AHK - Sistema dual

Tecnicatura superior en sistemas IT

Electrónica General

Evaluación - Módulo 3 - Ejercicio 3 - Anexo: Situación de las comas

Cámara de Industria y Comercio
Argentino-Alemana
Deutsch-Argentinische
Industrie- und Handelskammer

Docente: Dr. Ing. Matias Hampel

Alumna: C. Lorena Perugini

Mail: C. Lorena Perugini

Días: Lunes de 08:00hs a 10:50hs

Primer año

Empresa: Boehringer Ingelheim

Ciclo lectivo: 2021

Índice

Introducción

Este anexo es la explicación de la lógica detrás de la elección de las condiciones dentro de los condicionales de las líneas 312, 318 y 324 dentro de la función "muestreo" del tercer ejercicio de la evaluación correspondiente al tercer módulo de la materia.

Realizo este anexo porque no me gusta que queden cosas que parecen sacadas de un sombrero en mis evaluaciones, menos en esta materia, mucho menos con un lenguaje que disfruto tanto como Arduino. Además, la verdad que, la explicación me pareció muy interesante ya que conecta temas del primer módulo. Me pareció sumamente divertido realizar esta lógica. Por todos esos motivos me pareció coherente realizar este anexo.

Debo aclarar que en la consigna se habla sobre canales de un datalogger que pueden tomar diferentes valores de tensión. En este anexo me abstraigo de eso ya que considero a los canales como elementos booleanos según su estado en el datalogger (activo o inactivo, 1 o 0). Entonces cuando menciono a "**estadoCX**" refiero al estado del canal X en el datalogger. Aplica lo mismo cuando refiero a **"CX"**, solo para este anexo las considero expresiones similares.

Por otro lado, en la primer entrega de la evaluación coloqué este código en la línea 324:

```
if( !(((estadoC1+estadoC2+estadoC3+estadoC4) == 1) &&
estadoC4) )
```

Esto es incorrecto y me percaté de ello hoy, la correción es la siguiente. (La parte incorrecta son los paréntesis de **este color**).

```
if( (!((estadoC1+estadoC2+estadoC3+estadoC4) == 1)) &&
estadoC4)
```

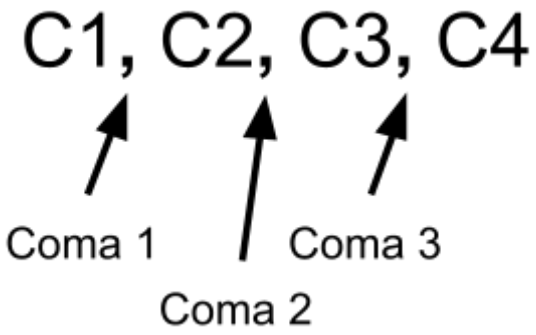
La lógica de ambos códigos la explico en esta sección.

Igualmente a la hora de querer plantearlo para poderlo explicar me dí cuenta que había un método mucho más simple utilizando tablas de verdad y mapas de Karnaugh. Ese es el método que explicaré primero. Le llamé versión 1.1 ya que el cambio con lo que entregué en la evaluación es mínimo.

Explicación versión 1.1

En el ejercicio hay un punto que dice lo siguiente: ***"Los datos deberán ser transmitidos separados por coma".***

Antes de resolver el ejercicio es importante comprender cuáles son las posibles comas:



Entonces, sabemos que cada canal puede estar activo o inactivo y según ese estado las comas deben estar o no. Una forma de resolver este planteo es con una tabla de verdad y un mapa de Karnaugh para cada una de las 3 posibles comas.

Acá hay que tomar una decisión, por ejemplo, en el caso donde C1 y C3 están activos ¿Qué coma muestro? ¿La coma 1 o la coma 2? Por comodidad considero que para estos casos se elige la coma más cercana al siguiente valor a mostrar. Ejemplo, en el caso planteado mostraría la coma 2. Por ende **la coma 1 siempre va a separar a C1 de C2.** En contraparte la coma 2 siempre separará a C3 de C2 o de C1. Y por último la coma 3 siempre separará a C4 de cualquier otro canal.

Coma 1

Esta es la coma más simple, ya que solo depende de C1 y C2 porque, como expliqué antes, su función es separar a C1 y a C2.

C_1	C_2	$Coma\ 1$
0	0	0
0	1	0
1	0	0
1	1	1

El mapa de Karnaugh resultante es el siguiente:

C_1C_2	$\overline{C_1}$	C_1
$\overline{C_2}$	0	0
C_2	0	1

La ecuación resultante es:

$$Coma\,1 = C_1 * C_2$$

Lo que se traslada al código de la siguiente forma.

```
if(estadoC1 && estadoC2)
    Serial.print(",");
```

Coma 2

Para esta coma requerimos de los valores de C1, C2 y C3. Esto ya que es la coma encargada de separar a C3 de C1 o de C2.

Por lo tanto la tabla de verdad es la siguiente.

