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**
Valentin Bellini	B-6127/1
Iván Saitta	S-5435/6

Corrigió	**Calificación**

Año 2024

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 Maquina 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:

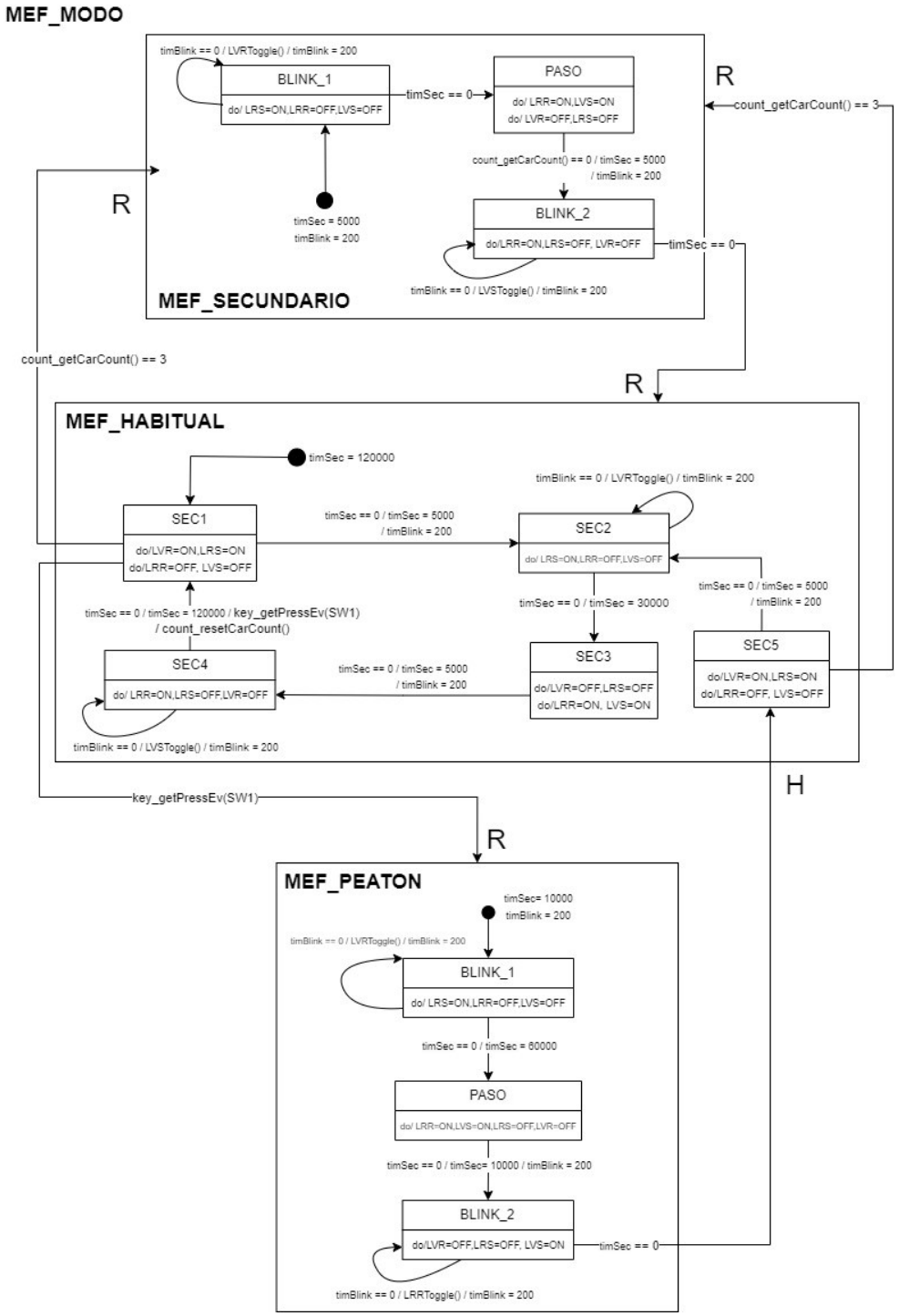
- El modelo completo de la solución del problema planteado. El mismo deberá ser claro y legible.
- El informe de las tareas realizadas en base a la plantilla oportunamente subida al campus.
- El código de la aplicación desarrollada.

