Manejo de señales con temporizadores

Sistemas con Microprocesadores

Mg.Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

http://www.microprocesadores.unt.edu.ar/procesadores

Cronograma

Actividad	Inicio	Descripción	Completa
Presentación	24/08	Presentación y reglamento	✔
Tema 1	24/08	Estructura de las computadoras	✔
Tema 2	26/08	Proyecto con un microcontrolador	✔
Tema 3	31/08	Descripción funcional del ISA RV32IM	✔
Tema 4	07/09	Programación en lenguaje ensamblador	✔
Tema 5	18/09	Diseño del camino de datos de un procesador	✔
Tema 6	30/09	Diseño de la unidad de control del procesador	✔
Tema 7	05/10	Tecnologias y sistema de memoria	✔
Tema 8	09/10	Dispositivos de entrada/salida	✔
Tema 9	19/10	Entradas/salidas digitales (GPIO)	✔
Tema 10	28/10	Generalidades de las interrupciones	✔
Tema 11	04/11	Interrupciones y excepciones en RISC-V	✔
Tema 12	13/10	Manejo de señales con temporizadores	←
Tema 13	18/11	Transferencias de datos	
Proyectos	30/11	Seminarios de proyectos	

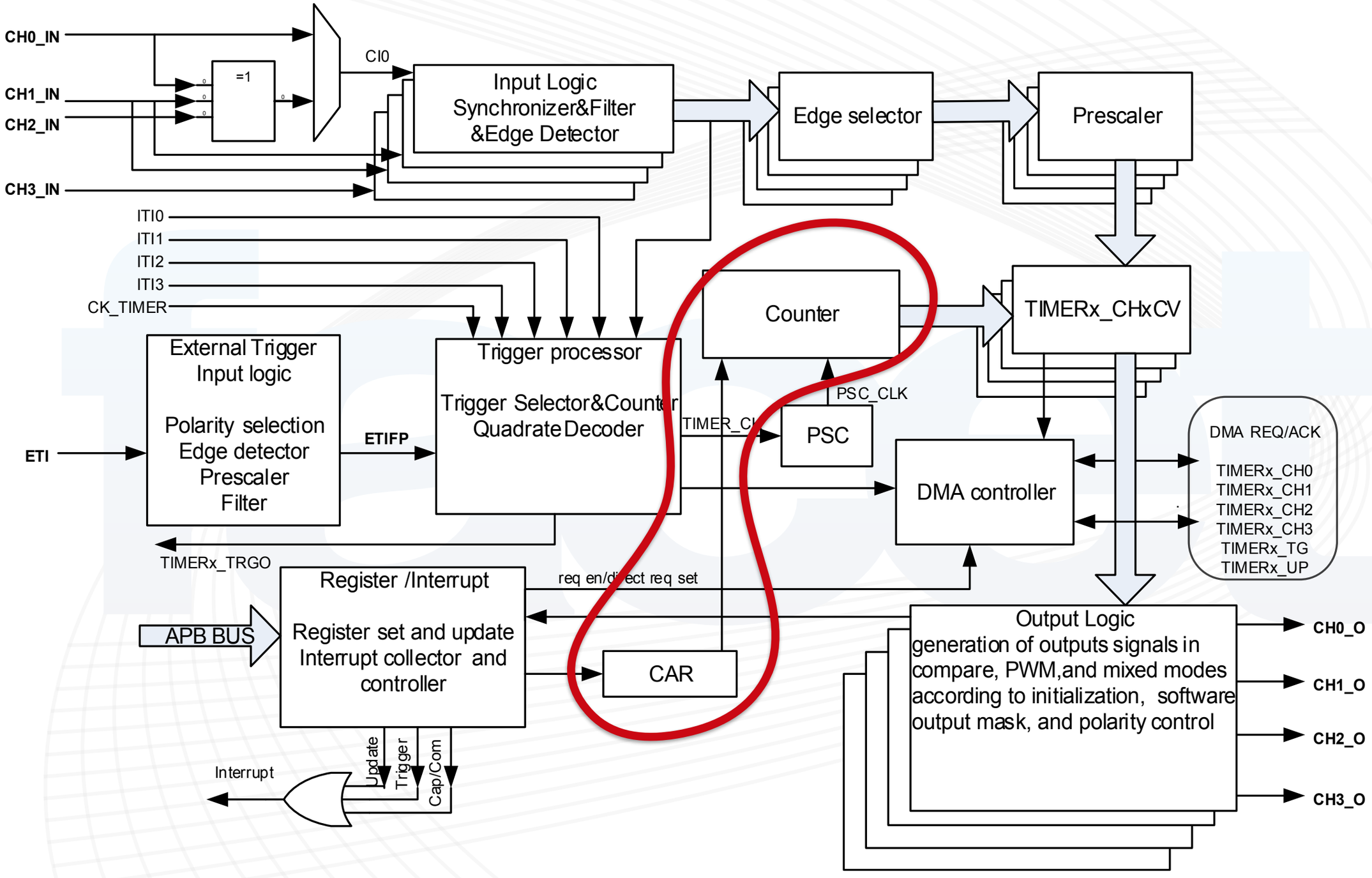
Generación de señales por software

- Utilizando el temporizador del sistema y los puertos GPIO se puede generar una señal periódica.
- En la rutina de servicio de la interrupción se cambia el estado de la salida y se programa una demora para el siguiente evento.
- La latencia en la respuesta a la interrupción introduce una fluctuación (jitter) en la forma de onda de la señal generada.
- El mismo problema se presenta al medir el tiempo entre dos flancos en una señal generada por un sensor.
- Para resolver este problema se utilizan los temporizadores.

