CG2271 Real Time Operating Systems

Lab Lecture 1.1

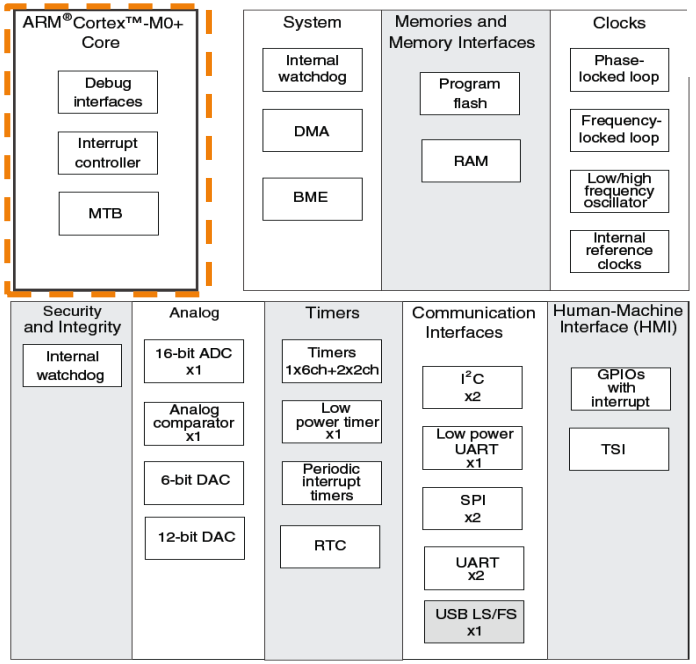
Introduction to the Cortex M0+ Microcontroller

Overview

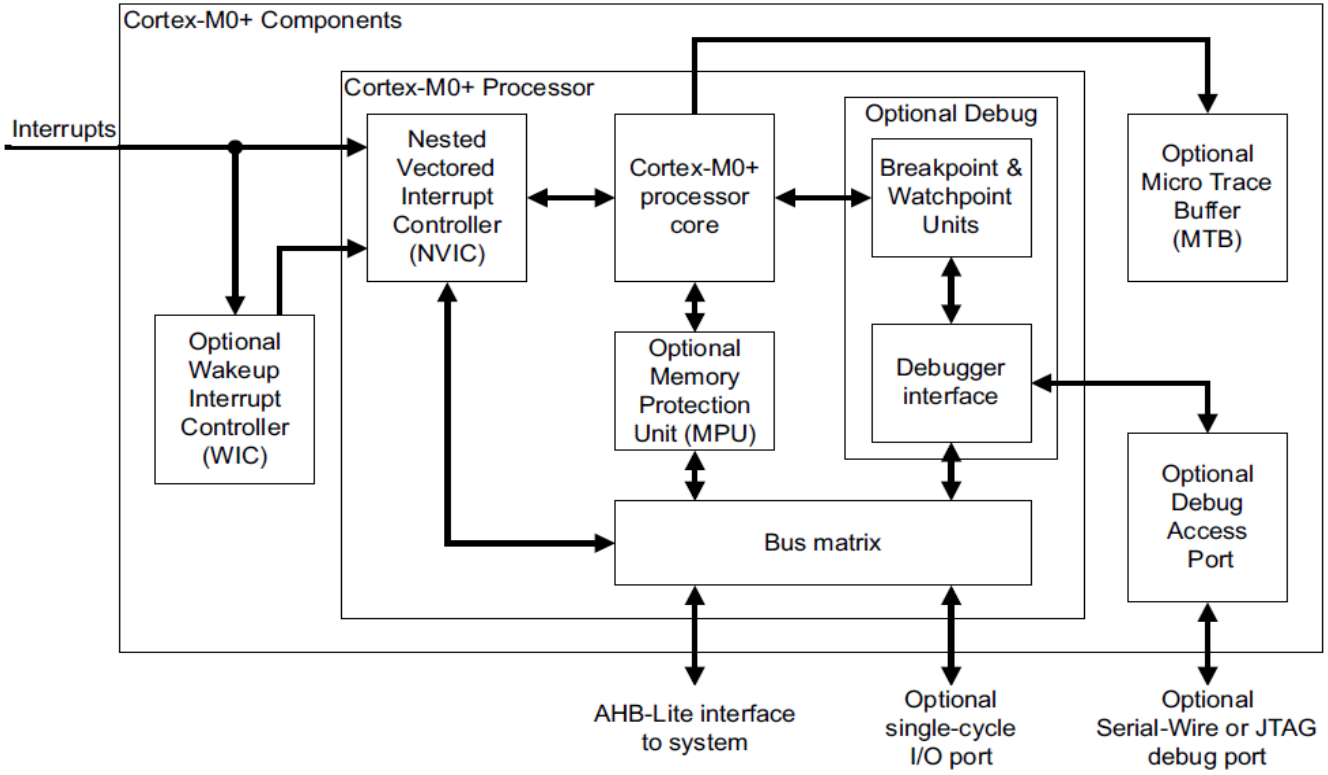
- Cortex-M0+ Processor Core Registers
- Memory System and Addressing

Microcontroller vs. Microprocessor

- Both have a CPU core to execute instructions
- Microcontroller has peripherals for embedded interfacing and control
  - Analog
  - Timers
  - Digital I/O
  - …



Cortex-M0+ Core

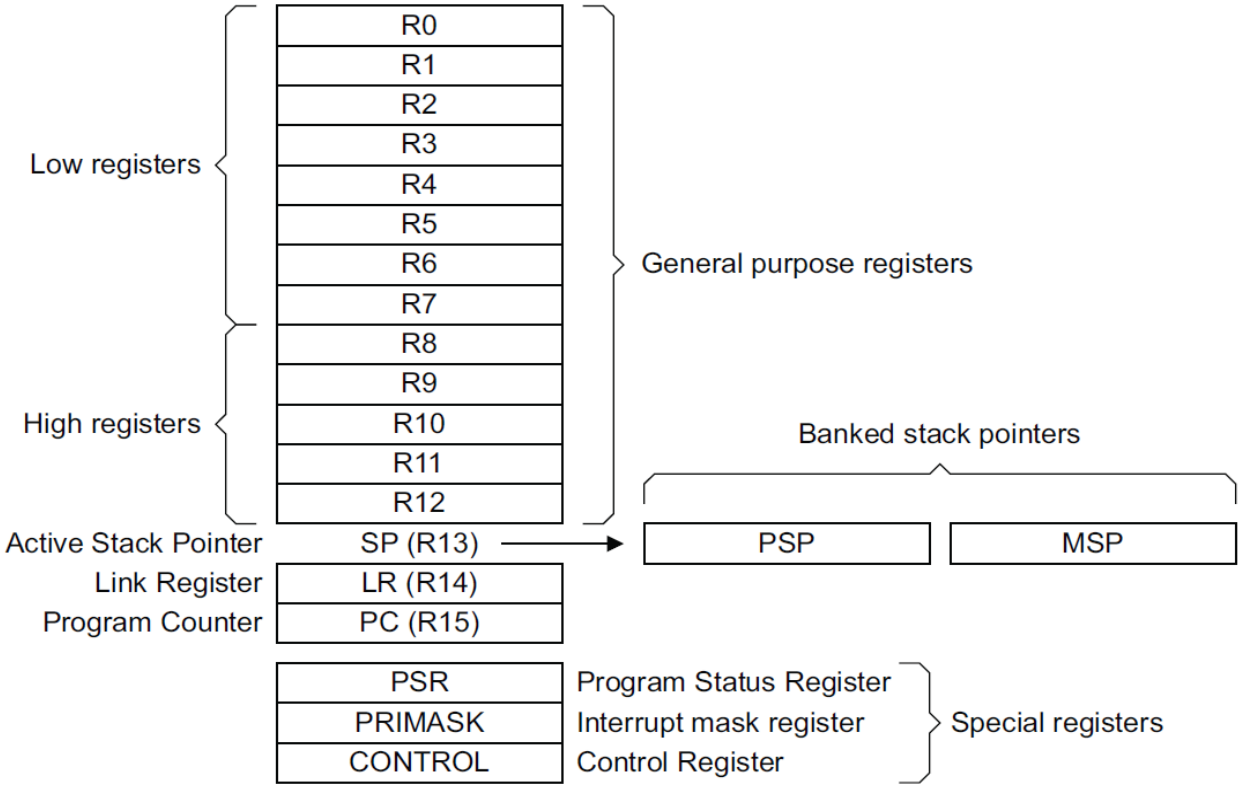


Architectures and Memory Speed

- Load/Store Architecture
  - Developed to simplify CPU design and improve performance
    - *Memory wall*: CPUs keep getting faster than memory
    - Memory accesses slow down CPU, limit compiler optimizations
    - Change instruction set to make most instructions *independent* of memory
  - Data processing instructions can access registers only
    1. Load data into the registers
    2. Process the data
    3. Store results back into memory
  - More effective when more registers are available