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

La solución planteada a través de MEF jerárquica se presenta a continuación:



Como se puede observar, se plantea una MEF jerárquica con 3 MEFs subordinadas dentro representando a cada posibilidad de cruce con un estado asociado. La *mef_habitual* será el estado donde se encuentra el funcionamiento habitual del programa, con el semáforo funcionando de manera secuencial, la *mef_peaton* representará el estado donde los peatones son la preferencia y tienen cruce por sobre la ruta principal y por último la *mef_secundario* es un estado donde se habilita particularmente el camino secundario, en una configuración distinta a la del funcionamiento habitual. A continuación, se explicará el funcionamiento de cada MEF de manera resumida.

• **MEF habitual:** La Máquina de Estados Finitos (MEF) habitual está compuesta por 5 estados. Los estados del 1 al 4 están dispuestos secuencialmente en forma de anillo, y las únicas transiciones entre ellos se realizan mediante temporizadores. Por lo tanto, si el programa se deja en este funcionamiento, resultará en un ciclo que se repetirá con la configuración de luces de semáforo adecuada para cada estado. El estado 1 es el más relevante en esta MEF, ya que indica el inicio de la secuencia y habilita la ruta principal. Desde este estado pueden surgir dos situaciones: transicionar a la MEF peatonal si se presiona el interruptor SW1, o transicionar a la MEF secundaria si se acumulan 3 o más vehículos en la calle secundaria. El estado 5 se implementa para retornar desde la MEF peatonal a un estado, similar en funcionamiento al estado 1, pero que no permite realizar transiciones a la MEF peatonal nuevamente. La función de la MEF está definida como un tipo de enumeración, donde los valores de retorno posibles son "*TR_NONE*", "*TR_TO_PEATON*" y "*TR_TO_SECUNDARIO*". Cuando se evalúa la función, cada devolución provoca una transición desde esta MEF hacia la indicada.

• **MEF Peaton:** En este caso, nos enfrentamos a una implementación secuencial ininterrumpible. Esta MEF facilita el paso de los peatones y también del camino secundario durante el tiempo definido en las macros. La función que implementa esta MEF es de tipo booleano: devolverá *true* si ha finalizado el tiempo de ejecución y *false* en caso contrario.

• **MEF Secundario:** Similar a las anteriores, contamos con una función implementada de tipo booleano que devuelve *true* cuando se permite el regreso a la MEF Habitual. En esta MEF se ingresa cuando la cuenta de autos es igual o mayor a 3. Por lo tanto, será necesario presionar el pulsador al menos 3 veces para que la cuenta llegue a cero, indicando que todos los autos han cruzado, y así poder devolver *true* para volver al funcionamiento habitual.

• **MEF Modo:** Esta MEF es la MEF jerárquica, construida con 3 superestados descritos anteriormente, y se encarga de ejecutar las funciones en orden para mantener la lógica requerida. La función *mef_modo_init()* establece la cuenta de autos en cero e inicializa las sub-MEFs. La función *mef_modo_task1ms()* se encarga de ejecutar la tarea de interrupción cada 1 ms de la función adecuada según el estado en el que se encuentre, para así poder decrementar los temporizadores adecuados. Por último, la ejecución de la MEF propiamente dicha se encuentra en la función *mef_modo()*. La mejor manera de explicarla es viendo el código comentado:

```
void mef_modo(void){
   switch(estado_MEF_modo){
      case EST_MODO_HABITUAL:                          // Si estamos en modo habitual:
         switch(mef_habitual()){                       // Evaluamos mef_habitual (return tr_enum)
            case TR_TO_PEATON:                         // Si devuelve transición a peaton:
               estado_MEF_modo = EST_MODO_PEATON;      // Vamos a MEF peaton con reset.
               mef_peaton_reset();
            break;
            case TR_TO_SECUNDARIO:                     // Si devuelve transición a secundario:
               estado_MEF_modo = EST_MODO_SECUNDARIO;  // Vamos a MEF Secundario con reset.
               mef_secundario_reset();
            break;
            default:                                   // Por defecto devuelve sin transición.
               estado_MEF_modo = EST_MODO_HABITUAL;    // Por lo tanto se queda en mef habitual.
            break;
         }
      break;
      case EST_MODO_PEATON:                            // Si estamos en modo peaton:
         if(mef_peaton()){                             // Evaluamos mef_peaton. Si return True,
            estado_MEF_modo = EST_MODO_HABITUAL;       // es porque se terminó el tiempo de cruce
            mef_habitual_init_sec_5();                 // Init mef_habitual en "EST_SEC_5"
         }                                             // Transición con "Historia" se conservan
                                                       // los timers.
      break;
      case EST_MODO_SECUNDARIO:                        // Si estamos en modo secundario:
         if(mef_secundario()){                         // Evaluamos mef_secundario. True si la
                                                       // la cuenta quedó en cero.
            estado_MEF_modo = EST_MODO_HABITUAL;
            mef_habitual_reset();                      // Transición a MEF Habitual con reset.
         }
      break;
   }
}
```

Este es el funcionamiento de las 4 MEFs implementadas por el equipo para la solución del problema planteado. Además, hicimos uso de algunas funciones suministradas por la cátedra como los archivos *key.c* y *SD2_board.c* en los cuales encontramos funciones abstractas del funcionamiento de leds, switches y pulsadas del kit de desarrollo con el cuál se realizó la implementación del trabajo práctico.

