Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Emulador programable de señal digital empleando PRU-ICSS

Autor: Rubén Comerón Galán

Tutor: Juan Ignacio García Tejedor

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Emulador programable de señal digital empleando PRU-ICSS

Autor: Rubén Comerón Galán

Tutor: Juan Ignacio García Tejedor

Tribunal:

Presidente: Antonio Da Silva Fariña

Vocal 1º: Pablo Parra Espada

Vocal 2º: Juan Ignacio García Tejedor

Fecha de depósito: 15 de septiembre de 2022

Agradecimientos

Quiero aprovechar este espacio para agradecer a los que han estado a mi lado durante esta etapa de mi vida.

A mi familia por haberme ayudado y brindarme la oportunidad de estudiar este grado.

A mis amigos, tanto los que empezaron conmigo como los que he hecho por el camino, por apoyarme en los que han sido los años más difíciles de mi vida.

A todos ellos, mil gracias. Sin vosotros nada de esto habría sido posible.

Resumen

El objetivo de este trabajo es la implementación de un sistema capaz de generar distintas señales digitales, como por ejemplo un pulso similar al generado por un detector de partículas, utilizando para ello una BeagleBone Black, mediante su subsistema Programmable in Real-time Unit and Industrial Communications SubSystem (PRU-ICSS) consistente de 2 módulos PRU. Para ello se creará un programa en el subsistema Advanced RISC Machine (ARM) que contiene un sistema Linux de forma que una PRU se comunique con la restante y esta última muestre los valores necesarios a través de los pines previamente configurados.

Palabras clave: PRU, pin, remoteproc, RPMSG, DRDY.

Abstract

The target of this proyect is the implementation of a system able to generate different digital signals, like a pulse similar to the one obtained from a particule detector, using a BeagleBone Black by means of its PRU-ICSS subsystem containing 2 PRU modules. For that purpose, I will create a program in the ARM subsystem running a Linux system. This program will make that one PRU communicates with the other one and the last mentioned shows the signal at the previously configured pins.

Keywords: PRU, pin, remoteproc, RPMsg, DRDY.

Resumen extendido

El propósito de este trabajo se basa en poder recrear los estímulos que recibiría un sistema a la salida de un detector de radiación imitando los datos que ofrecería un Analog to Digital Converter (ADC) ante señales características y un pulso de radiación. Los datos ofrecidos por el ADC son 12 bits de la señal característica y un bit que representa una señal Data Ready (DRDY) la cual tendrá un flanco ascendente una vez la muestra de la señal esté preparada y estabilizada. Con el resultado de este trabajo se pretende comprobar el tratamiento de la señal que ofrece un sistema ante pulsos de radiación y señales características como un diente de sierra en formato digital.

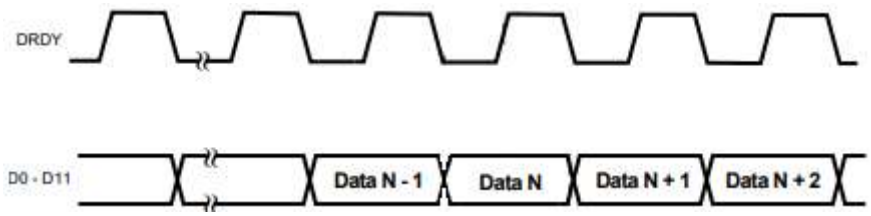


Figura 2: Señales que producirá el emulador.

Para este próposito he utilizado una BeagleBone Black debido a su subsistema PRU-ICSS, compuesto por dos núcleos programables de manera independiente a la Central Processing Unit (CPU) y que no hacen uso de sistema operativo. Sus características más destacables radican en la frecuencia de su reloj (200 MHz) y en su Ciclos Por Instrucción (CPI) (muchas instrucciones se ejecutan en un 1 ciclo). Esto permite obtener velocidades de muestreo aceptables conservando la regularidad en el ritmo al que se obtienen las muestras.

Dado que la BeagleBone Black permite a la PRU1 el acceso a más pines que a la PRU0, he escogido la PRU1 para mostrar la señal. Esta tarea ocupará la PRU1 por completo ya que la señal es de varios Mega Samples Per Second (MSPS) por lo que la PRU0 será la encargada de comunicarse con el usuario a la hora de la elección de la señal y de almacenar las muestras de dicha señal para la PRU1.

Con el fin de simplificar la selección de la señal he desarrollado un programa que ofrece una interfaz de usuario sencilla a la hora de indicar la señal que debe ser mostrada. Este programa se ejecutará en el sistema Linux de la BBB sobre el núcleo ARM.

A la hora de estructurar las etapas del trabajo he diferenciado 4: búsqueda de información respecto a la BeagleBone Black y al procesador AM335x, comprobación del funcionamiento de distintos ejemplos, planteamiento del protocolo de selección de señal y de comunicación entre PRUs y, finalmente, desarrollo del firmware y verificación de los resultados.

En la primera etapa me he centrado en la forma de escoger la configuración de los pines de la BeagleBone Black, en el proceso de compilación de firmware para las PRUs y de su ejecución y en qué se fundamenta el módulo Remote Processor Messaging (RPMsg) que permite la comunicación entre la

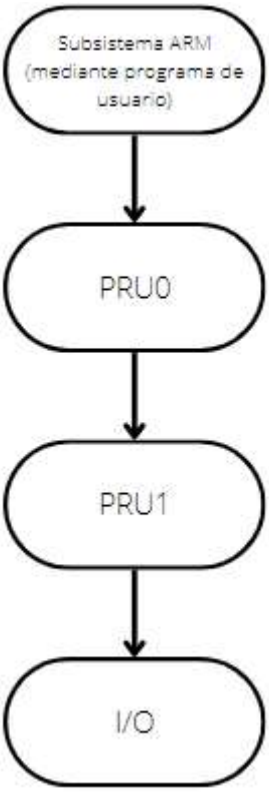


Figura 3: Diagrama sobre el planteamiento del trabajo.

PRU0 y el núcleo ARM. Esta información se encuentra tanto en foros de usuarios como en los documentos informativos por parte de Texas Instruments (TI).

En la segunda etapa he utilizado 4 ejemplos: 2 de ellos para el uso del compilador y la forma de ejecutar el programa, 1 para la incorporación de funciones en ensamblador dentro del programa a ejecutar y 1 para comprobar el funcionamiento del modulo RPMsg.

En la tercera etapa planteo el protocolo de selección de señal a través del módulo RPMsg y el método de comunicación entre ambas PRUs a través de su Shared Random Access Memory (SRAM).

En la cuarta etapa compruebo que la señal resultante se corresponde con lo necesario para cumplir el objetivo del trabajo tanto es términos de frecuencia, como de regularidad. También compruebo que la señal DRDY se ajusta a las necesidades del sistema.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En cualquier ámbito dentro del campo de la ingeniería es de vital importancia poder comprobar el correcto funcionamiento de un sistema previo a su implementación. Para conseguir este objetivo es necesario acercarnos lo máximo posible a las condiciones reales en las que se encontrará el sistema a probar. Estas condiciones se pueden englobar en distintas categorías siendo una de ellas los estímulos de entrada.

El objetivo de este trabajo consiste en poder recrear los estímulos que recibiría un sistema a la salida de un detector de radiación imitando los datos que ofrecería un ADC, como por ejemplo el ADC12C080 [7] con señales características y con la señal que se recibiría frente a un pulso de radiación.

Para llevar a cabo este objetivo, deberemos disponer de un sistema que cumpla con dos condiciones: ser lo suficientemente rápido como para poder reproducir las señales a una velocidad comparable con la velocidad de un ADC y que la generación de muestras ocurra a un ritmo estable. Este sistema también nos debe permitir cambiar entre distintos estímulos para poder comprobar como se trata posteriormente dicha señal. Dentro de estas posibilidades, podemos encontrar sistemas como un dispositivo Arduino, un dispositivo Raspberry y un dispositivo BeagleBone. Las opciones incluidas en estas 3 ramas suponen gran versatilidad a un bajo coste de adquisición.

Por un lado, las opciones de Arduino disponen de una gran cantidad de ejemplos acerca de su funcionamiento debido a su alto índice de popularidad y su sencillez. En cambio, no pueden otorgar un rendimiento comparable con el ADC a emular por lo que no sería adecuado. Por ejemplo, las opciones Arduino Uno y Arduino Mega2560 disponen de una frecuencia de reloj de 16 MHz con la que obtendríamos las muestras a una frecuencia muy inferior a la de un ADC.

Por otro lado, las opciones de la plataforma Raspberry Pi, como por ejemplo Raspberry Pi 3 modelo A+ o Raspberry Pi 4 modelo B, también disponen de muchos ejemplos acerca de su funcionamiento por lo que añadiría sencillez a su uso. En este caso, también tiene un rendimiento comparable al ADC pero para alcanzarlo habría que abandonar la idea de poder generar los distintos estímulos sin tener que pararlo por completo [8]. En cambio, no cumple con la regularidad a la hora de generar las muestras dado que el programa lo ejecutaría el sistema operativo.

Por último, disponemos de las opciones BeagleBone que sí cumplen las especificaciones de crear los estímulos a una velocidad aceptable a la vez que otorga una temporización estricta de la generación de la señal debido a la independencia de su módulo PRU-ICSS con respecto al sistema operativo.

Conociendo esto, opto por usar el dispositivo BeagleBone Black debido a que ofrece el subsistema PRU-ICSS que nos otorga versatilidad a la hora de programación y rapidez a la hora de mostrar las señales.

Capítulo 2

Estudio teórico

2.1 Introducción

En este capítulo se explicará desde un punto de vista teórico todo aquello necesario para el desarrollo del trabajo.

El capítulo se estructura en 6 apartados en los que veremos las características importantes del procesador, como utilizar el compilador que ofrece TI, el lenguaje ensamblador que utilizan las PRUs, los módulos remoteproc y RPMsg y como configurar los pines de la BeagleBone Black. Estos apartados constituyen todo lo necesario para llevar a cabo el resto del proyecto.

2.2 Arquitectura AM335x

En este apartado explicaremos la arquitectura del procesador AM335x. En primer lugar tenemos 2 grandes bloques: subsistema ARM y subsistema PRU-ICSS 2.1. Estos bloques son capaces de comunicarse entre sí a través del módulo RPMsg que describo más adelante, en la sección 2.6. El núcleo ARM es capaz de arrancar, parar y cargar firmware a través de remoteproc (3.3.1) ofreciendo una opción de utilizar el subsistema PRU-ICSS de forma sencilla.

2.2.1 Subsistema ARM

Respecto al subsistema ARM, podemos ver un diagrama funcional sobre él en la figura 2.1. Este subsistema hace uso un cortex-A8 con un reloj de hasta 1 GHz, con 176 kB de memoria ROM de arranque y 64 kB de memoria RAM. También dispone de controlador de interrupciones que permite hasta 128 peticiones de interrupcion. Es capaz de ejecutar un sistema Linux que, en el desarrollo del trabajo, consistirá de el sistema operativo Debian. Este núcleo no es adecuado ya que no es posible obtener una señal lo suficientemente regular y rápida como para conseguir emular el ADC [9] debido al uso del sistema operativo.

2.2.2 Subsistema PRU-ICSS

En cuánto al subsistema PRU-ICSS, está formado por 2 módulos PRUs con un reloj de 200 MHz cada uno. Estos módulos disponen de acceso a memorias en común y también trabajan con un sistema de

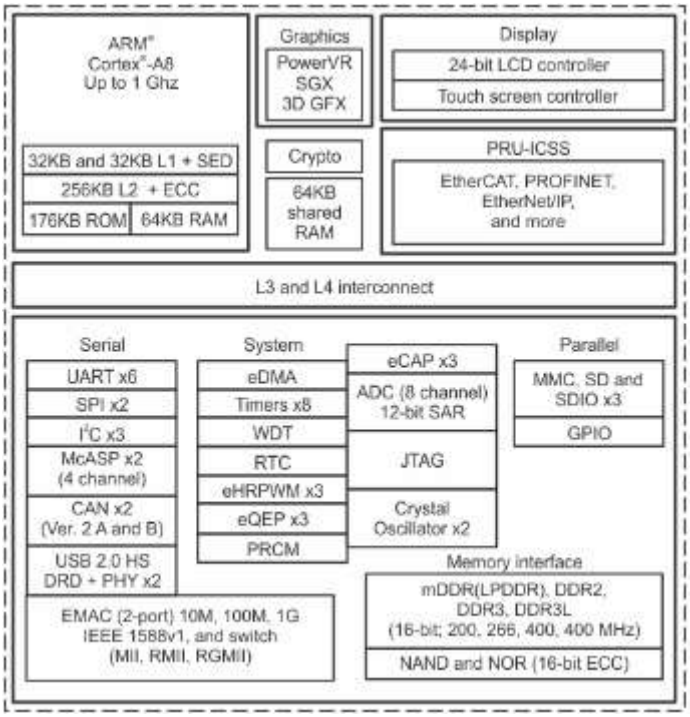


Figura 2.1: Diagrama funcional sobre AM335x [1].

interrupciones detallado más adelante. Desde el núcleo ARM se puede acceder a las distintas memorias de este subsistema a partir de la dirección de memoria 0x4A30_0000. Este subsistema tiene como propósito aplicaciones de tiempo real ya que la mayoría de instrucciones de las PRUs se ejecutan en 1 ciclo, necesitando así 5 ns por instrucción.

2.2.3 Memoria

En primer lugar, es necesario mencionar que las PRUs disponen de 32 registros. El registro R30 es utilizado como General Purpose Output (GPO) por lo que su valor se refleja en el exterior del procesador. El registro R31 es utilizado como General Purpose Input (GPI) por lo que puede capturar el valor del exterior del procesador. En la BeagleBone Black no todos los bits de estos registros son accesibles como muestro en la sección 2.7. El resto de registros son de propósito general y algunos de ellos ocupan un uso específico 2.4.2.

Los módulos PRUs disponen de varias regiones de memoria a las que pueden acceder. En primer lugar, se encuentra la memoria Random Access Memory (RAM) de datos de la propia PRU seguida de la memoria de datos de la otra PRU (la PRU0 percibe su memoria de datos en la dirección 0x0000 y la memoria correspondiente a la PRU1 en la dirección 0x2000 mientras que la PRU1 encuentra su memoria de datos en la dirección 0x0 y la memoria de la PRU0 en la dirección 0x2000). A continuación, ambas PRUs disponen de una memoria SRAM de 12 kB en común a partir de la dirección 0x10000. La siguiente memoria accesible almacena los registros correspondientes a las interrupciones que menciono en el apartado 2.2.4. Por último, también tienen acceso a las memorias con los registros de control acerca de las PRUs como por ejemplo el registro `CYCLE` que nos permite contar los ciclos que transcurren desde el inicio de nuestro programa. En la tabla 2.1 se muestran las direcciones de memoria a las que tienen acceso.

Aparte de las distintas direcciones de memoria mostradas anteriormente, las PRUs disponen de otro elemento que les permite almacenar datos. Este elemento es un banco de scratch-pads que otorgan la

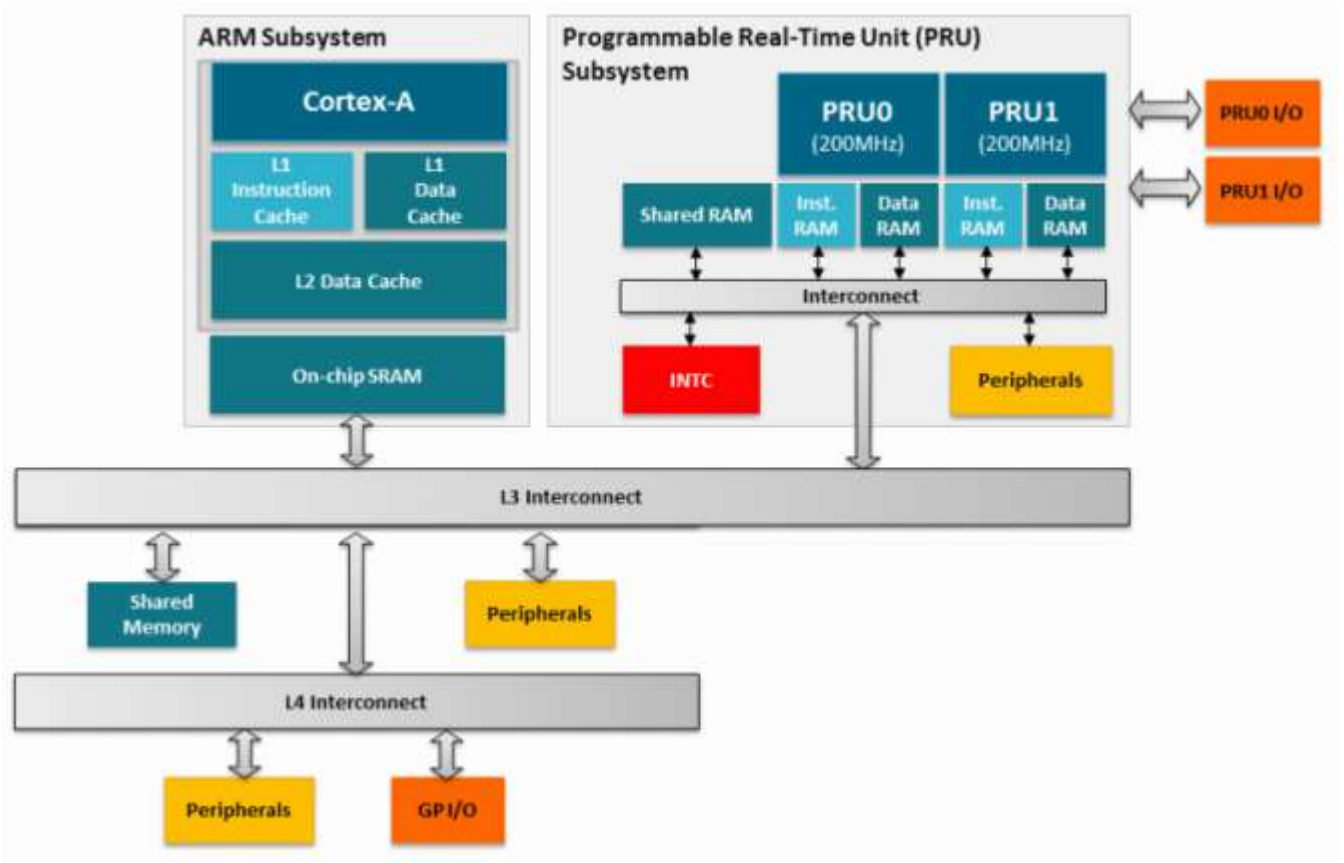


Figura 2.2: Diagrama acerca del acceso a las distintas memorias [2].

Dirección de inicio	Significado
0x0000_0000	8 kB RAM de datos de la propia PRU
0x0000_2000	8 kB RAM de datos de la otra PRU
0x0001_0000	12 kB SRAM
0x0002_0000	Interrupt Controller (INTC)
0x0002_2000	Control de la PRU0
0x0002_4000	Control de la PRU1

Tabla 2.1: Direcciones accesibles por los módulos PRUs.

posibilidad de almacenar y recuperar el valor de los registros en un solo ciclo utilizando las instrucciones XFR (XIN y XOUT). El banco de scratch-pads se compone por 3 bancos de registros (cada uno puede almacenar desde el R0 hasta el R29) y un modo directo en el que las PRU pueden enviar y recoger valores sin necesidad de almacenarlos previamente 2.3.