- Register/Memory Architecture
  - Data processing instructions can access memory or registers
  - Memory wall is not very high at lower CPU speeds (e.g. under 50 MHz)

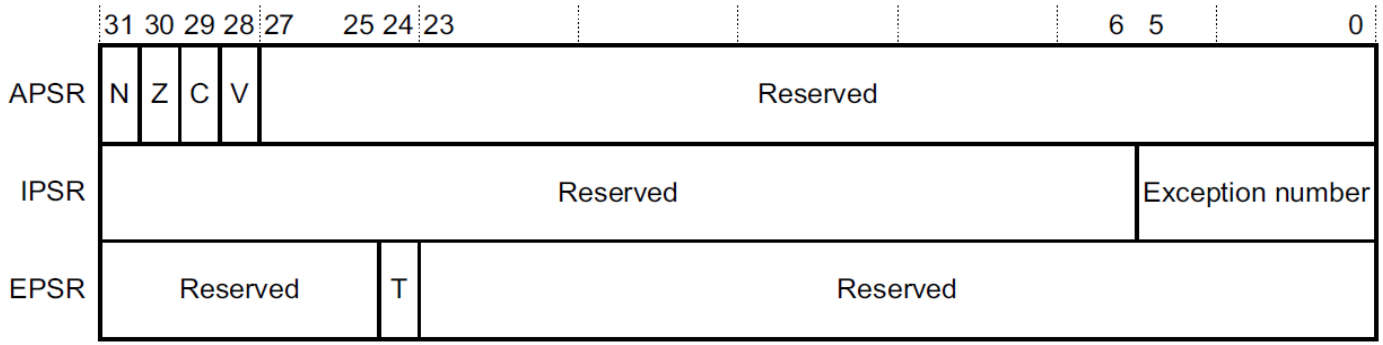
ARM Processor Core Registers



ARM Processor Core Registers (32 bits each)

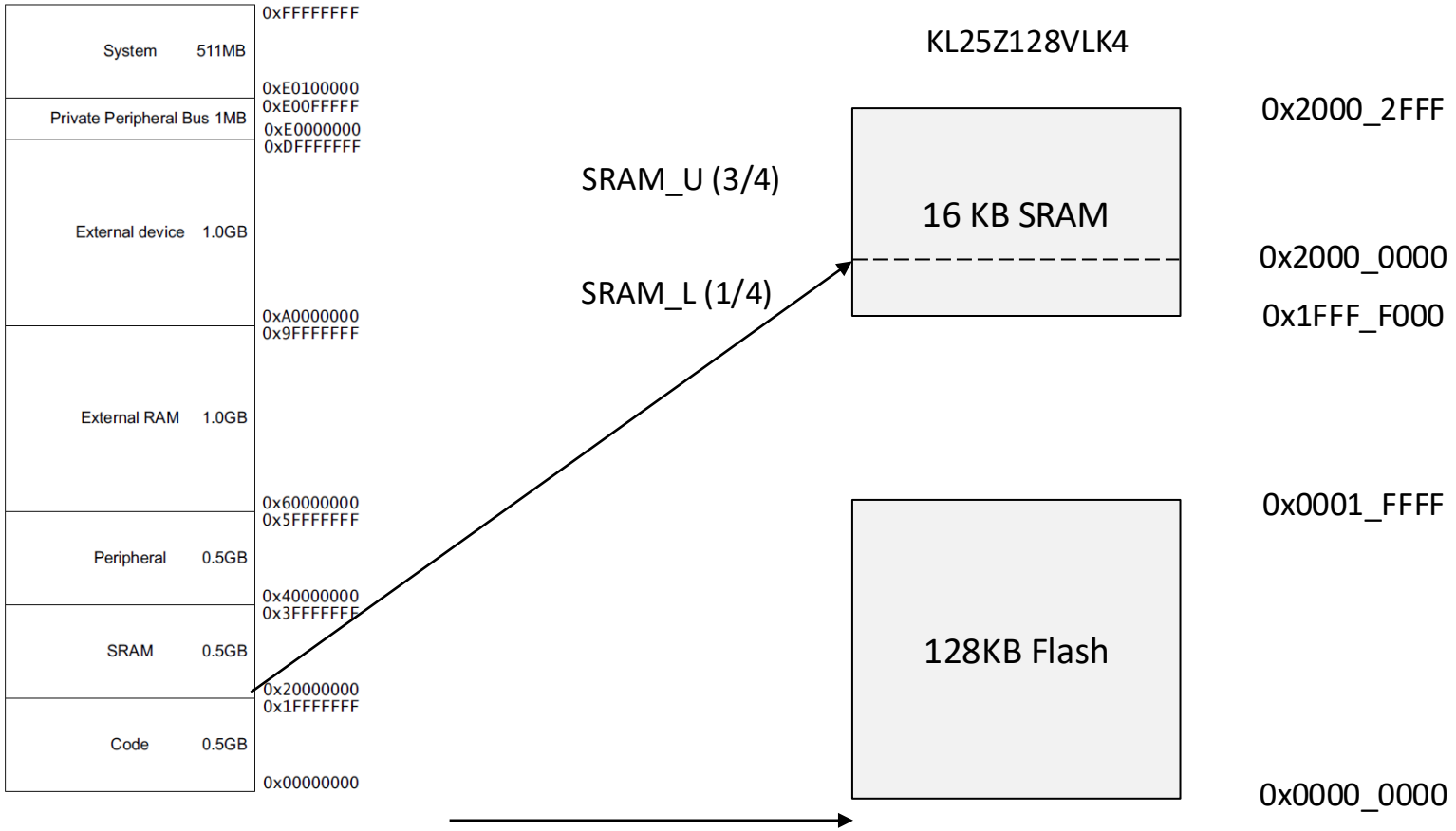
- R0-R12 - General purpose registers for data processing
- SP - Stack pointer (R13)
  - Can refer to one of two SPs
    - Main Stack Pointer (MSP)
    - Process Stack Pointer (PSP)
  - Uses MSP initially, and whenever in Handler mode
  - When in Thread mode, can select either MSP or PSP using SPSEL flag in CONTROL register.
- LR - Link Register (R14)
  - Holds return address when called with Branch & Link instruction (B&L)
- PC - program counter (R15)

ARM Processor Core Registers



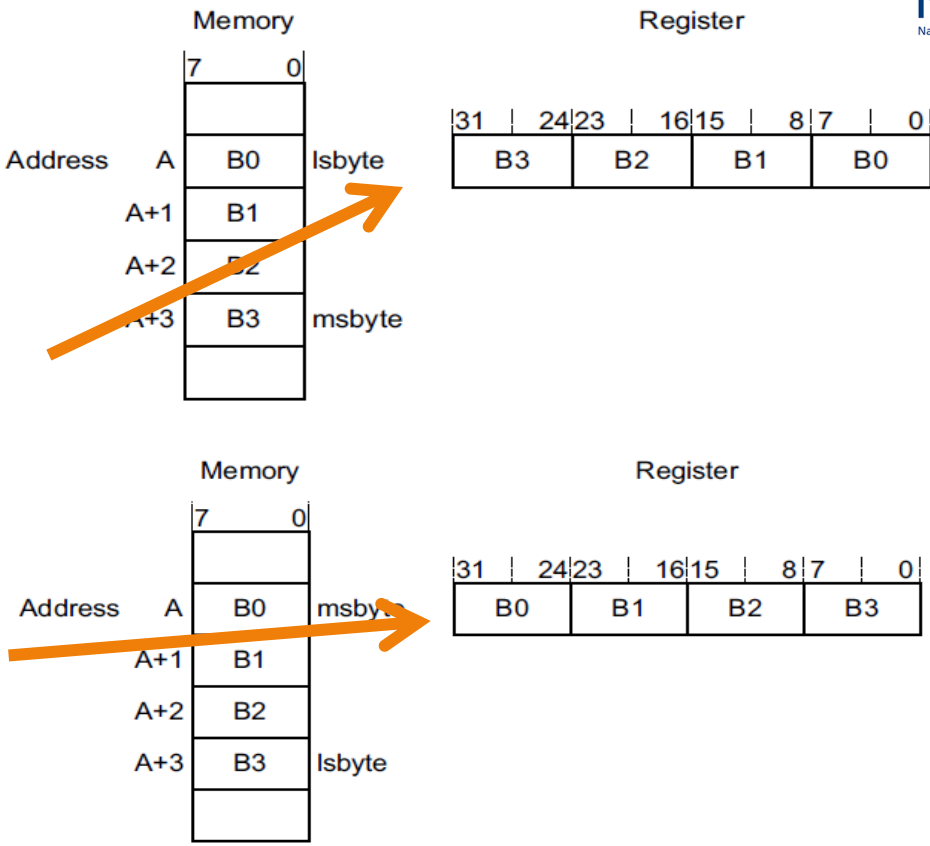
- Program Status Register (PSR) is three views of same register
  - Application PSR (APSR)
    - Condition code flag bits Negative, Zero, oVerflow, Carry
  - Interrupt PSR (IPSR)
    - Holds exception number of currently executing ISR
  - Execution PSR (EPSR)
    - Thumb state

