7. Implementación de los temporizadores/contadores: ATMegaClassic

7.1. Introducción

El archivo ***ATMegaClassic.java*** encapsula parte de la funcionalidad común que comparten los micros de la familia ATMega del fabricante Atmel. En concreto, implementa los diferentes tipos de temporizadores/contadores que se soportan en estos dispositivos. Basados en una señal de sincronismo (bien sea la propia del sistema o una externa), estos módulos se utilizan para medir intervalos de tiempo. Son capaces de generar interrupciones para denotar que ha transcurrido cierto período y, a la vez, generar pulsos que puedan ser empleados para controlar otros elementos internos o externos al microcontrolador. Dentro de la gama ATMega, se definen dos tipos fundamentales sobre los que se apoyan los distintos temporizadores/contadores implementados en cada MCU: de 8 y de 16 bits. La única diferencia entre ellos es el rango de operación que, evidentemente, es mayor para el caso de 16 bits que para el de 8 bits. Para el caso del ATMega1281 existen cuatro dispositivos basados en los anteriores: Timer0 (8 bits), Timer1 (16 bits), Timer2 (8 bits) y Timer3 (16 bits). Aunque cada uno de ellos posee características propias, comparten la mayoría de la funcionalidad básica: subsistemas integrados, modos de operación, etc.

En este capítulo se hace una presentación de los cuatro tipos de temporizadores/contadores que se incluyen en el ATMega1281 [16] (características, registros, modos de operación, etc.). A continuación, se indican las principales razones por las que ha habido que realizar modificaciones sobre ***ATMegaClassic.java*** para que se adapte al propósito de este proyecto. Por último, se presenta el código incluído en este archivo. En él se definen, en primer lugar, los dos tipos básicos y, posteriormente, se implementan cada uno de los temporizadores/contadores que incluye el ATMega1281.

7.2. Descripción de los temporizadores/contadores del ATMega1281

7.2.1. Timer0

Este elemento es un módulo temporizador/contador de 8 bits de propósito general con dos unidades de comparación independientes y soporte para la modulación de ancho de pulso. Permite la ejecución de programas con precisión (gestión por eventos) y generación de pulsos.

Su funcionamiento está determinado por una serie de registros que controlan todos sus parámetros: modo de operación, cuenta actual, interrupciones, etc. Entre ellos cabe destacar:

- ***TCNT0***: Registro de 8 bits que se utiliza para llevar la cuenta del Timer0. El valor mínimo desde donde comienza es 0x00 y el máximo donde se resetea es 0xFF.
- ***OCR0A* y *OCR0B***: Son dos registros de 8 bits cuyo contenido se compara continuamente con el valor del contador (*TCNT0*). Una coincidencia entre *TCNT0 y OC0A* o *OC0B* se puede usar para provocar una interrupción y generar asimismo una forma de onda en el pin de salida correspondiente (*OC0A* o *OC0B*).
- ***TIMSK0***: Registro de máscara de interrupción. Gracias a él se pueden activar/desactivar las interrupciones asociadas al Timer0. Son tres las posibles interrupciones que se pueden generar y que están asociadas al Timer0: *TOIE0* (Interrupción de desbordamiento de cuenta del Timer0), *OCIE0A* (Interrupción de unidad de comparación A) y *OCIE0B* (Interrupción de unidad de comparación B).
- ***TIFR0***: Registro de banderas de interrupción. Cualquier interrupción que se produzca relacionada con el temporizador se refleja en él. Las tres posibles banderas son: *TOV0* (Bandera de desbordamiento de cuenta del Timer0), *OCF0A* (Bandera de unidad de comparación A) y *OCF0B* (Bandera de unidad de comparación B).
- ***TCCR0A* y *TCCR0B***: Registro de control del Timer0. Controla diferentes parámetros de configuración tales como el modo de operación, el valor máximo de cuenta y la forma de onda generada en el pin de salida *OC0A*.

El Timer0 está formado por una serie de bloques funcionales básicos que lo dotan de distintas capacidades. Dos de ellos son los más relevantes y se pasa a describirlos brevemente a continuación:

- **Unidad de cuenta**: Es la componente principal del Timer0. Dependiendo del modo de operación seleccionado, el contador se incrementa, decrementa

o reinicia con cada pulso de la señal de reloj que lo controla. Esta señal de reloj puede ser externa o interna y se elige a través de tres bits (*CS2:0*) del registro *TCCR0B* (registro de control B del Timer0). Si no se selecciona ninguna fuente de reloj, el temporizador se desactiva. La secuencia de conteo se determina mediante los bits *WGM00*, *WGM01* y *WGM02* que se encuentran en los registros *TCCR0A* y *TCCR0B*, bits que seleccionan entre los distintos modos de operación del contador. Las formas de onda generadas se presentan en los pines *OC0A* y *OC0B*. Cuando el temporizador se desborda, se genera la interrupción *TOV0*. Su configuración varía con el modo de operación escogido y puede ser utilizada como interrupción de la CPU.

