Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ingeniería Electrónica
Sistemas Digitales II

Trabajo Práctico Nº 1

Desarrollo de un controlador aplicando el modelo de MEF Statecharts UML Implementación en la placa FRDM-KL46Z

Autor/es:

Grupo Nº	
Nombre y Apellido	Nº de Legajo
Luciano Raffagnini	R-4419/9
Sebastian I. Rodríguez	R-4422/9

Corrigió	Calificación

2023

Índice

1. Introducción

Este trabajo práctico aplica los contenidos temáticos de la asignatura al desarrollo de un controlador implementado en un dispositivo de la familia KL46 de la placa de desarrollo FRDM-KL46Z. El funcionamiento del sistema se modela utilizando el formalismo de Máquina de Estado Finito / Statecharts UML y el código C debe reflejar el modelo propuesto. El desarrollo se apoya en las funciones de biblioteca provistas por el fabricante.

La aplicación se programará y depurará utilizando el ambiente MCUXpresso y las bibliotecas asociadas.

2. Objetivos

Objetivos cognitivos:

Se espera que los alumnos sean capaces de:

1. Especificar el comportamiento del sistema utilizando el modelo de Máquina de Estado Finito / Statecharts UML.
2. Aplicar los conocimientos adquiridos sobre la arquitectura de la familia de microcontroladores KL46 para desarrollar una aplicación basada en la placa FRDM-KL46Z.
3. Utilizar las funciones de biblioteca provistas por el fabricante para soportar el desarrollo de la aplicación software.
4. Aplicar el criterio de reutilización de código al definir la estructura del proyecto, realizando la implementación de las diferentes MEFs en archivos separados.

Objetivos actitudinales:

1. Promover el trabajo en equipo para obtener la solución a un problema.
2. Promover la habilidad de realizar una defensa de la solución propuesta para el problema planteado.
3. Promover la habilidad de elaborar un reporte escrito sobre el trabajo realizado.

3. Pautas para la entrega de material ligado a TP1

Material a entregar:

- El modelo completo de la solución del problema planteado. El mismo deberá ser claro y legible.
- El informe de las tareas realizadas en base a la plantilla oportunamente subida al campus.
- El código de la aplicación desarrollada.

Aspectos a tener en cuenta para la entrega:

- Se sugiere utilizar MEF jerárquicas. Debe quedar claro cuál es la MEF top o principal y cuales las MEF subordinadas en los superestados.
- Se debe indicar el tipo de transición que hay entre superestados (con historia o con reset) y especificar con claridad qué recursos garantizan que se realizarán en forma adecuada. Esto puede tener implicancias en el código y las mismas se deberán detallar explícitamente.
- Se debe explicar de qué modo se comunican las MEF entre sí y enumerar los recursos que permitirán plasmar dicha comunicación. Se sugiere el uso de funciones y no se recomienda el uso de variables globales para la interacción entre las MEFs.
- Si intervienen IRQs de periféricos, indicar cuáles y qué tareas se llevan adelante en sus rutinas de servicio.
- Si se definen funciones, explicar de qué tareas son responsables.
- Explicar dónde se ubican en el código las declaraciones de variables y funciones y donde se las invoca.
- Tener en cuenta las actividades que se solicitan en la plantilla e incluirlas en el informe.
- En la medida de lo posible, reutilizar funciones que se hayan definido en forma previa.
- Documentar el código de la aplicación.

4. Tareas desarrolladas

Modelado MEF

Para el modelado del problema, elegimos utilizar una MEF Jerarquica, compuesta por 3 superestados, a cada uno de los cuales, le corresponde una MEF que modela cada secuencia que identificamos en el problema:

- **MEF Habitual:** Ciclo principal de control de tráfico, donde la circulación por la ruta y el camino secundario se intercambia según intervalos regulares de tiempo.
- **MEF Cruce:** Situación en la cual la circulación por la ruta está habilitada y algun peaton pulsa el botón para habilitar el cruce
- **MEF Tráfico:** Situación en la cual se habilita la circulación por el camino secundario debido a una acumulación de tráfico.