Temporizadores en el GD32VF103

- Hay un total de 7 temporizadores, todos de 16 bits.
- El temporizador 0 es el más avanzado y tiene funciones específicas para generación de señales trifásicas
- Los temporizadores 1 al 4 son un poco más simples y están pensados para manejo de señales en general
- Los temporizadores 5 y 6 son los más básicos y son similares al temporizar del sistema
- Se estudiaran los temporizadores 1 al 4

Diagrama de bloques



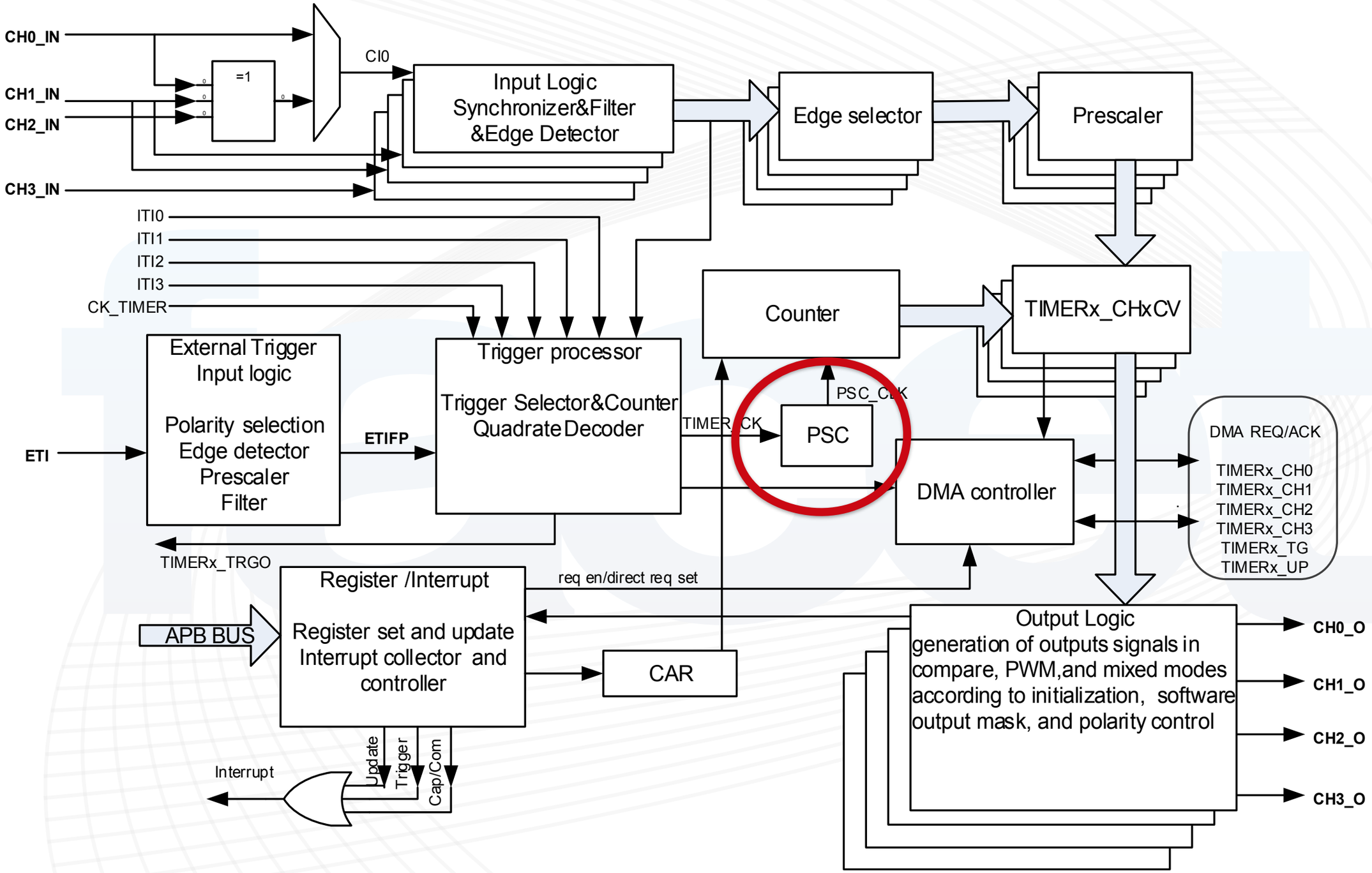
Núcleo del temporizador

- El núcleo es un contador **`TIMERx_CNT`** de 16 bits que puede operar en forma ascendente o descendente
- Un registro de recarga **`TIMERx_CAR`** permite definir:
  - El valor máximo del contador, cuando opera en forma ascendente
  - El valor inicial del contador, cuando opera en forma descendente
- Un registro de control **`TIMERx_CTL0`** permite configurar el funcionamiento del contador
  - Iniciar o detener la cuenta
  - Definir si la cuenta es ascendente o descendente
  - Definir si se cuenta una vez o se repite indefinidamente
  - Definir que sucede cuando el contador alcanza el valor límite

Registos de configuración

Nombre	Descripción
`TIMERx_CNT`	Valor actual del contador
`TIMERx_CTL0`	Registro de control del contador
`TIMERx_CAR`	Valor máximo o inicial del contador

Diagrama de bloques



Preescaler: un divisor programable

- La señal de reloj pasa un divisor programable
- Está formado por un contador, un registro de limite y un comparador, todos de 16 bits
- Cuanto el contador alcanza el valor limite reinicia la cuenta y genera un pulso de salida
  - El registro permite fijar el factor de división entre 1 y 65536
  - El factor de división es el valor del registro + 1
- El valor de límite se escribe en el registro **`TIMERx_PSC`**
  - Este valor se puede cambiar con el contador en funcionamiento
  - Se actualiza con el siguiente update del contador principal

Registos de configuración

Nombre	Descripción
TIMERx_CNT	Valor actual del contador
TIMERx_CTL0	Registro de control del contador
TIMERx_CAR	Valor máximo o inicial del contador
TIMERx_PSC	Factor de división para la señal de reloj

Evento de actualización

- Cuando el contador principal alcanza el valor limite y se recarga se puede generar un evento de update
- Este evento se puede utilizar para actualizar el factor de división y el valor de recarga
  - Los registros **TIMERx_PSC** y **TIMERx_CAR** tienen asociado un segundo registro (shadow)
  - La escritura se realiza en el registro oculto
  - Cuando ocurre un evento de actualización el valor del registro oculto se copia al registro principal
- El evento tambien se puede generar por software escribiendo el bit **UPG** del registro **TIMERx_SWEVG**
- Se puede consultar y borrar con el registro **TIMERx_INTF**

Registos de configuración

Nombre	Descripción
TIMERx_CNT	Valor actual del contador
TIMERx_CTL0	Registro de control del contador
TIMERx_CAR	Valor máximo o inicial del contador
TIMERx_PSC	Factor de división para la señal de reloj
TIMERx_INTF	Registro de estado con las banderas de los eventos
TIMERx_SWEVG	Registro para generar los eventos por software

Ejemplo de aplicación (1-encuesta)