También definimos un archivo llamado *cont_autos.c* donde contamos con 3 funciones que manejan el contador de autos necesario para pasar a la habilitación del camino secundario. Este archivo se ha intentado trabajar lo más genérico posible para poder contar autos a partir de cualquier switch definido en *SD2_Board.h* y es por esto que se define un array de contadores de la longitud de switches definidos. Luego, cuando queramos sumar, restar, resetear u obtener el valor del contador, deberemos pasarle el switch sobre el cuál queremos accionar.

• La función **count_updateCarCount()** recibe como parámetros la enumeración del switch y una enumeración que define si queremos sumar o restar sobre el contador. El llamado de esta función siempre debería ir acompañado de *if(key_getPressEv(switch))* para identificar que se ha pulsado el switch sobre el cual queremos sumar o restar.

• La función **count_getCarCount()** nos pide como parámetro el switch del cual deseamos la cuenta y nos devuelve un entero con la cuenta de autos para ese elemento del array.

• La función **count_resetCarCount()** también requiere como parámetro el switch sobre el cuál accionar y setea en cero su contador. Esta función se utiliza en el programa para reiniciar la cuenta de autos cuando se habilita el camino secundario, por ejemplo, en el cruce peaton, asumiendo que los autos que estaban esperando, cruzarán el camino.

Finalmente tenemos el *main.c*, el punto de entrada de la aplicación y donde se inicia la ejecución del programa. En el mismo se realizan los "inits" de todas las funciones y se procede a configurar el *SysTick timer* que utilizaremos para generar interrupciones periódicas de 1ms y a partir de su *handler*, manejar la evolución de los estados de las MEFs. Por último en un bucle infinito se ejecuta la función *MEF_modo()* descrita anteriormente.

En el Anexo al final del informe se podrá ver cada código implementado.

5. Equipamiento utilizado

El equipamiento utilizado para la realización del trabajo práctico:

• Placa de desarrollo FRDM-KL43Z.

• 2 resistencias para conexión de LEDs de 300 ohm.

• 2 LEDs para simular los leds del camino secundario.

6. Resultados obtenidos

Los resultados obtenidos fueron los esperados. El funcionamiento de la aplicación es correcto y cumple con los requisitos pedidos en el enunciado del trabajo. Las decisiones tomadas por parte del equipo para algunas partes de la solución nos parecen adecuadas y siguen la lógica del enunciado. Se ha trabajado de manera ordenada en cuanto a la distribución de funciones en archivos, definición de funciones, definición de macros y variables con el objetivo de tener un código más claro y fácil de modificar.

7. Conclusiones

Se puede concluir que la implementación de MEFs jerárquicas facilita el desarrollo de programas y ayuda a mantener un mejor orden separando en distintos archivos funcionalidades distintas de la solución. Este último punto nos fue de gran ayuda ya que durante el trabajo nos facilitó resolver inconvenientes puntuales con algunas funciones siendo más fácil ubicarlas.

Se plantearon distintas soluciones en algunos casos, como el quinto estado agregado en la MEF habitual. Una alternativa a esta solución era definir una variable local que represente si se puede o no transicionar a la MEF peaton y que estaría en false cuando se vuelve de la misma (para no permitir que se presione nuevamente el pulsador del cruce peatonal). Si bien esta alternativa también podría haber funcionado, el equipo decidió una solución más estructural agregando un nuevo estado.

En todos los casos se trató de abstraerse lo más posible de la solución particular para fomentar la reutilización de funciones, como en el caso de la detección de pulsadas, las inicializaciones de la placa o la cuenta de autos.

8. Bibliografía

[1] Sistemas Digitales II, “MEF Jerárquicas”: Modelo del sistema. Código C.

[2] Gunther Gridling, Bettina Weiss, “Introduction to Microcontrollers”, Vienna University of Technology, Institute of Computer Engineering, Embedded Computing Systems Group

February 26, 2007, Version 1.4.

[3] NXP Semiconductors, “KL43 Sub-Family Reference Manual”, Document Number: KL43P64M48SF6RM, July 2016, Rev. 5.1.

