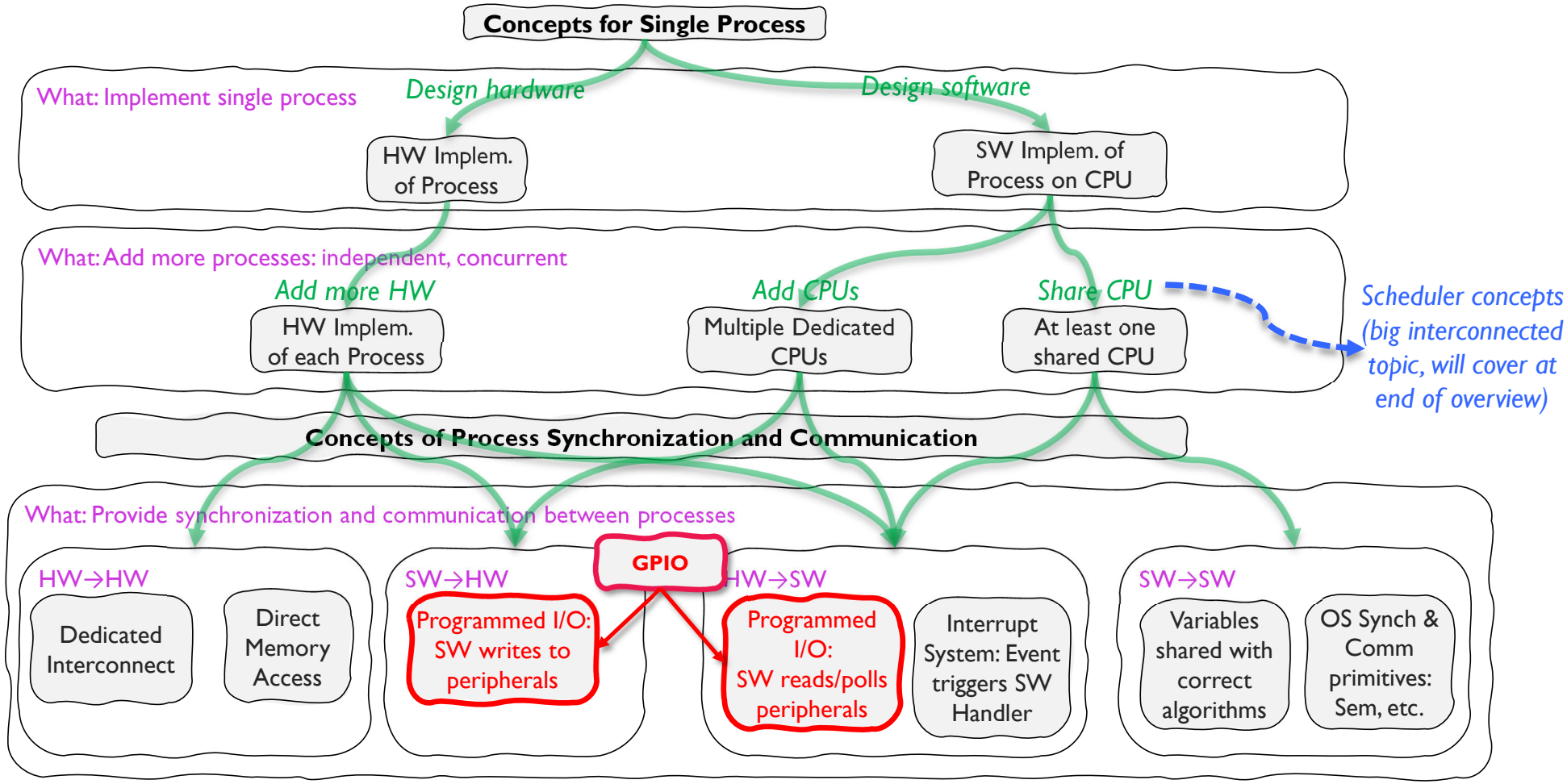
NC STATE UNIVERSITY

Port and GPIO Peripherals
CMSIS Device Peripheral HAL Support

9/16/2024

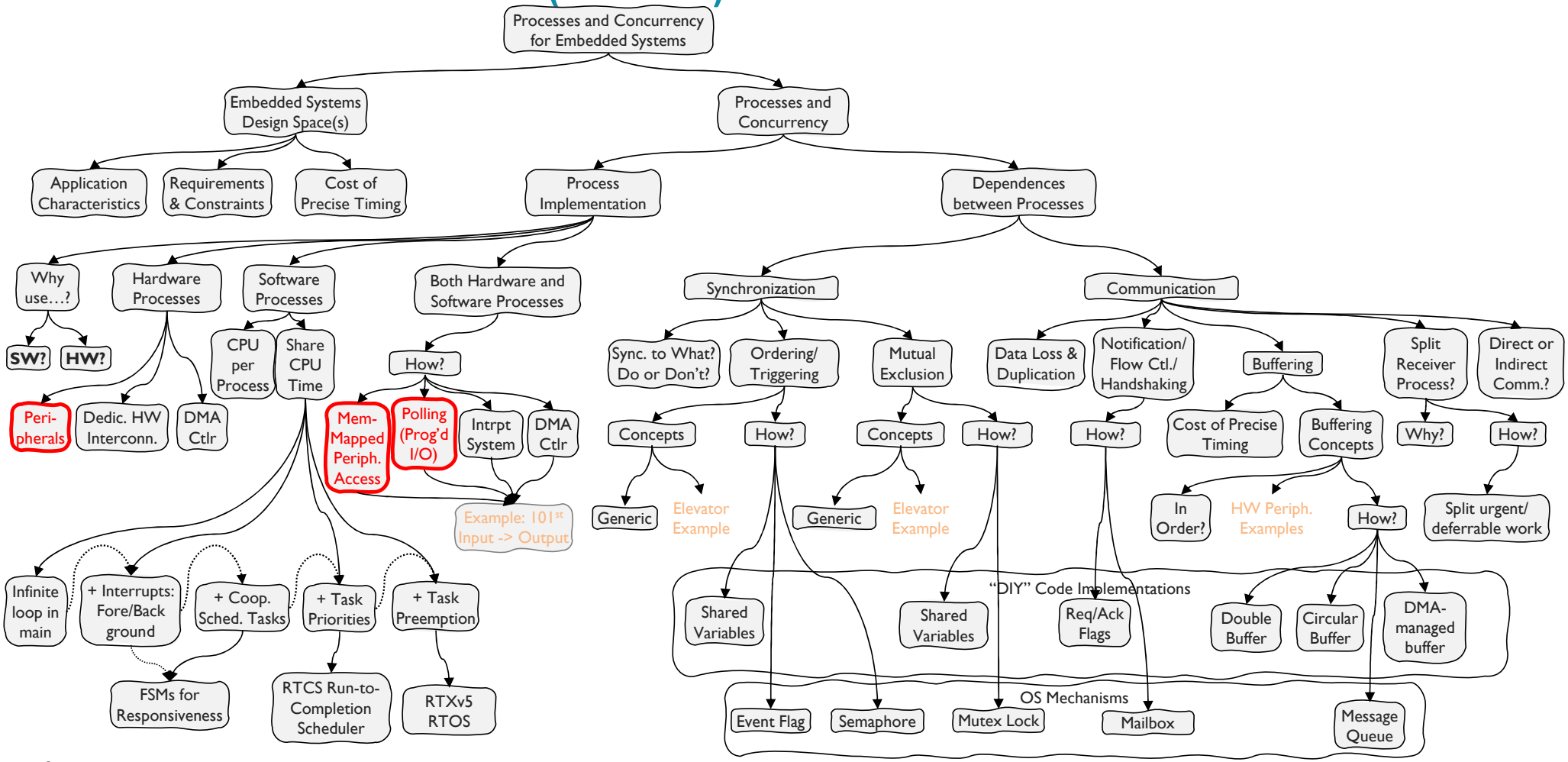
NC STATE UNIVERSITY

GPIO: Where Are We (view 1)?



NC STATE UNIVERSITY

GPIO: Where Are We (view 2)?



NC STATE UNIVERSITY

Overview

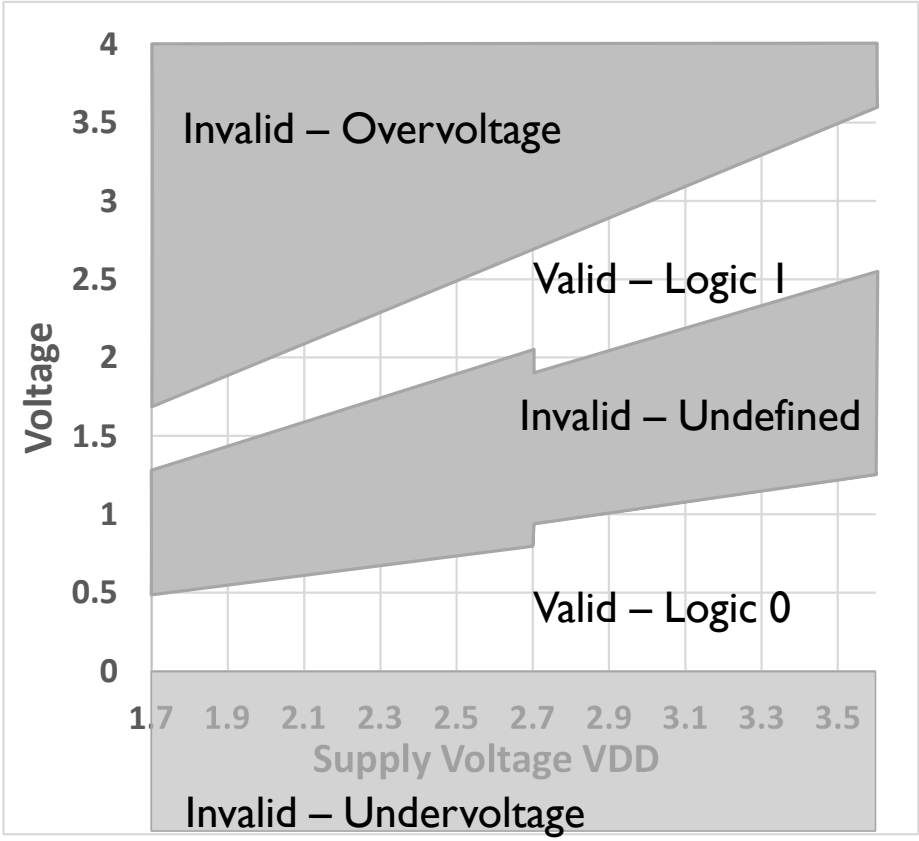
- Hardware
  - Input
  - Output
- Software to access control registers

NC STATE UNIVERSITY

DIGITAL INPUTS AND OUTPUTS

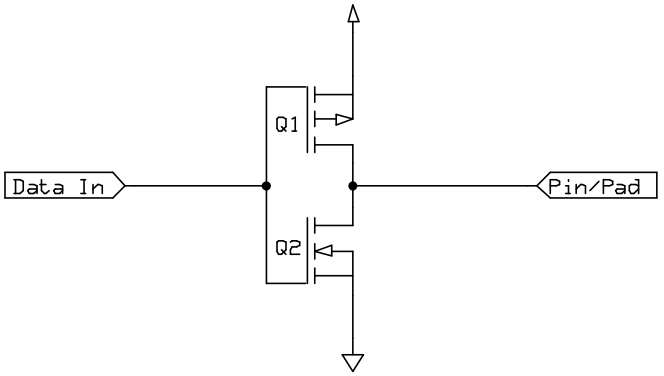
Inputs: What's a One? A Zero?

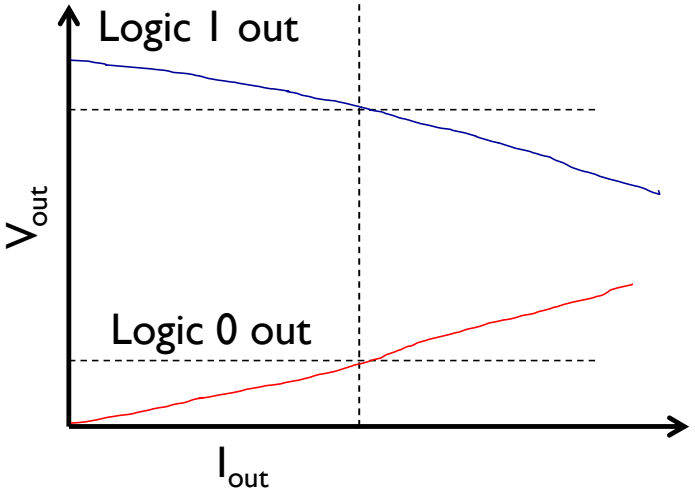
- Input signal's value is determined by voltage
- Input threshold voltages depend on supply voltage V_{DD}
- Exceeding V_{DD} or GND (0V) may damage chip



Outputs: What's a One? A Zero?

