EVALUACIÓN Y ANÁLISIS DEL DESEMPEÑO A NIVEL FÍSICO DE UN SISTEMA DE COMUNICACIÓN DE DATOS DE CORTO ALCANCE VÍA RADIO EN 2.4 GHZ BASADO EN ESPECTRO ENSANCHADO POR SECUENCIA DIRECTA (DSSS)

ANEXOS

ANDRÉS FERNANDO ORDÓÑEZ HURTADO
YAMITH ENRIQUE NARVÁEZ MATITUY

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
GRIAL - Grupo de Radio e InALámbricas
GNTT – Grupo I+D Nuevas Tecnologías en Telecomunicaciones
Señales y Sistemas de Acceso y Difusión Basados en Radio
Gestión Integrada de Redes, Servicios y Arquitecturas de Telecomunicaciones
Popayán
2012

EVALUACIÓN Y ANÁLISIS DEL DESEMPEÑO A NIVEL FÍSICO DE UN SISTEMA DE COMUNICACIÓN DE DATOS DE CORTO ALCANCE VÍA RADIO EN 2.4 GHZ BASADO EN ESPECTRO ENSANCHADO POR SECUENCIA DIRECTA (DSSS)

ANEXOS

ANDRÉS FERNANDO ORDÓÑEZ HURTADO
YAMITH ENRIQUE NARVÁEZ MATITUY

Trabajo de Grado presentado como requisito para obtener el título de Ingeniero en Electrónica y Telecomunicaciones

Director
Víctor Manuel Quintero Flórez

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
GRIAL - Grupo de Radio e InALámbricas
GNTT – Grupo I+D Nuevas Tecnologías en Telecomunicaciones
Señales y Sistemas de Acceso y Difusión Basados en Radio
Gestión Integrada de Redes, Servicios y Arquitecturas de Telecomunicaciones
Popayán
2012

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

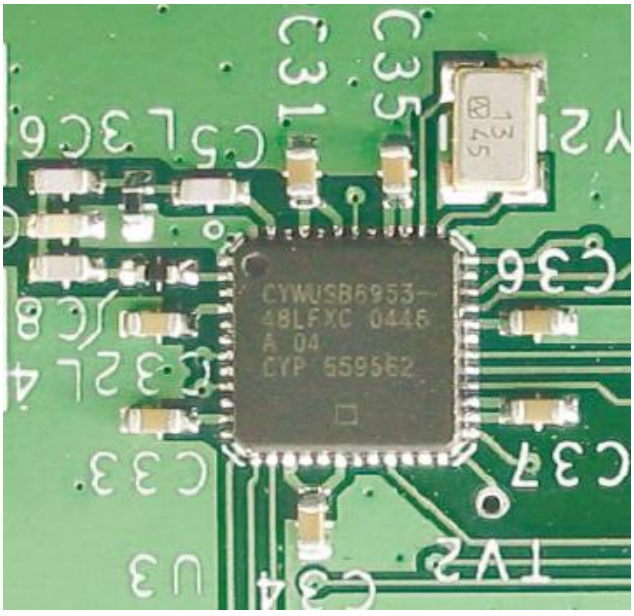
ANEXO A

MANEJO DE LAS HERRAMIENTAS SOFTWARE DEL KIT DE DESARROLLO CY3653 WIRELESSUSB PROC

La configuración de los dispositivos PRoC incluidos dentro del kit de desarrollo CY3653 de *Cypress Semiconductor Corp* se realizó mediante el programa *PSoC Designer*. Esta es una herramienta software que cuenta con un IDE adecuado que facilita la configuración y programación de la mayoría de los dispositivos PSoC distribuidos por *Cypress Semiconductor*, basándose en la premisa: "*Programmability. Flexibility. Ease of use*". En este anexo se describe el proceso necesario para realizar dicha configuración, indicando las instrucciones necesarias para la creación de un proyecto hasta la programación directa del microcontrolador.

Como ya se ha mencionado, el microcontrolador con que cuentan los dispositivos PRoC es el CYWUSB6953, mostrado en la Figura A.1. En este anexo se describen los pasos para la creación de un proyecto basado en este dispositivo.

