



Keys

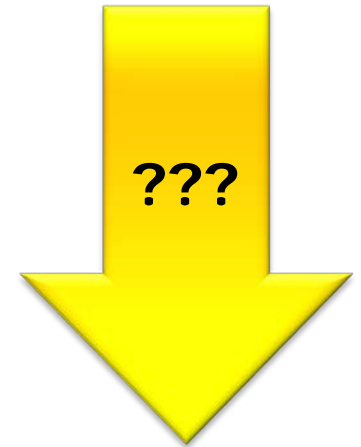
"We are going to use the available switches on the boards as additional way the user can provide input."

Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

**Scriptum:
Bouncing Switch**

Learning Topics

- Problem: use keys/switches for application
- Keys
 - Hardware/Connectivity
 - Pull-Up, Pull-Down
 - Interrupt
 - Polling
- Software
 - Key driver
 - Control and data flow

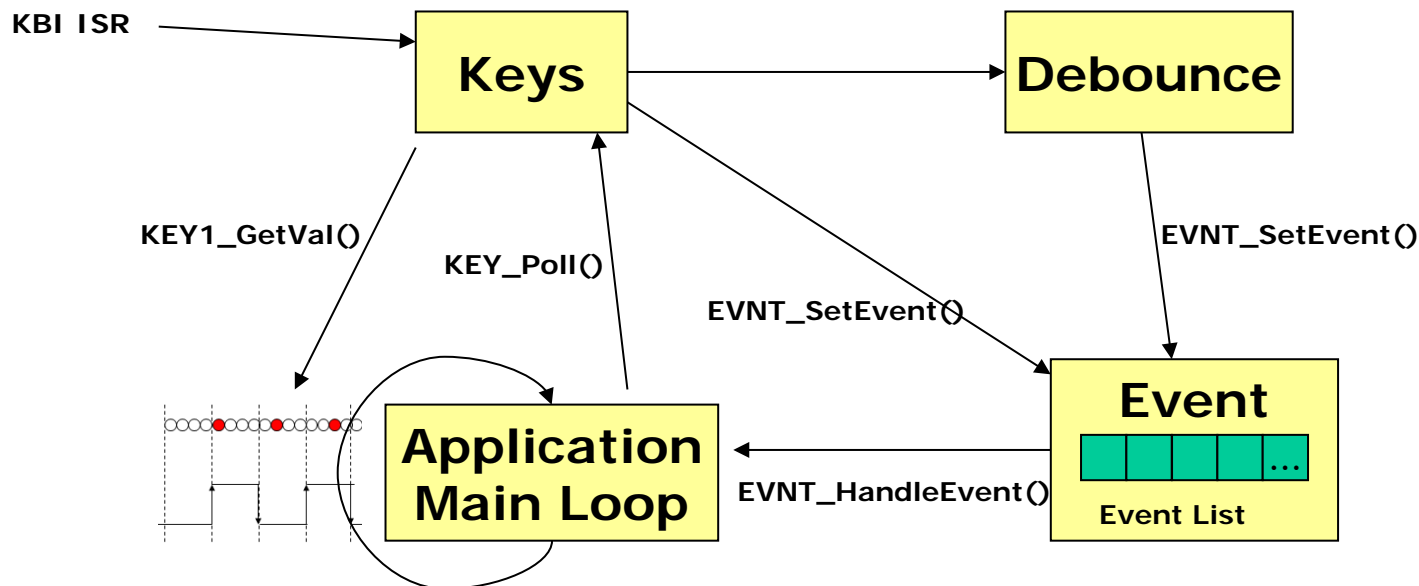


Push Buttons



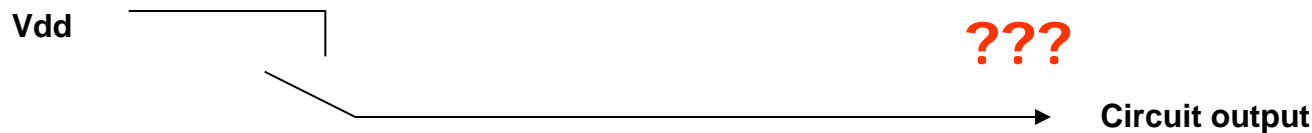
Key Scanning

- *Key press detection (Polling and Interrupts)*
- Debounce; long and short key detection

