Exploring the Use of GPIO Ports and Watchdog Timer on TMS320F28379D DSP for Developing a Counter.

Juan Camilo García Perez. [*] **Luis Felipe Garzón Camacho.** [*]

[*] *Department of Electronic Engineering, Santo Tomás University, Bogotá, Colombia (e-mail: { juancgarciap, luisgarzonc} @usantotomas.edu.co)*

Abstract: The TMS320F28379D is a high-performance digital signal processor (DSP) with extensive peripheral integration, including General-Purpose Input/Output (GPIO) ports and a Watchdog Timer (WDT). In this paper, we explore the use of these peripherals in DSP-based systems. The GPIO ports provide a simple and efficient way to interface the DSP with various peripheral devices, such as sensors, actuators, or other microcontrollers. The WDT is designed to prevent system crashes caused by software errors or other unexpected events, thereby improving the overall reliability of the system. We discuss the configuration and use of these peripherals in detail and provide code examples to demonstrate their functionality. Finally, we present a counter to evaluate the effectiveness of the GPIO ports and WDT in a real-world application. The information presented in this paper is intended to help developers design robust and efficient DSP-based systems that meet their specific application requirements.

Keywords: GPIO ports, Watchdog Timer (WDT), TMS320F28379D DSP, Code composer, C.

1. INTRODUCTION

The TMS320F28379D is a high-performance digital signal processor (DSP) from Texas Instruments that provides extensive peripheral integration to support a wide range of applications. One of the most important aspects of using a DSP is interfacing it with the outside world, and this is where the General-Purpose Input/Output (GPIO) ports and the Watchdog Timer (WDT) come into play.

The GPIO ports are a set of general-purpose pins that can be configured to either input or output digital signals. These ports provide a simple and efficient way to interface the DSP with various peripheral devices, such as sensors, actuators, or other microcontrollers. The TMS320F28379D has a total of 175 GPIO pins, which can be grouped into different ports depending on their functionality.

In addition to the GPIO ports, the TMS320F28379D also includes a Watchdog Timer (WDT) that is designed to prevent system crashes caused by software errors or other unexpected events. The WDT works by resetting the system if a predefined time period elapses without a specific software action being performed. This provides an additional layer of protection against system failures and can help improve the overall reliability of the DSP-based system.

Overall, the GPIO ports and the Watchdog Timer are critical components of any TMS320F28379D-based system, as they provide a means to interface with the outside world and ensure reliable system operation. Understanding how to use these peripherals effectively can help developers design robust and efficient systems that meet their specific application requirements.

1.1 Main objective:

To demonstrate the use of the GPIO ports and Watchdog Timer (WDT) on a TMS320F28379D DSP for developing robust and efficient counter.

1.2 Specific objective:

- To explain the configuration and use of GPIO ports in interfacing the DSP.
- To describe the function and operation of the WDT in preventing system crashes caused by software errors or unexpected events.
- To provide a code written in C using the DSP's driver lib.

2. METHODOLOGY

2.1 General Purpose Input/Outpu

This section describes how a counter is powered by a DSP, the first stage of this process was to configure a circuit of how the counter is visualized (Fig. 1.). To design this circuit is asked the counter must be ascending from 0 to 15 and reset the count once it overflows, the count should be done every 2 seconds, and the current for each LED must be limited to $5mA$ by means of the resistor. So, to configure the R value. It's know the $V_o = 3.3V$, so, using Ohm's law:

$$R = \frac{V}{I} = \frac{3.3V}{5mA} = 660\Omega \quad (1)$$

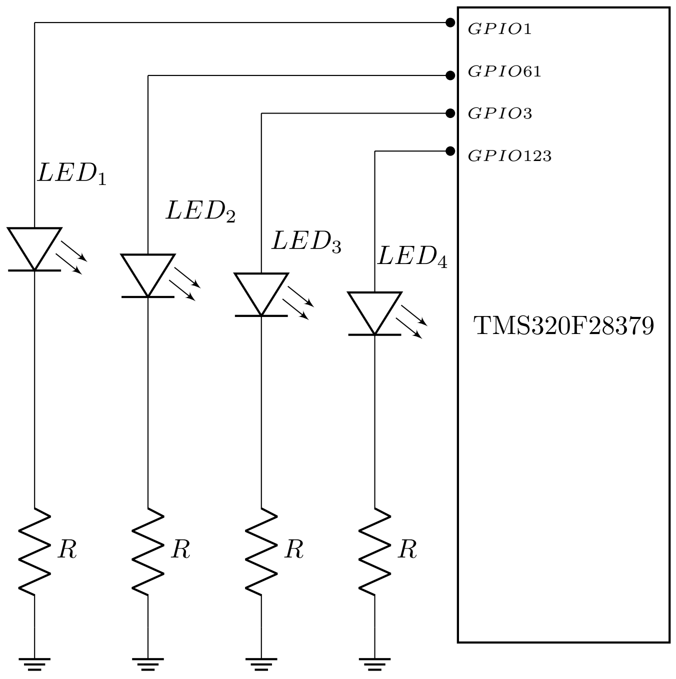


Fig. 1. DSP's counter circuit.