Memory Maps For Cortex M0+ and MCU



Endianness

- For a multi-byte value, in what order are the bytes stored?
- Little-Endian: Start with least-significant byte
- Big-Endian: Start with most-significant byte



ARMv6-M Endianness

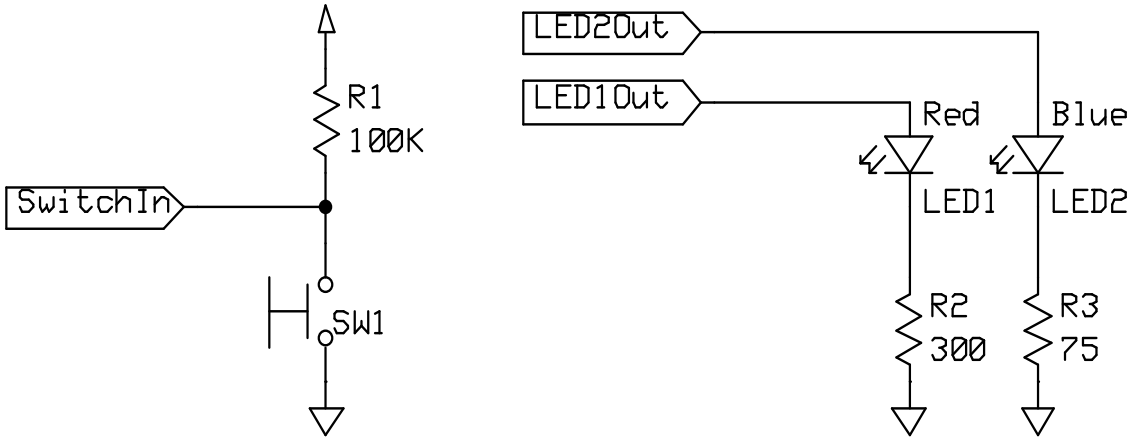
- Instructions are always little-endian
- Loads and stores to Private Peripheral Bus are always little-endian
- Data: Depends on implementation, or from reset configuration
  - Kinetis processors are little-endian

CG2271 Real Time Operating Systems

Lab Lecture 1.2
GPIO Programming
(Reference Guide Chapter 41)

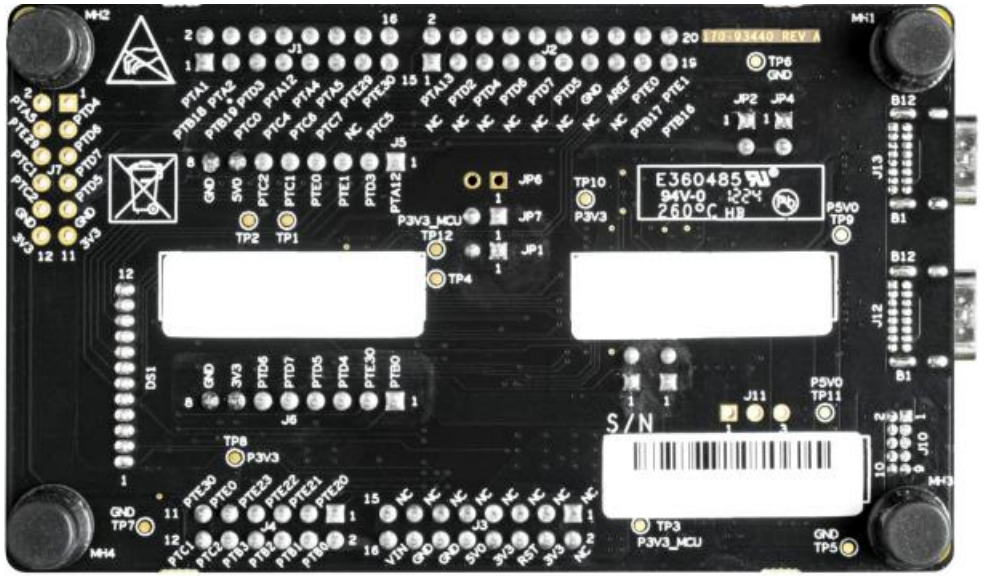
General Purpose I/O

- General Purpose I/O or GPIO pins provide binary input and output.
  - Ideal for reading switches and driving LEDs.
  - Input and output voltages on the MCXC444 chip is 3.3 volts, NOT 5 volts!



GPIO on the MCXC444

- GPIO pins on the MCXC444 are splits across 5 ports, GPIOA to GPIOE.
  - In theory this will give us $32 \times 5 = 160$ GPIO pins, but in practice not all pins are available.
  - Easiest to just check the pinouts on the underside of the FRDM-MCXC444 board to see what can be used.



Power Management on the MCXC444

- Almost all hardware on the MCXC444 is switched off by default, to save power.
  - I2C, SPI, UART, ADC, timers, GPIO, DMA, etc.
- Before using these devices, you need to set the corresponding bit in the correct System Clock Gating Control Register (SCGC).
  - Part of the System Integration Module (SIM).
  - Reference Guide: Sections 12.3.8 to 12.3.11
- For GPIO, the clock gating control bits are in the SIM_SCGC5 register.

Power Management on the MCXC444

- To turn on the clock for GPIOx (Replace x with A, B, C, D or E. E.g. GPIOE)

```
SIM->SCGC5 |= SIM_SCGC5_PORTx_MASK;
```

E.g. to turn on GPIOE

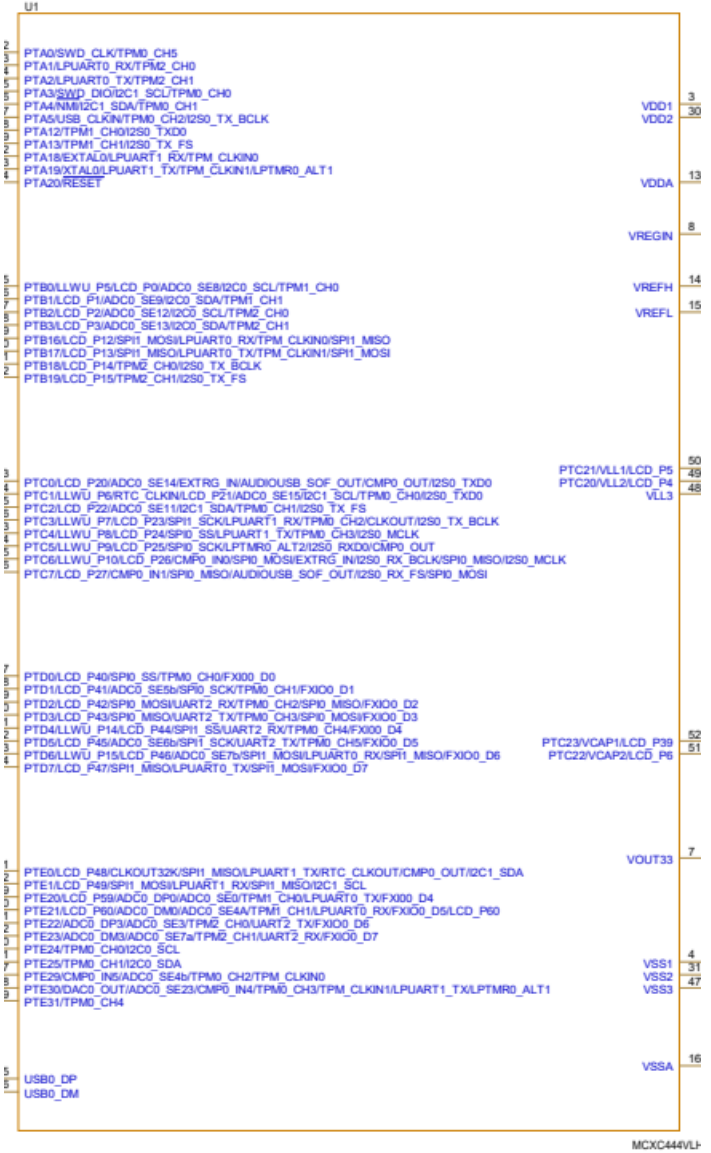
```
SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;
```

- To turn off:

```
SIM->SCGC5 &= ~SIM_SCGC5_PORTx_MASK;
```

Pin Multiplexing on the MCXC444

- Like all microcontrollers, the MCXC444 uses each pin for multiple purposes:
  - E.g. PTA1 (Port A pin 1) is shared with the receive pin of low power UART 0 (LPUART0), and channel 0 of timer 2 (TPM2)



Pin Multiplexing on the MCXC444

- We make use of the Pin Control Register (PCR) of each pin to choose the function we want, along with other configurations:
  - E.g. PORTA_PCR5 configures pin 5 of port A. (See Section 11.7.1 for details)

Address: Base address + 0h offset + (4d × i), where i=0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	*	*	*	0	*	0	*	0	*	*	*

Pin Multiplexing on the MCXC444

PORTx_PCR*n* field descriptions

Field	Description
31–25 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
24 ISF	Interrupt Status Flag This field is read-only for pins that do not support interrupt generation. The pin interrupt configuration is valid in all digital pin muxing modes.

