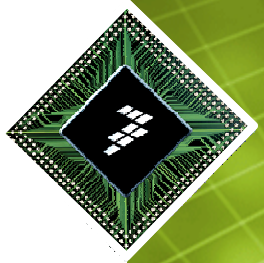
freescale™

Kinetis L GPIO

General Purpose Input/Output



Agenda

1. Overview
2. A Typical GPIO Design
3. Registers
4. Functional Description
5. GPIO Hands-On

Agenda

1. Overview
2. A Typical GPIO Design
3. Registers
4. Functional Description
5. GPIO Hands-On

General Purpose Input/Output

- **Overview**
  - What is the purpose of a microcontroller? Is it to compute some value, or monitor some signal? Either way data must enter and exit the microcontroller through some interface. The most basic of these interfaces are the General Purpose Input/Outputs (GPIOs). These modules allow the programmer to get input from devices such as switches, and outputting the values to display a status or result, for example, on LEDs.
- **Learning Objectives**
  - In this unit we will see what the GPIO modules consist of and what setting are needed for correct operation. We will also look into how they work together to with other units in an "embedded system".
- **Success Criteria**
  - When you have completed this module you will be able to setup and correctly use a GPIO pin in the Kinetis L Family of Microcontrollers.

Agenda

1. Overview
2. A Typical GPIO Design
3. Registers
4. Functional Description
5. GPIO Hands-On

A Typical GPIO Design

- Figure 1 shows the block diagram of the GPIO modules used in the Kinetis Family of microcontrollers. In general, most microcontrollers separate individual GPIO pins into groups, called ports.
- In the Kinetis microcontrollers, each port has 32 pins [31:0] as shown in the next diagram. Each of these ports are connected to the microcontroller via the peripheral bridge, which is a bus that allows slower components to talk to the microcontroller on a shared bus.
- This design is very common and helps to save on space within the chip. Since GPIO pins are not a high speed interface, they are normally either multiplexed or connected in a bus. Later on, we will see how each of these ports is organized.