Once the hardware was set, the counter function had to be made. Two functions were disposed to write the counter. The firs one was called $SHOW()$ and the other one $COUNTER()$. The leds were defined from $LEDB0 - LEDB3$ in order to create a struct and then uniting it witt an $Uint32$ Struct called "ALL".

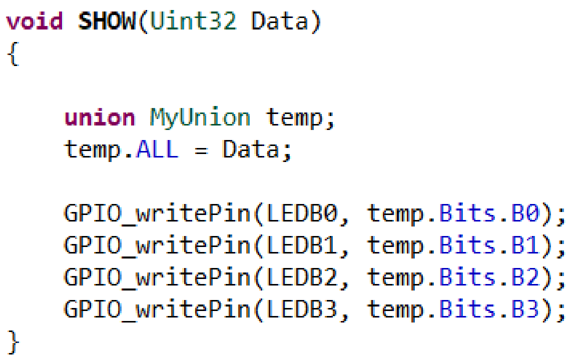

```
void SHOW(Uint32 Data)
{

    union MyUnion temp;
    temp.ALL = Data;

    GPIO_writePin(LEDB0, temp.Bits.B0);
    GPIO_writePin(LEDB1, temp.Bits.B1);
    GPIO_writePin(LEDB2, temp.Bits.B2);
    GPIO_writePin(LEDB3, temp.Bits.B3);
}
```

Fig. 2. SHOW() Function.

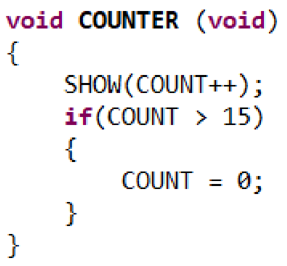

```
void COUNTER (void)
{
    SHOW(COUNT++);
    if(COUNT > 15)
    {
        COUNT = 0;
    }
}
```

Fig. 3. COUNTER() Function.

Here all the data is load into a temporary variable called "temp" and then all that data will be send for each DSP's pin there's a driverlib function called "$GPIO_writePin()$" using it, it is possible to provide a direction to each defined pins and then load the value (Fig. 2.). Once the $SHOW()$ function is set, the counter is written (Fig. 3.).

2.2 Watchdog Timer

Once was proved this code works, we can continue to the second part of the paper, the watchdog timer, the DSP has the following specifications:

$$\text{DSPF} = 10\text{Mhz}$$

$$\text{OSCCLK} = 512$$

$$\text{WDPS} = [1, 2, 4, 8, 16, 32, 64]$$

$$\text{WDW} = 210$$

DSPF is the DSP frequency, OSCCLK is the oscillator clock, WDPS is the watchdog timer prescaler (it's set 64) and WDW is a proposed window in order to verify the WD behavior. So, in order to have the watchdog clock value (EQ. 2.).

$$WDCLK = \frac{DSPF * WDW}{OSCCLK * WDPS} = \frac{(10e6) * 210)}{(512 * 64)} \tag{2}$$

Having the $WDCLK$ value, it's possibly to find the following values:

$$\text{WDT} = 1/\text{WDCLK}$$

$$\text{WDTF} = \text{WDT*RESETKEY}$$

$$\text{MULT} = 2/\text{WDTF}$$

$$\text{WDTF1} = \text{WDTF*1000000}$$

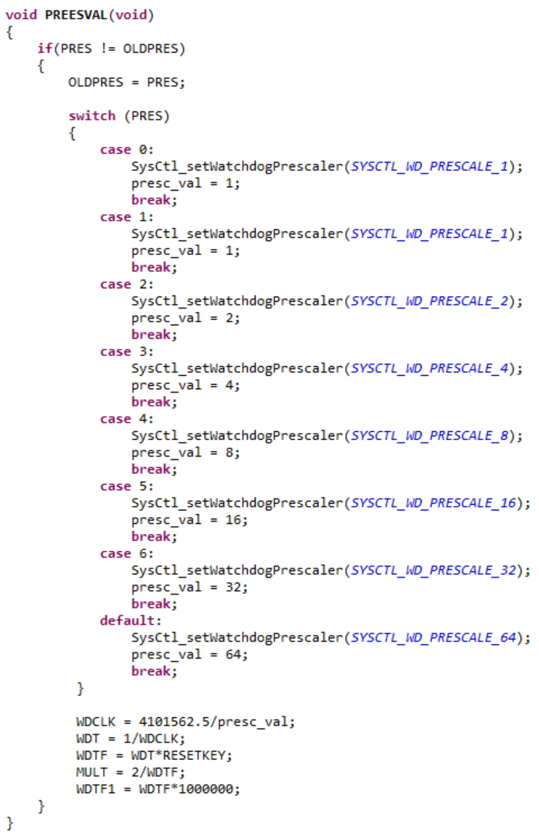
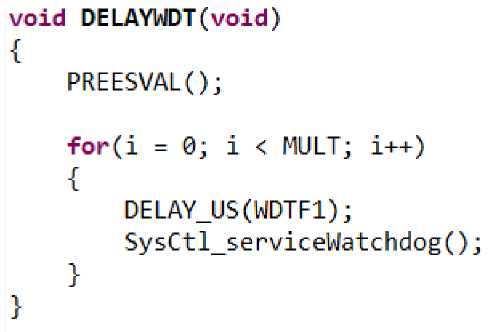

```
void PREESVAL(void)
{
    if(PRES != OLDPRES)
    {
        OLDPRES = PRES;

        switch (PRES)
        {
            case 0:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_1);
                presc_val = 1;
                break;
            case 1:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_1);
                presc_val = 1;
                break;
            case 2:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_2);
                presc_val = 2;
                break;
            case 3:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_4);
                presc_val = 4;
                break;
            case 4:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_8);
                presc_val = 8;
                break;
            case 5:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_16);
                presc_val = 16;
                break;
            case 6:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_32);
                presc_val = 32;
                break;
            default:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_64);
                presc_val = 64;
                break;
        }

        WDCLK = 4101562.5/presc_val;
        WDT = 1/WDCLK;
        WDTF = WDT*RESETKEY;
        MULT = 2/WDTF;
        WDTF1 = WDTF*1000000;
    }
}
```

Fig. 4. PREESVAL() Function.

Where WDT is the watchdog timer time, WDTF is the final watchdog timer time mulplied by a RESETKEY (fpr this excercise was proposed 180), MULT which is a multiplier, and WDTF1 is WDTF but in microseconds. So it's written another function called $PREESVAL()$ in order to make the code able to differentiate the prescaler value and proceed to perform EQ. 2.

In Fig. 3. It's possible to observe the code's main process flow where the first step is verifying the prescaler value in the PREES variable, and then decide what prescaler set according to the prescaler defined, that's why is used the driverlib function: $SysCtl\_setWatchdogPrescaler()$ and then set the specific case of prescaler saving its value in a new variable called "$presc\_val$" to finally sending it and performing the EQ. 2. (Fig. 4.).

At this moment was necessary to create a function that allow us to create a delay in order to set the watchdog timer while the counter loop was being executed we called another driverlib function $SysCtl\_serviceWatchdog()$ which is set to reset.



```
void DELAYWDT(void)
{
    PREESVAL();

    for(i = 0; i < MULT; i++)
    {
        DELAY_US(WDTF1);
        SysCtl_serviceWatchdog();
    }
}
```

Fig. 5. DELAYWDT() Function.

Fig. 5. can be sumarized as the calling of the prescaler value using the $PREESVAL()$. function, then create a loop where the program is going to work if it's minor to the MULT variable explained before, and the delay time is set in $WDTF1 = 712\mu s$.

3. CONCLUSION

The paper describes the process of configuring a circuit for a counter powered by a DSP, to count from 0 to 15, resetting the count once it overflows, and display the count every 2 seconds. The current for each LED is limited to $5mA$ by means of a resistor, which was calculated using Ohm's law. The paper also describes the functions written for the counter, including the $SHOW()$ and $COUNTER()$ functions, and how the data is loaded into a temporary variable and sent to each DSP pin using the $GPIO_writePin()$ function. Once the counter function was confirmed to be working, the paper goes on to describe the process of setting up a watchdog timer for the DSP, including specifications for the DSPF, OSCCLK, WDPS, and WDW vriables. The paper provides equations (EQ. 2.) for calculating the watchdog clock value and describes the importance of setting a proper window in order to verify the watchdog behavior. Overall, the paper provides a thorough explanation of the methodology used to configure the circuit and set up the counter and watchdog timer for the DSP.

