NC STATE UNIVERSITY

22: EVENT FLAGS FOR THREADS AND INTERRUPTS

v2

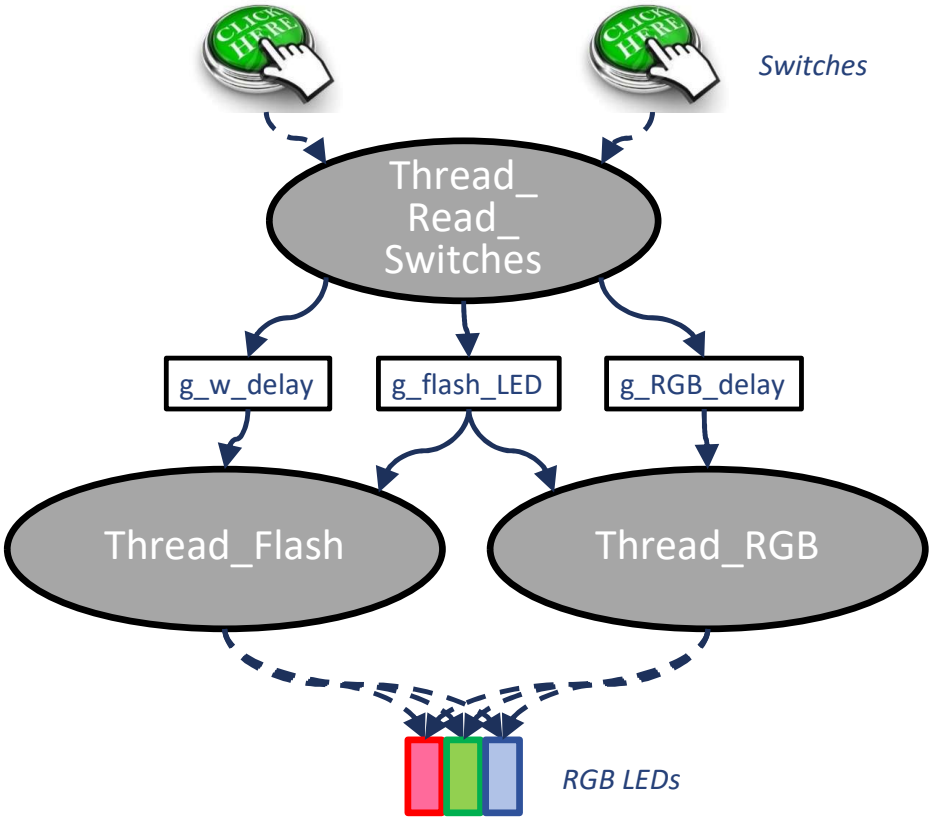
NC STATE UNIVERSITY

EVENTS FLAGS AND THREADS

RTX5 Demo Tasks and Delays: Ongoing RGB/Flasher Example

- Continue with LED example from ESF textbook
  - Port to use RTX5
- Start with simple version. Make minimal changes because we aren't experts on the RTOS yet:
  - We know about just a few RTOS features
  - We don't know which of those features actually matter

- Set program up as three separate threads
- Threads communicate using shared variables
  - g_flash_LED selects operating mode
  - g_w_delay, g_RGB_delay set flashing speed

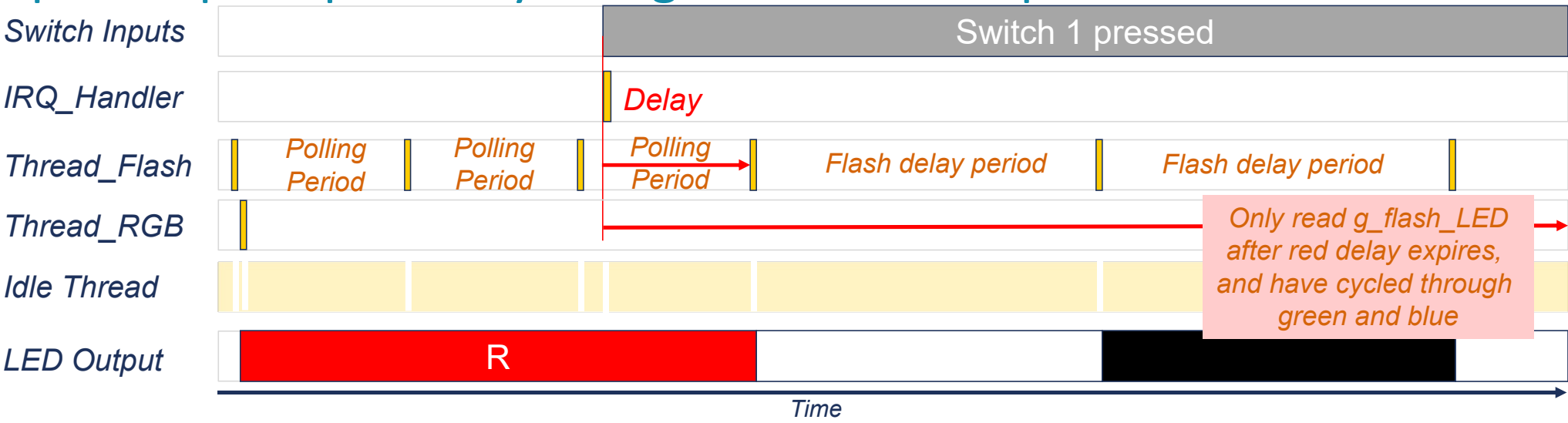


Task Structure Modification

- Use infinite loop in each task
- Need a blocking call in each path through the loop
  - Add osDelay call in else cases for Thread_Flash and Thread_RGB
    - Other methods possible
  - Thread_Read_Switches already has osDelay which executes on every path through its loop
- Shared variable (g_flash_LED)
  - Defines operating mode
  - Thread does scheduling internally with test (if g_flash_LED { … })
  - Only provides loose synchronization**, doesn't leverage OS scheduler**

```
void Thread_Flash(void * arg) {
    while (1) {
        if (g_flash_LED) {
            Control_RGB_LEDs(1, 1, 1);
            osDelay(g_w_delay);
            Control_RGB_LEDs(0, 0, 0);
            osDelay(g_w_delay);
        } else {
            osDelay(1);
        }
    }
}
```