- Nominal output voltages
  - 1: V_{DD}-0.5 V to V_{DD}
  - 0: 0 to 0.5 V
- Note: Output voltage depends on current drawn by load on pin
  - Need to consider source-to-drain resistance in the transistor
  - Above values only specified when current < 5 mA (18 mA for high-drive pads) and V_{DD} > 2.7 V

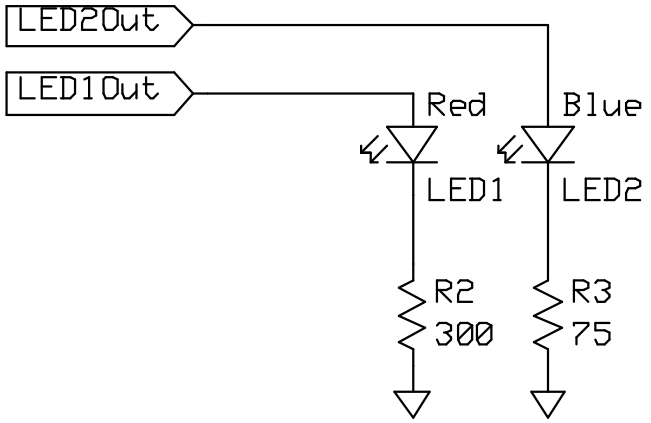




Details

Output Example: Driving LEDs

- Need to limit current to a value which is safe for both LED and MCU port driver
- Use current-limiting resistor
  - $R = (V_{DD} – V_{LED})/I_{LED}$
- Set I_{LED} = 4 mA
- V_{LED} depends on type of LED (mainly color)
  - Red: ~1.8V
  - Blue: ~2.7 V
- Solve for R given VDD = ~3.0 V
  - Red: 300 Ω
  - Blue: 75 Ω
- Demonstration code in Basic Light Switching Example



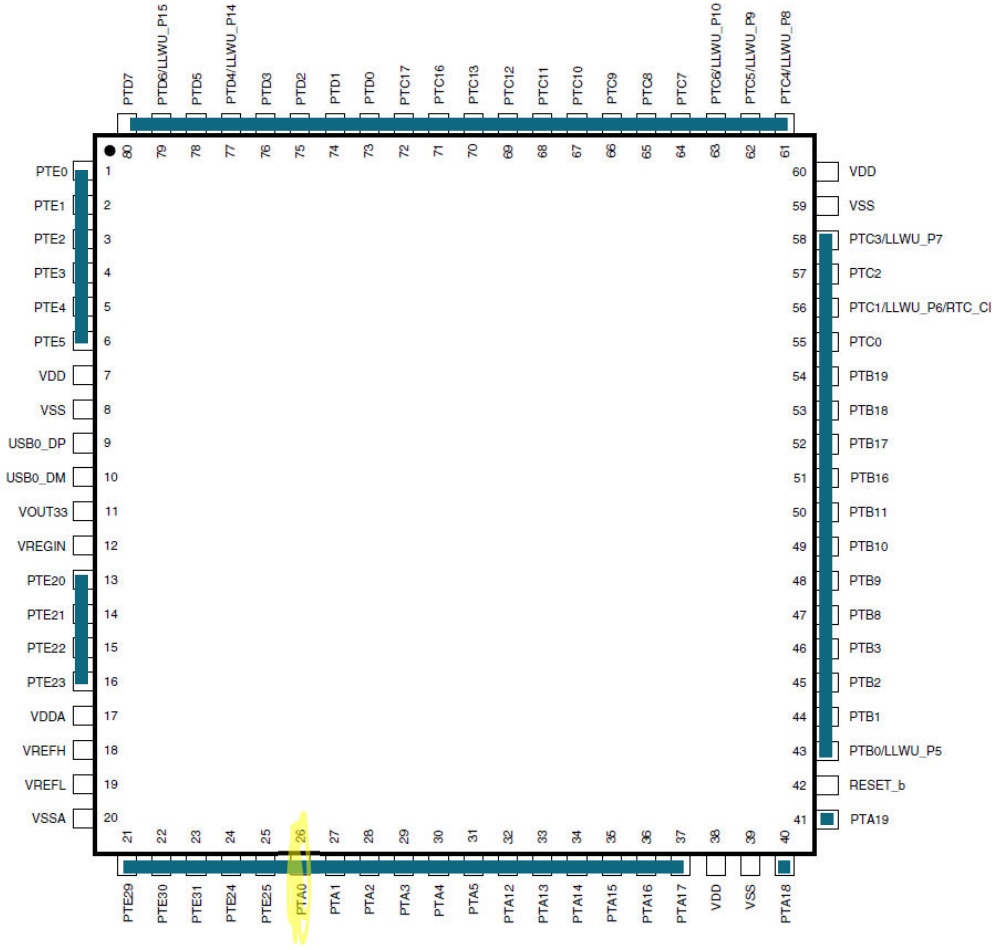
NC STATE UNIVERSITY

GPIO AND PORT HARDWARE BASICS

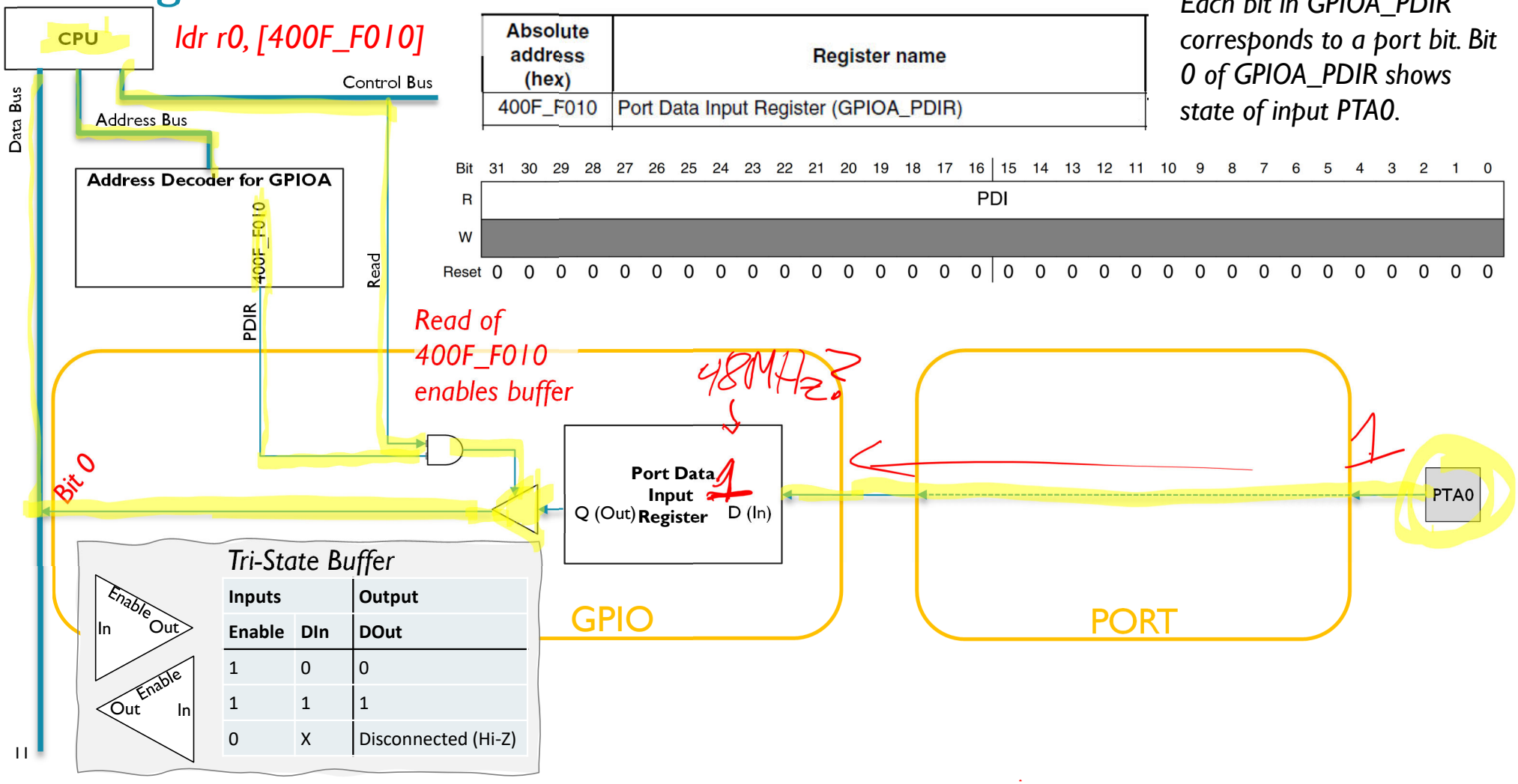
NC STATE UNIVERSITY

Our MCU's GPIO

- Port A (PTA) through Port E (PTE)
- Not all port bits are available
  - Five 32-bit ports would need 160 pins!
- Quantity depends on package's pin count
- KL25Z MCUs available in different packages
  - 32, 48, 64, 80 pins
  - Ours: MKL25Z128VLK4 uses 80 pin LQFP



Getting a Bit Into the CPU

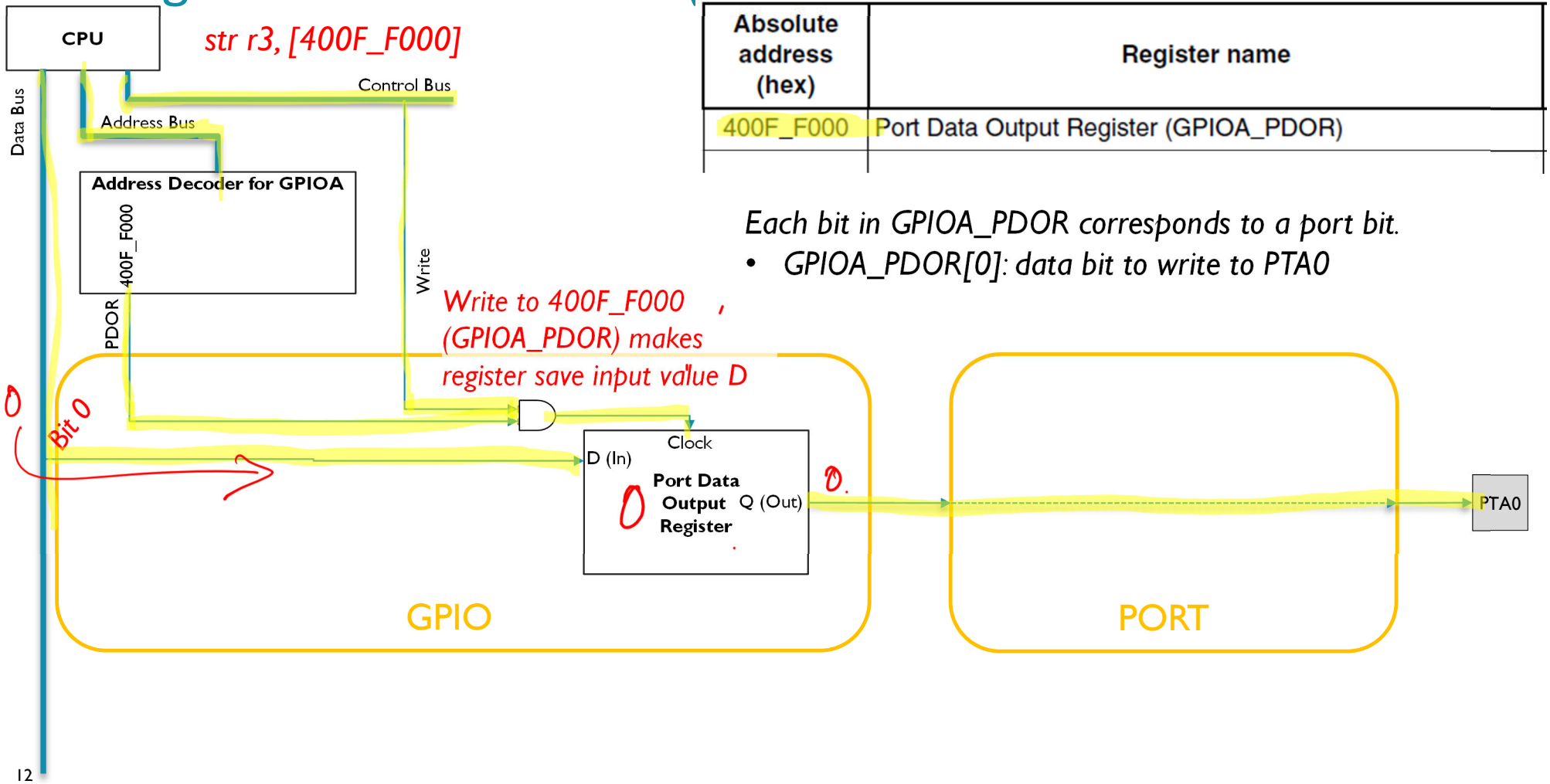


Absolute address (hex)	Register name
400F_F010	Port Data Input Register (GPIOA_PDIR)

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	PDI																															
W																																
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

Each bit in GPIOA_PDIR corresponds to a port bit. Bit 0 of GPIOA_PDIR shows state of input PTA0.

Getting a Bit Out of the CPU (Basic)

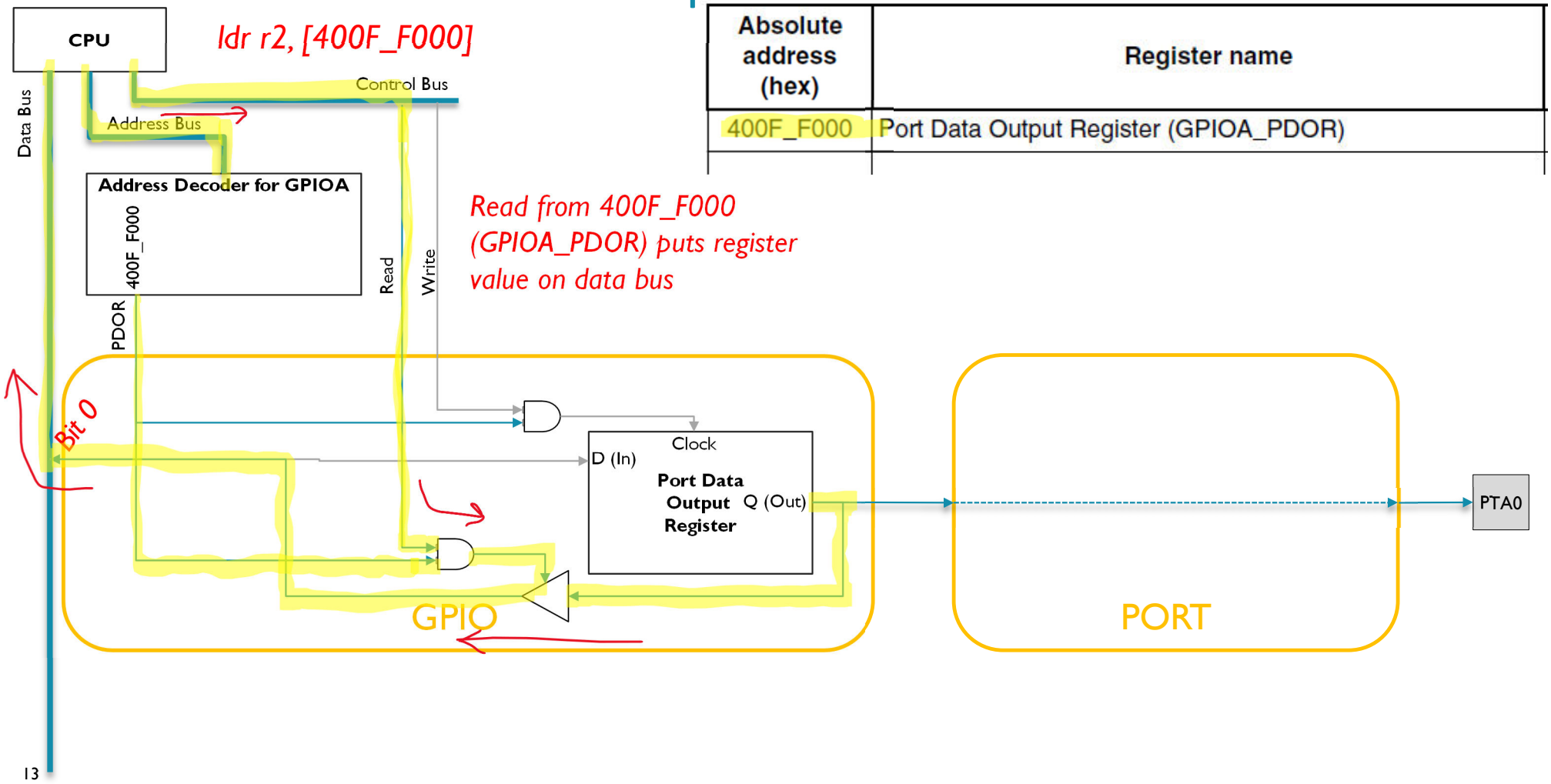


Absolute address (hex)	Register name
400F_F000	Port Data Output Register (GPIOA_PDOR)

Each bit in GPIOA_PDOR corresponds to a port bit.