Table continues on the next page...

Pin Multiplexing on the MCXC444

PORTx_PCRn field descriptions (continued)

Field	Description
	0 Configured interrupt is not detected. 1 Configured interrupt is detected. If the pin is configured to generate a DMA request, then the corresponding flag will be cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic 1 is written to the flag. If the pin is configured for a level sensitive interrupt and the pin remains asserted, then the flag is set again immediately after it is cleared.
23–20 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
19–16 IRQC	Interrupt Configuration This field is read-only for pins that do not support interrupt generation. The pin interrupt configuration is valid in all digital pin muxing modes. The corresponding pin is configured to generate interrupt/DMA request as follows: 0000 Interrupt Status Flag (ISF) is disabled. 0001 ISF flag and DMA request on rising edge. 0010 ISF flag and DMA request on falling edge. 0011 ISF flag and DMA request on either edge. 0100 Reserved. 0101 Reserved. 0110 Reserved. 0111 Reserved. 1000 ISF flag and Interrupt when logic 0. 1001 ISF flag and Interrupt on rising-edge. 1010 ISF flag and Interrupt on falling-edge. 1011 ISF flag and Interrupt on either edge. 1100 ISF flag and Interrupt when logic 1. 1101 Reserved. 1110 Reserved. 1111 Reserved.
15–11 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (Alternative 0) (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

Table continues on the next page

PORTx_PCRn field descriptions (continued)

Field	Description
6 DSE	Drive Strength Enable This field is read-only for pins that do not support a configurable drive strength. Drive strength configuration is valid in all digital pin muxing modes. 0 Low drive strength is configured on the corresponding pin, if pin is configured as a digital output. 1 High drive strength is configured on the corresponding pin, if pin is configured as a digital output.
5 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
4 PFE	Passive Filter Enable This field is read-only for pins that do not support a configurable passive input filter. Passive filter configuration is valid in all digital pin muxing modes. 0 Passive input filter is disabled on the corresponding pin. 1 Passive input filter is enabled on the corresponding pin, if the pin is configured as a digital input. Refer to the device data sheet for filter characteristics.
3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
2 SRE	Slew Rate Enable This field is read-only for pins that do not support a configurable slew rate. Slew rate configuration is valid in all digital pin muxing modes. 0 Fast slew rate is configured on the corresponding pin, if the pin is configured as a digital output. 1 Slow slew rate is configured on the corresponding pin, if the pin is configured as a digital output.
1 PE	Pull Enable This field is read-only for pins that do not support a configurable pull resistor. Refer to the Chapter of Signal Multiplexing and Signal Descriptions for the pins that support a configurable pull resistor. Pull configuration is valid in all digital pin muxing modes. 0 Internal pullup or pulldown resistor is not enabled on the corresponding pin. 1 Internal pullup or pulldown resistor is enabled on the corresponding pin, if the pin is configured as a digital input.
0 PS	Pull Select This bit is read only for pins that do not support a configurable pull resistor direction. Pull configuration is valid in all digital pin muxing modes. 0 Internal pulldown resistor is enabled on the corresponding pin, if the corresponding PE field is set. 1 Internal pullup resistor is enabled on the corresponding pin, if the corresponding PE field is set.

Choosing the Pin Function

- We make use of the MUX bits in the PORTx_PCRy register (pin y of port x).
  - The tables in Section 10.3.1 show us which number to use for each function.

64 MAP BGA	64 LQFP	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
A1	1	PTE0	DISABLED	LCD_P48	PTE0/ CLKOUT32K	SPI1_MISO	LPUART1_TX	RTC_CLKOUT	CMP0_OUT	I2C1_SDA	LCD_P48
B1	2	PTE1	DISABLED	LCD_P49	PTE1	SPI1_MOSI	LPUART1_RX		SPI1_MISO	I2C1_SCL	LCD_P49

  - For example, if we wanted to use GPIO on PTE1, the correct function is ALT1.

Choosing the Pin Function

- We access PORTx_PCRy using:

```
PORTx->PCR[y]
```

- Example to choose GPIO for pin PTE1:

```
// Zero all the MUX bits
PORTE->PCR[1]  &= ~PORT_PCR_MUX_MASK;

// Choose ALT1 for PTE1
PORTE->PCR[1] |= PORT_PCR_MUX(1);
```

Setting the Pin Direction

- Pins can be input or output. To set the direction, we set the corresponding bit in the GPIOx_PDDR (pin data direction register):
  - 0 – Pin is input, 1 – pin is output

```
// Set pin pinnum on port x as input
GPIOx->PDDR &= ~(1 << pinnum);

// Set pin pinnum on port x as output
GPIOx ->PDDR |= (1 << pinnum);

// Set PTE1 as input
GPIOE->PDDR &= ~(1 << 1);
```

Writing to the GPIO pin

- To write to a GPIO pin we must first set it as output. Here we use PTE1 as an example:

```
// Set PTE1 as output
GPIOE->PDDR |= (1 << 1);
```

- We now have several choices on how to set/clear/toggle the GPIO pin:
  - GPIOx_PCOR: Write a 1 to clear the pin
  - GPIOx_PSOR: Write a 1 to set the pin
  - GPIOx_PDOR: Write a 1 to set and a 0 to clear
  - GPIOx_PTOR: Write a 1 to toggle the pin

Writing to the GPIO pin

- Example to clear PTE1:

```
#define PIN1    1

// Set PTE1 to output
GPIOE->PDDR |= (1 << PIN1);

// Clear PTE1
GPIOE->PCOR |= (1 << PIN1);

// Set PTE1:
GPIOE->PSOR |= (1 << PIN1);

// Toggle PTE1:
GPIOE->PTOR |= (1 << PIN1);
```

Reading a GPIO Pin

- To read a GPIO pin, we set the pin as input and then use the GPIOx_PDIR register. We take PTB2 as an example.

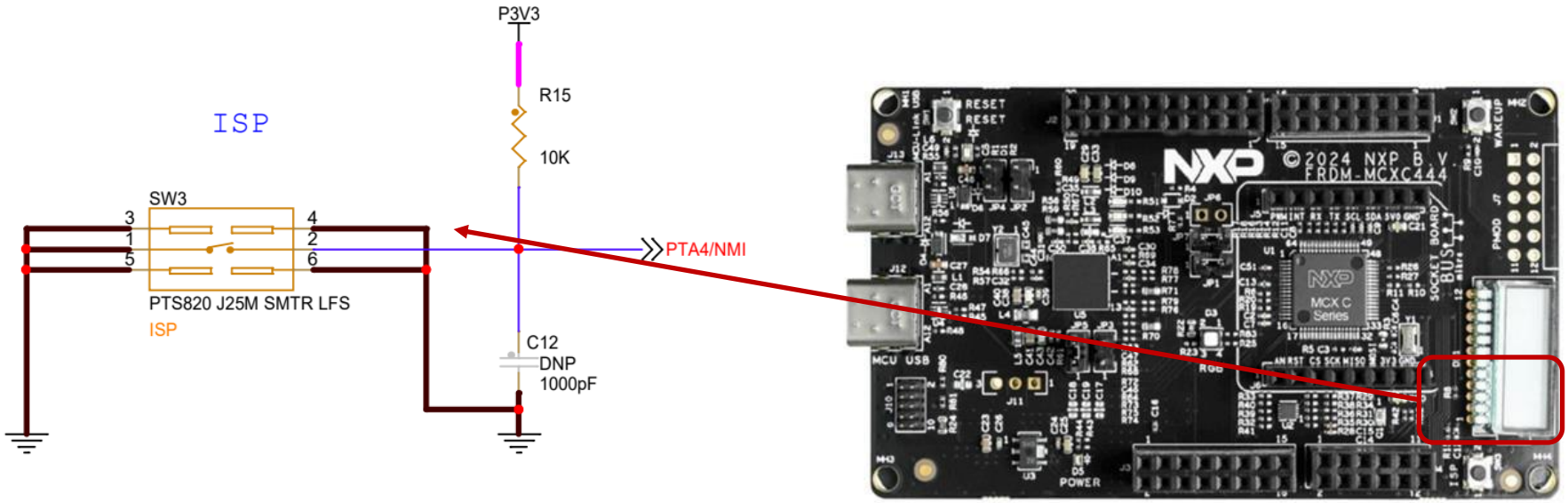
```
#define PIN2    2

// Set PTB2 as input
GPIOB->PDDR &= ~(1 << PIN2)

// If PTB2 is 1 do something
if(GPIOB->PDIR & (1 << PIN2)) {
    .. Do something ..
}
```