- Se utiliza el temporizador TIMER1 con la fuente de reloj interna
- Se fija el factor de división para obtener una señal de reloj de 1 Khz
- Se fija el valor de recarga en 500
  - El contador principal alcanza el límite cada medio segundo
- Se habilita la generación del evento de update
- Se espera en un lazo el evento de update
  - Se borra el evento y se cambia el estado del led

Ejemplo de aplicación (1-encuesta)

```
int main(void) {
    SystemCoreClockUpdate();

    rcu_periph_clock_enable(RCU_GPIOA);
    gpio_init(GPIOA, GPIO_MODE_OUT_PP, GPIO_OSPEED_2MHZ,
              BIT(LED_ROJO) | BIT(LED_VERDE) | BIT(LED_AZUL));

    rcu_periph_clock_enable(RCU_TIMER1);
    timer_internal_clock_config(TIMER1);
    timer_prescaler_config(TIMER1, SystemCoreClock/1000, TIMER_PSC_RELOAD_NOW);
    timer_autoreload_value_config(TIMER1, 500);

    timer_update_event_enable(TIMER1);
    while (1) {
        if (timer_flag_get(TIMER1, TIMER_INT_UP)) {
            timer_interrupt_flag_clear(TIMER1, TIMER_INT_UP);
            if (gpio_output_bit_get(GPIOA, BIT(LED_AZUL))) {
                gpio_bit_reset(GPIOA, BIT(LED_AZUL));
            } else {
                gpio_bit_set(GPIOA, BIT(LED_AZUL));
            }
        }
    }
}
```

Resolución y rango de los tiempos

- La resolución es el menor valor con el que se puede fijar un tiempo
  - Esta determinado por el periodo de la señal de reloj que alimenta al contador principal del temporizador
- El rango es la relación entre el menor valor tiempo y el menor tiempo que se pueden especificar
  - Esta determinado por la cantidad de bits en el contador principal del temporizador

Determinación del periodo

- Los registros del prescaler y de recarga y del contador son de 16 bits
- Con el procesador funcionando a la frecuencia máxima de 108 MHz
  - Con un prescaler de 0 se obtiene un factor de división de 1
    - La resolución es máxima, aproximadamente 9,25 ns
    - El rango será de 9,25 ns hasta 606,81 us
  - Con un prescaler de 65535 se obtiene un factor de división de 65536
    - La frecuencia del contador será de 1,647 Hz
    - La resolución será de 606,81 us y el rango hasta 39,76 s

Uso por interrupciones

- Los eventos de un temporizador pueden generar una interrupción
  - Un temporizador puede generar otros tipos de eventos ademas del update
- Todas las interrupciones de un temporizador son tratadas por la misma rutina de servicio
  - La rutina puede utilizar el registro **`TIMERx_INTF`** para determinar la causa y tratarla adecuadamente
  - La rutina debe borrar los pedidos del registro **`TIMERx_INTF`** antes de terminar
  - En este caso el registro se borra escribiendo un cero en el bit correspondiente

Ejemplo de aplicación (2-recarga)

- La configuración del temporizador es muy parecida a la del ejemplo anterior
  - Sin embargo ahora el procesador opera a 108 MHz
  - Se deben elegir otros valores de prescaler y de recarga
- Pero se se habilita el ECLIC y la interrupciones globales
- Ademas se habilita al temporizador a pedir interrupción a partir del evento de update
- Finalmente se habilita en el ECLIC la interrupción correspondiente y se le fija la prioridad
- En la rutina de servicio se borra la bandera
  - No es necesario identificar la causa porque solo se habilitaron las interrupciones para el evento de update

Ejemplo de aplicación (2-recarga)

```
void TIMER1_IRQHandler(void) {
    timer_interrupt_flag_clear(TIMER1, TIMER_INT_UP);
    ...
}

int main(void) {
    SystemInit();
    ...

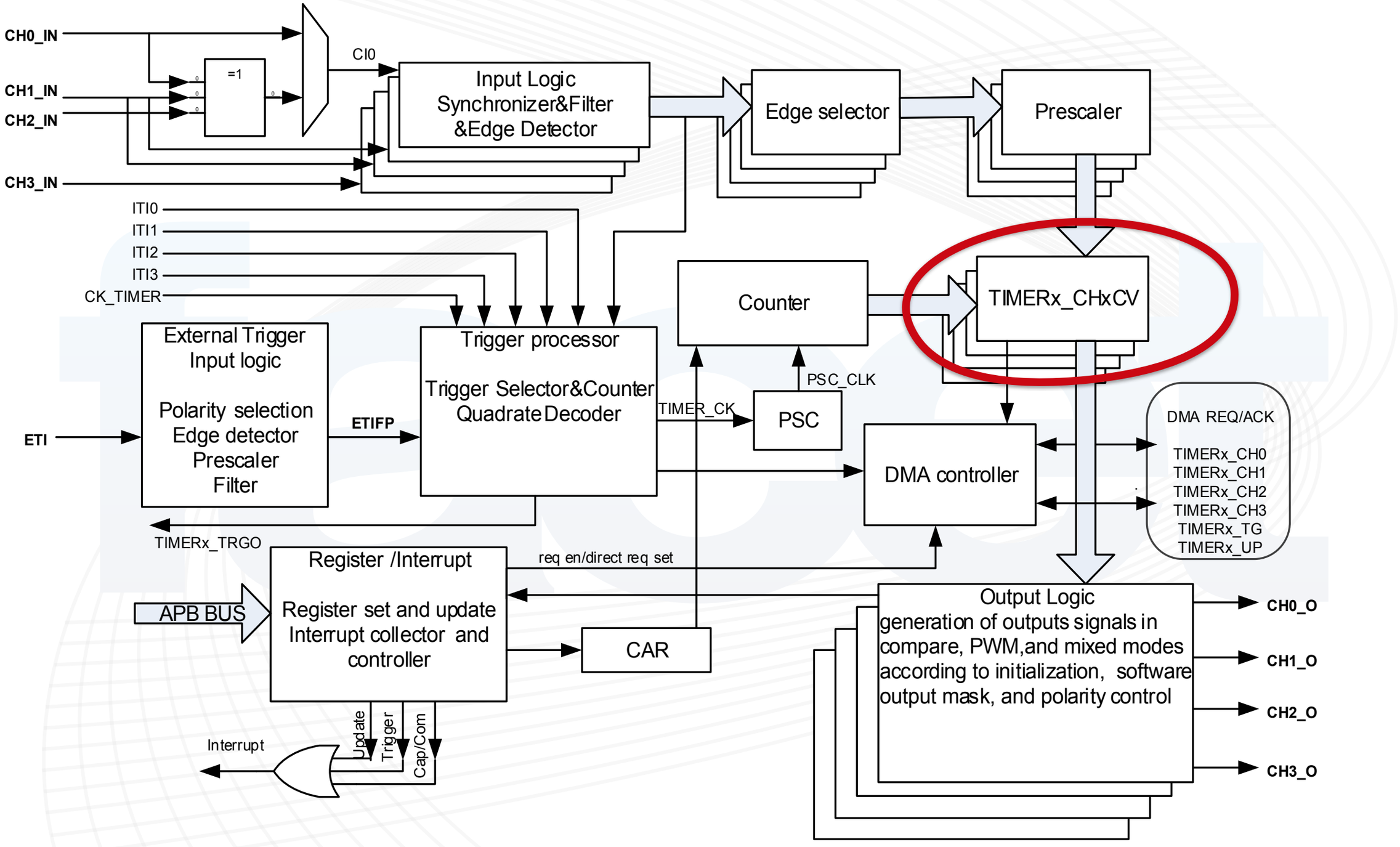
    eclic_mode_enable();
    eclic_global_interrupt_enable();

    rcu_periph_clock_enable(RCU_TIMER1);
    timer_internal_clock_config(TIMER1);
    timer_prescaler_config(TIMER1, SystemCoreClock/10000, TIMER_PSC_RELOAD_NOW);
    timer_autoreload_value_config(TIMER1, 5000);
    timer_update_event_enable(TIMER1);

    timer_interrupt_flag_clear(TIMER1, TIMER_INT_UP);
    timer_interrupt_enable(TIMER1,TIMER_INT_UP);
    eclic_irq_enable(TIMER1_IRQn, 1, 1);

    timer_enable(TIMER1);
    while (1) {
        asm("wfi");
    }
}
```

