

Instituto Tecnológico de Buenos Aires

22.99 LABORATORIO DE MICROPROCESADORES

Guía N°2: Introducción a Kinetis

Grupo 1

ÁLVAREZ, Lisandro	57771
GONZÁLEZ ORLANDO, Tomás Agustín	57090
PARRA, Rocío	57669
REINA KIPERMAN, Gonzalo Julián	56102

Profesores

JACOBY, Daniel Andrés
MAGLIOLA, Nicolás
ISMIRLIAN, Diego Matías

Presentado: 21/08/2019

Índice

1. Abreviaciones	2
2. Ejercicio 1	2
2.1. Ítem a	2
2.2. Ítem b	2
2.3. Ítem c	2
2.4. Ítem d	2
2.5. Ítem e	3
3. Ejercicio 2	3
3.1. Ítem f	3
4. Ejercicio 3	3
5. Ejercicio 4	4
5.1. ítem a	4
5.2. ítem b	5
6. Ejercicio 5	5
6.1. ítem a	5
6.2. ítem b	5
7. Ejercicio 6	6

1. Abreviaciones

Incluimos la siguientes abreviaciones para agilizar la lectura del informe:

- KRM: Kinetis K64 SubFamily Reference Manual.
- KD: Kinetis K64F SubFamily Datasheet.

2. Ejercicio 1

2.1. Ítem a

El puerto PTA12 se encuentra en el pin 42.

Este dato se encuentra en el KRM, en la sección 10.3.1, sección de pinout, tanto en las tablas de la sección 10.3.1. como en el diagrama de pinout (figura 10.5) del 100 LQFP.

A su vez, el mismo se encuentra en la KD, en la sección 5.1, sección de pinout.

2.2. Ítem b

Al inspeccionar la hoja de datos del microcontrolador, se encuentran cuántos y cuáles son los pines que tienen conversores AD. El fabricante nos proporciona una lista con las interfaces analógicas disponibles, es decir, que no están reservados para el microcontrolador. El microcontrolador cuenta con 11 pines disponibles que admiten entrada analógica, se muestran en la Figura 1

Module name	Pins	Recommendation if unused
ADC	ADC0_DP1, ADC0_DM1, ADC1_DP1, ADC1_DM1, ADC0_DP0/ADC1_DP3, ADC0_DM0/ADC1_DM3, ADC1_DP0/ ADC0_DP3, ADC1_DM0/ADC0_DM3, ADC1_SE16/ADC0_SE22, ADC0_SE16/ADC0_SE21, ADC1_SE18	Ground

Figura 1: Lista de interfaces ADC

2.3. Ítem c

En este modelo de MCU se encuentran 10 pines disponibles para el puerto PTE, los pines 1 a 7 y 31 a 33. Esto se obtiene tanto de la KD en la sección 5.1, como del KRM en la sección 10.3.1 y de la figura 10.5 ya mencionadas en el ítem a.

2.4. Ítem d

VDD es la tensión de alimentación digital utilizada, cuyo rango puede variar entre [-0.3; 3.8](V).

El rango para que la tensión de high VIH sea considerada como tal, depende de la tensión VDD, de manera tal que si $VDD \in [2.7; 3.6](V)$, entonces la tensión $VIH \in [0.7 \cdot VDD; V_{max}](V)$, donde V_{max} es la tensión máxima que admite el pin digital. Al mismo tiempo, si $VDD \in [1.7; 2.7](V)$, entonces $VIH \in [0.75 \cdot VDD; V_{max}](V)$.

En la sección 1.4 del KD, Voltage and current operating ratings, se encuentra el rango de entrada y salida digital para los pines de entrada y salida: [-0.3; 5.5](V) para aquellos pines que no son el

RESET, XTAL y EXTAL. Para estos últimos pines el rango es de [-0.3; VDD+0.3] (V). Es por esto que para los pines que no son el RESET, XTAL y EXTAL, los mismos aceptan una tensión de 5(V). Por otro lado, los pines RESET, XTAL y EXTAL NO aceptan una tensión de 5(V).

2.5. Ítem e

En la sección 1.4 de la KD, se observa que la máxima corriente que puede entregar un pin cualquiera es 25mA.

3. Ejercicio 2

A continuación se presenta una foto del microcontrolador corriendo el programa Blink.