Figura A.1. Microcontrolador CYWUSB6953



El kit de desarrollo CY3653 cuenta con un Disco Compacto (CD, *Compact Disc*) de instalación de las herramientas *PSoC Designer* y *PSoC Programmer,* cuyas versiones, al momento de realizar la adquisición del kit de desarrollo estaban desactualizadas. Por esta razón, para el desarrollo de este trabajo de grado se utilizaron las versiones *PSoC Designer 5.1* y *PSoC Programmer 3.13*[1], las cuales fueron las más recientes en el momento del desarrollo del sistema prototipo de comunicación.

[1] Las versiones más recientes de estos programas se pueden encontrar en: http://www.cypress.com/?rID=41083.

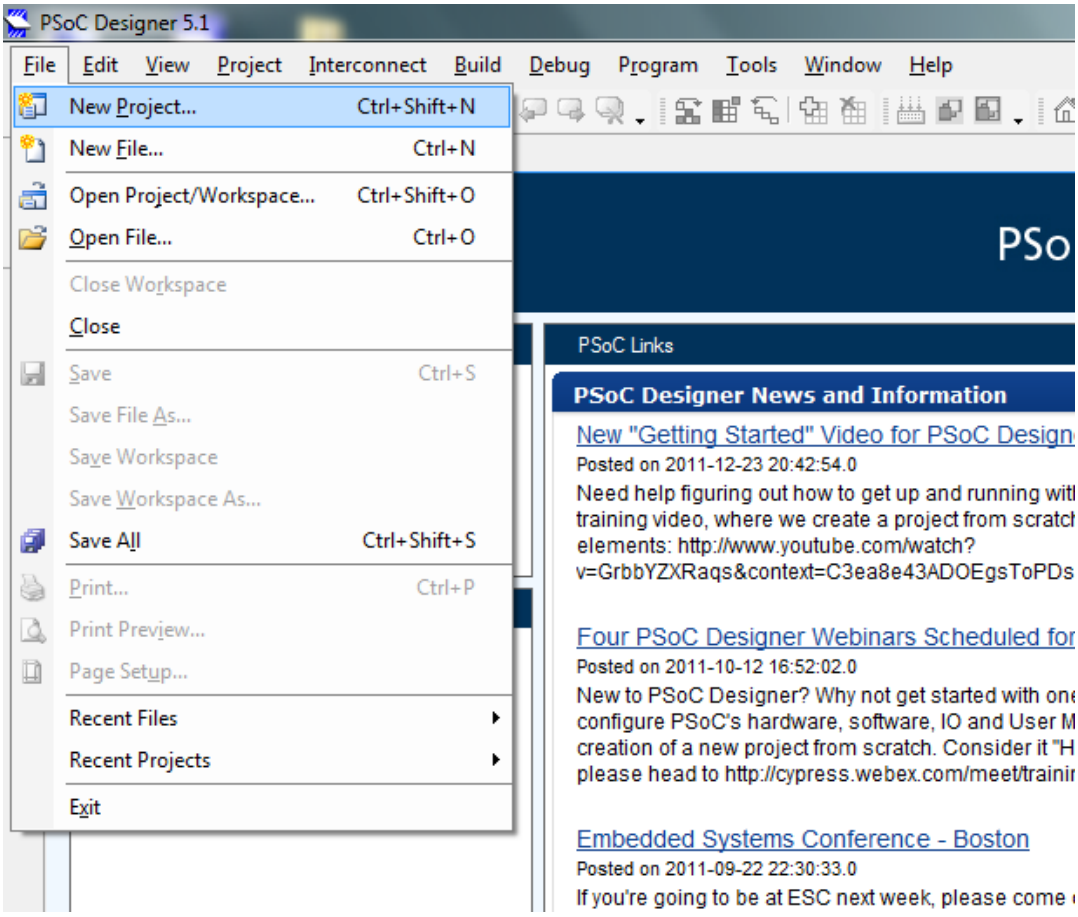
Una vez que se haya completado el proceso típico de instalación, se cuenta con las capacidades para trabajar con estas dos herramientas. La primera de ellas es la que contiene todos los elementos que se requieren para ajustar los parámetros de los dispositivos PRoC de acuerdo a las características del sistema.

A.1. MANEJO DE LA HERRAMIENTA PSoC DESIGNER

La interfaz de usuario del *PSoC Designer* es bastante intuitiva, lo que hace agradable su proceso de manejo para el desarrollo de los proyectos, lo cual se convierte en un aspecto clave para que los nuevos usuarios puedan adquirir experiencia fácilmente.