- **Unidad de comparación**: En este módulo, el comparador de 8 bits está continuamente comparando el valor del registro *TCNT0* con el contenido de los registros de comparación *OCR0A* y *OCR0B*. Cuando se igualan, el comparador indica que se ha producido dicha coincidencia mediante las banderas *OCF0A* y *OCF0B*. Si sus respectivas interrupciones están activas, se generará una interrupción, que le puede servir a la CPU para conocer que se ha producido esta circunstancia y actuar en consecuencia. Asimismo, la activación de *OCF0A* u *OCF0B* sirve al generador de formas de onda para generar una señal de salida conforme al modo de operación establecido y el modo de comparación de salida.

El modo de operación, que es la manera en la que se comporta el dispositivo, se controla mediante una serie de bits contenidos en los registros *TCCR0A* y *TCCR0B*. Hay tres modos fundamentales de operación:

- **Modo normal (cuenta)**. En este modo, el contador siempre se está incrementando. En el momento en que llega al máximo valor posible (0xFF), se reinicia a su valor más bajo (0x00) y comienza de nuevo. En este punto, la bandera TOV0 se activa indicando que el contador se ha reiniciado y se genera, si se encuentra habilitada, la interrupción *TOIE0*. Como se ha comentado, el contador no detiene su cuenta en ningún momento.

- **Modo comparación**. El Timer0 posee dos unidades de comparación independientes, la A y la B. Son exactamente iguales, así que lo que se va a explicar para una, será equivalente para la otra. La explicación se va a centrar en la primera, la unidad A. En este modo, el registro *OCR0A* se emplea para manipular la resolución del contador. Cuando la cuenta (contenido de *TCNT0*) es igual al valor indicado en *OCR0A*, el contador se resetea. Esta situación se indica mediante la bandera *OCF0A* y se generará la interrupción correspondiente (si está permitida mediante el bit *OCIE0A*). Este modo de operación es muy útil para generar pulsos de una frecuencia determinada gracias a que es posible conmutar la salida de la unidad de comparación

(representada por el pin *OC0A*) cada vez que se produzca la igualdad de comparación anterior. La frecuencia máxima alcanzable mediante este método es $f_{clkI/O}/2$ (que es igual a la frecuencia a la que funciona el microcontrolador, por defecto, 1 MHz). El conjunto de valores posibles para la frecuencia de la señal de salida viene impuesto por la expresión (N es el factor de preescalado, que puede tomar como valores 1, 8, 64, 256 y 1024):

$$f_{OC0A} = \frac{f_{clkI/O}}{2 \cdot N \cdot (1 + OCR10A)} \qquad con\ N = 1, 8, 64, 256, 1024$$

- **Modo PWM**[1] Es una potente técnica usada para manejar circuitos analógicos con las salidas digitales de un microcontrolador. Se presentan dos variantes:

  - PWM rápido. Sirve para generar pulsos de alta frecuencia y distintos anchos de pulso. El contador se está incrementando continuamente desde su valor mínimo al máximo. Cuando el contador es igual al valor de *OCR0A*, si se encuentra en modo de comparación no invertido (*WGM2:0 = 3*), el pin de salida se pone a nivel bajo y se vuelve a poner a nivel alto cuando se resetea. En modo de comparación invertido (*WGM2:0 = 7*) ocurre todo lo contrario. La bandera e interrupción asociada se activan cuando la cuenta llega al máximo. La frecuencia de la señal que se puede generar en esta variante de este modo de operación es el doble de la que se consigue en la siguiente variante. Por este motivo, este modo es adecuado para regulación de potencia, rectificación y aplicaciones DAC. Posibles valores de la frecuencia de la señal de salida son los que resultan de la expresión (N es el factor de preescalado, que puede tomar como valores 1, 8, 64, 256 y 1024):

$$f_{OC0APWM} = \frac{f_{clkI/O}}{N \cdot 256} \qquad con\ N = 1, 8, 64, 256, 1024$$

  - PWM de fase correcta. El contador se incrementa desde su mínimo hasta el máximo para después volver a bajar al mínimo. Si se escoge el modo de comparación no invertido (*WGM2:0* = 1), el pin de salida de la unidad de comparación (*OC0A*) se pone a nivel bajo cuando se produce la igualdad entre *TCNT0* y *OCR0A* mientras que la cuenta se está incrementando, y a nivel alto cuando se produce dicha igualdad y la cuenta del contador es descendente. En el modo de comparación invertido (*WGM2:0* = 5) ocurre justo al contrario. La bandera e interrupción asociada se activan cuando la cuenta llega al máximo. Los valores de frecuencia que se pueden conseguir vienen dados por la siguiente expresión (N es el factor de preescalado, que puede tomar como valores 1, 8, 64, 256 y 1024):

$$f_{OC0APCPWM} = \frac{f_{clkI/O}}{N \cdot 510} \qquad con\ N = 1, 8, 64, 256, 1024$$