Diagrama de bloques



Canales del temporizador

- El temporizador incluye ademas cuatro canales
  - Cada canal tiene un registro **`TIMERx_CHyCV`**
- Un canal se puede configurar en modo output compare
  - Cuando el valor del contador **`TIMERx_CNT`** supera al valor del registro del canal **`TIMERx_CHyCV`** se genera un evento
  - Los cuatro canales se configuran usando los registros de control **`TIMERx_CHCTL0`** y **`TIMERx_CHCTL1`** (cada byte configura un canal)
- Este evento se puede atender por encuestas o por interrupciones como el evento de update

Registos de configuración

Nombre	Descripción
TIMERx_CNT	Valor actual del contador
TIMERx_CTL0	Registro de control del contador
TIMERx_CAR	Valor máximo o inicial del contador
TIMERx_PSC	Factor de división para la señal de reloj
TIMERx_CHCTL0	Registro de control de los canales 0 y 1
TIMERx_CHCTL1	Registro de control de los canales 2 y 3
TIMERx_CHyCV	Valor del registro del canal
TIMERx_INTF	Registro de estado con las banderas de los eventos
TIMERx_SWEVG	Registro para generar los eventos por software

Ejemplo de aplicación (3-canales)

- La configuración general del temporizador se mantiene
  - Como el ejemplo funciona por interrupciones tambien se mantiene la configuración del ECLIC
- Se configuran los tres primeros canales para generar eventos en 250, 500 y 750 ms
- En la rutina de servicio se identifica la causa de la interrupción
  - Se borra la bandera correspondiente
- Se prende un color del led en cada evento de los canales
- Se apagan todos los colores en el evento de update

Ejemplo de aplicación (3-canales)

```
int main(void) {
    ...
    rcu_periph_clock_enable(RCU_TIMER1);
    timer_internal_clock_config(TIMER1);
    timer_prescaler_config(TIMER1, SystemCoreClock/10000, TIMER_PSC_RELOAD_NOW);
    timer_autoreload_value_config(TIMER1, 10000);
    timer_update_event_enable(TIMER1);

    timer_channel_output_mode_config(TIMER1, TIMER_CH_0, TIMER_OC_MODE_TIMING);
    timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_0, 250);

    timer_channel_output_mode_config(TIMER1, TIMER_CH_1, TIMER_OC_MODE_TIMING);
    timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_1, 500);

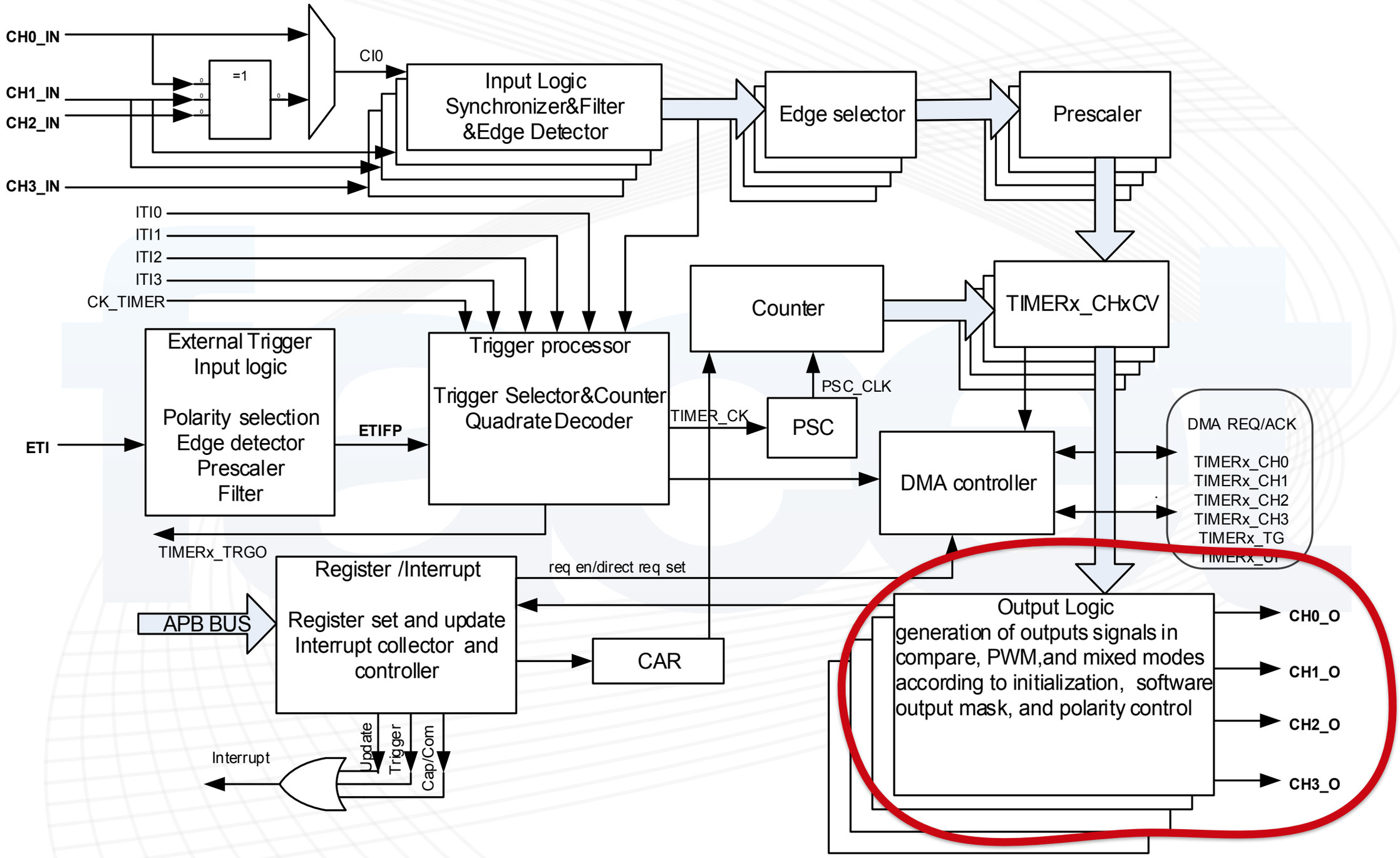
    timer_channel_output_mode_config(TIMER1, TIMER_CH_2, TIMER_OC_MODE_TIMING);
    timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_2, 750);

    timer_interrupt_enable(TIMER1,TIMER_INT_CH0);
    timer_interrupt_enable(TIMER1,TIMER_INT_CH1);
    timer_interrupt_enable(TIMER1,TIMER_INT_CH2);
    timer_interrupt_enable(TIMER1,TIMER_INT_UP);
    eclic_irq_enable(TIMER1_IRQn, 1, 1);
    ...
}
```

