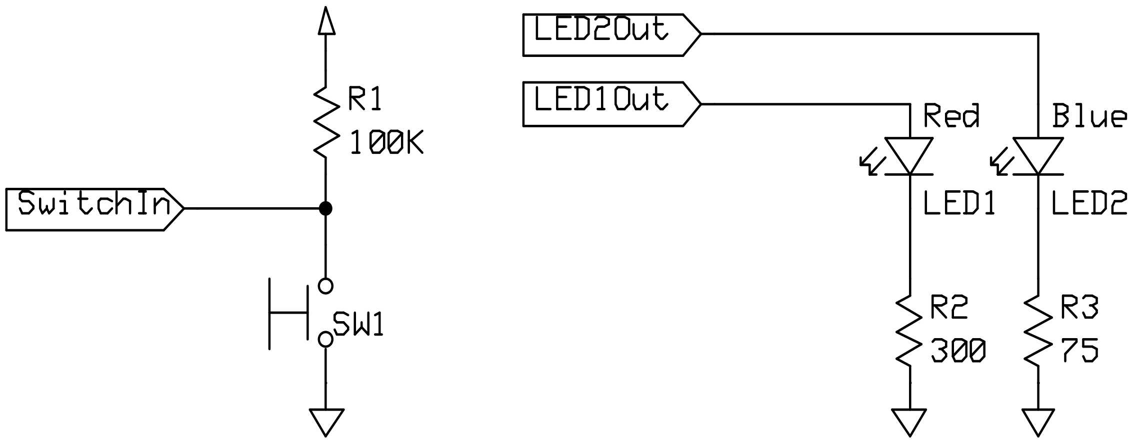
General Purpose I/O

Ravi Suppiah
Lecturer, NUS SoC

Overview

- How do we make a program light up LEDs in response to a switch?
- GPIO
  - Basic Concepts
  - Port Circuitry
  - Control Registers
  - Accessing Hardware Registers in C
  - Clocking and Muxing
- Circuit Interfacing
  - Inputs
  - Outputs
- Additional Port Configuration