- *GPIOA_PDOR[0]: data bit to write to PTA0*

Can Also *Read* Port Data Output Register



Absolute address (hex)	Register name
400F_F000	Port Data Output Register (GPIOA_PDOR)

Changing Some (but Not All) Output Bits with Software

- Issue: Writing to PDOR changes **all** 32 bits
  - What if we want to change bits 3&4 (not all bits)?
- Need to perform read/modify/write operation, taking at least three instructions
  - Change bits 3&4 in temp
    - How? Details later
  - Write temp to PDOR
- Drawbacks
  - Takes at least three software instructions with load/store ISA
  - Slows down program
  - Creates concurrency issue (***another*** critical section)

P1

ldr r0,[ ]

mods

str r0,[ ]

P2

ldr r3,[ ]

mod.

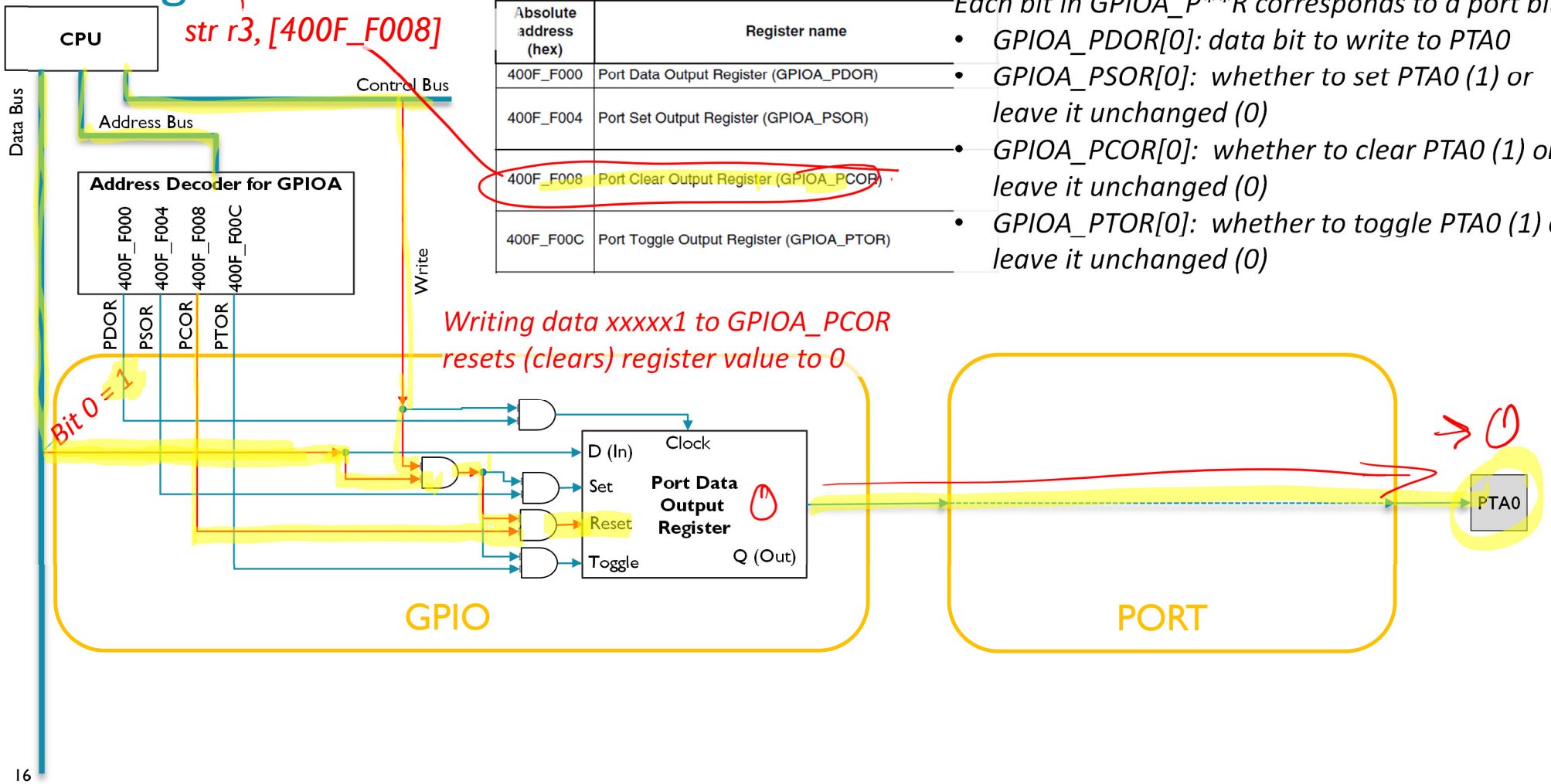
str r0,[ ]

Changing Some (but Not All) Output Bits with Hardware

- MCU's ports have extra hardware so software R/M/W is **not needed**
  - Specific to NXP Kinetis MCUs, but similar features available on other MCUs
- Three operations available
  - Set: raise specified bits to one
  - Clear: lower specified bits to zero
  - Toggle: invert specified bits
- Use **mask** to specify bits to modify (e.g. 3 and 4)
  - Bin: 00000000 00000000 00000000 00011000
  - Hexadecimal: 0x00000018
- Trigger operation by writing mask to port's output operation register
  - PSOR: Port **Set** Output Register
  - PCOR: Port **Clear** Output Register
  - PTOR: Port **Toggle** Output Register

Absolute address (hex)	Register name
400F_F000	Port Data Output Register (GPIOA_PDOR)
400F_F004	Port Set Output Register (GPIOA_PSOR)
400F_F008	Port Clear Output Register (GPIOA_PCOR)
400F_F00C	Port Toggle Output Register (GPIOA_PTOR)

Getting a Bit Out of the CPU (Hardware)



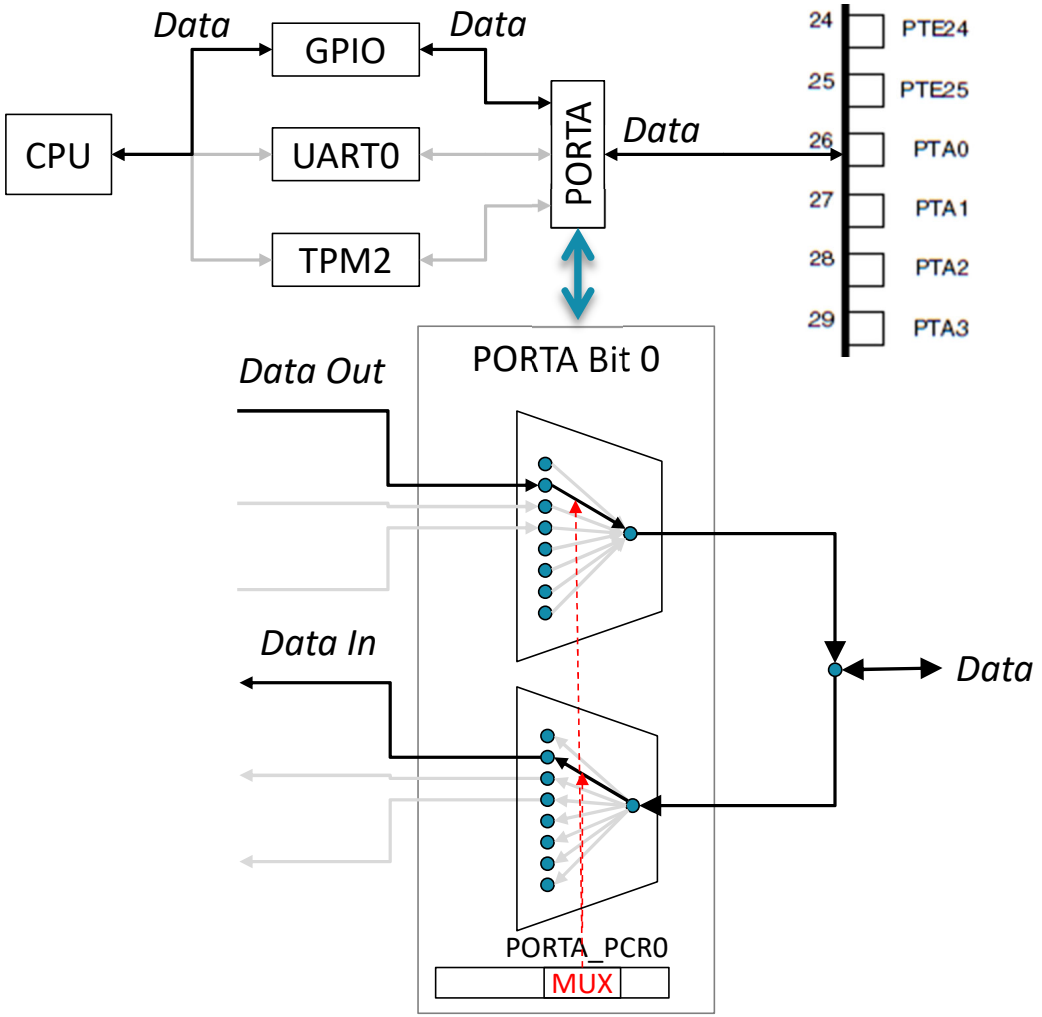
Absolute address (hex)	Register name
400F_F000	Port Data Output Register (GPIOA_PDOR)
400F_F004	Port Set Output Register (GPIOA_PSOR)
400F_F008	Port Clear Output Register (GPIOA_PCOR)
400F_F00C	Port Toggle Output Register (GPIOA_PTOR)

*Each bit in GPIOA_P**R corresponds to a port bit.*

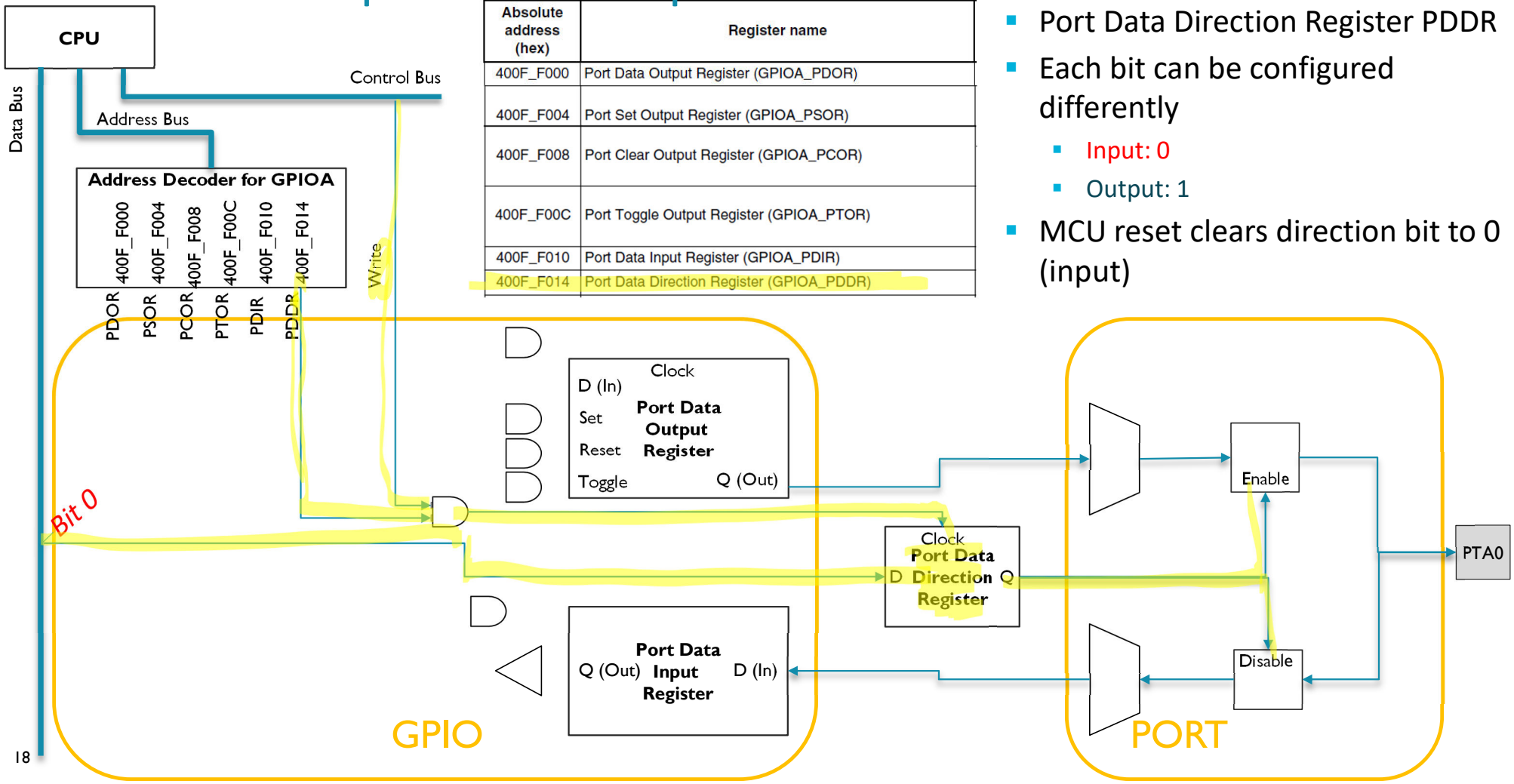
- *GPIOA_PDOR[0]: data bit to write to PTA0*
- *GPIOA_PSOR[0]: whether to set PTA0 (1) or leave it unchanged (0)*
- *GPIOA_PCOR[0]: whether to clear PTA0 (1) or leave it unchanged (0)*
- *GPIOA_PTOR[0]: whether to toggle PTA0 (1) or leave it unchanged (0)*

NC STATE UNIVERSITY

How to Share MCU Pins – Pin Multiplexing



Is Port Bit an Input or an Output?