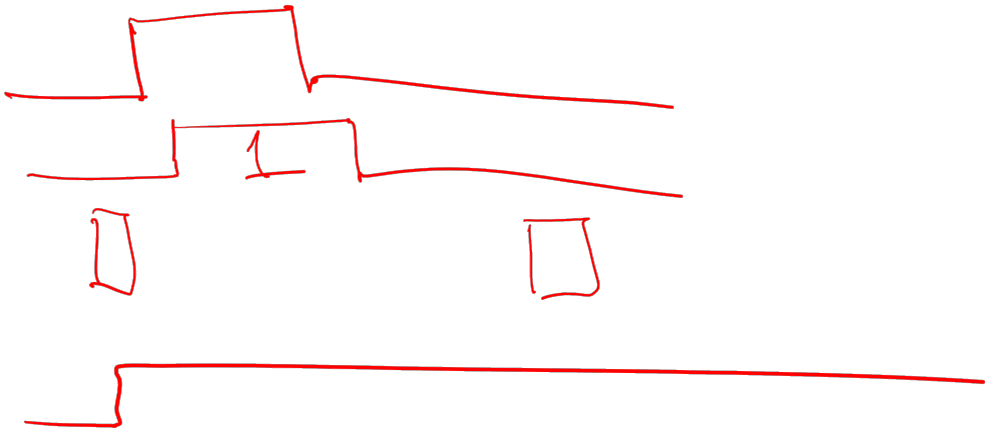
Speed Up Response By Using Switch Interrupt?



- Polling switches is slow, so let's use a **switch change interrupt** to update the shared variables
- **When** are Thread_Flash, Thread_RGB affected by change in g_flash_LED?
  - RTOS Scheduler is oblivious to changes
    - Doesn't know of connection between g_flash_LED and threads
  - Depends on thread behavior
    - Coder must build in polling support, thread yield operations
    - When: After running long enough for thread to read g_flash_LED
    - … depends on polling period, what else is running, priority, etc.
- Delay is dominated by task running and polling the shared variable g_flash_LED
  - Thread_Flash sees change soon, because 1 tick (1 ms) osDelay period for polling when inactive
  - Thread_RGB sees change much later, because it is blocking with osDelay call, and will still do green and blue before polling the variable

More Limitations

- What about a very quick button press?
  - g_flash_LED is set to 1 and then 0 before Thread_Flash gets a chance to run
  - We lose that switch press.
  - Does that matter? *Depends on application.*
- What about a ten second button press?
  - g_flash_LED only goes up to 1
  - We lose track of how long the switch is pressed.
  - Does that matter? *Depends on application.*

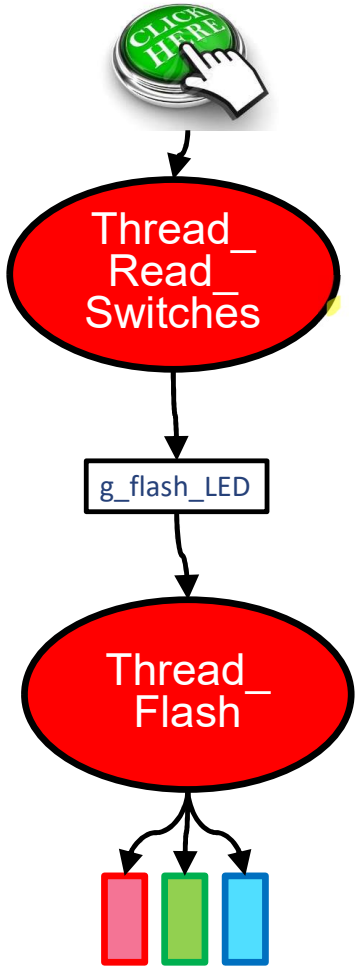


New Example Program: RTX5 Demo Events

Code available in Class Repository under RTX5

- Start with simple version:
  - #define DON'T_USE_EVENTS (1)
  - Uses shared variable g_flash_LED
- Thread_Read_Switches
  - Polls switches periodically (not using interrupt here yet)
  - If SW1 is newly pressed,
    tell Thread_Flash: set g_flash_LED to 1
    else
    clear g_flash_LED to 0
- Thread_Flash waits for switch press
  - If g_flash_LED is 1,
    clear it to zero
    flash LEDs magenta/blue five times

- Same limitations as non-preemptive task version
  - Sampling period in Thread_Read_Switches drives max response time, min switch press duration
  - Polling slows response, raises CPU load
  - RTOS can't help much, doesn't know relationship between threads and g_flash_LED. Coder must implement polling and yield operations.
- Improve by using RTOS ***event flag*** feature
  - Will synchronize part of Thread_Flash to run after Thread_Read_Switches detects a switch event