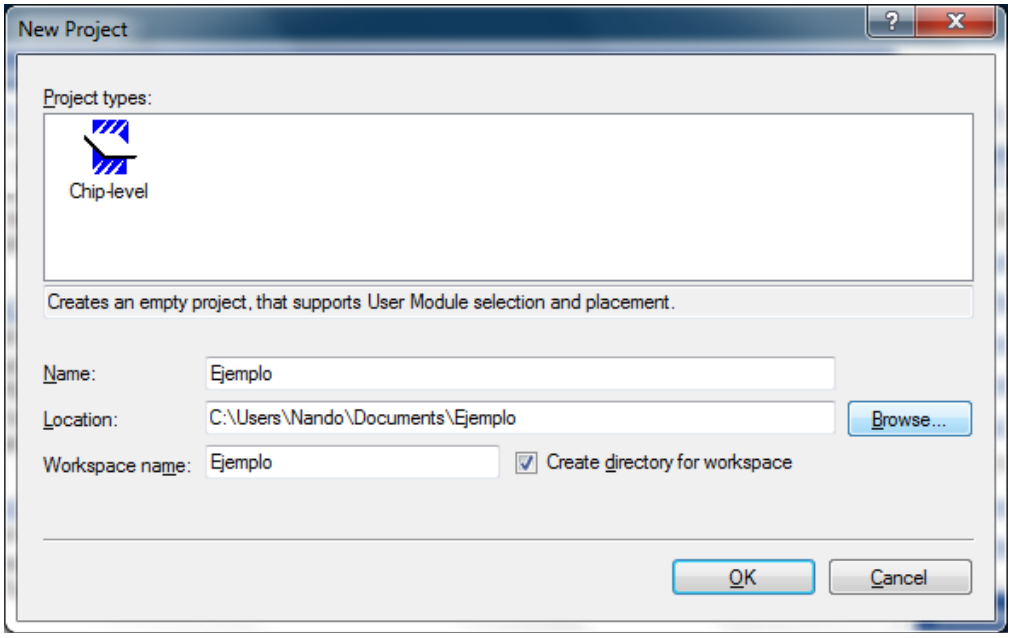
Para la creación de un proyecto, el primer paso consiste en seleccionar la opción *New Project* en el menú *File* ubicado sobre la barra de menú de *PsoC Designer*, como se muestra en la Figura A.2.

Figura A.2. PSoC Designer



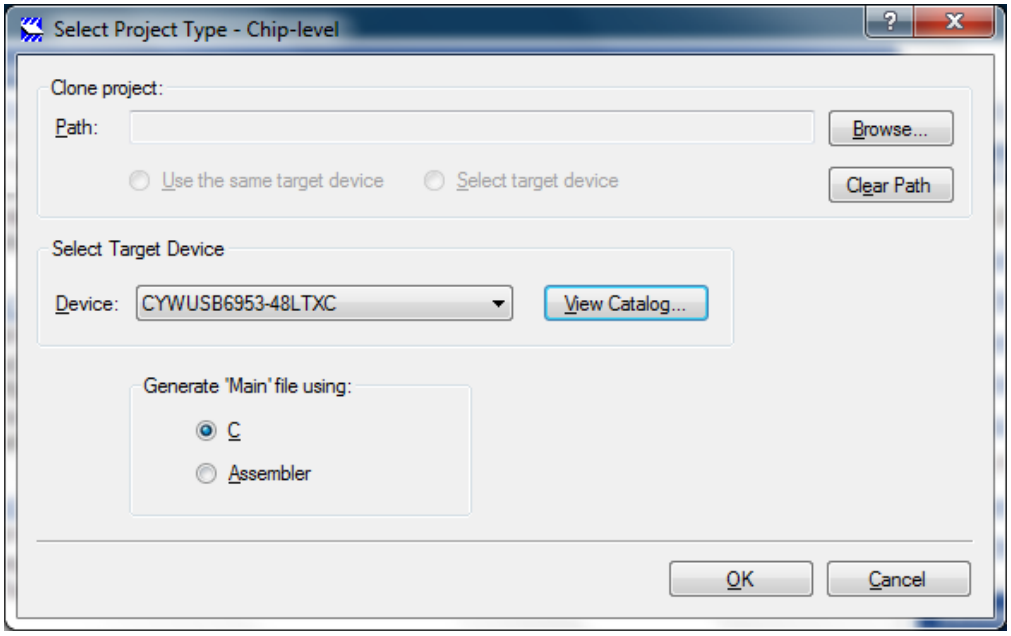
Al seleccionar esta opción, el programa solicita el ingreso de los datos correspondientes al nombre deseado y ubicación del proyecto, como se indica en la Figura A.3.

