログラム

4. イベントフラグの紹介
   - タスク待ち行列でのTA_TFIFO又はTA_TPRIの選択について
   - 補足
5. QA(2回目)