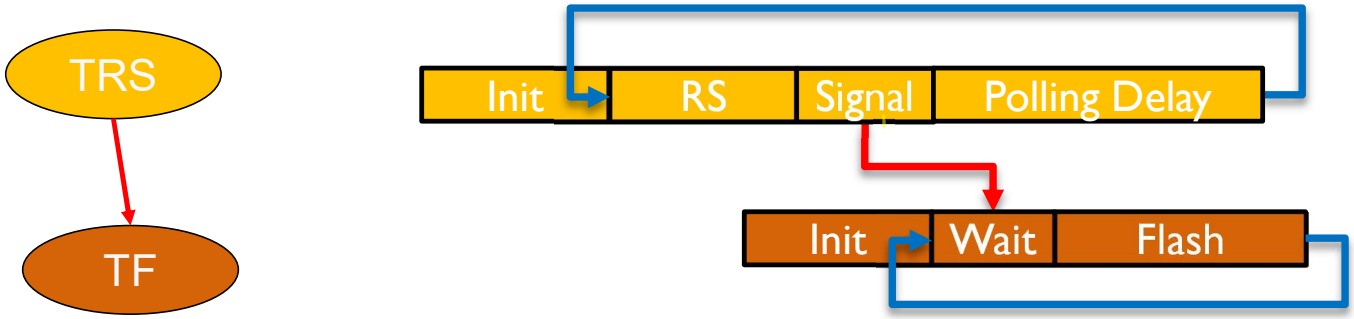


Concepts of Synchronization with Events

- Background
  - Concurrent preemptive threads have arbitrary execution order and interleaving
  - Sometimes want to **synchronize** code: enforce **order** of code execution between parts of different threads
- OS has synchronization primitive to signal an event has occurred: **event flag**

- Code structure:
  - A uses *signal event* OS call to trigger (signal, release) B
  - B uses *wait for event* OS call
- Can put *wait for event* call into a loop to allow it to service a series of events

Applying Synchronization

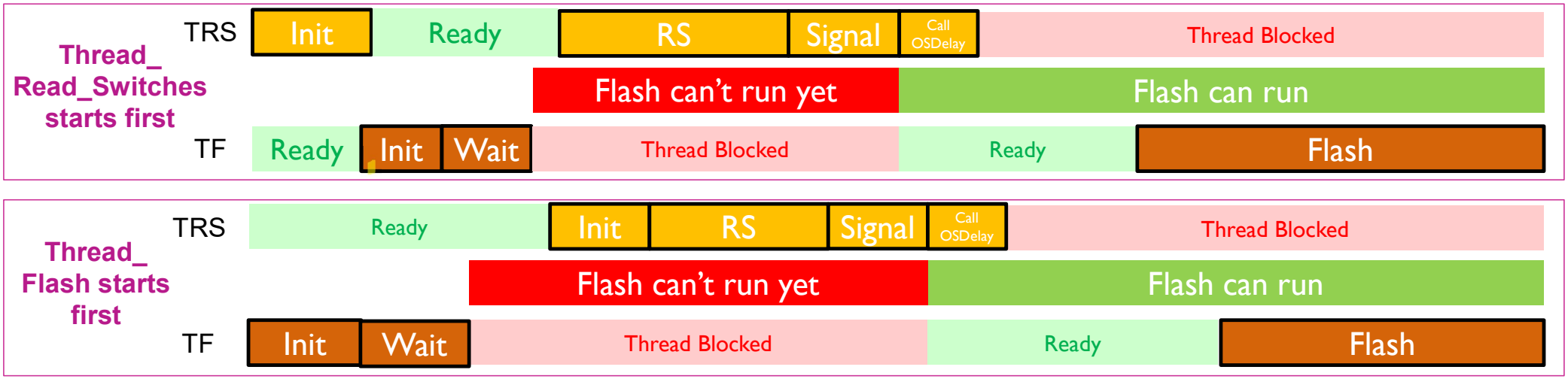


- Goal: Don't flash LED until **after** leading edge of switch press
- TRS: Thread_Read_Switches
  - Signals TF to flash LED each time switch is pressed
  - Three parts: initialization, read switches, signal that switch has been pressed
- TF: Thread_Flash
  - Flashes LED five times each time it is signaled
  - Three parts: initialization, wait for signal, flash LED

NC STATE UNIVERSITY

Behavior Using Event Flags

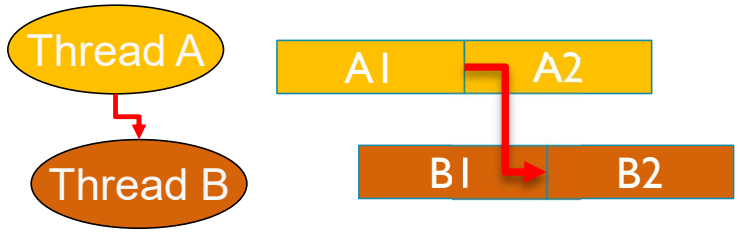
Use to indicate switch has been pressed or released