Consideramos que esta es una buena solución ya que permite dividir el problema en 3 subproblemas separados y claramente definidos que son sencillos de manejar.

Otra opción, por ejemplo, habría sido utilizar una sola máquina de estado que contenga todos los estados de las 3 secuencias, sin embargo la implementación de dicha MEF sería más engorrosa a la hora de dividir y atacar el problema planteado.

Descripción general de la implementación de las MEF

Cada una de las MEF obedece una lógica común de funcionamiento en lo que respecta a sus funciones. Todas contienen métodos que comienzan con el nombre de la MEF seguido de un guión bajo y su funcionalidad.

Dichos métodos son:

+ init(): deja a la MEF en condiciones iniciales de funcionamiento.
+ run(): permite a la MEF evolucionar, cambiar de estado y realizar las acciones que correspondan.
+ periodicTask1ms(): debe ser llamado cada 1 ms, ejecuta todas las tareas de la MEF que requieran una temporización específica y/o resulte práctico que se realicen a una frecuencia determinada.

Por otro lado, cada una está implementada en archivos separados, en el formato indicado por la cátedra. De esta manera, el estado de la MEF junto a otras posibles variables de interés pueden mantenerse fuera del alcance del resto del programa, lo que reduce las posibilidades de cometer errores y logra “encapsular” el comportamiento de la MEF.

Temporizaciones

Para las temporizaciones que se deben realizar, se utilizará el timer SysTick, que lanzará una interrupción, en intervalos de 1 ms. Está interrupción invocará a la funcion SysTick_Handler(), que a su vez invocará a la función mefJerarquica_periodicTask1ms(), que proveerá todas las temporizaciones necesarias a sus MEFs anidadas.

Consideraciones con respecto al conteo de vehículos en el camino secundario

Para manejar el conteo de vehículos del camino secundario, implementamos en un archivo separado una serie de métodos:

+ actualizar_autos_en_espera(bool camino_secundario_habilitado): verifica si ocurrió un evento asociado a la llegada de un vehículo (SW3 fue presionado) y de haber ocurrido, incrementa o decrementa la cantidad de autos en espera según el valor de *camino_secundario_habilitado*. Si está en alto decrementa, si está en bajo incrementa.
+ get_autos_en_espera(): devuelve la cantidad de autos en espera sobre el camino secundario (int).
+ reset_autos_en_espera(): pone en cero la cantidad de autos en espera.

En los casos en que se habilita el camino secundario en la MEF Habitual y en la MEF de Cruce consideramos que todos los autos que pudieran haber estado esperando para cruzar, efectivamente cruzan (por lo que al deshabilitar el camino, la cantidad de autos se setea en cero).

Consideramos que esto es razonable debido a que necesariamente en estas situaciones tendremos a lo sumo 2 autos esperando; en caso de que fueran más, se hubiera realizado la transición hacia la MEF Tráfico, ya que se cumpliría la condición (*autos_en_espera >= 3*).

De está forma no se hace necesario implementar la funcionalidad de decrementar la cantidad de autos al presionar el SW3 mientras se está en las MEFs Habitual y de Cruce.

En la MEF Habitual, sin embargo. sí se implementará la funcionalidad de que, si la ruta está habilitada, al presionar SW3, se incrementa la cantidad de autos en espera. También se decrementará la cantidad de autos si sé está en la MEF Tráfico, con el camino secundario habilitado.

5. Equipamiento utilizado

- Computadora
- Placa de Desarrollo KL43Z
- Protoboard
- Leds Rojo y Verde
- Resistencias
- Cable USB - Mini USB

6. Resultados obtenidos

El programa funciona según lo representado en el diagrama de estados, realizando correctamente todas las acciones esperadas.

7. Conclusiones

Consideramos, que logramos desarrollar un código eficiente, simple, escalable y organizado. El aplicar buenas prácticas de programación permitió que cada parte del código tenga un propósito único y claramente definido. Además, debido a la división en archivos separados, cada archivo expone al exterior sólo las funciones o variables necesarias, permitiendo encapsular la complejidad de cada sección del programa.