Basic Concepts

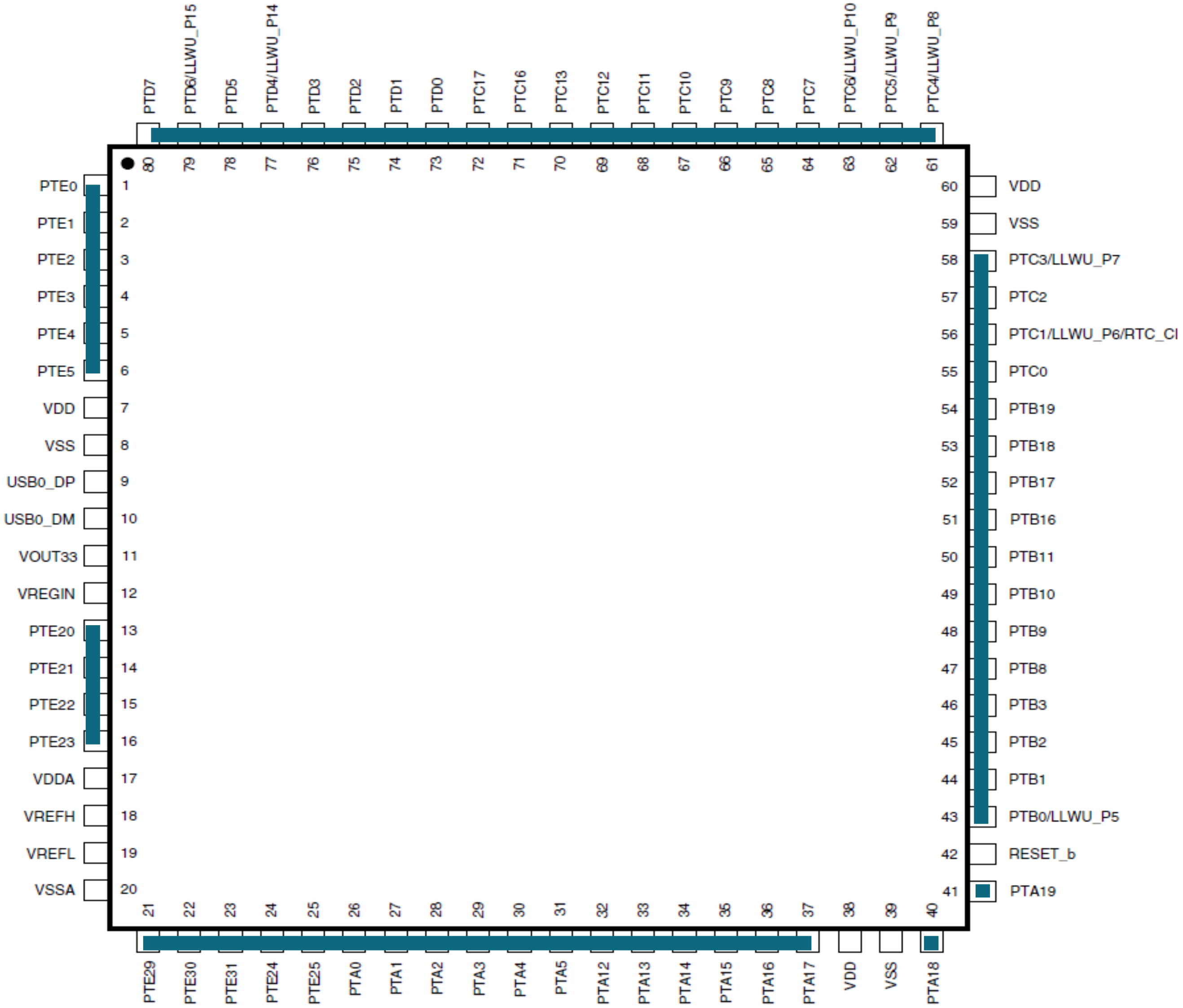


Let's do a Poll
pollev.com/rsnus

- Goal: light either LED1 or LED2 based on switch SW1 position
- GPIO = General-purpose input and output (digital)
  - Input: program can determine if input signal is a 1 or a 0
  - Output: program can set output to 1 or 0
- Can use this to interface with external devices
  - Input: switch
  - Output: LEDs

KL25Z GPIO Ports

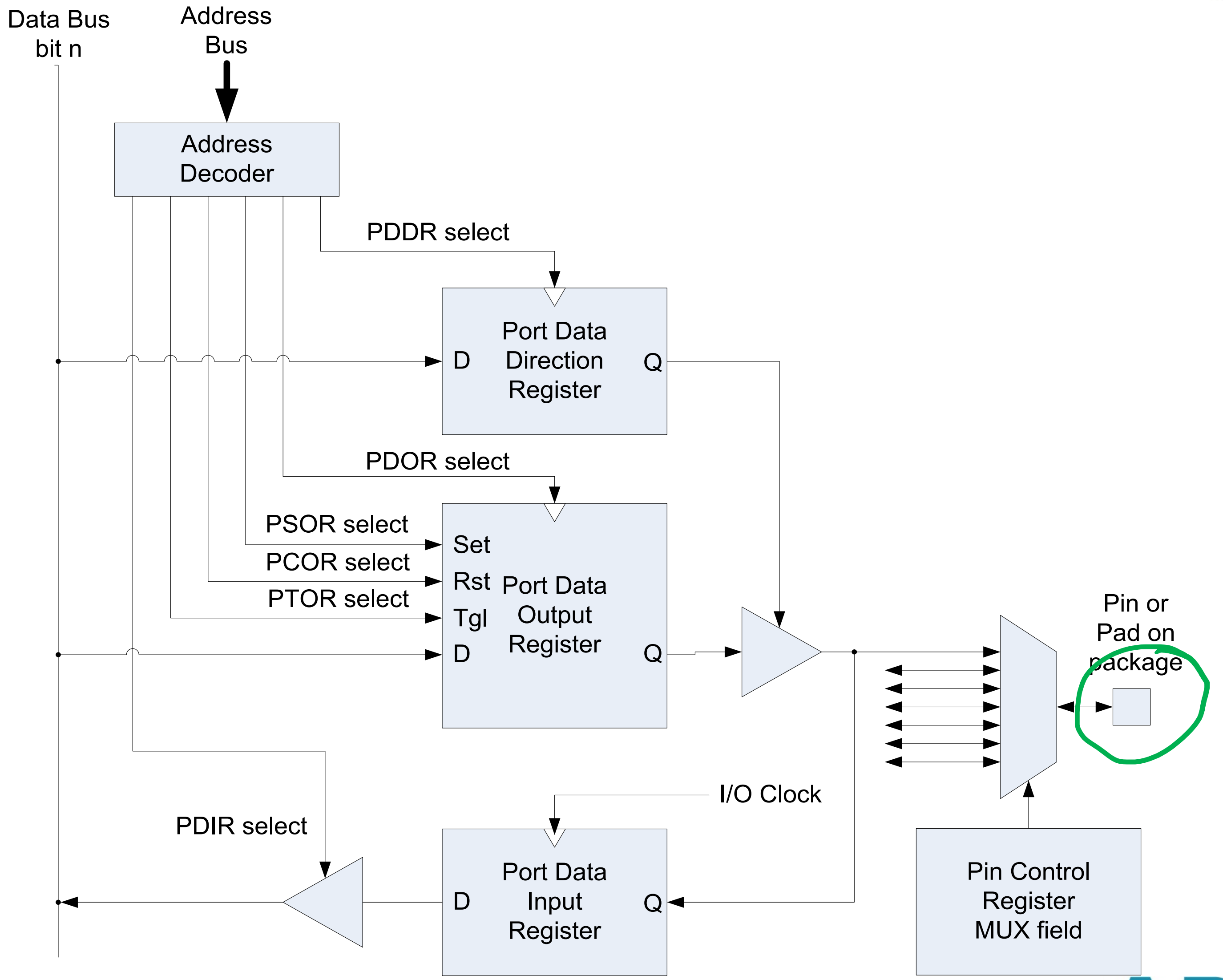
- Port A (PTA) through Port E (PTE)
- Not all port bits are available
- Quantity depends on package pin count