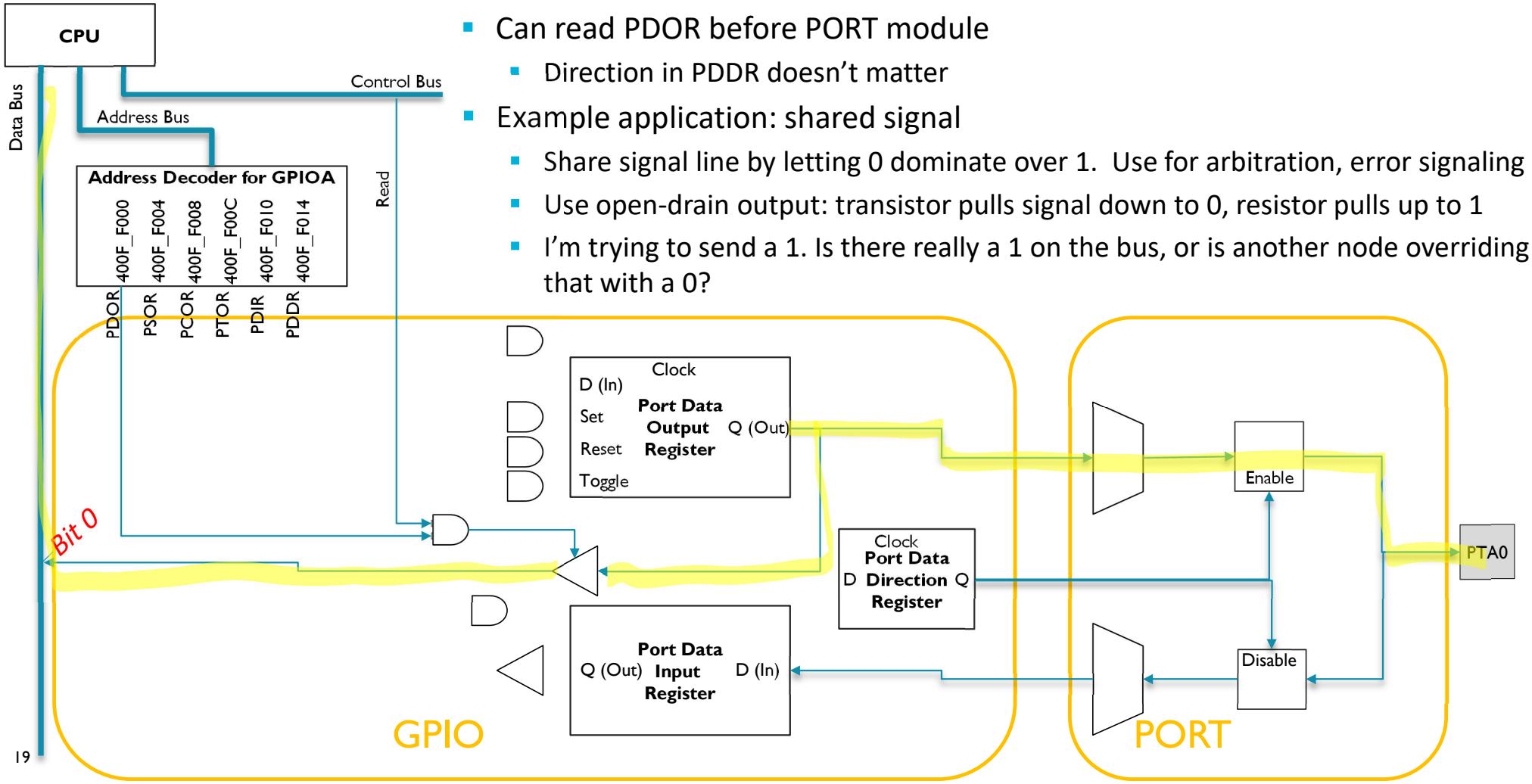


Absolute address (hex)	Register name
400F_F000	Port Data Output Register (GPIOA_PDOR)
400F_F004	Port Set Output Register (GPIOA_PSOR)
400F_F008	Port Clear Output Register (GPIOA_PCOR)
400F_F00C	Port Toggle Output Register (GPIOA_PTOR)
400F_F010	Port Data Input Register (GPIOA_PDIR)
400F_F014	Port Data Direction Register (GPIOA_PDDR)

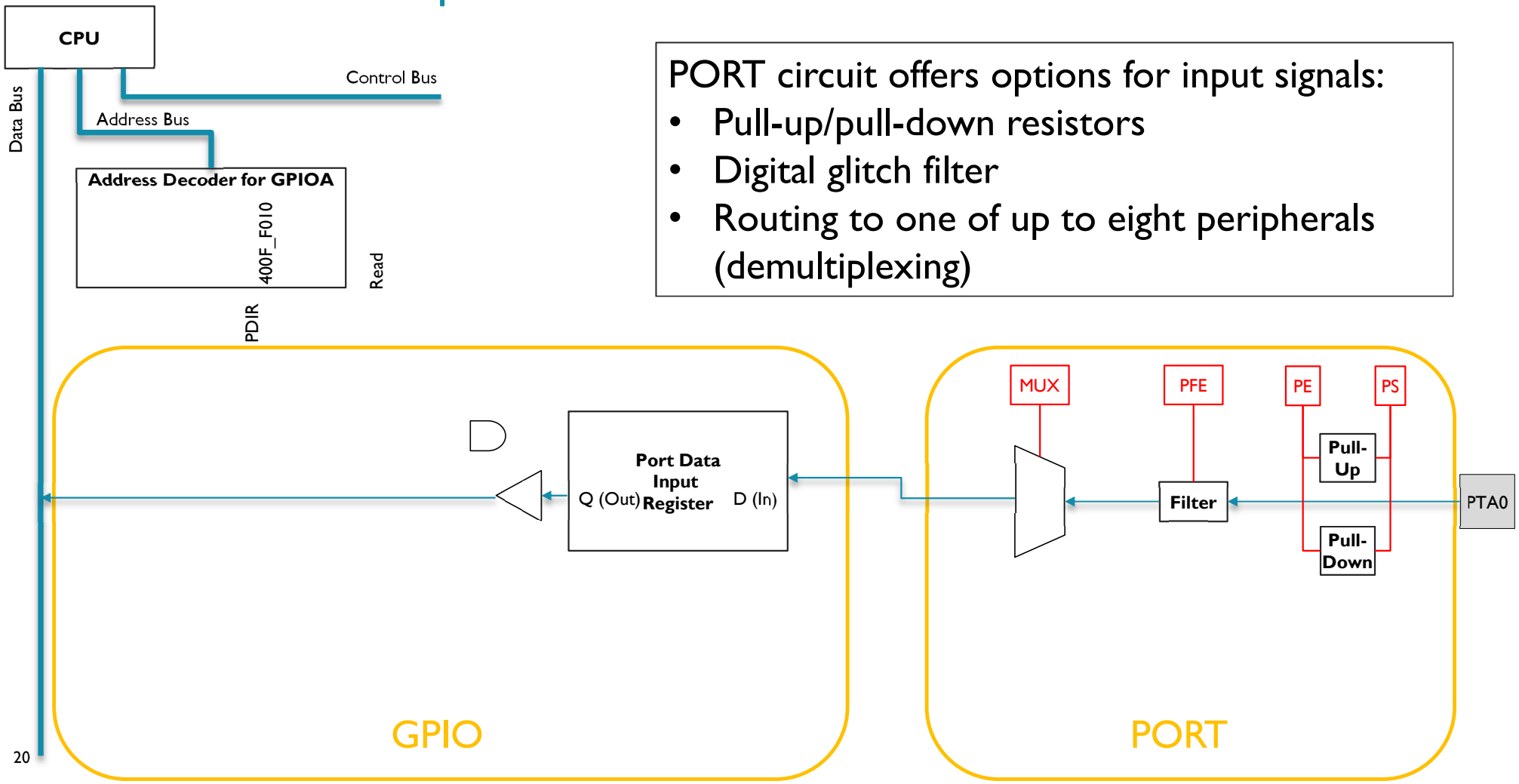
- Port Data Direction Register PDDR
- Each bit can be configured differently
  - Input: 0
  - Output: 1
- MCU reset clears direction bit to 0 (input)

Other Reasons to Read PDOR Before Port

- Can read PDOR before PORT module
  - Direction in PDDR doesn't matter
- Example application: shared signal
  - Share signal line by letting 0 dominate over 1. Use for arbitration, error signaling
  - Use open-drain output: transistor pulls signal down to 0, resistor pulls up to 1
  - I'm trying to send a 1. Is there really a 1 on the bus, or is another node overriding that with a 0?



Port Module for Inputs



PORT circuit offers options for input signals:

- Pull-up/pull-down resistors
- Digital glitch filter
- Routing to one of up to eight peripherals (demultiplexing)

NC STATE UNIVERSITY

KL25Z Pull-Ups and Pull-Downs

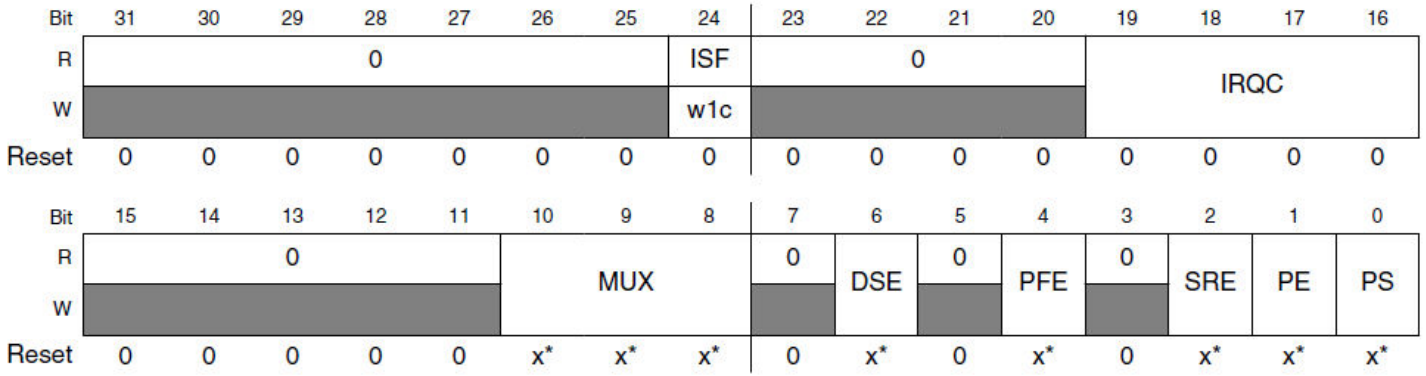
Port	Bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A	↕	↑	↑	↑	↑	↑							↑	↑	↑	↑	↑	↑	↑	↑	↑											
B	↑	↑	↑	↑					↑	↑	↑	↑					↑	↑	↑	↑												
C	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑			↑	↑														
D	↑	↑	↑	↑	↑	↑	↑	↑																								
E	↑	↑	↑	↑	↑	↑											↑	↑	↑	↑	↑	↑	↑	↑	↑	↑				↑	↑	↑

- Available GPIO and pulls for 80-pin package
- ↑ pull-up available
- ↕ pull-up and pull-down available

Details

NC STATE UNIVERSITY

Additional Configuration in PCR

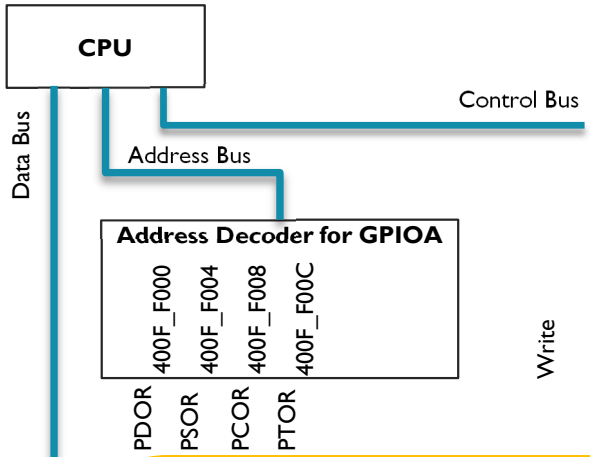


Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

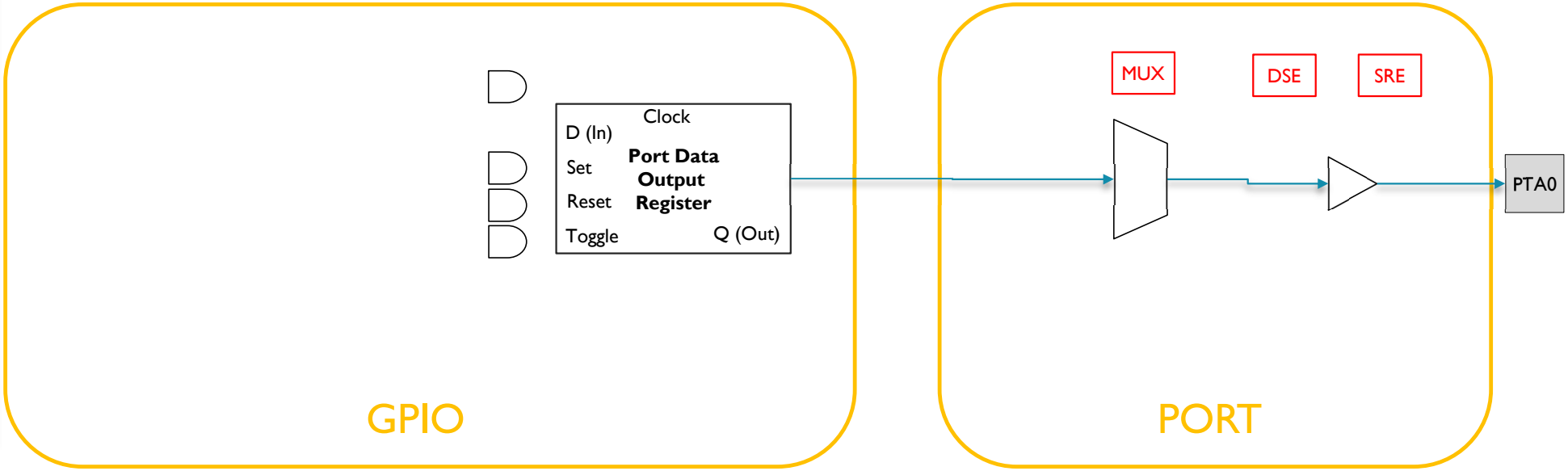
- **Available on some pins, is MCU-dependent**
- Optional pull-up and pull-down resistors
  - Used to ensure input signal voltage is pulled to correct value when floating, disconnected or otherwise high-impedance
  - PE: Pull Enable. 1 enables the pull resistor
  - PS: Pull Select. 1 pulls up, 0 pulls down.
- Optional high current drive strength
  - DSE: Set to 1 to drive more current (e.g. 18 mA vs. 5 mA @ > 2.7 V, or 6 mA vs. 1.5 mA @ <2.7 V)