Como reflexión final, opinamos que este trabajo nos permitió ejercitar el desglose de un problema en problemas más pequeños, la modelización a través de MEF y la utilización de buenas prácticas de programación.

8. Bibliografía

- Apuntes provistos por la Cátedra

TP2-DSII.c

```c
#include "mefJerarquica.h"

int main(void)
{
    //SystemCoreClock = 48MHz
    SysTick_Config(SystemCoreClock / 1000U);

    mefJerarquica_init();

    while(true)
    {
        mefJerarquica_run();
    }

    return 0;
}

void SysTick_Handler(void)
{
    mefJerarquica_periodicTask1ms();
}
```

mefJerarquica.c

```
#include "mefJerarquica.h"
#include "deteccion_trafico.h"

typedef enum
{
    MEF_HABITUAL = 0,
    MEF_CRUCE,
    MEF_TRAFICO
} estado_mefJerarquica;

static estado_mefJerarquica estado;
static bool no_se_ha_cortado;

void mefJerarquica_init(void)
{
    estado = MEF_HABITUAL;
    no_se_ha_cortado = true;
    board_init();
    key_init();
    reset_autos_en_espera();
    mefHabitual_init();
    mefCruce_init();
    mefTrafico_init();
}

int mefJerarquica_run(void)
{
    switch (estado)
    {

    case MEF_HABITUAL:
    {
        bool ruta_habilitada = mefHabitual_run();

        if (ruta_habilitada == 0)
        {
            // Considramos que todos los autos que estuvieran esperando cruzan
            // Si abandona el estado "RUTA_HABILITADA" puede interrumpirse para
cruce peatonal
            no_se_ha_cortado = true;
            reset_autos_en_espera();
        }
```

```
        if (key_getPressEv(BOARD_SW_ID_1) && ruta_habilitada &&
no_se_ha_cortado)
        {
            estado = MEF_CRUCE;
            no_se_ha_cortado = false;
        }

        if (get_autos_en_espera() >= 3 && ruta_habilitada)
        {
            estado = MEF_TRAFICO;
        }
    }
    break;

    case MEF_CRUCE:
    {
        bool salir = mefCruce_run();
        reset_autos_en_espera();

        if (salir)
        {
            mefCruce_init();
            key_getPressEv(BOARD_SW_ID_1);
            estado = MEF_HABITUAL;
        }
    }
    break;

    case MEF_TRAFICO:
    {
        bool salir = mefTrafico_run();

        if (salir)
        {
            mefHabitual_init();
            mefTrafico_init();
            key_getPressEv(BOARD_SW_ID_1);
            // Quito cualquier presionada de botón que se pudo haber efectuado
            estado = MEF_HABITUAL;
        }
    }
    break;

    default:
        break;
    }
```

```
    return 0;
}

void mefJerarquica_periodicTask1ms(void)
{
    key_periodicTask1ms();

    switch (estado)
    {
    case MEF_HABITUAL:
        mefHabitual_periodicTask1ms();
        break;

    case MEF_CRUCE:
        mefCruce_periodicTask1ms();
        break;

    case MEF_TRAFICO:
        mefTrafico_periodicTask1ms();
        break;


    default:
        break;
    }
}
```