[1] Pulse Width Modulation (PWM): Modulación del ancho del pulso

Además de lo anterior, el dispositivo cuenta con un sistema de preescalado que adapta la frecuencia de la señal de reloj de entrada del temporizador/contador a otra frecuencia distinta que el temporizador/contador utilizará para su funcionamiento. Si se denomina a esta entrada $f_{clk\_I/O}$ (señal de reloj del sistema), se pueden obtener las frecuencias $f_{clk\_I/O}/8$, $f_{clk\_I/O}/64$, $f_{clk\_I/O}/256$ y $f_{clk\_I/O}/1024$.

7.2.2. Timer2

Es el otro temporizador/contador de 8 bits de los dos con los que cuenta el ATMega1281. Presenta las mismas características que el anterior: posee dos unidades comparadoras, generación de pulsos utilizando la técnica PWM, preescalador, etc. La diferencia que lo distingue de aquel es que es capaz de funcionar en un modo especial llamado Modo Asíncrono. Este modo consiste en que la fuente de sincronismo del Timer2 se obtiene a partir de un oscilador externo, de valor 32768 Hz, suministrado mediante los pines apropiados del microcontrolador en lugar de obtenerla de la señal de reloj del sistema.

Los registros asociados al Timer2 son análogos al del Timer0 con el cambio de nomenclatura en el que el "0" (indicando Timer0), se reemplaza por el "2" (indica Timer2):

- ***TCNT2***: Registro de cuenta del Timer2. El valor mínimo es 0x00 y el máximo 0xFF.
- ***OCR2A*** y ***OCR2B***: Registros de 8 bits utilizados para comparar su contenido con el valor del contador (*TCNT2*). Cuando se produzca coincidencia de ambos se generará una interrupción y un pulso en el pin de salida *OC2A* u *OC2B*, respectivamente.
- ***TIMSK2***: Registro de máscara de interrupción usado para activar/desactivar las interrupciones asociadas al Timer2: *TOIE2* (Interrupción de desbordamiento de cuenta del Timer2), *OCIE2A* (Interrupción de unidad de comparación A) y *OCIE2B* (Interrupción de unidad de comparación B).
- ***TIFR2***: Registro de banderas de interrupción: *TOV2* (Bandera de desbordamiento de cuenta del Timer2), *OCF2A* (Bandera de unidad de comparación A) y *OCF2B* (Bandera de unidad de comparación B).
- ***TCCR2A*** y ***TCCR2B***: Registro de control del Timer2 que establece el modo de operación, el valor máximo de cuenta y la forma de onda generada en el pin de salida *OC2A*.

Los modos de operación son los mismos que para el Timer0, con la inclusión del citado nuevo modo Asíncrono.

7.2.3. Timer1 y Timer3

Sendos módulos representan los dos temporizadores/contadores de 16 bits con los que cuenta el ATMega1281. Se emplean para controlar la ejecución precisa de programas (gestión por eventos) y generación de formas de onda. Cada uno de ellos integra tres unidades de comparación independientes (por dos de los anteriores), una unidad de captura de entrada y contador de eventos externos, soporte PWM y generador de frecuencias. Ambos dispositivos son idénticos por lo que, a continuación, sólo se describirá uno de ellos, en este caso, el Timer1.

El Timer1 se controla por una serie de registros asociados, que determinan desde el modo en el que opera dicho dispositivo hasta las interrupciones que pueden acaecer. Cabe destacar los siguientes:

- *TCNT1*: Se encarga de llevar la cuenta del temporizador/contador. El valor mínimo de la cuenta es 0x0000 y el valor máximo 0xFFFF.
- *OCR1A*, *OCR1B* y *OCR1C*: Registros usados por cada una de las tres unidades de comparación. Sus valores se comparan continuamente con la cuenta del temporizador/contador. Si se produce coincidencia, se genera la interrupción correspondiente y se actualiza el valor del pin de salida de dichas unidades.
- *TCCR1*: Representa el registro de control del Timer1. Controla, básicamente, el modo de operación en el que se encuentra el temporizador/contador. También indica el valor máximo de cuenta y en qué momento se genera la interrupción.
- *TIMSK1*: Registro de máscara de interrupción. Permite habilitar o deshabilitar las interrupciones asociadas al Timer1: TOIE1 (Interrupción de desbordamiento de cuenta del Timer1), *OCIE1A*/*OCIE1B*/*OCIE1C* (Interrupción de unidad de comparación A/B/C) y *ICIE1* (Interrupción de captura de entrada).
- *TIFR1*: Registro de banderas de interrupción: *TOV1* (Bandera de desbordamiento de cuenta del Timer1), *OCF1A*/*OCF1B*/*OCF1C* (Bandera de unidad de comparación A/B/C) y *ICF1* (Bandera de captura de entrada).