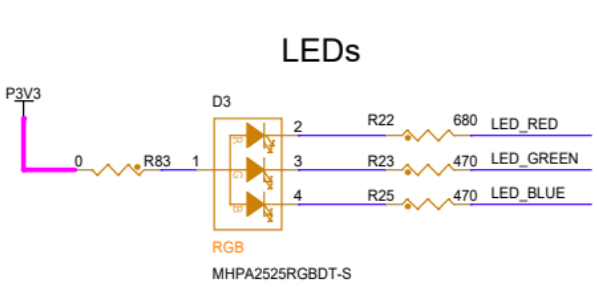
Putting it All Together

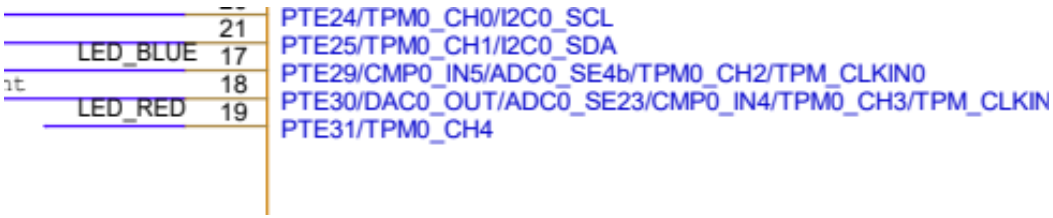
- We will now study an example where we use a switch on the FRDM-MCXC444 board to toggle an LED.
  - We will use switch SW3 on the bottom right of the board, which is connected to PTA4:
    - Note that the switch is active-low.



Putting it All Together

- The FRDM-MCXC444 board comes with an RGB LED configured in active-low configuration (i.e. we write a 0 to turn on the LED and a 1 to turn it off).
  - We will use the red LED, connected to PTE31





Putting it All Together

- Steps:
  - Define our pin numbers:

```
#define SW_PIN   4
#define LED_PIN  31
```

  - Turn on the clocking to Ports A and E:

```
// Turn on the clocking to ports A and E
SIM->SCGC5 |= (SIM_SCGC5_PORTA_MASK
      | SIM_SCGC5_PORTE_MASK);
```

Putting it All Together

- Steps:
  - Select the correct pin function for PTA4 and PTE31:

64 MAP BGA	64 LQFP	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
G5	26	PTA4	NMI_b		PTA4	I2C1_SDA	TPM0_CH1				NMI_b
H3	19	PTE31	DISABLED		PTE31		TPM0_CH4				

```
// Choose the GPIO function for PTA4
PORTA->PCR[SW_PIN] &= ~PORT_PCR_MUX_MASK;
PORTA->PCR[SW_PIN] |= PORT_PCR_MUX(1);

// Choose the GPIO function for PTE31
PORTE->PCR[LED_PIN] &= ~PORT_PCR_MUX_MASK;
PORTE->PCR[LED_PIN] |= PORT_PCR_MUX(1);
```

Putting it All Together

- Set the switch at PTA4 and the LED at PTE31 as input and output respectively.

```
// Set PTA4 as input
GPIOA->PDDR &= ~(1 << SW_PIN);

// Set PTE31 as output
GPIOE->PDDR |= (1 << LED_PIN);
```

- Switch off the LED.

```
// Switch off the red LED at PTE31. Note that it is active low
GPIOE->PSOR |= (1 << LED  PIN);
```

Putting it All Together

- Finally poll the switch at PTA4 and toggle the LED at PTE31 whenever it is pressed.

```
while(1) {

   // If SW_PIN is high, toggle PTE31
   if(!(GPIOA->PDIR & (0b1 << SW_PIN))) {
      GPIOE->PTOR |= (1 << LED_PIN);
   }
}
```

Conclusion

- In this lecture we've looked at how to configure, read and write GPIO pins on the MCXC444.
- In the next lecture we will look at the interrupt architecture on this chip, and how to program the GPIO interrupts.

CG2271 Real Time Operating Systems

Lab Lecture 1.3

Interrupt Programming

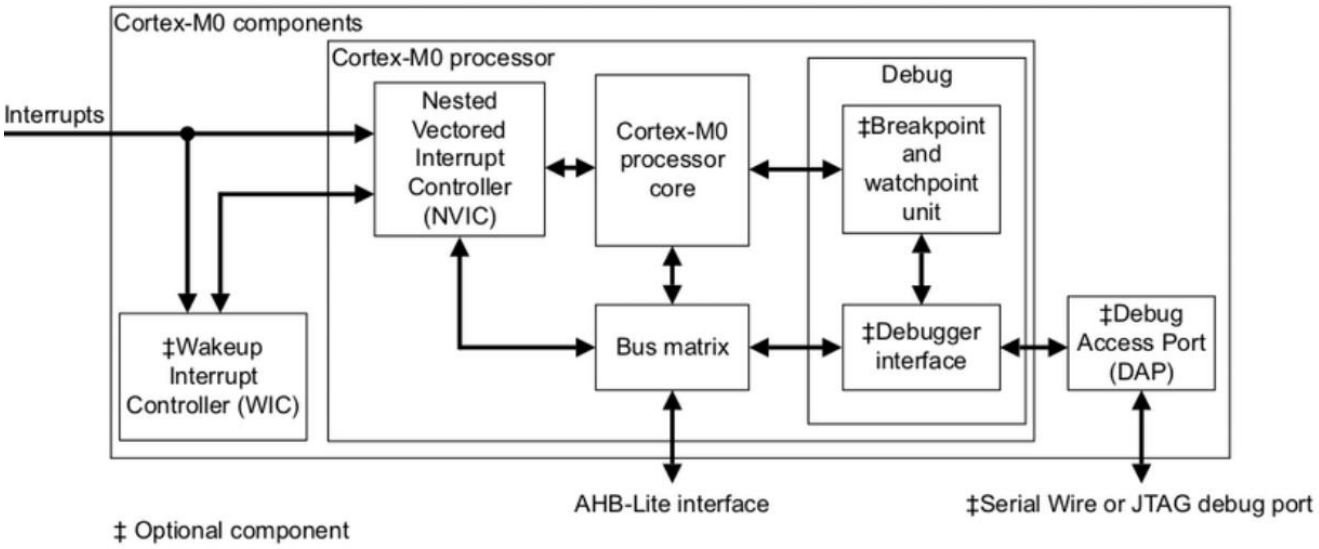
(Reference Guide Section 3.2)

Lab Lecture 1.3 – Interrupt Programming

INTRODUCTION

Introduction

- The MCXC444 uses a priority interrupt system centred around a Nested Vector Interrupt Controller (NVIC).
  - All interrupts generated by devices on the MCU pass through the NVIC:



Lab Lecture 1.3 – Interrupt Programming

NESTED VECTORED INTERRUPT CONTROLLER

Nested Vectored Interrupt Controller

- The NVIC is:
  - Nested:
    - Meaning that higher priority interrupts can interrupt ongoing lower priority interrupts.
    - Alternatively, lower priority interrupts are delayed until higher priority interrupts are completed.
  - Vectored:
    - When an interrupt is triggered, execution is “vectored” to a particular interrupt service routine (ISR) to handle the interrupt.
    - To facilitate this, every interrupt is assigned a number – “vector”.
    - Interrupt numbers 0 to 15 are reserved for the MCU core.
      - System errors, non-maskable interrupt, RESET, etc.
    - Interrupts 16 and above are for various hardware units.