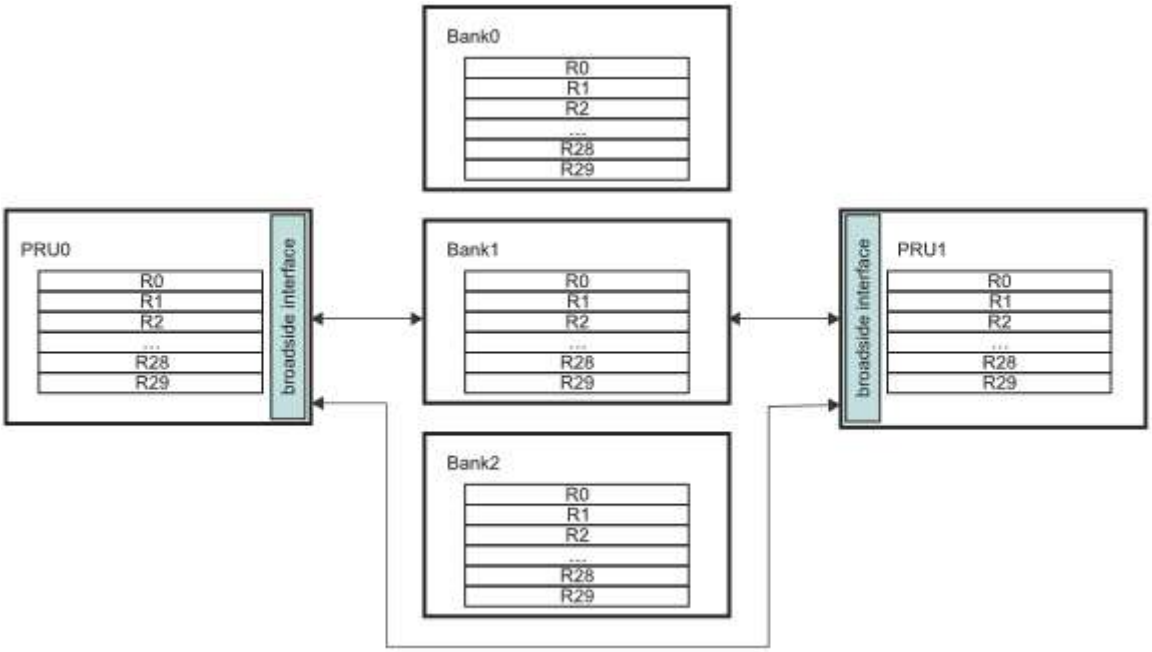


Figura 2.3: Diagrama sobre los scratch-pads.

En las instrucciones XFR, cada opción se determina con un valor numérico: el banco 1, con el valor 10; el banco 2, con el valor 11; el banco 3, con el valor 12, y el modo directo, el valor 14.

2.2.4 Interrupciones

Respecto a las interrupciones, encontramos 64 posibles eventos que pueden provocar la interrupción. De estos 64 eventos posibles, solo 16 pueden ser provocados por las PRUs. Estos eventos se pueden agrupar en 10 canales que se activan al activarse uno de los eventos que tiene asociados. Estos canales están a su vez asignados a un "host"que permiten que el canal pueda ser dirigido a una localización distinta. Solo los 2 primeros "hostsrepercuten en el subsistema PRU-ICSS.

El registro R31 de las PRUs da soporte a las interrupciones utilizando el bit 31 como estado para interrumpir a la PRU1 y el bit 30 para interrumpir a la PRU0. Para provocar un evento tenemos que escribir un 1 en el bit 5 del registro 31 indicando, en los bits [3:0], el número de interrupción causables por las PRUs. Si los bits [3:0] valieran 0, se generaría el evento pr1_pru_mst_int[0]_intr_req que corresponde con la interrupción número 16 [10]. Pese a existir 64 posibles eventos (solo 16 causables por las PRUs), estos deben asignarse a un canal y, dicho canal, a un "host". El número de canales y el de "hosts.[es] de 10, por lo que se recomienda asignar x canal a x "host"(solo pueden asignarse a un "hostz cada "host"solo puede tener un canal asignado). Se pueden asignar a cada canal tantos eventos como se desee. El "host"0 está conectado con el bit 30 del registro R31 y el "host"1 con el bit 31 del registro R31. En la figura 2.4 vemos un ejemplo de una asignación que no sigue el consejo de asignar a x "host"x canal.

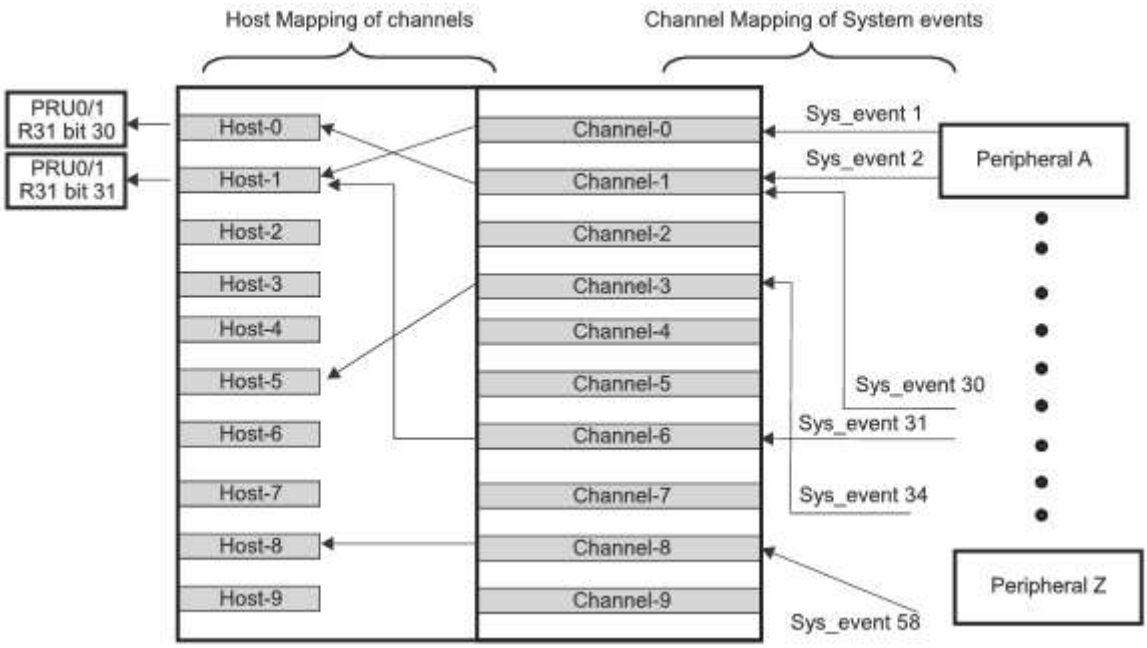


Figura 2.4: Ejemplo sobre la asignación de hosts, canales y eventos.

Para saber qué evento ha sido activado, debemos leer el registro System Interrupt Status Enabled/-Clear Registers (SECR). Este registro se encuentra dividido en 2, SECR0 y SECR1 (0x280 y 0x284 de offset respectivamente sobre 0x0002_0000). La lectura de este registro devolverá un valor en el cuál cada bit que valga 1 que el evento con el mismo número que ese bit ha sido activado (por ejemplo el bit 16 a 1 indicará que ha ocurrido el evento 16). Para limpiar el estado de los eventos, podemos escribir en el registro SECR un valor en el que cada bit a 1 represente el evento cuyo estado queremos desactivar (por ejemplo escribiendo un valor con un 1 en la posición 3 desactivamos el estado del evento 3). También se puede realizar escribiendo en el registro Status Indexed Clear Register (SICR) (con offset de 0x24 con respecto a 0x0002_0000) el número de evento cuyo estado queremos desactivar (por ejemplo si escribimos el valor 19 desactivamos el estado del evento 19).

2.3 Compilador clpru

En este apartado se explicará el funcionamiento del compilador clpru que ofrece TI para la compilación de programas en C/C++ para su ejecución en los módulos PRU. Este compilador no viene por defecto en las imágenes que ofrece TI. Sin embargo, se puede descargar en la página de descargas de la herramienta PRU-CGT [11].

2.3.1 Uso del compilador

En primer lugar, es necesario mencionar que este compilador admite la compilación de programas en C, C++ y ensamblador mediante una sola llamada al compilador. Las llamadas a este compilador se hacen siguiendo el siguiente formato [12]:

```
clpru [options] [filename] [--run_linker [linker_options] [object_files]
```

Un ejemplo de este formato podría ser el siguiente:

```
clpru file.c file2.c assembly.asm --run_linker --library=lnk.cmd --output_file=program.out
```

En este caso se desea compilar los archivos file.c, file2.c y assembly.asm estando los 2 primeros programados en lenguaje C y el último en lenguaje ensamblador que se verá en la sección 2.4. Una aclaración importante acerca de la terminación de los nombres de los archivos es que indican al compilador que tipo de archivo es, por lo cuál debe seguir unas normas. Las terminaciones recogidas se pueden ver en la tabla 2.2 aunque se puede modificar como interpreta el compilador los archivos según la terminación con determinadas opciones que se verán en el apartado 2.3.2.

Extensión	Tipo de archivo
.asm, .abs, .s* (comienza por s)	Archivo en ensamblador
.c	Archivo en C
.C	Depende del sistema operativo
.cpp, .cxx, .cc	Archivo en C++
.obj, .o*, .dll, .so	Objeto

Tabla 2.2: Terminaciones aceptadas por defecto por el compilador clpru.

2.3.2 Opciones del compilador

A continuación, se explicarán las opciones que acepta al compilador y como trabajar con ellas. No se explicarán todas las opciones ya que no se hacen uso de gran parte de ellas en el transcurso de este trabajo sino que se explicarán los parámetros utilizados incluidos en los ejemplos proporcionados por TI [13]. Si en algún caso futuro fuese necesario el uso de alguna de las opciones que no se incluyen este documento, debe referirse al manual de compilador clpru que ofrece TI [12]. Una parte de las opciones que acepta el compilador disponen de 2 formas: una corta y poco descriptiva y otra más larga que ofrece una mayor descripción. Normalmente la opción corta solo hace uso de un guión mientras que la opción larga hace uso de 2 guiones, por ejemplo la opción `-silicon_version` en formato largo o `-v` en formato corto. También, es importante mencionar que todas las opciones deben ser continuadas con un signo = y el valor que se le desee dar a dicha opción sin ser posible introducir espacios entre la opción y el igual o el igual y el valor, como por ejemplo `-v=3`. Además, se debe tener cuidado al nombrar las opciones o los valores ya que discrimina mayús./minús. Las opciones permiten una nomenclatura más cómoda que consiste en no hacer uso de un igual sino escribir juntos la opción y el valor de la misma como por ejemplo `-v3` equivaldría a `-v=3`.

La tabla 2.3 muestra las opciones utilizadas a lo largo de este trabajo.

Tabla 2.3: Opciones de compilación más utilizadas.

Opción	Abreviatura	Función
–include_path=*directory*	-i	Añade *directory* a la lista de directorios en los que el compilador busca los archivos llamados en las sentencias `#include files`. Esta opción de acepta varias veces siempre y cuando estén separadas por un espacio entre ellas.
–silicon_version={1|2|3}	-v	Selecciona la versión del procesador.
Continúa en la página siguiente		

Opción	Abreviatura	Función
–opt_level=*n*	-O	Indica el nivel de optimización del compilador. Los distintos valores para *n* son Off, 0, 1, 2 y 3.
–display_error_number=*number*		Muestra el identificador numérico del diagnosis.
–endian={big|little}		Especifica el formato `little endian` o `big endian` de cara a la compilación. Por defecto se utiliza `little endian`.
–hardware_mac={on|off}		Habilita o deshabilita el uso de hardware Multiply and Accumulate (MAC) disponible en algunos núcleos PRU [14]. Se puede utilizar para acelerar multiplicaciones de 32 bits u operaciones de multiplicar y acumular de 64 bits. Por defecto se encuentra en `off`.
–obj_directory=*directory*	-fr	Especifica un directorio para los archivos objeto.
–pp_directory=*directory*		Especifica un directorio para los archivos del prepocesador.
–preproc_dependency[=*filename*]	-ppd	Continúa con la compilación después del preprocesado.
–preproc_with_compile	-ppa	Continúa con la compilación después del preprocesado.
–run_linker	-z	Ejecuta el enlazador (linker) sobre los archivos especificados. Todas las opciones especificadas a continuación se envían al enlazador.
–output_file=*filename*	-o	Especifica el nombre del archivo final.
–map_file=*filename*	-m	Produce un archivo con las secciones de entrada y salida con el nombre *filename*.
–library=*filename*	-l	Especifica una librería.
–output_file=*filename*	-fe	Especifica un nombre para el archivo final de compilación.
–reread_libs	-x	Fuerza a volver a leer las librerías.
–warn_sections	-w	Muestra un mensaje cuando una sección de salida indefinida es creada.
–stack_size=*size*	[-]-stack	Define el tamaño de la pila.
–heap_size=*size*	[-]-heap	Define el tamaño de la pila para memoria dinámica.

2.4 Lenguaje ensamblador

En este apartado se explicarán las distintas instrucciones que componen el lenguaje ensamblador de los módulos PRU que utilizo a lo largo del trabajo.

2.4.1 Nomenclatura

En primer lugar es imprescindible explicar como se van a representar los distintos conceptos que se utilizarán en la explicación de las distintas instrucciones. En la tabla 2.4 se puede ver la explicación de cada término con algunos ejemplos de cada caso.

Parámetro	Significado	Ejemplos
REG1, REG2, ...	Cualquier campo de un registro desde 8 bits a 32 bits.	r0 (registro 0 entero) r1.w0 (primera palabra del registro 1) r3.b2 (tercer byte del registro 3)
Rn1, Rn2, ...	Un registro de 32 bits entero.	r0 (registro 0 entero) r1 (registro 1 entero)
Rn.tx	Especifica un solo bit de cualquier registro.	r0.t23 (bit 23 del registro 0) r1.b2.t5 (quinto bit del tercer byte del registro 1)
Rn.bx	Especifica un byte de cualquier registro. Debe ser `b0`, `b1`, `b2` o `b3`.	r0 (registro 0 entero) r1.w0 (primera palabra del registro 1) r3.b2 (tercer byte del registro 3)
Rn.wx	Especifica dos bytes (palabra) de cualquier registro. Debe ser `w0` (bytes 0 y 1), `w1` (bytes 1 y 2) o `w2` (bytes 2 y 3).	r0.w0 (primeros 2 bytes del registro 0) r1.w1 (segundo y tercer bytes del registro 1)
Cn1, Cn2, ...	Especifica la constante utilizada de la tabla de constantes [15].	c0, c1
LABEL	Especifica una etiqueta con o sin paréntesis. También puede especificar la dirección de la instrucción.	loop1 (loop1) 0x0000
IM(n)	Especifica un valor numérico (valor inmediato). También son aceptables etiquetas y direcciones de registros.	#23 0b0110 0xF2 2+2 &r3.w2
OP(n)	La unión de REG e IM(n).	r0 r1.w0 #0x7F 1«3 loop1 &r2.w3

Tabla 2.4: Nomenclatura del lenguaje ensamblador de los módulos PRU.

2.4.2 Instrucciones

A continuación, se mostrarán las instrucciones utilizadas explicando su funcionamiento, la sintaxis para utilizarla y la operación que llevan a cabo tal y como se muestra en la tabla 2.5.

Tabla 2.5: Instrucciones de lenguaje ensamblador de los módulos PRU..

Descripción	Sintaxis	Operación
Suma de enteros sin signo	ADD REG1, REG2, OP(255)	REG1 = REG2 + OP(255) carry = ((REG2 + OP(255)) »bit-width(REG1))
Resta de enteros sin signo	SUB REG1, REG2, OP(255)	REG1 = REG2 + OP(255) carry = ((REG2 + OP(255)) »bit-width(REG1))& 1
Establecer un determinado bit a 0	CLR REG1, REG2, OP(31)	REG1 = REG2 & (1 «(OP(31) & 0x1f))
Establecer un determinado bit a 1	SET REG1, REG2, OP(31)	REG1 = REG2 | (1 «(OP(31) & 0x1f))
Copia el valor de un registro a otro registro (con extensión de 0)	MOV REG1, REG2	REG1 = REG2
Carga un valor determinado a un registro (con extensión de 0) (LDI32 especifica 32 bits)	LDI REG1, IM(65535)	REG1 = IM(65535)
Lee y guarda en los registros cierta cantidad de bytes de la dirección seleccionada	LBBO &REG1, Rn2, OP(255), IM(124)	memcpy(offset(REG1), Rn2+OP(255), IM(124))
Almacena en la dirección seleccionada cierta cantidad de bytes de los registros	SBBO &REG1, Rn2, OP(255), IM(124)	memcpy(Rn2+OP(255), offset(REG1), IM(124))
Carga los valores de un "scratch pad" seleccionado en los registros elegidos	XIN IM(253), REG, IM(124)	Lee del "scratch pad" IM(253) IM(124) bytes empezando en el registro REG
Guarda los valores de los registros elegidos en un "scratch pad" seleccionado	XOUT IM(253), REG, IM(124)	Guarda en el "scratch pad" IM(253) IM(124) bytes empezando en el registro REG
Salta incondicionalmente a la dirección o etiqueta seleccionada	JMP OP(65535)	Puntero de instrucción = OP(65535)
Salta si el valor inmediato es mayor que el valor del registro	QBGT LABEL, REG1, OP(255)	Salta si OP(255) >REG1

Continúa en la página siguiente

Nombre	Sintaxis	Operación
Salta si el valor inmediato es menor que el valor del registro	QBLT LABEL, REG1, OP(255)	Salta si OP(255) <REG1
Salta si el valor de sus parámetros es el mismo	QBEQ LABEL, REG1, OP(255)	Salta si OP(255) == REG1
Salta si el valor de sus parámetros es distinto	QBNE LABEL, REG1, OP(255)	Salta si OP(255) != REG1
Salta si el bit seleccionado vale 1	QBBS LABEL, REG1, OP(31)	Salta si (REG1 & (1 «(OP(31) & 0x1f))) == 1
Espera hasta que el bit seleccionado valga 1	WBS REG1, OP(31)	Continúa si (REG1 & (1 «(OP(31) & 0x1f))) == 1
Para la PRU	HALT	Deshabilita la PRU

Por último, es importante conocer que algunos de los registros de las PRUs albergan usos especiales [16]. Por ejemplo, la palabra 2 del registro 3 (`R3.w2`) contiene la dirección de retorno y el registro 14 contiene el valor que pasado como parámetro a la función en ensamblador y, también, el valor de retorno al volver de la función.

2.4.3 Prudebug

Esta herramienta se basa en la lectura y escritura de las posiciones de memoria correspondientes al módulo PRU-ICSS en los distintos procesadores que lo incluyen [17]. Para utilizarlo es necesario descargarlo del repositorio en GitHub [17] y compilar el programa. Al ejecutarlo acepta distintos parámetros como indicar el procesador en el que se ejecuta. Una vez ejecutado, podemos consultar los distintos comandos introduciendo "help". Entre ellos encontramos el comando "G"que inicia la PRU, "HALT"que detiene la PRU, "BR"que añade un punto de ruptura o "D"que muestra el contenido de la memoria de datos. También incluye los comandos "PRU número de la PRU"para cambiar la PRU que está siendo utilizada y el comando R"para mostrar el contenido de los registros. Un ejemplo de como los muestra es la imagen 2.5.

2.5 Módulo remoteproc

Una vez analizados todos los pasos enteriores, es posible continuar en el objetivo de hacer funcionar las PRUs. En este apartado se explicará que es el módulo remoteproc y cómo utilizarlo para el propósito de este trabajo.