Figura A.1. Nuevo Proyecto



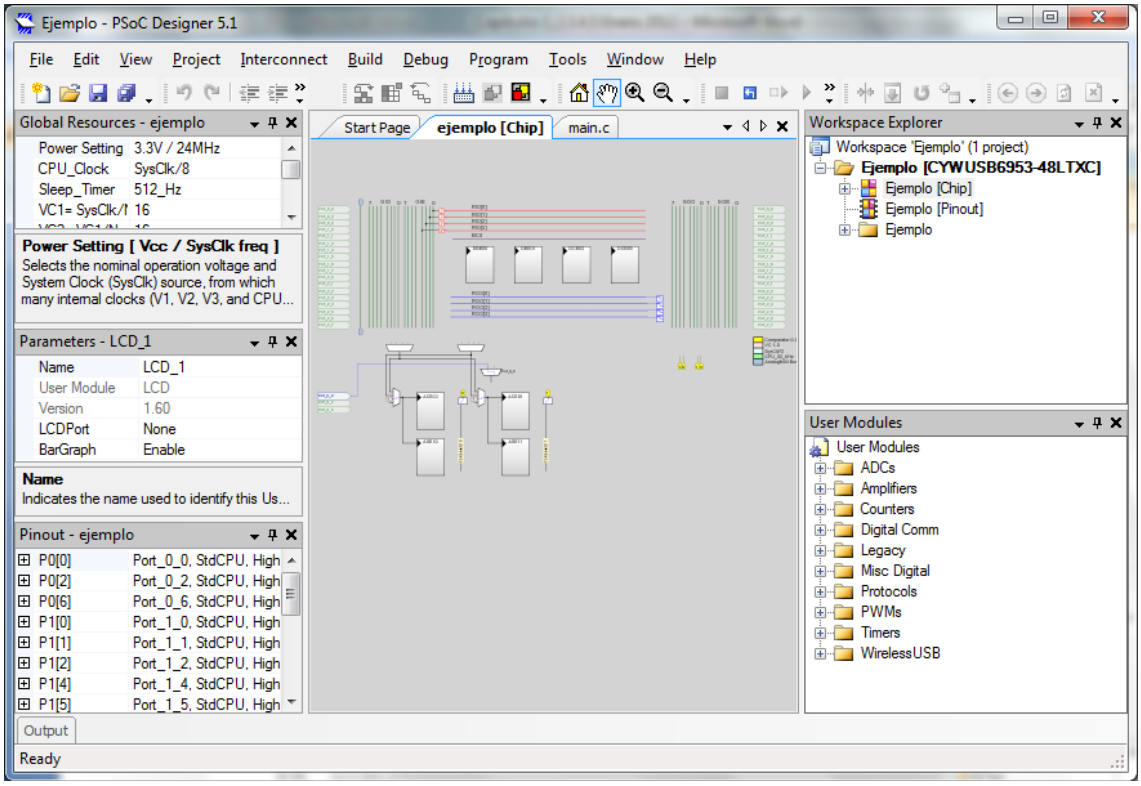
Ahora, es necesario seleccionar el dispositivo base con el cual se va a desarrollar el proyecto y el lenguaje utilizado para la programación del mismo. *PSoC Designer* cuenta con un catálogo del cual es posible seleccionar el dispositivo que se desea usar en el proyecto. Este paso se muestra en la Figura A.4.

Figura A.4. Selección del Dispositivo



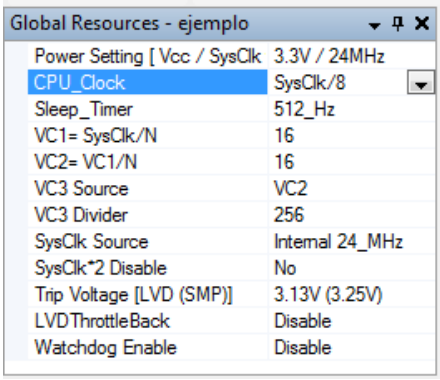
Una vez realizados estos pasos el proyecto ha sido creado exitosamente. El proyecto se abre en la ventana principal, la cual contiene la vista del *Chip Editor*, en donde es posible diseñar el proyecto a nivel del *hardware* interno en el microcontrolador.