GPIO Port Bit Circuitry in MCU

- Control
  - Direction
  - MUX

- Data
  - Output (different ways to access it)
  - Input



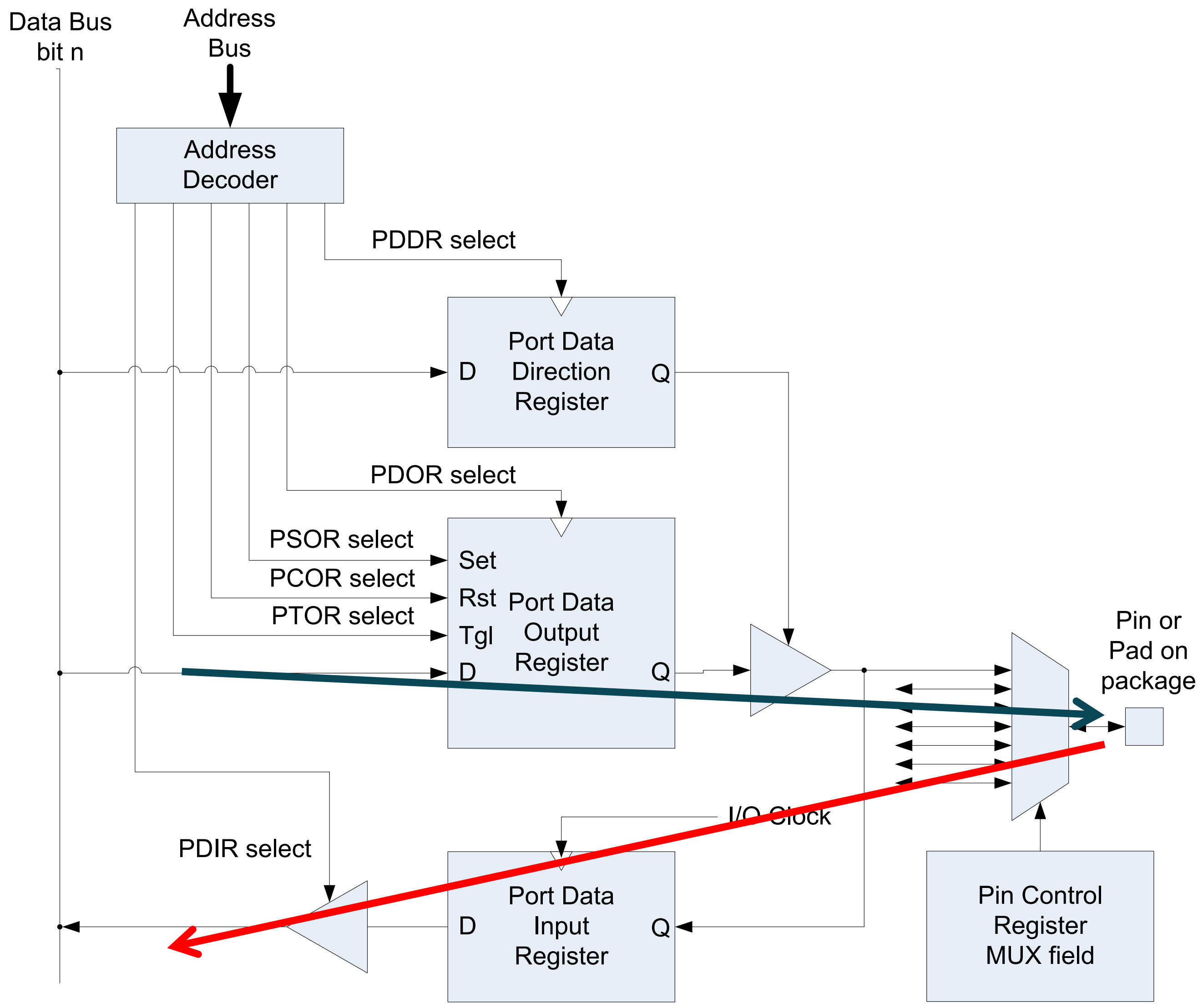
Control Registers

Absolute address (hex)	Register name	Width (in bits)
400F_F000	Port Data Output Register (GPIOA_PDOR)	32
400F_F004	Port Set Output Register (GPIOA_PSOR)	32
400F_F008	Port Clear Output Register (GPIOA_PCOR)	32
400F_F00C	Port Toggle Output Register (GPIOA_PTOR)	32
400F_F010	Port Data Input Register (GPIOA_PDIR)	32
400F_F014	Port Data Direction Register (GPIOA_PDDR)	32