El modo de operación del Timer1, como se ha comentado, se controla mediante los bits del registro TCCR1. Consultando la hoja de características[2], el dispositivo presenta los mismos modos de funcionamiento que el caso de los temporizadores de 8 bits: modo normal, modo comparación y modo PWM. La diferencia estriba en que al tener 16 bits para la cuenta, el valor que se puede contabilizar es mayor que en el caso de los de 8 bits (65535 por 255).

[2]Hoja de características del ATMega1281, pag. 176

7.3. Modificaciones sobre ATMegaClassic.java

Como se expuso en la sección 1.1, el fichero ***«ATMegaClassic.java»*** contiene la implementación de los distintos tipos de temporizadores/contadores que se encuentran integrados en los microcontroladores de la familia ATMega de Atmel. La versión de dicho archivo incluído en Avrora da soporte a este tipo de dispositivos para una serie de MCU's concretas y que se encuentran, actualmente, anticuadas: ATMega32, ATMega169 y ATMega128. El problema radica en que, a la hora de implementar el microcontrolador objeto de este proyecto, sus temporizadores/contadores no se encuentran representados adecuadamente con los que se incluyen en tal fichero. Esto se debe a que existen diferencias muy significativas entre ellos, por lo que habrá que modificar el código desarrollado para que se ajusten al nuevo micro ATMega1281.

Tomando como referencia el conjunto de temporizadores/contadores del ATMega128 (antecesor del ATMega1281) y comparando, se observan las siguientes disparidades:

- Respecto al Timer0, en la versión antigua dispone de sólo una unidad de comparación por las dos del nuevo micro. Además, incluye el modo de operación asíncrono, mientras que en el nuevo, esta opción no se incluye.
- En cuanto al Timer2, es éste quién, en la nueva MCU, posee el modo de operación asíncrono en detrimento del Timer0. Asimismo, posee dos unidades de comparación frente a su antecesor.

Es importante que el código desarrollado represente, de la manera más fiel posible, a los dispositivos físicos del nuevo microcontrolador para que, posteriormente en la simulación, se pueda obtener un comportamiento más cercano al real. Para ello, hay que examinar escrupulosamente todas estas divergencias y adaptar el código ya creado a ellas o, si llegara el caso, desarrollar nuevas clases que encapsulen todas estas nuevas características. Esto es lo que se ha realizado en este proyecto. Se han creado varias clases nuevas, que son adaptaciones de las ya existentes, con el fin de lograr la completa integración de los temporizadores/contadores del ATMega1281.

7.4. Implementación de los temporizadores/contadores

El archivo ***ATMegaClassic.java*** contiene una serie de clases creadas con el objetivo de dar soporte dentro del simulador a los distintos tipos de temporizadores/contadores que integran ciertos microcontroladores de la familia ATMega de Atmel. Analizando detenidamente su contenido, en primer lugar se observa que se crean las dos clases básicas que encapsulan los dos tipos de contadores

fundamentales que ya fueron introducidos en la sección 7.1: de 8 y de 16 bits. Una vez hecho esto, se crea una clase por cada tipo de dispositivo real que integra la MCU, es decir, en el caso del ATMega1281, Timer0, Timer1, Timer2 y Timer3. Estas clases contendrán características de personalización propias a cada elemento y que no han sido incluídas en la definición de las clases base. En las siguientes subsecciones se presentarán los conceptos citados uno a uno y se describirá brevemente el código que incluye cada una de ellas.

7.4.1. Temporizador/contador de 8 bits

Como se introdujo en la sección 7.1, es uno de los dos tipos fundamentales sobre el que se apoyan determinados contadores que el dispositivo físico (MCU) lleva incorporado. Concretamente, para el caso del ATMega1281, son el Timer0 y el Timer2 quienes se sustentan sobre él. Su implementación engloba todas aquellas características comunes a todos los tipos que dependan de él.

La clase creada se denomina ***«Timer8Bit»***. Hereda de la clase ***«ATMega-Timer»*** que representa y desarrolla el concepto de «Temporizador/Contador» para los microcontroladores de la serie ATMega. De ésta es de quien hereda los conceptos y estructuras fundamentales.

Pasando a describir la clase, y comenzando por su constructor, se puede ver que tiene el siguiente formato:

```
Timer8Bit(int n, AtmelMicrocontroller m, int[] periods,
          String ovfName, String acfName)
```

Contiene cinco parámetros que lo identifican unívocamente:

- *n:* Identificador numérico.
- *m:* Objeto de tipo ***«AtmelMicrocontroller»*** representando a la MCU.
- *periods:* Factor de preescalado de la frecuencia principal de funcionamiento (frecuencia del sistema).
- *ovfName:* Interrupción asociada al desbordamiento del contador.
- *acfName:* Interrupción asociada a la unidad de comparación.

