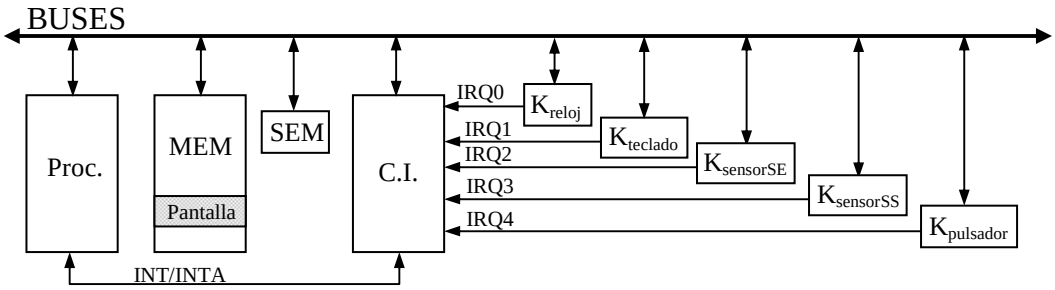
Universidad del País Vasco
Facultad de Informática

Departamento de Arquitectura y Tecnología de Computadores

Arquitectura de Computadores I

Subsistema de entrada/salida 1 (solución): Aeropuerto

Se dispone de un sistema de control de la presencia de aves en el área de despegue y aterrizaje de aviones en un aeropuerto**.** La estructura hardware del sistema es la clásica, como se puede observar en la figura, con la salvedad de que se han añadido dos sensores (SE, SS) y un pulsador, cuyo funcionamiento se detallará posteriormente. Además, se ha de controlar un semáforo, cuyo controlador posee únicamente un registro de control, mapeado en memoria, en la dirección SEM.



Los periféricos que debe controlar el microprocesador son idénticos a los vistos en la asignatura, exceptuando los dos sensores, el pulsador y el semáforo, cuyas características se exponen a continuación:

*** Sensor SE**: Sensor de entrada de aves. Detecta cuándo entra algún ave (una o más) en la zona controlada y genera una petición de interrupción. En su registro de datos se puede leer entonces el número de aves que han entrado en ese instante. Una vez leído el registro de datos, es necesario hacer `strobe` sobre su registro de control.

*** Sensor SS**: Sensor de salida de aves. Detecta cuándo sale algún ave (una o más) de la zona controlada y genera una petición de interrupción. En su registro de datos se puede leer entonces el número de aves que han salido en ese instante. Una vez leído el registro de datos, es necesario hacer `strobe` sobre su registro de control.

*** Semáforo SEM**: Semáforo de aviso a los aviones. Puede estar **Verde**, en cuyo caso significa que no hay aves en la zona controlada y que el avión puede realizar la maniobra correspondiente, o **Rojo**, en caso contrario. Como se ha indicado, posee un registro de control mapeado en memoria en la dirección SEM.

*** Pulsador**: Genera una petición de interrupción al ser pulsado cada vez que el personal encargado de ahuyentar a las aves captura una de ellas.

*** Reloj**: Interrumpe 18 veces por segundo.

*** Teclado**: Cada vez que se pulsa una tecla produce dos peticiones de interrupción: la primera indica que se ha pulsado tecla (MAKE) y la segunda indica que se ha liberado (BREAK). Posee un registro de datos, en el que se puede leer el código de posición ("*scan code*") correspondiente a la última tecla pulsada, y un registro de control, en el que es necesario escribir una secuencia de `strobe` cada vez que se acepta una interrupción. No posee registro de estado.

*** Controlador de interrupciones**: Posee los registros: máscara IMR, de petición de interrupción IRR y de interrupción en servicio ISR.

El **funcionamiento** del sistema es el siguiente. La situación o **estado normal** es que no haya aves en la zona controlada y, por tanto, el semáforo estará **verde**. En cualquier otra situación, el semáforo estará rojo por precaución.

En el momento en que se detecta la entrada de aves, el sistema pasa a un **estado de pre-alarma**, estado en el que permanecerá mientras haya aves en la zona controlada (por supuesto, las aves pueden salir "de motu propio") y el operador encargado del sistema no indique la urgencia de ahuyentar a las aves por la inminente maniobra de un avión. En este caso de urgencia, el operador pulsará la **tecla A**, lo que hará que el sistema pase a un **estado de alarma**.

En el estado de alarma, además de mantener el semáforo rojo, se genera una alarma sonora (podemos suponer que para ello disponemos de la rutina ya escrita `genera_alarma()`) de aviso al personal encargado de ahuyentar a las aves. Esta "brigada de limpieza" debe atrapar con una red las aves presentes en la zona controlada, apretando el pulsador tantas veces como aves queden atrapadas en dicha red. De nuevo, por supuesto, las aves pueden salir "de motu propio" ahuyentadas por la presencia humana o por el sonido de la alarma.

Cuando por fin la brigada ahuyentadora consiga que se haya vaciado de aves la zona controlada, el sistema pasará a un **estado de post-alarma**, y se deberá desactivar la alarma sonora (supongamos que disponemos de la rutina ya escrita `desactiva_alarma()`). En ese estado de post-alarma, el sistema se mantendrá durante **2 minutos**, por precaución. Si durante este tiempo se detecta de nuevo la entrada de aves, se vuelve al estado de alarma. Si no, pasado ese tiempo sin aves, el sistema vuelve al estado normal.

Se pide: escribir en lenguaje pseudo algorítmico el programa principal y las rutinas de servicio siguientes: sensor SE, sensor SS, pulsador, teclado y reloj. Para simplificar el trabajo, conviene representar gráficamente el funcionamiento del sistema mediante un autómata de estados.

Solución

En primer lugar, es conveniente y de gran ayuda deducir un autómata de estados que refleje el funcionamiento del sistema. Para ello es fundamental distinguir bien todas las situaciones o estados posibles del funcionamiento del sistema, así como identificar correctamente cuáles son las condiciones que se han de cumplir para que se produzcan las transiciones de unos estados a otros. En general, las transiciones entre estados serán consecuencia de sucesos o eventos externos al propio sistema, que serán detectados por medio de los periféricos presentes en el sistema, sea cual sea el modo en que dichos periféricos se sincronicen con el procesador, tanto por encuesta como por interrupción.
En este ejercicio, al ser el primero planteado, los estados del sistema vienen definidos claramente en el propio enunciado. De hecho, estos son los estados en los que puede encontrarse el sistema: *normal*, *pre-alarma*, *alarma* y *post-alarma*.
Analicemos ahora detalladamente todas las posibles transiciones de estados del sistema, estado por estado. Es decir, analicemos todos los eventos que pueden producirse en este sistema, y veamos cuáles deben ser las consecuencias de los mismos según cuál sea el estado del sistema en el momento en que se producen. Para todo ello, procederemos a releer el enunciado detalladamente.