A Typical GPIO Design

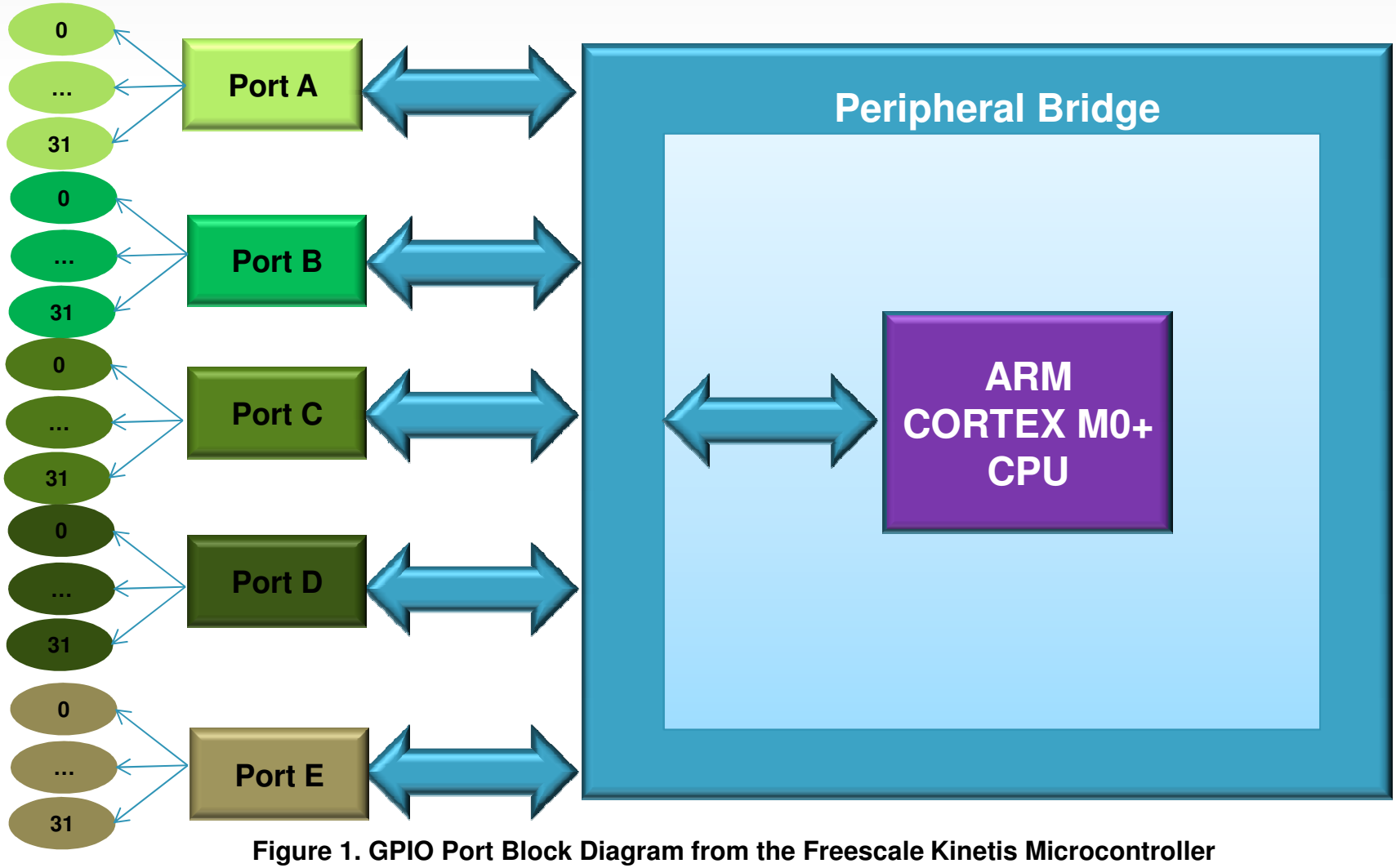


Figure 1. GPIO Port Block Diagram from the Freescale Kinetis Microcontroller

Agenda

1. Overview
2. A Typical GPIO Design
3. Registers
4. Functional Description
5. GPIO Hands-On

Registers

- As explained previously, in the Kinetis microcontrollers, the GPIO are connected into ports of 32 pins each. Each of these ports have a separate set of registers to control the pins. To control whether the GPIO pin is used as an input or an output, the user can set a bit in the Port Data Direction Register (GPIOx_PDDR) shown below in Figure 2.

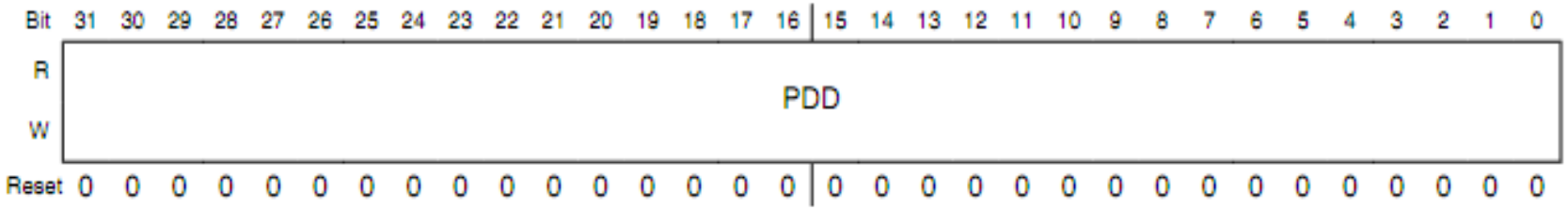


Figure 2. GPIO Port Data Direction Register (GPIOx_PDDR)

Registers

- If the pin is setup as an output there are four registers that can be modified to set the output value. The Port Data Output Register (GPIOx_PDOR) sets the value of a pin to logic level 0 when a 0 is written, and a logic level 1 when a 1 is written.
- The Port Set Output Register (GPIOx_PSOR) allows the user to write a 1 to it in order to set the logic level at 1. The Port Clear Output Register (GPIOx_PCOR) allows the user to write a 1 to it in order to clear the pin, thus setting it to logic level 0.
- This may seem like overkill, to have two extra registers, when you can do the same with 1. However, it really makes it easier to always set a bit to change the value of a output pin.

Registers

- The Port Toggle Output Register (GPIOx_PTOR) allows the user to write a 1 to it in order to toggle the current logic state.
- For example, if the current value is 1 then writing a 1 to the register will flip the output logic level to 0. Also, writing a 1 to the register when the current level is 0, flips the output logic level to 1.
- If the pin is setup as an input, then reading the value in the Port Data Input Register (GPIOx_PDIR) will give the current logic level on the pin. Through the use of these register, the user can read and write data onto the pin with ease.

Clocking GPIO modules

- The Kinetis microcontrollers were designed for very low power operation. One of the ways to reduce the power on a chip is to turn off the clocks to particular areas that are not in use. The default setting for the microcontroller is to have the clock disabled. In order to use a particular peripheral or resource the user needs to enable this clock to each item individually. This is a great feature, and builds upon the latest research in preventing wasted power by reducing the dynamic power required for a chip.

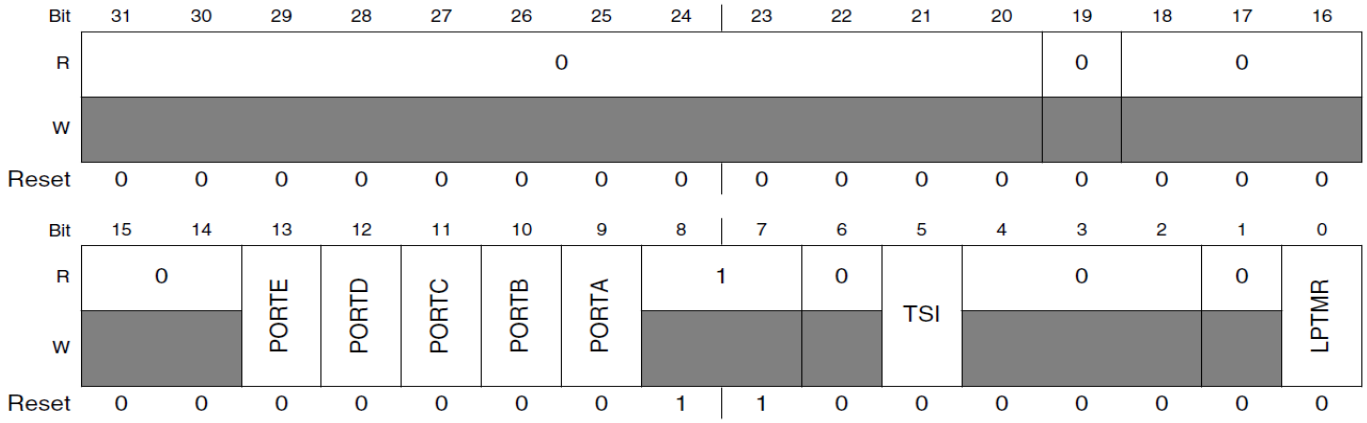


Figure 3. System Clock Gating Control Register 5 (SIM_SCGC5)

GPIO multiplexing

- Each pin on the microcontroller is connected to a multiplexer. This allows each pin to perform several functions as well as optimizing functionality in small packages. The signal multiplexer and other pin options can be configured in the Pin Control Register.

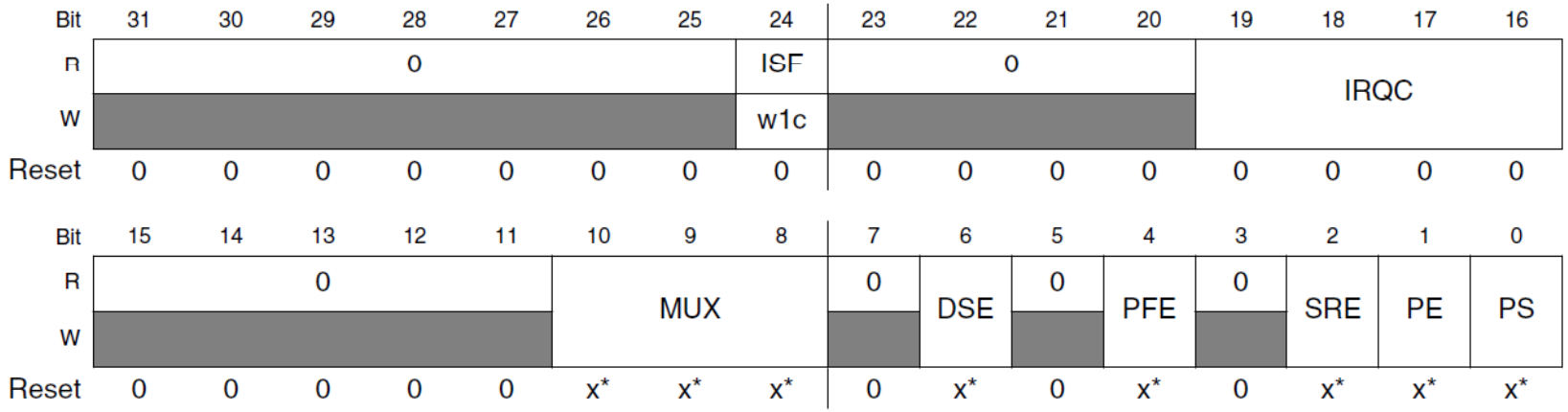


Figure 4. Pin Control Register n (PORTx_PCRn)

Agenda

1. Overview
2. A Typical GPIO Design
3. Registers
4. Functional Description
5. GPIO Hands-On

Functional Description

- Each GPIO pin has 6 registers that are used to modify and evaluate the state of the pin. The block diagram for the GPIO pin operations used in the Kinetis microcontrollers is shown below in Figure 5.

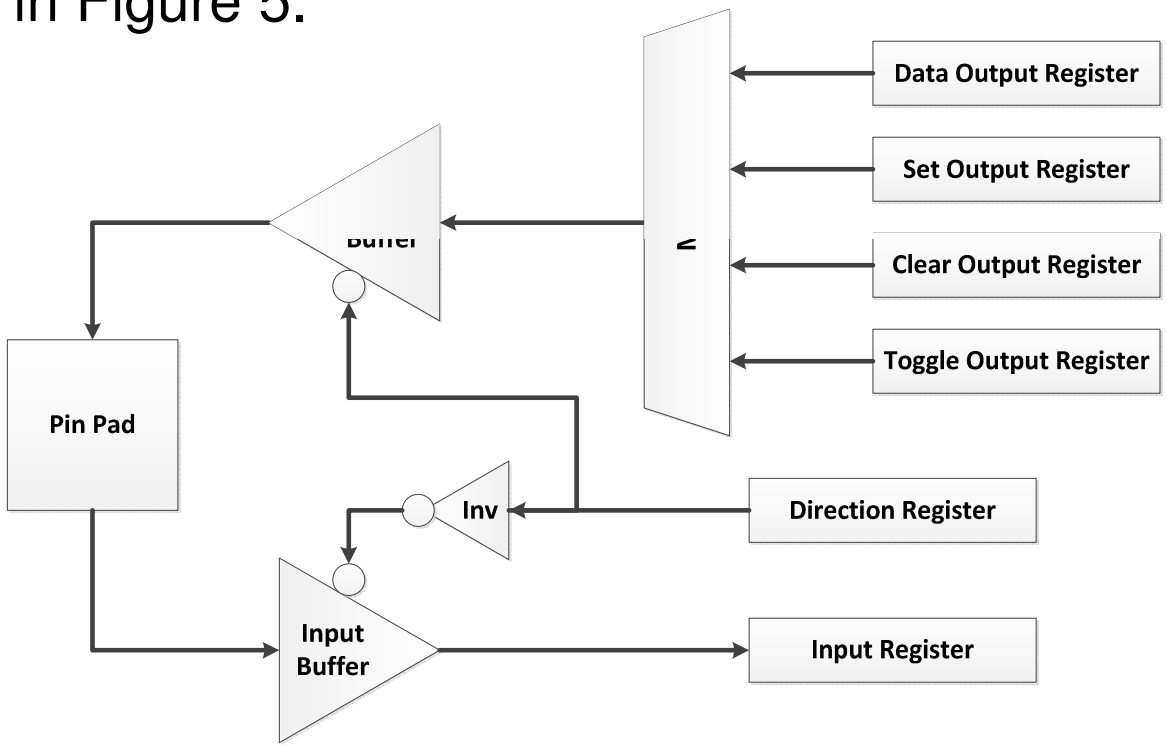


Figure 5: A GPIO Pin Block Diagram

Functional Description

- The actual GPIO pin is buffered into and out of the microcontroller so that the voltage levels do not short out the delicate chips circuits. These buffers are actually tri-state buffers since they can be set to a low logic level, a high logic level, and a high impedance value. These buffers are controlled via the data direction register.
- As is shown on the diagram, there is an inverter between this register and the input buffer, so in order to use the pin as an input a 0 must be written to the register. This also would turn off the output buffer so that the user doesn't accidentally drive the value on the pin and read it at the same time.

Functional Description

- The four output registers are multiplexed to the output buffer and this is controlled by logic that enables the register that was written to last. This way the user can write to any register at any time and update the value. The input buffer connects directly to the input register that can then be read by the user to determine the logic level on the pin.

Agenda

1. Overview
2. A Typical GPIO Design
3. Registers
4. Functional Description
5. GPIO Hands-On

GPIO Hands-On

- The FRDM-KL25Z board has one RGB LED connected to three different GPIOs, and each LED color could be enabled separately asserting the proper GPIO.

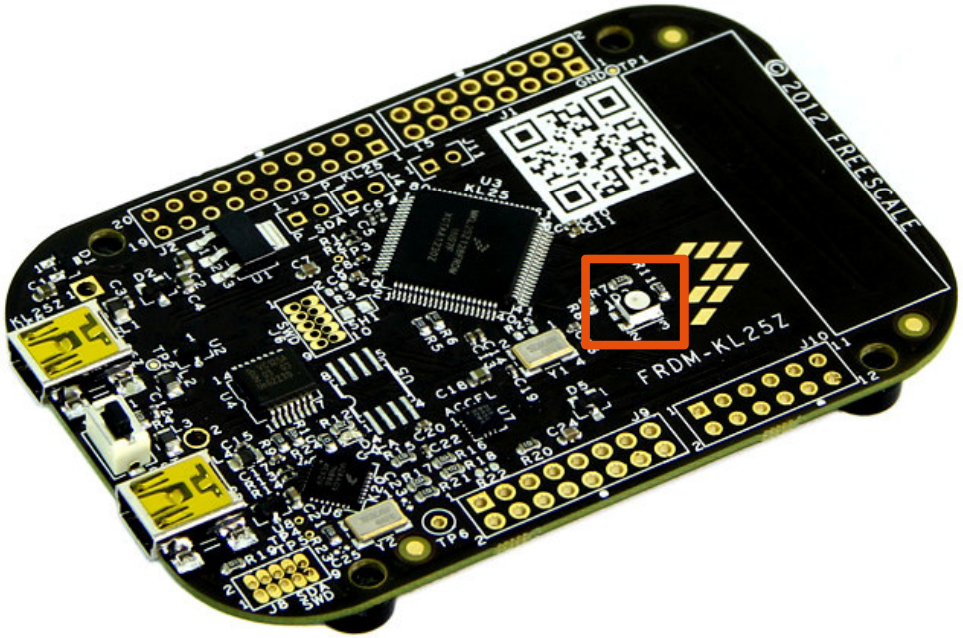


Figure 6: RGB LED on FRDM-KL25Z board

GPIO Hands-On

- Referring to the FRDM-KL25Z schematic, it indicates that Red LED is connected to PTB18, Green LED is connected to PTB18 and Blue LED is connected to PTD1.
- Besides, it is also shown that the RGB LED is internally connected in Common-Anode configuration, therefore, each LED should be turned off with “0”s.

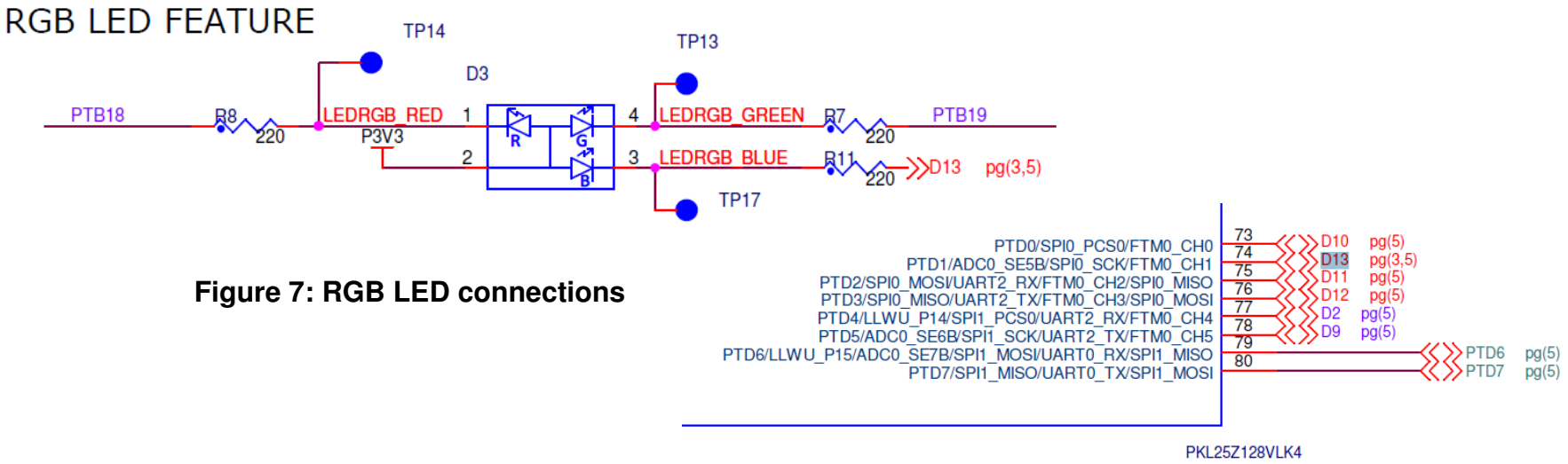


Figure 7: RGB LED connections

GPIO Hands-On

- The example code calls a function named “rgb_init” which first of all enables the clock gating for PORTB and PORTD.
- Then, the multiplexer registers are configured in order to use the pins as GPIOs.
- Then, the port data register is loaded with the initial values.
- Finally, the data direction registers configure the pins as outputs.

```
void rgb_init(void)
{
    /* Turn on clock to PortB and PortD module */
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK|SIM_SCGC5_PORTD_MASK;

    /* Set the PTB18 pin multiplexer to GPIO mode */
    PORTB_PCR18 = PORT_PCR_MUX(1);

    /* Set the initial output state to high */
    GPIOB_PSOR |= RED_SHIFT;

    /* Set the pins direction to output */
    GPIOB_PDDR |= RED_SHIFT;
```

GPIO Hands-On

- After the initialization, the code enters into an infinite loop on where each LED will be turned on and off in a specific sequence (Red, Green, Blue, Red+Green, Red+Blue, Green+Blue, and Red+Green+Blue).
- This code uses macros that easily write on Port Data registers, Port Set registers, Port Clear registers and Port Toggle registers.

```
int main(void)
{
    rgb_init();

    for(;;)
    {
        //RED only
        RED_ON;
        delay();
        RED_OFF;
        delay();

        //GREEN only
        GREEN_ON;
        delay();
        GREEN_OFF;
        delay();
```

Agenda

1. Overview
2. A Typical GPIO Design
3. Registers
4. Functional Description
5. GPIO Hands-On

freescale™


freescale™
semiconductor