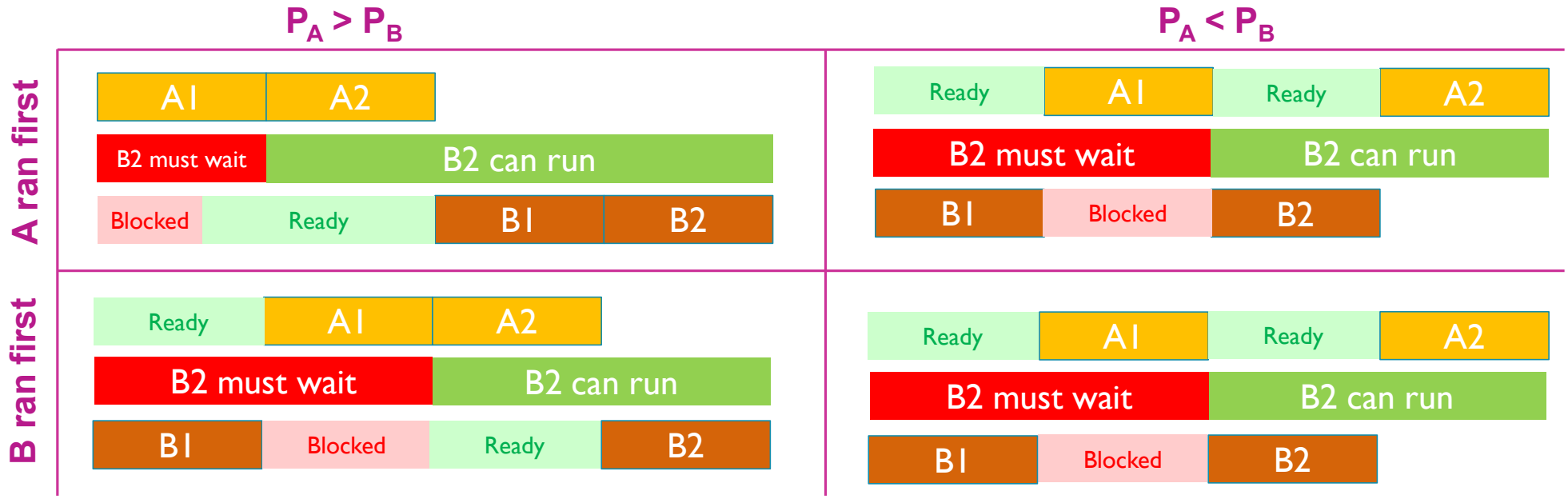


- Thread_Read_Switches (low priority) will set event flag based on switches
- Thread_Flash (high priority) will wait for event flag to become set with only one call to RTOS.
  - If event flag is set, Thread_Flash can continue executing (remains ready, returns to running)
  - If event flag is not set, RTOS moves Thread_Flash to blocked state. When event flag is later set, RTOS will move Thread_Flash to ready state and clear that event flag.
- Examine two (of many) possible execution sequences

General Synchronization

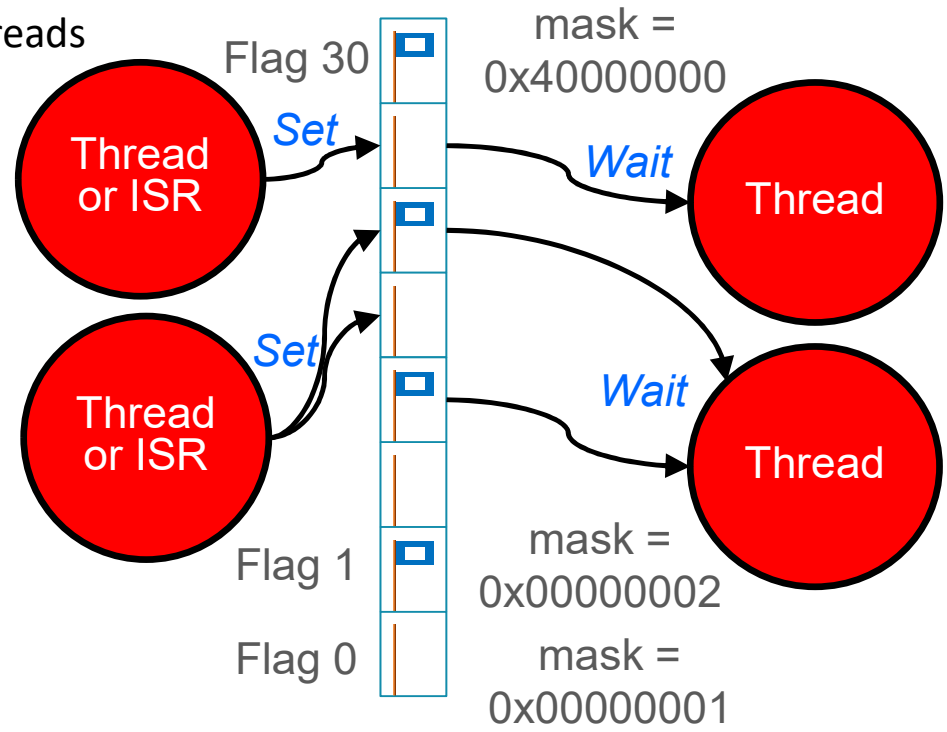
- Don't let Thread B start to execute section B2 until Thread A has completed section A1
- Four possible cases based on
  - Thread priority
  - Initial thread execution order