2.5.1 Adición al kernel

En primer lugar, es importante conocer su introducción en los sistemas Linux ya que es posible que nuestra BeagleBone Black no disponga de él. Este driver se introdujo en octubre de 2016 permitiendo tratar a los distintos procesadores remotos de una forma sencilla que otorga las posibilidades de encender el procesador, apagarlo o cargar cierto firmware en él [18]. Estos procesadores remotos son todos aquellos

```
PRU0> PRU 1
Active PRU is PRU1.

PRU1> r
Register info for PRU1
    Control register: 0x00000001
      Reset PC:0x0000  STOPPED, FREE_RUN, COUNTER_DISABLED, NOT_SLEEPING, PROC_DISABLED

    Program counter: 0x0083
      Current instruction: SET R30, R30, 0x0c

    R00: 0x00000012    R08: 0xfac9866e    R16: 0xeaf833fa    R24: 0x418958c1
    R01: 0x00000000    R09: 0x00002ffe    R17: 0xf25f8ce2    R25: 0x08ff8cc1
    R02: 0x000000f6    R10: 0x00010000    R18: 0xfb699115    R26: 0x00000fff
    R03: 0x00a1f675    R11: 0x00010000    R19: 0x08ddb8e5    R27: 0x00000000
    R04: 0x00020024    R12: 0x0e988755    R20: 0x04071478    R28: 0x00000000
    R05: 0xa6f59cf0    R13: 0xbdfc6151    R21: 0xc60304b4    R29: 0x00000000
    R06: 0x6a461d3e    R14: 0x00000000    R22: 0x0f89eee2    R30: 0x00000000
    R07: 0x3804d411    R15: 0x269dca7b    R23: 0x68565b9f    R31: 0x00000000
```

Figura 2.5: Valores mostrados por prudebug.

que no estén ejecutando un sistema Linux. En este caso, esos procesadores serían ambas PRUs. En las versiones anteriores del kernel este framework no está disponible, por ello sería conveniente actualizar el SO de la BeagleBone Black a una versión más reciente. Para esto se puede seguir la guía que ofrece BeagleBone para actualizar el SO. En ella nos guía a un enlace en el que se pueden obtener distintas imágenes para los distintos dispositivos. Entre ellas se encuentran distintas imágenes para todas los dispositivos de BeagleBone [19].

2.5.2 Directorio /sys/class/remoteproc/

A continuación, se describirá el directorio `/sys/class/remoteproc/`. Este directorio contiene distintos subdirectorios en los cuáles se encuentran los datos de cada procesador de forma individual. Existirá un subdirectorio por cada procesador, tanto aquellos procesadores remotos como el que se encuentra haciendo uso del sistema Linux. Estos subdirectorios tienen la forma de `remoteprocN/` dónde N es el número de procesador cuya información contiene. En el caso específico de la BeagleBone Black, es posible ver 3 subdirectorios distintos, `remoteproc0/`, `remoteproc1/` y `remoteproc2/`. Como ya se ha discutido previamente, cada uno de estos directorios corresponde a un procesador, en este caso, el subdirectorio `remoteproc0/` corresponde al procesador que se encuentra ejecutando el sistema Linux mientras que los subdirectorios `remoteproc1/` y `remoteproc2/` corresponden a las 2 PRUs disponibles en el SoC. Más específicamente, el subdirectorio `remoteproc1/` corresponde a la PRU0 mientras que el subdirectorio `remoteproc2/` corresponde a la PRU1.

2.5.3 Directorio /sys/class/remoteproc/remoteprocN/

Dentro de cada uno de estos directorios (cuya ruta es `/sys/class/remoteproc/remoteprocN/`) se pueden encontrar 2 archivos [20]. Estos archivos son `state` y `firmware`. También es posible que en futuras versiones de los SSOO disponibles para la BeagleBone Black se encuentren archivos adicionales en esos subdirectorios ya que incluirá versiones más nuevas del kernel. Estos archivos son `name`, `coredump` y `recovery` [20]. No se explicará la funcionalidad de estos archivos ya que no son necesarios para el propósito de este trabajo.

2.5.3.1 Archivo state

El archivo `state` nos ofrece información acerca del estado del procesador al que pertenece el directorio. Este archivo se compone únicamente de la palabra que describa el estado del procesador. Se puede obtener el estado del procesador mediante el comando:

```
sudo cat /sys/class/remoteproc/remoteproc1/state
sudo cat /sys/class/remoteproc/remoteproc2/state
```

Sus opciones son `offline`, `suspended`, `running`, `crashed` e `invalid`. El estado `offline` indica que el procesador remoto se encuentra apagado y que, por tanto, no está ejecuntando ningún programa. El estado suspended indica que el procesador remoto se encuentra suspendido y debe ser "despertado"para poder recibir mensajes. El estado `running` indica que el procesador se encuentra activo y ejecutando el firmware correspondiente. El estado `crashed` indica que ha habido un problema en el procesador remoto. El estado `invalid` se devuelve cuando se desconoce el estado en el que se encuentra el procesador remoto.

En este archivo se pueden escribir 2 valores distintos siendo estos `start` y `stop` con los comandos:

```
sudo echo "start" > /sys/class/remoteproc/remoteproc1/state
sudo echo "stop" > /sys/class/remoteproc/remoteproc1/state

sudo echo "start" > /sys/class/remoteproc/remoteproc2/state
sudo echo "stop" > /sys/class/remoteproc/remoteproc2/state
```

Al escribir `start` se intentará pasar al estado `running` ejecutando el firmware correspondiente. Al escribir `stop` se intentará parar el procesador y este pasará al estado `offline`.

2.5.3.2 Archivo firmware

Además, es apreciable la existencia del archivo `firmware`. Este archivo contiene el nombre del programa del firmware que se intentará ejecutar en el procesador remoto una vez se escriba la palabra `start` en el archivo `state`. Al hacerlo, se intentará obtener el programa con ese nombre desde el directorio `/lib/firmware/`. Por ejemplo, si el archivo `firmware` contiene la palabra "prueba", al iniciar los módulos PRU (escribiendo start en el archivo `state`) se buscará un ejecutable con el nombre "prueba"dentro del directorio `/lib/firmware`. Si no existe un ejecutable con el mismo nombre especificado en el archivo `firmware`, ocurrirá un error y el procesador remoto no se iniciará. Es importante matizar que ambas PRUs comparten el firmware en la ruta `/lib/firmware/`, es decir, ambas PRUs pueden hacer uso de los mismos ejecutables siempre y cuando se encuentren en el directorio `/lib/firmware/` y el nombre está correctamente especificado en el archivo firmware.

2.6 Módulo RPMsg

El módulo RPMsg nos proporciona la posibilidad en enviar mensajes a las PRUs y que estas también envíen mensajes al ARM. Este módulo solicita recursos a través de la tabla de recursos necesaria para el módulo remoteproc y se construye sobre Virtual Input Output (VirtIO). VirtIO hace uso de las estructuras de datos VirtQueue para llevar a cabo operaciones en las estructuras VRing.

La implementación de este módulo consta de 2 partes: la implementación por parte del ARM y la implementación por parte de la PRU. En el lado del ARM, la comunicación se recibe en el espacio del kernel. Para ello, se dispone de una interfaz que crea un archivo en el que se puede leer o escribir para

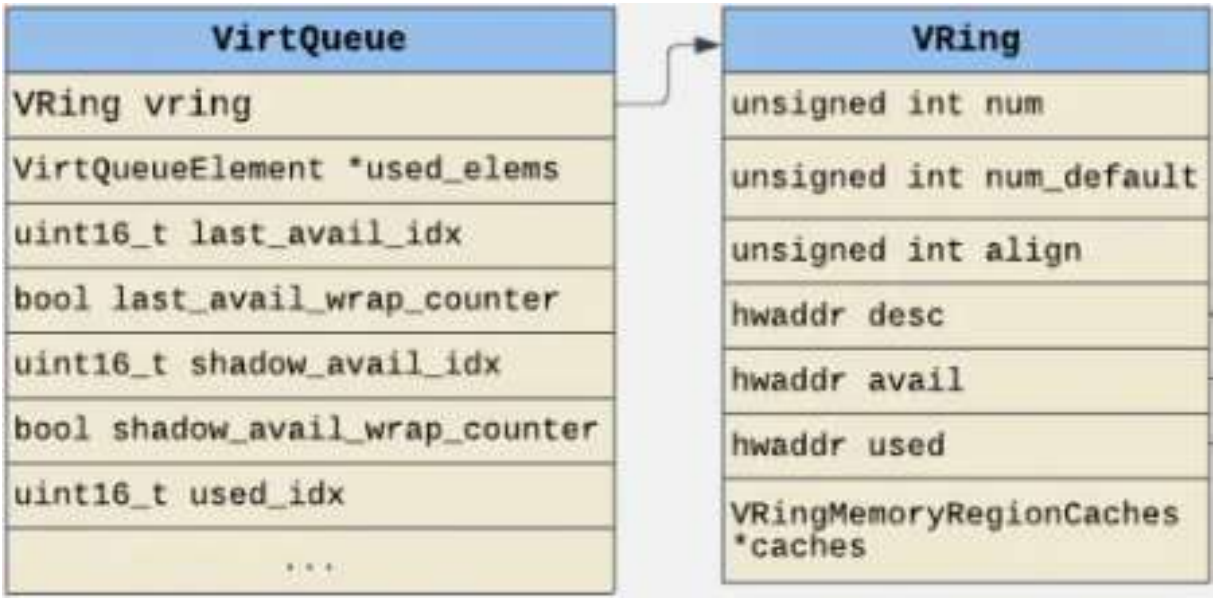


Figura 2.6: Estructuras de datos VirtQueue y VRing [3].

recibir o mandar mensajes. Este archivo se encuentra en el directorio `/dev/`. En el lado de la PRU, esta debe reservar el espacio de memoria para las VRINGS (memoria compartida que mantiene temporalmente los mensajes) (una para cada sentido). A continuación, debe crear el canal RPMsg (ubicado en `/dev/`).

En la figura 2.7a, se ve el proceso para enviar mensajes desde el ARM hacia la PRU. En el primer paso se aloja un búfer (1a si es nuevo y 1b si se ha utilizado). En el segundo paso se copia el mensaje en el búfer y, en el tercer paso, se coloca dicho búfer en la lista de disponibles dentro de la vring1. En el cuarto paso se escribe el índice (1) en el buzón 2, lo cual desata el quinto paso, que consiste en la notificación de que existe un mensaje. En el sexto paso se recupera el mensaje del búfer y, en el séptimo paso, se copia en el dato. A continuación, se coloca dicho búfer en la lista de utilizados y, por último, se escribe el índice (1) en el buzón 3.

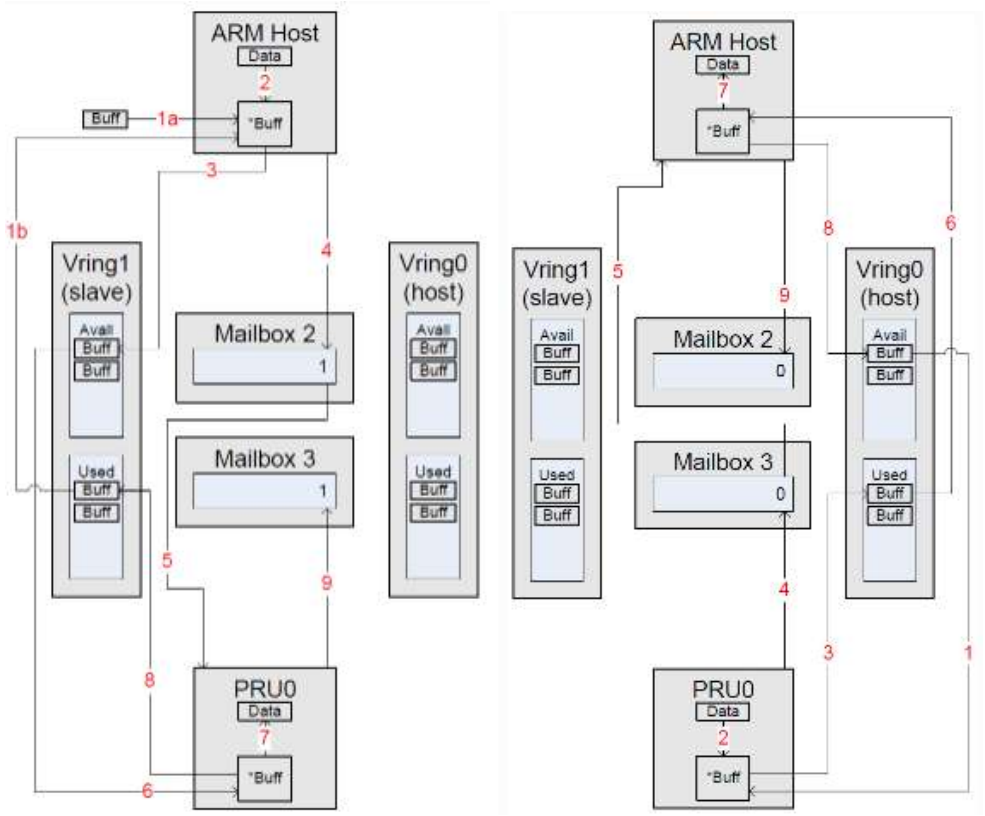
En la figura 2.7b, se ve el proceso para enviar mensajes desde la PRU hacia el ARM. Los pasos son iguales que en caso anterior exceptuando el índice que se escribe (0 en vez de 1), la vring utilizada y que tanto el búfer se debe obtener de los disponibles y, al terminar, se coloca de nuevo en esa lista.

2.7 Configuración de pines

De cara a la realización del trabajo es esencial ser capaces de obtener la señal creada por los módulos PRU a través de los pines de la BeagleBone Black. En este apartado veremos cómo configurarlos según sea necesario y las distintas herramientas para ello.

2.7.1 Herramienta config-pin

Para ello, las imágenes que nos ofrece el grupo BeagleBone incorporan una herramienta llamada `config-pin` [21] con la que se podrá listar los distintos modos de cada uno de los pines, modificar el modo en el que se encuentra y comprobar el modo en el que esté configurado en ese determinado momento. Al invocar esta herramienta sin ningún otro parámetro nos arrojará las distintas opciones mostradas en la figura 2.8.



(a) Proceso para el envío de mensajes desde el ARM hacia la PRU.

(b) Proceso para el envío de mensajes desde la PRU hacia el ARM.

Figura 2.7: Proceso para el envío de mensajes a través del módulo RPMsg.

2.7.1.1 Opción -c

En primer lugar, existe la opción de modificar el modo de uno o varios pines haciendo uso de un archivo. En este archivo se debe indicar el pin deseado mediante `[P8|P9]_[Pin]` seguido del modo deseado. Por ejemplo, al incluir `P8_30 pruout` y ejecutarlo, se cambiaría el modo del pin P8_30 a pruout. Para modificar más pines simplemente se repite el ejemplo anterior en una nueva línea por cada pin. Esta opción es más cómoda a partir de una cierta cantidad de pines a cambiar.

2.7.1.2 Opción -l

En segundo lugar, la opción –l permite listar los distintos modos del pin seleccionado. Una posible salida es:

```
Available modes for P8\_30 are: default gpio gpio\_pu gpio\_pd gpio\_input pruout pruin
```

2.7.1.3 Opción -q

En tercer lugar, se puede comprobar el actual estado de un determinado pin con la opción –q. Esta opción nos arrojará un valor de los que recibimos al ejecutar la opción –l. Por último, es posible cambiar el modo del pin especificando el pin y el modo deseado.

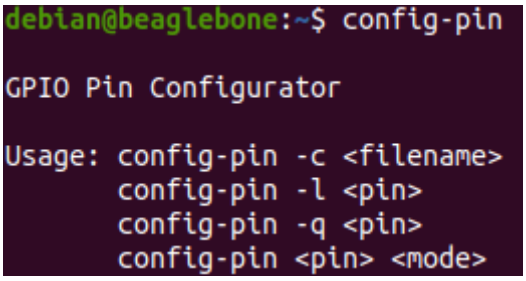


Figura 2.8: Opciones de la herramienta `config-pin` [4]

2.7.1.4 Opción directa

En último lugar, existe la opción de cambiar el modo de un solo pin en concreto a un determinado modo. Esta opción será más cómoda cuándo se desee cambiar de modo pocos pines ya que, si es necesario cambiar más pines, será más apropiado utilizar la opción -c (2.7.1.1).

2.7.2 Archivo uEnv.txt

Al principio no se dispondrá de todos los pines para poder elegir su modo. Esto se debe a que, en un inicio, la BeagleBone Black está preparada para habilitar ciertas posibilidades como reproducción de video, de audio, conexión inalámbrica o ADC al arrancarse. Estas posibilidades no permiten modificar el modo de ciertos pines ya que están reservados para esos fines. Por este motivo, si mostramos el contenido de la carpeta que contiene los datos de los pines (`/sys/devices/platform/ocp/`) se puede ver que los pines cuyo modo permite modificar son del `P8_07` al `P8_19`, `P8_26`, del `P9_11` al `P9_24`, `P9_26`, `P9_27`, `P9_30`, `P9_41`, `P9_42`, `P9_91` y `P9_92`. Para el desarrollo del trabajo usaremos los pines `P8_46`, `P8_45`, `P8_44`, `P8_43`, `P8_42`, `P8_41`, `P8_40`, `P8_39`, `P8_30`, `P8_29`, `P8_28`, `P8_27` y `P8_21`. Estos pines están inicialmente utilizados para el vídeo y para la lectura de embedded Multi-Media Card (eMMC) [22]. Para poder utilizar estos pines como pruout será necesario deshabilitar la lectura de eMMC por lo que necesitaremos utilizar una tarjeta microSD con un SO proporcionado por Beagleboard.org siguiendo las instrucciones que proporciona BeagleBone [4]. En este trabajo, he usado la imagen de Debian 10.3 para funciones Internet of things (IoT) con fecha de 2020-04-06 [23]. Para poder deshabilitar estas opciones y así tener la posibilidad de elegir el modo de los pines es necesario acceder al archivo /boot/uEnv.txt [22]. En él, se debe descomentar las siguientes líneas:

Listado 2.1: Líneas en uEnv para vídeo y eMMC

```
disable_uboot_overlay_video=1
disable_uboot_overlay_emmc=1
```

A continuación se debe reiniciar la BeagleBone Black y, para comprobar la posibilidad de cambiar de modo pines previamente no disponibles, se puede volver a mostrar el contenido de `/sys/devices/platform/ocp/` y se podrá comprobar que se han añadido los pines que usaremos para el trabajo.

2.7.3 Directorio /sys/kernel/debug/pinctrl/

Por otro lado, existe un directorio dentro del sistema que contiene archivos con información acerca de los pines . Su ruta es /sys/kernel/debug/pinctrl/44e10800.pinmux.pinctrl-single/ [5] (el nombre del último

directorio puede cambiar de un sistema a otro). En él, se pueden ver distintos archivos. Para la comprobación del estado y del uso de los pines, se ha hecho uso de los archivos pins y pinmux-pins. Al mostrar el contenido del archivo pins con el comando sudo cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins se obtendrá una salida como la siguiente:

```
registered pins: 142
pin 0 (PIN0) 44e10800 00000031 pinctrl-single
pin 1 (PIN1) 44e10804 00000031 pinctrl-single
...
```

Aparecerá una línea descriptiva por cada uno de los 142 pines disponibles con el mismo formato que tienen anteriormente los pines 0 y 1. En este caso, los pines 0 y 1 corresponden a `P8_25` y `P8_24` respectivamente. Esto se puede comprobar por el offset (últimos 3 dígitos en `44e10800` o `44e10804` [5]) de acuerdo a las siguientes imágenes. La primera [24] muestra los datos (offset, todos los modos disponibles y el número de General Purpose Input/Output (GPIO) que supone) acerca de los pines en P8 mientras que la segunda [25] muestra los datos acerca de los pines en P9.