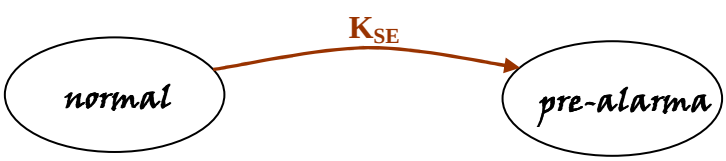
1. Estado *normal*:

- Características del estado:

  En el enunciado podemos leer: "La situación o **estado normal** es que no haya aves en la zona controlada y, por tanto, el semáforo estará **verde**".

- Transiciones posibles:

a) En el enunciado podemos leer: "En el momento en que se detecta la entrada de aves, el sistema pasa a un **estado de pre-alarma**".
¿Y cómo detecta el sistema la entrada de aves en la zona controlada? Pues para eso está precisamente el sensor denominado SE: "Sensor de entrada de aves". El controlador de este sensor solicita una petición de interrupción cada vez que detecta que un ave o más han entrado en la zona bajo control. Cuando el procesador acepte la interrupción, interrumpirá la ejecución del programa que esté ejecutando en ese momento y saltará a ejecutar la rutina de servicio o de atención a la interrupción correspondiente al sensor SE. Por tanto, será en dicha rutina de servicio a la interrupción en donde el programador del sistema deberá reflejar el cambio de estado.
En el autómata lo reflejaremos así: la flecha entre los dos estados indica el sentido de la transición —del estado *normal* al estado de *pre-alarma*—, y la etiqueta que aparece sobre la flecha, por su parte, indica el periférico o el evento que da lugar a dicha transición de estado.



- Acciones a realizar en la rutina de servicio a la interrupción del sensor SE:

Además de la transición de estado indicada en el autómata, la rutina de atención a la interrupción del sensor SE tiene otra serie de tareas que realizar:

SE.1) Se debe leer el registro de datos del controlador del sensor SE para saber la cantidad de aves que han entrado en la zona controlada. Utilizaremos para ello la función ya escrita `InPort(dir_reg)`, que recibe como parámetro la dirección del mapa de entrada/salida en la que se encuentra el registro; indicaremos esa dirección con la etiqueta `R_DAT_KSE`. El valor devuelto por esa función lo almacenaremos en una variable local, así:

```
cant_aves_entr = InPort(R_DAT_KSE);
```

SE.2) Está claro que en el estado ***normal*** no hay aves en la zona controlada (ya que ése es el punto de partida del sistema), pero en el resto de estados del sistema es necesario saber con exactitud la cantidad de aves que hay en la zona controlada, ya que es posible que las aves salgan de la zona de control de la misma manera que han entrado, por lo que podría darse el caso de que el sistema tuviese que pasar del estado de ***pre-alarma*** al estado ***normal*** en el caso de que todas las aves presentes en el área controlada salieran de la misma (analizaremos esta posibilidad un poco más adelante). Por lo tanto, es necesario saber cuántas aves hay en cada instante dentro de la zona controlada. Para ello, utilizaremos una variable global que denominaremos `cant_aves_zona` (es necesario que sea global debido a que su valor deberá ser actualizado o modificado en más de una rutina de servicio a interrupciones, como veremos en el transcurso de la resolución). En consecuencia, el valor inicial de dicha variable global será 0 (habitualmente, las inicializaciones necesarias de las variables globales las realizaremos en el programa principal), pero al detectar el sensor SE que han entrado más aves en la zona controlada, es necesario actualizar adecuadamente el valor de dicha variable global, sumándole la cantidad de aves que acaban de ser detectadas en la entrada. Así:

```
cant_aves_zona = cant_aves_zona + cant_aves_entr;
```

SE.3) Continuando con la información dada en el enunciado, una vez leído el registro de datos se debe realizar una secuencia de `strobe` sobre el registro de control del controlador del sensor SE (`R_CONT_KSE`). Para ello, supondremos que disponemos de una función ya escrita:

```
strobe(R_CONT_KSE);
```

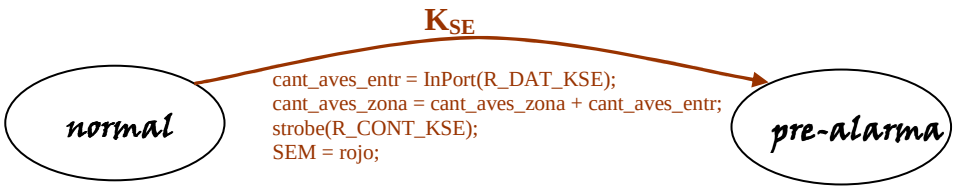
SE.4) Finalmente, hay que cambiar el color del semáforo, poniéndolo en rojo, color en el que se mantendrá mientras en la zona controlada haya aves. Dado que el controlador del semáforo únicamente dispone de un registro de control mapeado en memoria, en la dirección etiquetada como `SEM`, podemos manejar dicha etiqueta como si fuera una variable global, ya que indica una posición de memoria. Por tanto, el cambio de color del semáforo lo indicaremos así:

```
SEM = rojo;
```

No obstante, en lugar de manejar la etiqueta `SEM` como variable global, también podríamos manejarla como dirección de memoria, por lo que podríamos utilizar la función `escribe_mem(dir,dato)` para escribir en memoria, así:

```
escribe_mem(SEM,rojo);
```

Es conveniente indicar todas esas acciones sobre el autómata, ya que ello resultará de gran ayuda a la hora de escribir el código de la rutina de servicio a la interrupción. Por tanto, bajo la flecha que indica la transición de estado, escribiremos todas esas acciones, así:

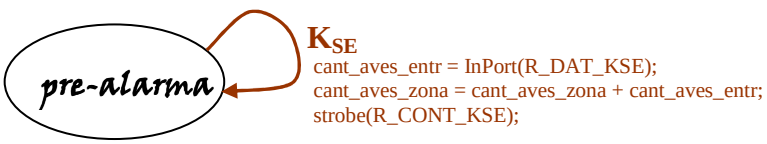


b) En principio, no hay más transiciones posibles que hagan que el sistema salga del estado *normal*.

2. Estado de *pre-alarma*:

- Características del estado:
  En el enunciado podemos leer: “el sistema pasará a un **estado de pre-alarma**, estado en el que permanecerá mientras haya aves en la zona controlada (por supuesto, las aves pueden salir "de motu propio") y el operador encargado del sistema no indique la urgencia de ahuyentar a las aves por la inminente maniobra de un avión”.
  De ese párrafo se deducen varias posibilidades:

  ▪ Por una parte, mientras el sistema está en el estado de *pre-alarma*, es posible que entren más aves en la zona, lo cual no conllevará una transición a otro estado, ya que el sistema debe permanecer en dicho estado aunque entren más aves en la zona controlada, pero sí será necesario actualizar adecuadamente la cantidad de aves presentes dentro de la zona controlada. Y todo ello hay que tenerlo en cuenta en la rutina de servicio a la interrupción del controlador del sensor SE. Así, las acciones anteriores (SE.1, SE.2 y SE.3) se deben realizar como en el caso anterior, pero debemos tener claro que es necesario modificar ligeramente la acción SE.4, ya que, en este caso, dado que el sistema ya está en el estado de *pre-alarma*, no es necesario realizar la transición de estado, y además, no es necesario cambiar el color del semáforo, puesto que ya está en rojo. Por tanto, en el autómata plasmaremos así esta posibilidad:



Y la acción SE.4 la modificaremos como sigue, para tener en cuenta cuál es el estado del autómata en el momento en que se produce la petición de interrupción del sensor SE:

SE.4) La transición de estado y el cambio de color del semáforo únicamente deben realizarse en el caso de que el estado del autómata sea el estado ***normal***, no en otros estados. Así, es necesario disponer de una variable global que nos permita saber en todo momento cuál es el estado del sistema; a dicha variable la denominaremos `estado_automata`. En consecuencia, en la rutina de atención a la interrupción del sensor SE deberemos analizar el estado del sistema para ver si es necesario realizar la transición de estado y el cambio de color del semáforo:

```
if (estado_automata == normal)
   {
       SEM = rojo;
       estado_automata = pre-alarma;
   }
```

- Por otra parte, mientras el sistema está en el estado de ***pre-alarma***, es posible que algunas de las aves presentes en la zona controlada salgan de la misma, pero ello no tiene que tener como consecuencia otra transición de estado siempre que en la zona controlada queden aves todavía. Eso sí, también en ese caso es necesario actualizar adecuadamente la cantidad de aves presentes en la zona controlada. Pero, ¿cómo sabe el sistema que algunas aves han salido de la zona controlada? Pues para eso precisamente está el sensor SS: "Sensor de salida de aves". El controlador de este sensor solicitará una petición de interrupción cada vez que el sensor detecte que una o más aves han abandonado la zona controlada. Al ser aceptada dicha petición de interrupción, el procesador pasará a ejecutar la rutina de servicio correspondiente al sensor SS, y en ella tendremos que programar las acciones que debe realizar el sistema. He aquí:

↳ Acciones a realizar en la rutina de servicio a la interrupción del sensor SS:

SS.1) Se debe leer el registro de datos del controlador del sensor SS (su dirección en el mapa de entrada/salida es `R_DAT_KSS`), para saber la cantidad de aves que han abandonado la zona controlada. Para ello, el valor devuelto por la función `InPort` lo almacenaremos en una variable local, así:

```
cant_aves_sal = InPort(R_DAT_KSS);
```

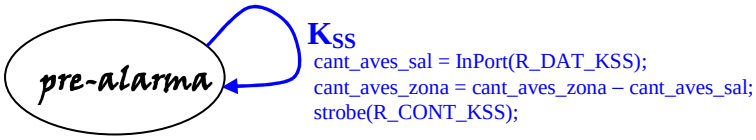
SS.2) Se debe actualizar adecuadamente la variable global `cant_aves_zona` restándole la cantidad de aves que acaban de abandonar la zona controlada, así:

```
cant_aves_zona = cant_aves_zona - cant_aves_sal;
```

SS.3) También en el caso de este sensor, es necesario realizar una secuencia de `strobe` sobre su registro del control (`R_CONT_KSS`), así:

```
strobe(R_CONT_KSS);
```

En el autómata plasmaremos así esta posibilidad:



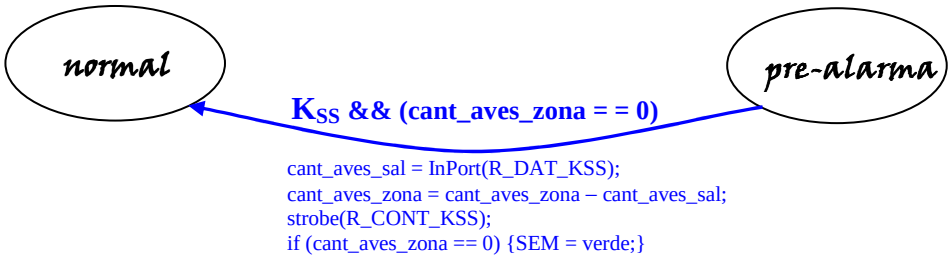
- Pero ¿qué ocurre cuando, una vez actualizada la variable `cant_aves_zona`, el valor resultante es 0? Es decir, ¿qué sucede cuando todas las aves que había en la zona controlada han abandonado la misma? En el enunciado no se dice nada claramente, pero aquí hay que hacerle caso a la lógica: lo más lógico es que, en ese caso, el sistema vuelva al estado ***normal***, puesto que en la zona controlada ya no queda ningún ave. Como eso es una transición de estado, la analizaremos en el apartado siguiente.
- Finalmente, si leemos nuevamente el enunciado, nos daremos cuenta de que "el sistema permanecerá (en el estado de pre-alarma) mientras […] el operador encargado del sistema no indique la urgencia de ahuyentar a las aves por la inminente maniobra de un avión". Está claro, por tanto, que también esa circunstancia supondrá una transición de estado, por lo que la analizaremos en el apartado siguiente.

- Transiciones posibles:

a) Así pues, cuando el sensor SS solicita una interrupción, se debe actualizar la variable global `cant_aves_zona`, y si el valor resultante es 0, entonces el sistema deberá volver al estado ***normal***, ya que no queda ningún ave en la zona de control. Por la misma razón, el semáforo deberá volver a ponerse en verde. Indicaremos todo ello en el autómata mediante una flecha entre ambos estados, y en la rutina de servicio a la interrupción del sensor SS como se indica a continuación:

SS.4) Transición de estado y cambio de color del semáforo si es necesario, es decir, si el sistema está en el estado de ***pre-alarma*** y en la zona controlada no queda ningún ave más:

```
if ((cant_aves_zona == 0)&&( estado_automata == pre-alarma))
      {
          SEM = verde;
          estado_automata = normal;
      }
```



b) Recordemos que el sistema saldrá del estado de ***pre-alarma*** cuando "el operador encargado del sistema indique la urgencia de ahuyentar a las aves por la inminente maniobra de un avión". En ese caso, el operario pulsará la tecla A y el sistema deberá pasar al estado de ***alarma***. Está claro, por tanto, que el evento desencadenante de dicha transición será la petición de interrupción por parte del teclado, pero sólo se deberá tener en cuenta si la tecla pulsada es la A, ignorándose cualquier otra tecla. Además, sólo se deberá producir la transición de estado al pulsar la tecla A, no al liberarla —recordemos que el controlador del teclado genera dos peticiones de interrupción por cada tecla: una al pulsarla (MAKE), y otra al liberarla (BREAK)—. En consecuencia, la transición de estado es condicional: únicamente cuando se pulsa la tecla A. Las acciones a realizar en la rutina de atención a la interrupción del teclado son las siguientes:

↳ Acciones a realizar en la rutina de servicio a la interrupción del teclado:

Tec.1) Se debe leer el registro de datos del controlador del teclado (dirección `R_DAT_Ktec`) para saber qué tecla ha sido pulsada (o liberada). Se debe tener en cuenta que el valor leído en el registro de datos es el código de posición o "*scan code*" de la tecla, no el carácter alfanumérico que la tecla tiene asignado en el teclado. Por esa razón, normalmente se suele traducir dicho código de posición, para saber qué carácter concreto le corresponde a la tecla pulsada (o liberada).

```
código_tecla = InPort(R_DAT_Ktec);
```