- One set of control registers per port
- Each bit in a control register corresponds to a port bit

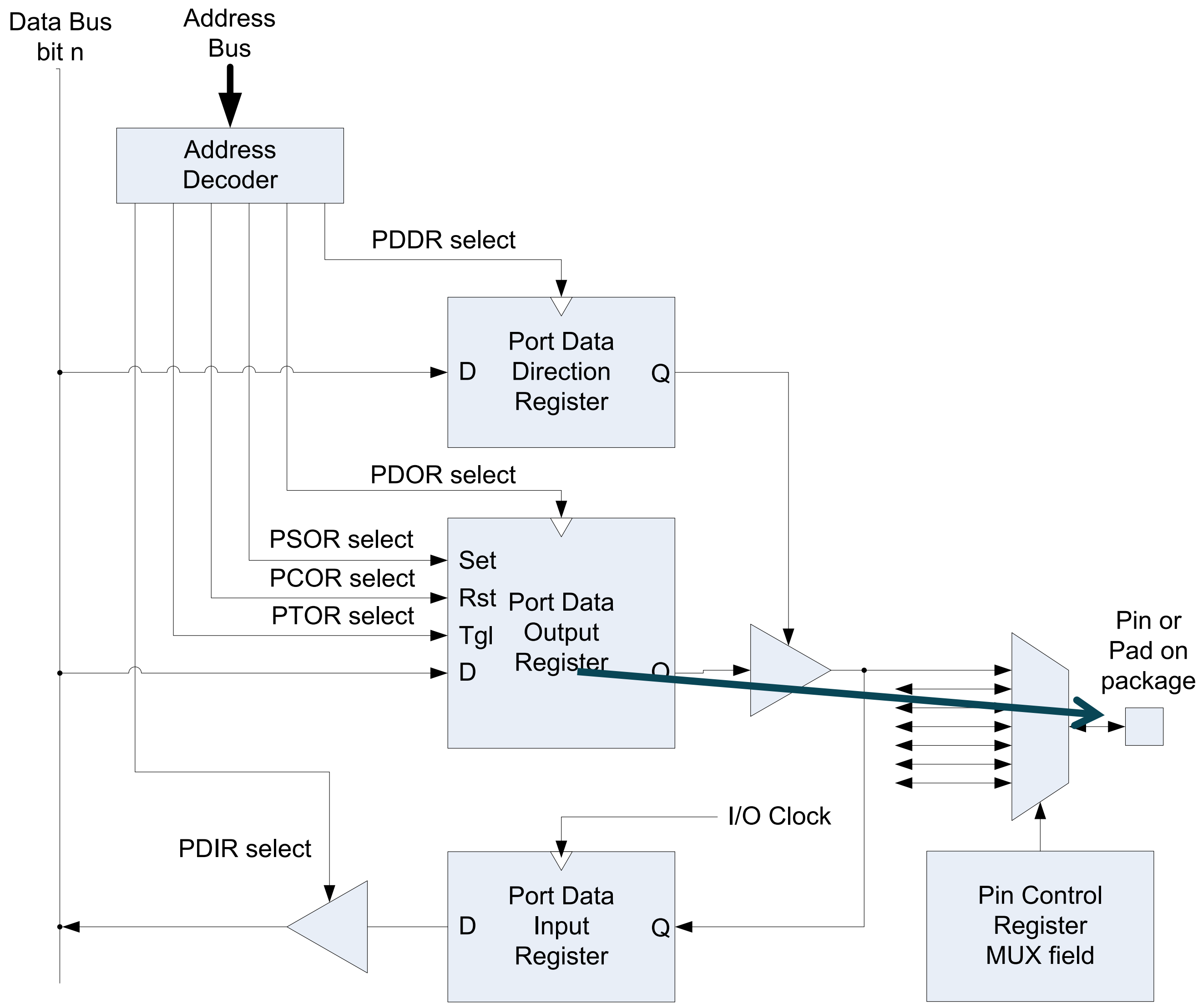
PDDR: Port Data Direction

- Each bit can be configured differently
- Input: 0
- Output: 1
- Reset clears port bit direction to 0