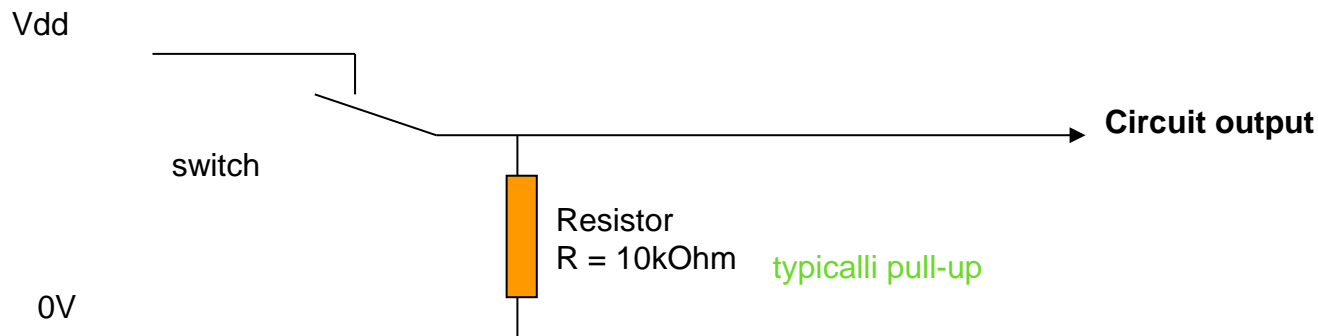


Pull-Up, Pull-Down

- Open switch?
- undefined!

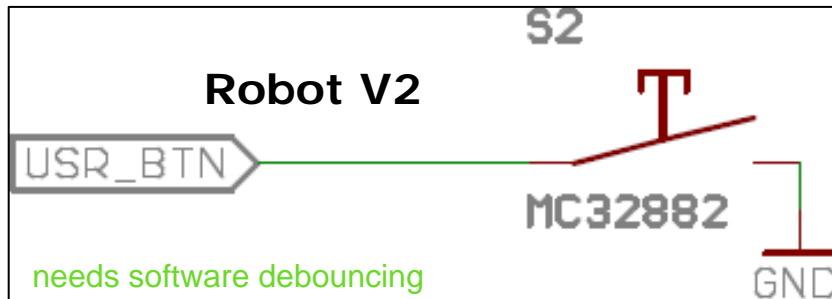


- Solution
 - Enforcing of defined logical level
 - Internal circuit (resistor/port configuration)
 - external circuit (resistor)