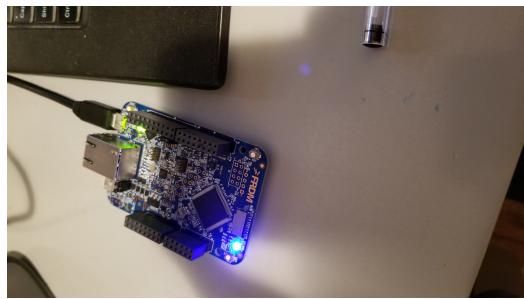


Figura 2: Corriendo el programa Blink

3.1. Ítem f

Se observa que al correr el programa con un nivel de optimización de Optimize Most, el compilador saltea las instrucciones que considera redundantes, en particular en el programa Blink, ignora las iteraciones de la función de delay ya que esta función comprende un while en el que únicamente se incrementa una variable que nunca se utiliza. De esta manera, al ignorar la línea de delay, el compilador estaría optimizando el programa al eliminar código redundante, que en nuestro caso particular afecta al funcionamiento general del mismo.

4. Ejercicio 3

A continuación se presenta una foto del microcontrolador corriendo el programa Blink con el LED correspondiente en verde con una frecuencia de 0.5Hz, como requería la consigna.

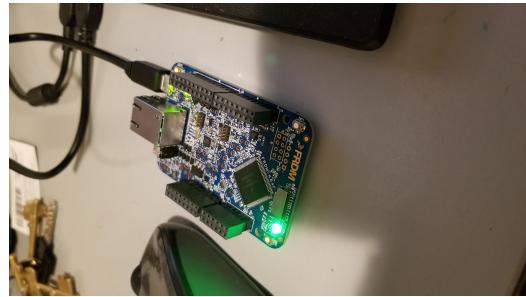


Figura 3: Corriendo el programa Blink

A fin de poner en evidencia la frecuencia a la que se produce el blink, se muestra a continuación una imagen del pin correspondiente:

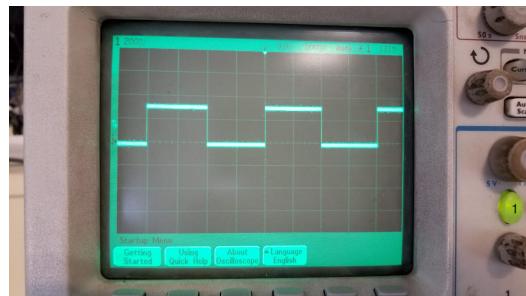


Figura 4: Corriendo el programa Blink con el LED en verde y a una frecuencia de 0.5Hz

5. Ejercicio 4

5.1. ítem a

El código utilizado se muestra a continuación. Se muestran solamente las definiciones de los pines y el código implementado para la consigna.

```
#define PIN_LED_GREEN    PORTNUM2PIN(PE,26) //PTE26
#define PIN_SW3           PORTNUM2PIN(PA,4)

int past_state;
int curr_state;
int button_pressed;

void App_Init (void){
    gpioMode(PIN_LED_GREEN, OUTPUT);
    gpioMode(PIN_SW3, INPUT);

    past_state = LOW;      //el led se prende en LOW
    curr_state = past_state;
    gpioWrite (PIN_LED_GREEN, past_state);
    button_pressed = false;
}
```

```

void App_Run (void){
    curr_state = !gpioRead(PIN_SW3);
    if( curr_state != past_state){
        if(button_pressed)
            button_pressed = !button_pressed;
        else
            gpioToggle(PIN_LED_GREEN);
            button_pressed = 1;
    }
    past_state = curr_state;
}

```

El código utilizado configuró el PIN del SW3 en modo INPUT (con el pullup por software desabilitado) y el PIN del LED en OUTPUT. Como era de esperarse el código funcionó correctamente.

5.2. ítem b

A continuación se pidió modificar el código del ítem anterior para controlar el LED con el pulsador SW2 incorporado en la placa. Según el diseño de la placa, en principio, el código no debería cumplir correctamente con la función solicitada. Esto se debe a que en una primera inspección se determinó que el pulsador no cuenta con una resistencia de pullup, con lo cual al configurar el PIN de entrada del SW2 como input sin el pullup por software habilitado, el PIN debería quedar 'flotando' al no ser pulsado, y causar un estado indeterminado del pulsador.

Sin embargo, al correr el código, el programa siguió funcionando correctamente. Una inspección mas minuciosa determinó que el PIN de entrada del SW2 estaba también conectado al terminal INT1 del acelerómetro incorporado en la placa. El terminal INT1 del acelerómetro es del tipo push-pull, y hace las veces de pullup al terminal de entrada del pulsador SW2, permitiendo que el programa funcione correctamente.

Se midió la 'resistencia' de pullup que impone el terminal INT1 del acelerómetro, resultado en un valor de 90Ω , lo cuál es un valor inesperadamente bajo, ya que al pulsar el pulsador SW2, circula una corriente elevada por la placa.