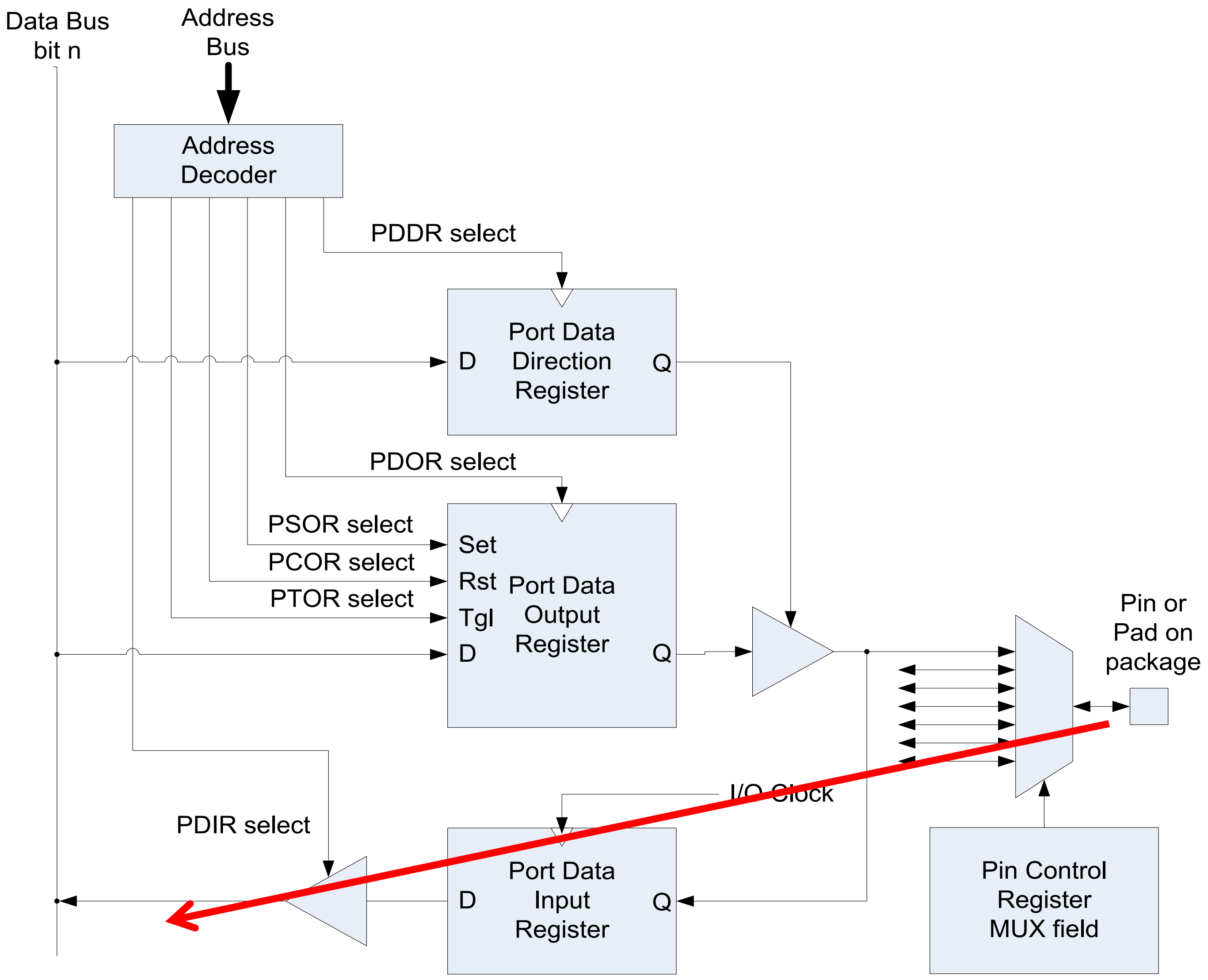
Writing Output Port Data

- Direct: write value to PDOR
- Toggle: write 1 to PTOR
- Clear (to 0): Write 1 to PCOR
- Set (to 1): write 1 to PSOR



Reading Input Port Data

- Read from PDIR
- Corresponding bit holds value which was read



Pseudocode for Program

```
// Make PTA1 and PTA2 outputs
set bits 1 and 2 of GPIOA_PDDR
// Make PTA5 input
clear bit 5 of GPIOA_PDDR
// Initialize the output data values: LED 1 off, LED 2 on
clear bit 1, set bit 2 of GPIOA_PDOR
// read switch, light LED accordingly
do forever {
	if bit 5 of GPIOA_PDIR is 1 {
		// switch is not pressed, then light LED 2
		set bit 2 of GPIOA_PDOR
		clear bit 1 of GPIO_PDOR
	} else {
		// switch is pressed, so light LED 1
		set bit 1 of GPIOA_PDOR
		clear bit 2 of GPIO_PDOR
	}
}
```