Port Module for Outputs

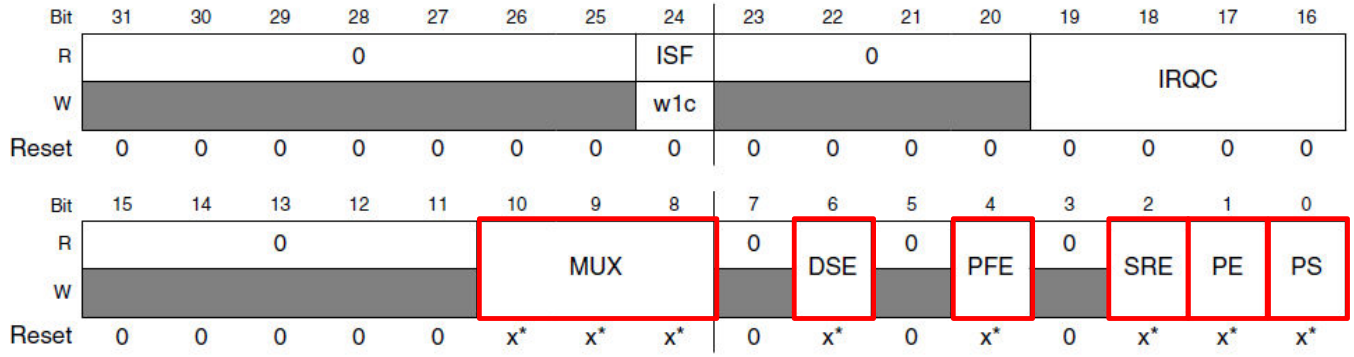


PORT circuit offers options for output signals:

- Routing from one of multiple peripherals (multiplexing)
- Adjustable drive strength (output impedance)
- Slew-rate limiting for transitions



PORT Pin Control Registers

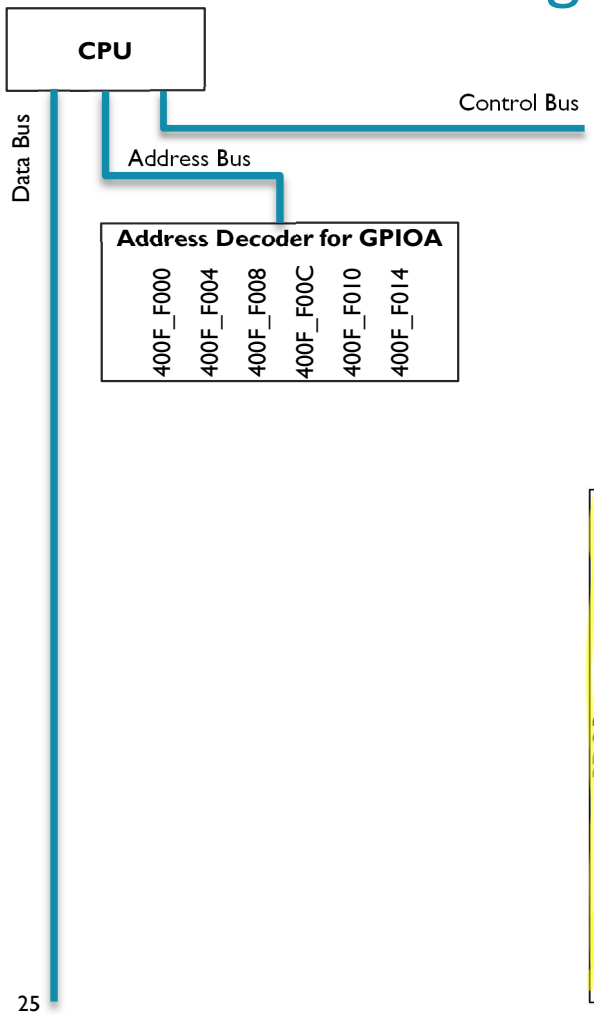


80 LQFP	64 LQFP	48 QFN	32 QFN	Pin Name	Default	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7
64	52	40	28	PTC7	CMP0_IN1	CMP0_IN1	PTC7	SPI0_MISO			SPI0_MOSI		
65	53	–	–	PTC8	CMP0_IN2	CMP0_IN2	PTC8	I2C0_SCL	TPM0_CH4				

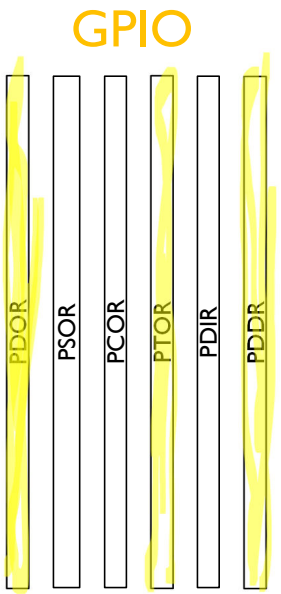
- One PCR per PORT bit
- MUX field of PCR defines connections

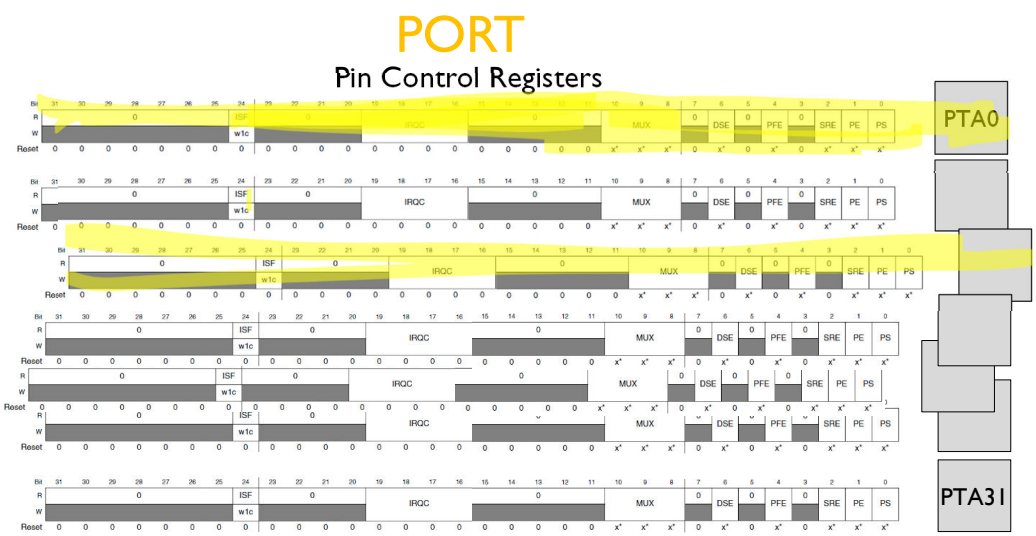
MUX (bits 10-8)	Configuration
000	Pin disabled (analog)
001	Alternative 1 – GPIO
010	Alternative 2
011	Alternative 3
100	Alternative 4
101	Alternative 5
110	Alternative 6
111	Alternative 7

GPIO vs. PORT Register Layouts

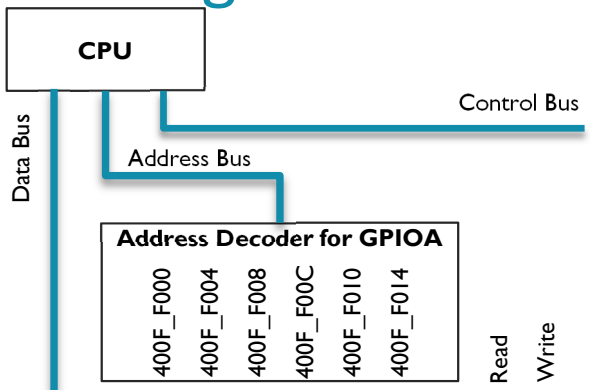


GPIO Register: Accesses **single** attribute of **all** bits
PORT Register: Accesses **all** attributes of **single** bit



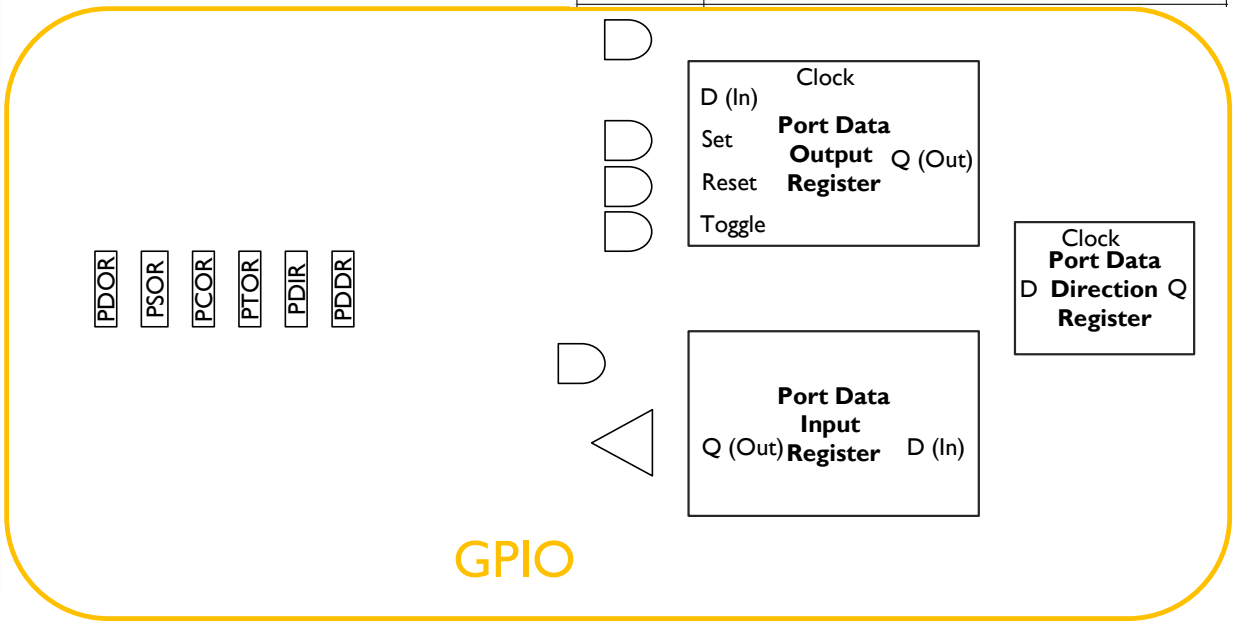


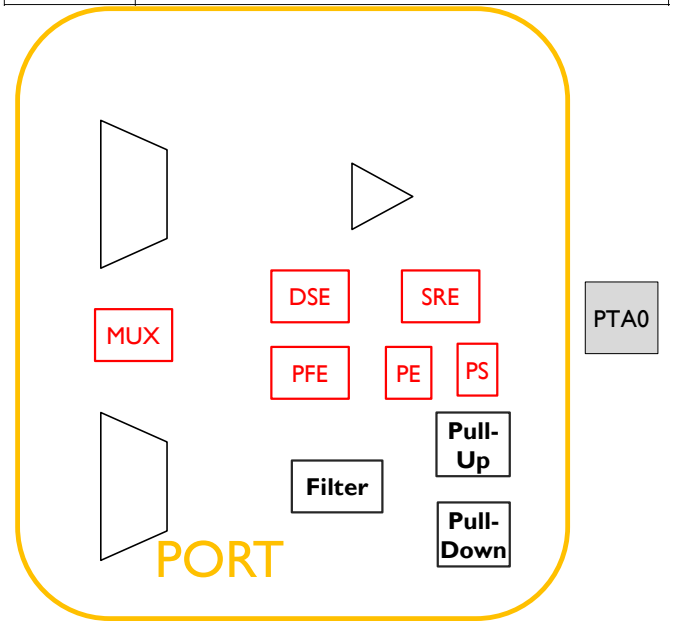
Putting GPIO and PORT Together



Absolute address (hex)	Register name
400F_F000	Port Data Output Register (GPIOA_PDOR)
400F_F004	Port Set Output Register (GPIOA_PSOR)
400F_F008	Port Clear Output Register (GPIOA_PCOR)
400F_F00C	Port Toggle Output Register (GPIOA_PTOR)
400F_F010	Port Data Input Register (GPIOA_PDIR)
400F_F014	Port Data Direction Register (GPIOA_PDDR)

Absolute address (hex)	Register name
4004_9000	Pin Control Register n (PORTA_PCR0)
4004_9004	Pin Control Register n (PORTA_PCR1)
4004_9008	Pin Control Register n (PORTA_PCR2)
4004_900C	Pin Control Register n (PORTA_PCR3)
4004_9010	Pin Control Register n (PORTA_PCR4)
4004_9014	Pin Control Register n (PORTA_PCR5)
4004_9074	Pin Control Register n (PORTA_PCR29)
4004_9078	Pin Control Register n (PORTA_PCR30)
4004_907C	Pin Control Register n (PORTA_PCR31)

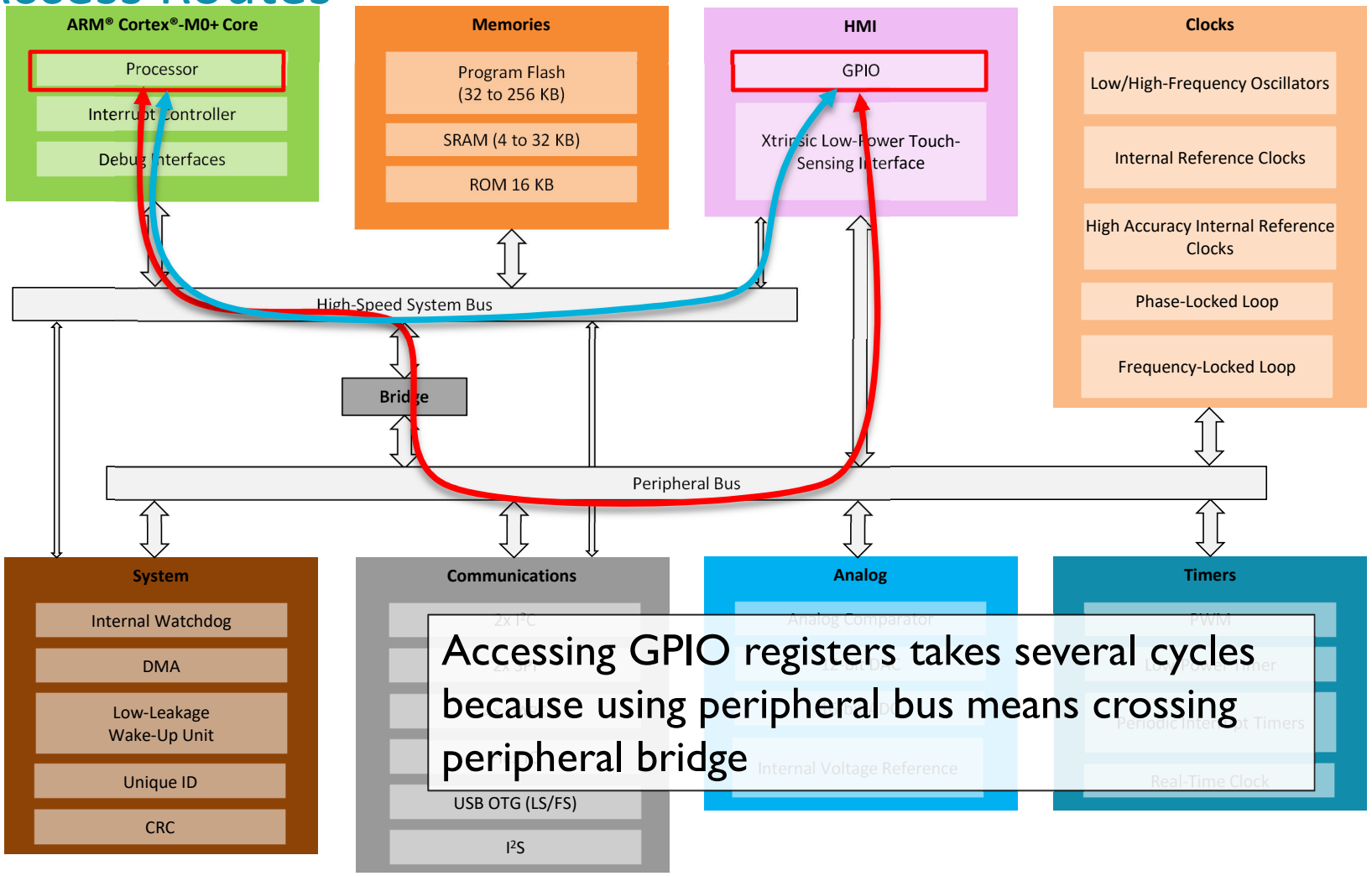




NC STATE UNIVERSITY

FAST GPIO HARDWARE

GPIO Access Routes



Accessing GPIO registers takes several cycles because using peripheral bus means crossing peripheral bridge