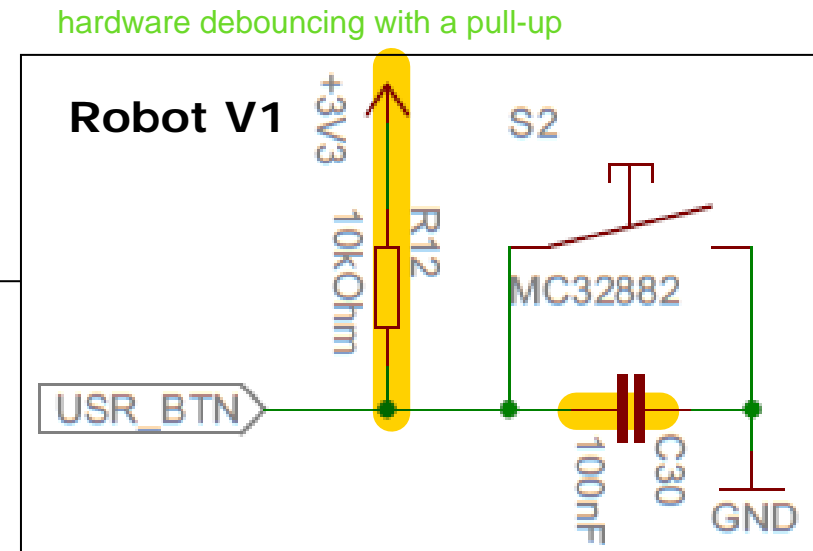
V2 Hardware: Push Button

- No hardware pull-up resistor for USR_BTN
- Use microcontroller internal pull-up



```
#if PL_CONFIG_BOARD_IS_ROBO_V2
    #include "PORT_PDD.h"
#endif

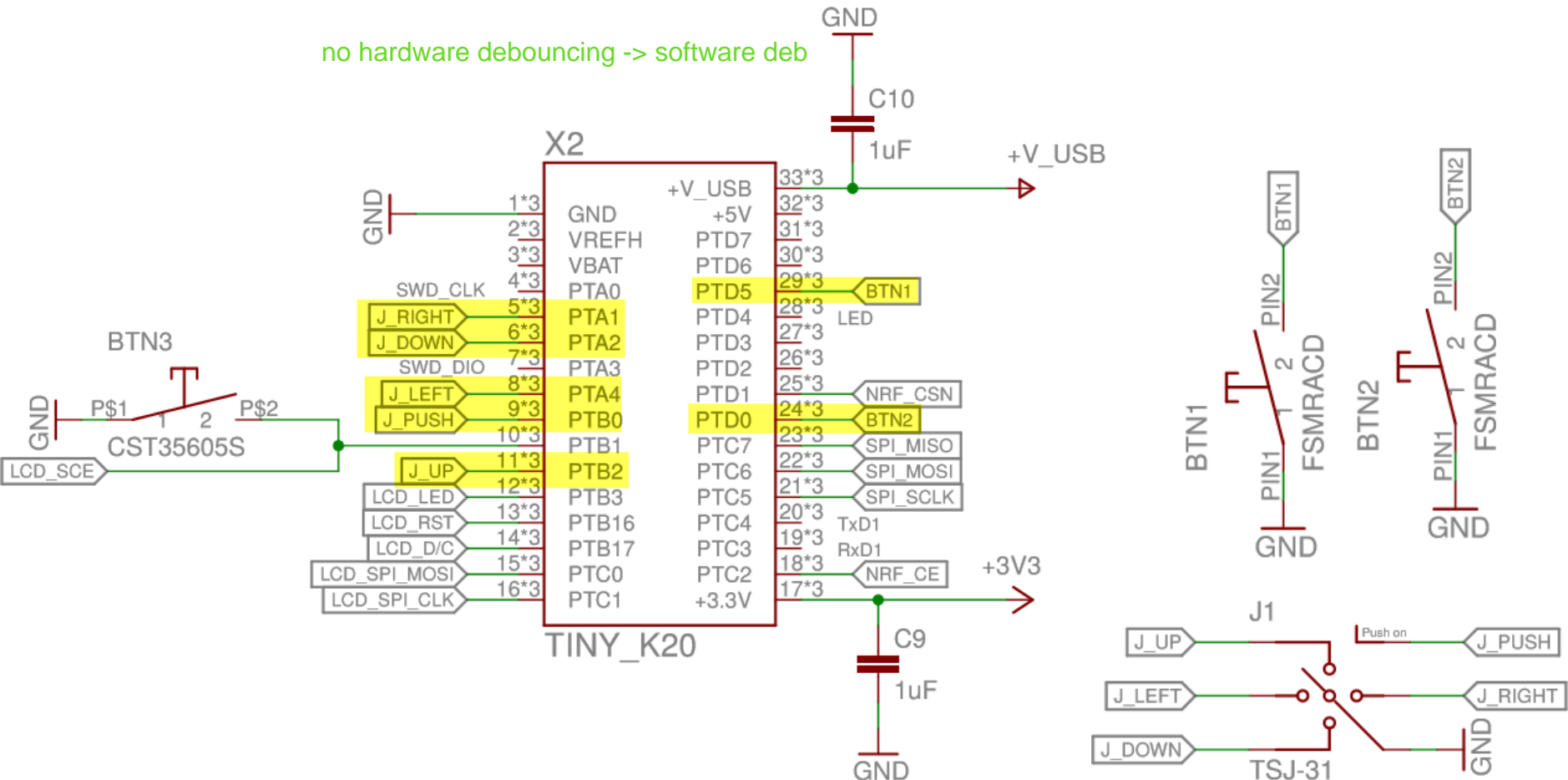
void KEY_Init(void) {
    #if PL_CONFIG_BOARD_IS_ROBO_V2
        /* enable and turn on pull-up resistor for PTA14 */
        PORT_PDD_SetPinPullSelect(PORTA_BASE_PTR, 14, PORT_PDD_PULL_UP);
        PORT_PDD_SetPinPullEnable(PORTA_BASE_PTR, 14, PORT_PDD_PULL_ENABLE);
    #endif
}
```