REFERENCES

[1] TMS320F2837xD Dual-Core Microcontrollers Technical Reference Manual.

[2] F2837xD Firmware Development Package. USER'S GUIDE.

[3] DSP2839D Plans. Texas Instruments.

```c
#include "F28x_Project.h"
#include "driverlib.h"
#include "Device.h"
#include "math.h"

//*****************************************************************************

#define LEDB0           1
#define LEDB1           61
#define LEDB2           3
#define LEDB3           123

//*****************************************************************************

Uint32      COUNT;
Uint16      PRES, OLDPRES, i, RESETKEY = 180;
float       WDCLK, MULT, WDT, WDTF, WDTF1;
uint16_t    presc_val;

//*****************************************************************************


struct MyBits
{
    Uint16 B0:1;
    Uint16 B1:1;
    Uint16 B2:1;
    Uint16 B3:1;
    Uint16 B4:1;
    Uint16 B5:1;
    Uint16 B6:1;
    Uint16 B7:1;

};

union MyUnion                       // 2 different names for the same memory address
{
    Uint32 ALL;
    struct MyBits Bits;
};


//*****************************************************************************
void Config_GPIO_Driverlib(void);
void SHOW(Uint32 Data);
void PREESVAL(void);
void COUNTER (void);
void DELAYWDT(void);

void main (void)
{
//    InitSysCtrl();
    Device_init();
    SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_64);
    SysCtl_setWatchdogMode(SYSCTL_WD_MODE_RESET);
    SysCtl_serviceWatchdog();
    SysCtl_setWatchdogWindowValue(RESETKEY);
    SysCtl_enableWatchdogReset();

    PRES = 7;
    OLDPRES = 0;
    COUNT = 0;

    Config_GPIO_Driverlib();

    while(1)
    {
       COUNTER();
       DELAYWDT();
    }

}


//*****************************************************************************

void COUNTER (void)
{
    SHOW(COUNT++);
    if(COUNT > 15)
    {
        COUNT = 0;
    }
}


void Config_GPIO_Driverlib(void)
{
    GPIO_setPinConfig(GPIO_1_GPIO1);
    GPIO_setPinConfig(GPIO_61_GPIO61);
    GPIO_setPinConfig(GPIO_3_GPIO3);
    GPIO_setPinConfig(GPIO_123_GPIO123);

    GPIO_setDirectionMode(LEDB0,GPIO_DIR_MODE_OUT);
    GPIO_setDirectionMode(LEDB1,GPIO_DIR_MODE_OUT);
    GPIO_setDirectionMode(LEDB2,GPIO_DIR_MODE_OUT);
    GPIO_setDirectionMode(LEDB3,GPIO_DIR_MODE_OUT);

}

void SHOW(Uint32 Data)
{

    union MyUnion temp;
    temp.ALL = Data;                            // Passing and loading data in temp

    GPIO_writePin(LEDB0, temp.Bits.B0);
    GPIO_writePin(LEDB1, temp.Bits.B1);
    GPIO_writePin(LEDB2, temp.Bits.B2);
    GPIO_writePin(LEDB3, temp.Bits.B3);
}
//

void PREESVAL(void)
```

```c
{
    if(PRES != OLDPRES)
    {
        OLDPRES = PRES;
        //SysCtl_setWatchdogPrescaler((SYSCTL_WDPrescaler)PRES);
        switch (PRES)
        {
            case 0:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_1);
                presc_val = 1;
                break;
            case 1:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_1);
                presc_val = 1;
                break;
            case 2:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_2);
                presc_val = 2;
                break;
            case 3:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_4);
                presc_val = 4;
                break;
            case 4:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_8);
                presc_val = 8;
                break;
            case 5:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_16);
                presc_val = 16;
                break;
            case 6:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_32);
                presc_val = 32;
                break;
            default:
                SysCtl_setWatchdogPrescaler(SYSCTL_WD_PRESCALE_64);
                presc_val = 64;
                break;
         }
         //WDCLK = ((10e6)*210)/(512*64);
         WDCLK = 4101562.5/presc_val;
         WDT = 1/WDCLK;
         WDTF = WDT*RESETKEY;
         MULT = 2/WDTF;
         WDTF1 = WDTF*1000000;
    }
}


void DELAYWDT(void)
{
    PREESVAL();

    for(i = 0; i < MULT; i++)
    {
        DELAY_US(WDTF1);
        SysCtl_serviceWatchdog();
    }
}

```