CMSIS - Accessing Hardware Registers in C

- Header file MKL25Z4.h defines C data structure types to represent hardware registers in MCU with CMSIS-Core hardware abstraction layer

```
/** GPIO - Register Layout Typedef */
typedef struct {
  __IO uint32_t PDOR;      /**< Port Data Output Register, offset: 0x0 */
  __O  uint32_t PSOR;      /**< Port Set Output Register, offset: 0x4 */
  __O  uint32_t PCOR;      /**< Port Clear Output Register, offset: 0x8 */
  __O  uint32_t PTOR;      /**< Port Toggle Output Register, offset: 0xC */
  __I  uint32_t PDIR;      /**< Port Data Input Register, offset: 0x10 */
  __IO uint32_t PDDR;      /**< Port Data Direction Register, offset: 0x14 */
} GPIO_Type;
```

Accessing Hardware Registers in C (2)

- Header file MKL25Z4.h declares pointers to the registers

```
/* GPIO - Peripheral instance base addresses */
/** Peripheral PTA base address */
#define PTA_BASE    (0x400FF000u)
/** Peripheral PTA base pointer */
#define PTA         ((GPIO_Type *)PTA_BASE)

PTA->PDOR = …
```

Coding Style and 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);
```

- Create macro to do shifting to create mask

```
#define MASK(x) (1UL << (x))
n = MASK(GREEN_LED_POS) | MASK(YELLOW_LED_POS);
```

Using Masks

- Overwrite existing value in n with mask

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

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

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

- Complement the bit value of the mask

```
~MASK(foo);
```

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

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

C Code

```c
#define LED1_POS (1)
#define LED2_POS (2)
#define SW1_POS (5)
#define MASK(x) (1UL << (x))

PTA->PDDR |= MASK(LED1_POS) | MASK (LED2_POS); // set LED bits to outputs
PTA->PDDR &= ~MASK(SW1_POS); // clear Switch bit to input

PTA->PDOR = MASK(LED2_POS);  // turn on LED1, turn off LED2

while (1) {
      if (PTA->PDIR & MASK(SW1_POS)) {
        // switch is not pressed, then light LED 2
        PTA->PDOR = MASK(LED2_POS);
      } else {
        // switch is pressed, so light LED 1
        PTA->PDOR = MASK(LED1_POS);
      }
}
```

Clocking Logic

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

- Need to enable clock to GPIO module
- By default, GPIO modules are disabled to save power
- Writing to an unclocked module triggers a hardware fault!
- Control register SIM_SCGC5 gates clocks to GPIO ports
- Enable clock to Port A

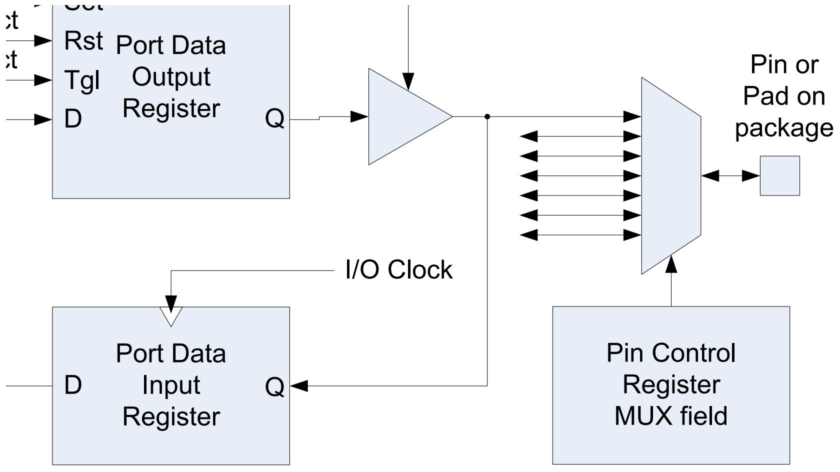
```
SIM->SCGC5 |= (1UL <<  9);
```

- Header file MKL25Z4.h has definitions

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

- #define SIM_SCGC5_PORTA_MASK 0x200u
- #define SIM_SCGC5_PORTA_SHIFT 9
- #define SIM_SCGC5_PORTA_WIDTH 1
- #define SIM_SCGC5_PORTA(x) (((uint32_t)(((uint32_t)(x))<<SIM_SCGC5_PORTA_SHIFT))&SIM_SCGC5_PORTA_MASK)

Connecting a GPIO Signal to a Pin



- Multiplexer used to increase configurability - what should pin be connected with internally?
- Each configurable pin has a Pin Control Register

Pin Control Register

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*

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				

- 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

CMSIS C Support for PCR

- 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;
```

CMSIS C Support for PCR

- Header file defines pointers to PORT_Type registers

```
/* PORT - Peripheral instance base addresses */
/** Peripheral PORTA base address */
#define PORTA_BASE          (0x40049000u)
/** Peripheral PORTA base pointer */
#define PORTA               ((PORT_Type *)PORTA_BASE)
```

