IMPLEMENTACION

Funcion Transit

NOTA: Para migrar la funcion transit en ST fue necesario implementar POO.

Codigo en C, variables implementadas:

```
//stattef.h
typedef void (*STATE2)(const DU8 sig, DU8 par);

enum SIGNAL
{
    SIG_DO = 0,
    SIG_ENTRY,
    SIG_EXIT,
    SIG_FIRST_USER
};
```

```
//main.c
static STATE2 myState[2];
static DU32   myStateCnt[2];
```

Estructura de la funcion:

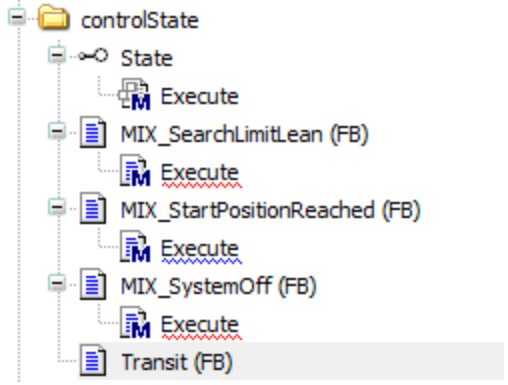
```
// State function declaration
static void Transit( const STATE2 newState, DU8 mixer );
```

```
static void Transit( const STATE2 newState, DU8 mixer )
{
        if ( newState != myState[mixer] )
        {
                if ( myState[mixer] != 0 )
                {
                        myState[mixer](SIG_EXIT, mixer);
                }

                myState[mixer] = newState;

            if ( myState[mixer] != 0 )
            {
                 myState[mixer](SIG_ENTRY, mixer);
            }

                myStateCnt[mixer] = 0;
        }
}
```

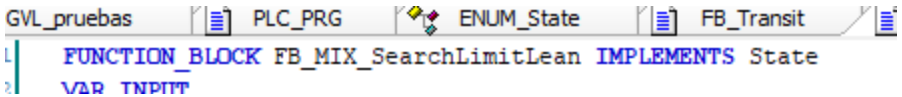
Ejemplo de funciones que implementan apuntado:

```
static void MIX_SystemOff(const DU8 sig, DU8 mixer)
static void MIX_MoveRich(const DU8 sig, DU8 mixer)
static void MIX_SearchLimitLean(const DU8 sig, DU8 mixer)
```

Migracion:



Cada uno de los FB se implementan de la interfaz



Codigo en ST

```
FUNCTION_BLOCK FB_Transit
VAR_INPUT
        newState : State;
    mixer : INT;
END_VAR
VAR_OUTPUT
END_VAR
VAR
        myState : ARRAY[0..1] OF  State;
    myStateCnt : ARRAY[0..1] OF INT;
        simulateState : State;
        pointerToSearchLimitLean : FB_MIX_SearchLimitLean;
END_VAR
```

```
// Simulación de un estado actual para entrar al condicional
simulateState := pointerToSearchLimitLean;
myState[1] := simulateState;

IF newState <> myState[mixer] THEN
    IF myState[mixer] <> 0 THEN
        myState[mixer].Execute(SIG_EXIT, mixer);
    END_IF;

    myState[mixer] := newState;

    IF myState[mixer] <> 0 THEN
        myState[mixer].Execute(SIG_ENTRY, mixer);
    END_IF;

    myStateCnt[mixer] := 0;
END_IF;
```

CAMBIOS: La logica de programacion en C para la maquina de estados consiste en punteros a funciones previamente declaradas. Aplicando los conceptos de herencia y polimorfismo de POO podemos lograr la misma funcionalidad. En vez de un puntero se crea una interfaz con un metodo en su estructura (El metodo tiene la misma estructura de el puntero STATE2); por ultimo, todas las funciones a las que se apuntaba STATE2 se crean implementadas de la interfaz, de esta manera generamos el contrato para cada FB's de nuestramaquina de estados.

Al implementar POO logramos escalabilidad y organizacion de codigo, si necesitamos generar algun otro estado de la maquina o un estado para pruebas basta con implementar otra FB y reescribir el metodo(polimorfismo) a necesidad.

Variables desconocidas:

```
//No hay variables desconocidas, más sin embargo no se encontró en el programa en C en donde se hace el llamado a Transit que ejecuta el cambio de estados.
```

SIMULACION

Se debe inicializar un estado actual (myState[mixer]) para que logre entrar al condicional (Puede inicializarse localmente).

Llamado desde la funcion principal del proyecto:

```
        newState : FB_MIX_SystemOff;
    stateInterfaz : State;
        transit : FB_Transit;

    stateInterfaz := newState;
    transit.newState := stateInterfaz;
    transit(mixer := 1);
```

Llamadado a la funcion Transist desde FB's (State)

```
        transit : Transit;
        STATE2 : State;
        pointerToStartPositionReached : MIX_StartPositionReached;

        STATE2 := pointerToStartPositionReached;
        transit.newState := STATE2;
        transit(mixer := 1);
```