NVIC Vectors

- Core Vectors (0 to 15)

Address	Vector	IRQ[1]	NVIC IPR register number[2]	Source module	Source description
ARM core system handler vectors					
0x0000_0000	0	—	—	ARM core	Initial stack pointer
0x0000_0004	1	—	—	ARM core	Initial program counter
0x0000_0008	2	—	—	ARM core	Non-maskable interrupt (NMI)
0x0000_000C	3	—	—	ARM core	Hard fault
0x0000_0010	4	—	—	—	—
0x0000_0014	5	—	—	—	—
0x0000_0018	6	—	—	—	—
0x0000_001C	7	—	—	—	—
0x0000_0020	8	—	—	—	—
0x0000_0024	9	—	—	—	—
0x0000_0028	10	—	—	—	—
0x0000_002C	11	—	—	ARM core	Supervisor call (SVCall)
0x0000_0030	12	—	—	—	—
0x0000_0034	13	—	—	—	—
0x0000_0038	14	—	—	ARM core	Pendable request for system service (PendableSrvReq)
0x0000_003C	15	—	—	ARM core	System tick timer (SysTick)

NVIC Vectors

- Non-core Vectors (≥ 16)

Non-Core Vectors					
0x0000_0040	16	0	0	DMA	DMA channel 0 transfer complete and error
0x0000_0044	17	1	0	DMA	DMA channel 1 transfer complete and error
0x0000_0048	18	2	0	DMA	DMA channel 2 transfer complete and error
0x0000_004C	19	3	0	DMA	DMA channel 3 transfer complete and error
0x0000_0050	20	4	1	—	—
0x0000_0054	21	5	1	FTFA	Command complete and read collision
0x0000_0058	22	6	1	PMC	Low-voltage detect, low-voltage warning
0x0000_005C	23	7	1	LLWU	Low Leakage Wakeup
0x0000_0060	24	8	2	I^2C0	Status and Timeout and wakeup flags
0x0000_0064	25	9	2	I^2C1	Status and Timeout and wakeup flags
0x0000_0068	26	10	2	SPI0	Single interrupt vector for all sources
0x0000_006C	27	11	2	SPI1	Single interrupt vector for all sources
0x0000_0070	28	12	3	LPUART0	Status and error
0x0000_0074	29	13	3	LPUART1	Status and error
0x0000_0078	30	14	3	UART2 or FlexIO	Status and error—

NVIC Vectors

- Non-core Vectors (≥ 16)

Address	Vector	IRQ[1]	NVIC IPR register number[2]	Source module	Source description
0x0000_007C	31	15	3	ADC0	Conversion complete
0x0000_0080	32	16	4	CMP0	Rising or falling edge of comparator output
0x0000_0084	33	17	4	TPM0	Overflow or channel interrupt
0x0000_0088	34	18	4	TPM1	Overflow or channel interrupt
0x0000_008C	35	19	4	TPM2	Overflow or channel interrupt
0x0000_0090	36	20	5	RTC	Alarm interrupt
0x0000_0094	37	21	5	RTC	Seconds interrupt
0x0000_0098	38	22	5	PIT	Single interrupt vector for all channels
0x0000_009C	39	23	5	I^2S0	Single interrupt vector for all sources
0x0000_00A0	40	24	6	USB	—
0x0000_00A4	41	25	6	DAC0	—
0x0000_00A8	42	26	6	—	—
0x0000_00AC	43	27	6	—	—
0x0000_00B0	44	28	7	LPTMR0	LP Timer compare match
0x0000_00B4	45	29	7	SLCD	—
0x0000_00B8	46	30	7	Port control module	Pin detect (Port A)
0x0000_00BC	47	31	7	Port control module	Pin detect (Single interrupt vector for Port C, Port D)

Lab Lecture 1.3 – Interrupt Programming

EXTERNAL INTERRUPTS

External Interrupts

- Like the ATMega328P on the Arduino, GPIO pins on the MCXC444 can be configured to generate interrupts
  - Divided into two groups:
    - Port A
    - Ports C and D are combined into one group.

External Interrupts

- Every port (A, C, and D) has an Interrupt Status Flag Register (ISFR), e.g. PORTA_ISFR.
  - If an interrupt is trigged on a pin (e.g. PTA4), bit 4 in PORTA->ISFR would be set.

Address: Base address + A0h offset

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ISF																															
W	w1c																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PORTx_ISFR field descriptions

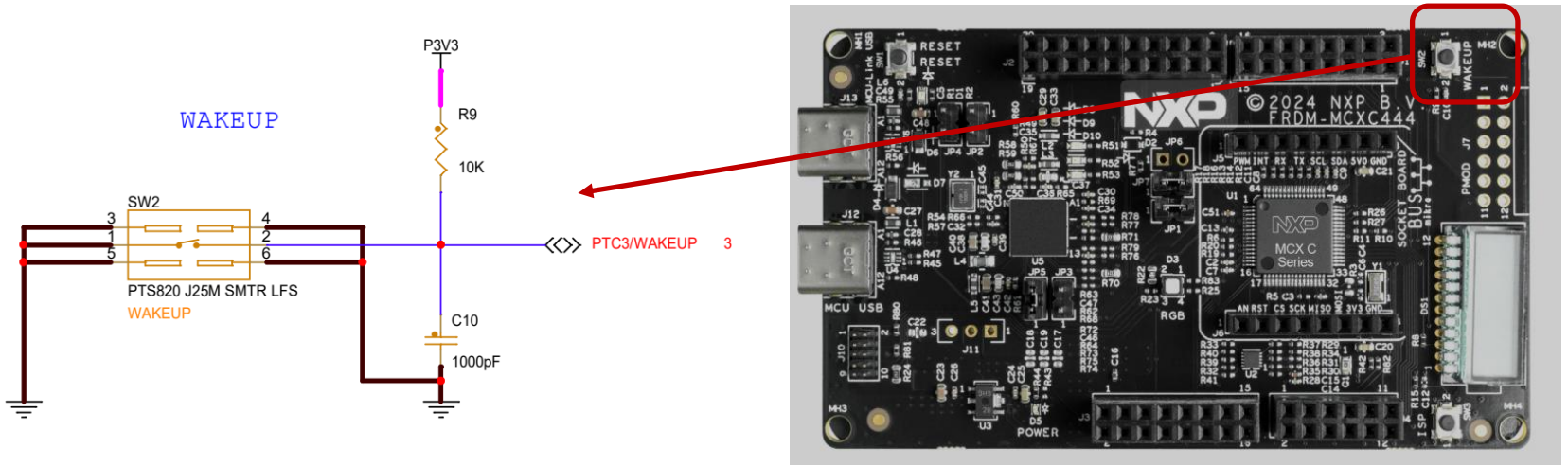
Field	Description
ISF	Interrupt Status Flag Each bit in the field indicates the detection of the configured interrupt of the same number as the field. 0 Configured interrupt is not detected. 1 Configured interrupt is detected. If the pin is configured to generate a DMA request, then the corresponding flag will be cleared automatically at the completion of the requested DMA transfer. Otherwise, the flag remains set until a logic 1 is written to the flag. If the pin is configured for a level sensitive interrupt and the pin remains asserted, then the flag is set again immediately after it is cleared.

Lab Lecture 1.3 – Interrupt Programming

CONFIGURING EXTERNAL INTERRUPTS

Configuring External Interrupts

- We will now look at an example of how to configure an external interrupt.
  - We choose SW2 on the FRDM-MCXC444 board, which is connected to PTC3.
  - Gives us a chance to see how to use the combined PORTC_PORTD interrupt vector.



Initial Steps

- Step 0: Define SWITCH_PIN:

```
#define SWITCH_PIN   3  // SW2 is at PTC3
```

- Step 1: Disable the PORTC_PORTD interrupt.

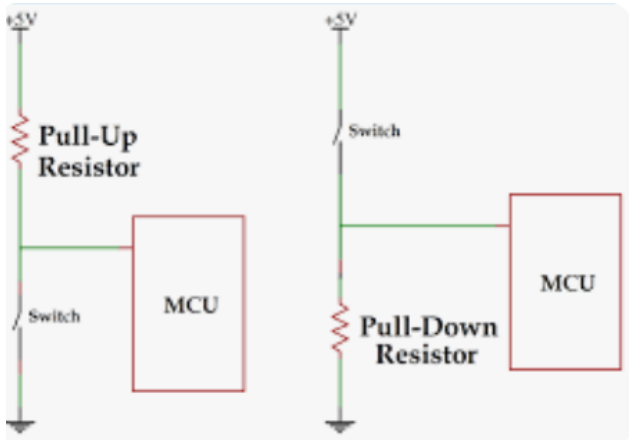
```
// Disable interrupts
NVIC_DisableIRQ(PORTC_PORTD_IRQn);
```

- Step 2: Turn on clocking to Port C using SIM_SCGC5:

```
SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;
```

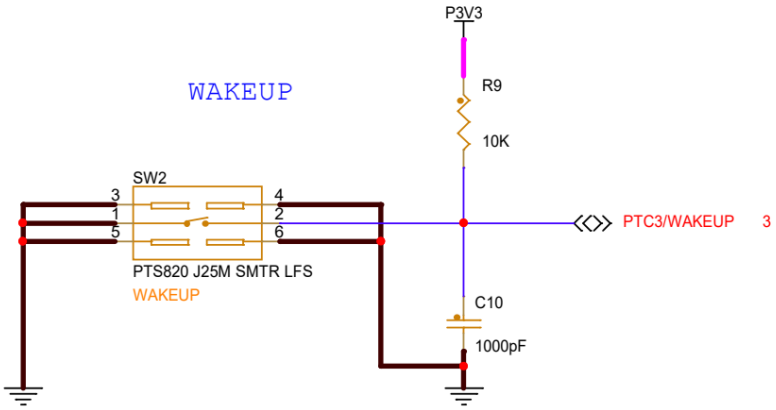
Pull-up/down Resistors

- Step 3: Enable internal pull-up resistor.
  - The MCXC444 has built-in pull-up or pull-down resistors.
    - Pull-up resistors: weakly drives a line high until an event (e.g. closing a switch) drives it low.
    - Pull-down resistor: Weakly drives a line low until an event drives it high.