Figura A.2. Proyecto creado exitosamente.



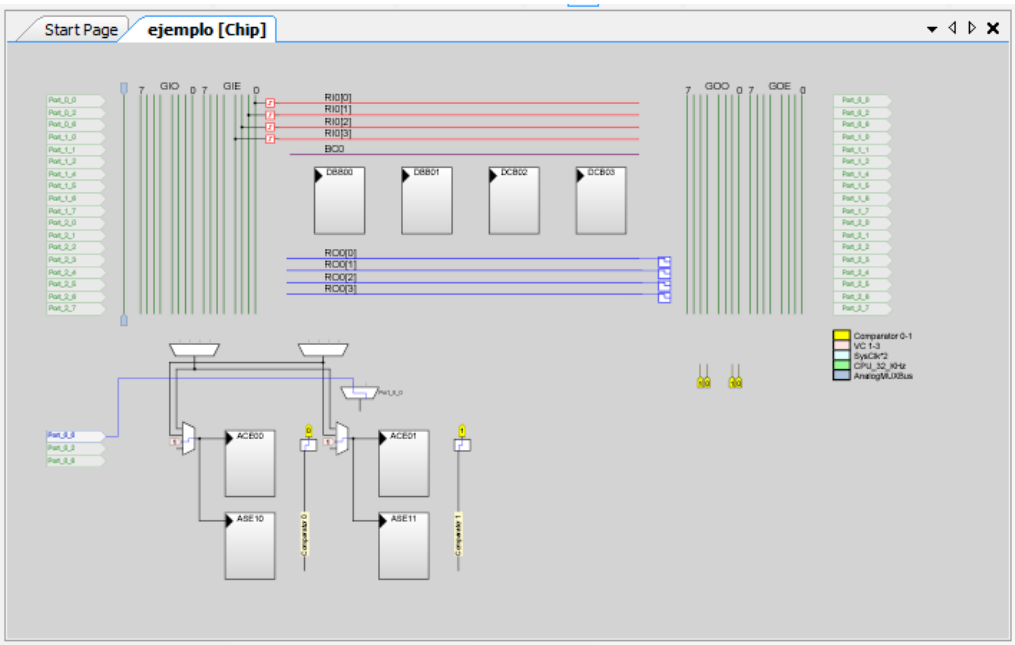
La configuración inicial del proyecto consiste en seleccionar adecuadamente los Recursos Globales (*Global Resources*) del microcontrolador, dentro de los cuales se encuentra el reloj de la CPU y las características de voltaje del microcontrolador, entre otros. Los Recursos Globales del sistema se encuentran sobre la barra lateral izquierda de la interfaz del PSoC Designer, cuyo diseño se muestra en la Figura A.6.

Figura A.3. Recursos Globales



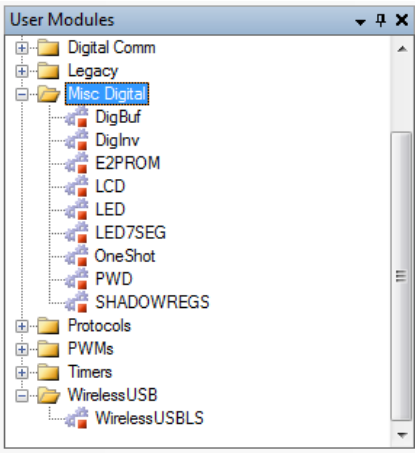
Por otra parte, el CYWUSB6953 cuenta con 4 bloques analógicos y 4 digitales. Los bloques analógicos aparecen agrupados sobre la parte inferior del *Chip Editor*, mientras que los digitales se encuentran en la parte superior, como se muestra en la Figura A.7. También se muestran las entradas y salidas con las que cuenta el microcontrolador. Estas pueden ser usadas para interconectar los puertos con los diferentes módulos de usuario que sean adicionados en los bloques disponibles en el microcontrolador.