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P8_01		Offset from:		DGND										Ground
P8_02		44e10800		DGND										Ground
P8_03	6	0x818/018	38	GPIO1_6	gpio1[6]						mmc1_dat6	gpmc_ad6	R9	Allocated emmc2
P8_04	7	0x81c/01c	39	GPIO1_7	gpio1[7]						mmc1_dat7	gpmc_ad7	T9	Allocated emmc2
P8_05	2	0x808/008	34	GPIO1_2	gpio1[2]						mmc1_dat2	gpmc_ad2	R8	Allocated emmc2
P8_06	3	0x80c/00c	35	GPIO1_3	gpio1[3]						mmc1_dat3	gpmc_ad3	T8	Allocated emmc2
P8_07	36	0x890/090	66	TIMER4	gpio2[2]					timer4		gpmc_advn_ale	R7	
P8_08	37	0x894/094	67	TIMER7	gpio2[3]					timer7		gpmc_oen_ren	T7	
P8_09	39	0x89c/09c	69	TIMER5	gpio2[5]					timer5		gpmc_be0n_cle	T6	
P8_10	38	0x898/098	68	TIMER6	gpio2[4]					timer6		gpmc_wen	U6	
P8_11	13	0x834/034	45	GPIO1_13	gpio1[13]	pr1_pru0_pru_r30_15		eQEP2B_in	mmc2_dat1	mmc1_dat5	lcd_data18	gpmc_ad13	R12	
P8_12	12	0x830/030	44	GPIO1_12	gpio1[12]	pr1_pru0_pru_r30_14		EQEP2A_IN	MMC2_DAT0	MMC1_DAT4	LCD_DATA19	GPMC_AD12	T12	
P8_13	9	0x824/024	23	EHRPWM2B	gpio0[23]			ehrpwm2B	mmc2_dat5	mmc1_dat1	lcd_data22	gpmc_ad9	T10	
P8_14	10	0x828/028	26	GPIO0_26	gpio0[26]			ehrpwm2_tripzone_in	mmc2_dat6	mmc1_dat2	lcd_data21	gpmc_ad10	T11	
P8_15	15	0x83c/03c	47	GPIO1_15	gpio1[15]	pr1_pru0_pru_r31_15		eQEP2_strobe	mmc2_dat3	mmc1_dat7	lcd_data16	gpmc_ad15	U13	
P8_16	14	0x838/038	46	GPIO1_14	gpio1[14]	pr1_pru0_pru_r31_14		eQEP2_index	mmc2_dat2	mmc1_dat6	lcd_data17	gpmc_ad14	V13	
P8_17	11	0x82c/02c	27	GPIO0_27	gpio0[27]			ehrpwm0_synco	mmc2_dat7	mmc1_dat3	lcd_data20	gpmc_ad11	U12	
P8_18	35	0x88c/08c	65	GPIO2_1	gpio2[1]	mcasp0_fsr			mmc2_clk	gpmc_wait1	lcd_memory_clk	gpmc_clk_mux0	V12	
P8_19	8	0x820/020	22	EHRPWM2A	gpio0[22]			ehrpwm2A	mmc2_dat4	mmc1_dat0	lcd_data23	gpmc_ad8	U10	
P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13			mmc1_cmd	gpmc_be1n	gpmc_csn2	V9	Allocated emmc2
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12			mmc1_clk	gpmc_clk	gpmc_csn1	U9	Allocated emmc2
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]						mmc1_dat5	gpmc_ad5	V8	Allocated emmc2
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]						mmc1_dat4	gpmc_ad4	U8	Allocated emmc2
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]						mmc1_dat1	gpmc_ad1	V7	Allocated emmc2
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]						mmc1_dat0	gpmc_ad0	U7	Allocated emmc2
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]							gpmc_csn0	V6	
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8				gpmc_a8	lcd_vsync	U5	Allocated HDMI
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10				gpmc_a10	lcd_pclk	V5	Allocated HDMI
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9				gpmc_a9	lcd_hsync	R5	Allocated HDMI
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]	pr1_pru1_pru_r31_11	pr1_pru1_pru_r30_11				gpmc_a11	lcd_ac_bias_en	R6	Allocated HDMI
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn		uart5_rxd	mcasp0_axr1	eQEP1_index	gpmc_a18	lcd_data14	V4	Allocated HDMI
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn		mcasp0_axr3	mcasp0_ahclkx	eQEP1_strobe	gpmc_a19	lcd_data15	T5	Allocated HDMI
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn		mcasp0_axr3	mcasp0_fsr	eQEP1B_in	gpmc_a17	lcd_data13	V3	Allocated HDMI
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn		mcasp0_axr2	mcasp0_ahclkr	ehrpwm1B	gpmc_a15	lcd_data11	U4	Allocated HDMI
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn		mcasp0_axr2	mcasp0_aclkr	eQEP1A_in	gpmc_a16	lcd_data12	V2	Allocated HDMI
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn			mcasp0_axr0	ehrpwm1A	gpmc_a14	lcd_data10	U3	Allocated HDMI
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn		uart5_txd	mcasp0_aclkx	ehrpwm1_tripzone_in	gpmc_a12	lcd_data8	U1	Allocated HDMI
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn		uart5_rxd	mcasp0_fsx	ehrpwm0_synco	gpmc_a13	lcd_data9	U2	Allocated HDMI
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6		eQEP2_index		gpmc_a6	lcd_data6	T3	Allocated HDMI
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7	pr1_edio_data_out7	eQEP2_strobe		gpmc_a7	lcd_data7	T4	Allocated HDMI
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4		eQEP2A_in		gpmc_a4	lcd_data4	T1	Allocated HDMI
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5		eQEP2B_in		gpmc_a5	lcd_data5	T2	Allocated HDMI
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2		ehrpwm2_tripzone_in		gpmc_a2	lcd_data2	R3	Allocated HDMI
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3		ehrpwm0_synco		gpmc_a3	lcd_data3	R4	Allocated HDMI
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0		ehrpwm2A		gpmc_a0	lcd_data0	R1	Allocated HDMI
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1		ehrpwm2B		gpmc_a1	lcd_data1	R2	Allocated HDMI
P9 Header	cat $PINS	ADDR +	GPIO NO.	Name	Mode 7	Mode 6	Mode 5	Mode 4	Mode 3	Mode 2	Mode 1	Mode 0	CPU	

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P9_01		44e10000		GND										Ground
P9_02		Offset from:		GND										Ground
P9_03		44e10800		DC_3.3V										250mA Max Current
P9_04				DC_3.3V										250mA Max Current
P9_05				VDD_5V										1A Max Current
P9_06				VDD_5V										1A Max Current
P9_07				SYS_5V										250mA Max Current
P9_08				SYS_5V										250mA Max Current
P9_09				PWR_BUT										5V Level (pulled up PMIC)
P9_10				SYS_RESETn								RESET_OUT	A10	
P9_11	28	0x870/070	30	UART4_RXD	gpio0[30]	uart4_rxd_mux2		mmc1_sdcd	rmii2_crs_dv	gpmc_csn4	mii2_crs	gpmc_wait0	T17	All GPIOs to 4-6mA output
P9_12	30	0x878/078	60	GPIO1_28	gpio1[28]	mcasp0_aclkr_mux3		gpmc_dir	mmc2_dat3	gpmc_csn6	mii2_col	gpmc_be1n	U18	and approx. 8mA on input.
P9_13	29	0x874/074	31	UART4_TXD	gpio0[31]	uart4_txd_mux2		mmc2_sdcd	rmii2_rxerr	gpmc_csn5	mii2_rxerr	gpmc_wpn	U17	
P9_14	18	0x848/048	50	EHRPWM1A	gpio1[18]	ehrpwm1A_mux1		gpmc_a18	mmc2_dat1	rgmii2_td3	mii2_txd3	gpmc_a2	U14	
P9_15	16	0x840/040	48	GPIO1_16	gpio1[16]	ehrpwm1_tripzone_input		gpmc_a16	mii2_txen	rmii2_tctl	gmii2_txen	gpmc_a0	R13	
P9_16	19	0x84c/04c	51	EHRPWM1B	gpio1[19]	ehrpwm1B_mux1		gpmc_a19	mmc2_dat2	rgmii2_td2	mii2_txd2	gpmc_a3	T14	
P9_17	87	0x95c/15c	5	I2C1_SCL	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0	A16	
P9_18	86	0x958/158	4	I2C1_SDA	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1	B16	
P9_19	95	0x97c/17c	13	I2C2_SCL	gpio0[13]		pr1_uart0_rts_n	spi1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_rtsn	D17	Allocated I2C2
P9_20	94	0x978/178	12	I2C2_SDA	gpio0[12]		pr1_uart0_cts_n	spi1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn	D18	Allocated I2C2
P9_21	85	0x954/154	3	UART2_TXD	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0	B17	
P9_22	84	0x950/150	2	UART2_RXD	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk	A17	
P9_23	17	0x844/044	49	GPIO1_17	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1	V14	
P9_24	97	0x984/184	15	UART1_TXD	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd		I2C1_SCL	dcan1_rx	mmc2_sdwp	uart1_txd	D15	
P9_25	107	0x9ac/1ac	117	GPIO3_21	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7	EMU4_mux2	mcasp1_axr1	mcasp0_axr3	eQEP0_strobe	mcasp0_ahclkx	A14	Allocated mcasp0_pins
P9_26	96	0x980/180	14	UART1_RXD	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd		I2C1_SDA	dcan1_tx	mmc1_sdwp	uart1_rxd	D16	
P9_27	105	0x9a4/1a4	115	GPIO3_19	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5	EMU2_mux2	mcasp1_fsx	mcasp0_axr3	eQEP0B_in	mcasp0_fsr	C13	Allocated mcasp0_pins
P9_28	103	0x99c/19c	113	SPI1_CS0	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3	eCAP2_in_PWM2_out	spi1_cs0	mcasp0_axr2	ehrpwm0_synci	mcasp0_ahclkr	C12	Allocated mcasp0_pins
P9_29	101	0x994/194	111	SPI1_D0	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1	mmc1_sdcd_mux1	spi1_d0		ehrpwm0B	mcasp0_fsx	B13	Allocated mcasp0_pins
P9_30	102	0x998/198	112	SPI1_D1	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2	mmc2_sdcd_mux1	spi1_d1		ehrpwm0_tripzone	mcasp0_axr0	D12	Allocated mcasp0_pins
P9_31	100	0x990/190	110	SPI1_SCLK	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0	mmc0_sdcd_mux1	spi1_sclk		ehrpwm0A	mcasp0_aclkx	A13	Allocated mcasp0_pins
P9_32				VADC										1.8 ADC Volt. Ref.
P9_33				AIN4									C8	1.8V input
P9_34				AGND										Ground for ADC
P9_35				AIN6									A8	1.8V input
P9_36				AIN5									B8	1.8V input
P9_37				AIN2									B7	1.8V input
P9_38				AIN3									A7	1.8V input
P9_39				AIN0									B6	1.8V input
P9_40				AIN1									C7	1.8V input
P9_41A	109	0x9b4/1b4	20	CLKOUT2	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16	timer7_mux1	clkout2	tclkin		xdma_event_intr1	D14	Both to P21 of P11
P9_41B		0x9a8/1a8	116	GPIO3_20	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6	emu3	Mcasp1_axr0		eQEP0_index	mcasp0_axr1	D13	Both to P21 of P11
P9_42A	89	0x964/164	7	GPIO0_7	gpio0[7]	xdma_event_intr2	mmc0_sdwp	spi1_sclk	pr1_ecap0_ecap_capin_apwm_o	spi1_cs1	uart3_txd	eCAP0_in_PWM0_out	C18	Both to P22 of P11
P9_42B		0x9a0/1a0	114	GPIO3_18	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4		Mcasp1_aclkx	Mcaspo_axr2	eQEP0A_in	Mcasp0_aclkr	B12	Allocated mcasp0_pins
P9_43				GND										- See Pg.50 of the SRM
P9_44				GND										Ground
P9_45				GND										Ground
P9_46	cat		(Mode 7)	GND										Ground
P9	$PINS	ADDR +	GPIO NO.	Name	Mode 7						Mode 1	Mode 0	CPU	Notes

El siguiente dato que ofrece el archivo después del `offset` constituye la configuración del pin. Concretamente, los dos dígitos de menor peso son los que indican la configuración de acorde a la figura 2.9.

```
Bit Number      8 7 6 5 4 3 2 1 0
                  c r s p m m m     m = 3 mode bits [0-7]
                                    p = 0 pullups/pulldowns enabled, 1 pullups/pulldowns disabled
                                    s = 0 pulldown selected, 1 pullup selected
                                    r = 0 pin is output, 1 pin is input
                                    c = 0 fast slew control, 1 = slow slew control
```

Figura 2.9: Utilidad de cada bit de la configuración de los pines [5].

Con esta información, se puede comprobar que los pines `P8_24` y `P8_25` están en el modo 1 (`mmc1_dat1` y `mmc_dat0` respectivamente [24]), son entradas y tienen habilitado pull-up. Una vez se haya configurado, por ejemplo, el `P8_46` como pruout, se debería obtener una salida como la siguiente:

```
pin 41 (PIN41) 44e108a4 00000008 pinctrl-single
```

El siguiente paso consiste en el conocimiento de la localización de los pines en la BeagleBone Black. La figura 2.10 representa la ubicación de los pines en el dispositivo mientras que la 2.11 muestra los bits del registro 30 de las PRUs que están accesibles en los pines de la BeagleBone Black.

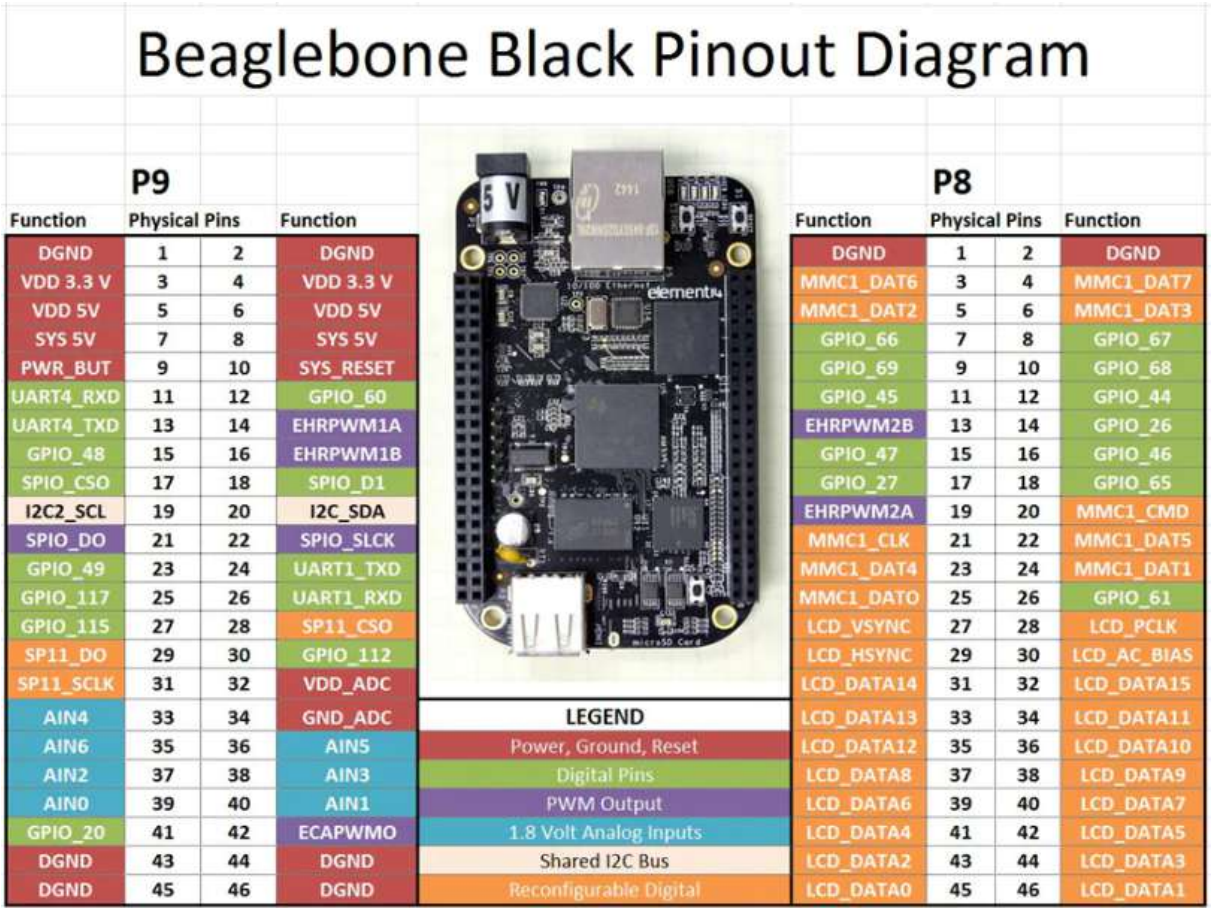


Figura 2.10: Pinout de la BeagleBone Black [6].

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
PRU0_7	25	26	PRU1_16 IN
PRU0_5	27	28	PRU0_3
PRU0_1	29	30	PRU0_2
PRU0_0	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
PRU0_6	41	42	PRU0_4
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
PRU0_15 OUT	11	12	PRU0_14 OUT
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	PRU1_13
PRU1_12	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
PRU1_8	27	28	PRU1_10
PRU1_9	29	30	PRU1_11
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
PRU1_6	39	40	PRU1_7
PRU1_4	41	42	PRU1_5
PRU1_2	43	44	PRU1_3
PRU1_0	45	46	PRU1_1

Figura 2.11: Bits del registro 30 de las PRUs accesibles en la BeagleBone Black [6].

Capítulo 3

Desarrollo

3.1 Introducción

En este capítulo se incluirá la descripción del desarrollo del trabajo.

El capítulo se estructura en 5 apartados: los primeros pasos en los que muestro las herramientas necesarias para el desarrollo del trabajo, las pruebas llevadas a cabo con ejemplos y la descripción de los programas creados para cumplir con el objetivo del trabajo (conector, programa para PRU0 y programa para la PRU1).

3.2 Primeros pasos

3.2.1 Acceso a la BeagleBone Black

Al conectar la BeagleBone Black, se deberá acceder al dispositivo USB y abrir el archivo `START.htm` [4] o `README.htm` (ambos archivos son iguales). En él se verá una lista de pasos (3.1) para iniciar a tratar con el dispositivo.

3.2.1.1 Actualizar imagen

En el primer paso indicado en los archivos, guía hacia la actualización del sistema operativo incluido en la BeagleBone Black para asegurarse de que su uso es el más simple posible aunque no es de carácter obligatorio. En este trabajo he utilizado la imagen a una Debian 10.3 para que incluya el módulo remoteproc que he instalado en una tarjeta microSD [19].

3.2.1.2 Conectividad

En el tercer paso indicado en los archivos, se brinda información acerca de la conectividad de la BeagleBone Black. Respecto a la conexión en el puerto Universal Serial Bus (USB) mencionada anteriormente, se dice que la BeagleBone Black funcionará como un servidor Dynamic Host configuration Protocol (DHCP) y asignará las direcciones 192.168.7.2 o 192.168.6.2 a sí misma y 192.168.7.1 o 192.168.6.1 a la máquina dependiendo del SO del computador. También es necesario instalar los drivers en la máquina para poder conectarnos correctamente al dispositivo.

Figura 3.1: Pasos de incio en la página `START.htm` [4]

3.2.1.3 Navegador

En el cuarto paso indicado en los archivos, indica cómo acceder al navegador web de la BeagleBone Black. Esta opción no supone gran interés dado que mostrará algunos ejemplos mediante BeagleScript que, por el momento, no funcionan correctamente [26], y dará acceso al editor Cloud9 que no será suficiente para el cumplimiento del objetivo de este trabajo.

3.2.2 Herramientas

Para el desarrollo de este trabajo he utilizado el compilador Compiler for PRUs (CLPRU) [11] y la herramienta `prudebug`. Para esta última he tenido que compilar el programa disponible en GitHub [17].

3.3 Programas de prueba

En esta sección explico los programas usados para probar el funcionamiento de las utilidades necesarias para el desarrollo del trabajo. En primer lugar sería necesario descargar el compilador clpru [11] e instalarlo en el dispositivo.

3.3.1 Compilación y carga de firmware

En este apartado veremos ejemplos de como compilar y cargar programas ya sea puramente en C o bien compuestos por C y ensamblador.