Ejemplo de aplicación (3-canales)

```
void TIMER1_IRQHandler(void) {
    if (timer_flag_get(TIMER1, TIMER_INT_CH0)) {
        timer_interrupt_flag_clear(TIMER1, TIMER_INT_CH0);
        gpio_bit_set(GPIOA, BIT(LED_AZUL));
    }
    if (timer_flag_get(TIMER1, TIMER_INT_CH1)) {
        timer_interrupt_flag_clear(TIMER1, TIMER_INT_CH1);
        gpio_bit_set(GPIOA, BIT(LED_ROJO));
    }
    if (timer_flag_get(TIMER1, TIMER_INT_CH2)) {
        timer_interrupt_flag_clear(TIMER1, TIMER_INT_CH2);
        gpio_bit_set(GPIOA, BIT(LED_VERDE));
    }
    if (timer_flag_get(TIMER1, TIMER_INT_UP)) {
        timer_interrupt_flag_clear(TIMER1, TIMER_INT_UP);
        gpio_bit_reset(GPIOA, BIT(LED_AZUL));
        gpio_bit_reset(GPIOA, BIT(LED_ROJO));
        gpio_bit_reset(GPIOA, BIT(LED_VERDE));
    }
}
```

Diagrama de bloques



Generación de señales

- Para generar una señal con la máxima precisión el evento del comparador puede actuar sobre una salida por hardware
  - Cada canal puede actuar sobre una sola salida
  - La salida se puede mover entre tres posible terminales físicos
  - El terminal se debe configurar en función alternativa
  - Se debe habilitar el control del terminal por el canal y elegir si la señal es activa en nivel alto o en bajo usando el registro **`TIMERx_CHCTL2`**
- Si la señal a generar es periódica, no se necesita nada de software para generarla
- Si la señal es no periódica se puede utilizar la interrupción para programar el próximo evento a partir del evento anterior
  - Esto permite que la latencia en la atención de la interrupción no interfiera en la generación de la señal

Registos de configuración

Nombre	Descripción
TIMERx_CNT	Valor actual del contador
TIMERx_CTL0	Registro de control del contador
TIMERx_CAR	Valor máximo o inicial del contador
TIMERx_PSC	Factor de división para la señal de reloj
TIMERx_CHCTL0	Registro de control de los canales 0 y 1
TIMERx_CHCTL1	Registro de control de los canales 2 y 3
TIMERx_CHCTL2	Registro para activar el terminal y selecionar la polaridad
TIMERx_CHyCV	Valor del registro del canal
TIMERx_INTF	Registro de estado con las banderas de los eventos
TIMERx_SWEVG	Registro para generar los eventos por software

Ejemplo de aplicación (4-hardware)

- La configuración general del temporizador se mantiene
  - Se elimina la configuración del ECLIC, no habrá interrupciones
- Los terminales de los led se configuran para la función alterna (temporizador) y no para salidas digitales (gpio)
- Se configuran los canales para prender los tres colores en el evento de update del contador principal
- Se programan los registros para apagar el terminal correspondiente a los 250ms, 500ms, y 750 ms
- El ciclado de los colores se hace por hardware
  - Ee único software que se ejecuta es para la configuración inicial