[4] NXP Semiconductors “MCUXpresso IDE User Guide”, 26 October, 2023, Rev. 11.8.0.

9. Anexo

SD2_board.c

```
/*==================[inclusions]=============================================*/
#include <SD2_board.h>
#include "fsl_port.h"
#include "fsl_gpio.h"
#include "fsl_clock.h"
#include "pin_mux.h"
/*==================[internal data declaration]==============================*/
static const board_gpioInfo_type board_gpioLeds[] ={
    {PORTE, GPIOE, 31},      /* LED ROJO KL43*/
    {PORTD, GPIOD, 5},       /* LED VERDE KL43*/
    {PORTE, GPIOE, 20},      /* LED ROJO EXTERIOR */
    {PORTB, GPIOB, 0},       /* LED VERDE EXTERIOR */
};

static const board_gpioInfo_type board_gpioSw[] ={
    {PORTA, GPIOA, 4},       /* SW1 KL43 */
    {PORTC, GPIOC, 3},      /* SW3 KL43 */
};
/*==================[external functions definition]==========================*/
void board_init(void){
        int32_t i;
        gpio_pin_config_t gpio_led_config ={
                .outputLogic = 1,
                .pinDirection = kGPIO_DigitalOutput,
        };
        gpio_pin_config_t gpio_sw_config = {
                .pinDirection = kGPIO_DigitalInput,
                .outputLogic = 0U
        };
        const port pin config t port led config = {
                .pullSelect = kPORT_PullDisable, /* Internal pull-up/down resistor is disabled */
                .slewRate = kPORT_SlowSlewRate, /* Slow slew rate is configured */
                .passiveFilterEnable = kPORT_PassiveFilterDisable, /* Passive filter disabled */
                .driveStrength = kPORT_LowDriveStrength, /* Low drive strength is configured */
                .mux = kPORT_MuxAsGpio, /* Pin is configured as PTC3 */
        };
        const port_pin_config_t port_sw_config = {
                .pullSelect = kPORT_PullUp, /* Internal pull-up resistor is enabled */
                .slewRate = kPORT_FastSlewRate, /* Fast slew rate is configured */
                .passiveFilterEnable = kPORT_PassiveFilterDisable, /* Passive filter disabled */
                .driveStrength = kPORT_LowDriveStrength, /* Low drive strength is configured *
                .mux = kPORT_MuxAsGpio, /* Pin is configured as PTC3 */
        };

        CLOCK_EnableClock(kCLOCK_PortA);
        CLOCK_EnableClock(kCLOCK_PortB);
        CLOCK_EnableClock(kCLOCK_PortC);
        CLOCK_EnableClock(kCLOCK_PortD);
        CLOCK_EnableClock(kCLOCK_PortE);

        for (i = 0 ; i < BOARD_LED_ID_TOTAL ; i++){ /* inicialización de leds */
           PORT_SetPinConfig(board_gpioLeds[i].port, board_gpioLeds[i].pin, &port_led_config);
           GPIO_PinInit(board_gpioLeds[i].gpio, board_gpioLeds[i].pin, &gpio_led_config);
        }
        for (i = 0 ; i < BOARD_SW_ID_TOTAL ; i++){         /* inicialización de SWs */
           PORT_SetPinConfig(board_gpioSw[i].port, board_gpioSw[i].pin, &port_sw_config);
           GPIO_PinInit(board_gpioSw[i].gpio, board_gpioSw[i].pin, &gpio_sw_config);
        }
}
void board_setLed(board_ledId_enum id, board_ledMsg_enum msg){
    switch (msg){
        case BOARD_LED_MSG_OFF:
        GPIO_PortSet(board_gpioLeds[id].gpio, 1<<board_gpioLeds[id].pin);
            break;
        case BOARD_LED_MSG_ON:
        GPIO_PortClear(board_gpioLeds[id].gpio, 1<<board_gpioLeds[id].pin);
            break;
        case BOARD_LED_MSG_TOGGLE:
        GPIO_PortToggle(board_gpioLeds[id].gpio, 1<<board_gpioLeds[id].pin);
            break;
        default:
            break;
    }
}
bool board_getSw(board_swId_enum id){
    return !GPIO_PinRead(board_gpioSw[id].gpio, board_gpioSw[id].pin);
}
```