CMSIS-RTOS2 Event Flags

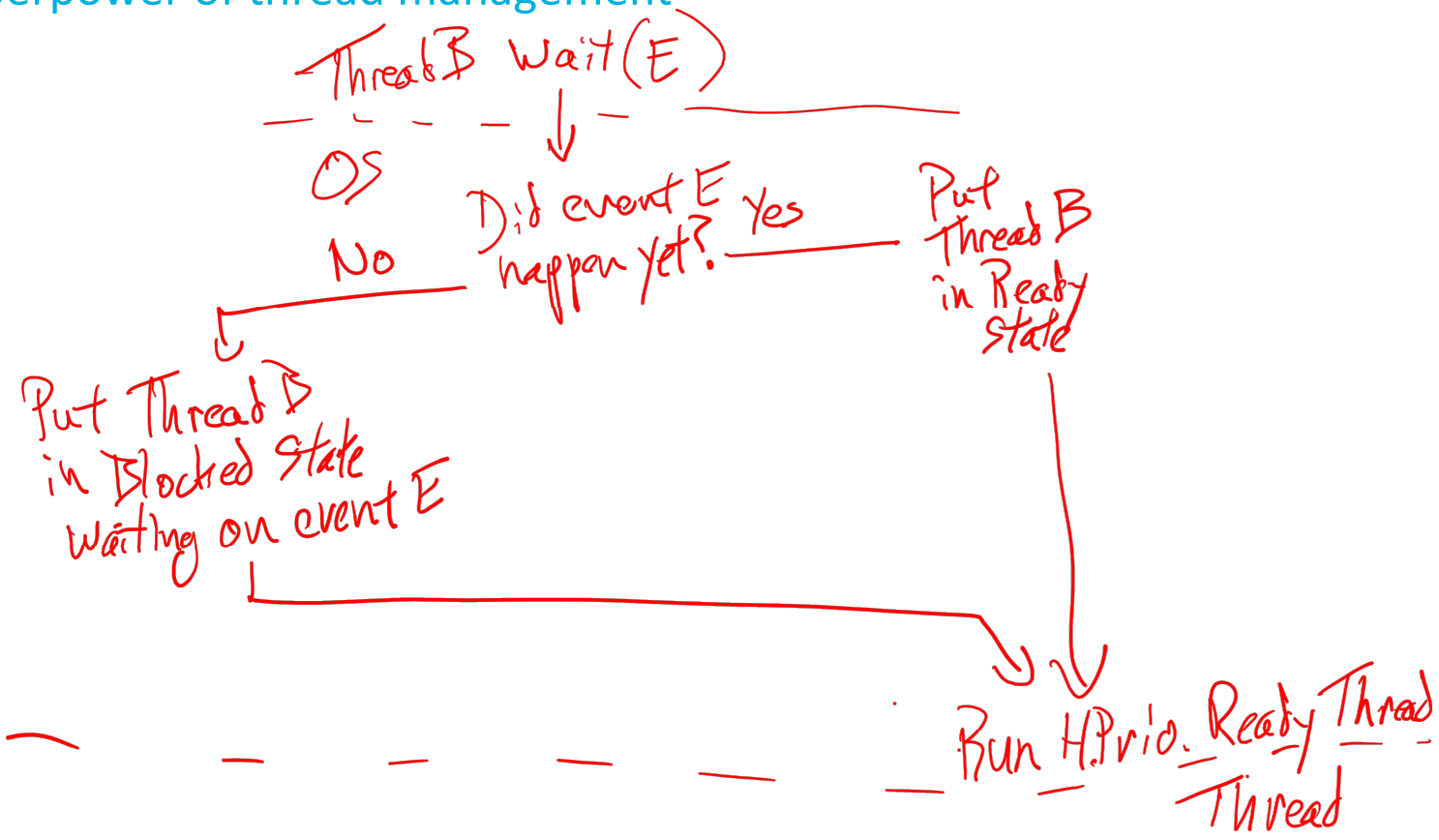
- Allows threads and ISRs to **set** event to signal other threads
  - 0: "Event 3 hasn't occurred (since last service (if any))"
  - 1: "Event 3 occurred and hasn't been serviced"
- Can trigger on combination of flags. Thread can **wait** for …
  - Any specified flag to be set
  - All of its specified flags to be set
- Specify which flags matter with mask argument
- Event flags object holds 31 binary flags
- *Thread flags* also exist
  - Event flag object already built into each thread
- Limitation: No handshaking enforced
  - OK to raise a flag which is already up,
  - *Of course, you could add code to specifically test to see if it is up before trying to raise it, but there are better ways to get handshaking*



NC STATE UNIVERSITY

Behavior of Thread Calling **Wait for Event**

Applies OS superpower of thread management



CMSIS-RTOS2 Event Flag Functions

https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__EventFlags.html

- osEventFlagsNew(attr)
  - Create new event flags object
  - Can use attr to define attributes
- osEventFlagsSet (ef_id, flags)
  - Set event flags (for ef_id) which are set (1) in flags argument
  - Returns previous event flags, or error (unknown, illegal param., invalid resource state)
- osEventFlagsClear(ef_id, flags) – used less often
  - Clears thread's signal flags which are set (1) in flags argument
  - Returns previous event flags, or error code (unknown, illegal param., invalid resource state)
- osEventFlagsGet(ef_id) – used less often
  - Returns current value of flags

- osEventFlagsWait (ef_id, flags, options, timeout)
  - options: osFlagsWaitAny, osFlagsWaitAll
  - Checks to see if specified flags are set
    - If so, clears them and returns immediately
    - If not, waits until they are set, then clears them and returns
    - If timeout occurs, returns with error code
  - Return value: bit 31 indicates error
    - 0? OK, Event flags before OS cleared them.
      - Using osFlagsWaitAny? Tells which event(s) occurred
    - 1? Error code: unknown, timeout, parameter, resource.
    - Mask for bit 31 defined in cmsis_os2.h