Tec.2) Ahora se debe analizar si la tecla ha sido pulsada (MAKE) o liberada (BREAK) —supondremos que disponemos para ello de una función ya escrita: `MAKE(código_tecla)`, que devuelve un 1 si se trata de MAKE, o un 0 si se trata de BREAK—, y además debemos analizar si se trata de la tecla A —para ello, traduciremos el código de posición de la tecla al código ASCII que le corresponde, leyendo una tabla que tenemos almacenada en memoria, empleando el código de posición de la tecla como índice dentro de la tabla: `TABLA_ASCII [código_tecla]`, y todo eso hay que hacerlo únicamente en el estado de ***pre-alarma***, ya que en cualquier otro estado del sistema no hay que hacerle caso al teclado. Así pues, cuando se cumplen esas tres condiciones, además de realizar el cambio de estado, se debe generar una alarma sonora, para lo que disponemos de una función ya escrita: `genera_alarma()`. En consecuencia, el código correspondiente queda como sigue:

```
if ((MAKE(código_tecla))&&(TABLA_ASCII [código_tecla]== 'A')
        &&(estado_automata == pre-alarma))
                {
                    genera_alarma();
                    estado_automata = alarma;
                }
```

Tec.3) También en el caso del teclado es necesario realizar una secuencia de `strobe` sobre su registro del control (`R_CONT_Ktec`):

```
strobe(R_CONT_Ktec);
```

En el autómata plasmaremos así esta posibilidad:

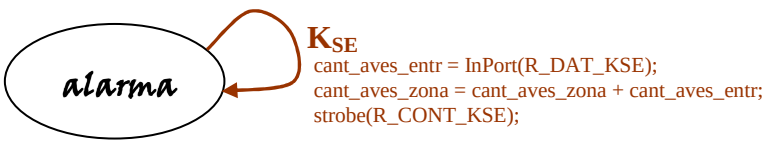
3. Estado de *alarma*:

- Características del estado:

  En el enunciado de puede leer: "En el estado de ***alarma***, además de mantener el semáforo rojo, se genera una alarma sonora […] de aviso al personal encargado de ahuyentar a las aves. Esta "brigada de limpieza" debe atrapar con una red las aves presentes en la zona controlada, apretando el pulsador tantas veces como aves atrapadas en dicha red".

  Igual que en el caso anterior, pueden darse varias posibilidades:

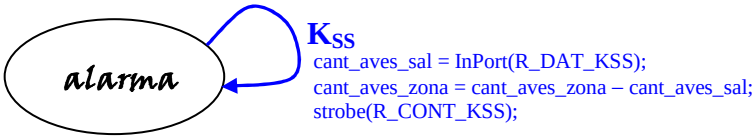
  - Por una parte, también en el estado de ***alarma*** es posible que entren más aves en la zona de control, y eso no tendrá efecto sobre el estado del sistema, es decir, el sistema seguirá en el mismo estado aunque entren más aves. Pero sí que habrá que actualizar el número de aves presentes en la zona de control. Por tanto, si el sensor de entrada solicita une interrupción estando el sistema en el estado de ***alarma***, entonces hay que realizar las acciones vistas anteriormente, pero en este caso no se debe realizar ni transición de estado ni cambio del color del semáforo (puesto que ya está en rojo). En consecuencia, el autómata quedará así:



    Y en la rutina de servicio del sensor SE no debemos modificar la acción SE.4, puesto que ya habíamos tenido en cuenta que el cambio del color del semáforo y la transición de estado sólo deben producirse en el estado ***normal***.

  - Por otra parte, también en el estado de ***alarma*** es posible que algunas aves salgan de la zona de control por iniciativa propia, asustadas, tal vez, por el sonido de la alarma, y esto tampoco deberá tener ninguna influencia sobre el estado del sistema, mientras en la zona de control todavía queden aves.

El autómata quedará así:



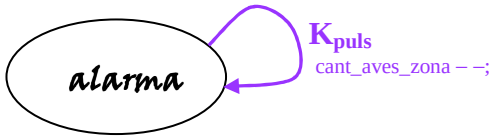
- En cualquier caso, en el estado de ***alarma*** los operarios de la brigada de limpieza deberán atrapar con una red todas las aves que aún permanezcan en la zona de control, y apretar el pulsador correspondiente tantas veces como aves atrapadas. En ese momento, el controlador del pulsador activará una petición de interrupción. En consecuencia, en la rutina de servicio a la interrupción del pulsador se deberá decrementar en una unidad la cantidad de aves presentes en la zona bajo control (`cant_aves_zona--`), ya que sabemos que hay un ave menos, porque ya ha sido atrapada. Pero no se debe producir ninguna transición de estado mientras en la zona bajo control siga habiendo aves. Eso sí, al igual que en el caso del teclado, las interrupciones del pulsador sólo deben ser tenidas en cuenta cuando el sistema está en el estado de ***alarma***, en ningún otro estado, para evitar que se tenga en cuenta la pulsación accidental del pulsador en cualquier otro estado.

↳ Acciones a realizar en la rutina de servicio a la interrupción del pulsador:

Puls.1) Si el sistema está en el estado de ***alarma***, se debe actualizar el valor de la variable `cant_aves_zona`, teniendo en cuenta que la brigada ha atrapado una sola ave:

```
if (estado_automata == alarma)
   {
     cant_aves_zona --;
   }
```

Y en el autómata:



- Finalmente, ¿qué ocurre en los dos casos que acabamos de analizar (en las rutinas de servicio a las interrupciones de los controladores K_{SS} y K_{puls}) si, una vez actualizado el valor de la variable `cant_aves_zona`, resulta un 0? Es decir, ¿qué ocurre cuando en la zona bajo control ya no queda ningún ave? En el enunciado queda claro: el sistema deberá pasar a un estado de ***post-alarma***. Como se trata de una transición de estado, lo veremos en el subapartado siguiente.

- Transiciones posibles:

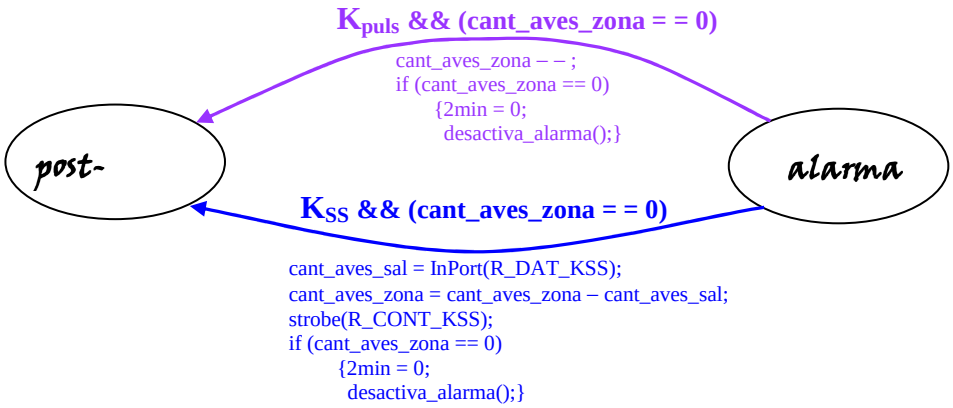
a) Recordemos lo que dice el enunciado acerca del estado de ***alarma***: "Cuando por fin la brigada ahuyentadora consiga que se haya vaciado de aves la zona controlada, el sistema pasará a un **estado de *post-alarma***". Así pues, la transición de estado se producirá cuando en la zona bajo control no quede ningún ave, y esto podrá ocurrir tanto cuando se produzca la interrupción del pulsador, como cuando interrumpa el sensor de salida, SS, si, después de actualizada la variable `cant_aves_zona` en