3.3.1.1 Hello world

En este apartado hablaré sobre el ejemplo que ofrece Glenn Klockwood [27] puramente en C para hacer parpadear un LED conectado al pin P9_31 que muestra el bit 0 del registro R30. Para ello usará el código en 3.1.

```
#define CYCLES_PER_SECOND 200000000 /* PRU has 200 MHz clock */

#define P9_31 (1 << 0) /* R30 at 0x1 = pru1_pru0_pru_r30_0 = ball A13 = P9_31 */

volatile register uint32_t __R30; /* output register for PRU */

void main(void) {
  while (1) {
    __R30 |= P9_31; /* set first bit in register 30 */
    __delay_cycles(CYCLES_PER_SECOND / 4); /* wait 0.5 seconds */
    __R30 &= ~P9_31; /* unset first bit in register 30 */
    __delay_cycles(CYCLES_PER_SECOND / 4); /* wait 0.5 seconds */
  }
}
```

Listado 3.1: Código de ejemplo de Glenn Klockwood.

Como se puede comprobar, en este código se alterna entre poner el bit 0 del registro R30 (llamado P9_31) a nivel alto y a nivel bajo dejando entre cada cambio un cuarto de segundo pese a que en los comentarios especifique medio segundo. Este ejemplo también incluye la tabla de recursos pero, dado que la explicaremos más en profundidad en el ejemplo 3.3.1.4. Este ejemplo llevado a la práctica no funciona debido a los parámetros (3.2) seleccionados a la hora de compilar.

Listado 3.2: Parámetros utilizados en la compilación.

```
CFLAGS = --include_path=$(PRU_SWPKG)/include \
--include_path=$(PRU_SWPKG)/include/am335x
LDFLAGS = $(PRU_SWPKG)/labs/lab_2/AM335x_PRU.cmd
```

Como se puede ver, solo incluye algunas librerías pero no usa el resto de opciones que usa TI en sus ejemplos.

3.3.1.2 GPIO toggle

En este ejemplo [28] proporcionado por TI veremos como, en contraste con el ejemplo anterior, se hace uso de más opciones durante la compilación y, por tanto, funciona correctamente.

```
volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
  volatile uint32_t gpio;

  /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
```

```
  CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

  /* Toggle GPO pins */
  /* Note: 0xFFFF_FFFF toggles all GPO pins */
  gpio = 0xFFFFFFFF;

  /* TODO: Create stop condition, else it will toggle indefinitely */
  while (1) {
    __R30 ^= gpio;
    __delay_cycles(100000000);
  }
}
```

Listado 3.3: Código para parpadear un LED.

En este caso vemos que el código que ofrece TI tiene como objetivo alternar los niveles de todos los bits del registro R30 cada medio segundo (100 millones de ciclos a 200 millones de ciclos por segundo). En comparación con el ejemplo anterior, este ejemplo utiliza las opciones de compilación y enlazado que se ven en 3.4.

Listado 3.4: Parámetros utilizados por el ejemplo de TI.

```
CFLAGS=-v3 -O2 --display_error_number --endian=little --hardware_mac=on
        --obj_directory=$(GEN_DIR) --pp_directory=$(GEN_DIR) -ppd -ppa
LFLAGS=--reread_libs --warn_sections --stack_size=$(STACK_SIZE)
        --heap_size=$(HEAP_SIZE)
```

En este caso, el programa funciona correctamente por lo que ya tendríamos la opción de continuar utilizando el lenguaje C pero, dado que busco precisión a la hora de generar las señales, requiero usar lenguaje ensamblador.

3.3.1.3 ASM blinky

El próximo paso es buscar poder utilizar un programa en ensamblador. Este ejemplo deberá tener 2 archivos: uno en C en el cual se llame al archivo en ensamblador y otro en ensamblador que contenga la referencia utilizada en el programa en C. El código de este ejemplo [29] se puede ver en 3.5 y en 3.6.

```
extern void start(void);

void main(void)
{
  start();
}
```

Listado 3.5: Código en C del ejemplo de BeagleScope.

```
    .cdecls "main_pru1.c"

  DELAY .macro time
  LDI32 R0, time
  QBEQ  $E?, R0, 0
  $M?:  SUB R0, R0, 1
  QBNE  $M?, R0, 0
  $E?:
  .endm
```

```
.clink
.global start
start:
LDI32    R30, 0xFFFFFFFF
DELAY    100000000
LDI32    R30, 0x00000000
DELAY    100000000
JMP start

HALT
```

Listado 3.6: Código en ensamblador del ejemplo de BeagleScope.

En el primer extracto de código podemos ver que solamente hace la declaración de la función `start` de forma que no devuelve ningún valor y tampoco lo necesita para ser llamada. En el segundo extracto, podemos ver que dentro de la función `start` entra en un bucle infinito en el que cada medio segundo carga o bien un valor poniendo todos los bits de R30 a 1 o un valor poniendo todos los bits de R30 a 0. Para este ejemplo se usan las mismas opciones de compilación y enlazado que en el ejemplo anterior.

3.3.1.4 RPMsg

El último ejemplo necesario para poder tener la base necesaria para poder diseñar el programa final consiste en un ejemplo que haga uso de RPMsg. Este ejemplo concretamente viene del proyecto BeagleScope [30] y es una adaptación del ejemplo que provee TI [31].

```
void main(void)
{
  struct pru_rpmsg_transport transport;
  uint16_t src, dst, len;
  volatile uint8_t *status;

  /* allow OCP master port access by the PRU so the PRU can read external memories */
  CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

  /* clear the status of the PRU-ICSS system event that the ARM will use to 'kick' us */
  CT_INTC.SICR_bit.STS_CLR_IDX = SE_ARM_TO_PRU0;

  /* Make sure the Linux drivers are ready for RPMsg communication */
  status = &resourceTable.rpmsg_vdev.status;
  while (!(*status & VIRTIO_CONFIG_S_DRIVER_OK));

  /* Initialize pru_virtqueue corresponding to vring0 (PRU to ARM Host direction) */
  pru_virtqueue_init(&transport.virtqueue0, &resourceTable.rpmsg_vring0, SE_PRU0_TO_ARM,
    SE_ARM_TO_PRU0);

  /* Initialize pru_virtqueue corresponding to vring1 (ARM Host to PRU direction) */
  pru_virtqueue_init(&transport.virtqueue1, &resourceTable.rpmsg_vring1, SE_PRU0_TO_ARM,
    SE_ARM_TO_PRU0);

  /* Create the RPMsg channel between the PRU and ARM user space using the transport
     structure. */
  while (pru_rpmsg_channel(RPMSG_NS_CREATE, &transport, RPMSG_CHAN_NAME,
    PRU0_RPMSG_CHAN_DESC, PRU0_RPMSG_CHAN_PORT) != PRU_RPMSG_SUCCESS);
  while (1) {
    /* Check bit 30 of register R31 to see if the ARM has kicked us */
    if (check_host_int(HOST_ARM_TO_PRU0)) {
      /* Clear the event status */
```

```
      CT_INTC.SICR_bit.STS_CLR_IDX = SE_ARM_TO_PRU0;
      /* Receive all available messages, multiple messages can be sent per kick */
      while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) ==
    PRU_RPMSG_SUCCESS) {
        /* Echo the message back to the same address from which we just received */
        while(1){
          if (check_host_int(HOST_PRU1_TO_PRU0)){
            CT_INTC.SICR_bit.STS_CLR_IDX = SE_PRU1_TO_PRU0;
            pru_rpmsg_send(&transport, dst, src, "Interrupted\n",sizeof("Interupted\n"));
          }
        }
      }
    }
  }
}
```

Listado 3.7: Código de la PRU0 para RPMsg.

Este primer extracto de código 3.7 corresponde a la PRU0. Para comenzar, se debe dar la posibilidad de acceder al puerto Open Core Protocol (OCP) maestro para poder leer la memoria externa. A continuación, se inicializa el estado del evento para comunicar el ARM con la PRU0. Después, se inicializan las colas para los mensajes y se crea el canal RPMsg. Dentro del bucle, comprueba si ha recibido algún evento por parte del ARM y, si es así, recibe el mensaje y lo reenvía cada vez que la PRU1 provoca un evento.

```
void main(void)
{
  while (1) {
    generate_sys_eve(SE_PRU1_TO_PRU0);
    __delay_cycles(100000000);
  }
}
```

Listado 3.8: Código de la PRU1 para RPMsg.

Este segundo extracto 3.8 corresponde al código de la PRU1 y consiste en provocar un evento cada medio segundo.

```
#define SE_PRU0_TO_ARM      16
#define SE_ARM_TO_PRU0      17
#define SE_PRU1_TO_PRU0     18

#define HOST1_INT     ((uint32_t) 1<<31)
#define HOST0_INT     ((uint32_t) 1<<30)

#define R31_VECTOR_VALID_STROBE_BIT 5

#define check_host_int(host)\
(__R31 & host)

#define generate_sys_eve(sys_eve)\
__R31 = ( (1 << R31_VECTOR_VALID_STROBE_BIT) | (SE_PRU1_TO_PRU0-16))
```

Listado 3.9: Código en archivo pru_defs.

Este tercer extracto 3.9 corresponde a un archivo .h en el que se especifican algunas definiciones como por ejemplo qué evento se utiliza en dirección PRU0 a ARM. También se encuentran las definiciones de `check_host_int` y `generate_sys_eve`.

```
struct ch_map pru_intc_map[] = { {16, 2},
```

```
  {17, 0},
  {18, 1},

};

...

  /* Channel-to-host mapping, 255 for unused */
  0, 1, 2, HOST_UNUSED, HOST_UNUSED,
  HOST_UNUSED, HOST_UNUSED, HOST_UNUSED, HOST_UNUSED, HOST_UNUSED,
```

Listado 3.10: Tabla de recursos en el ejemplo BeagleScope.

Este último extracto 3.10 corresponde a la tabla de recursos de la PRU0 y, en él, se reservan los canales 0, 1 y 2 para los eventos 17, 18 y 16 respectivamente. En la segunda parte del extracto, se especifica el host para cada canal utilizado, en este caso cada canal conducirá a su host análogo.

3.4 Conector

En esta sección explico el programa utilizado para implementar una interfaz de usuario sencilla para poder comunicarse con la PRU0. En primer lugar, muestro las distintas señales disponibles 3.11.

```
do{
 do{
   printf("\n\nSeleccione una opcion:\n"
   "\t1) Diente de sierra\n"
   "\t2) Senoidal\n"
   "\t3) Triangular\n"
   "\t4) PWM\n"
   "\t5) Valor fijo\n"
   "\t6) Pulso de radiacion\n"
   "\t7) Interrumpir senal\n"
   "\t8) Apagar PRUs y terminar el programa\n\n");
   scanf("%d", &opcion);
   clear();
 }while(opcion<1 || opcion>8);
```

Listado 3.11: Menú con las señales posibles.

A continuación, pediremos el parámetro según la opción elegida. En el caso del seno, señal triangular y diente de sierra podremos elegir la velocidad. En el caso de señal PWM, podremos elegir el porcentaje de tiempo en alto. En el caso de valor fijo podremos escoger el valor a mostrar. En 3.12 muestro las opciones disponibles del parámetro en la opción diente de sierra. El significado de que la variable código valga "D"se explica en el apartado 3.5.

```
   switch(opcion){
   case 1:
     strcpy(codigo, "D");
     printf("\n\nIntroduzca una velocidad (1 (mas rapida), 2 o 3 (mas lenta)):\n\n");
     scanf("%c", &codigo[1]);
     clear();
   break;
```

Listado 3.12: Opciones de parámetro.

El siguiente paso es abrir el archivo de RPMsg y enviar el código 3.13. También es necesario cerrarlo para poder abrirlo a continuación en modo lectura.

```
    file = fopen("/dev/rpmsg_pru30","w");
if(file == NULL)
{
  printf("Error! Archivo rpmsg_pru30 no encontrado.\n");
  exit(1);
}
fprintf(file,"%s", codigo);
fclose(file);
```

Listado 3.13: Apertura y cierre del archivo para escribir el código.

Por último, abriré el archivo para recibir el mensaje que envíe la PRU y lo cerraré.

```
if ((file = fopen("/dev/rpmsg_pru30","r")) == NULL){
  printf("Error! Archivo rpmsg_pru30 no encontrado.\n");

  exit(1);
}

fscanf(file,"%s", &respuesta);

printf("\n%s\n\n\n", respuesta);
fclose(file);
```

Listado 3.14: Apertura y cierre del archivo para recibir respuesta.

3.5 Programa PRU0

En este apartado explicaremos el proceso para llegar al programa utilizado por la PRU0 para guardar los datos en la memoria e indicarle a la PRU1 el tamaño de la señal a mostrar. En primer lugar explico el programa en lenguaje C y, posteriormente, las funciones en ensamblador.

3.5.1 C

En este programa sigo el ejemplo proporcionado por TI hasta el primer bucle. De esta forma, dispongo del canal RPMsg bajo el nombre `rpmsg-pru30`. Dentro del bucle, compruebo si el ARM se comunica con la PRU y, si se detecta evento, analizo el mensaje que ha enviado. El primer carácter (símbolo) del mensaje indicará la opción y, dependiendo de la opción, se requerirá un segundo carácter que especifique el parámetro de la opción.

Opción	Símbolo	Parámetro
Valor fijo	F	Cualquier valor entre 0 y 4095 incluidos.
Señal senoidal	S	1, 2 o 3.
Diente de sierra	D	1, 2 o 3.
Señal triangular	T	1, 2 o 3.
Señal PWM	W	1, 2 o 3.
Pulso de radiación	R	-
Interrumpir señal	I	-
Apagar PRUs	H	-

Tabla 3.1: Señales muestreables con su carácter indicativo y posibles parámetros.

En la tabla 3.1 muestro las opciones con sus posibles parámetros. En el caso del valor fijo, el parámetro que recibe es el valor que se mostrará por los pines y en el caso de la señal PWM el parámetro indica

el porcentaje de ciclo a nivel alto (1=25 %, 2=50 % y 3=75 %) mientras que en las opciones de señal senoidal, diente de sierra y señal triangular el parámetro que recibe representa la velocidad de la señal. Como la PRU1 estará leyendo constantemente las muestras en la memoria, para cambiar de una señal a otra primero interrumpo la señal. En el siguiente código muestro como llamo a la función `start0_I` (utilizada para interrumpir la señal) y, a continuación, llamo a la función `start0_F` pasándole como parámetro el valor a mostrar.

```
while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) == PRU_RPMSG_SUCCESS) {
  if(payload[0]=='F'){
    start0_I();                //interrumpir antes de cambiar la senal
    generate_sys_eve(SE_PRU0_TO_PRU1); //interrumpir antes de cambiar la senal
    pru_rpmsg_send(&transport, dst, src, "Mostrando_valor_fijo\n", sizeof("
   Mostrando_valor_fijo\n"));
    param = atoi(&payload[1]);
    start0_F(param);
    generate_sys_eve(SE_PRU0_TO_PRU1);
```

Listado 3.15: Código por opción en la PRU0.

Una vez he llamado a la función para introducir los parámetros en la memoria, causo un evento a la PRU1 que la tratará de acuerdo a lo que se explicará en la sección 3.6. En las señales que no quepan en la memoria se activará el evento dentro de la rutina en ensamblador, no en el caso en C como muestra 3.15. En la parte de los eventos he reservado los 16, 17, 18 y 19 como muestro en 3.16. Antes de empezar a introducir los datos en la memoria interrumpo la señal para evitar errores. Es importante mencionar que, en el caso de que se envíe el código para apagar las PRUs, se incluirá la instrucción ___halt();. Ni en la opción de interrumpir ni en la opción de apagar se enviará anteriormente el código para interrumpir ya que estas señales no pueden llegar a causar errores.

```
struct ch_map pru_intc_map[] = { {16, 2},
  {17, 0},
  {18, 1},
  {19, 0}

};
```

Listado 3.16: Reserva de eventos en la tabla de recursos.

El evento 16 como evento desde PRU0 al ARM, el evento 17 en dirección ARM a PRU, el evento 18 en dirección PRU0 a PRU1 y el evento 19 en dirección PRU1 a PRU1. Cada canal está conectado a su host análogo.

3.5.2 ASM

En este apartado muestro las funciones en lenguaje ensamblador que utilizo para crear y guardar los valores de las distintas señales. En primer lugar, explico del código que uso para indicar a la PRU1 el tamaño de la señal. Utilizo los primeros 4 bytes en la memoria SRAM para indicar el tamaño de la señal o, en su defecto, un código que explico en 3.2. Los primeros 4 bytes se leerán como se indica en la figura 3.2.

El código especial solo se tendrá en cuenta si el tamaño es 0 (tamaño utilizado para indicar uso del código especial). En el resto de los casos el código especial valdrá 0. Es importante mencionar que el tamaño de la señal debe indicarse en bytes. Cada muestra a mostrar por los pines ocupará 2 bytes por lo que el tamaño de la señal indicado será el doble del número de muestras.

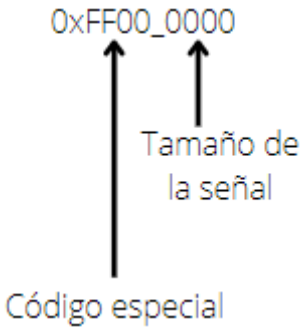


Figura 3.2: Lectura de los primeros 4 bytes de la SRAM.

Señal	Código especial
Señal PWM	0x0001
Valor fijo	0x0002
Pulso de radiación	0xFFFF
Interrumpir señal	0xFF00
Apagar PRUs	0x00FF

Tabla 3.2: Códigos especiales.

3.5.2.1 Valor fijo

En esta opción guardaremos el código especial en los primeros 4 bytes y guardará el valor fijo en el registro 29 del scratch-pad 10 3.17.

```
start0_F:
  LDI32 R10, 0x00010000
  LDI32   R13, 0x00020000
  SBBO  &R13, R10, 0, 4
  MOV   R29, R14
  XOUT  0x0b, &R29.b0, 0x04
  JMP   R3.w2
```

Listado 3.17: Código para valor fijo.

3.5.2.2 Señal senoidal

En la señal senoidal he establecido que la velocidad 1 represente un ciclo completo de la señal en 100 muestras, la velocidad 2 en 200 muestras y la velocidad 3 en 300 muestras. En 3.18 muestro el código utilizado para obtener las muestras de las 3 velocidades distintas de seno.

```
#include <stdio.h>
#include <math.h>

#define pi 3.1415926535

void main(){
  for(int i=0; i<NUM_MUESTRAS; i++){
    printf("0x%x\n", (int)(4095*(0.5+(0.5*sin(2*pi*i/NUM_MUESTRAS)))));
  }
}
```

Listado 3.18: Programa para obtener las muestras del seno.

A continuación, introduciremos los valores en los registros con la instrucción `LDI32` y los guardaremos en la memoria con la instrucción `SBBO` apuntando a la dirección 0x0001_0000.

```
sen1:
  LDI32 R10, 0x00010000
  LDI32 R11, 0x00010130 ;244+60 (ultimo offset mas ultima cantidad de bytes escrita)
  LDI32   R9, 200
  SBBO  &R9, R10, 0, 4
  LDI R15.w0, 0x7ff
  LDI R15.w2, 0x880
  LDI R16.w0, 0x900
  LDI R16.w2, 0x97f
  ...
  LDI R29.w0, 0xfda
  LDI R29.w2, 0xfbe
  SBBO  &R15, R10, 4, 60
```

Listado 3.19: Programa para almacenar las muestras del seno.

3.5.2.3 Diente de sierra

Para la generación del diente de sierra llevaremos a código el diagrama de flujo 3.3. El código resultante corresponde a 3.20.