Para representar la funcionalidad completa del registro *TCNTn* que, como se vió en la sección 7.2, se encarga de llevar la cuenta propiamente del temporizador, es necesario crear un objeto de tipo ***«ActiveRegister»***. Este tipo de registros tienen un objetivo: cualquier escritura o lectura realizada en dichos registros desencadena alguna acción dentro de la propia simulación (usualmente, este tipo de registros controlan la funcionalidad de un determinado elemento hardware, como ocurre en este caso). Por este motivo, es muy adecuado para encapsular a *TCNTn*.

```
TCNTn_reg = new TCNTnRegister("TCNT" + n, m.getIOReg("TCNT" + n));
m.installIOReg(TCNTn_reg.name, TCNTn_reg);
```

Como se puede observar, solo es necesario pasarle el nombre completo del registro y asociarlo a la instancia del microcontrolador.

El siguiente paso es crear la unidad de comparación que se integra dentro del temporizador. Para ello, se hace uso de una nueva clase que se utiliza para crear objetos que representan a este tipo de módulo y todas sus peculiaridades. Esta clase se denomina ***«OutputCompareUnit»***. En ella se especifican varios elementos:

- Registro asociado ($OCRn$).
- El pin de salida de la unidad de comparación (OCn).
- Interrupción asociada.

```
AtmelMicrocontroller.Pin pin = (AtmelMicrocontroller.Pin)
                                m.getPin("OC" + timerNumber);
int interrupt = m.properties.getInterrupt(acfName);
addComparator(Comparator._, new OutputCompareUnit(n, m, Comparator._,
              interrupt, pin));
```

Una vez definida la estructura interna, se deben especificar los posibles modos de funcionamiento en los que es posible trabajar. Como se introdujo en la sección 7.2, son cuatro estos posibles modos: Normal, Modo comparación, PWM rápido y PWM de fase correcta. Todos ellos se encuentran ya desarrollados en ***«ATMegaTimer»***.

```
modes = new Mode[]{
   new Mode(Mode.NORMAL.class, null, FixedTop.FF),
   new Mode(Mode.FC_PWM.class, null, FixedTop.FF),
   new Mode(Mode.CTC.class, m.getRegisterSet().getField(ocfn),
            m.getIOReg(ocrn)),
   new Mode(Mode.FAST_PWM.class, null, FixedTop.FF)};
```

Para finalizar, la clase presenta una serie de métodos útiles para ciertas operaciones usuales, por ejemplo, obtener la cuenta actual del contador, el máximo de la cuenta, resetear el modo de funcionamiento, etc.

7.4.2. Creación del nuevo temporizador/contador de 8 bits

En esta subsección se va a presentar la clase desarrollada para cubrir todas las funcionalidades y características del contador de 8 bits que el ATMega1281 trae integrado. Esta clase se basa en la que ha sido presentada en 7.4.1, haciendo las pertinentes modificaciones para cumplir los objetivos marcados.

Antes de comenzar, se debe tener claro lo que se va a modificar. Como se introdujo en 1.3, la diferencia fundamental entre la versión antigua del dispositivo de 8 bits y la moderna es que incorpora una unidad de comparación adicional. Evidentemente se le debe dar el soporte adecuado. Esto exige la creación de todos los registros asociados, pines de salida, etc. En lo que resta de subsección se presentan estos cambios realizados en el código.

A la clase creada se le ha dado el nombre de **«NewTimer8Bit»**, para indicar explícitamente que representa la nueva versión de contador de 8 bits. Su constructor tiene la forma:

```
NewTimer8Bit(int n, AtmelMicrocontroller m, int[] periods,
             String ovfName, String acfName1, String acfName2)
```

donde:

- *n:* Identificador numérico.
- *m:* Objeto de tipo «AtmelMicrocontroller» representando a la MCU.
- *periods:* Factor de preescalado de la frecuencia principal de funcionamiento (frecuencia del sistema).
- *ovfName:* Interrupción asociada al desbordamiento del contador.
- *acfName1:* Interrupción asociada a la unidad de comparación 1.
- *acfName2:* Interrupción asociada a la unidad de comparación 2.

Aquí aparece la primera diferencia con el caso anterior. Como existen dos unidades de comparación, al constructor se le debe proporcionar el nombre de la interrupción asociada a cada una de ellas para que lo asigne a la unidad correspondiente.