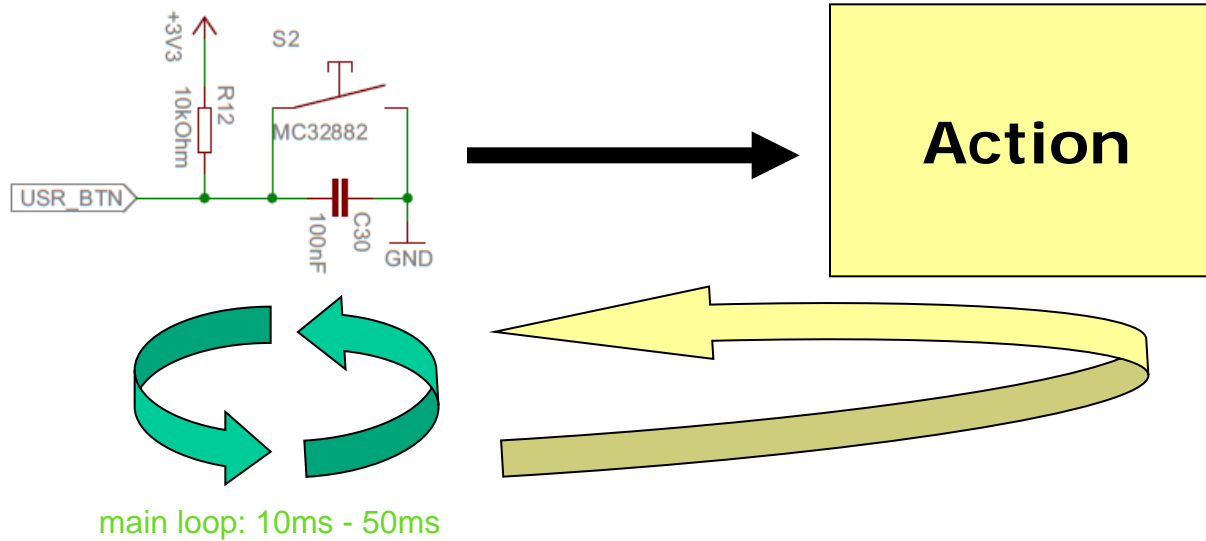
Remote Schematics

- No pull-ups and no debouncing capacitors

no hardware debouncing -> software deb



Keyboard Driver: Solution 1

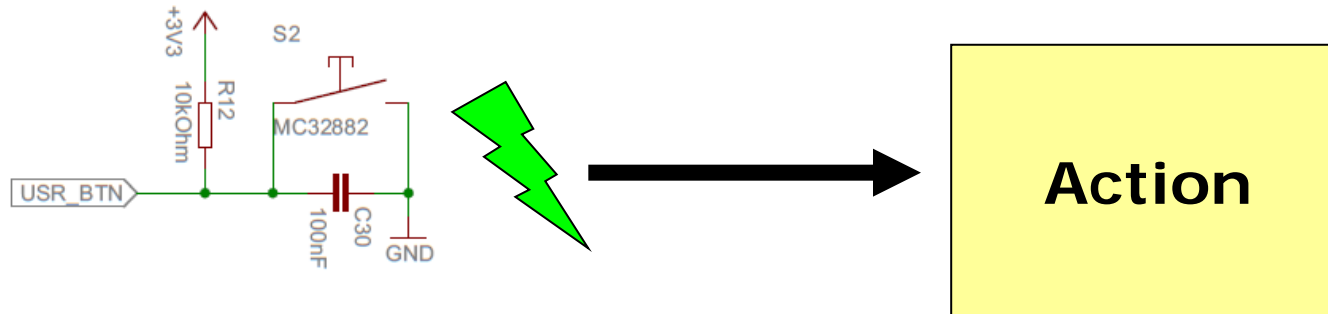


- Realtime Synchronization
- Gadfly Synchronization
polling a bit (status bit)

Simple Realtime/Gadfly Synchronization

```
if (SW1_Get()) {  
    WAIT1_Waitms(50); /* simple debounce */  
    if (SW1_Get()) { /* still pressed? */  
        cnt = 0;  
        while(SW1_Get()) {  
            WAIT1_Waitms(1);  
            cnt++; /* measure time */  
        } /* wait until released */  
        if (cnt<=1000) { /* short press*/  
            EVNT_SetEvent(EVNT_SW1_SHORT_PRESSED); blinking an led in this event  
        } else { /* long press*/  
            EVNT_SetEvent(EVNT_SW1_LONG_PRESSED);  
        }  
    }  
}
```