Key.c

```c
/*==================[inclusions]=============================================*/
#include "key.h"

/*==================[macros and definitions]=================================*/
typedef enum{
    ESPERANDO_ACTIVACION = 0,
    ESPERANDO_DESACTIVACION,
}estPul_enum;

/*==================[internal data declaration]==============================*/
static estPul_enum estSW[BOARD_SW_ID_TOTAL];
static bool eventSW[BOARD_SW_ID_TOTAL];

/*==================[external functions definition]==========================*/

void key_init(void){
    int32_t i;
    for (i = 0 ; i < BOARD_SW_ID_TOTAL ; i++){
        estSW[i] = ESPERANDO_ACTIVACION;
        eventSW[i] = 0;
    }
}

bool key_getPressEv(board_swId_enum id){
    bool ret = false;
    if (eventSW[id]){
        eventSW[id] = 0;
        ret = true;
    }
    return ret;
}

void key_periodicTask1ms(void){
    int32_t i;
    for (i = 0 ; i < BOARD_SW_ID_TOTAL ; i++){
        switch (estSW[i])
        {
            case ESPERANDO_ACTIVACION:
                if (board_getSw(i)){
                    eventSW[i] = 1;
                    estSW[i] = ESPERANDO_DESACTIVACION;
                }
                break;

            case ESPERANDO_DESACTIVACION:
                if (!board_getSw(i){
                    estSW[i] = ESPERANDO_ACTIVACION;
                }
                break;

            default:
                estSW[i] = ESPERANDO_ACTIVACION;
                break;
        }
    }
}

// Limpia los indicadores de eventos de un switch específico:
extern void key_clearFlags(board_swId_enum id){
        eventSW[id] = 0;    // Reinicia el indicador de pulsación del switch
}
```

Main.c

```c
/*==================[inclusions]=============================================*/
#include <stdio.h>
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL43Z4.h"
#include "fsl_debug_console.h"
#include "board.h"

#include <MEFs/mef_modo.h>
#include "SD2_board.h"
#include "key.h"

int main(void) {
	/* ================== [ initialization ] ================================== */
	BOARD_InitBootPins();
	BOARD_InitBootClocks();
	BOARD_InitBootPeripherals();
	#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
	    BOARD_InitDebugConsole();
	#endif
	board_init();
	key_init();
	mef_modo_init();

	/* Interrupción de systick seteada cada 1 ms */
	SysTick_Config(SystemCoreClock / 1000U);

	while (1){
		mef_modo();
	}
	return 0 ;
}

void SysTick_Handler(void){ // Handler de interrupciones del SysTick
	key_periodicTask1ms();
	mef_modo_task1ms();
}
```

Cont_autos.c

```c
/*==================[inclusions]=============================================*/
#include "cont_autos.h"

/*==================[internal data declaration]==============================*/
static uint8_t car_count[BOARD_SW_ID_TOTAL]

/*==================[external functions definition]==========================*/
/* updateCarCount update the selected switch count with the selected mode. */
void count_updateCarCount(board_swId_enum id, car_count_mode_enum mode){
	if(mode == SUMAR)car_count[id]++;
	else{
	   if(car_count[id]) car_count[id]--;
	}
}

uint8_t count_getCarCount(board_swId_enum id){
	return car_count[id];
}

void count_resetCarCount(board_swId_enum id){
	car_count[id] = 0;
}
```

MEF_modo.c

```c
/*=================[inclusions]=============================================*/
#include <MEFs/mef_modo.h>
#include <MEFs/mef_habitual.h>
#include <MEFs/mef_peaton.h>
#include <MEFs/mef_secundario.h>
#include <stdint.h>

#include "SD2_board.h"
#include "key.h"
#include "cont_autos.h"

/*==================[macros and typedef]=====================================*/
typedef enum { // MEF states
    EST_MODO_HABITUAL = 0,
    EST_MODO_PEATON,
    EST_MODO_SECUNDARIO,
} estMefModo_enum;

/*==================[internal data definition]===============================*/
static estMefModo_enum estado_MEF_modo;

/*==================[external functions definition]==========================*/
void mef_modo_init(void){
    estado_MEF_modo = EST_MODO_HABITUAL;  // Initial State
    count_resetCarCount(BOARD_SW_ID_3);   // Setting car count at initial value: 0
    /* sub-mefs initialization */
    mef_habitual_init();
    mef_peaton_init();
    mef_secundario_init();
}

void mef_modo(void){
    switch(estado_MEF_modo){
        case EST_MODO_HABITUAL:
         switch(mef_habitual()){   // MEF evaluation return tr_enum type
                 case TR_TO_PEATON:
                         estado_MEF_modo = EST_MODO_PEATON;
                         mef_peaton_reset();
                 break;
                 case TR_TO_SECUNDARIO:
                         estado_MEF_modo = EST_MODO_SECUNDARIO;
                         mef secundario reset();
                 break;
                 default:         // No transition to a different MEF. (return is "TR_NONE")
                         estado_MEF_modo = EST_MODO_HABITUAL;
                 break;
         }
         break;
         case EST_MODO_PEATON:
             if(mef_peaton()){     // MEF evaluation return bool. True if Timeout on mef_peaton.
                  estado_MEF_modo = EST_MODO_HABITUAL;
                  mef_habitual_init_sec_5();// Init mef_habitual in state "EST_SEC_5"
             }
             break;
         case EST_MODO_SECUNDARIO:
             if(mef_secundario()){          // MEF evaluation return bool. True if car count is 0.
                  estado MEF modo = EST MODO HABITUAL;
                  mef_habitual_reset();     // Transition with Reset.
             }
             break;
    }
}

void mef_modo_task1ms(void){
    // The condition is evaluated to decrease the appropriate counters.
    if (estado_MEF_modo == EST_MODO_HABITUAL) mef_habitual_task1ms();
    if (estado_MEF_modo == EST_MODO_PEATON) mef_peaton_task1ms();
    if (estado_MEF_modo == EST_MODO_SECUNDARIO) mef_secundario_task1ms();
}
```

MEF_habitual.c

```
/*==================[inclusions]=============================================*/
#include <stdint.h>
#include "mef_habitual.h"
#include "SD2_board.h"
#include "key.h"
#include "cont_autos.h"

/*==================[macros and typedef]=====================================*/

// time duration macros definition
#define DURATION_SEC_1                          15000
#define DURATION_SEC_2                          5000
#define DURATION_SEC_3                          10000
#define DURATION_SEC_4                          5000
#define DURATION_BLINK                          100
#define CAR_COUNT_MAX                           3               // Number of cars needed to
enable secondary road.
// Times in mS.

typedef enum{
        EST_SEC_1 = 0,   // "RUTA PRINCIPAL" enabled (LVR ON).
        EST_SEC_2,              // Blink LVR indicating change of enablement.
        EST_SEC_3,              // "CAMINO SECUNDARIO" enabled (LRR ON).
    EST_SEC_4,          // Blink LVS indicating change of enablement.
        EST_SEC_5               // Idem SEC_1 but you can't move on to MEF_PEATONAL
} estMefHabitual_enum;

/*==================[internal data definition]===============================*/

static estMefHabitual_enum estado_MEF_habitual;
static uint32_t timSec_habitual;  // Timer to move on differents states.
static uint32_t timBlink_habitual;        // Timer to blink leds.

/*==================[external functions definition]==========================*/

extern void mef_habitual_init(){
        mef_habitual_reset();
}

extern void mef_habitual_reset(){
        timSec_habitual = DURATION_SEC_1;
        estado_MEF_habitual = EST_SEC_1;
        key_clearFlags(BOARD_SW_ID_1);
}

extern void mef_habitual_init_sec_5(){
        estado_MEF_habitual = EST_SEC_5;
        key_clearFlags(BOARD_SW_ID_1);
        key_clearFlags(BOARD_SW_ID_3);
        count_resetCarCount(BOARD_SW_ID_3);
}

extern tr_enum mef_habitual(void){
    switch(estado_MEF_habitual){
        case EST_SEC_1:
            board_setLed(LVR, BOARD_LED_MSG_ON);
            board_setLed(LRS, BOARD_LED_MSG_ON);
            board_setLed(LRR, BOARD_LED_MSG_OFF);
            board_setLed(LVS, BOARD_LED_MSG_OFF);

            if(key_getPressEv(BOARD_SW_ID_3)) count_updateCarCount(BOARD_SW_ID_3, SUMAR); // Add a
car to the count if the switch flags is True
            if(key_getPressEv(BOARD_SW_ID_1)) return TR_TO_PEATON;  // Return transition to
mef_peaton
            if(count_getCarCount(BOARD_SW_ID_3) >= CAR_COUNT_MAX)return TR_TO_SECUNDARIO;    //
Return transition to mef_secundario
            if(timSec_habitual == 0){
                 timSec_habitual = DURATION_SEC_2;
                estado_MEF_habitual = EST_SEC_2;
                timBlink_habitual = DURATION_BLINK;
            }
            break;
        case EST_SEC_2:
            board_setLed(LRS, BOARD_LED_MSG_ON);
            board_setLed(LRR, BOARD_LED_MSG_OFF);
            board_setLed(LVS, BOARD_LED_MSG_OFF);
```

```
            if(timBlink_habitual == 0){
                 board_setLed(LVR, BOARD_LED_MSG_TOGGLE);
                timBlink_habitual = DURATION_BLINK;
            }
            if(timSec_habitual == 0){
                 timSec_habitual = DURATION_SEC_3;
                estado_MEF_habitual = EST_SEC_3;
            }
            break;
        case EST_SEC_3:
            board_setLed(LRR, BOARD_LED_MSG_ON);
            board_setLed(LVS, BOARD_LED_MSG_ON);
            board_setLed(LVR, BOARD_LED_MSG_OFF);
            board_setLed(LRS, BOARD_LED_MSG_OFF);

            if(timSec_habitual == 0){
                timSec_habitual = DURATION_SEC_4;
                estado_MEF_habitual = EST_SEC_4;
                timBlink_habitual = DURATION_BLINK;
            }
            break;
        case EST_SEC_4:
            board_setLed(LRR, BOARD_LED_MSG_ON);
            board_setLed(LRS, BOARD_LED_MSG_OFF);
            board_setLed(LVR, BOARD_LED_MSG_OFF);

            if(timBlink_habitual == 0){
                 board_setLed(LVS, BOARD_LED_MSG_TOGGLE);
                timBlink_habitual = DURATION_BLINK;
            }
            if(timSec_habitual == 0){
                 key_clearFlags(BOARD_SW_ID_1);
                 key clearFlags(BOARD SW ID 3);
                 count_resetCarCount(BOARD_SW_ID_3);
                timSec_habitual = DURATION_SEC_1;
                estado_MEF_habitual = EST_SEC_1;
            }
            break;
        case EST_SEC_5: // It is entered only if mef_habitual_init_sec_5() is executed.
         board_setLed(LVR, BOARD_LED_MSG_ON);
         board_setLed(LRS, BOARD_LED_MSG_ON);
         board_setLed(LRR, BOARD_LED_MSG_OFF);
            board setLed(LVS, BOARD LED MSG OFF);

            if(key getPressEv(BOARD SW ID 3)) count updateCarCount(BOARD SW ID 3, SUMAR); // Add a
car to the count if the switch flags is True
            if(count_getCarCount(BOARD_SW_ID_3) >= CAR_COUNT_MAX) return TR_TO_SECUNDARIO; //
Return transition to mef_secundario
            if(timSec_habitual == 0){
                                timSec_habitual = DURATION_SEC_2;
                                estado_MEF_habitual = EST_SEC_2;
                                timBlink_habitual = DURATION_BLINK;
            }
            break;
    }
    return TR_NONE;       // By default the MEF returns none transition
}

extern void mef_habitual_task1ms(void){
    if(timSec_habitual) timSec_habitual--;
    if(timBlink_habitual) timBlink_habitual--;
}
```

MEF_Secundario.c

```c
/*==================[inclusions]=============================================*/
#include <stdint.h>
#include "SD2_board.h"
#include "key.h"
#include "cont_autos.h"
#include "mef_secundario.h"

/*==================[macros and typedef]=====================================*/
#define DURATION_EST_BLINK_1              5000
#define DURATION_EST_BLINK_2              5000
#define DURATION_BLINK                            500
typedef enum{
        EST_BLINK_1 = 0,          // Initial state (BLINK_1): Blink LVR
        EST_PASO,                 // "Paso" state: Secondary route enabled
        EST_BLINK_2               // blink_2 state: Blink LVS
} estMefSecundario_enum;

/*==================[internal data definition]===============================*/
static estMefSecundario_enum estado_MEF_secundario;
static uint32_t timSec_secundario;                // Timer to move on differents states.
static uint32_t timBlink secundario;              // Timer to blink leds.

/*==================[external functions definition]==========================*/
extern void mef_secundario_init(void){
        mef_secundario_reset();
}

extern void mef_secundario_reset(void){
    timSec_secundario = DURATION_EST_BLINK_1;
    timBlink_secundario = DURATION_BLINK;
        estado_MEF_secundario = EST_BLINK_1;
}

extern bool mef_secundario(void){
    switch(estado_MEF_secundario){
        case EST_BLINK_1:
            board_setLed(LRS, BOARD_LED_MSG_ON);
            board_setLed(LRR, BOARD_LED_MSG_OFF);
            board_setLed(LVS, BOARD_LED_MSG_OFF);
            if(key_getPressEv(BOARD_SW_ID_3)) count_updateCarCount(BOARD_SW_ID_3, SUMAR);
            if(timBlink_secundario == 0){
                timBlink_secundario = DURATION_BLINK;
                board_setLed(LVR, BOARD_LED_MSG_TOGGLE);
            }
            if(timSec_secundario == 0) estado_MEF_secundario = EST_PASO;
            break;
        case EST_PASO:
            board_setLed(LVR, BOARD_LED_MSG_OFF);
            board_setLed(LRS, BOARD_LED_MSG_OFF);
            board_setLed(LRR, BOARD_LED_MSG_ON);
            board_setLed(LVS, BOARD_LED_MSG_ON);
            if(key_getPressEv(BOARD_SW_ID_3)) count_updateCarCount(BOARD_SW_ID_3, RESTAR);
            if(count_getCarCount(BOARD_SW_ID_3) == 0){
                timSec_secundario = DURATION_EST_BLINK_2;
                timBlink_secundario = DURATION_BLINK;
                estado_MEF_secundario = EST_BLINK_2;
            }
            break;
        case EST_BLINK_2:
            board_setLed(LRS, BOARD_LED_MSG_OFF);
            board_setLed(LRR, BOARD_LED_MSG_ON);
            board_setLed(LVR, BOARD_LED_MSG_OFF);
            if(timBlink_secundario == 0){
                timBlink_secundario = DURATION_BLINK;
                board_setLed(LVS, BOARD_LED_MSG_TOGGLE);
            }
            if(timSec_secundario == 0) return true;
            break;
    }
    return false;         // By default the MEF returns false.
}

extern void mef_secundario_task1ms(void){
    if(timSec_secundario) timSec_secundario--;
    if(timBlink_secundario) timBlink_secundario--;
}
```

MEF_peaton.c

```c
/*==================[inclusions]=============================================*/
#include <stdint.h>
#include "SD2_board.h"
#include "mef_peaton.h"

/*==================[macros and typedef]=====================================*/
#define DURATION_EST_BLINK_1      5000
#define DURATION_EST_PASO         10000
#define DURATION_EST_BLINK_2      5000
#define DURATION_BLINK            200

typedef enum{
        EST_BLINK_1 = 0, // Initial state (blink_1): Blink LVR
        EST_PASO,        // "Paso" state: Pedestrian crossing on main route enabled
        EST_BLINK_2      // blink_2 state: Blink LRR
} estMefPeaton_enum;

/*==================[internal data definition]===============================*/
static estMefPeaton_enum estado_MEF_peaton;
static uint32_t timSec_peaton;                    // Timer to move on differents states.
static uint32 t timBlink peaton;                  // Timer to blink leds.

/*==================[external functions definition]==========================*/

extern void mef_peaton_init(void)mef_peaton_reset();

extern void mef_peaton_reset(void){
    timSec_peaton = DURATION_EST_BLINK_1;
    timBlink_peaton = DURATION_BLINK;
    estado_MEF_peaton = EST_BLINK_1;
}
/* mef_peaton: Return bool True to move on to MEF_Habitual */
extern bool mef_peaton(void){
    switch(estado_MEF_peaton){
        case EST_BLINK_1:
            board_setLed(LRS, BOARD_LED_MSG_ON);
            board_setLed(LRR, BOARD_LED_MSG_OFF);
            board_setLed(LVS, BOARD_LED_MSG_OFF);
            if(timBlink_peaton == 0){
                timBlink_peaton = DURATION_BLINK;
                board_setLed(LVR, BOARD_LED_MSG_TOGGLE);
            }
            if(timSec_peaton == 0) {
                timSec_peaton = DURATION_EST_PASO;
                estado_MEF_peaton = EST_PASO;
            }
            break;
        case EST_PASO:
            board_setLed(LVR, BOARD_LED_MSG_OFF);
            board_setLed(LRS, BOARD_LED_MSG_OFF);
            board_setLed(LRR, BOARD_LED_MSG_ON);
            board_setLed(LVS, BOARD_LED_MSG_ON);
            if(timSec_peaton == 0) {
                timSec_peaton = DURATION_EST_BLINK_2;
                timBlink_peaton = DURATION_BLINK;
                estado_MEF_peaton = EST_BLINK_2;
            }
            break;
        case EST_BLINK_2:
            board_setLed(LVR, BOARD_LED_MSG_OFF);
            board_setLed(LRS, BOARD_LED_MSG_OFF);
            board_setLed(LVS, BOARD_LED_MSG_ON);
            if(timBlink_peaton == 0){
                timBlink_peaton = DURATION_BLINK;
                board_setLed(LRR, BOARD_LED_MSG_TOGGLE);
            }
            if(timSec_peaton == 0) return true;
            break;
    }
    return false; // By default the MEF returns false.
}

extern void mef_peaton_task1ms(void){
    if(timSec_peaton) timSec_peaton--;
    if(timBlink_peaton) timBlink_peaton--;
}
```