Las siguientes líneas de código se centran en crear las propias unidades y todo los elementos asociados. En primer lugar se define el pin de salida de cada unidad:

```
pinA = (AtmelMicrocontroller.Pin) m.getPin("OC" + timerNumber + "A");
pinB = (AtmelMicrocontroller.Pin) m.getPin("OC" + timerNumber + "B");
```

donde «timerNumber» representa el identificador númerico del módulo contador que se base en éste para su creación (0 para el Timer0, 1 para el Timer1, etc). A continuación se obtienen los números de las interrupciones que han sido pasadas antes como parámetros:

```
int interrupt1 = m.properties.getInterrupt(acfName1)
int interrupt2 = m.properties.getInterrupt(acfName2);
```

Una vez hecho esto, ya es posible crear ambas unidades. Para ello, es necesario indicarle varios parámetros: temporizador/contador del que formará parte, un objeto de tipo microcontrolador que incluirá a éste, el nombre de la unidad y el pin de salida y su interrupción asociada:

```
addComparator(Comparator.A, new OutputCompareUnit(n, m,
              Comparator.A, interrupt1, pinA))
addComparator(Comparator.B, new OutputCompareUnit(n, m,
              Comparator.B, interrupt2, pinB))
```

Para terminar, se deben especificar los modos de operación del dispositivo. Éstos son idénticos a los del Timer8Bit. Sólo hay que tener en cuenta que, ahora, como se dispone de dos unidades de comparación, estos modos estarán duplicados. De igual forma, los registros asociados a estos módulos y que intervienen directamente en los modos de operación se indican mediante su nombre:

```
modes1 = new Mode[]{
   new Mode(Mode.NORMAL.class, null, FixedTop.FF),
   new Mode(Mode.FC_PWM.class, null, FixedTop.FF),
   new Mode(Mode.CTC.class, m.getRegisterSet().getField(ocfna),
            m.getIOReg(ocrna)),
   new Mode(Mode.FAST_PWM.class, null, FixedTop.FF)
};
modes2 = new Mode[]{
   new Mode(Mode.NORMAL.class, null, FixedTop.FF),
   new Mode(Mode.FC_PWM.class, null, FixedTop.FF),
   new Mode(Mode.CTC.class, m.getRegisterSet().getField(ocfnb),
            m.getIOReg(ocrnb)),
   new Mode(Mode.FAST_PWM.class, null, FixedTop.FF)
};
```

7.4.3. Temporizador/contador de 16 bits

El segundo tipo de contador dentro de la gama de MCU's ATMega es el de 16 bits. Al igual que el de 8 bits, proporciona la estructura básica sobre el cual se apoyan otra serie de temporizadores/contadores que se incluyen en el ATMega1281. Concretamente, estos elementos son el Timer1 y Timer3. La diferencia fundamental

entre el de 8 bits y el de 16 bits, aparte del rango de resolución de la cuenta, es que en este tipo de temporizadores se incluyen tres unidades de comparación (por dos del caso de 8 bits) y una nueva unidad, la unidad de captura de entrada. Respecto a su implementación software, dentro de Avrora se ha desarrollado la clase ***«Timer16Bit»*** que define todos los subsistemas internos que son comunes a este tipo de dispositivos. Después, en el momento de la implementación de cada temporizador real del microcontrolador, cada uno de ellos añadirá sus propias características, pero partiendo de la base desarrollada en ***«Timer16Bit»***.

El código incluido en la citada clase es prácticamente idéntico al de ***«Timer8Bit»***. Sólo son necesarios aquellos cambios requeridos para proporcionar el soporte adecuado para los nuevos subsistemas que se han añadido. Todo ello se describe a continuación.

En primer lugar, el constructor de la clase:

```
Timer16Bit(int n, AtmelMicrocontroller m, int[] periods,
           String ovfName, String[] cfn)
```

Es similar al de 8 bit. Posee el identificador del temporizador, un objeto que representa al microcontrolador del que formará parte, el conjunto de factores de preescalado de la frecuencia principal y el nombre de la interrupción de desbordamiento del contador. El último parámetro es el que cambia. Es un array de cadenas que contiene el nombre de cada una de las posibles interrupciones que se pueden producir (sin contar la de desbordamiento, que ya ha sido incluida) y que se originan directamente por este dispositivo. Según se introdujo en la sección 1.2, existirán cuatro interrupciones:

- Una por cada unidad de comparación.
- Una relativa a la unidad de captura de entrada.

El siguiente cambio destacable es relativo al propio hecho de que es un contador de 16 bits, en lugar de 8 como se ha visto en secciones anteriores. Como los registros internos del microcontrolador tienen un tamaño de 8 bits, son necesarios dos registros para almacenar la cuenta actual del dispositivo. Uno de ellos contendrá la parte alta (bit 15 - bit 8) y el otro la parte baja (bit 7 - bit 0). Estos dos registros son, respectivamente, *TCNTnH* y *TCNTnL*.

```
TCNTn_reg = new RW16Register();
TCNTnH_reg = (HighRegister) m.installIOReg("TCNT" + n + "H",
              new HighRegister(TCNTn_reg))
TCNTnL_reg = new TCNTnRegister("TCNT" + n + "L",
             new LowRegister(TCNTn_reg))
```

El registro se define como un registro de 16 bits (clase ***«RW16Register»***) y se crean cada uno de los subregistros (*TCNTnH_reg* y *TCNTnL_reg*).