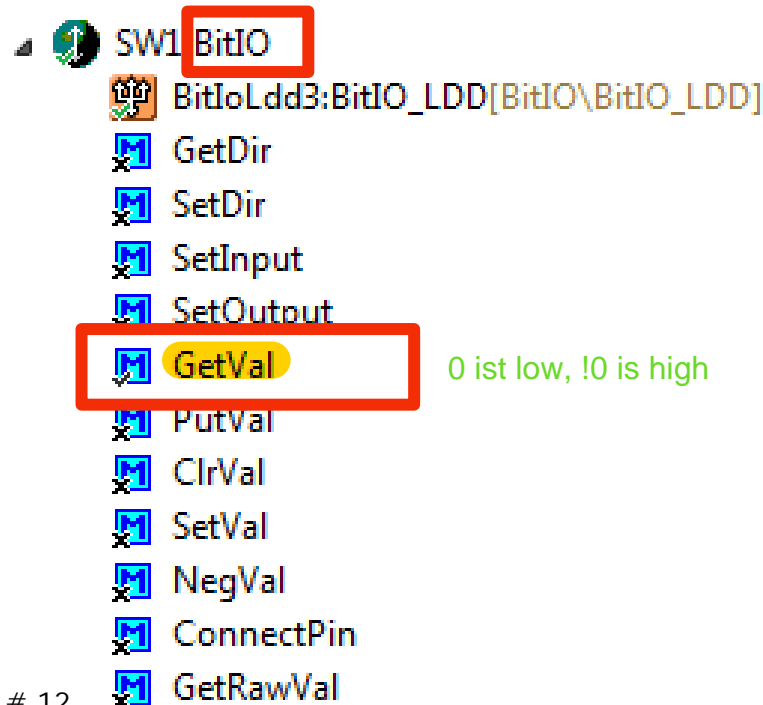
Keyboard Driver: Solution 2



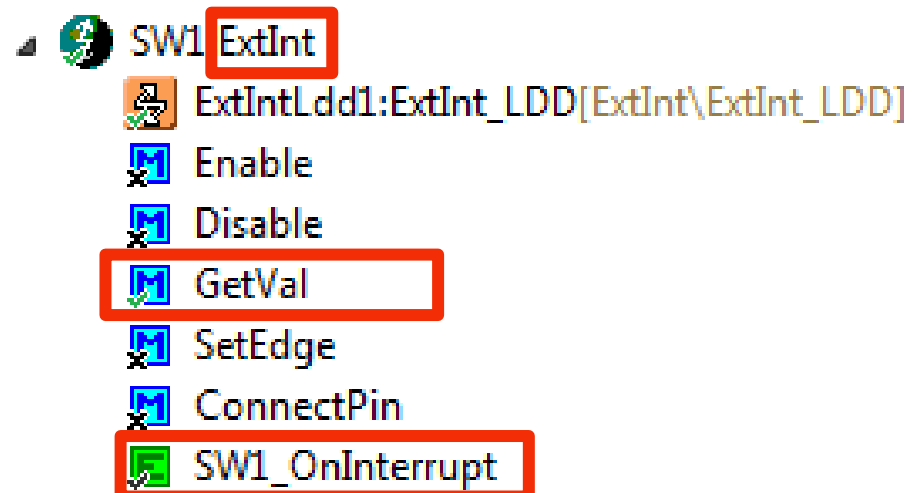
- Interrupt Synchronization
- No Busy Waiting/Polling
- Requires that pin is able to generate interrupt

Key Pin Synchronization

- BitIO Component
 - Polling/Gadfly
- ExtInt Component
 - Polling/Gadfly, Interrupt
 - Not every pin has interrupt capability!



falling, rising edge



Control and Data Flow with Polling

```
callback(EVNT_Handle event):
    switch(event) {
        case EVNT_SW1_PRESSED:
            print("SW1 pressed");
```

Main Application Loop:

```
for(;;) {
    KEY_Scan();