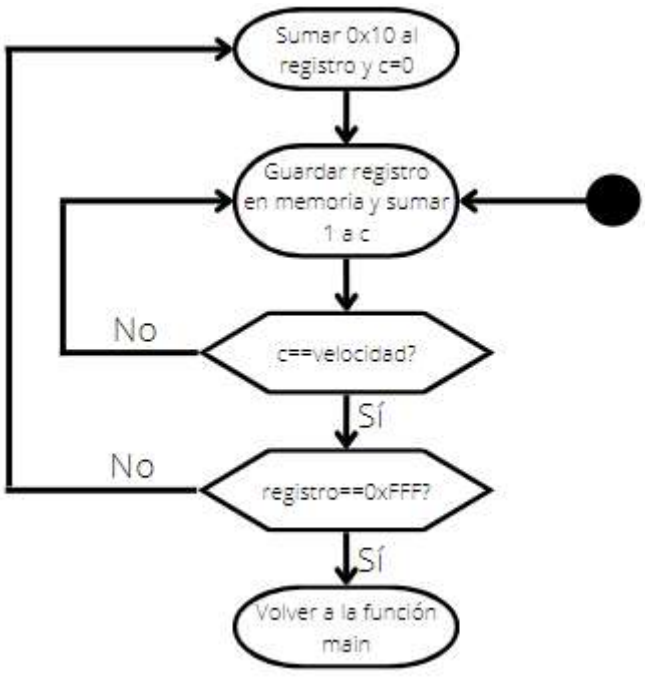


Figura 3.3: Diagrama de flujo para el diente de sierra.

```
start0_D:
  LDI32 R10, 0x00010000
  LDI R6, 0x0
  LDI32 R7, 0x280
  LDI32 R8, 0x00020000
  LDI32 R24, 17
  LDI32 R25, 19
  LDI R18, 0xFFF
  LDI R9, 0x2FFE
  QBNE aux_D1, R14, 1
  LDI32 R9, 8192
aux_D1: SBBO &R9, R10, 0, 4
  LDI32 R17, 4
bucle_D: LDI32 R15, 0xFFFFFFFF
aux_D2: LDI32 R16, 0
  ADD R15, R15, 0x1
aux_D3: SBBO &R15.w0, R10, R17, 2
  QBEQ comienzo_D, R17, R9
cont_D: ADD R16, R16, 1
  ADD R17, R17, 2
  QBLT aux_D3, R14, R16
  QBLT aux_D2, R18, R15
  QBNE bucle_D, R14, 1
  LDI R31, 0x22
  JMP R3.w2

comienzo_D:
  QBNE llamar_D, R6, 0
  LDI32 R31, 0x22
  LDI R6, 0x1
  SBBO &R24, R8, 0x24, 4
llamar_D:
  WBS R31, 30
  LBBO &R26, R8, R7, 4
  QBBS volver, R26, 17
  SBBO &R25, R8, 0x24, 4
  LDI R17, 0x2
  JMP cont_D

volver:
  JMP R3.w2
```

Listado 3.20: Código generador de las muestras de diente de sierra.

Es importante mencionar que en la velocidad 1 las muestras sí caben en la memoria por lo que en cuánto se guardan todas las muestras en la memoria se notifica a la PRU0 y se vuelve a la función main(). Esta función aumenta el valor guardado en la memoria y lo guarda 1 vez para la velocidad 1, 2 veces para la velocidad 2 y 3 veces para la velocidad 3. En la subrutina comienzo_D se activa en dirección a la PRU0 y en la subrutina llamar_D se espera la notificación de la PRU1 para guardar las siguientes muestras desde el inicio de la memoria. Un aspecto esencial consiste en asegurarse que la PRU1 tardará más tiempo en leer las muestras que la PRU0 en guardarlas. En esta rutina cuando es necesario aumentar el valor guardado necesitamos 9 ciclos y, cuando no, 6 ciclos y, en el peor de los casos, se alterna uno de cada (velocidad 2) por lo que, de media, cada muestra se guarda cada 7,5 ciclos cumpliendo así la condición.

3.5.2.4 Señal triangular

Para la generación de la señal triangular utilizaremos el diagrama de flujo 3.3 para la tendencia ascendente y, cambiando el sumar 0x1 y comparar con 0xFFF por restar 0x1 y comparar con 0 respectivamente para la tendencia descendente. El código resultante corresponde a 3.21.

```
start0_T:
  LDI32 R10, 0x00010000
  LDI R6, 0x0
  LDI32 R7, 0x280
  LDI32 R8, 0x00020000
  LDI R9, 0x2FFE
  LDI32 R18, 0x0FFF
  LDI32 R24, 17    ;para comprobar
  LDI32 R25, 19
  SBBO &R9, R10, 0, 4
  LDI32   R17, 4
bucle_T:  LDI32   R15, 0xFFFFFFFF
aux_T1: LDI32   R16, 0
  ADD R15, R15, 0x1
aux_T2: SBBO &R15.w0, R10, R17, 2
  QBEQ  comienzo_T1, R17, R9
cont_T1:  ADD R16, R16, 1
  ADD R17, R17, 2
  QBLT  aux_T2, R14, R16
  QBNE  aux_T1, R15, R18
aux_T3: LDI32   R16, 0
  SUB R15, R15, 0x1
aux_T4: SBBO &R15.w0, R10, R17, 2
  QBEQ  comienzo_T2, R17, R9
cont_T2:  ADD R16, R16, 1
  ADD R17, R17, 2
  QBLT  aux_T4, R14, R16
  QBNE  aux_T3, R15, 0x1
  JMP bucle_T

comienzo_T1:
  QBNE  llamar_T1, R6, 0
  LDI R31, 0x22
  LDI R6, 0x1
  SBBO &R24, R8, 0x24, 4
llamar_T1:
  WBS R31, 30
  LBBO &R26, R8, R7, 4
  QBBS  volver, R26, 17
  SBBO &R25, R8, 0x24, 4
  LDI R17, 0x2
  JMP cont_T1

comienzo_T2:
  QBNE  llamar_T2, R6, 0
  LDI R31, 0x22
  LDI R6, 0x1
  SBBO &R24, R8, 0x24, 4
llamar_T2:
  WBS R31, 30
  LBBO &R26, R8, R7, 4
  QBBS  volver, R26, 17
  SBBO &R25, R8, 0x24, 4
```

```
LDI R17, 0x2
JMP cont_T2
```

Listado 3.21: Código generador de las muestras de la señal triangular.

Dado que la generación de esta señal se compone del mismo bucle que el diente de sierra, se sigue cumpliendo que la PRU0 guarda las muestras más rápido de lo que la PRU1 las lee.

3.5.2.5 Señal PWM

Para la señal PWM uso un código especial ya que, a diferencia de las señales normales, no uso la memoria SRAM sino los distintos scratch-pads con intención de mostrar otra forma de comunicarse entre las PRUs aunque más limitada en tamaño que la memoria SRAM. Guardaremos los valores en los registros `R26`, `R27`, `R28` y `R29` y los introduciremos en los scratch-pads. Si el parámetro vale 1, solo R26 estará a nivel alto. Si el parámetro vale 2, R26 y R27 estarán a nivel alto. Si el parámetro vale 3, R26, R27 y R28 estarán a nivel alto. El código para la señal PWM de 25 % corresponde con 3.22.

```
pwm_1:
  LDI R26, 0xFFF
  LDI R27, 0x000
  LDI R28, 0x000
  LDI R29, 0x000
  XOUT  0x0a, &R26.b0, 16
  XOUT  0x0b, &R26.b0, 16
  XOUT  0x0c, &R26.b0, 16
  JMP R3.w2
```

Listado 3.22: Código para señal PWM de velocidad 1.

3.5.2.6 Pulso de radiación

Para esta señal me he fijado en la representación de la señal temporal generada por un amplificador en un datector de radiación [32] y lo he modelado para obtener la señal correspondiente a la figura 3.4.

Como podemos comprobar, el gráfico ya contiene la señal temporal normalizada y con una duración de 8 μs. También es importante mencionar que la señal tendrá un valor mínimo de 0x3F que se mantendrá hasta el siguiente pulso que, según veremos en el apartado 3.6.1, tardará 6 ms.

3.6 Programa PRU1

En este apartado explicaremos el proceso para llegar al programa utilizado por la PRU1 para sacar los datos por los pines. Respecto al programa en C solo comprobará si ha sido interrumpido y, si ha sido así, cambiará el estado del evento y llamará a la función `start1`.

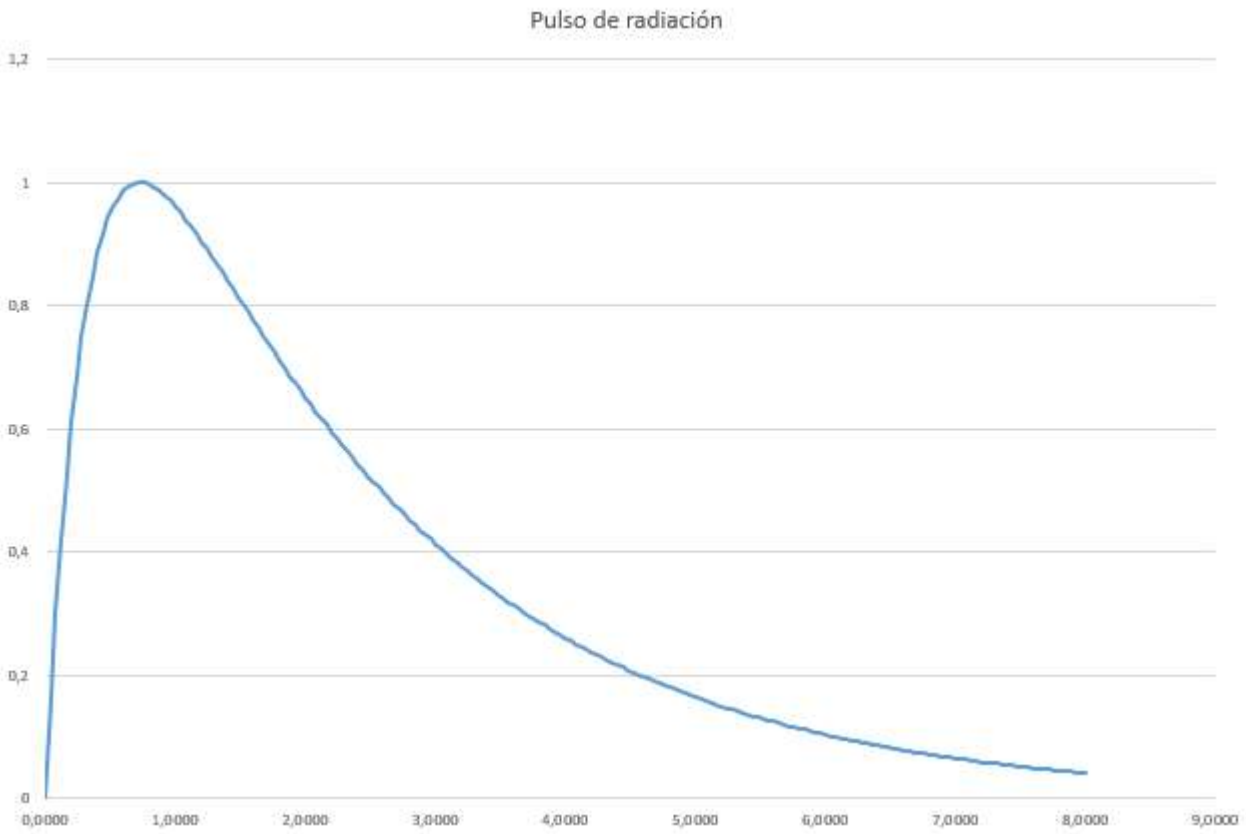


Figura 3.4: Posible señal temporal generada por un amplificador en un detector de radiación normalizada.

```
void main(void)
{
  while (1) {
    // Checks if PRU0 has sent a signal
    if (check_host_int(HOST_PRU0_TO_PRU1)){
      // Clears the signal state and calls the ASM function
      CT_INTC.SICR_bit.STS_CLR_IDX = SE_PRU0_TO_PRU1;
      start1();
    }
  }
}
```

Listado 3.23: Programa en C de la PRU1.

En primer lugar, es necesario poder detectar cuando es necesario leer el código especial y cuando nos encontramos ante una señal cuyas muestras no caben en la memoria 3.24. En ambos casos saltará a la subrutina correspondiente. Si no se da ninguno de estos casos, se llamará a la subrutina `sram`. El uso de los registros se explica en la tabla 3.3.

Registro	Uso
R8	Posición de memoria en la que activar el evento a la PRU0. Número de instrucciones en 6 ms (señal de radiación).
R9	Última posición a leer de la memoria si la señal no cabe en ella. Contador de instrucciones realizadas (señal de radiación).
R10	Inicio de la memoria SRAM.
R11	Última posición a leer si la señal cabe entera en la memoria SRAM.
R12	Contador de bytes leídos (también usado como offset a la hora de leer).
R29	Muestra en el caso de un valor fijo.
R26, R27, R28 y R29	Muestras en la señal PWM.

Tabla 3.3: Uso de los registros en la PRU1.

```
start1:
  LDI32 R10, 0x10000
  LDI32 R9, 0x2FFE
  LDI32 R8, 0x2000
  LBBO  &R11, R10, 0, 4
  QBEQ  scratch_pad, R11.w0, 0
  LDI R12, 4
  QBEQ  entero, R11, R9
  ADD R11, R11, 4       ;hay que tener en cuenta los 4 primeros bytes para el tamano de
    la senal
  QBEQ  rad_1, R11.w0, 3  ;0xFFFF + 0x4 = 0x3
  JMP sram
```

Listado 3.24: Inicio de la función de la PRU1.

En segundo lugar, es necesario establecer las tareas que debe llevar a cabo y cuántos ciclos de reloj necesita cada una. Estas son:

- Leer muestra (2 bytes) de la memoria SRAM (3 ciclos).
- Cambiar la señal DRDY (1 ciclo).
- Comprobar señal de la PRU0 (1 ciclo).
- Sumar los bytes leídos al offset para leer de la memoria (1 ciclo).
- Comprobar si se han leído todos los bytes de la señal (1 ciclo).

Aunque en la guía [2] vienen especificados la latencia, he hecho uso del registro `CYCLE` [33] para estar seguros de los ciclos necesarios para cada instrucción. También lo he utilizado para verificar que el número de ciclos esperado corresponde con el número de ciclos obtenido. Este registro se encuentra en la dirección de memoria 0x0002_2000 para el contador de la PRU0 y la dirección 0x0002_4000 para la PRU1. Utilizando este recurso de la forma que muestro en 3.25 podemos conocer la latencia en cualquier código en ensamblador.

```
LDI32 R10, 0x00024000

LBBO  &R11, R10, 12, 4
LBBO  &R12, R10, 12, 4

Codigo a medir
LBBO  &R13, R10, 12, 4
```

Listado 3.25: Forma de conocer los ciclos necesarios para ejecutar determinado código.

Las dos primeras veces que se consulta el registro se hace para comprobar los ciclos necesarios para leerlo ya que habrá que tenerlo en cuenta al contar los ciclos de nuestro programa a medir.

Teniendo todo esto en cuenta, he utilizado el programa 3.26 que provoca un flanco ascendente en la señal DRDY cada 8 ciclos por lo que tendremos una muestra cada 40 ns.

```
sram:
  LBBO  &R30.w0, R10, R12, 2
  NOP
ci5:  ADD R12, R12, 2
  QBBS  volver, R31, 31
  SET R30, R30, 12
  QBEQ  cond8, R11, R12
  LBBO  &R30.w0, R10, R12, 2
  NOP
  ADD R12, R12, 2
  QBEQ  cond14, R11, R12
ci15: SET R30, R30, 12
  JMP sram

cond8:
  LBBO  &R30.w0, R10, 4, 2
  LDI R12, 6
  NOP
  NOP
  JMP ci15

cond14:
  SET R30, R30, 12
  LDI R12, 4
  LBBO  &R30.w0, R10, R12, 2
  JMP ci5
```

Listado 3.26: Código utilizado para las señales senoidales, triangulares y dientes de sierra.

Los registros utilizados `R10` para almacenar la dirección de inicio de la memoria SRAM (0x0001_0000), `R11` para almacenar el número de bytes para leer más los 4 bytes iniciales usados para el tamaño de la señal y `R12` como contador de los bytes leídos (inicialmente vale 4). Se suma 4 a `R11` y `R12` empieza en 4 para poder utilizarlo de offset en la instrucción `SBBO`. A las etiquetas `cond8` y `cond14` se las llama en caso de que se terminen de leer todas las muestras, lo cuál se conoce comparando el contador con el número de muestras a leer. Al darse está condición, se reiniciará el valor de `R12` siguiendo con el orden de lectura de muestra y de cambio de la señal DRDY tal y como se hace antes de llegar al final de la memoria.

En las señales que ocupen más de lo que la memoria es capaz de albergar, usaremos la rutina 3.26 con una modificación que permite activar un evento al llegar a una posición de memoria determinada (he utilizado el offset 0x2000 respecto al inicio de la memoria SRAM). Hago uso de las instrucciones NOP

para comparar el contador con el offset en el cuál activar el evento mediante la instrucción `LDI R31, 0x23`.

```
entero:
  LBBO  &R30.w0, R10, R12, 2
  QBEQ  ent4, R8, R12
ent5: ADD R12, R12, 2
  QBBS  volver, R31, 31
  SET R30, R30, 12
  QBEQ  ent8, R11, R12
ent9: LBBO  &R30.w0, R10, R12, 2
  QBEQ  ent12, R8, R12
  ADD R12, R12, 2
  QBEQ  ent14, R11, R12
ent15:  SET R30, R30, 12
  JMP entero

ent8:
  LBBO  &R30.w0, R10, 4, 2
  LDI R12, 6
  NOP
  NOP
  JMP ent15

ent14:
  SET R30, R30, 12
  LDI R12, 4
  LBBO  &R30.w0, R10, R12, 2
  JMP ent5

ent4:
  ADD R12, R12, 2
  LDI R31, 0x23
  SET R30, R30, 12
  JMP ent9

ent12:
  ADD R12, R12, 2
  LDI R31, 0x23
  SET R30, R30, 12
  JMP entero
```

Listado 3.27: Código utilizado para las señales senoidales, triangulares y dientes de sierra.

3.6.1 Pulso de radiación

El pulso de radiación seguirá el código explicado en 3.26. La única diferencia consiste en que es necesario dejar 6 ms entre pulso y pulso, por lo que, cuando se lea la última muestra de la memoria, mantengo la última muestra leída hasta transcurridos 6 ms, momento en el que se volverá a leer desde el comienzo de la memoria.

```
pausa1:
  LDI32 R9, 0
  CLR R30, R30, 12
  LDI R12, 4
  LDI32 R8, 0x124F80  ;6 ms hasta el siguiente pulso
  SET R30, R30, 12
  NOP
```

```
  NOP
  JMP aux_R

pausa2:
  SET R30, R30, 12
  LDI R12, 4
  CLR R30, R30, 12
  LDI32 R8, 0x124F80  ;6 ms hasta el siguiente pulso
  NOP
  SET R30, R30, 12
  LDI32 R9, 0
  NOP
aux_R:  CLR R30, R30, 12
  QBBS  volver, R31, 31
  ADD R9, R9, 8
  NOP
  SET R30, R30, 12
  QBGT  rad_2, R8, R9
  NOP
  JMP aux_R
```

Listado 3.28: Código utilizado para el pulso de radiación.

En 3.28, las etiquetas `pausa1` y `pausa2` sustituyen a las etiquetas `cond8` y `cond14`.

3.6.2 Señal PWM

Para la señal PWM leo los distintos scratch-pads guardando así los distintos valores en los registros `R26`, `R27`, `R28` y `R29`.

```
  XIN 0x0b, &R26, 0x10
  MOV R30, R26
  NOP
  SET R30.t12
  NOP
  MOV R30, R27
  NOP
  SET R30.t12
  NOP
  MOV R30, R28
  NOP
  SET R30.t12
  NOP
  MOV R30, R29
  NOP
  SET R30.t12
  XIN 0x0c, &R26, 0x10
```

Listado 3.29: Código utilizado para la señal PWM.