Ejemplo de aplicación (4-compare)

```
int main(void) {
    ...
    rcu_periph_clock_enable(RCU_AF);
    rcu_periph_clock_enable(RCU_GPIOA);
    gpio_init(GPIOA, GPIO_MODE_AF_PP, GPIO_OSPEED_2MHZ,
              BIT(LED_ROJO) | BIT(LED_VERDE) | BIT(LED_AZUL));

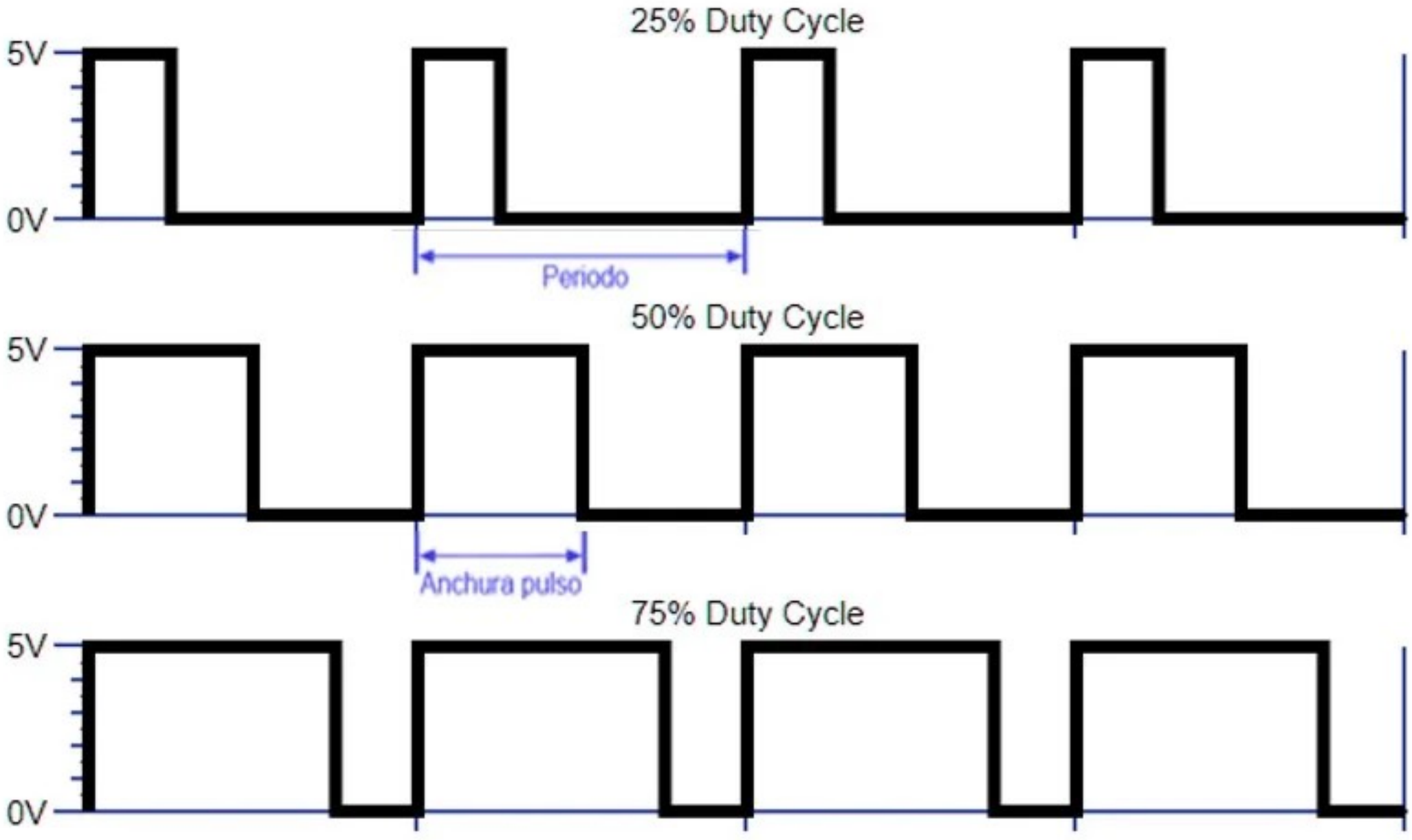
    rcu_periph_clock_enable(RCU_TIMER1);
    timer_internal_clock_config(TIMER1);
    timer_prescaler_config(TIMER1, SystemCoreClock / 10000, TIMER_PSC_RELOAD_NOW);
    timer_autoreload_value_config(TIMER1, 10000);

    timer_channel_output_state_config(TIMER1, TIMER_CH_1, TIMER_CCX_ENABLE);
    timer_channel_output_polarity_config(TIMER1, TIMER_CH_1, TIMER_OC_POLARITY_HIGH);
    timer_channel_output_mode_config(TIMER1, TIMER_CH_1, TIMER_OC_MODE_PWM0);
    timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_1, 7500);
    ...
}
```

Modulación del ancho del pulso (PWM)

- Una forma simple de controlar la potencia en un dispositivo es por PWM
- Se genera una señal cuadrada de periodo constante
- La tensión continua equivalente será igual al factor de trabajo
  - La relación entre el tiempo en que la señal tiene una tensión alta respecto al tiempo de la señal con la tensión baja
- Cambiando el factor de trabajo se puede cambiar tensión continua equivalente sobre el dispositivo
- Los transistores trabajan solo en corte o saturación, y por lo tanto casi no disipan potencia

Modulación del ancho del pulso (PWM)



Ejemplo de aplicación (5-pwm)

- Se configura el prescaler y el valor de recarga para obtener una frecuencia de 10 KHz
  - Se suele elegir un valor de recarga múltiplo de 10 para facilitar el calculo de la potencia
- Se mantiene la configuración de los canales del ejemplo anterior
- El programa utiliza el evento de update para generar una rampa en el valor de potencia
  - Esta rampa ademas se va aplicando a diferentes canales para cambiar los colores

Ejemplo de aplicación (5-pwm)

```
int main(void) {
    ...
     rcu_periph_clock_enable(RCU_TIMER1);
     timer_internal_clock_config(TIMER1);
     timer_prescaler_config(TIMER1, SystemCoreClock/10000000, TIMER_PSC_RELOAD_NOW);
     timer_autoreload_value_config(TIMER1, PERIODO);
     timer_update_event_enable(TIMER1);
     ...
     timer_enable(TIMER1);
     while (1) {
         if (timer_flag_get(TIMER1, TIMER_INT_UP)) {
             timer_interrupt_flag_clear(TIMER1, TIMER_INT_UP);
             ...
             timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_1, r);
             timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_2, b);
             timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_3, g);
         }
     }
}
```

PWM centrado

- Las señales de los canales del ejemplo anterior están alineadas en uno de los flancos
  - En el ejemplo el flanco ascendente de todas las señales se produce al mismo tiempo en el evento de update del contador
- Cuando se manejan elementos de potencia esta conmutación simultanea puede afectar a la fuente de alimentación y causar transitorios problemáticos
- Una solución es alinear los pulsos por el centro y no por un flanco
- Para esto se puede configurar que el contador para que primero cuente en forma ascendente hasta el valor máximo y después en forma descendente hasta cero
- De esta forma el canal se activa cuando la cuenta ascendente es mayor que el valor del canal y se apaga cuando la cuenta descendente es menor que el mismo valor