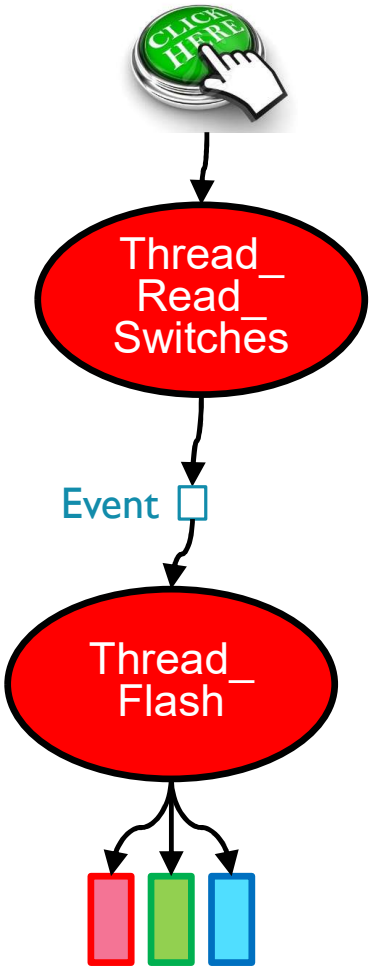
```
#define osFlagsError 0x80000000U
```

```
if (return_value & osFlagsError)
    identify error and handle it
else
    identify which event(s) occurred
```

A Better Version: RTX5 Demo Using RTOS Events

#define DON'T_USE_EVENTS (0)

- Use RTOS **event flag** to indicate switch has been pressed
  - Thread RS can set event flag
  - Thread F can wait for event flag to become set with only one call to RTOS.
    - If event flag is set, RTOS clears event flag and thread F can continue executing (remains ready)
    - If event flag is not set, RTOS moves thread F to blocked state. When event flag is later set, RTOS will move F to ready state and clear that event flag.
- Thread_Read_Switches polls switches
  - If SW1 newly pressed, send event to Thread_Flash
  - Later we will replace this thread with an interrupt
- Thread_Flash waits for event
  - When event occurs,
    - Event is cleared by RTOS
    - Thread unblocks, resumes and flashes LEDs magenta/blue five times



Demo Initialization Code

```
osEventFlagsId_t evflags_id;    // Use bit 0 (value of 1) for flash request

void Init_My_RTOS_Objects(void) {
  tid_Flash = osThreadNew(Thread_Flash, NULL, NULL);    // Create thread
  tid_Read_Switches = osThreadNew(Thread_Read_Switches, NULL, NULL);    // Create thread
  evflags_id = osEventFlagsNew(NULL);
}

int main (void) {
  // System Initialization
  SystemCoreClockUpdate();

  Init_RGB_LEDs();
  Init_Switches();

  osKernelInitialize();                 // Initialize CMSIS-RTOS

  Init_My_RTOS_Objects();

  osKernelStart();                      // Start thread execution
  for (;;) {}
}
```

Demo Event Code

```
// Event flag masks
#define PRESSED (1)
#define RELEASED (2)
```

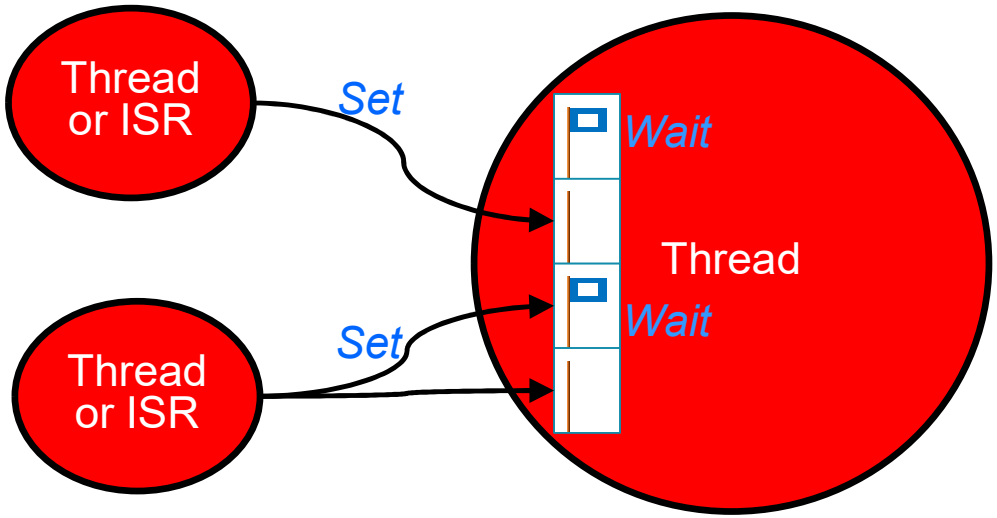
```
void Thread_Read_Switches(void  * arg) {
  int previously_pressed=0;
  while (1) {
    osDelay(200);
    if (SWITCH_PRESSED(SW1_POS)) {
      if (previously_pressed == 0)
        osEventFlagsSet(evflags_id, PRESSED);
      previously_pressed = 1;
    } else {
      previously_pressed = 0;
    }
  }
}
```

```
void Thread_Flash(void * arg) {
  int n;
  uint32_t result;
  while (1) {
    result = osEventFlagsWait(evflags_id, PRESSED,
    osFlagsWaitAny, osWaitForever);
    if (result & osFlagsError) {
      // identify error, handle it
    } else { // identify event, handle it
      if (result & PRESSED) {
        for (n=0; n<5; n++) {
          Control_RGB_LEDs(1, 0, 1);
          osDelay(g_w_delay);
          Control_RGB_LEDs(0, 0, 1);
          osDelay(g_w_delay);
        }
        Control_RGB_LEDs(0, 0, 0);
      } // else other events here
    }
  }
}
```