En el extracto 3.29 muestro como se lee el scratch-pad 10 y cargan los valores en los pines de salida. Con este método busco una frecuencia de la señal DRDY de 50 MHz a diferencia de los 25 MHz que espero con el código de la sección 3.26.

3.6.3 Valor fijo

Para esta señal también he utilizado el scratch-pad 10 aunque se podría utilizar la SRAM con el código 3.26.

```
valor_fijo:
  XIN 0x0b, &R29.b0, 0x04
  MOV R30, R29
  SET R30, R30, 12
  NOP
aux_F:  CLR R30, R30, 12
  QBBS  volver, R31, 31
  SET R30, R30, 12
  JMP aux_F
```

Listado 3.30: Código utilizado para un valor fijo.

En este caso también busco una frecuencia de 50 MHz de la señal DRDY por lo que cargo el valor enviado y modifico el valor de la señal DRDY cada 2 ciclos.

3.6.4 Parar

En caso de recibir el código que implique interrumpir la señal, la PRU1 dejará de cambiar el valor mostrado por los pines, tanto la muestra como la señal DRDY. Para esto hago que simplemente vuelva a la función main 3.31.

```
parar:
  JMP r3.w2
```

Listado 3.31: Código para interrumpir la señal.

3.6.5 Apagar

En caso de que reciba el código correspondiente a apagar la PRU se llamará a la instrucción HALT 3.32.

```
apagar:
  HALT
```

Listado 3.32: Código para interrumpir la señal.

Capítulo 4

Resultados

4.1 Introducción

En este capítulo muestro los resultados obtenidos probando los códigos expuestos en la sección 3. Es necesario conocer que la herramienta utilizada para la medición cambia la frecuencia de las muestras guardadas en el archivo .csv según la ventana de tiempo capturada. Por ejemplo, en las señales senoidales las guarda cada 4 ns como se muestra en 4.1. En cambio, en señales más grandes como el diente de sierra, la señal triangular o el pulso de radiación, se almacenan las muestras de cada 80 ns como se comprueba en 4.2.

Listado 4.1: Muestras obtenidas con el analozador lógico en señales senoidales.

```
1.19688e−08,0,0,1,1,1,0,1,0,0,0,0,0,0
1.59688e−08,0,0,1,1,1,0,1,0,0,0,0,0,0
1.99688e−08,1,0,1,1,1,0,1,0,0,0,0,0,0
2.39688e−08,1,0,1,1,1,0,1,0,0,0,0,0,0
```

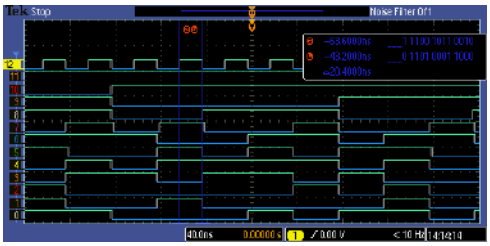
Listado 4.2: Muestras obtenidas con el analozador lógico en señales más grandes.

```
−3.45680e−03,1,0,0,1,0,1,1,0,0,0,1,1,1
−3.45672e−03,0,0,0,1,0,1,1,0,0,0,1,0,0
−3.45664e−03,1,0,0,1,0,1,1,0,0,1,0,1,1
−3.45656e−03,0,0,0,1,0,1,1,0,0,1,0,0,0
```

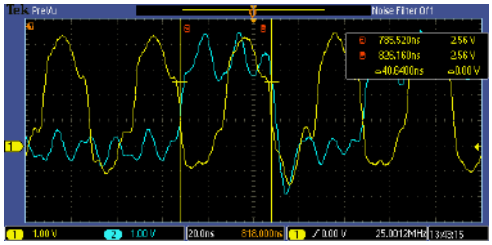
Para comprobar las señales que disponen de varias velocidad, he comprobado las 3 velocidades que he habilitado prestando especial atención en que la señal DRDY se muestre correctamente. También, muestro toda la señal capturada seguida de la misma señal pero solo aquellas muestras en las que la señal DRDY tenga como valor 1 ya que, cuando la señal DRDY cambia a nivel bajo, recojo la muestra de la memoria y es posible que algunos valores no se hayan capturado correctamente debido a la fluctuación previa a la estabilización de la señal.

4.2 Señal DRDY

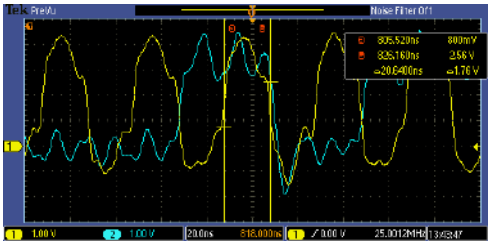
En primer lugar, voy a comprobar que el tiempo de la señal DRDY se corresponde con los esperados. En las ocasiones en las que utilice la memoria SRAM esta señal tendrá una frecuencia de 25 MHz estando en nivel alto 20 ns y a nivel bajo otros 20 ns.



(a) DRDY en señal senoidal.

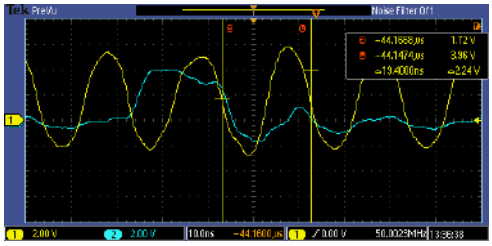


(b) Período de la señal DRDY.

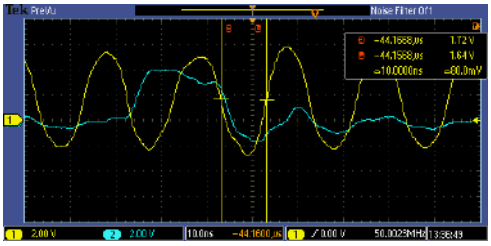


(c) Tiempo en nivel alto de la señal DRDY.

Figura 4.1: Capturas de la señal DRDY en señales que utilizan la memoria SRAM.



(a) Período de la señal DRDY.



(b) Tiempo en nivel bajo de la señal DRDY.

Figura 4.2: Capturas de la señal DRDY en señales que utilizan los scratch-pads.

En la subfigura 4.1a la señal DRDY corresponde al bit 12 mientras que en las subfiguras 4.1b y 4.1c se corresponde con la señal amarilla. Conociendo esto, podemos comprobar que la señal DRDY cambia de acuerdo a los tiempos esperados.

En el caso de las imágenes en 4.2, vemos que los tiempos de la señal DRDY se han reducido a 10 ns en nivel alto y 10 en nivel bajo, consiguiendo así una frecuencia de 50 MHz. En este ejemplo se utilizan los scratch-pads lo cual nos permite leer muestras, por lo menos, 3 veces más rápido que si utilizara la memoria SRAM.

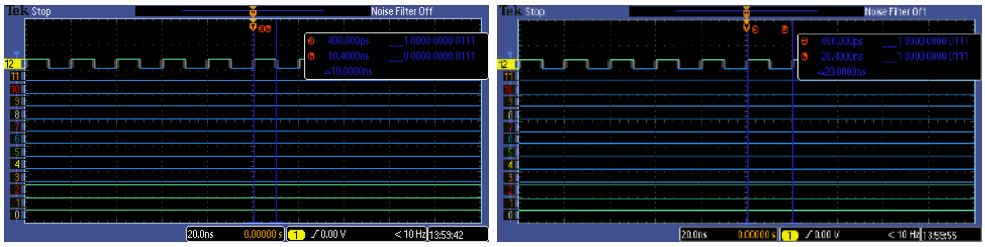
4.3 Valor fijo

Para comprobar que el valor fijo se obtiene correctamente he probado a mostrar el valor 7 siendo el resultado las capturas mostradas en 4.3.

Podemos comprobar que se cumple que la señal de reloj esté 10 ns en nivel alto y 10 ns en nivel bajo. Además, vemos que el valor mostrado por los pines corresponde al 7 que pasado como dato.

4.4 Señal senoidal

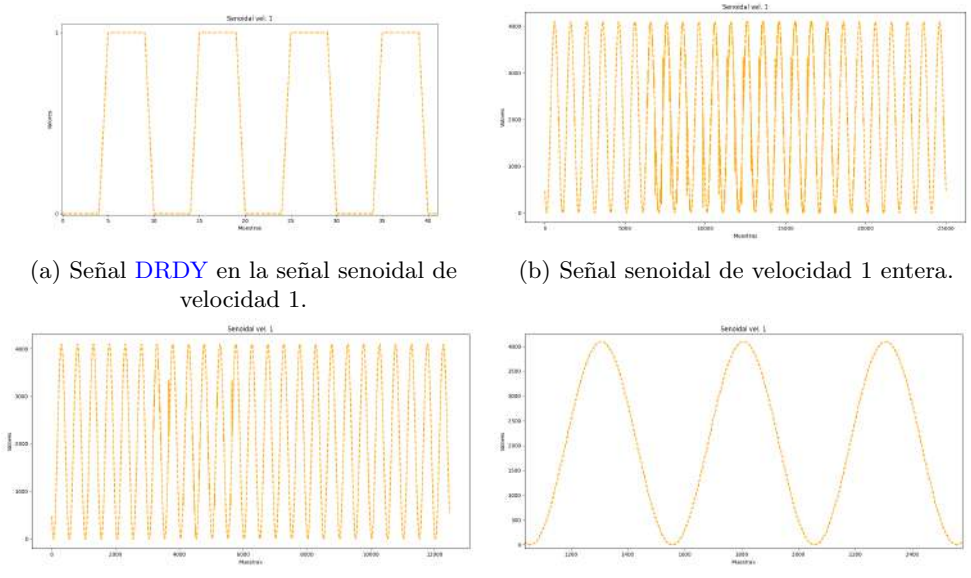
Dado que esta señal dispone de 3 velocidades, compruebo para cada de ellas la señal DRDY, la señal entera y cuando DRDY tiene valor 1.



(a) Tiempo en nivel bajo de la señal DRDY con un valor fijo.

(b) Período de la señal DRDY con un valor fijo.

Figura 4.3: Capturas mostrando un valor fijo.



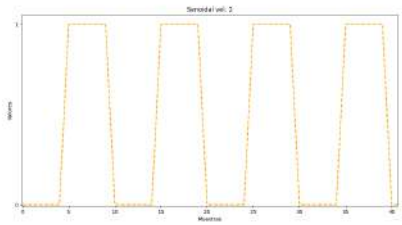
(a) Señal DRDY en la señal senoidal de velocidad 1.

(b) Señal senoidal de velocidad 1 entera.

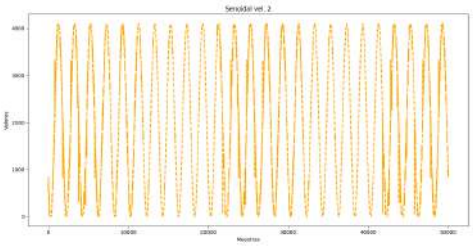
(c) Señal senoidal de velocidad 1 cuando DRDY vale 1.

(d) Señal senoidal de velocidad 1 cuando DRDY vale 1 ampliada.

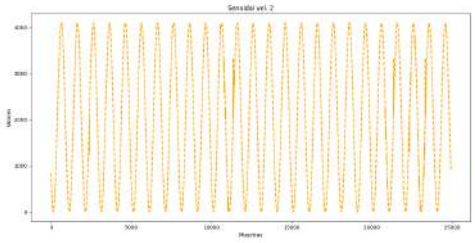
Figura 4.4: Capturas mostrando la señal senoidal de velocidad 1 y su señal DRDY.



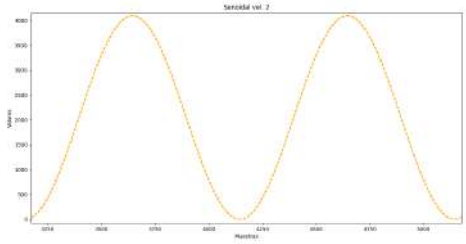
(a) Señal DRDY en la señal senoidal de velocidad 2.



(b) Señal senoidal de velocidad 2 entera.

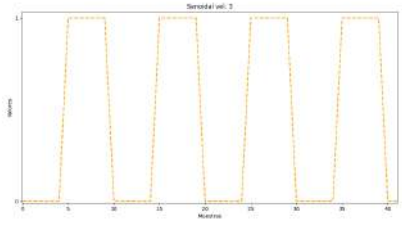


(c) Señal senoidal de velocidad 2 cuando DRDY vale 1.

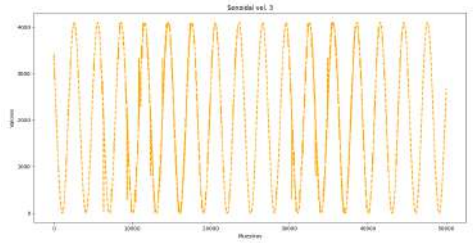


(d) Señal senoidal de velocidad 2 cuando DRDY vale 1 ampliada.

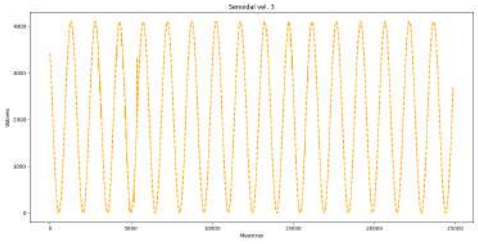
Figura 4.5: Capturas mostrando la señal senoidal de velocidad 2 y su señal DRDY.



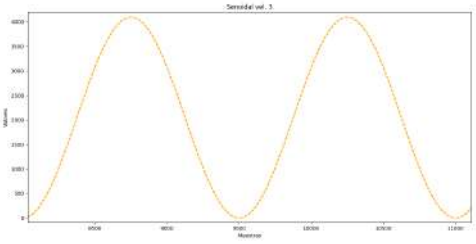
(a) Señal DRDY en la señal senoidal a de velocidad 3.



(b) Señal senoidal de velocidad 3 entera.



(c) Señal senoidal de velocidad 3 cuando DRDY vale 1.



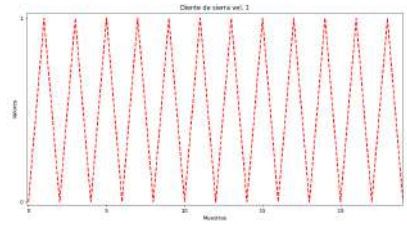
(d) Señal senoidal de velocidad 3 cuando DRDY vale 1 ampliada.

Figura 4.6: Capturas mostrando la señal senoidal de velocidad 3 y su señal DRDY.

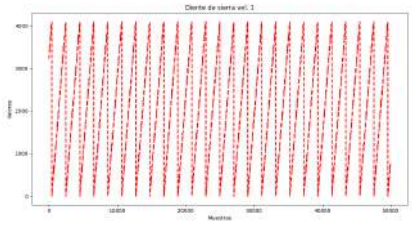
Como podemos observar, tanto las distintas señales DRDY como las señales triangulares se corresponden con lo esperado. También podemos comprobar la diferencia de muestras necesarias para reproducir un ciclo completo en cada una de las velocidades manteniendo la relación de la velocidad 2 tardando el doble que la 1 y la velocidad 3 tardando el triple que la 1.

4.5 Diente de sierra

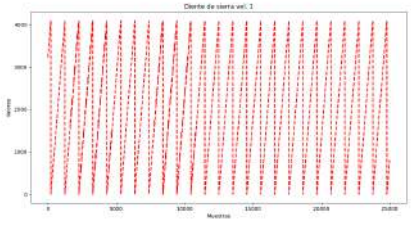
Dado que esta señal dispone de 3 velocidades, compruebo para cada de ellas la señal DRDY, la señal entera y cuando DRDY tiene valor 1.



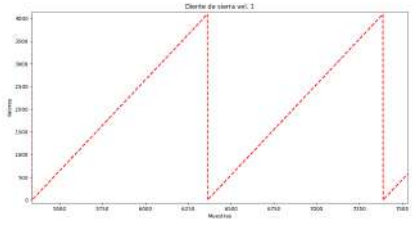
(a) Señal DRDY en el diente de sierra de velocidad 1.



(b) Diente de sierra de velocidad 1 entera.

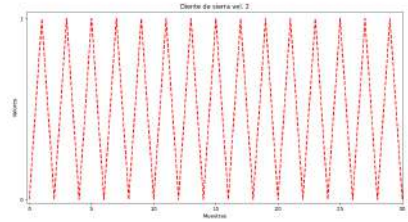


(c) Diente de sierra de velocidad 1 cuando DRDY vale 1.

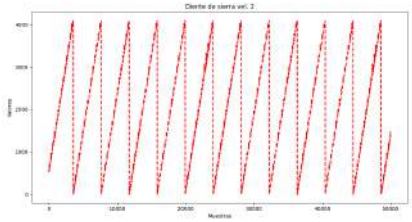


(d) Diente de sierra de velocidad 1 cuando DRDY vale 1 ampliada.

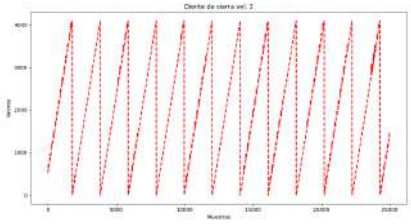
Figura 4.7: Capturas mostrando el diente de sierra de velocidad 1 y su señal DRDY.



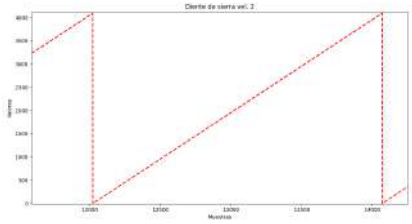
(a) Señal DRDY en el diente de sierra de velocidad 2.



(b) Diente de sierra de velocidad 2 entera.

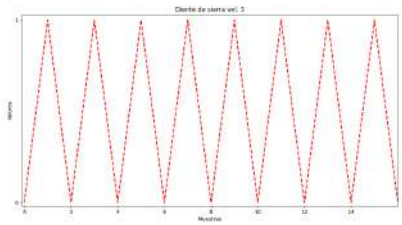


(c) Diente de sierra de velocidad 2 cuando DRDY vale 1.

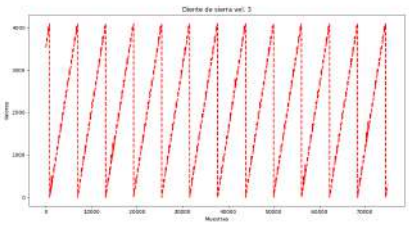


(d) Diente de sierra de velocidad 2 cuando DRDY vale 1 ampliada.

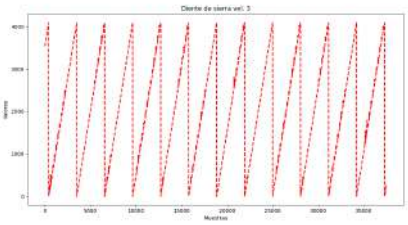
Figura 4.8: Capturas mostrando el diente de sierra de velocidad 2 y su señal DRDY.



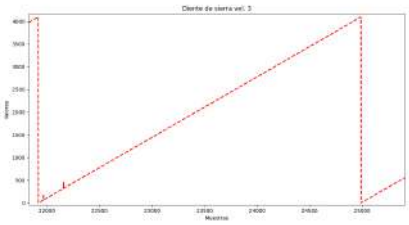
(a) Señal DRDY en el diente de sierra de velocidad 3.



(b) Diente de sierra de velocidad 3 entera.



(c) Diente de sierra de velocidad 3 cuando DRDY vale 1.



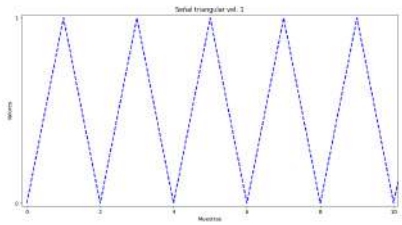
(d) Diente de sierra de velocidad 3 cuando DRDY vale 1 ampliada.

Figura 4.9: Capturas mostrando el diente de sierra de velocidad 3 y su señal DRDY.