    EVNT_HandleEvent(callback);
}
```

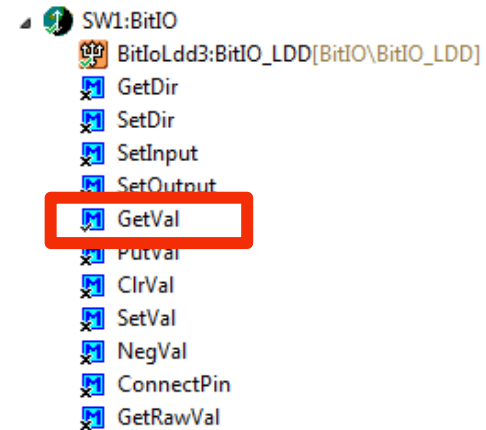
Keys.c

```
KEY_Scan(void):
    if (SW1_GetVal()) {
        EVNT_SetEvent(EVNT_SW1_PRESSED);
```

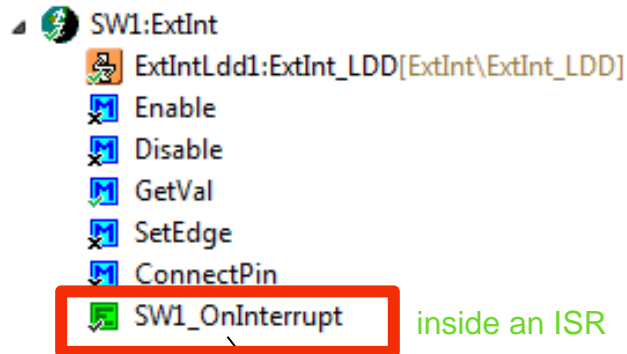
Event



Event Array



Control and Data Flow with Interrupts



Events.c

```
SW1_OnInterrupt():
    KEY_OnInterrupt(KEY_BTN1);
```

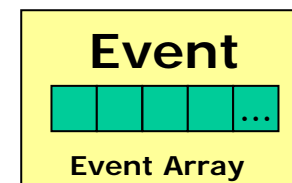
Keys.c

```
KEY_OnInterrupt(KEY_Buttons button):
    if (button==KEY_BTN1) {
        EVNT_SetEvent(EVNT_SW1_PRESSED);
```

```
callback(EVNT_Handle event):
    switch(event) {
        case EVNT_SW1_PRESSED:
            print("SW1 pressed");
```