NC STATE UNIVERSITY

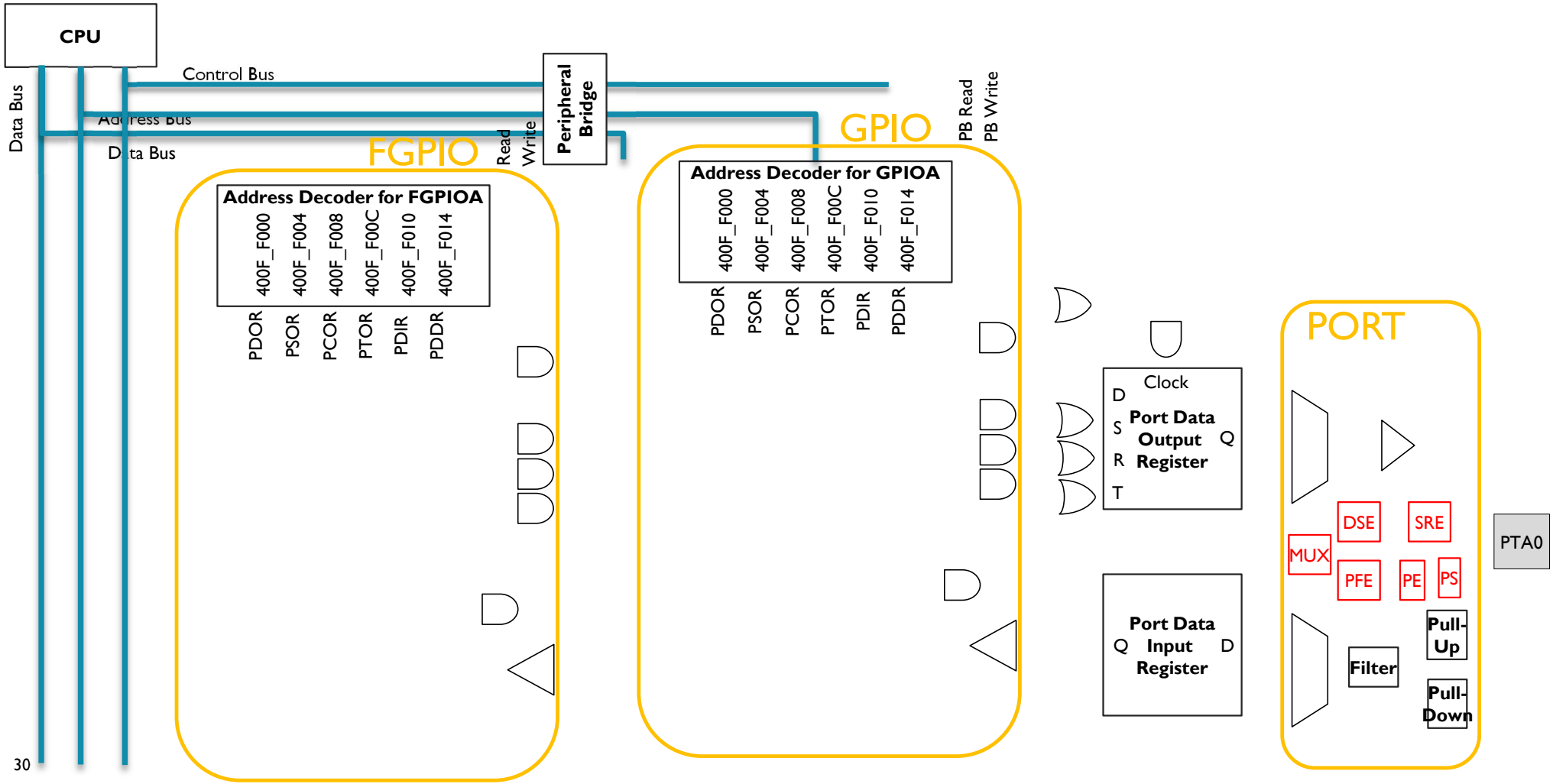
IOPORT Module (Fast GPIO, FGPIO)

Absolute address (hex)	Register name
F80F_F000	Port Data Output Register (FGPIOA_PDOR)
F80F_F004	Port Set Output Register (FGPIOA_PSOR)
F80F_F100	Port Data Output Register (FGPIOE_PDOR)
F80F_F104	Port Set Output Register (FGPIOE_PSOR)
F80F_F108	Port Clear Output Register (FGPIOE_PCOR)
F80F_F10C	Port Toggle Output Register (FGPIOE_PTOR)
F80F_F110	Port Data Input Register (FGPIOE_PDIR)
F80F_F114	Port Data Direction Register (FGPIOE_PDDR)

- MCU provides IOPORT to access GPIO registers directly over High-Speed System Bus
- CPU core can do *single-cycle* access to GPIO register through the IOPORT addresses
- So, two ways to access GPIO registers
  - Via GPIO
  - Via FPGIO

Details

Fast GPIO

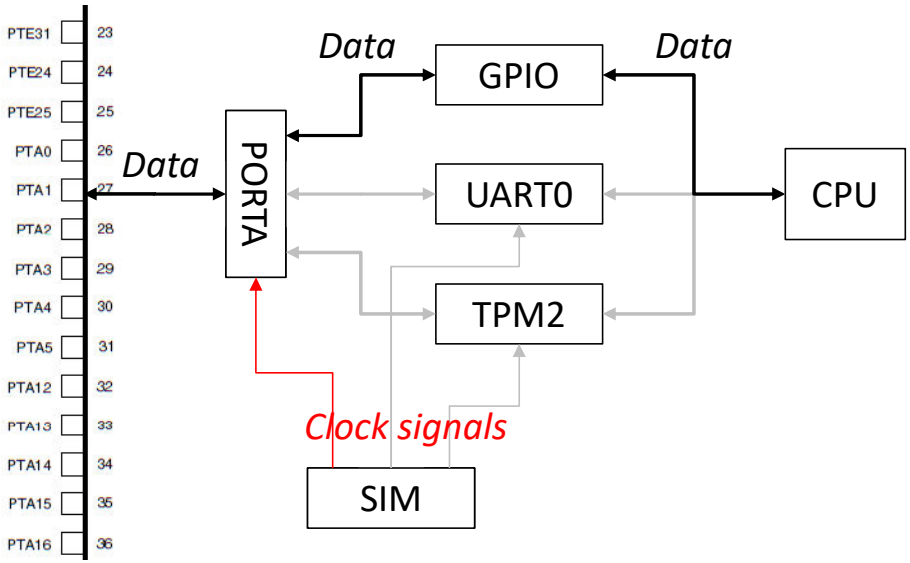


NC STATE UNIVERSITY

PIN USE CHECKLIST

NC STATE UNIVERSITY

Pin Use Checklist



- SIM: Enable logic clock signals to…
  - Pin's Port module
  - Peripheral module (may or may not be Port)
- Configure pin control multiplexer
- Configure peripheral
- Configure interrupt system (if needed)

NC STATE UNIVERSITY

Enabling Clock for Ports (& other Peripherals)

Bit	Port
13	PORTE
12	PORTD
11	PORTC
10	PORTB
9	PORTA

- Need to enable clock to port module
- By default, port modules are disabled to save power
- Writing to an unclocked module triggers a hardware fault!
- Control register SIM_SCGC5 gates (enables) clocks to the ports

- Enable clock to Port A

```
SIM->SCGC5 |= (1UL << 9);
```

- Header file MKL25Z4.h has definitions

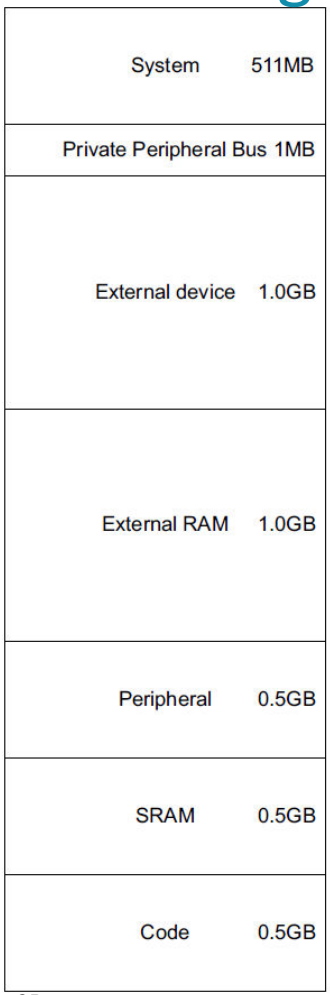
```
SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
```

- *Note: Syntax explained later in these slides*

NC STATE UNIVERSITY

SOFTWARE FOR EASILY ACCESSING PERIPHERAL REGISTERS

Accessing Peripheral Control Registers in Memory



0xFFFFFFFF
0xE0100000
0xE00FFFFF
0xE0000000
0xDFFFFFFF
0xA0000000
0x9FFFFFFF
0x60000000
0x5FFFFFFF
0x40000000
0x3FFFFFFF
0x20000000
0x1FFFFFFF
0x00000000

Address	Peripheral
4004_9000 …	PORTA
4004_A000 …	PORTB
4004_B000 …	PORTC
4004_C000 …	PORTD
4004_D000 …	PORTE
400F_F000 …	GPIOA
400F_F040 …	GPIOB
400F_F080 …	GPIOC
400F_F0C0 …	GPIOD
400F_F100 …	GPIOE

Address	Register
4004_9000	PORTA_PCR0
	PORTA_PCR1
…	…
4004_9078	PORTA_PCR30
4004_907C	PORTA_PCR31
4004_9080	PORTA_GPCLR
4004_9084	PORTA_GPCHR
…	(reserved)
4004_90A0	PORTA_ISFR

Address	Register Name
400F_F000	GPIOA_PDOR
400F_F004	GPIOA_PSOR
400F_F008	GPIOA_PCOR
400F_F00C	GPIOA_PTOR
400F_F010	GPIOA_PDIR
400F_F014	GPIOA_PDDR

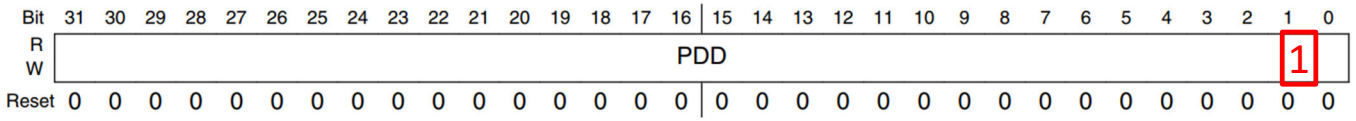
- Control registers…
  - Located at specific addresses
  - May be grouped together
  - One register may contain multiple fields
- Let's look at software methods to organize and access *groups of control registers at specific addresses*
  - C language data types: structs, bit fields and arrays
  - CMSIS-CORE Device Peripheral HAL

NC STATE UNIVERSITY

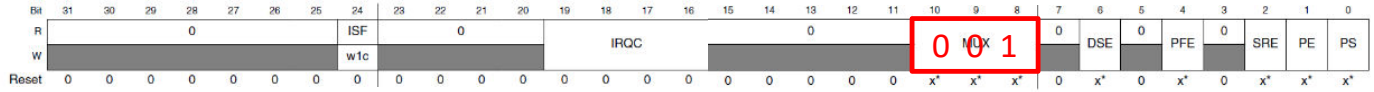
Example: Configure Pin PTA1 as GPIO Output

- What needs to happen?

