IMPLEMENTACION

Función Transit

NOTA: Para migrar la función transit en ST fue necesario implementar POO. Código 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;

}

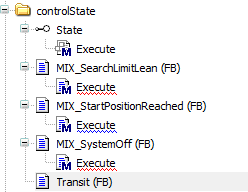
}

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



Código 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  myState[1] := | := pointerToSearchLimitLean;  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 lógica de programación en C para la máquina 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 método en su estructura (El método tiene la misma estructura de el puntero STATE2); por último, 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 nuestra máquina de estados.

Al implementar POO logramos escalabilidad y organización de código, si necesitamos generar algún otro estado de la maquina o un estado para pruebas basta con implementar otra FB y reescribir el método(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 función principal del proyecto:

newState : FB\_MIX\_SystemOff;

stateInterfaz : State; transit : FB\_Transit;

stateInterfaz := newState;

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

Llamado a la función Transist desde FB's (State)

transit : Transit;

STATE2 : State;

pointerToStartPositionReached : MIX\_StartPositionReached;

STATE2 := pointerToStartPositionReached;

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