- Also defines macros and constants

```
#define PORT_PCR_MUX_MASK      0x700u
#define PORT_PCR_MUX_SHIFT     8
#define PORT_PCR_MUX(x)
(((uint32_t)(((uint32_t)(x))<<PORT_PCR_MUX_SHIFT)) &PORT_PCR_MUX_MASK)
```

Resulting C Code for Clock Control and Mux

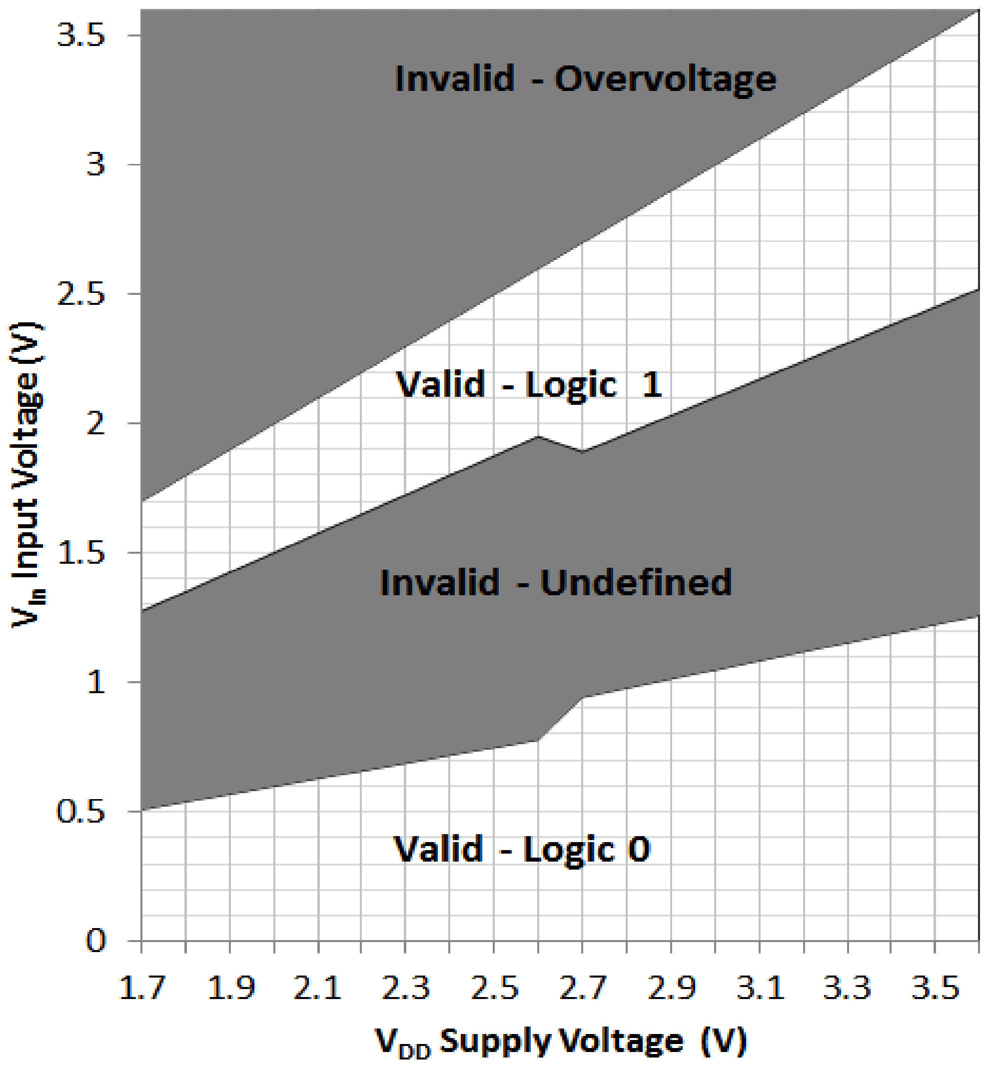
```
// Enable Clock to Port A
SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

// Make 3 pins GPIO
PORTA->PCR[LED1_POS] &= ~PORT_PCR_MUX_MASK;
PORTA->PCR[LED1_POS] |= PORT_PCR_MUX(1);
PORTA->PCR[LED2_POS] &= ~PORT_PCR_MUX_MASK;
PORTA->PCR[LED2_POS] |= PORT_PCR_MUX(1);
PORTA->PCR[SW1_POS] &= ~PORT_PCR_MUX_MASK;
PORTA->PCR[SW1_POS] |= PORT_PCR_MUX(1);
```