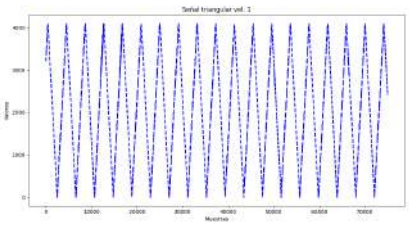
Como podemos observar, tanto las distintas señales DRDY como los dientes de sierra se corresponde con lo esperado. También podemos comprobar la diferencia de muestras necesarias para reproducir un ciclo completo en cada una de las velocidades manteniendo la relación de la velocidad 2 tardando el doble que la 1 y la velocidad 3 tardando el triple que la 1.

4.6 Señal triangular

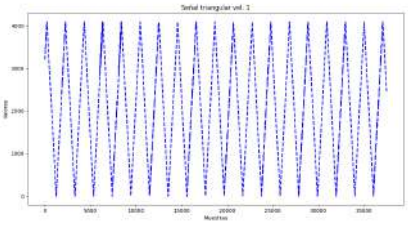
Dado que esta señal dispone de 3 velocidades, compruebo para cada de ellas la señal DRDY, la señal entera y cuando DRDY tiene valor 1.



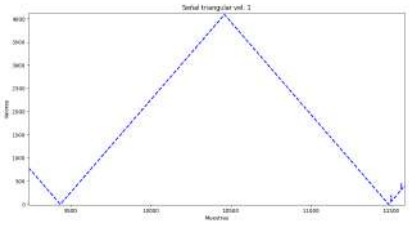
(a) Señal DRDY en la señal triangular de velocidad 1.



(b) Señal triangular de velocidad 1 entera.

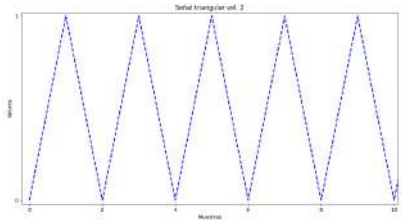


(c) Señal triangular de velocidad 1 cuando DRDY vale 1.

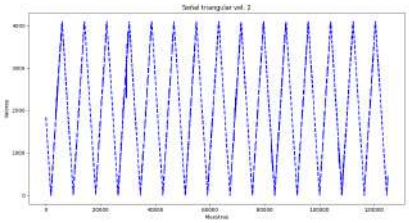


(d) Señal triangular de velocidad 1 cuando DRDY vale 1 ampliada.

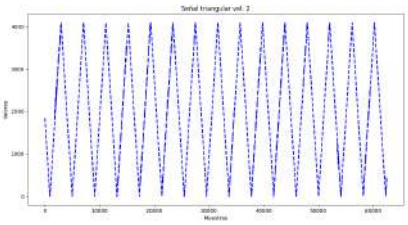
Figura 4.10: Capturas mostrando la señal triangular de velocidad 1 y su señal DRDY.



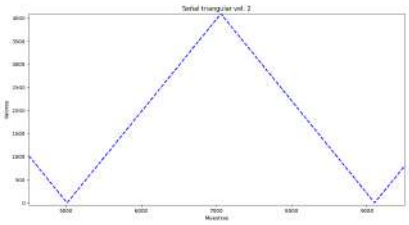
(a) Señal DRDY en la señal triangular de velocidad 2.



(b) Señal triangular de velocidad 2 entera.

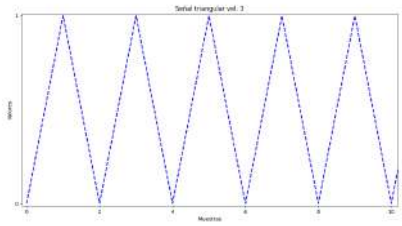


(c) Señal triangular de velocidad 2 cuando DRDY vale 1.

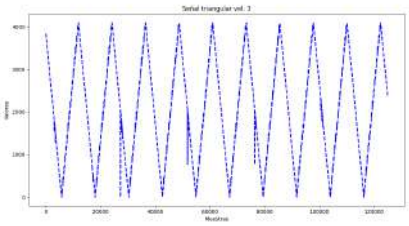


(d) Señal triangular de velocidad 2 cuando DRDY vale 1 ampliada.

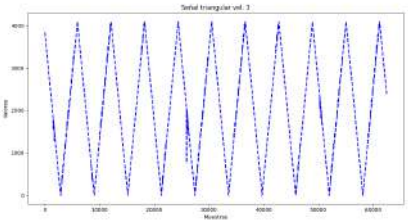
Figura 4.11: Capturas mostrando la señal triangular de velocidad 2 y su señal DRDY.



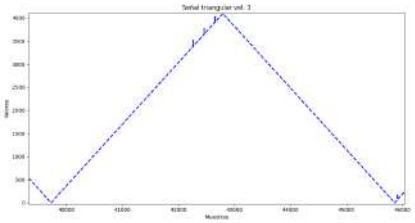
(a) Señal DRDY en la señal triangular de velocidad 3.



(b) Señal triangular de velocidad 3 entera.



(c) Señal triangular de velocidad 3 cuando DRDY vale 1.



(d) Señal triangular de velocidad 3 cuando DRDY vale 1 ampliada.

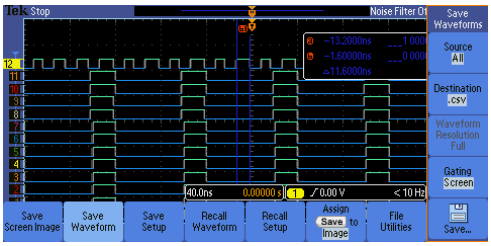
Figura 4.12: Capturas mostrando la señal triangular de velocidad 3 y su señal DRDY.

Como podemos observar, tanto las distintas señales DRDY como las señales senoidales se corresponde con lo esperado. También podemos comprobar la diferencia de muestras necesarias para reproducir un ciclo completo en cada una de las velocidades manteniendo la relación de la velocidad 2 tardando el doble que la 1 y la velocidad 3 tardando el triple que la 1.

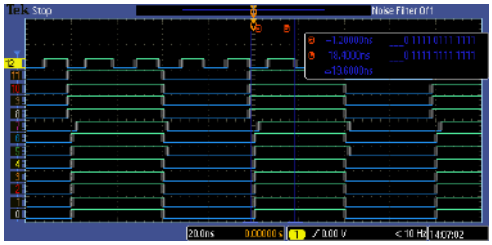
4.7 Señal PWM

Dado que esta señal dispone de 3 velocidades, he comprobado las 3 de forma independiente.

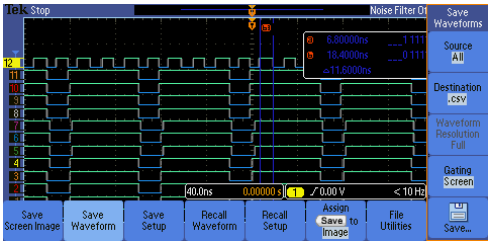
(a) Señal PWM de velocidad 1 con señal DRDY.



(b) Señal PWM de velocidad 1.



(c) Señal PWM de velocidad 2.



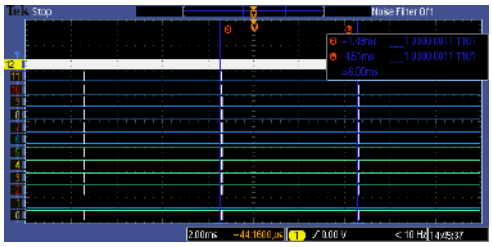
(d) Señal PWM de velocidad 3.

Figura 4.13: Capturas mostrando las señales PWM y sus señales DRDY.

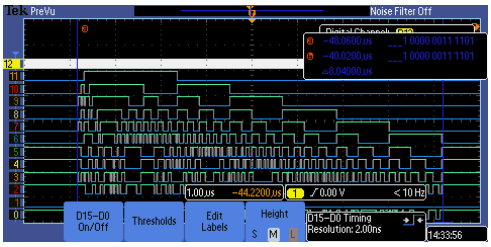
Como se puede ver, en la captura 4.13b percibimos un ciclo de 25 %, en la captura 4.13c es de un 50 % y en la captura 4.13d es de 75 %. En las capturas puede parecer que el los ciclos de la señal DRDY (bit 12) son irregulares, pero como se muestra en la figura 4.13a, los ciclos de la señal DRDY son regulares teniendo 4 ciclos en 80 ns.

4.8 Pulso de radiación

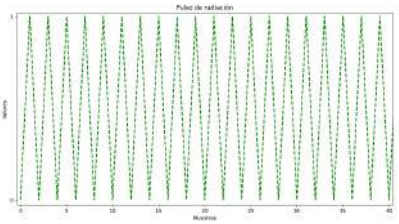
En este caso solo tenemos una opción disponible pero debemos medir tanto el tiempo que dura el pulso como el tiempo entre pulso y pulso.



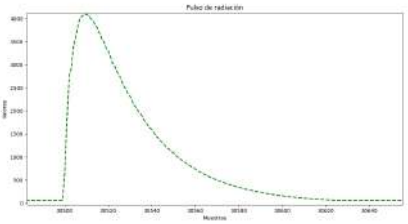
(a) Tiempo entre pulsos de radiación.



(b) Duración de cada pulso de radiación.



(c) Señal DRDY en el pulso de radiación.



(d) Pulso de radiación cuando DRDY vale 1.

Figura 4.14: Capturas el pulso de radiación y su señal DRDY.

Como podemos comprobar en la figura 4.14a, el tiempo entre pulsos es de 6 ms y, como muestra la figura 4.14b, cada pulso tiene una duración de 8 μs. La señal DRDY en 4.14c cambia de valor a cada muestra ya que es una señal larga.

Capítulo 5

Conclusiones y líneas futuras

5.1 Conclusiones

Durante este proyecto he podido comprobar la dificultad que, en trabajos como este, supone encontrar ejemplos que funcionen correctamente debido a que muchos de ellos (como el ejemplo que muestro en 3.3). También conlleva cierta dificultad encontrar la solución a determinados problemas que surgen durante el desarrollo del trabajo dado que solo se puede disponer de dudas previas de otros usuarios.

En general, puedo decir que he conseguido los objetivos propuestos y que he otorgado otra opción para la representación de las señales aparte de la inicial utilizando la memoria SRAM.

5.2 Líneas futuras

Este proyecto establece una base sobre la que poder basar distintos proyectos utilizando una BeagleBone Black. También, gracias a los ejemplos con los scratch-pads, propone una opción que, si fuera necesaria en determinados proyectos, podría usarse para reproducir señales más velozmente. Dada la limitación en cuanto a tamaño de los scratch-pads, para utilizarlos con señales más grandes, sería necesario habilitar y tratar un evento en la dirección PRU1 a PRU0 para que introduzca las siguientes muestras.

Por otro lado, este trabajo podría ser utilizable con otro dispositivo, la BeagleBone AI. Dado que dispone de 2 subsistemas PRU-ICSS dispondríamos de un subsistema extra para otras funciones. Para conseguir esto debería comprobarse la forma de configurar los pines y que pines serían necesarios habilitar ya que existiría una diferencia con la configuración de los pines en la BeagleBone Black.

Bibliografía

[1] "Diagrama funcional del AM335x." https://www.ti.com/product/AM3358 [Último acceso 14/7//2022].

[2] *PRU Read Latencies*, Texas Instruments, December 2018.

[3] "Explicación sobre VirtIO." https://blogs.oracle.com/linux/post/introduction-to-virtio [Último acceso 17/6/2022].

[4] "Página de con distintas imágenes para BeagleBone Black." https://beagleboard.org/latest-images [Último acceso 7/4/2022].

[5] "Página sobre los archivos en `/sys/kernel/debug/pinctrl/`," https://ofitselfso.com/BeagleNotes/UsingDeviceTreesToConfigurePRUIOPins.php [Último acceso 27/4/2022].

[6] "Página de beagleboard mostrando el pinout de la beaglebone black." https://beagleboard.org/support/bone101 [Último acceso 25/4/2022].

[7] *ADC12C080 12-Bit, 65/80 MSPS A/D Converter*, Texas Instruments, April 2013.

[8] "Foro hablando acerca de la velocidad máxima de GPIO de la raspberry pi 3+." https://raspberrypi.stackexchange.com/questions/87846/how-fast-can-gpio-pins-toggle [Último acceso 14/7/2022].

[9] "Latencia al cambiar el estado de GPIO mediante el ARM." https://training.ti.com/sites/default/files/docs/PRU_Building_Blocks_M1_Hardware.pdf [Último acceso 17/6/2022].

[10] *Technical Reference Manual*, Texas Instruments, December 2019.

[11] "Enlace para descargar la herramienta clpru." https://dr-download.ti.com/software-development/ide-configuration-compiler-or-debugger/MD-FaNNGkDH7s/2.3.3/ti_cgt_pru_2.3.3_armlinuxa8hf_busybox_installer.sh [Último acceso 27/4/2022].

[12] *PRU C & C++ compiler*, Texas Instruments, October 2017.

[13] "Archivo makefile de TI." https://git.ti.com/cgit/pru-software-support-package/pru-software-support-package/tree/examples/am335x/PRU_RPMsg_Echo_Interrupt0/Makefile [Último acceso 20/5/2022].

[14] "Página de TI acerca de la disponibilidad de las operaciones MAC," https://www.ti.com/lit/an/sprac90e/sprac90e.pdf?ts=1654165284497&ref_url=https%253A%252F%252Fwww.bing.com%252F [Último acceso 29/5/2022].

[15] *AM335x PRU-ICSS*, June 2013.

[16] "Página de TI acerca de los registros de las pru." https://training.ti.com/sites/default/files/docs/PRU_Compiler_Tips_slides_0.pdf [Último acceso 20/5/2022].

[17] "Página de github sobre prudebug." https://github.com/poopgiggle/prudebug [Último acceso 15/6/2022].

[18] "Página del kernel de Linux acerca remoteproc," https://www.kernel.org/doc/html/latest/staging/remoteproc.html [Último acceso 25/4/2022].

[19] "Página de beagleboard con los distintos SSOO," https://beagleboard.org/latest-images [Último acceso 3/5/2022].

[20] "Página del kernel de linux acerca del directorio `/sys/class/remoteproc/`," https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-class-remoteproc [Último acceso 25/4/2022].

[21] "Explicación de la herramienta config-pin," https://www.bacpeters.com/2020/01/25/configuring-the-beaglebone-black-gpio-pins-permanently/ [Último acceso 11/4/2022].

[22] "Deshabilitar el vídeo en uEnv.txt," https://ofitselfso.com/BeagleNotes/Disabling_Video_On_The_Beaglebone_Black_And_Running_Headless.php [Último acceso 27/4l/2022].

[23] "Página de inicio dentro de la BeagleBone Black," https://beagleboard.org/getting-started [Último acceso 7/4/2022].

[24] "Tabla con offset, modos, GPIO y puerto del P8," https://ofitselfso.com/BeagleNotes/BeagleboneBlackP8HeaderPinMuxModes.pdf [Último acceso 27/4/2022].

[25] "Tabla con offset, modos, GPIO y puerto del P9," https://ofitselfso.com/BeagleNotes/BeagleboneBlackP9HeaderPinMuxModes.pdf [Último acceso 27/4l/2022].

[26] "Discusión acerca del funcionamiento de beaglescript," https://groups.google.com/g/beagleboard/c/mFTpBYv5PNI [Último acceso 11/4/2022].

[27] "Página de klenn klockwood con el ejemplo "Hello world"." https://www.glennklockwood.com/embedded/beaglebone-pru.html#set-gpio-pin-modes [Último acceso 10/4/2022].

[28] "Página de TI con el ejemplo de parpadeo de led." https://www.glennklockwood.com/embedded/beaglebone-pru.html#set-gpio-pin-modes [Último acceso 10/5/2022].

[29] "Ejemplo de BeagleScope usando lenguaje ensamblador." https://github.com/ZeekHuge/BeagleScope/tree/master/examples/firmware_exmples/PRU_inline_asm_blinky [Último acceso 20/5/2022].

[30] "Ejemplo de BeagleScope usando RPMsg y señales entre PRUs." https://github.com/ZeekHuge/BeagleScope/tree/master/examples/firmware_exmples/pru1_to_pru0_to_arm [Último acceso 15/6/2022].

[31] "Ejemplo de TI usando RPMsg y señales entre PRUs." https://git.ti.com/cgit/pru-software-support-package/pru-software-support-package/tree/examples/am335x/PRU_RPMsg_Echo_Interrupt0 [Último acceso 15/6/2022].

[32] G. F. Knoll, *Radiation detection and measurement*. Wiley.

[33] "Página explicando la forma de contar los ciclos de los programas." https://nerdhut.de/2016/06/18/beaglebone-clock-cycle-counter/ [Último acceso 10/6/2022].

Apéndice A

Manual de usuario

A.1 Manual

En primer lugar se debe obtener la BeagleBone Black, un computador y una tarjeta microSD. A continuación se debe usar Balena Etcher para guardar una imagen de Debian 10.3 en la tarjeta microSD (tal y como indica BeagleBone). El siguiente paso consiste en descargar el compilador CLPRU de TI e instalarlo en la BeagleBone Black. El último paso preparativo es descargar la carpeta del trabajo y ejecutar `sudo make` dentro de ella. Esto compilará los programas, colocará los ejecutables para ser ejecutados por las PRU, configurará los pines y compilará el programa conector. Para iniciar las PRUs solo será necesario ejecutar `sudo make start` y para pararlas `sudo make stop`. Una vez iniciadas las PRUs, ejecutaremos el programa conector.

Al ejecutar el programa conector, nos muestra las siguientes opciones:

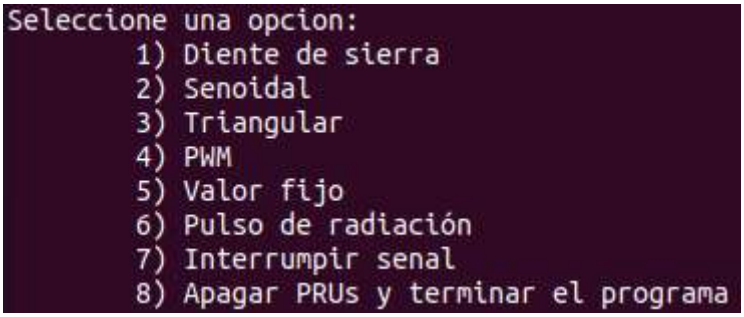


Figura A.1: Menú inicial mostrado por el programa conector.

Para seleccionar la opción deseada debemos introducir el número que lo acompaña (1 para diente de sierra, 2 para senoidal,...).

En caso de seleccionar la opción 1, 2 o 3 permite al usuario elegir la velocidad de la señal.

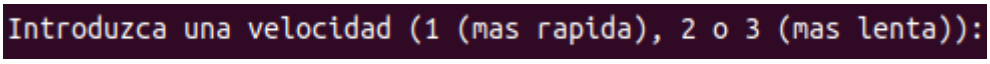


Figura A.2: Menú mostrando las posibilidades acerca de la velocidad de las señales.

En caso de seleccionar la opción 4 se permite al usuario elegir el porcentaje de ciclo en estado alto.

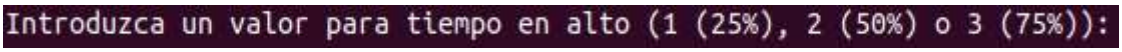


Figura A.3: Menú mostrando las posibilidades acerca de la señal PWM.

En caso de seleccionar la opción 5 se permite al usuario elegir el valor mostrado por los pines.

Figura A.4: Menú pidiendo el valor para mostrar.

Una vez seleccionado el parámetro en las opciones 1, 2, 3, 4 o 5 se mostrará la señal por los pines. En la opción 6 no hace falta parámetro. La opción 7 mantiene el último valor mostrado y se mantiene a la espera de la siguiente señal. En la opción 8 se apagan ambas PRUs y se cierra el programa.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR