# Interfacing GPIO using STM32F407 Microcontrollers

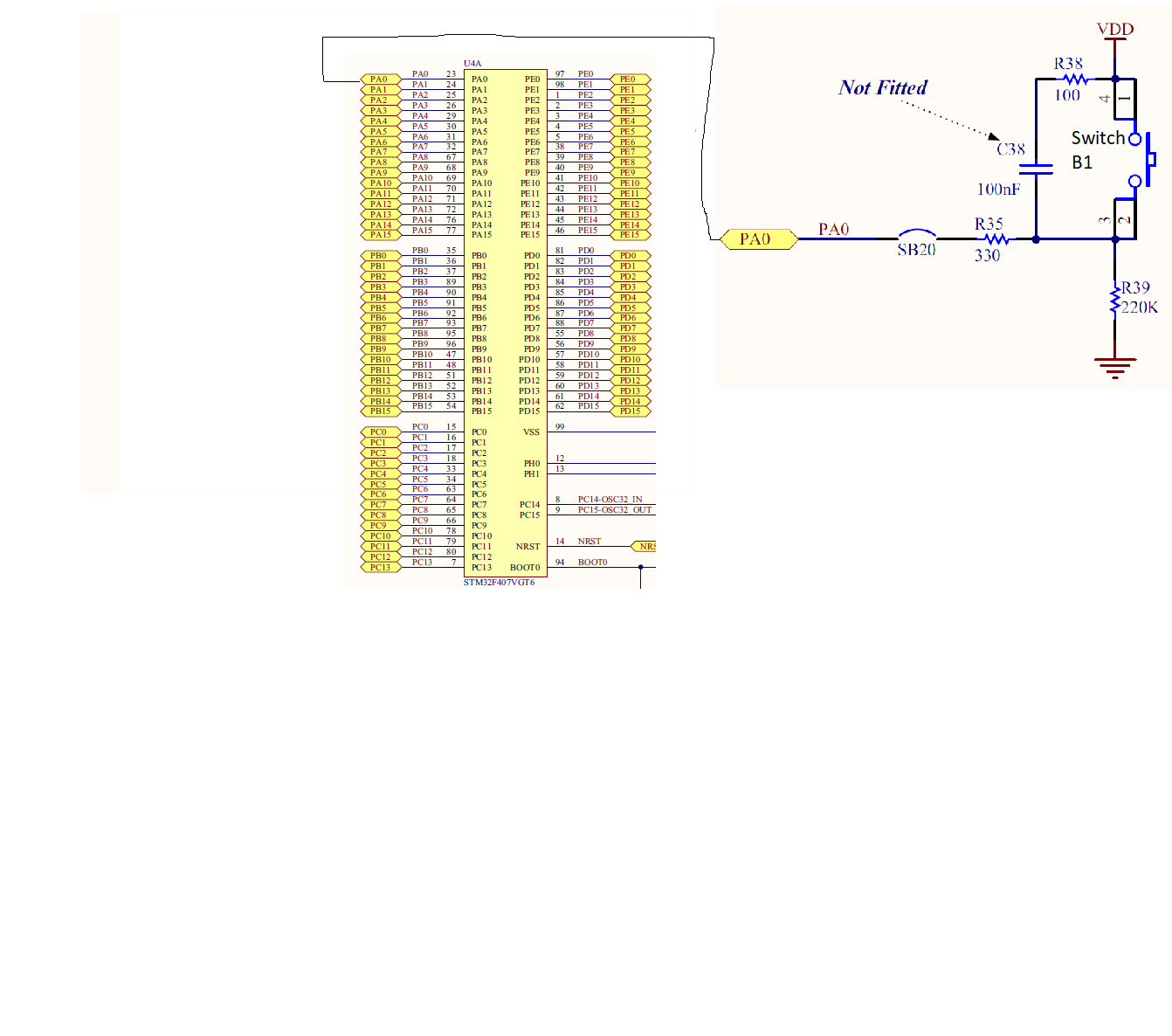
## General Purpose Input Output (GPIO):

A general purpose input output pin is used to read input from external sensor/input and process it or to generate output to drive an actuator/output. A microprocessor may require input or output of many types including:

1. Digital Input/Digital Output

A digital input is one that is usually 3.3 V high and 0 V low signal. Microcontroller reads 3.3 V as 1 and 0 V as 0. Similarly, microcontroller can generate 3.3V output when 1 is written on it and 0 V output when 0 is written on it

As an example, PA0 (port A pin 0) on STM32 microcontroller is connected to switch B1 shown below. We are interested in sensing if B1 is pressed or not. If VDD is 3.3 V, we can see easily that when B1 is not pressed, PA0 will have 0 V. When B1 is pressed, PA0 will have 3.3 V. If our program can read PA0, we will get bit 0 if B1 is not pressed and bit 1 if B1 is pressed. By sensing B1, we can make decision and can controller any actuator (LED or relay) attached to turn ON or OFF.



In this section, we will study digital input and output extensively

1. Analogue input or analogue output

An analogue input can be from any time varying sensor measurement. It can vary from 0 to 3.3 V e.g., 2.4 V value. Such quantities are usually the result of temperature or humidity measurement, position measurement, voltage or current measurement with may be directly proportional to measured quantity. Since a microcontroller can store only digital values, an analog to digital converter (ADC) is used to convert measure value into digital bits. In order to generate digital values at output pin, Digital to Analogue Converter DAC is used to generate a value from given bits.

1. Time multiplexed input/output

The time multiplexed functionality in controller is used generate signals. Signal are generated mainly for communication with external peripherals. It is also used for measurement of timed events. Assume we PC to control external peripherals. We can send signals to microcontroller by implementing any communication protocol like USB, serial or ethernet. One example is to create our own barcode reader using STM32F407 and then interface it with PC using USB interface.

### Interfacing General Purpose Input Output of STM32F407

Like any other microcontroller, GPIO peripheral is designed rich in features it can support. A single GPIO can be configured to operate as:

1. Digital Input
2. Digital Output
3. Analogue Input
4. Analogue Output
5. Communication Interface Line

Note that pin can have one functionality enabled at a time. Some pins may have all functionalities listed above while others may have limited functionality available.

Most if not all peripherals available in ARM Cortex-M4 processors have gated clock signal. What this means is that peripheral clock are kept disabled at power rest to save power consumption. Only those peripherals that are used in project require clock signals enabled. Hence the first step in using any peripheral in STM32F4 is to enable it’s clock.

The steps to configure GPIO in STM32 for input and output are mentioned below:

1. First, we need to identify Reset and Clock Control (RCC) register responsible to gate enable GPIO clock. In case of STM32 GPIO ports, all port clocks are available on Advanced High Performance Bus (AHB1) (see datasheet section 7.3.16).

Diagram, calendar

Description automatically generated

As an example, the C instruction to enable port A clock will be:

RCC -> AHB1ENR = RCC -> AHB1ENR | = 0x01;

Here note that GPIOAEN is bit 0 of register AHB1ENR and mask for bit 0 is 0x1.

The instruction can be written in short as:

RCC -> AHB1ENR | = 0x01;

Similarly, the C instruction to enable port C is:

RCC -> AHB1ENR | = 0x04;

Here note that GPIOAEN is bit 2 of register AHB1ENR and mask for bit 2 is 0x4 (b0100).

1. The GPIO itself consists of multiple registers to configure different functionalities. The first register mode register (MODER) is used to configure GPIO pin as:
   1. Input
   2. Output
   3. Alternate function
   4. Analog

A picture containing table

Description automatically generated

Here, we can see two are used to specify one of four options for each pin. As an example, we can configure GPIOA pin 2 for output using following C instructions:

GPIOA -> MODER = GPIOA -> MODER & ~(0x30);

GPIOA -> MODER = GPIOA -> MODER | (0x10);

Here, we first clear bits for pin 2 which are given as MODER2[1:0] in above diagram. The mask for these bits is 0x30 (b00110000). We first clear these bits to 0 using and operation and then set these bits to 01 using mask 0x10 (b00010000: remember each hex digits consist for 4 bits). A more convenient way is to write these C instructions as generic instruction is given as:

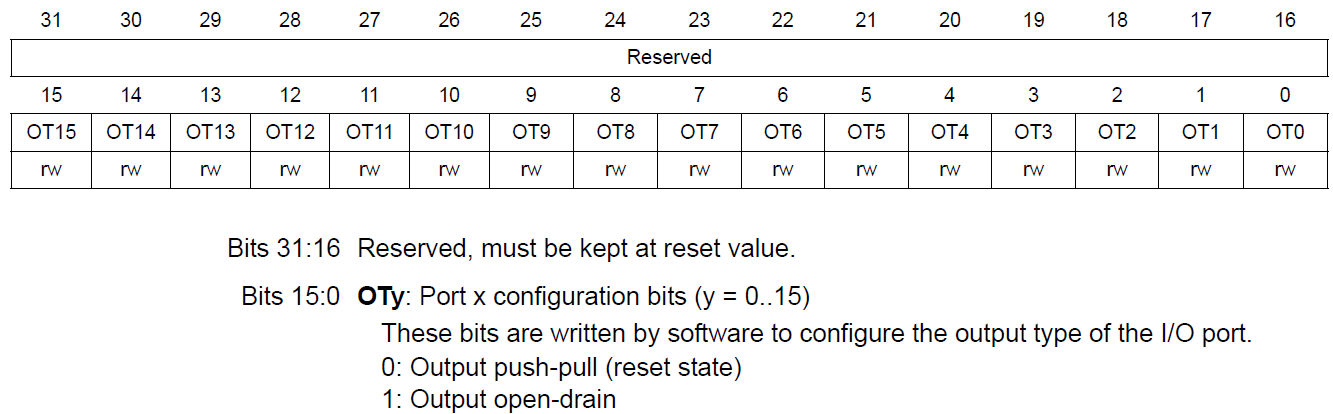
int pin = 2;

GPIOA -> MODER &= ~(0x3 << pin \* 2);

GPIOA -> MODER |= (0x1 << pin \* 2);

The second statement will generate same mask as the first statements and can be used for any pin number.

1. Next we have output type register (OTYPER). The output type register only needs to be configured if we are using pin as output pin or as alternate function pin. The register is given below:



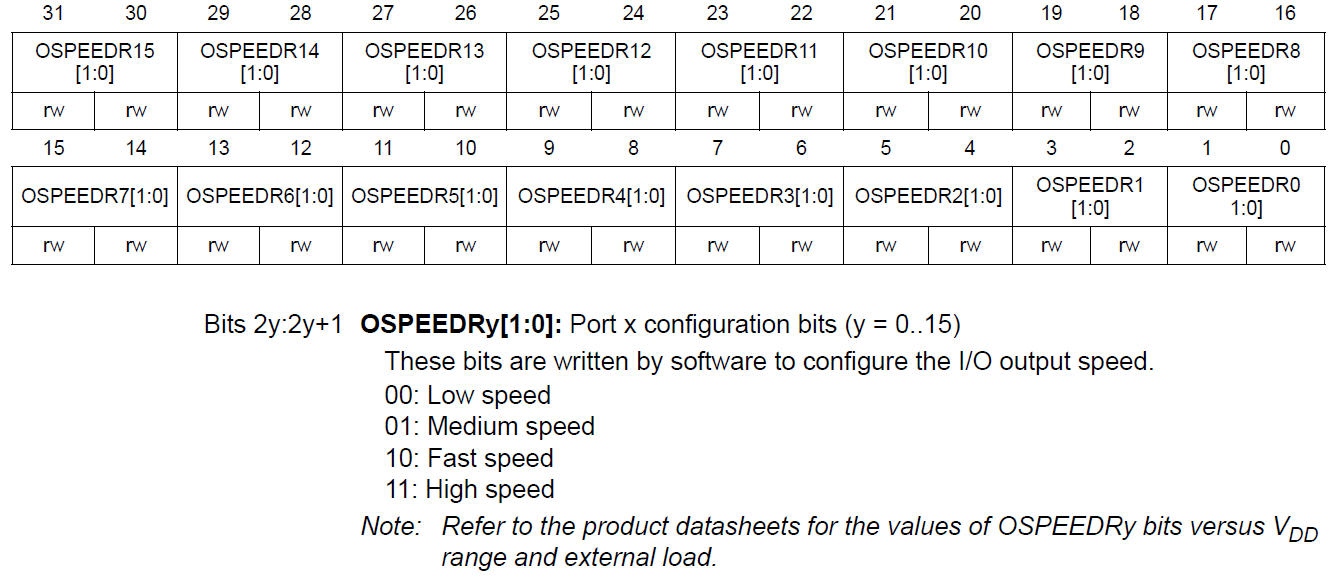
The output type is usually push-pull unless we need to drive some load from processor in which case we can make it open drain.

As an example, we can clear port C bit 9 to ensure it is push-pull type using following C instructions:

GPIOC -> OTYPER &= ~(0x100);

Here mask for bit 9 is 0x100 (b100000000).

1. Next we need to configure GPIO pin speed if pin is configured for output or alternate function using output speed register OSPEEDR.



STM supports 4 output speeds: low, medium, fast, high. The four options are set for each pin using 2 bits. As an example, the instruction to make port D pin 9 speed to fast speed, we can write following C instructions:

int pin = 9;

GPIOD -> OSPEEDR &= ~(0x3 << pin \* 2);

GPIOD -> OSPEEDR |= (0x2 << pin \* 2);

1. We need to configure output pin or depending upon peripheral type alternate function pin as pull-up using pull-up/pull-down register PUPDR. The pin can be left floating, pulled up or pulled down.

Table

Description automatically generated

Here 2 bits are used to configure a single pin on port. As an example, to configure pin 3 on port C as pulled down, we can write following C instructions:

int pin = 3;

GPIOC -> PUPDR &= ~(0x3 << pin \* 2);

GPIOC -> PUPDR |= (0x2 << pin \* 2);

There are other registers for GPIO as well, however simple GPIO configuration can be done using above 5 steps.

1. The next step is to read from a GPIO pin or write to a GPIO pin. To read from a GPIO pin, input data register IDR is used.

Graphical user interface, application, table, Excel

Description automatically generated

To read pin 3 from port E, we can use following C instruction:

int pin\_value = GPIOE -> IDR & 0x08;

In above statement, we are reading complete port IDR. However, we are interested in pin 3 only. The mask for pin 3 is b01000 or 0x08. Hence, using and operation all bits of IDR are 0 except bit 3 mapped to pin 3.

01011000

& 00001000

00001000

Hence, in this way we can capture value of pin 3. Here, please note that value is not captured as either 0 or 1, but as either 0 or 0x08 since we are directly storing pin 3 in variable without shifting.

1. To set a pin high or low after configuring it as output pin, we use output data register (ODR).

Graphical user interface, text, application

Description automatically generated

To set output pin 9 of GPIO port B as 1, we can use following instruction:

GPIOB -> ODR |= 0x100; // b100000000

To reset output pin 6 of port A, we can use following C instruction:

GPIOA -> ODR &= ~(0x40); // b01000000

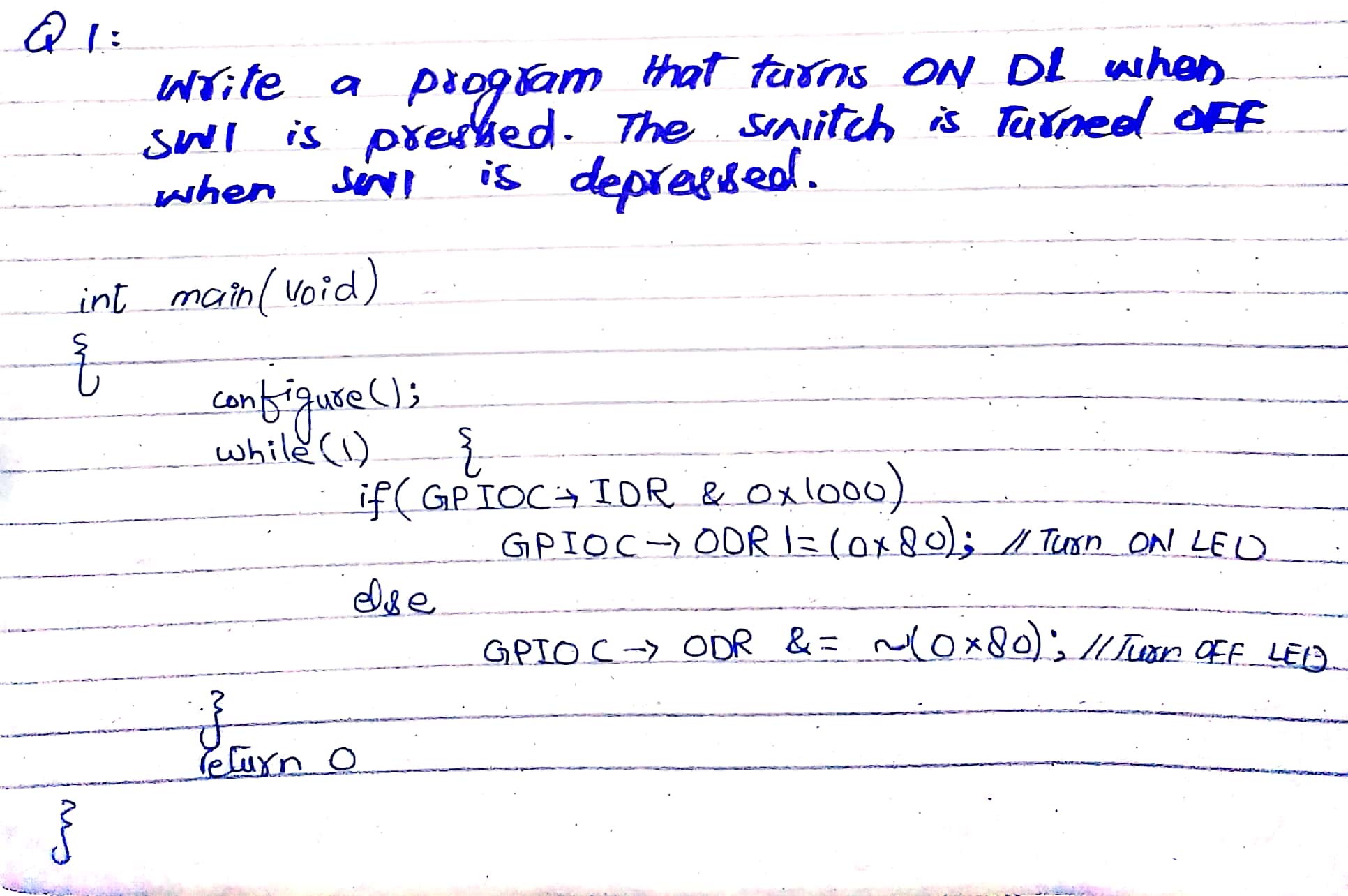
## Example:

A switch SW1 and LED D1 are connected to an STM32F4 processor. The SW1 is connected to port C pin 12 and D1 is connected to port C pin 7. We are connecting SW1 with car door open lamp. When door is closed (SW1 pressed), the door open indicator D1 stays off. When door is open (SW1 depressed), D1 turns on. You are required to write logic to turn off D1 when SW1 is pressed. The D1 remains on if SW1 is depressed.

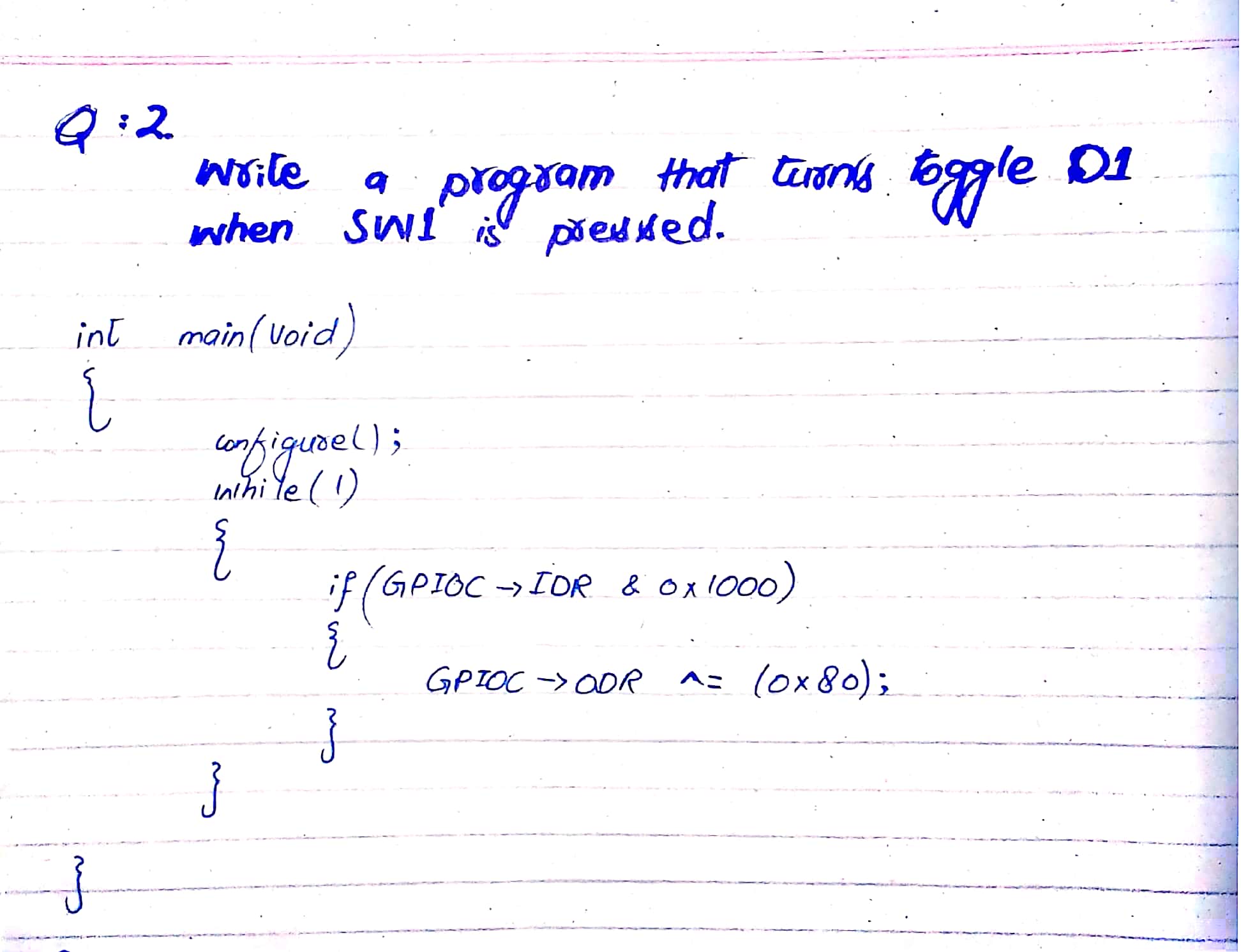
|  |
| --- |
| void configure(void)  {  int pin = 0;  // Enable clock for port C (RCC register bit 2 set to 1)  RCC->AHB1ENR |= 0x4;  // Update MODE register for port C pin 12 to input  pin = 12;  GPIOC->MODER &= ~(0x3 << pin \* 2);  GPIOC->MODER |= (0x0 << pin \* 2); // This statement can be skipped due to 0  // Update pin output type register to push-pull  GPIOC->OTYPER &= ~(0x1 << pin);  // Update pin speed register to high speed  GPIOC->OSPEEDR &= ~(0x3 << pin \* 2);  GPIOC->OSPEEDR |= (0x3 << pin \* 2);  // Update pin pull down register to pull down  GPIOC->PUPDR &= ~(0x3 << pin \* 2);  GPIOC->PUPDR |= (0x2 << pin \* 2);  // Update MODE register for port C pin 7 to output  pin = 7;  GPIOC->MODER &= ~(0x3 << pin \* 2);  GPIOC->MODER |= (0x1 << pin \* 2);  // Update pin output type register to push-pull  GPIOC->OTYPER &= ~(0x1 << pin);  // Update pin speed register to high speed  GPIOC->OSPEEDR &= ~(0x3 << pin \* 2); // This statment can be skipped due to 3  GPIOC->OSPEEDR |= (0x3 << pin \* 2);  // Update pin pull down register to no-pull  GPIOC->PUPDR &= ~(0x3 << pin \* 2);  GPIOC->PUPDR |= (0x0 << pin \* 2); // This statment can be skipped due to 0  }  int main(void)  {  configure();  while (1)  {  // Check if SW1 is pressed  if (GPIOC -> IDR & 0x1000) // bit12 b1 0000 0000 0000 = 0x1000  {  // Turn off D1  GPIOC->IDR &= ~(0x10); // reset bit7  }  else  {  // Turn ON D1  GPIOC->IDR |= (0x10); // set bit7  }  }  } |

## Assignment:

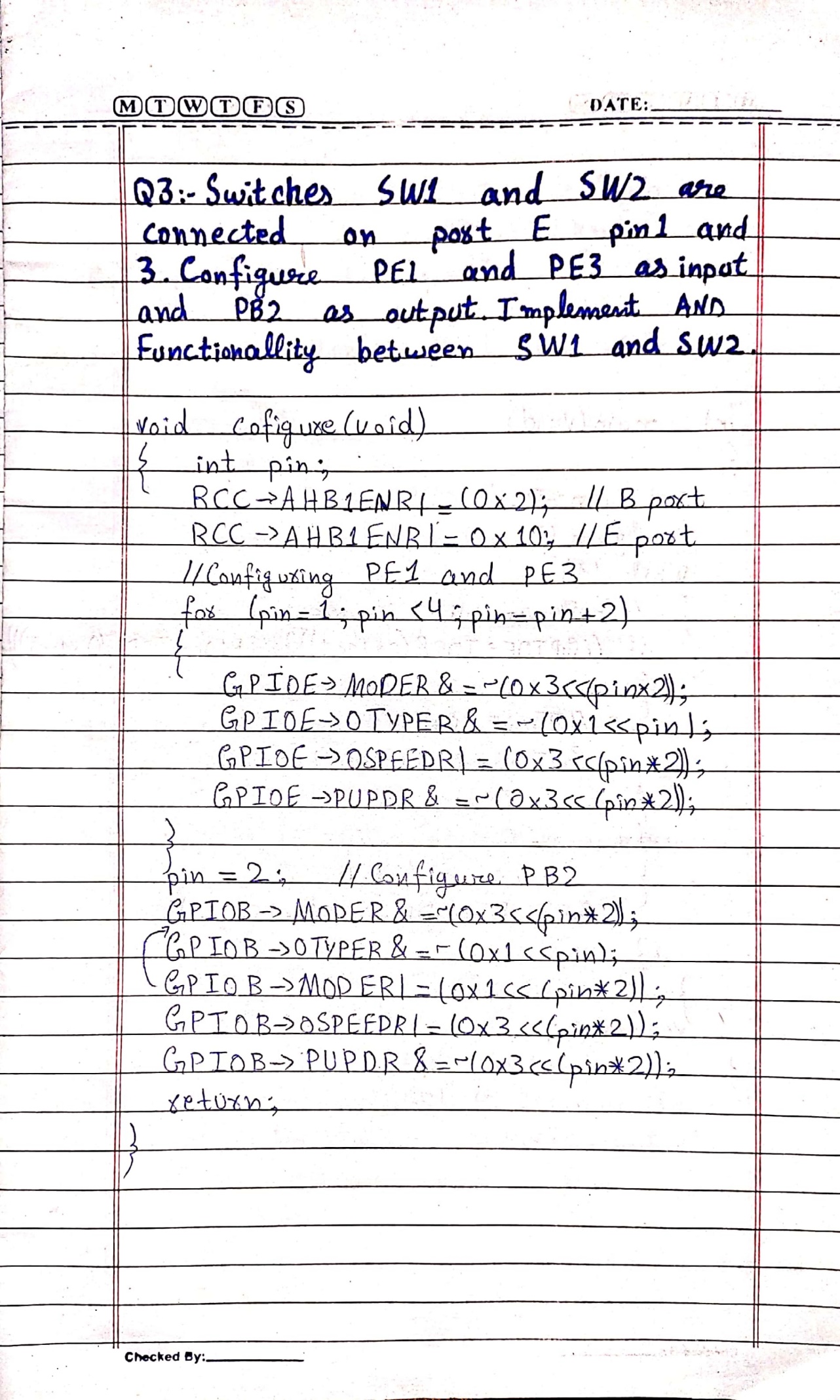
1. Write a program that turns ON D1 when SW1 is pressed. The switch is turned OFF when SW1 is depressed

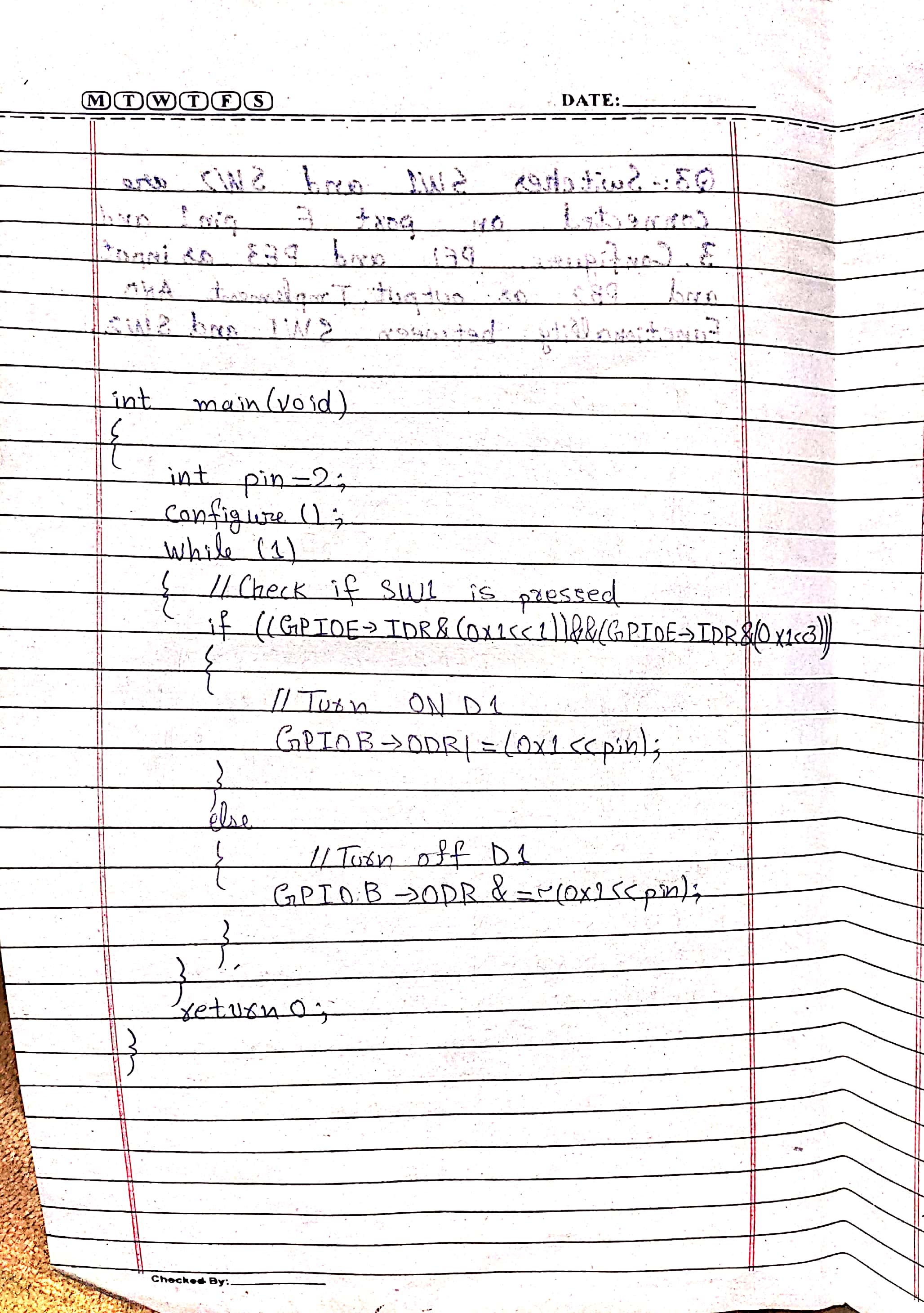


1. Write a program that turns toggles D1 when SW1 is pressed. Ignore debouncing

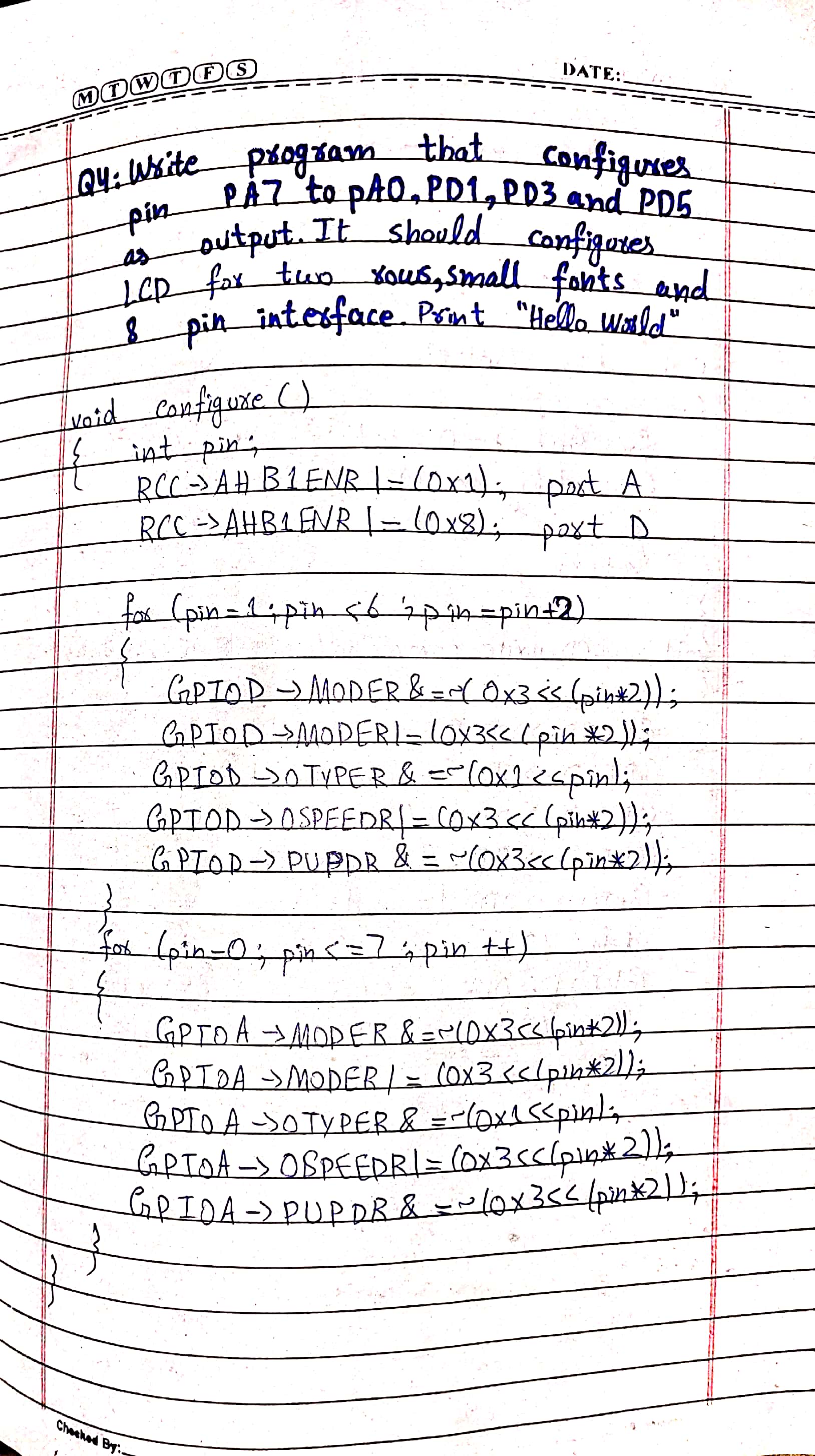


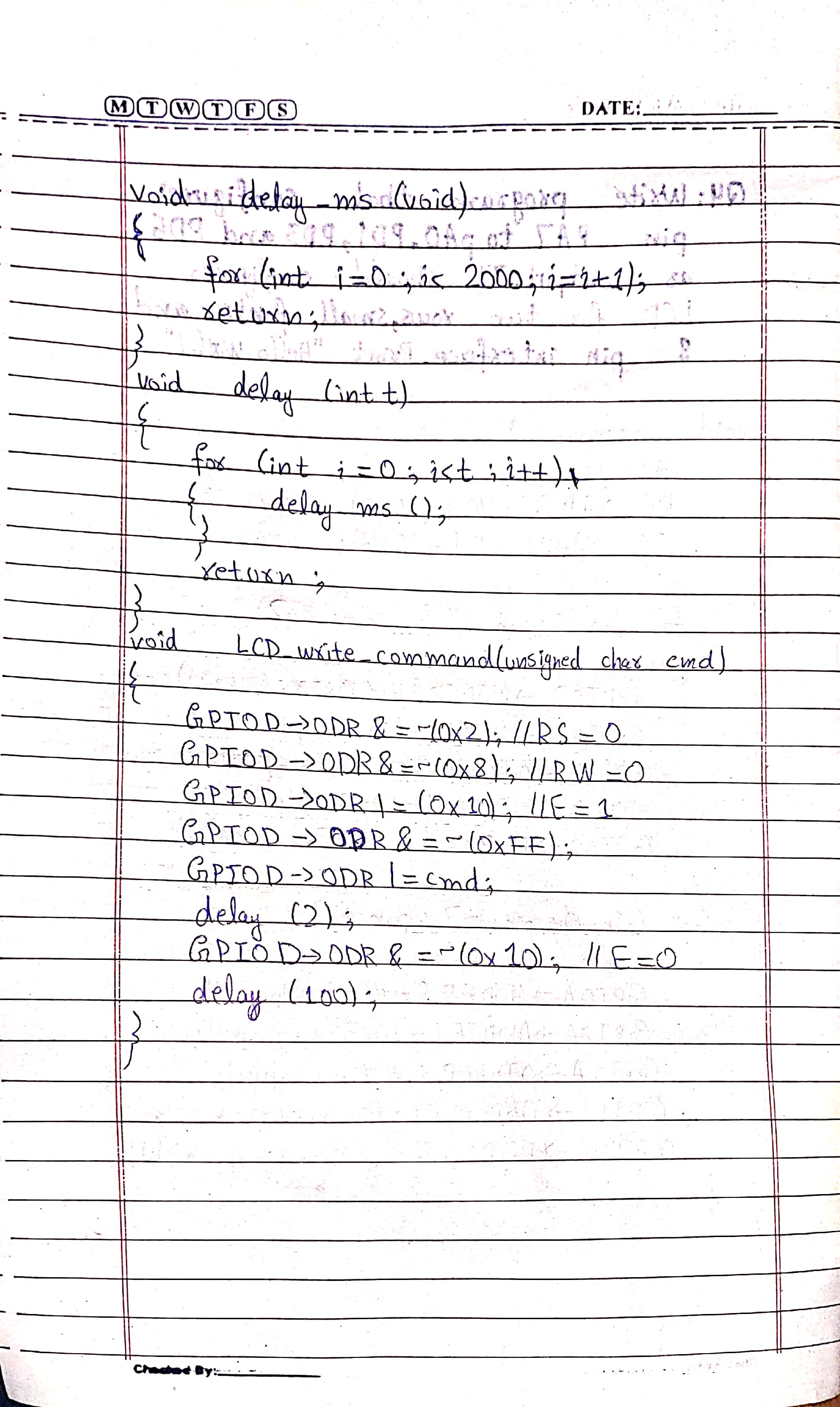
1. Switches SW1 and SW2 are connected on port E pin1 and pin 3 respectively. LED D1 is connected on port B pin 2. Write a program that configures PE1 and PE3 as input and PB2 as output. Write a program that implements AND functionality between SW1 and SW3 i.e., D1 is on only if PE1 and PE3 are pressed, otherwise D1 stays OFF

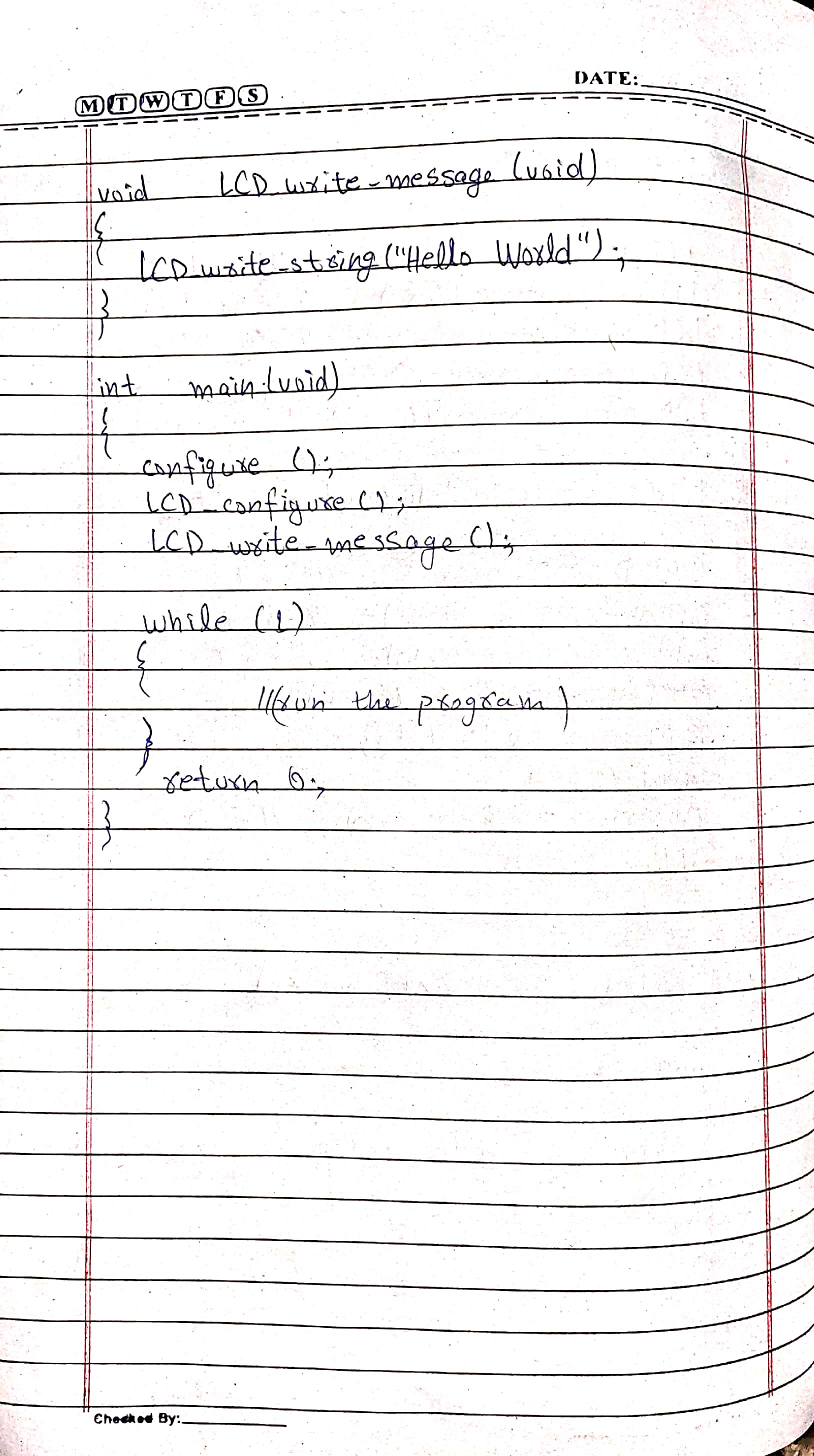
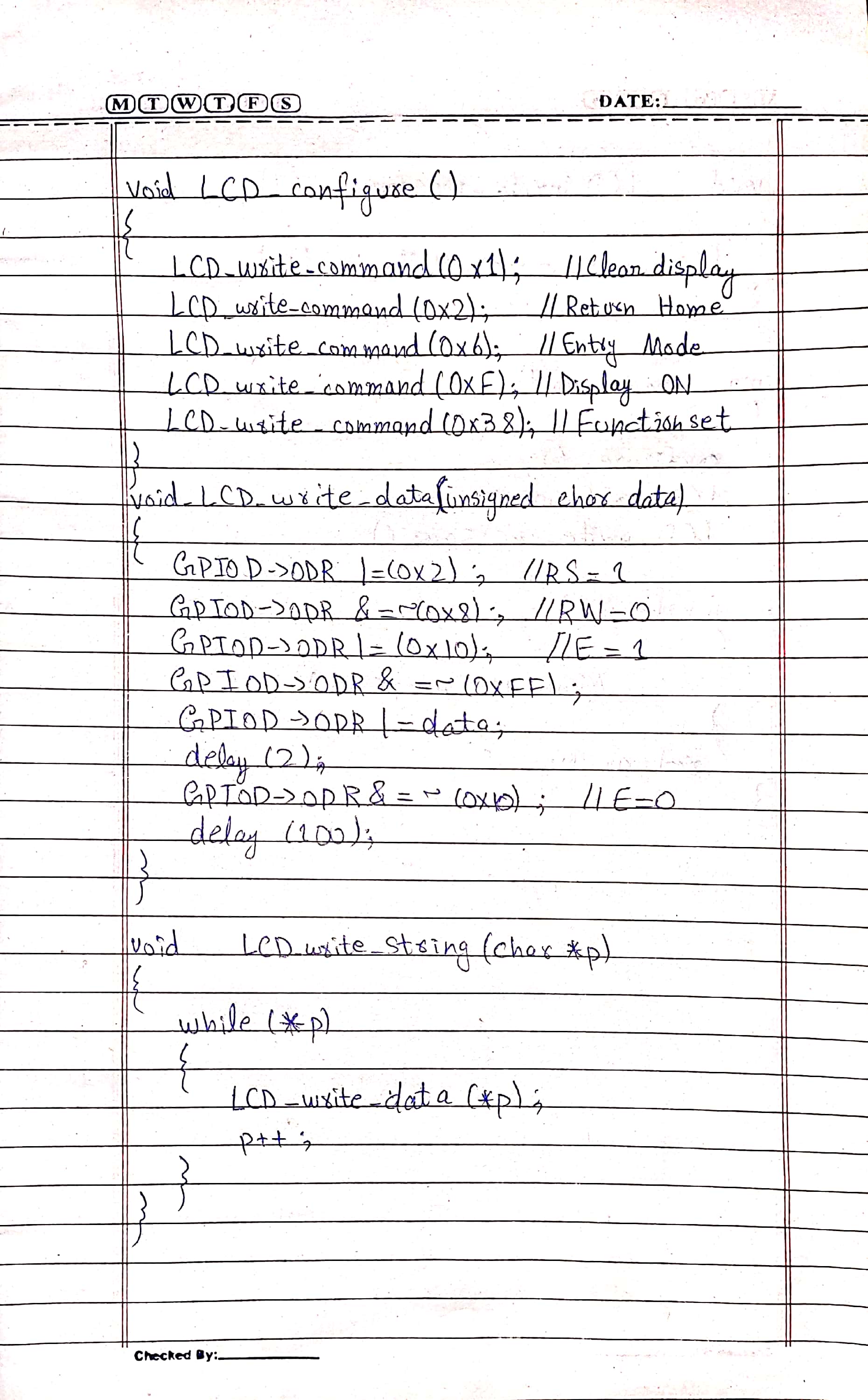




1. LCD 1602 is connected to port A with data pin D7-D0 connected to PA7-PA0 respectively. The control pins RS, RW and E are connected to port D pin PD1, PD3 and PD5 respectively. Write program that configures pin PA7 to PA0, PD1, PD3 and PD5 as output. The program should configure LCD for 2 rows, small fonts with 8 pin interface. The configure LCD and print ‘Hello world’.







1. Consider SW1 connected to PC1 and SW2 connected to PC3. LED D1 is connected to PB3 and LED D2 is connected to PB5. Configure SW1, SW2 as input and D1 and D2 as output. Write a program that turns On D1 when SW1 is pressed. The program should turn off D2 when SW2 is pressed. SW2 remains ON when SW2 is depressed.