mefHabitual.c

```
#include "mef_habitual.h"

typedef enum
{
    RUTA_HABILITADA = 0,
    RUTA_CORTANDO,
    SECUNDARIO_HABILITADO,
    SECUNDARIO_CORTANDO,
}
estado_mefHabitual;

//Temporizaciones en [ms]
static const unsigned int TIEMPO_RUTA_HABILITADA = 120000;
static const unsigned int TIEMPO_RUTA_CORTANDO = 5000;
static const unsigned int TIEMPO_SECUNDARIO_HABILITADO = 30000;
static const unsigned int TIEMPO_SECUNDARIO_CORTANDO = 5000;
static const unsigned int PERIODO_LVR = 200;
static const unsigned int PERIODO_LVS = 200;

static unsigned int tim_mefHabitual;
static unsigned int contador_titilar;
static estado_mefHabitual estado;

void mefHabitual_init(void)
{
    contador_titilar = 0;
    tim_mefHabitual = TIEMPO_RUTA_HABILITADA;
    estado = RUTA_HABILITADA;
    reset_autos_en_espera();
}

bool mefHabitual_run(void)
{
    // Devuelve verdadero si la ruta está habilitada, falso en cualquier otro
caso
    switch (estado)
    {

    case RUTA_HABILITADA:
        board_setLed(LVR, ON);
        board_setLed(LRS, ON);
        board_setLed(LRR, OFF);
        board_setLed(LVS, OFF);

        if (tim_mefHabitual <= 0)
```

```
        {
            estado = RUTA_CORTANDO;
            tim_mefHabitual = TIEMPO_RUTA_CORTANDO;
        }

        break;

    case RUTA_CORTANDO:
        board_setLed(LRS, ON);
        board_setLed(LRR, OFF);
        board_setLed(LVS, OFF);

        if (contador_titilar <= 0)
        {
            contador_titilar = PERIODO_LVR;
            board_setLed(LVR, TOGGLE);
        }

        if (tim_mefHabitual <= 0)
        {
            estado = SECUNDARIO_HABILITADO;
            tim_mefHabitual = TIEMPO_SECUNDARIO_HABILITADO;
        }

        break;

    case SECUNDARIO_HABILITADO:
        board_setLed(LVR, OFF);
        board_setLed(LRS, OFF);
        board_setLed(LRR, ON);
        board_setLed(LVS, ON);

        if (tim_mefHabitual <= 0)
        {
            estado = SECUNDARIO_CORTANDO;
            tim_mefHabitual = TIEMPO_SECUNDARIO_CORTANDO;
        }
        break;

    case SECUNDARIO_CORTANDO:
        board_setLed(LVR, OFF);
        board_setLed(LRS, OFF);
        board_setLed(LRR, ON);

        if (contador_titilar <= 0)
        {
```

```
            contador_titilar = PERIODO_LVS;
            board_setLed(LVS, TOGGLE);
        }

        if (tim_mefHabitual <= 0)
        {
            estado = RUTA_HABILITADA;
            tim_mefHabitual = TIEMPO_RUTA_HABILITADA;
        }
        break;

    default:
        break;
    }

    return (estado == RUTA_HABILITADA);
}

void mefHabitual_periodicTask1ms(void)
{
    if (tim_mefHabitual)
        tim_mefHabitual--;

    if (contador_titilar)
        contador_titilar--;

    actualizar_autos_en_espera(estado == RUTA_HABILITADA);
}
```

mefTrafico.c

```
#include "mef_trafico.h"

typedef enum
{
    AVISO_HABILITACION_CAMINO,
    CAMINO_HABILITADO,
    AVISO_CORTE_CAMINO
}
estado_mefTrafico;

//Temporizaciones en [ms]
static const unsigned int TIEMPO_AVISO_HABILITACION_CAMINO = 5000;
static const unsigned int TIEMPO_AVISO_CORTE_CAMINO = 5000;
static const unsigned int PERIODO_LVR = 200;
static const unsigned int PERIODO_LVS = 200;

static unsigned int tim_mefTrafico;
static unsigned int contador_titilar;
static estado_mefTrafico estado;

void mefTrafico_init(void)
{
    contador_titilar = 0;
    tim_mefTrafico = TIEMPO_AVISO_HABILITACION_CAMINO;
    estado = AVISO_HABILITACION_CAMINO;
}

bool mefTrafico_run()
{
    switch (estado)
    {
        case AVISO_HABILITACION_CAMINO:
            {
            board_setLed(LRS, ON);
            board_setLed(LRR, OFF);
            board_setLed(LVS, OFF);

            if (contador_titilar <= 0) {
                contador_titilar = PERIODO_LVR;
                board_setLed(LVR, TOGGLE);
            }

            if (tim_mefTrafico <= 0) {
                estado = CAMINO_HABILITADO;
            }
```

```
            }
            break;

        case CAMINO_HABILITADO:
            {
            board_setLed(LVR, OFF);
            board_setLed(LRS, OFF);
            board_setLed(LRR, ON);
            board_setLed(LVS, ON);

            if (get_autos_en_espera() <= 0) {
                estado = AVISO_CORTE_CAMINO;
                tim_mefTrafico = TIEMPO_AVISO_CORTE_CAMINO;
            }
            }
            break;

        case AVISO_CORTE_CAMINO:
            {
            board_setLed(LVR, OFF);
            board_setLed(LRS, OFF);
            board_setLed(LRR, ON);

            if (contador_titilar <= 0) {
                contador_titilar = PERIODO_LVS;
                board_setLed(LVS, TOGGLE);
            }
            if (tim_mefTrafico <= 0) { return true;}
            }
            break;

        default:
            break;
    }
    return false;
}

void mefTrafico_periodicTask1ms(void)
{
    if (tim_mefTrafico)
        tim_mefTrafico--;
    if (contador_titilar)
        contador_titilar--;

    actualizar_autos_en_espera(estado != CAMINO_HABILITADO);
}
```

