

Effective use of GPIO pins as
Debugging pins

Mehul Mehta

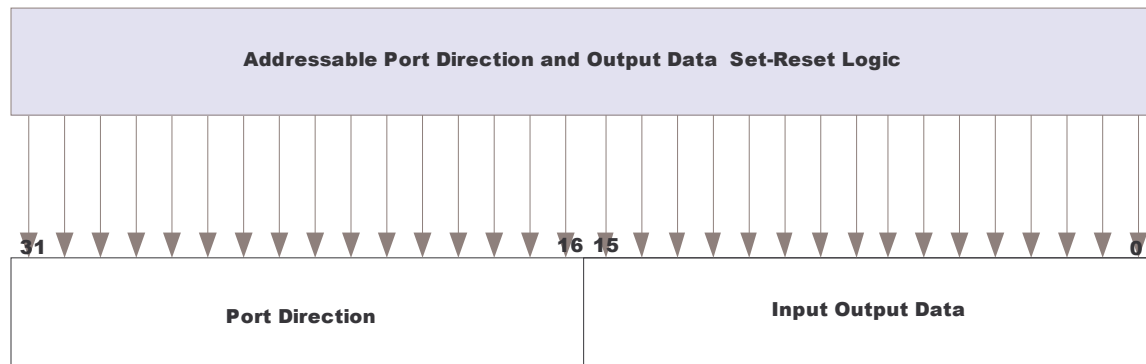


Effective use of GPIO pins as Debugging pins

Generally, most of the chip contains GPIO [General Purpose Input Output] pins for I/O purpose. These pins can be used for different purposes based on different requirements by configuring them as INPUT or OUTPUT.

BASIC CONCEPT OF GPIO :

As explained above, GPIO pins are I/O pins that can be configured by a **GPIO_CONTROL_REGISTER**. This configuration is common as mentioned below:



GPIO CONTROL REGISTER FOR 16 GPIO PINS

The GPIO register can be used to

1. Define whether a port pin is an input or an output.
2. Define the data sent to an output configured port.

Programming the GPIO register is done 2 bits at a time. Considering 16 GPIO pins, we will have 32 bit GPIO register. This register will contain 16 Port direction bits and 16 Input/Output data bits as shown above. A “one” on bits 31 through 16 will configure the appropriate port pin as an output. To set the least significant GPIO as input, bit 16 must be set to zero. To drive the next to least significant GPIO as an output with data set to a value of zero, bit 17 must be set to a “one” with bit 1 a “zero”, and so on.

HOW TO USE GPIO PINS AS DEBUGGING PINS ON HARDWARE FOR BRINGUP ACTIVITIES ?

GPIO pins can be used at any time in test execution to display any numeric debugging/status message. This way we can debug any issue on the hardware by providing different debugging message at different stage of test. Though these numeric debugging messages won't be useful for complex test on chip for some application, but still it helps a lot for the activities on bring up board.

```

// include section-----
//
//
// Delay routine
void delay(unsigned int loops) {
    while (loops != 0) {
        loops--;
    }
}

// Write routine to write some value at given addr.
void write(unsigned int addr, unsigned int value) {
    unsigned int *reg_addr;
    reg_addr = (unsigned int *) addr;
    *reg_addr = value;
}

// Read/Compare routine to read and compare the read data with expected value
void read_cmp(unsigned int expected, unsigned int addr) {
    unsigned int word;
    unsigned int *reg_addr;
    reg_addr = (unsigned int *) addr;
    word = *reg_addr;
    if (expected != word) {
        // Expected and read data is not correct display DEAD on GPIO
        *((unsigned int*)(GPIO_CONTROL_REGISTER)) = 0xFFFFDEAD;
    }
    else {
        // Expected and read data is not correct display B000 on GPIO
        *((unsigned int*)(GPIO_CONTROL_REGISTER)) = 0xFFFFB000;
    }
    delay(1000);
}

// Display routine to display input data on GPIO
void display(unsigned int data) {
    unsigned int *gpio;
    gpio = (unsigned int*)(GPIO_CONTROL_REGISTER);
    *gpio = (0xFFFF0000 | data);
    delay(1000);
}

```

```

//MAIN routine
//It write and reads diff. data at CONFIG_REG_ADDR and also compares the
read back data with expected data.
// Displays START_OF_TEST, END_OF_TEST and TEST_NUMBERS

int main(void) {
    unsigned int test_number ;           // TEST_NUMBER
    test_number = 0xBABE;                //START_OF_TEST = BABE
    display(test_number);

    test_number = 0;
    write(CONFIG_REG_ADDR, 0x55);
    read_cmp(0x55, CONFIG_REG_ADDR);
    test_number++;
    display(test_number);

    write(CONFIG_REG_ADDR, 0xaa);
    read_cmp(0xaa, CONFIG_REG_ADDR);
    test_number++;
    display(test_number);

    write(CONFIG_REG_ADDR, 0x00);
    read_cmp(0x00, CONFIG_REG_ADDR);
    test_number++;
    display(test_number);

    write(CONFIG_REG_ADDR, 0xff);
    read_cmp(0xff, CONFIG_REG_ADDR);
    test_number++;
    display(test_number);

    while(1) {
        test_number = CAFE;             // END_OF_TEST = CAFÉ BABE
        display(test_number);
        test_number = BABE;
        display(test_number);
    }
}

```

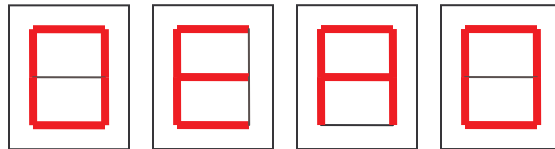
As shown in the dummy code above, code does write and read operation on some CONFIG_REG. It compares the read value with expected data and if expected data is not

matching it will display “**DEAD**”, else it will display “**B000**” on GPIO pins. At the same time there are 4 write and read operations, so it will also display **TEST_COUNT** at each write/read_cmp operation. It will also give you **START_OF_TEST** and **END_OF_TEST** message.

HARDWARE SETUP :

Generally, on bring up board we use a software tool to program/burn the flash data on flash memory using JTAG port. Using the same JTAG port we can scan the GPIO pins or GPIO Register data every time to get the GPIO numeric message.

We can use some displays like 7-segment which displays different nibbles of GPIO data as shown below :



16-BIT GPIO DISPLAY 1-SEGMENT PER NIBBLE

There are some tools available like **UNIVERSAL SCAN TOOL** which provides FLASH programming through JTAG and it also gives pin scanning facility with above display arrangements based on our requirement for scanning different pins of the chip. We can also display FLASH ADDR and FLASH DATA during execution.

PRECAUTIONS WHILE USING GPIO AS DEBUGGING PINS :

In order to use GPIO pins as debugging pins, we should take care of following:

- 1) Make sure whatever GPIO pins are used as debugging message display are not being used for any other purpose.
- 2) GPIO pins being bi-directional have to be configured as OUTPUT pins when used to display any debugging message.
- 3) There should be proper amount of DELAY in between two messages to visualize each display message.

OTHER APPLICATIONS OF GPIO :

Since GPIO are configurable I/O pins, we can use them to develop some small and easy interfaces like I2C, UART etc. All that is needed is the software programming for the pins to be used for new interface as per the specification of the interface.

For example, we can make any two GPIO pins as bi-directional SCL and SDA pins of I2C interface and using the software, I2C could be used to configure Slave devices. This will be very useful as a work-around when actual I2C interface on chip is not working due to some bug for bring up activities.