Ejemplo de aplicación (6-centrado)

- El temporizador se configura prácticamente igual que en el ejemplo 4-pwm
  - Se configura a baja velocidad para que se note la diferencia de funcionamiento respecto al PWM alineado
- Se configura al contador principal para generar un PWM centrado
  - Esta configuración afecta a todos los canales del temporizador
- A pesar de utilizar los mismos valores de preescaler y de recarga la frecuencia de la señal generada es la mitad que en el ejemplo anterior
  - El periodo de la señal es ahora el tiempo que demora una cuenta ascendente y una descendente

Ejemplo de aplicación (6-centrado)

```
int main(void) {
    ...
    rcu_periph_clock_enable(RCU_AF);
    rcu_periph_clock_enable(RCU_GPIOA);
    gpio_init(GPIOA, GPIO_MODE_AF_PP, GPIO_OSPEED_2MHZ,
              BIT(LED_ROJO) | BIT(LED_VERDE) | BIT(LED_AZUL));

    rcu_periph_clock_enable(RCU_TIMER1);
    timer_internal_clock_config(TIMER1);
    timer_prescaler_config(TIMER1, SystemCoreClock / 10000, TIMER_PSC_RELOAD_NOW);
    timer_autoreload_value_config(TIMER1, 10000);
    timer_counter_alignment(TIMER1, TIMER_COUNTER_CENTER_DOWN);


    timer_channel_output_state_config(TIMER1, TIMER_CH_1, TIMER_CCX_ENABLE);
    timer_channel_output_polarity_config(TIMER1, TIMER_CH_1, TIMER_OC_POLARITY_HIGH);
    timer_channel_output_mode_config(TIMER1, TIMER_CH_1, TIMER_OC_MODE_PWM0);
    timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_1, 7500);
    ...
}
```

Señales no periódicas

- Tambien se pueden generar señales no periódicas
- Se fija el valor de recarga del contador principal en **`0xFFFF`**
- Se habilitan las interrupciones por evento de coincidencia en el canal
  - En cada interrupción se suma al valor de comparación anterior el ancho del pulso
  - Es similar a lo que se hizo con el temporizador del sistema pero la latencia de la interrupción no afecta a la señal
- Esta misma solución se puede utilizar cuando las señales generadas en cada canal tienen un periodo diferente
  - La función para calcular el siguiente evento puede ser tan compleja como sea necesario

Ejemplo de aplicación (7-compare)

- En este caso se generan tres señales con diferentes periodos
  - Todas tienen un 50% de factor de trabajo
  - El canal rojo tiene un periodo de 1500 ms
  - El canal azul tiene un periodo de 1000 ms
  - El canal verde tiene un periodo de 500 ms
- El canal se programa para producir un cambio de la señal en cada coincidencia
  - Se programa el valor del primer evento con la mitad del periodo en la configuración inicial del programa
  - En cada interrupción se programa el siguiente evento sumando la mitad del periodo al valor anterior

Ejemplo de aplicación (7-compare)

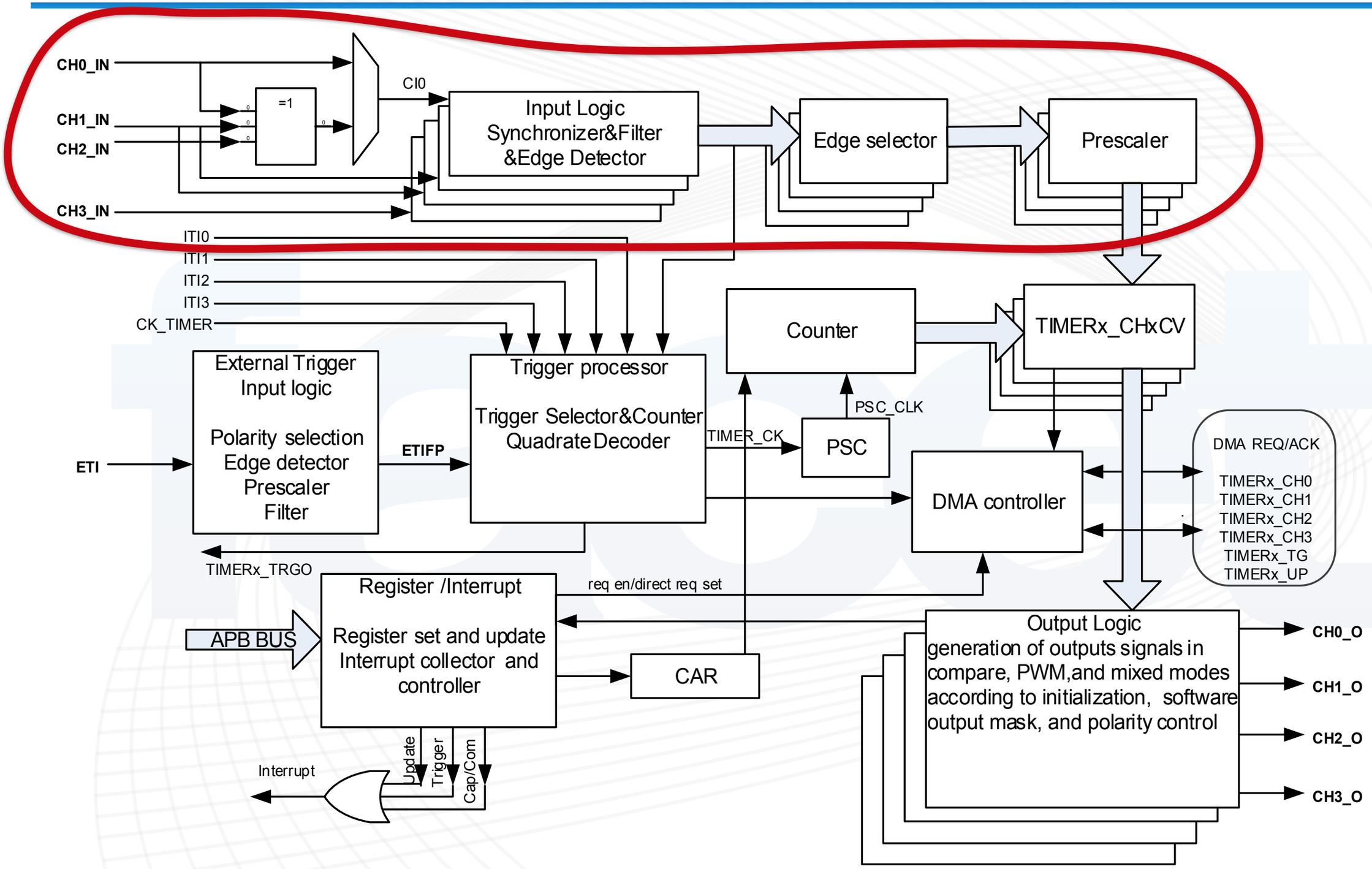
```
void TIMER1_IRQHandler(void) {
    if (timer_flag_get(TIMER1, TIMER_INT_CH1)) {
        timer_interrupt_flag_clear(TIMER1, TIMER_INT_CH1);
        timer_channel_output_pulse_value_config(
            TIMER1, TIMER_CH_1,
            timer_channel_capture_value_register_read(TIMER1, TIMER_CH_1) + 7500);
    }
    ...
}

int main(void) {
    ...
    eclic_mode_enable();
    eclic_global_interrupt_enable();
    ...
    timer_prescaler_config(TIMER1, SystemCoreClock / 5000, TIMER_PSC_RELOAD_NOW);
    timer_autoreload_value_config(TIMER1, 0xFFFF);

    // Control del canal rojo del led RGB
    timer_channel_output_state_config(TIMER1, TIMER_CH_1, TIMER_CCX_ENABLE);
    timer_channel_output_polarity_config(TIMER1, TIMER_CH_1, TIMER_OC_POLARITY_HIGH);
    timer_channel_output_mode_config(TIMER1, TIMER_CH_1, TIMER_OC_MODE_TOGGLE);
    timer_channel_output_pulse_value_config(TIMER1, TIMER_CH_1, 7500);

    timer_interrupt_enable(TIMER1, TIMER_INT_CH1);
    ...
    eclic_irq_enable(TIMER1_IRQn, 1, 1);
    ...
}
```