6. Ejercicio 5

6.1. ítem a

En el siguiente ejercicio se nos solicitó implementar el mismo programa que en el ejercicio 4, a diferencia que los circuitos correspondientes al pulsador y al LED se debieron implementar de forma externa, es decir, fuera de la placa provista. Para esto se implementaron los circuitos ilustrados en la Figura 6.1.

La entrada del pulsador se configuró en el PIN PTC9, mientras que la salida del LED se configuró en el PIN PTB23. Se reutilizó el código implementado en el ejercicio 4 con la configuración de pines mencionada. El programa funcionó correctamente, como era de esperarse.

6.2. ítem b

En la segunda parte del ejercicio se nos solicitó cambiar los pines del LED y el pulsador implementados en el ítem a. El pulsador se configuró en el pin PTC0 y el LED en el pin PTA0. El pin PTA0 no figura en el esquemático de la placa, por lo tanto, hubo que identificarlo en otro documento, para

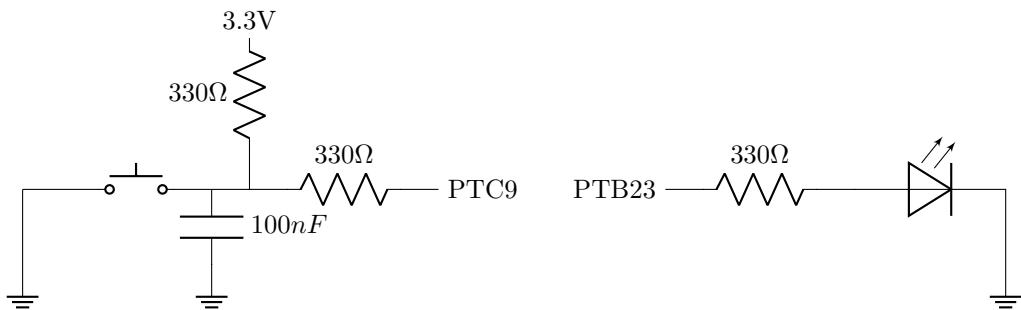


Figura 5: Circuito de pulsador y LED externos

luego encontrar la etiqueta del pin en el esquemático. El problema no presento mayor dificultad y el programa funcionó correctamente.

7. Ejercicio 6

Para el último problema, se implementaron tanto el LED que hace las veces de baliza, como el pulsador, de forma externa al igual que en el ejercicio 5. El código implementado fue el siguiente:

```
#define PIN_LED_EXT    PORTNUM2PIN(PC,12) // PTB23
#define PIN_LED_RED    PORTNUM2PIN(PB,22) // PTB22
#define PIN_SW_EXT     PORTNUM2PIN(PC,0)
static bool tooglear_baliza();

int past_state;
int curr_state;
bool button_pressed;
bool baliza_on;
void App_Init (void){
    gpioMode(PIN_LED_EXT, OUTPUT);
    gpioMode(PIN_LED_RED, OUTPUT);
    gpioMode(PIN_SW_EXT, INPUT);
    past_state = LOW;      //el led se prende en LOW
    curr_state = past_state;
    gpioWrite (PIN_LED_EXT, past_state);
    gpioWrite (PIN_LED_RED, HIGH);
    button_pressed = false;
    baliza_on = false;
}

void App_Run (void){
    int timer_count = 0;
    while(timer_count < 10){
        curr_state = !gpioRead(PIN_SW_EXT);
        if( curr_state != past_state){
            if(button_pressed)
                button_pressed = !button_pressed; //boton dejó de presionarse
            else if(tooglear_baliza())          //el botón recién se presionó
                break;
            past_state = curr_state;
        }
        delayLoop(1200000uL);
        timer_count++;
    }
}
```

```

    }

    if(baliza_on)
        gpioToggle(PIN_LED_EXT);
}

static bool tooglear_baliza(){
    bool apagada = false;
    baliza_on = !baliza_on;           //se tooglea la baliza
    gpioWrite(PIN_LED_RED, !baliza_on); //el led sigue a la baliza
    button_pressed = 1;
    if (!baliza_on){
        if(gpioRead(PIN_LED_EXT) == HIGH){
            gpioToggle(PIN_LED_EXT);
            apagada = true;
        }
    }
    return apagada;
}

```

En esencia, lo que se intenta hacer es implementar un loop que acumule delays de tiempo bloqueantes (es decir, durante el tiempo de delay no puedo leer input de botón, técnicamente), pero de muy corta duración, de forma que prácticamente no afecte el bloqueo de input del pulsador, o sea imperceptible. Una vez que acumulo una cantidad suficiente de delays, procedo a toglear la baliza. La complejidad del programa se centra en determinar el tiempo y la cantidad de delays que se ejecutan para no perder ningún input.