  - Configure GPIO: GPIOA_PDDR (at 0x400F_F014)
    - Set direction as output by changing bit 1 to 1

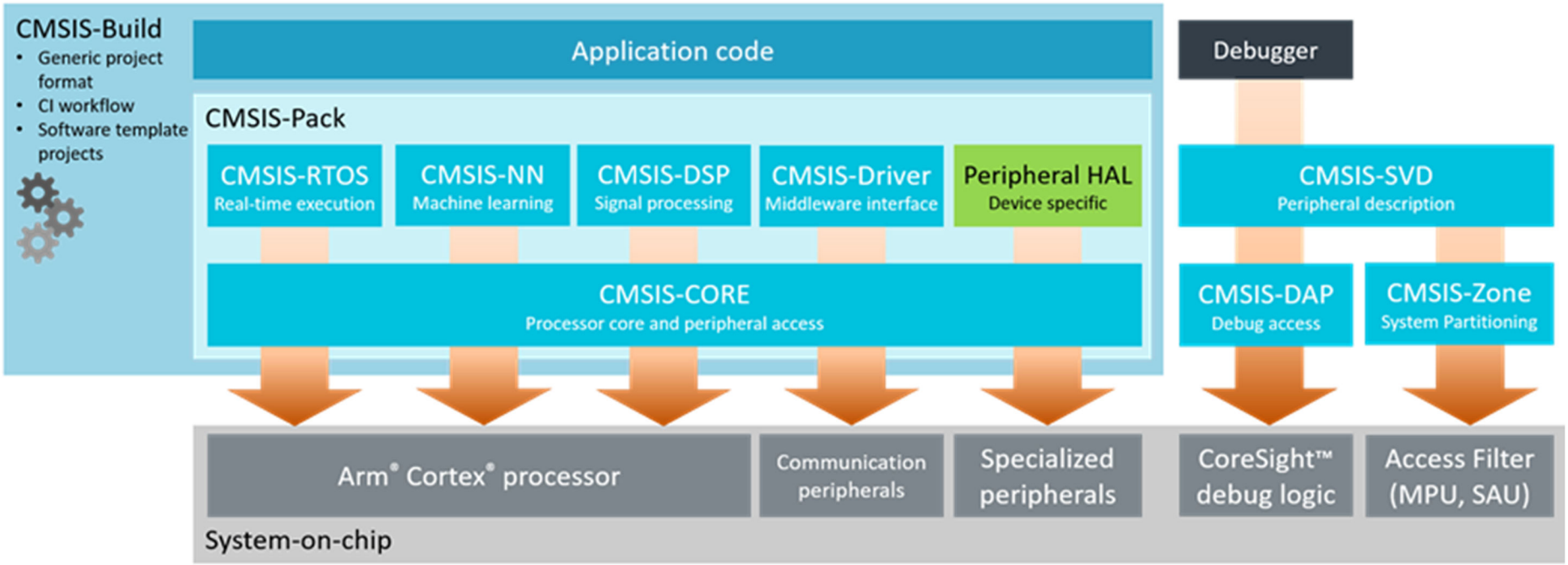


  - Configure PORT: PORTA_PCR1 (at 0x4004_9004)
    - Select GPIO by changing MUX field (bits 10-8) to 001



- How do we do this ***cleanly*** with software?
  - Big Picture: CMSIS-CORE Device Peripheral HAL support
  - C Language Mechanisms
    - A: Grouping **words** with data structures
    - B: Accessing **bits** within a word
      - B1: Bit fields
      - B2: Bitwise logical masking operations

CMSIS: Common Microcontroller Software Interface Standard



- "a set of tools, APIs, frameworks, and work flows …"
- "… that help to simplify software re-use, reduce the learning curve for microcontroller developers, speed-up project build and debug, and thus reduce the time to market for new applications"

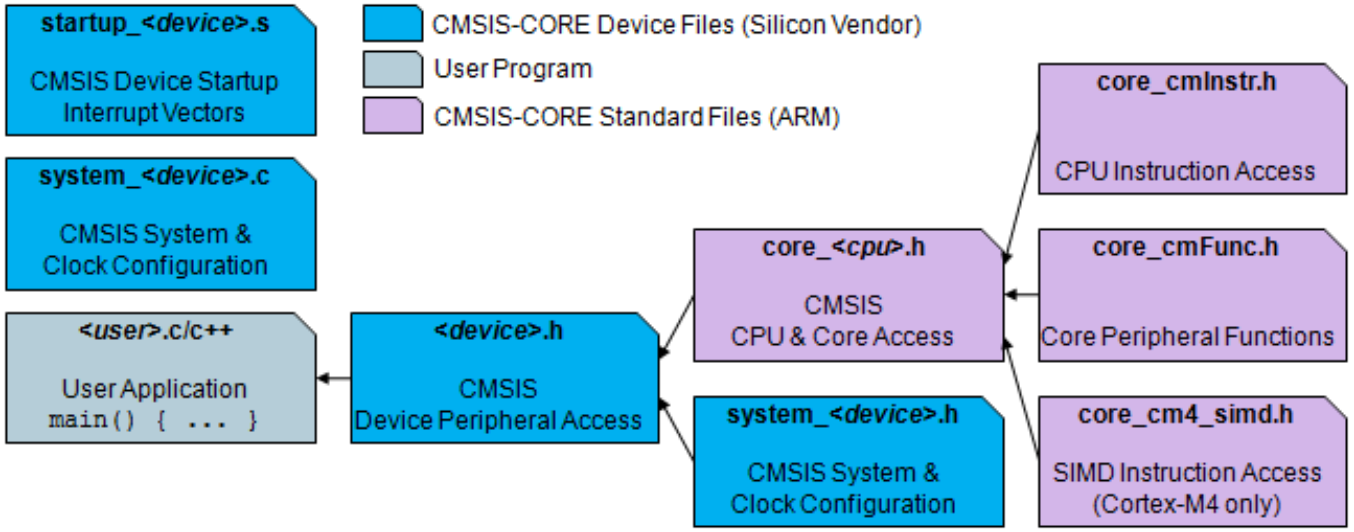
Details

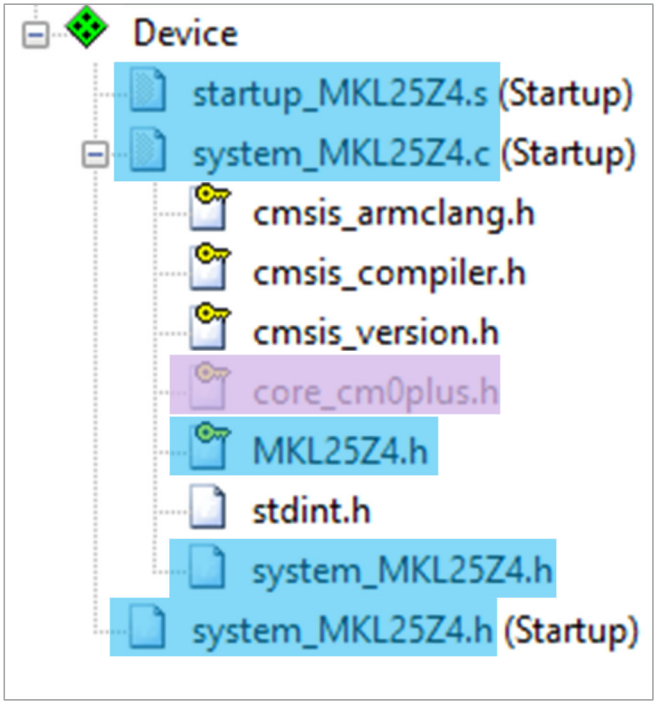
NC STATE UNIVERSITY

CMSIS: Cortex Microcontroller Software Interface Specification

- So much complexity!
  - Many vendors create Cortex-M-based MCUs
    - Peripherals are structured and accessed differently
  - More instructions available in more powerful CPUs
  - Many RTOSs available
  - Different approaches to implementing features not defined in C/C++ specifications
- What is CMSIS?
  - **Conventions** and **standards** for software interfaces, structure and names
    - Interfaces to peripherals, RTOS, debugger
  - **Libraries** of code optimized for specific processor cores
    - DSP, Neural Networks, etc.
- CMSIS Benefits
  - Shortens learning curve on new hardware
  - Simplifies software development
  - Simplifies software reuse
- References
  - Detailed documentation available through MDK: Help->Open Books Window, then select Tool User's Guide-> CMSIS Documentation
  - https://arm-software.github.io/CMSIS_5/General/html/index.html

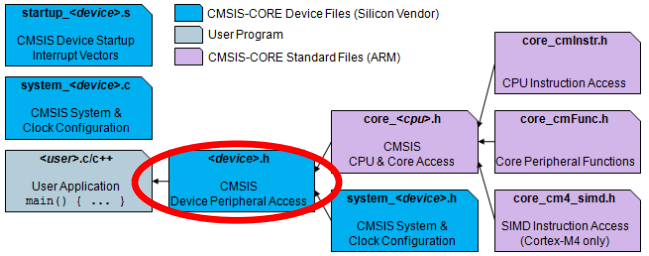
CMSIS Files to Support MCU (Device)





- Arm, MCU device vendor (e.g. NXP) provide support files
- Header files
  - *MKL25Z4.h:* CMSIS-defined access to MCU's peripherals
  - core_cm0plus.h: CMSIS-defined access to CPU core and its peripherals (system control register, SysTick timer, etc.)
- Code files
  - Start-up, interrupt vectors, configuration

(A) Grouping Words: Use Struct to Match Register Layout



- Header file MKL25Z4.h
  - Defines C data structure types to represent MCU hardware registers
  - Tells compiler a field's offset (in bytes) from the beginning of the struct and field's size (from data type)

```
/** GPIO - Register Layout Typedef */
typedef struct {
   __IO uint32_t PDOR; /**< offset: 0x0 */
   __O  uint32_t PSOR; /**< offset: 0x4 */
   __O  uint32_t PCOR; /**< offset: 0x8 */
   __O  uint32_t PTOR; /**< offset: 0xC */
   __I  uint32_t PDIR; /**< offset: 0x10 */
   __IO uint32_t PDDR; /**< offset: 0x14 */
} GPIO_Type;
```

Address	Register Name
?	GPIOA_PDOR
? + 0x04	GPIOA_PSOR
? + 0x08	GPIOA_PCOR
? + 0x0C	GPIOA_PTOR
? + 0x10	GPIOA_PDIR
? + 0x14	GPIOA_PDDR

Locating GPIOA Control Registers in Memory

- Header file MKL25Z4.h provides support including:
  - Pointers to the bases of registers as GPIO_Type structs
  - Register instance definitions matching register names
- Warning:
  - Older versions of header file (and the textbook **ESF**) use PTA to refer to GPIOA peripheral, not PORTA peripheral!
  - Newer versions use GPIOA, but also #define PTA as GPIOA for compatibility

Address	Register Name
400F_F000	GPIOA_PDOR
400F_F004	GPIOA_PSOR
400F_F008	GPIOA_PCOR
400F_F00C	GPIOA_PTOR
400F_F010	GPIOA_PDIR
400F_F014	GPIOA_PDDR

```
/** Peripheral GPIOA base address */
#define GPIOA_BASE   (0x400FF000u)
/** Peripheral GPIOA base pointer */
#define GPIOA ((GPIO_Type *)GPIOA_BASE)
/** GPIOA - Register accessors */
#define GPIO_PDOR_REG(base) ((base)->PDOR)
/* GPIO - Register instance definitions */
#define GPIOA_PDOR  GPIO_PDOR_REG(GPIOA)
```

```
// Option 1: use pointer & data structure
GPIOA->PDOR = my_data;
// GPIOA_BASE + Offset of 0 = 0x400FF000
GPIOA->PTOR = 0x00000018;
// GPIOA_BASE + Offset of 0xC = 0x400FF00C

// Option 2: use register instance defs
GPIOA_PDOR = my_data;
GPIOA_PTOR = 0x00000018;
```

Details

FGPIO: Single-Cycle GPIO Access

NC STATE UNIVERSITY

```
typedef struct {
    __IO uint32_t PDOR; /**< offset: 0x0 */
    __O uint32_t PSOR; /**< offset: 0x4 */
    __O uint32_t PCOR; /**< offset: 0x8 */
    __O uint32_t PTOR; /**< offset: 0xC */
    __I uint32_t PDIR; /**< offset: 0x10 */
    __IO uint32_t PDDR; /**< offset: 0x14 */
} FGPIO_Type;

#define FGPIOA_BASE (0xF80FF000u)
/** Peripheral FGPIOA base pointer */
#define FGPIOA ((FGPIO_Type *)FGPIOA_BASE)

FGPIOA->PDOR = MASK(LED2_POS);
```

Address	Register Name
F80F_F000	FGPIOA_PDOR
F80F_F004	FGPIOA_PSOR
F80F_F008	FGPIOA_PCOR
F80F_F00C	FGPIOA_PTOR
F80F_F010	FGPIOA_PDIR
F80F_F014	FGPIOA_PDDR

CMSIS C Support for PORT

- MKL25Z4.h defines PORT_Type structure with a PCR field (array of 32 integers)

```
/** PORT - Register Layout Typedef */
typedef struct {
      __IO uint32_t PCR[32]; /** Pin Control Register n, array
                             offset: 0x0, array step: 0x4 */
      __O  uint32_t GPCLR;   /** Global Pin Control Low Register,
                             offset: 0x80 */
      __O  uint32_t GPCHR;   /** Global Pin Control High
                             Register, offset: 0x84 */
           uint8_t RESERVED_0[24];
      __IO uint32_t ISFR;    /** Interrupt Status Flag Register,
                             offset: 0xA0 */
} PORT_Type;
```

Address	Register Name
?	PORTA_PCR0
? + 0x4	PORTA_PCR1
…	…
? + 0x78	PORTA_PCR30
? + 0x7C	PORTA_PCR31
? + 0x80	PORTA_GPCLR
? + 0x84	PORTA_GPCHR
…	(reserved)
? + 0xA0	PORTA_ISFR

Locating PORTA Control Register in Memory

- Header file MKL25Z4.h provides support including:
  - Pointers to the bases of registers as PORT_Type structs
  - Register instance definitions matching register names

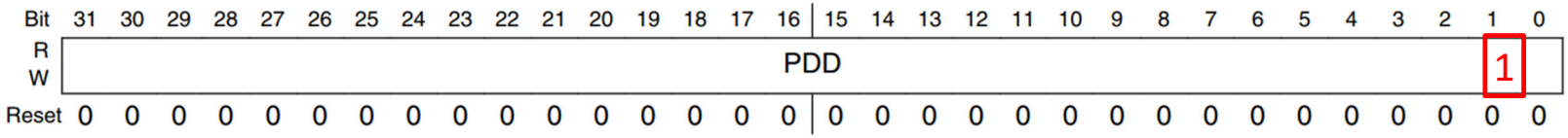
```
/** Peripheral PORTA base address */
#define PORTA_BASE     (0x40049000u)
/** Peripheral PORTA base pointer */
#define PORTA ((PORT_Type *)PORTA_BASE)
/* PORT - Register accessors */
#define PORT_PCR_REG(base,index) ((base)->PCR[index])
/* PORT - Register instance definitions */
#define PORTA_PCR0 PORT_PCR_REG(PORTA,0)
#define PORTA_PCR1 PORT_PCR_REG(PORTA,1)
...
```

Address	Register Name
4004_9000	PORTA_PCR0
4004_9004	PORTA_PCR1
…	…
4004_9078	PORTA_PCR30
4004_907C	PORTA_PCR31
4004_9080	PORTA_GPCLR
4004_9084	PORTA_GPCHR
…	(reserved)
4004_90A0	PORTA_ISFR

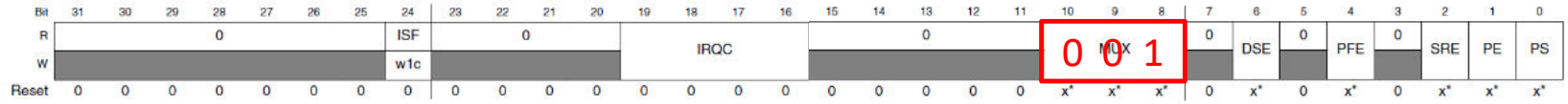
```
// Option 1: use pointer & data structure
PORTA->PCR[1] = …
PORTA->ISFR = …
// Option 2: use register instance defs
PORTA_PCR1 = …
PORTA_ISFR = …
```