ambos casos, el valor resultante es 0. El sistema pasará entonces al estado de ***post-alarma***.
Además, se debe desactivar la alarma sonora, para lo que disponemos de la rutina ya escrita `desactiva_alarma()`.
Como el sistema deberá permanecer en el estado de ***post-alarma*** un tiempo de 2 minutos, mientras no vuelvan a entrar aves, la transición hacia ese estado es el momento adecuado para inicializar la variable global que controlará ese intervalo de tiempo (la denominaremos `2min`).
En consecuencia, en las rutinas de atención a esos dos periféricos (pulsador y sensor de salida, SS) se deberán llevar a cabo las siguientes acciones:

Puls.2) y **SS.5)** Transición de estado, si fuera necesario:

```
if ((cant_aves_zona == 0)&&(estado_automata == alarma))
    {
        2min = 0;
        desactiva_alarma();
        estado_automata = post-alarma;
    }
```

Y el autómata quedará así:



4. Estado de *post-alarma*:

- Características del estado:
  Como se indica en el enunciado, "en ese estado de ***post-alarma*** el sistema se mantendrá durante **2 minutos**, por precaución. Si durante este tiempo se detecta de nuevo la entrada de aves, se vuelve al estado de ***alarma***. Si no, pasado ese tiempo sin aves, el sistema vuelve al estado ***normal***" y el avión recibirá permiso para maniobrar sin problemas.
  En el párrafo anterior están indicadas las dos transiciones de estado que se pueden producir en el estado de ***post-alarma***, pero las dejaremos para el apartado siguiente. En este momento lo que nos interesa es ver qué acciones debe realizar el sistema mientras esté en ese estado. Sabemos que debe permanecer en el mismo durante 2 minutos, por lo que será necesario controlar el paso del tiempo. Para ello, el sistema le debe hacer caso al reloj, para saber cuándo se acaba ese plazo de tiempo. Eso quiere decir que, en la rutina de atención a la interrupción del reloj, se debe actualizar adecuadamente la variable `2min`, y para ello hay más de una opción, como veremos a continuación.

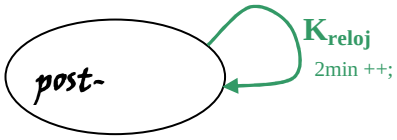
Sabemos que el controlador del reloj realiza 18 peticiones de interrupción en un segundo. Si actualizamos el valor de la variable `2min` en una unidad cada vez que se ejecuta la rutina de atención al reloj, sin utilizar ninguna variable "intermedia", es decir, si la variable `2min` cuenta el número de interrupciones del reloj, entonces sabremos que han transcurrido 2 minutos cuando dicha variable alcance el valor 2 × 60 × 18 (2 minutos × 60 segundos/minuto × 18 interrupciones/segundo). Eso sí, la variable `2min` únicamente debe actualizarse mientras el sistema está en el estado de *post-alarma*, pero no en ningún otro.
Así pues, en esta primera opción para el control del tiempo, las acciones a realizar en la rutina de atención a la interrupción del reloj son las siguientes:

- Acciones a realizar en la rutina de servicio a la interrupción del reloj:

Reloj.1) Actualización de la variable `2min` cada vez que interrumpe el reloj (1ª opción):

```
if (estado_automata == post-alarma)
   2min++;
```

En el autómata lo reflejaremos así:



Pero en algunas aplicaciones puede darse la circunstancia de que sea necesario controlar los segundos independientemente, debido a que cada segundo haya que hacer algo concreto. En ese caso, para controlar el paso de 1 segundo, contaremos el número de interrupciones de reloj que se vayan produciendo, para lo que utilizaremos una variable local auxiliar que denominaremos `tic`. Dicha variable la inicializaremos una única vez con el valor 0, cuando empiece a ejecutarse nuestro programa, la primera vez que interrumpa el reloj; para ello, en la rutina de atención a la interrupción del reloj, escribiremos lo siguiente: `static int tic = 0`. Gracias a esa variable auxiliar, sabremos que ha transcurrido un segundo cuando transcurran 18 `tic`, es decir, cuando la variable `tic` alcance el valor 18. A partir de ahí, podremos contabilizar los segundos y los minutos, y así detectar cuándo han transcurrido 2 minutos. Pero hacerlo así alarga el código que hay que escribir y para evitarlo actualizaremos el valor de la variable `2min` una vez cada segundo, sin emplear más variables intermedias. Haciéndolo así, sabremos que han transcurrido 2 minutos cuando la variable `2min` alcance el valor 2 × 60, ya que la estamos "midiendo" en segundos (2 minutos × 60 segundos/minuto).
Así, en esta segunda opción, las acciones a realizar en la rutina de atención a la interrupción del reloj son las siguientes:

- Acciones a realizar en la rutina de servicio a la interrupción del reloj:

Reloj.1) Actualización de la variable `2min` una vez por segundo (2ª opción):

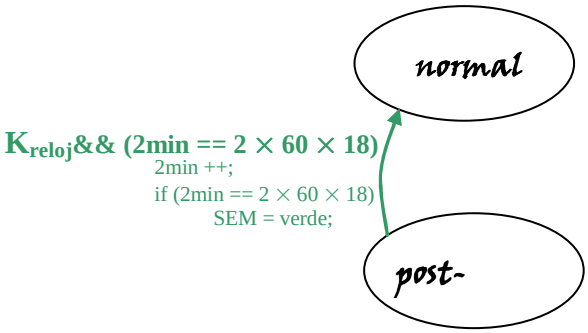
```
static int tic = 0;
tic++;
```

```
if (tic == 18)
 {
   tic = 0;
   if (estado_automata == post-alarma)
        2min++;
 }
```