Pull-up/down Resistors

- SW2 on the FRDM-MCXC444 is already pulled high:



- However just for learning purposes we will activate the internal pull-up resistor on the MCXC444.

Pull-up/down Resistors

- The pull-up/pull-down resistors are configured using the PE and PS bits in the PORTx_PCRy pin configuration register.

Address: Base address + 0h offset + (4d × i), where i=0d to 31d

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	*	*	*	0	*	0	*	0	*	*	*

Pull-up/down Resistors

- PE (Pull enable):
  - 0 - Disable pull-up/down resistor, 1 – Enable
- PS (Pull select):
  - 0 – Enable pull-down resistor if PE = 1.
  - 1 – Enable pull-up resistor if PE = 1

- We are going to enable the pull-up resistor and so PS = PE = 1.

```
// Set pullup resistor
PORTC->PCR[SWITCH_PIN] &= ~PORT_PCR_PS_MASK;
PORTC->PCR[SWITCH_PIN] |= PORT_PCR_PS(1);
PORTC->PCR[SWITCH_PIN] &= ~PORT_PCR_PE_MASK;
PORTC->PCR[SWITCH_PIN] |= PORT_PCR_PE(1);
```

Configure Pin as GPIO and Input

- Step 4: We configure PTC3 as GPIO and as an input pin (configuring as input is optional):

64 MAP BGA	64 LQFP	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
C8	46	PTC3/ LLWU_P7	LCD_P23	LCD_P23	PTC3/ LLWU_P7	SPI1_SCK	LPUART1_RX	TPM0_CH2	CLKOUT	I2S0_TX_ BCLK	LCD_P23

```
// Configure as GPIO
PORTC->PCR[SW2] &= ~PORT_PCR_MUX_MASK;
PORTC->PCR[SW2] |= PORT_PCR_MUX(1);
```

Configure Interrupt Triggering Type

- Step 5: Configure Triggering Type.
  - Like the ATMega328P, the MCXC444's interrupts can be triggered in several ways:
  - Controlled using the IRQC (Interrupt Configuration) bits of PORTx_PCRy:

Address: Base address + 0h offset + (4d × i), where i=0d to 31d

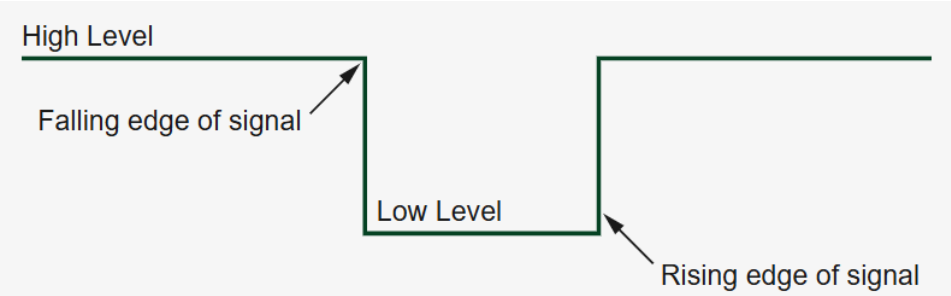
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Configure Interrupt Triggering Type

19–16 IRQC	Interrupt Configuration This field is read-only for pins that do not support interrupt generation. The pin interrupt configuration is valid in all digital pin muxing modes. The corresponding pin is configured to generate interrupt/DMA request as follows: 0000 Interrupt Status Flag (ISF) is disabled. 0001 ISF flag and DMA request on rising edge. 0010 ISF flag and DMA request on falling edge. 0011 ISF flag and DMA request on either edge. 0100 Reserved. 0101 Reserved. 0110 Reserved. 0111 Reserved. 1000 ISF flag and Interrupt when logic 0. 1001 ISF flag and Interrupt on rising-edge. 1010 ISF flag and Interrupt on falling-edge. 1011 ISF flag and Interrupt on either edge. 1100 ISF flag and Interrupt when logic 1. 1101 Reserved. 1110 Reserved. 1111 Reserved.

Configure Interrupt Triggering Type

- Since our switch is active low (i.e. pulls PTC3 low when pressed), we will trigger on the falling edge (0b1010)




```
// Configure the interrupt for falling edge
PORTC->PCR[SWITCH_PIN] &= ~PORT_PCR_IRQC_MASK;
PORTC->PCR[SWITCH_PIN] |= PORT_PCR_IRQC(0b1010);
```

Setting IRQ Priority

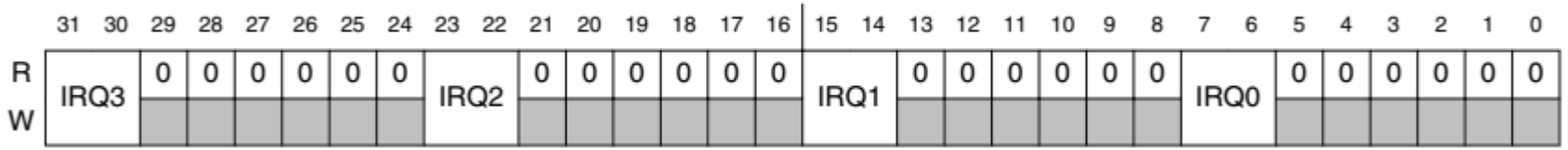
- Step 6: Set the Interrupt Priority Level.
  - The NVIC stores priority levels for each interrupt in a set of 8 interrupt priority registers (IPRs) in this format.

IPR Number

Non-Core Vectors					
0x0000_0040	16	0	0	DMA	DMA channel 0 transfer complete and error
0x0000_0044	17	1	0	DMA	DMA channel 1 transfer complete and error
0x0000_0048	18	2	0	DMA	DMA channel 2 transfer complete and error
0x0000_004C	19	3	0	DMA	DMA channel 3 transfer complete and error
0x0000_0050	20	4	1	—	—
0x0000_0054	21	5	1	FTFA	Command complete and read collision
0x0000_0058	22	6	1	PMC	Low-voltage detect, low-voltage warning
0x0000_005C	23	7	1	LLWU	Low Leakage Wakeup
0x0000_0060	24	8	2	I^2C0	Status and Timeout and wakeup flags
0x0000_0064	25	9	2	I^2C1	Status and Timeout and wakeup flags
0x0000_0068	26	10	2	SPI0	Single interrupt vector for all sources
0x0000_006C	27	11	2	SPI1	Single interrupt vector for all sources
0x0000_0070	28	12	3	LPUART0	Status and error
0x0000_0074	29	13	3	LPUART1	Status and error
0x0000_0078	30	14	3	UART2 or FlexIO	Status and error—

Setting IRQ Priority

- Each IPR stores the priority number for four interrupts.
  - Each priority is represented by a 2 bit number + 6 0's.