Figura A.4. Modulos Analógicos y Digitales



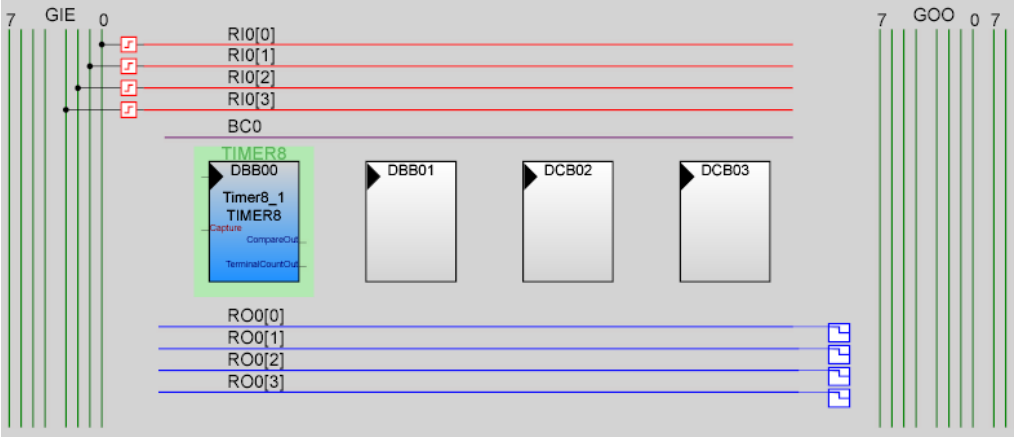
Los módulos de usuario (*User Modules*) son los recursos disponibles con que cuenta el CYWUSB6953, los cuales se pueden agregar al proyecto en cada uno de los bloques mencionados anteriormente. Estos recursos se encuentran ubicados sobre la sección inferior derecha del *Chip Editor*, y se muestran en la Figura A.8.

Figura A.5. Módulos de Usuario



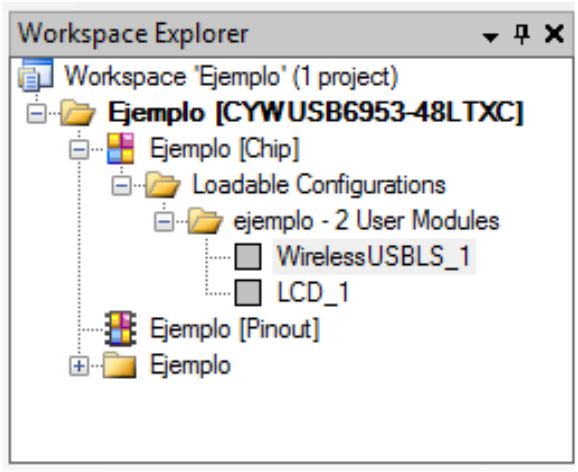
Al hacer doble clic sobre los módulos de usuario, estos se insertan en el proyecto, y pasan a ocupar un lugar dentro de los bloques disponibles, como se muestra en la Figura A.9, donde en forma de ejemplo se ha adicionado un Temporizador (*Timer*) y ha ocupado uno de los bloques digitales.

Figura A.6. Modulos Digitales



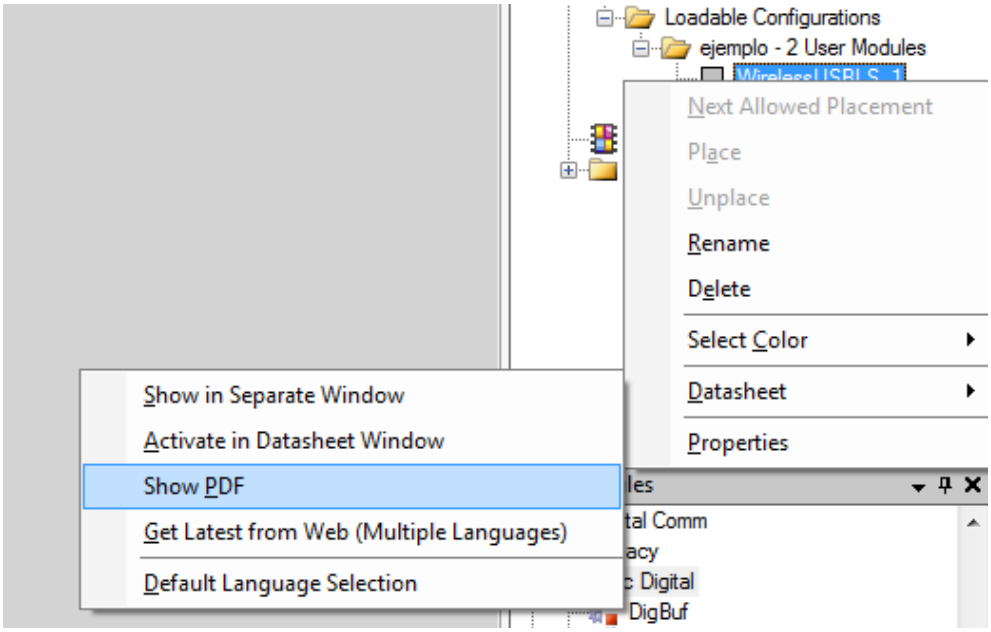
Los módulos de usuario que pertenecen al proyecto también ocupan un lugar en el Espacio de Trabajo (*Workspace*) del proyecto, el cual incluye todos los elementos del proyecto necesarios para la ejecución de este. Para la situación en la que se ha adicionado el módulo de usuario *WirelessUSB* y el módulo LCD, en la Figura A.10 se muestra el *Workspace Explorer* con dichos módulos.