Pero esta segunda opción acarrea un problema específico que pasamos a analizar ahora. Como ya hemos indicado, inicializaremos la variable `tic` al valor 0 al principio de la ejecución del programa, y luego cada segundo transcurrido también (cuando sea `tic` = 18, haremos nuevamente `tic` = 0, para volver a contar el número de interrupciones del reloj, otra vez hasta 18). Hay que tener claro que esta nueva inicialización no podemos hacerla en un momento arbitrario, ya que, en ese caso, haríamos que el control de los segundos resultara incorrecto y, además, aleatorio (es decir, si inicializáramos la variable `tic` en cualquier momento de la ejecución del programa, entonces el segundo que ya se estaba computando en ese momento podría durar 20 `tic`, no 18, si al poner la variable a 0 ya llevaba contadas dos interrupciones anteriores, o incluso podría durar hasta 35 `tic`, en el caso peor de que inicializáramos el valor de la variable `tic` cuando ya valía 17). Por ello, cuando queramos controlar un intervalo de tiempo determinado —2 minutos, por ejemplo—, no inicializaremos la variable `tic` en el preciso instante en que haya que comenzar a contar el intervalo de tiempo que nos interesa, sino que inicializaremos nuestra variable particular —la variable `2min`, por ejemplo—. ¿Qué puede ocurrir entonces? Pues cabe la posibilidad de que, cuando inicialicemos nuestra variable `2min`, "pillemos" a la variable `tic` con el valor 17, por ejemplo. Así, cuando se ejecute de nuevo la rutina de atención a la interrupción, se incrementará ese valor en 1, pasando a ser `tic` = 18, por lo que, como ya se ha producido la transición de estado correspondiente, se cumplirá la condición necesaria para actualizar nuestra variable `2min`, incrementándose en 1 su valor, como si ya hubiera transcurrido 1 segundo completo desde que había sido inicializada, mientras que únicamente ha transcurrido una mínima fracción del mismo: sólo 1 `tic`! Eso implica que, si utilizamos esta segunda opción, al computar el intervalo de tiempo deseado se puede producir un error de hasta casi 1 segundo, aleatoriamente, sin ningún tipo de control (este error no se puede producir en la primera opción, debido a que la variable de control `2min` no depende en absoluto de la variable `tic`). Por esta razón, cuando se deben controlar intervalos de tiempo relativamente pequeños, esta segunda opción no resulta adecuada, porque el valor relativo del error puede llegar a ser excesivamente elevado.

- Transiciones posibles:

a) Como hemos visto, una vez transcurrido el intervalo de 2 minutos sin que en la zona controlada hayan vuelto a entrar aves, el sistema deberá pasar al estado ***normal*** (como acabamos de decir, para la rutina de atención al reloj únicamente tomaremos en cuenta la primera opción analizada). En esta transición, el semáforo deberá ponerse en verde, para indicarle al avión que puede maniobrar sin riesgos.



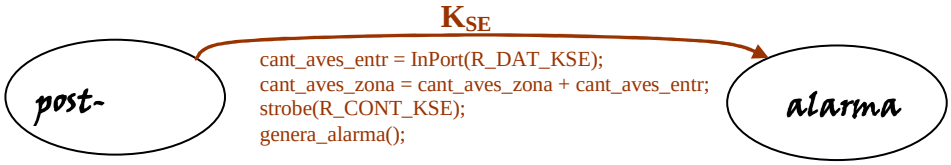
Y en la rutina de servicio a la interrupción, esta transición de estado quedará como sigue:

Reloj.2) Transición de estado, si es necesario, es decir, una vez transcurridos los 2 minutos (1ª opción):

```
if (2min == 2 × 60 × 18)
   {
        SEM = verde;
        estado_automata = normal;
   }
```

Recordemos que en la rutina de servicio a la interrupción del reloj, la primera acción que se debe realizar está condicionada por el estado del autómata, es decir, la variable `2min` sólo se actualizará si el sistema está en el estado *post-alarma*. Por esa razón, esta segunda acción también estará dentro de esa condición, es decir, el valor límite de la variable `2min` sólo se detectará en ese estado, no en ningún otro.

b) Pero si antes de transcurrido ese intervalo de 2 minutos vuelven a entrar aves en la zona controlada, entonces el sistema deberá volver al estado de *alarma*, hasta que de nuevo vuelvan a ser ahuyentadas todas las aves, para lo que será necesario volver a activar la alarma sonora.



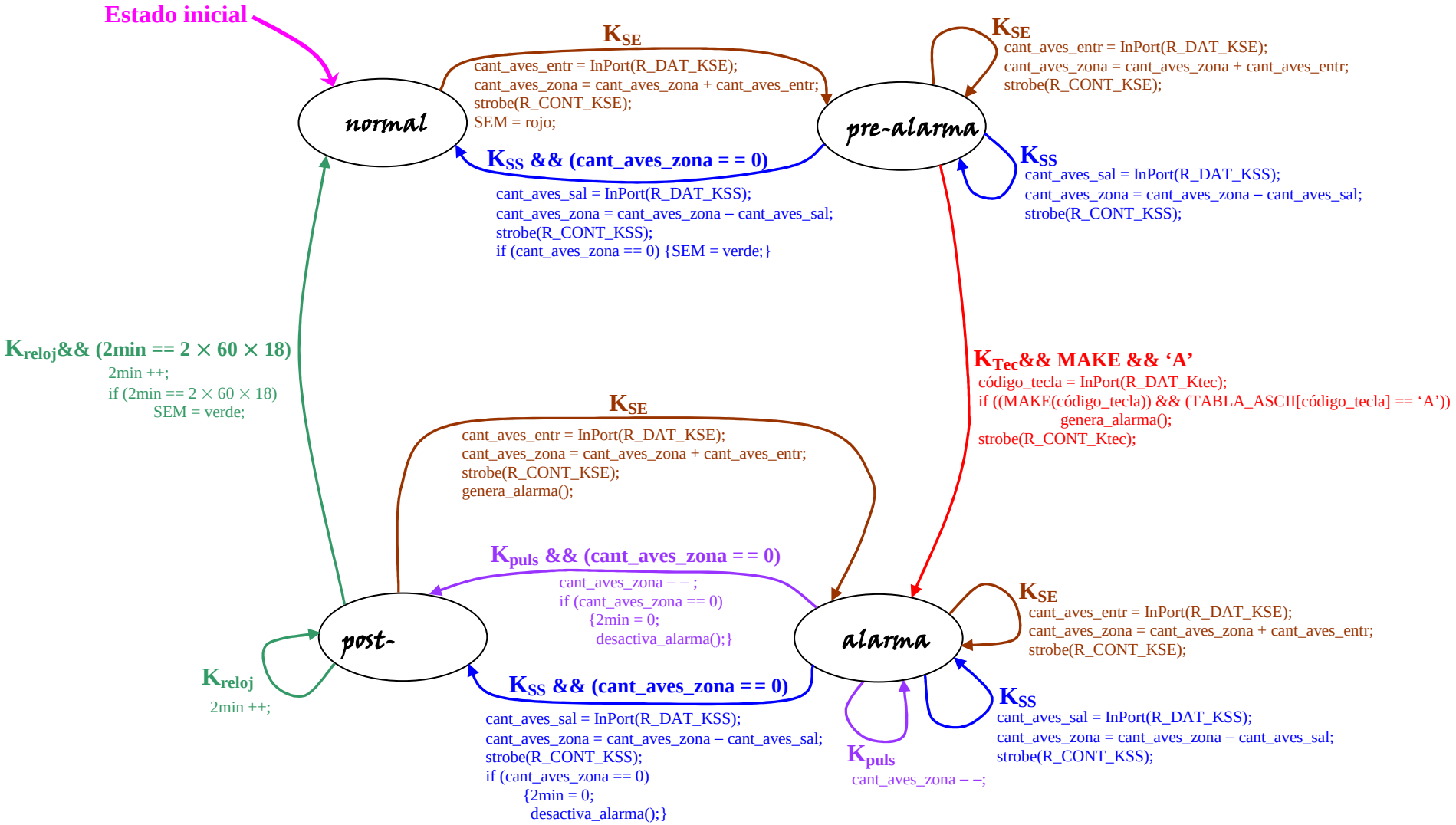
En consecuencia, la rutina de atención a la interrupción del sensor de entrada SE, deberá incluir esta posible transición de estado:

SE.5) Transición de estado, si es necesario:

```
if (estado_automata == post-alarma)
     {
          genera_alarma();
          estado_automata = alarma;
     }
```

Con esto hemos finalizado el análisis exhaustivo del funcionamiento del sistema y hemos definido tanto el autómata como las acciones a realizar en las rutinas de servicio a las interrupciones de los periféricos del sistema. Sólo nos queda, pues, unir todas las imágenes parciales del autómata en una figura global, y escribir el código del programa principal y de las rutinas de servicio a las interrupciones.