CMSIS-RTOS2 Thread Flags

Set of Event Flags built into each thread

- Can signal a specific thread
  - "Hey Bob! Event 3 occurred!"
  - Compare with Event Flags – accessible to all threads: "If anyone cares, Event 3 occurred."
- Each thread is allocated 31 event flags
- Thread can wait for
  - Any specified flags to be set
  - All specified flags to be set
- Other threads or ISRs can set or clear one or more of a thread's signal flags



CMSIS-RTOS2 Thread Flag Functions

- osThreadFlagsSet (tid, flags)
  - Set event flags (for t_id) which are set (1) in flags argument
  - Returns previous event flags, or error code (unknown, illegal parameter, resource in invalid state)
- osThreadFlagsClear(flags) – used less often
  - Clears specified (1) flags in currently-running thread
  - Returns previous event flags, or error (unknown, illegal parameter, resource in invalid state)
- osThreadFlagsGet() – used less often
  - Returns current value of flags
- osThreadFlagsWait (flags, options, timeout)
  - options: osFlagsWaitAny, osFlagsWaitAll, osFlagsNoClear
  - Checks to see if specified flags are set
    - If so, clears them and returns immediately
    - If not, waits until they are set, then clears them and returns
    - If timeout occurs, returns with error code
  - Return value
    - Event flags before clearing, or error code (unknown, timeout, parameter, resource)

NC STATE UNIVERSITY

Communication Between Threads (and ISRs)

	Receiver Thread	Information Provided	Can Accumulate Multiple Pending Events?	Handshake?
Event Flag	Any thread	"The event has occurred"	No	No
Thread Flag	Specified thread	"The event has occurred"	No	No
Semaphore	Any thread	"The event has occurred"	Yes (counting semaphore), No (binary semaphore)	Yes
Message Queue	Any thread	"An event described by this message has occurred"	Yes, up to number of available queue elements	Yes

NC STATE UNIVERSITY

USING INTERRUPTS WITH AN RTOS

Using Interrupts with RTX

- Design guidelines
  - **Must not** call functions which may block
  - Make ISR as short as practical to minimize delays to high-priority processing
  - Defer work to a thread – signal it with an event or another mechanism
  - To simplify development and debugging, avoid ISR nesting
- Be careful with RTOS calls from an ISR
  - Some functions will return error code if called from ISR
  - See documentation to see which functions can be called from ISR
  - RTOS call could cause blocking or unwanted context switching
- Data point: Automotive Software
  - AutoSAR uses OSEK/VDX scheduler
  - Developers allow interrupt nesting (handlers are preemptible)
  - AutoSAR divides interrupts into two classes, based on whether they are allowed to trigger context switches
  - Developers don't use mutexes, just disable and restore interrupts for critical sections (speed)

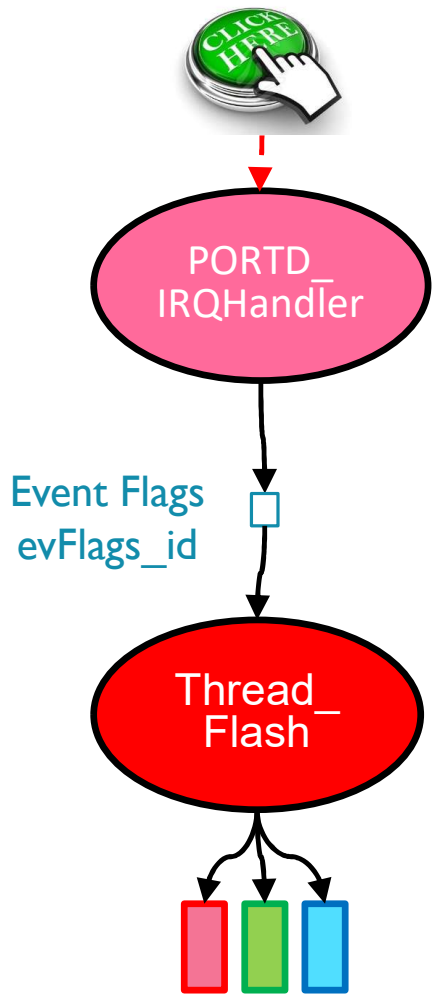
RTX5 Functions Which Can Be Called From An ISR

- May need specific values of arguments
  - E.g. set timeout = 0 so function will return immediately with error code rather than block

- osKernel
  - GetTickCount, GetTickFreq, GetSysTimerCount, GetSysTimerFreq
- osThreadFlags
  - Set
- osEventFlags
  - Set, Clear, Get, Wait
- osSemaphore
  - Acquire, Release, GetCount
- osMemoryPool
  - Alloc, Free, GetCapacity, GetBlockSize, GetCount, GetSpace
- osMessageQueue
  - Put, Get, GetCapacity, GetMsgSize, GetCount, GetSpace

RTX5 Demo: Events with Interrupts

- PORTD_IRQHandler responds to switches
  - If SW1 pressed or released, send event to Thread_Flash
- Thread_Flash waits for event
  - Flashes LEDs 5 times based on event
    - Press Event: Magenta/Blue
    - Release Event: Green/Yellow



Demo Code - Initialization

```c
int main (void) {
  // System Initialization
  SystemCoreClockUpdate();

  Init_RGB_LEDs();
  Init_Switches();

  osKernelInitialize();
  Init_My_RTOS_Objects();
  Initialize_Interrupts();

  osKernelStart();
}
```