Main Application Loop:

```
for(;;) {
    EVNT_HandleEvent(callback);
}
```



Interrupt Priorities

The screenshot shows a software configuration interface with a tabbed window. The 'Properties' tab is active, displaying a table of configuration parameters. The 'Interrupt priority' property is highlighted in yellow, and its dropdown menu is open, showing a list of priority levels. The 'Value' column shows 'medium priority' and '8'. The 'Details' column shows 'INT_FTM0'. The 'Interrupt period' is set to '10 ms'. The 'Same period in modes' is set to 'low priority'. The 'Component uses entire timer' is set to 'medium priority'. The 'Initialization' section shows 'Enabled in init. code' and 'Events enabled in init.' set to '0'. The 'CPU clock/speed selection' section shows 'High speed mode' set to '2', 'Low speed mode' set to '3', and 'Slow speed mode' set to '4'. The 'Referenced components' section shows a list of components from 1 to 6.

Name	Value	Details
Component name	TI1	
Periodic interrupt source	FTM0_MOD	FTM0_MOD
Counter	FTM0_CNT	FTM0_CNT
Interrupt service/event	Enabled	
Interrupt	INT_FTM0	INT_FTM0
Interrupt priority	medium priority	8
Interrupt period	minimal priority	10 ms
Same period in modes	low priority	
Component uses entire timer	medium priority	
Initialization		
Enabled in init. code	high priority	
Events enabled in init.	maximal priority	
CPU clock/speed selection		
High speed mode	1	
Low speed mode	2	This component is enabled
Slow speed mode	3	This component is disabled
Referenced components	4	This component is disabled
	5	
	6	

Port Interrupt Sharing

- Cortex-M0+: only 32 interrupt sources
- One Interrupt for all port pins
- Need to poll/check in ISR which pin triggered ISR

could throw another interrupt in the interrupt

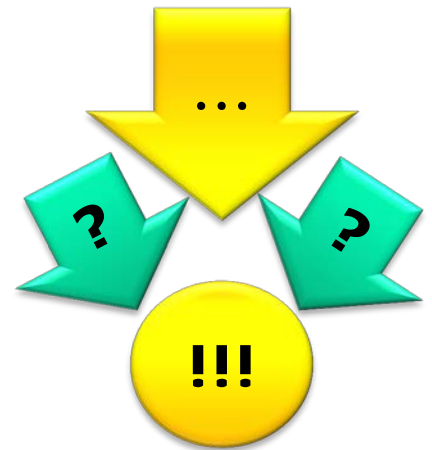
```

305  /** This ISR services the ivINT_PORTA interrupt shared by several
306  ** components.
307  ** This method is internal. It is used by Processor Expert only.
308  ** =====
309  */
310  PE_ISR(Cpu_ivINT_PORTA)
311  {
312      ExtIntLdd2_Interrupt();           /* Call the service routine */
313      ExtIntLdd3_Interrupt();           /* Call the service routine */
314      ExtIntLdd4_Interrupt();           /* Call the service routine */
315      ExtIntLdd5_Interrupt();           /* Call the service routine */
316  }
317
318  /*
319  ** =====
  
```

inside the ISR we need to read the status of the pin

Summary

- *Problem: we want to use keys/switches for our application*
- Keys
 - Pull-up/pull-down
- Synchronization
 - Realtime
 - Gadfly
 - Interrupts
- Creating Events for keys



Lab: Keys

- Robot: 1 push button
- Remote: 4 + 1 + 2 push buttons
- Keys.h, Keys.c
 - Implement polling for all keys
 - Add interrupts
 - Create events for key presses