A continuación, se procede a la integración de las unidades de comparación:

```
addComparator(Comparator.A, new OutputCompareUnit(n, m, Comparator.A,
              m.properties.getInterrupt(cfn[1]),
              (Pin) m.getPin("OC" + timerNumber + "A")))
addComparator(Comparator.B, new OutputCompareUnit(n, m, Comparator.B,
              m.properties.getInterrupt(cfn[2]),
              (Pin) m.getPin("OC" + timerNumber + "B")))
addComparator(Comparator.C, new OutputCompareUnit(n, m, Comparator.C,
              m.properties.getInterrupt(cfn[3]),
              (Pin) m.getPin("OC" + timerNumber + "C")))
```

El proceso para su inclusión en el modelo es muy parecido al de 8 bits. Se llama al constructor de la clase ***«OutputCompareUnit»*** (que implementa este tipo de elemento hardware) con los parámetros adecuados, que son:

- Identificador numérico del temporizador.
- Un objeto de tipo «Microcontroller» representando a la MCU que lo integra.
- Nombre de la unidad de comparación.
- Número de la interrupción asociada a este elemento.
- Pin de salida de la unidad.

Para el caso de la unidad de captura de entrada es análogo:

```
addComparator(Comparator.I, new InputCompareUnit(n, m, Comparator.I,
              m.properties.getInterrupt(cfn[0]),
              (Pin) m.getPin("ICP" + timerNumber)))
```

En esta ocasión, se hace uso de la clase ***«InputCompareUnit»***, que implementa la unidad de entrada. Recibe los mismos parámetros que en el caso anterior con la diferencia de que el pin es de entrada en lugar de salida.

Una vez añadidos estos módulos, es necesario especificar los diferentes modos de funcionamiento del dispositivo. Como ya se vio en la subsección 1.2.3, existen cinco modos fundamentales: normal, reseteo del contador por igualdad en comparación, PWM rápido, PWM corrector de fase y PWM corrector de fase y frecuencia. Cada uno de estos modos posee versiones diferentes según el valor máximo de dicho modo, que puede ser un valor fijo u otro marcado, o bien la primera unidad de comparación, o bien, la unidad de captura de entrada.

```
modes = new Mode[] {
   new Mode(Mode.NORMAL.class, null, FixedTop.FFFF),
   new Mode(Mode.PWM.class, null, FixedTop.FF),
   new Mode(Mode.PWM.class, null, FixedTop._1FF),
   new Mode(Mode.PWM.class, null, FixedTop._3FF),
   new Mode(Mode.CTC.class, getField(m, ocfn), gro(Comparator.A)),
   new Mode(Mode.FAST_PWM.class, null, FixedTop.FF),
   new Mode(Mode.FAST_PWM.class, null, FixedTop._1FF),
   new Mode(Mode.FAST_PWM.class, null, FixedTop._3FF),
   new Mode(Mode.FC_PWM.class, getField(m, icfn), gri(Comparator.I)),
   new Mode(Mode.FC_PWM.class, getField(m, ocfn), gro(Comparator.A)),
   new Mode(Mode.PWM.class, getField(m, icfn), gri(Comparator.I)),
   new Mode(Mode.PWM.class, getField(m, ocfn), gro(Comparator.A)),
   new Mode(Mode.CTC.class, getField(m, icfn), gri(Comparator.I)),null,
   new Mode(Mode.FAST_PWM.class, getField(m, icfn),gri(Comparator.I)),
   new Mode(Mode.FAST_PWM.class, getField(m, ocfn), gro(Comparator.A))
};
```

Cada modo se define mediante un nombre y un valor máximo de la cuenta (que puede ser fijo o establecido por la unidad utilizada, ya sea la de comparación o la de entrada).

Por último, y al igual que para el de 8 bits, se suministran una serie de métodos útiles para distintos cometidos, entre otros, la consulta de la cuenta, obtención del valor máximo, el nombre del registro que la está contabilizando, etc.