Diagrama de bloques



Medición de señales

- Los canales tambien se puede utilizar para medir tiempos entre flancos
  - Se puede medir el periodo de una señal, esperando dos flancos del mismo tipo, o el ancho de un pulso con dos flancos diferentes
- Se programa el canal en modo captura de entradas
  - El terminal para ingresar la señal es el mismo que genera la señal en modo de salida
  - Se puede programar en que flanco se produce el evento de captura
- Cuando se produce el flanco esperado el registro del canal copia el valor del contador principal
  - Tambien puede producir una interrupción para procesar el valor capturado
  - Como el valor del contador se almacena por hardware en el momento del evento la latencia en la interrupción no afecta a la medición

Ejemplo de aplicación (8-captura)

- En este caso se plantea un “juego" donde dos jugadores buscan presionar una tecla por el menor tiempo posible
  - Se remapean las funciones para conectar los canales 2 y 3 del temporizador 1 con los terminales **PB10** y **PB11**
- Se configuran los canales para captura en flanco ascendente
  - Espera que se presione la tecla aceptar o cancelar
- En la primera interrupción se almacena el valor medido y se cambia la configuración para captura en flanco descendente
  - Espera que se libere la tecla presionada
- En la segunda interrupción se calcula el ancho del pulso a partir del valor actual y del valor de la interrupción anterior
  - Se almacena en una variable para mostrarlo en el programa principal

Ejemplo de aplicación (8-captura)

```
int main(void) {

    rcu_periph_clock_enable(RCU_AF);
    rcu_periph_clock_enable(RCU_TIMER1);
    gpio_pin_remap_config(GPIO_TIMER1_PARTIAL_REMAP1, ENABLE);

    timer_internal_clock_config(TIMER1);
    timer_prescaler_config(TIMER1, SystemCoreClock / 100000, TIMER_PSC_RELOAD_NOW);
    timer_autoreload_value_config(TIMER1, 0xFFFF);

    timer_input_capture_config(TIMER1, TIMER_CH_2,
                               &(timer_ic_parameter_struct){
                                   .icpolarity = TIMER_IC_POLARITY_RISING,
                                   .icselection = TIMER_IC_SELECTION_DIRECTTI,
                                   .icprescaler = TIMER_IC_PSC_DIV1,
                                   .icfilter = 15,
                               });
    ...

    timer_interrupt_enable(TIMER1, TIMER_INT_CH2);
    timer_enable(TIMER1);
    timer_interrupt_flag_clear(TIMER1, TIMER_INT_CH2);
    eclic_irq_enable(TIMER1_IRQn, 1, 1);

}
```

Ejemplo de aplicación (8-captura)

```
typedef struct key_status_s {
    bool pressed;           /**< Primer o segundo flanco */
    uint16_t start_value; /**< Contador en el primer flanco */
    bool measured;          /**< Duración es válida */
    uint16_t duration;    /**< Duración de la ultima pulsación */
} * key_status_t;


void UpdateStatus(key_status_t key, uint16_t current_value) {
    if (!key->pressed) {
        key->start_value = current_value;
        key->pressed = true;
    } else {
        key->duration = current_value - key->start_value;
        if (key->duration > 9999) {
            key->duration = 9999;
        }
        key->pressed = false;
        key->measured = true;
    }
}
```

Ejemplo de aplicación (8-captura)

```
void TIMER1_IRQHandler(void) {
    if (timer_flag_get(TIMER1, TIMER_INT_CH2)) {
        timer_interrupt_flag_clear(TIMER1, TIMER_INT_CH2);
        UpdateStatus(&keys_status[0], TIMER_CH2CV(TIMER1));
        TIMER_CHCTL2(TIMER1) ^= TIMER_CHCTL2_CH2P;
    }
    ...
}

int main(void) {
    ...
    while (1) {
        asm("wfi");
        if (keys_status[0].measured) {
            keys_status[0].measured = false;
            DisplayWriteValue(board->display, keys_status[0].duration);
        }
        ...
        DisplaySetDot(board->display, 2);
    }
}
```

Conclusiones

- Las tres funciones de los temporizadores son muy utilizadas en los sistemas embebidos reales
  - El PWM se utiliza para controlar la potencia de motores, leds y resistencias
  - La generación de señales se utiliza para controlar motores sin escobillas y para codificar información que se transmitirá por diferentes medios
  - La medición de señales se utiliza para medición de velocidades, de distancias y para decodificar la información recibida por diferentes medios
- La combinación de funciones por hardware y por software por interrupciones permiten mucha flexibilidad con la menor carga posible de procesamiento