mefCruce.c

```
#include "mef_cruce.h"
typedef enum
{
    AVISO_CORTE_CAMINO,
    CRUCE_HABILITADO,
    AVISO_HABILITACION_RUTA
}
estado_mefCruce;

//Temporizaciones en [ms]
static const unsigned int TIEMPO_AVISO_CORTE_RUTA = 10000;
static const unsigned int TIEMPO_CRUCE_HABILITADO = 60000;
static const unsigned int TIEMPO_AVISO_HABILITACION_RUTA = 10000;
static const unsigned int PERIODO_LVR = 200;
static const unsigned int PERIODO_LRR = 200;

static unsigned int tim_mefCruce;
static unsigned int contador_titilar;
static estado_mefCruce estado;

void mefCruce_init(void)
{
    contador_titilar = 0;
    tim_mefCruce = TIEMPO_AVISO_CORTE_RUTA;
    estado = AVISO_CORTE_CAMINO;
}

bool mefCruce_run(void)
{
    switch (estado)
    {
        case AVISO_CORTE_CAMINO:
            board_setLed(LRS, ON);
            board_setLed(LRR, OFF);
            board_setLed(LVS, OFF);

            if (contador_titilar <= 0)
            {
                contador_titilar = PERIODO_LVR;
                board_setLed(LVR, TOGGLE);
            }
            if (tim_mefCruce <= 0)
            {
                estado = CRUCE_HABILITADO;
                tim_mefCruce = TIEMPO_CRUCE_HABILITADO;
```

```
            }
            break;

        case CRUCE_HABILITADO:
            board_setLed(LVR, OFF);
            board_setLed(LRS, OFF);
            board_setLed(LRR, ON);
            board_setLed(LVS, ON);

            if (tim_mefCruce <= 0)
            {
                estado = AVISO_HABILITACION_RUTA;
                tim_mefCruce = TIEMPO_AVISO_HABILITACION_RUTA;
            }
            break;

        case AVISO_HABILITACION_RUTA:
            board_setLed(LVR, OFF);
            board_setLed(LRS, OFF);
            board_setLed(LVS, ON);

            if (contador_titilar <= 0)
            {
                contador_titilar = PERIODO_LRR;
                board_setLed(LRR, TOGGLE);
            }
            if (tim_mefCruce <= 0)
            {
                return true;
            }
            break;

        default:
            break;
    }
    return false;
}

void mefCruce_periodicTask1ms(void)
{
    if (tim_mefCruce)
        tim_mefCruce--;

    if (contador_titilar)
        contador_titilar--;
}
```

deteccionTrafico.c

```c
#include "deteccion_trafico.h"
#include "key.h"

static int autos_en_espera;

int actualizar_autos_en_espera(bool camino_secundario_deshabilitado) {
    if (key_getPressEv(BOARD_SW_ID_3)) {
        if (camino_secundario_deshabilitado) {
            autos_en_espera++;
        } else if (autos_en_espera)
            autos_en_espera--;
    }
    return autos_en_espera;
}

int get_autos_en_espera() {
    return autos_en_espera;
}

void reset_autos_en_espera() {
    key_getPressEv(BOARD_SW_ID_3);
    autos_en_espera = 0;
}
```