C_1	C_2	C_3	$Coma\,2$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

El mapa de Karnaugh resultante es el siguiente:

$C_1C_2C_3$	$\overline{C_1} * \overline{C_2}$	$\overline{C_1} * C_2$	$C_1 * C_2$	$C_1 * \overline{C_2}$
$\overline{C_3}$	0	0	0	0
C_3	0	1	1	1

La ecuación resultante es:

$$Coma\ 2 = C_3 * C_2 + C_3 * C_1$$

Simplifico por factor común C_3.

$$Coma\ 2 = C_3 * (C_2 + C_1)$$

Reordeno por mera comodidad.

$$Coma\ 2 = (C_1 + C_2) * C_3$$

Lo que se traslada al código de la siguiente forma.

```
if((estadoC1 || estadoC2) && estadoC3)
    Serial.print(",");
```

Coma 3

Esta se podría decir que es la coma más complicada ya que depende de los 4 valores.

Por lo tanto la tabla de verdad es la siguiente.

C_1	C_2	C_3	C_4	$Coma\ 3$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

El mapa de Karnaugh resultante es el siguiente:

$C_1C_2C_3C_4$	$\overline{C_1} * \overline{C_2}$	$\overline{C_1} * C_2$	$C_1 * C_2$	$C_1 * \overline{C_2}$
$\overline{C_3} * \overline{C_4}$	0	0	0	0
$\overline{C_3} * C_4$	0	1	1	1
$C_3 * C_4$	1	1	1	1
$C_3 * \overline{C_4}$	0	0	0	0

La ecuación resultante es:

$$Coma\,3 = C_4 * C_3 + C_4 * C_2 + C_4 * C_1$$

Simplifico por factor común C_4.

$$Coma\,3 = C_4 * (C_3 + C_2 + C_1)$$

Reordeno por mera comodidad.

$$Coma\,3 = C_4 * (C_1 + C_2 + C_3)$$

Lo que se traslada al código de la siguiente forma.

```
if(estadoC4 && (estadoC1 || estadoC2 || estadoC3))
        Serial.print(",");
```

Explicación versión 1

Si bien el planteo anterior es muy bonito, lo que hice en la prueba fue un poco diferente. En lugar de plantearlo para cada coma yo planteé una tabla general donde escribí los 16 estados y cada una de las comas (posicionadas antes de un uno en lugar de después). Me quedó una tabla de este estilo:

C_1	C_2	C_3	C_4	$Coma\ 1$	$Coma\ 2$	$Coma\ 3$
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	0	1	1
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	1	1	0
1	1	1	1	1	1	1

Teniendo esta tabla de verdad me percaté que la mayoría de comas iban siempre que se cumple esto:

$$Coma\ n\ -> C_n\ *\ C_{n+1}$$

Lo cuál es correcto para la coma 1 pero incompleto para las demás.

Entonces pensé en dividir las comas en 2 tipos, las que cumplen esta regla y las que no. Marqué en la tabla anterior el segundo grupo con este color.

Entonces para la coma 2 me quedaron los siguientes casos.

1	0	1	0
1	0	1	1

Lo cuál se simplifica de la siguiente forma.

1	0	1	X

Entonces me dí cuenta que los casos del tipo 1 donde estaba la coma 2 siempre eran de la siguiente forma.

X	1	1	X

Y ahí llegué a la conclusión de la versión 1.1

$$Coma\ 2\ = (C_1\ +\ C_2)\ * C_3$$

Para la coma 3 me quedaron los siguientes casos del tipo 2

0	1	0	1
1	0	0	1
1	1	0	1

Me pareció curioso que se cumple

X	X	0	1

(Siempre que ambas X 's no sean 0).

Y entonces me dí cuenta que se cumplía en todos los casos lo siguiente.

X	X	X	1

(Siempre que las 3 X 's no sean 0).

Y acá es donde me emocioné y dije “Va en todos los casos menos en 0001” y puse el siguiente código:

```
if( !(((estadoC1+estadoC2+estadoC3+estadoC4) == 1) &&
estadoC4) )
```

Llegué a esta conclusión porque 0001 es equivalente a que haya un solo canal encendido y que ese sea C4. El problema con esto es que en mi intento de abstracción me abstraje de la importancia de que C4 esté activa, entonces esta coma se activaba en literalmente todos los casos menos 0001.

A la mañana siguiente lo corregí al siguiente código:

if((!((estadoC1+estadoC2+estadoC3+estadoC4) == 1)) && estadoC4)

Esto porque la coma va siempre que no haya un solo canal activo y C4 esté activo.

Vale aclarar que el paréntesis azul es código que utilicé en el condicional de la línea 230 en la función "seleccion_canales" para verificar que solo haya un canal activo.

Conclusión

El método de la versión 1.1 es más limpio y simple que el de la versión 1. Para las primeras 2 comas el resultado es el mismo. Para la tercera coma el resultado es equivalente y funcional pero no es el mismo. Y entre

if((!((estadoC1+estadoC2+estadoC3+estadoC4) == 1)) && estadoC4)

y

if(estadoC4 && (estadoC1 || estadoC2 || estadoC3))

Me quedo con la segunda forma porque es más fácil de interpretar a primera vista.

Otra conclusión es que debo aplicar siempre los conocimientos de forma conjunta. No está bien que comience haciendo una tabla de verdad y luego empiece a buscarle una lógica con un criterio tan aleatorio. Comencé la lógica haciendo una tabla de verdad y debo continuar haciendo un mapa de Karnaugh. Ese aprendizaje me lo llevo a mi vida diaria donde me encanta hacer tablas de verdad por tonterías pero nunca hago los mapas.