Teniendo en cuenta conjuntamente todos los casos posibles vistos, el autómata queda así:



Y ya sólo resta escribir el código. Tenemos que escribir el programa principal (en el que se ejecutará un bucle infinito) y las rutinas de atención a las interrupciones (RAI) de los periféricos: K_{SE}, K_{SS}, K_{tec}, K_{puls} y K_{reloj}.

↘ Programa principal.

En el programa principal se deben realizar determinada acciones, en concreto: inicialización de las variables (cuando sea necesario), modificación de algunos de los componentes del vector de interrupciones (al principio del programa) y recuperación de los valores iniciales de los mismos (al final del programa) —para asegurarnos de que se ejecutarán nuestras rutinas de servicio a las interrupciones, y no las de un hipotético sistema operativo—, y realización de un bucle infinito para que nuestro programa se ejecute de manera continua mientras no tenga que finalizar.

- Variables que hay que inicializar:

  Para empezar, supondremos que, cuando el sistema se pone en marcha, en la zona controlada no hay aves, y que el sistema estará en el estado *normal*. Debemos tener en cuenta, por tanto, que para asegurar el correcto funcionamiento del sistema es necesario saber en todo momento en qué estado se encuentra, para lo que utilizaremos una variable global, que denominaremos `estado_automata`, cuyo valor inicial será *normal*. Además, inicializaremos a 0 la variable que controla la cantidad de aves presentes en la zona bajo control, `cant_aves_zona`, y pondremos el semáforo en verde.

  Por otra parte, si para el control del reloj tomáramos en cuenta la segunda opción vista antes (es decir, contar las interrupciones del reloj para controlar el paso de los segundos), también tendríamos que inicializar la variable `tic`, para controlar correctamente en nuestro programa el paso del tiempo. Pero, como ya hemos visto, la variable `tic` se puede inicializar en la propia rutina de servicio a la interrupción del reloj, haciendo `static int tic = 0`, o bien al principio del programa principal.

  En resumen, las inicializaciones que hay que hacer son:

  - ✓ `estado_automata = normal`
  - ✓ `cant_aves_zona = 0`
  - ✓ `SEM = verde`
  - ✓ `tic = 0` (únicamente en la segunda opción vista, y si no hacemos `static int tic = 0` en la rutina de atención al reloj)

  Conviene subrayar que no es necesario inicializar el resto de variables utilizadas, ya que se inicializan con el valor adecuado en el preciso momento en que van a ser utilizadas. Por ejemplo, la variable `2min` se inicializa con el valor 0 en el instante en que el sistema pasa del estado de *alarma* al estado de *post-alarma*.

- Componentes del vector de interrupciones que hay que modificar:

  Si se emplea un PC para controlar el sistema, es necesario modificar los componentes del vector de interrupciones correspondientes a los periféricos del sistema, para que el PC realice el control que nosotros deseamos, es decir, para que, cuando interrumpa uno de los periféricos, se ejecuten las rutinas de servicio que escribamos nosotros, no las que tiene programadas el sistema operativo.

Así, en este caso concreto, los componentes que hay que modificar son los siguientes: el del reloj (entrada 0x1C del vector de interrupciones), el del teclado (entrada 0x09), el del sensor SE (entrada 0x0A), el del sensor SS (entrada 0x0B) y el del pulsador (entrada 0x0C). Para ello, supondremos que disponemos de una función ya escrita, que toma como parámetros las entradas del vector de interrupciones que hay que modificar:

- ✓ `CambiaVI(0x1C, 0x09, 0x0A, 0x0B, 0x0C)`

De la misma manera, cuando finalice la ejecución de nuestro programa, será necesario recuperar los valores originales de esos componentes, para que el sistema operativo tome de nuevo el control del PC:

- ✓ `RecuperaVI(0x1C, 0x09, 0x0A, 0x0B, 0x0C)`

- Control del bucle:

  Debemos asegurarnos de que nuestro programa se ejecuta de manera continua, mientras no le indiquemos que debe finalizar su ejecución. Para ello, pondremos un bucle, que se repetirá una y otra vez mientras no haya que dar por finalizada la ejecución del programa. En este caso concreto, releyendo el enunciado, no está previsto que se pueda finalizar la ejecución de nuestro programa, por lo que emplearemos un bucle infinito:

  - ✓ `while (true)` o `while (1)`

  Pero, en general, si se debiera finalizar la ejecución del programa, podríamos utilizar una variable global para controlarlo:

  - ✓ `while (!fin)`

  Así, el valor inicial de la variable `fin` sería 0 y el bucle se ejecutaría hasta que pasara a ser `fin = 1`, momento en el que finalizaría la ejecución del programa.
  Esto mismo se podría hacer utilizando un estado particular del autómata, y no una variable específica:

  - ✓ `while (estado_automata != último_estado)`

- Cuerpo del bucle:

  Dentro del `while` anterior, deberemos escribir el cuerpo del programa principal de nuestra aplicación concreta, es decir, las acciones que debe realizar nuestro programa.
  En general, en ese bucle se suelen realizar las encuestas de todos aquellos periféricos que se sincronizan por encuesta, leyéndose el contenido del registro de estado de cada periférico para detectar cuándo está listo para transferir un dato o para realizar la acción que tenga que realizar. Programando la encuesta dentro del bucle, nos aseguramos de que las acciones correspondientes se ejecutarán una y otra vez cuando el periférico esté preparado, mientras no finalice la ejecución del programa.
  Como en este ejercicio todos los periféricos se sincronizan por interrupción, el cuerpo del bucle estará vacío, debido a que el programa principal no tiene que llevar a cabo ninguna tarea en particular, realizándose todas las acciones en las rutinas de atención a las interrupciones de los periféricos. En consecuencia, el bucle quedará como sigue:

```
while (true);
```

El punto y coma (;) final indica que mientras se cumpla la condición del bucle no se hace nada, ya que no hay ni una sola sentencia en el cuerpo del bucle.