7.4.4. Timer0

El Timer0 es uno de los dos contadores de 8 bits que incluye el ATMega1281. Su implementación consiste en la creación de la clase ***«NewTimer0»***, que hereda directamente de ***«NewTimer8Bit»***. Como ya se ha dicho, esta última define la estructura base sobre la que se construyen los dispositivos temporizadores de 8 bits del microcontrolador. Sólo es necesario añadir convenientemente los detalles particulares de cada uno de estos elementos para completar su codificación. Como se expuso en la subsección 1.4.2, el nuevo temporizador Timer0 difiere del ya incluido en este fichero en que no soporta el modo asíncrono como su predecesor en el ATMega128. Por lo tanto, toda esta parte del código, en este caso, no es necesaria. La clase desarrollada ***«NewTimer0»*** es muy simple. Su constructor llama al constructor de la clase padre con una serie de parámetros:

```
super(0, ATMegaClassic.this, periods0, "TIMER0 OVF",
      "TIMER0 COMPA", "TIMER0 COMPB")
```

Los dos primeros parámetros son el identificador numérico del temporizador (“0”) y la instancia que representa al propio microcontrolador, respectivamente. El tercero representa los divisores de la frecuencia principal, de la forma:

```
int[] periods0 = {0, 1, 8, 32, 64, 128, 256, 1024}
```

Los tres restantes representan el nombre de la interrupción que se produce al desbordar la cuenta del contador 0 y las interrupciones asociadas a cada una de las dos unidades de comparación con las que cuenta el Timer0.

7.4.5. Timer2

Éste es el otro módulo temporizador/contador de 8 bits del ATMega1281. Es exactamente igual al anterior con la excepción de que incluye el modo asíncrono. La clase que implementa este dispositivo se llama ***«NewTimer2»*** y hereda de ***«NewTimer8Bit»***. Su constructor llama al de la clase padre ***«NewTimer8Bit»*** de la siguiente manera:

```
super(2, ATMegaClassic.this, periods2, "TIMER2 OVF",
      "TIMER2 COMPA", "TIMER2 COMPB")
```

con

```
int[] periods2 = {0, 1, 8, 64, 256, 1024}
```

Los parámetros tienen el mismo significado que en el caso anterior con la particularidad de que se refiere al temporizador número 2. La novedad se encuentra en la inclusión del modo asíncrono. Para gestionar dicho modo, el temporizador hace uso del registro *ASSR*, por lo que es necesario incluirlo dentro de la clase:

```
class ASSRRegister extends RWRegister
```

La clase ***«RWRegister»*** simplemente implementa el concepto registro de lectura/escritura, de tal manera que se puede leer o escribir un valor en él. La clase ***«ASSRRegister»*** hereda de ella e incluye, explícitamente, el nombre de cada uno de sus bits:

```
static final int EXCLK = 6;
static final int AS2 = 5;
static final int TCN2UB = 4;
static final int OCR2AUB = 3;
static final int OCR2BUB = 2;
static final int TCR2AUB = 1;
static final int TCR2BUB = 0;
```

Por último, es preciso añadir un método para modificar su contenido y actuar, si fuese necesario, en consecuencia. El bit *AS2* es el que especifica el reloj utilizado por el temporizador. Por lo tanto, cualquier escritura sobre *ASSR* comprobará el estado de dicho bit para usar la fuente de sincronismo apropiada.

7.4.6. Timer1

Es uno de los dos temporizadores/contadores de 16 bits del ATMega1281. La clase creada para representarlo es ***«Timer1»*** y hereda directamente de ***«Timer16Bit»***. En su constructor se realiza la llamada al constructor de la clase padre siguiente:

```
super(1, ATMegaClassic.this, periods1, "TIMER1 OVF", cf1Names)
```

Los dos primeros parámetros especificados en la llamada al constructor de ***«Timer16Bit»*** son el identificador del temporizador ("1") y una instancia del microcontrolador. El tercero identifica los posibles divisores de la frecuencia principal, de la forma:

```
int[] periods1 = new int[]{0, 1, 8, 64, 256, 1024}
```

El cuarto muestra el nombre de la interrupción de desbordamiento del temporizador 1. Por último, el quinto parámetro enumera los nombres de las interrupciones asociadas a las unidades de comparación y a la unidad de captura de entrada, que son los siguientes:

```
String[] cf1Names = new String[]{"TIMER1 CAPT", "TIMER1 COMPA",
                    "TIMER1 COMPB", "TIMER1 COMPC"}
```

7.4.7. Timer3

Es el dispositivo temporizador/contador de 16 bits restante. Se representa por la clase ***«Timer3»***, que, al igual que antes, hereda de ***«Timer16Bit»***. Es idéntico al Timer1. La llamada al constructor de ***«Timer16Bit»*** es:

```
super(3, ATMegaClassic.this, periods3, "TIMER3 OVF", cf3Names)
```

donde

```
int[] periods3 = {0, 1, 8, 64, 256, 1024}
String[] cf3Names = new String[]{"TIMER3 CAPT", "TIMER3 COMPA",
                    "TIMER3 COMPB", "TIMER3 COMPC"}
```

El significado de cada uno de los parámetros es equivalente al caso anterior.