- The priority levels are therefore:
  - 00|000000 = 0 (highest)
  - 01|000000 = 64
  - 10|000000 = 128
  - 11|000000 = 192 (lowest)

Setting IRQ Priority

- We set our SW2 button as the lowest priority:

```
// Set NVIC priority to 192,
// lowest priority
NVIC_SetPriority(PORTC_PORTD_IRQn, 192);
```

- Step 7: Clear existing interrupts and re-enable.

```
// Clear pending interrupts and enable interrupts
NVIC_ClearPendingIRQ(PORTC_PORTD_IRQn);
NVIC_EnableIRQ(PORTC_PORTD_IRQn);
```

Full Setup Code

```
// SW2 is connected to PTC3
void initSW2Interrupt() {

    // Disable PortC_D Interrupts
    NVIC_DisableIRQ(PORTC_PORTD_IRQn);

    // SW2 is at PTC3. Enable port C
    SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;

    // Configure as GPIO
    PORTC->PCR[SW2] &= ~PORT_PCR_MUX_MASK;
    PORTC->PCR[SW2] |= PORT_PCR_MUX(1);

    // Configure pullup
    PORTC->PCR[SW2] &= ~PORT_PCR_PS_MASK;
    PORTC->PCR[SW2] |= PORT_PCR_PS(1);

    // Configure as input
    GPIOC->PDDR &= ~(1 << SW2);


    PORTC->PCR[SW2] &= ~PORT_PCR_PE_MASK;
    PORTC->PCR[SW2] |= PORT_PCR_PE(1);


    // Configure IRQ type to trigger on rising edge
    // This is 0b1001 or 9 or 0x9
    PORTC->PCR[SW2] &= ~PORT_PCR_IRQC_MASK;
    PORTC->PCR[SW2] |= PORT_PCR_IRQC(0b1001);

    // Set lowest priority
    NVIC_SetPriority(PORTC_PORTD_IRQn, 192);

    // Clear pending interrupts
    NVIC_ClearPendingIRQ(PORTC_PORTD_IRQn);

    // Enable interrupts
    NVIC_EnableIRQ(PORTC_PORTD_IRQn);
}
```

Lab Lecture 1.3 – Interrupt Programming

WRITING THE ISR

Writing the ISR

- To write the ISR, we need to capture the PORTC_PORTD_IRQn interrupt vector.
  - This vector invokes the PORTC_PORTD_IRQHandler function.
- Steps in your ISR:
  - Clear the pending IRQ.
  - Use the ISFR register of the relevant port (in this case port C or port D) to see if the interrupt we are interested in is triggered.
  - Write a “1” to the corresponding ISFR bit to reset it back to 0.
- Code is shown on next page.

Writing the ISR

```
// Interrupt handler for PORT C/D
void PORTC_PORTD_IRQHandler() {

    NVIC_ClearPendingIRQ(PORTC_PORTD_IRQn);

    // We don't actually have to test that it is
    // PTC3 since there is only one interrupt
    // configured, but this shows how to do it.
    if(PORTC->ISFR & (1 << SWITCH_PIN)) {
        count = (count + 1) % 6;
    }

    // Write a 1 to clear the ISFR bit (yes, 1, not 0)
    PORTC->ISFR |= (1 << SWITCH_PIN);
}
```

Useful Functions and Macros

- Some useful functions and macros:
  - Interrupt vectors for the ports:
    - PORTA_IRQn, PORTC_PORTD_IRQn
  - NVIC Functions:
    - NVIC_DisableIRQ(vector): Disables interrupt *vector*
      - *NVIC_DisableIRQ(PORTA_IRQn);*
    - NVIC_EnableIRQ(vector): Enables interrupt *vector*
      - *NVIC_EnableIRQ(PORTA_IRQn);*
    - NVIC_SetPriority(vector, priority): Sets the priority level of the interrupt
      - *NVIC_SetPriority(PORTA_IRQn, 64);*
    - *NVIC_ClearPendingIRQ(vector): Clears pending interrupt requests.*
      - *NVIC_ClearPendingIRQ(PORTA_IRQn);*

Useful Functions and Macros

- IRQ Handler Names:

```
void PORTA_IRQHandler() {
}; // For Port A


void PORTC_PORTD_IRQHandler() {
}; // for Ports C and D
```

Conclusion

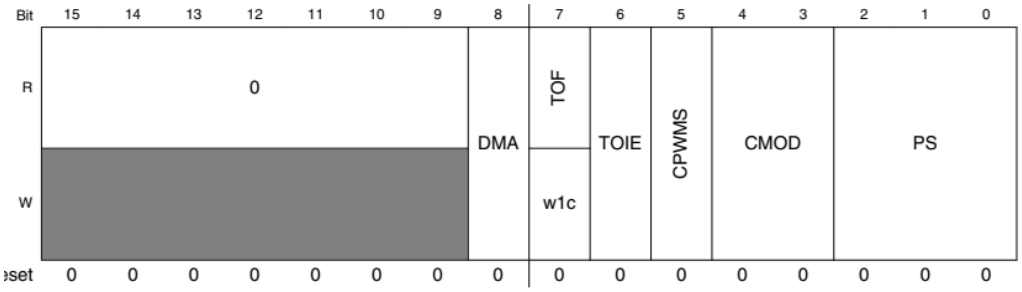
- In this lecture we looked at how to configure external interrupts on Ports A, C and D.
- In the next lecture we will see interrupts again as we learn how to program the Timer/PWM Modules (TPMs) on the MCXC444.

Appendix: Common Register Handling Mistakes

Setting/Clearing Bits (From EPP2)

- Setting a Bit
  - REGISTER |= (1 << BIT_POS)
- Clearing a Bit
  - REGISTER &= ~(1 << BIT_POS)

TPMx_SC




```
153 #define TPM_SC_CMOD_MASK                         (0x18U)
154 #define TPM_SC_CMOD_SHIFT                        (3U)
155/*! CMOD - Clock Mode Selection
156  *  0b00..TPM counter is disabled
157  *  0b01..TPM counter increments on every TPM counter clock
158  *  0b10..TPM counter increments on rising edge of TPM_EXTCLK synchronized to the TPM counter clock
159  *  0b11..Reserved
160  */
161 #define TPM_SC_CMOD(x)                           (((uint32_t)(((uint32_t)(x)) << TPM_SC_CMOD_SHIFT)) & TPM_SC_CMOD_MASK)
```

Field	Description
	0 TPM counter has not overflowed. 1 TPM counter has overflowed.
6 TOIE	Timer Overflow Interrupt Enable Enables TPM overflow interrupts. 0 Disable TOF interrupts. Use software polling or DMA request. 1 Enable TOF interrupts. An interrupt is generated when TOF equals one.
5 CPWMS	Center-Aligned PWM Select Selects CPWM mode. This mode configures the TPM to operate in up-down counting mode. This field is write protected. It can be written only when the counter is disabled. 0 TPM counter operates in up counting mode. 1 TPM counter operates in up-down counting mode.
4–3 CMOD	Clock Mode Selection Selects the TPM counter clock modes. When disabling the counter, this field remain set until acknolwedged in the TPM clock domain. 00 TPM counter is disabled 01 TPM counter increments on every TPM counter clock 10 TPM counter increments on rising edge of TPM_EXTCLK synchronized to the TPM counter clock 11 Reserved.
PS	Prescale Factor Selection Selects one of 8 division factors for the clock mode selected by CMOD. This field is write protected. It can be written only when the counter is disabled. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Divide by 8 100 Divide by 16 101 Divide by 32 110 Divide by 64 111 Divide by 128

What The Macros Effectively Do (Lower 8 Bits)

- TPM_SC_CMOD_MASK
  - 0001_1000
- TPM_SC_CMOD_SHIFT
  - 3
- TPM_SC_CMOD(x)
  - (x << 3)
  - Bitwise AND with CMOD_MASK to assert 0 <= x <= 3

What We Have (Timer Started)

Bit Pos	7	6	5	4	3	2	1	0
Field	TOF	TOIE	CPWMS	CMOD		PS		
Val	0	1	0	0	1	0	1	1

What We Saw (Stop Timer)

- TPM0->SC |= TPM_SC_CMOD(0b0) // Or 0b00
- TPM0->SC &= TPM_SC_CMOD(0b0)

What This Does

- TPM0->SC |= TPM_SC_CMOD(0b0) // Or 0b00
  - Equivalent to (0 << 3) = 0

Bit Pos	7	6	5	4	3	2	1	0
Field	TOF	TOIE	CPWMS	CMOD		PS		
Val	0	1	0	0	1	0	1	1
|=	0	0	0	0	0	0	0	0
Result	0	1	0	0	1	0	1	1

- Timer not stopped!

What This Does

- TPM0->SC &= TPM_SC_CMOD(0b0)
  - Equivalent to (0 << 3) = 0

Bit Pos	7	6	5	4	3	2	1	0
Field	TOF	TOIE	CPWMS	CMOD		PS		
Val	0	1	0	0	1	0	1	1
&=	0	0	0	0	0	0	0	0
Result	0	0	0	0	0	0	0	0

- Timer stopped, but TOIE and PS also cleared!

What Should Have Been Done

- TPM0->SC &= ~TPM_SC_CMOD_MASK // Preferred
- TPM0->SC &= ~(TPM_SC_CMOD(3))
  - Mask equivalent to (0b11 << 3) = (0b0001_1000)

Bit Pos	7	6	5	4	3	2	1	0
Field	TOF	TOIE	CPWMS	CMOD		PS		
Val	0	1	0	0	1	0	1	1
MASK	0	0	0	1	1	0	0	0
&= ~MASK	1	1	1	0	0	1	1	1
Result	0	1	0	0	0	0	1	1