Resumiendo, el código del programa principal queda así:

```
void main()
{
    //inicialización de variables
 estado_automata = normal;
 cant_aves_zona = 0;
 SEM = verde;
 tic = 0;  // (sólo en la segunda opción del reloj, y si no se hace static)
    //modificación de los componentes del vector de interrupciones
 CambiaVI(0x1C, 0x09, 0x0A, 0x0B, 0x0C);
    //bucle principal
    while (true);  //dentro del bucle no se hace nada
    //recuperación de los componentes del VI al finalizar el programa
 RecuperaVI(0x1C, 0x09, 0x0A, 0x0B, 0x0C);
 }
```

- Rutina de atención a la interrupción (RAI) del controlador K_{SE}.

Ya hemos visto cuáles son las acciones a realizar en esta rutina de servicio. Ahora sólo tenemos que ordenarlas:

```
void interrupt RAI_KSE()
{
    //leer la cantidad de aves que acaba de entrar en la zona controlada
  cant_aves_ent = InPort(R_DAT_KSE);
    //actualización de la cantidad de aves presentes en la zona controlada
  cant_aves_zona = cant_aves_zona + cant_aves_ent;
    // secuencia de "strobe" sobre el registro de control
  strobe(R_CONT_KSE);
    //transiciones de estado pertinentes
  if (estado_automata == normal)
   {
    SEM = rojo;
    estado_automata = pre-alarma;
   }

  if (estado_automata == post-alarma)
   {
    genera_alarma();
    estado_automata = alarma;
   }
  Eoi();
  IRET;
}
```

Conviene recalcar que el controlador K_{SE} puede generar una petición de interrupción en cualquier estado (véase el autómata); por ello, las primeras acciones (`InPort`, `Strobe`) se ejecutan siempre que se produzca la interrupción, independientemente del estado en que se encuentre el sistema.

➢ Rutina de atención a la interrupción (RAI) del controlador K_{SS}.

```
void interrupt RAI_KSS()
{
      //leer la cantidad de aves que acaba de salir de la zona controlada
 cant_aves_sal = InPort(R_DAT_KSS);
      // actualización de la cantidad de aves presentes en la zona controlada
 cant_aves_zona = cant_aves_zona - cant_aves_sal;
      //secuencia de "strobe" sobre el registro de control
 strobe(R_CONT_KSS);
      //transiciones de estado pertinentes
 if (cant_aves_zona == 0)
   {
       if(estado_automata == pre-alarma)
            {
               SEM = verde;
               estado_automata = normal;
            }
       else if (estado_automata == alarma)
            {
               2min = 0;
               desactiva_alarma();
               estado_automata = post-alarma;
            }
    }
 Eoi();
 IRET;
}
```

En el caso del controlador K_{SS} no ocurre lo que ocurría con el controlador K_{SE}, ya que K_{SS} no puede realizar una petición de interrupción en cualquier estado (véase el autómata), debido a que, cuando el sistema está en el estado ***normal*** o en el de ***post-alarma***, no hay aves en la zona controlada, por lo que es imposible que el controlador K_{SS} solicite una interrupción para indicar que han salido aves de la zona controlada. Pero si pensáramos que el funcionamiento del sensor no es completamente fiable, podríamos poner una condición en esta RAI, para que algunas de las acciones (la actualización de la variable, por ejemplo) únicamente se ejecutaran en unos estados determinados, y no en otros (eso es lo que hemos hecho en el caso del teclado y del pulsador, dado que en estos dos casos pudiera darse la circunstancia de que el usuario del sistema pulsara una tecla o el pulsador involuntariamente, y si no se pusiera la condición, entonces el sistema presentaría un funcionamiento incorrecto). Así pues, si quisiéramos asegurar el funcionamiento correcto del sistema en cualquier circunstancia, quedaría como sigue:

```
if ((estado_automata == pre-alarma)||( estado_automata == alarma))
  {  cant_aves_zona = cant_aves_zona - cant_aves_sal; }
```

Es decir, si añadimos esa condición, las variable sólo se actualizará en los estados de ***pre-alarma*** o de ***alarma***, aunque la interrupción pueda producirse también en otros estados.

- Rutina de atención a la interrupción (RAI) del controlador del teclado, K_{tec}.

```
void interrupt RAI_Ktec()
{
//leer el código de posición de la tecla pulsada
 código_tecla = InPort(R_DAT_Ktec);
//secuencia de "strobe" sobre el registro de control
 strobe(R_CONT_Ktec);
//transición de estado pertinente
 if ((MAKE(código_tecla))&&(TABLA_ASCII[código_tecla]=='A')
    &&(estado_automata == pre-alarma))
   {
     genera_alarma();
     estado_automata = alarma;
    }
 Eoi();
 IRET;
}
```

En este caso, tanto la lectura del registro de datos como la secuencia de `strobe` se realizan en todos los estados, pero sólo se le hace caso al teclado si la tecla pulsada ha sido la A y el sistema está en el estado de ***pre-alarma***.

- Rutina de atención a la interrupción del controlador K_{puls}.

```
void interrupt RAI_Kpuls()
{
       //actualización de la cantidad de aves presentes en la zona controlada
  if (estado_automata == alarma)
     {
       cant_aves_zona --;
      // transición de estado pertinente
       if (cant_aves_zona == 0)
       {
         2min = 0;
         desactiva_alarma();
         estado_automata = post-alarma;
       }
     }
  Eoi();
  IRET;
 }
```

También en este caso, las acciones correspondientes sólo se ejecutan si el sistema está en el estado de ***alarma***, no teniéndose en cuenta las interrupciones del pulsador en cualquier otro estado, en cuyo caso sólo se ejecutarían `Eoi()` e `IRET`.

- Rutina de atención a la interrupción del controlador del reloj K_{reloj} (1ª opción).

```
void interrupt RAI_Kreloj()
{
       //actualización de la variable 2min en cada interrupción del reloj
 if (estado_automata == post-alarma)
  {
    2min++;
// transición de estado pertinente
    if (2min == 2 × 60 × 18)
    {
      SEM = verde;
      estado_automata = normal;
    }
  }
 IRET;
}
```

También en este caso, las acciones correspondientes a esta rutina sólo se ejecutan si el sistema está en el estado de ***post-alarma***, no haciéndosele caso al reloj en cualesquiera otros estados, en los que sólo se ejecutará IRET.

- Rutina de atención a la interrupción del controlador del reloj K_{reloj} (2ª opción).

```
void interrupt RAI_Kreloj ()
{
 static int tic = 0;
      //control de los segundos transcurridos
 tic++;
 if (tic == 18)
      {
       tic = 0;
             // actualización de la variable 2min una vez por segundo
        if (estado_automata == post-alarma)
        {
           2min++;
             // transición de estado pertinente
           if (2min == 2 × 60)
         {
              SEM = verde;
              estado_automata = normal;
         }
        }
      }
 IRET;
}
```

Ya hemos indicado cuál puede ser el problema asociado a esta segunda opción: si cuando inicializamos a 0 la variable `2min` "pillamos" a la variable `tic` con el valor 17, entonces la variable `2min` incrementará en uno su valor mucho antes de que haya transcurrido 1 segundo. Por esa razón dejamos de lado esta opción.