Inputs and Outputs, Ones and Zeros, Voltages and Currents

INTERFACING

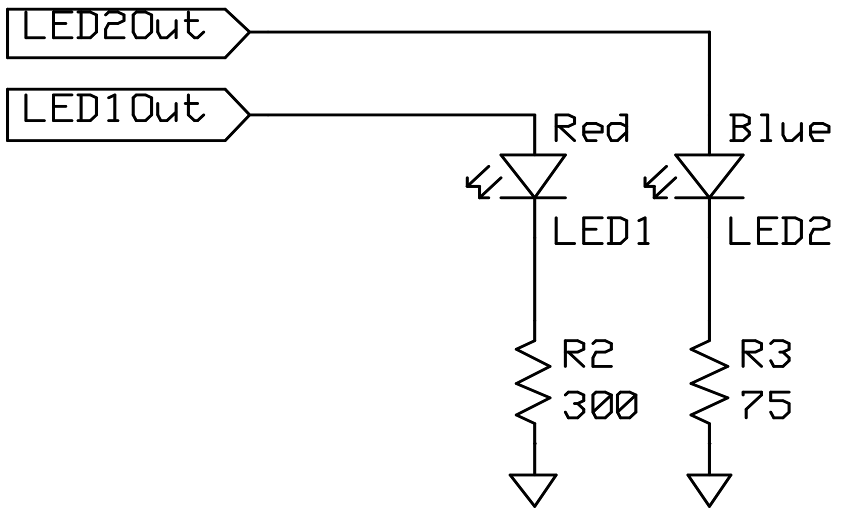
Inputs: What's a One? A Zero?

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



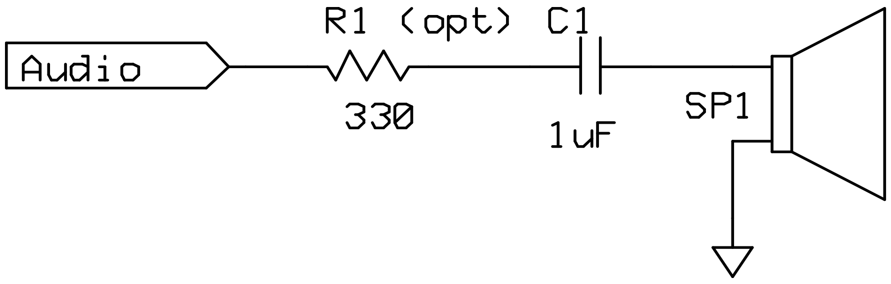
Output Example: Driving LEDs

- 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: ~2.7V
- Solve for R given VDD = ~3.0V
  - Red: 300 Ω
  - Blue: 75 Ω

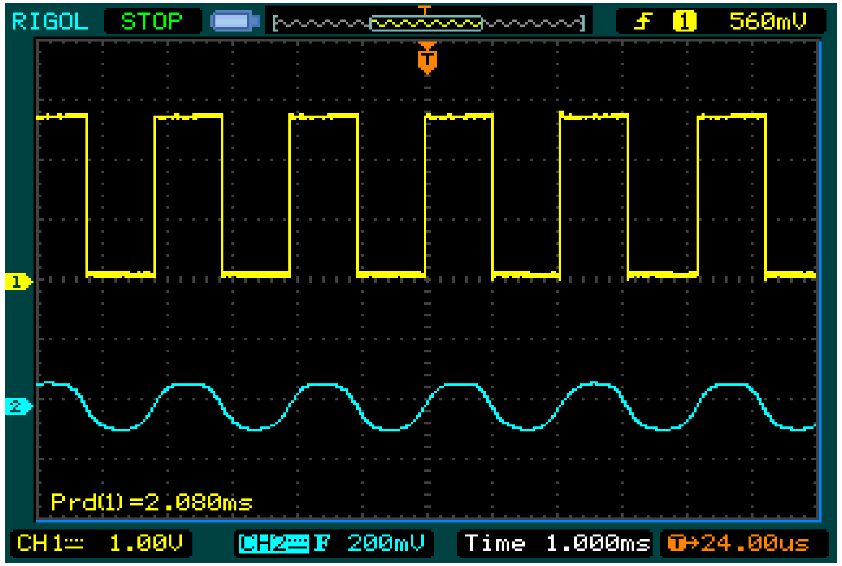


Output Example: Driving a Speaker

- Create a square wave with a GPIO output
- Use capacitor to block DC value
- Use resistor to reduce volume if needed
- Write to port toggle output register (PTOR) to simplify code




```
void Beep(void) {
   unsigned int period=20000;
   while (1) {
      PTC->PTOR = MASK(SPKR_POS);
      Delay(period/2);
   }
}
```



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*

- Pull-up and pull-down resistors
  - Used to ensure input signal voltage is pulled to correct value when high-impedance
  - PE: Pull Enable. 1 enables the pull resistor
  - PS: Pull Select. 1 pulls up, 0 pulls down.
- 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)
  - Available on some pins - MCU dependent

Lab 2

- Let's get prepared for Lab 2!