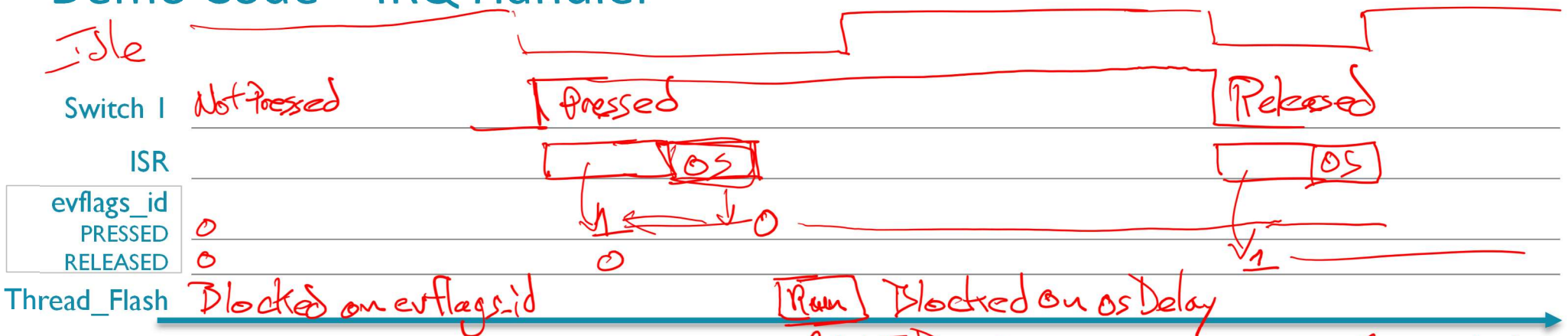
```c
osThreadId_t tid_Flash;
osEventFlagsId_t evflags_id;

void Init_My_RTOS_Objects(void) {
  tid_Flash = osThreadNew(Thread_Flash, NULL, NULL);
  evflags_id = osEventFlagsNew(NULL);
}

void Initialize_Interrupts(void) {
  /* Configure PORT peripheral. Select GPIO and enable pull-up
  resistors and interrupts on all edges for pins connected to switches */
  PORTD->PCR[SW1_POS] = PORT_PCR_MUX(1) | PORT_PCR_PS_MASK |
    PORT_PCR_PE_MASK | PORT_PCR_IRQC(11);
  PORTD->PCR[SW2_POS] = PORT_PCR_MUX(1) | PORT_PCR_PS_MASK |
    PORT_PCR_PE_MASK | PORT_PCR_IRQC(11);

  /* Configure NVIC */
  NVIC_SetPriority(PORTD_IRQn, 128);
  NVIC_ClearPendingIRQ(PORTD_IRQn);
  NVIC_EnableIRQ(PORTD_IRQn);

  /* Configure PRIMASK */
  __enable_irq();
}
```

Demo Code – IRQ Handler


```
void PORTD_IRQHandler(void) {
  PTB->PSOR = MASK(DBG_1);
  // Read switches
  if ((PORTD->ISFR & MASK(SW1_POS))) {
    if (SWITCH_PRESSED(SW1_POS)) {
      osEventFlagsSet(evflags_id, PRESSED);
    } else {
      osEventFlagsSet(evflags_id, RELEASED);
    }
  }
  // clear status flags
  PORTD->ISFR = 0xffffffff;
  PTB->PCOR = MASK(DBG_1);
}
```

```
// Event flag masks
#define PRESSED (1)
#define RELEASED (2)
```

Demo Code – IRQ Handler




```
void PORTD_IRQHandler(void) {
  PTB->PSOR = MASK(DBG_1);
  // Read switches
  if ((PORTD->ISFR & MASK(SW1_POS))) {
    if (SWITCH_PRESSED(SW1_POS)) {
      osEventFlagsSet(evflags_id, PRESSED);
    } else {
      osEventFlagsSet(evflags_id, RELEASED);
    }
  }
  // clear status flags
  PORTD->ISFR = 0xffffffff;
  PTB->PCOR = MASK(DBG_1);
}
```

OS

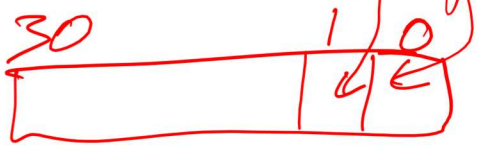
Return from Interrupt

```
// Event flag masks
#define PRESSED  (1)
#define RELEASED (2)
```

Green, Blue

Repeat

FOO



2^n $0 \le n \le 30$

Demo Code – Thread_Flash

```
// Event flag masks
#define PRESSED (1)
#define RELEASED (2)

void Thread_Flash(void * arg) {
  int n;
  uint32_t result;
```

- Why does it sometimes have extra flashes?

```
  while (1) {
    result = osEventFlagsWait(evflags_id,
    PRESSED | RELEASED, osFlagsWaitAny, osWaitForever);
    if (result & osFlagsError) {
      // identify error, handle it
    } else { // identify event, handle it
      if (result & PRESSED){
        for (n=0; n<5; n++) {
          Control_RGB_LEDs(1, 0, 1);
          osDelay(g_w_delay);
          Control_RGB_LEDs(0, 0, 1);
          osDelay(g_w_delay);
        }
      }
      if (result & RELEASED) {
        for (n=0; n<5; n++) {
          Control_RGB_LEDs(1, 1, 0);
          osDelay(g_w_delay);
          Control_RGB_LEDs(0, 1, 0);
          osDelay(g_w_delay);
        }
      }
      Control_RGB_LEDs(0, 0, 0);
    }
  }
}
```