Example: Configure Pin PTA1 as GPIO Output

- Configure GPIO: GPIOA_PDDR (at 0x400F_F014)
  - Set direction as output by changing bit 1 to 1



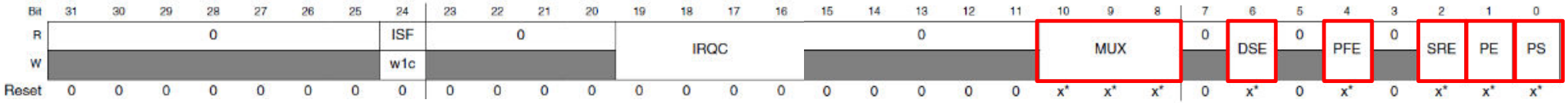
  - Access GPIOA_PDDR as `GPIOA->PDDR`
  - What next?
- Configure PORT: PORTA_PCR1 (at 0x4004_9004)
  - Select GPIO by changing MUX field (bits 10-8) to 001



  - Access PORTA_PCR1 field as `PORTA->PCR[1]`
  - What next? …

NC STATE UNIVERSITY

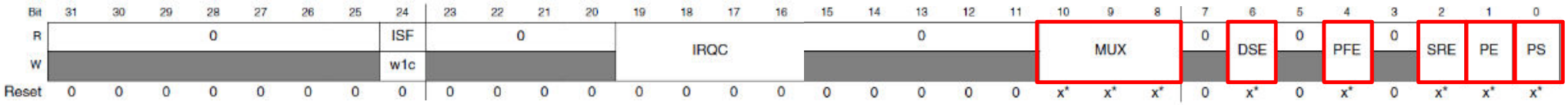
(B) Accessing Bits within a Register



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	0							ISF	0				IRQC				0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W								w1c																								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

- Software methods to access to *specific bits* within a control register
  - C language bitfields
  - Bitwise masking operations

(B1) Grouping Bits: Using Bitfields to Match Bit Layout

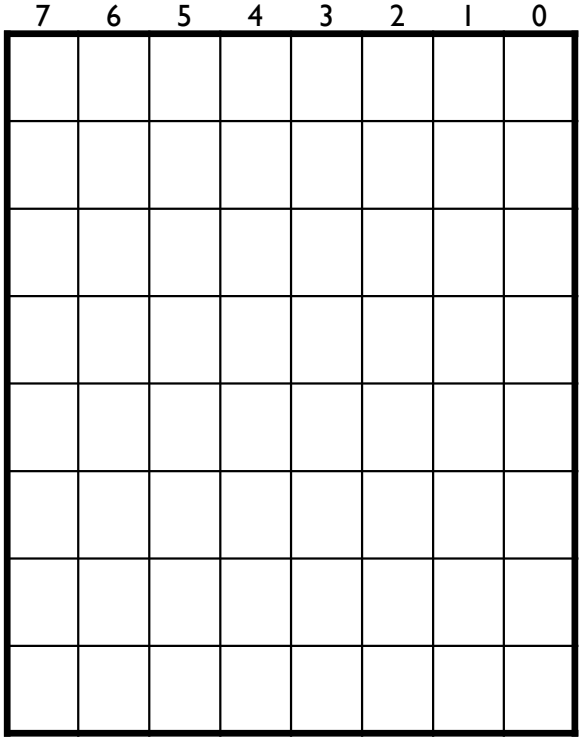


- Structure with Bitfields
  - Defines C data structure to represent MCU hardware register contents
  - Tells compiler the field's bit width (shorter than the base type)
  - Field is converted automatically to/from base type for processing
- Examples of use
  - Used by Arm for some core registers
  - Not used by NXP for peripherals, but would be possible

```
typedef struct {
   uint32_t PS:1;
   uint32_t PE:1;
   uint32_t SRE:1;
   uint32_t reserved_1:1;
   uint32_t PFE:1;
   uint32_t reserved_2:1;
   uint32_t DSE:1;
   uint32_t reserved_3:1;
   uint32_t MUX:3;
   …
} Example_PCR_Type;
```

(B2) Changing Bits with Bitwise Logical Operations

- No instructions to change some (but not all) bits in a register
  - Want to change two bit positions (b2, b1) of **var** to **val** (10_b) but leave other bits unchanged
- Improvise with bitwise logical operations and masks
- Zero out **var** bit positions to change by ANDing with mask
  - Mask has 0 in bit positions to change, 1 in all others
  - **var** &= 11111001
- Prepare value to write
  - **val** = 2
  - If needed, shift **val** to align with bits to change.
    (Left shift **<<** loads zeroes into new bits)
    - **val** <<= 1
  - Zero out bit positions in **val** where **reg** must be unchanged:
    - **val** &= 00000110
- Set **var** bit positions where corresponding **val** bit is one by ORing with **val**
  - **var** |= **val**



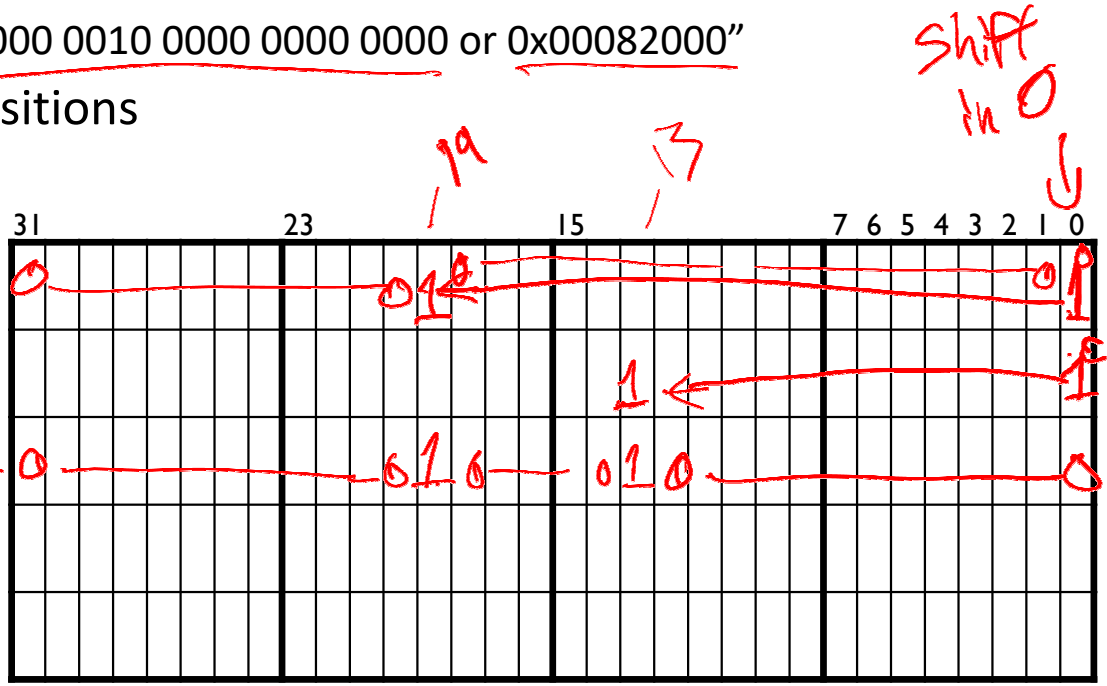
Using Macros to Simplify Bit Access

- Easy to make mistakes dealing with literal binary and hexadecimal values
  - "To set bits 13 and 19, use 0000 0000 0000 1000 0010 0000 0000 0000 or 0x00082000"
- Make the literal value from shifted bit positions

```
n = (1UL << 19) | (1UL << 13);
```

- Define names for bit positions

```
#define GREEN_LED_POS (19)
#define YELLOW_LED_POS (13)
n = (1UL << GREEN_LED_POS) |
    (1UL << YELLOW_LED_POS);
```



- Define macro to shift to create simple mask by shifting one bit x positions left

```
#define MASK(x) (1UL << (x))
```

- This simple MASK macro supports only one-bit wide masks, must repeat for more bits

```
n = MASK(GREEN_LED_POS) | MASK(YELLOW_LED_POS);
```

Bitwise Operations with Masks

- **=** **Overwrite** all bits of existing value in n with new

```
n = new;
```

- **|=** **Set** in n all the bits which are one in mask, leaving others unchanged

```
n |= MASK(foo);
```

- **&=** **Clear** in n all the bits which are **zero** in mask, leaving others unchanged

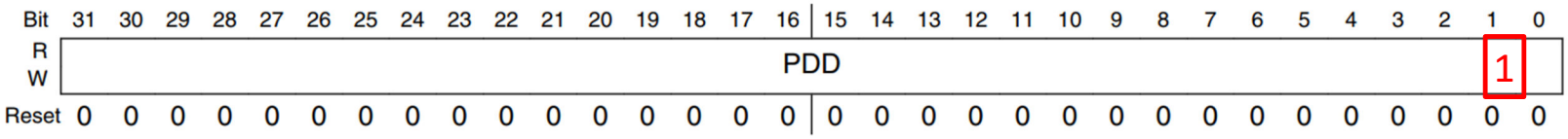
```
n &= MASK(foo);
```

- **~** Use the bitwise **complement** of a term. 1->0, 0->1
  - Example: **Clear** in n all the bits which are **one** in mask, leaving others unchanged

```
n &= ~MASK(foo);
```

Example: Configure GPIOA Bit 1 as GPIO Output

- GPIOA_PDDR (at 0x400F_F014)
  - Set direction as output by changing bit 1 to 1



  - Access GPIOA_PDDR as `GPIOA->PDDR`
- What next?
  - No names defined in CMSIS C support for individual bits, so need to create and apply own mask

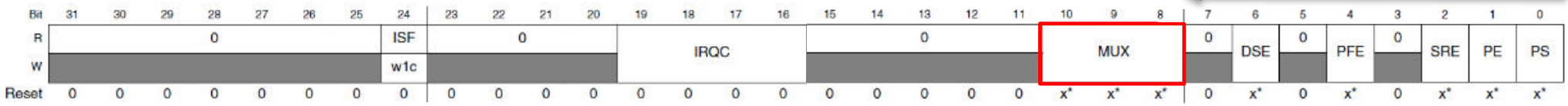
```
#define MASK(x)       (1UL << (x))
#define MY_BIT_POS    (1)
uint32_t new_val = 1; // assume it is aligned to bit 0
uint32_t shifted_val = new_val << MY_BIT_POS;
GPIOA->PDDR &= ~MASK(MY_BIT_POS); // zero out bit at MY_BIT_POS
GPIOA->PDDR |= shifted_val & MASK(MY_BIT_POS); // copy of shifted value
```

Better CMSIS-CORE Support for PORT PCR Registers

- Device Peripheral HAL provides better support for some peripheral's bit fields (e.g. PORT)
  - Device HAL file (MKL25Z4.h) provides mask, symbol and macros for each bit field
  - Naming conventions: *MODULE_REGISTER_FIELD […]*
- *MODULE_REGISTER_FIELD*_**SHIFT**: number indicating position of *FIELD*'s least-significant bit

```
#define PORT_PCR_MUX_SHIFT 8
```

PORT_PCR_MUX_SHIFT



- *MODULE_REGISTER_FIELD*_**MASK**: mask with 1s in *FIELD*'s bits, 0s in other bits

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

```
#define PORT_PCR_MUX_MASK  0x700u
```

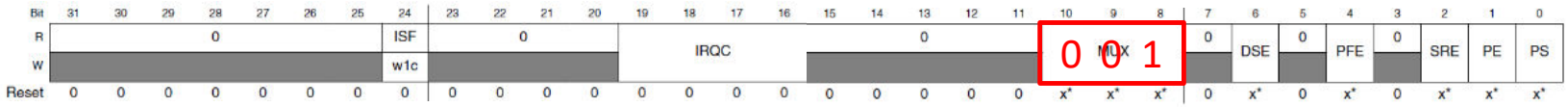
- *MODULE_REGISTER_FIELD*(**x**): macro to format x for *FIELD*.
  - Shift input x to align it with *FIELD*'s position
  - Zero out other bits

```
#define PORT_PCR_MUX(x) (((uint32_t)((uint32_t)(x))<<PORT_PCR_MUX_SHIFT))&PORT_PCR_MUX_MASK)
```

NC STATE UNIVERSITY

Example: Configure Pin PTA1 as GPIO Output

- PORTA_PCR1 (at 0x4004_9004)
  - Select GPIO by changing MUX field (bits 10-8) to 001



  - Access PORTA_PCR1 field as PORTA->PCR[1]

- What next? Macro PORT_PCR_MUX() defined for MUX field, so can use that macro

```
uint32_t new_val = 1;
PORTA->PCR[1] &= ~PORT_PCR_MUX_MASK; // zero out MUX field bits
PORTA->PCR[1] |=  PORT_PCR_MUX(new_val); // copy in new_val
```

CMSIS Macros

- Defined in core_cm0plus.h (included by MKL25Z4.h)
- Documentation: https://www.keil.com/pack/doc/CMSIS/Core/html/group__peripheral__gr.html
- Use preprocessor operators
  - ## makes preprocessor concatenate two terms
  - \ makes preprocessor ignore end of line and continue processing as if next line were appended to current line
- Convert value for placement in field

```
#define _VAL2FLD(field, value)  \
(((uint32_t)(value) << field ## _Pos) & field ## _Msk)
```

- Extract value of field

```
#define _FLD2VAL(field, value)  \
(((uint32_t)(value) & field ## _Msk) >> field ## _Pos)
```

NC STATE UNIVERSITY

Problem: Msk vs. MASK, Pos vs. SHIFT

- Different terms used in MKL25Z4.h, so let's create our own version of macros
- Convert value for placement in field

```
#define _VAL2FLDX(field, value) \
(((uint32_t)(value) << field ## _SHIFT) & field ## _MASK)
```

  - ## is preprocessor concatenation operator

```
_VAL2FLDX(MOD_REG_FLD,3) => (((uint32_t)(3)<<MOD_REG_FLD_SHIFT) & MOD_REG_FLD_MASK)
```

- Extract value of field

```
#define _FLD2VALX(field, value) \
(((uint32_t)(value) & field ## _MASK) >> field ## _SHIFT)
```

Another Useful Macro (Not in CMSIS)

- Perform read/modify/write with correct shifting and masking in one macro

```
#define MODIFY_FIELD(reg, field, value) \
((reg) = ((reg) & ~(field ## _MASK)) | \
(((uint32_t)(value) << field ## _SHIFT) & field ## _MASK))

PORTA->PCR[0] = MODIFY_FIELD(PORTA->PCR[0],PORT_PCR_MUX, 0x4);

((PORTA->PCR[0])) = ((PORTA->PCR[0])) & PORT_PCR_MUX_MASK) | \
(((uint32_t)(0x4) << PORT_PCR_MUX_SHIFT) & PORT_PCR_MUX_MASK));
```