Figura A.7. Workspace Explorer



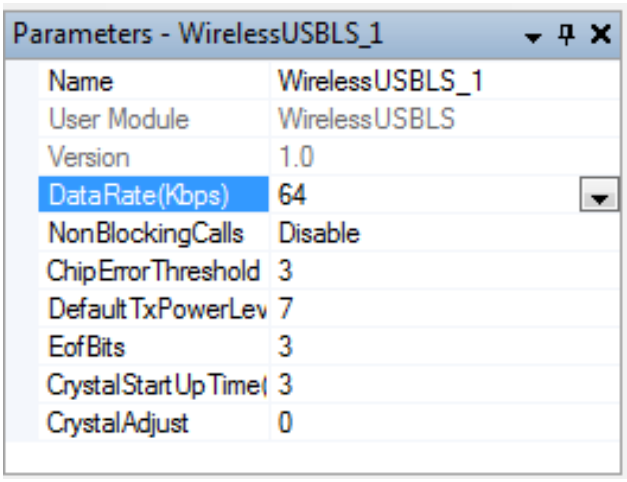
En la Figura A.11 se observa que cada uno de los módulos de usuario cuenta con una hoja de especificaciones (*datasheets*), que contiene los parámetros configurables de los módulos y las diferentes instrucciones de código que se pueden usar dentro del Editor de Código (*Code Editor*) del *PSoC Designer*.

Figura A.8. Hoja de Especificciones de los Módulos de Usuario



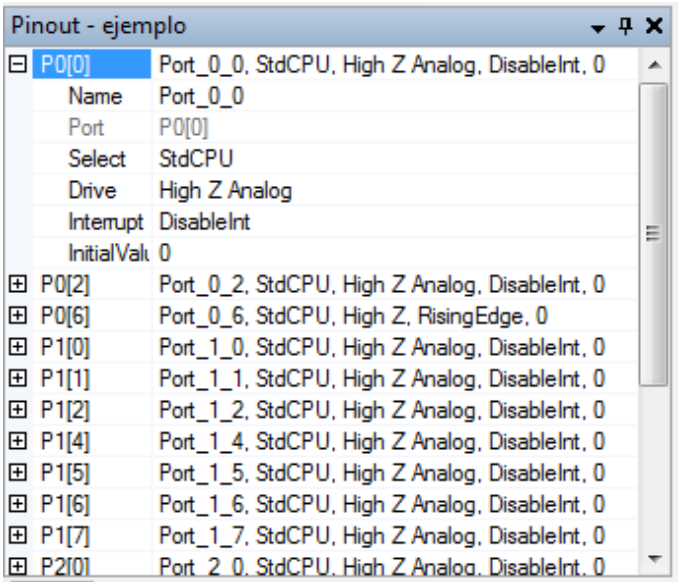
Los parámetros configurables que se pueden modificar directamente se encuentran sobre la barra lateral izquierda, en la sección de Parámetros (*Parameters*), como se muestra en la Figura A.12, en donde se observan los parámetros configurables del módulo *WirelessUSB*. Aquí es posible seleccionar la velocidad de transmisión de datos, la potencia de transmisión y el número máximo permitido de errores, entre otros.

Figura A.9. Parámetros Configurables de los Módulos de Usuario



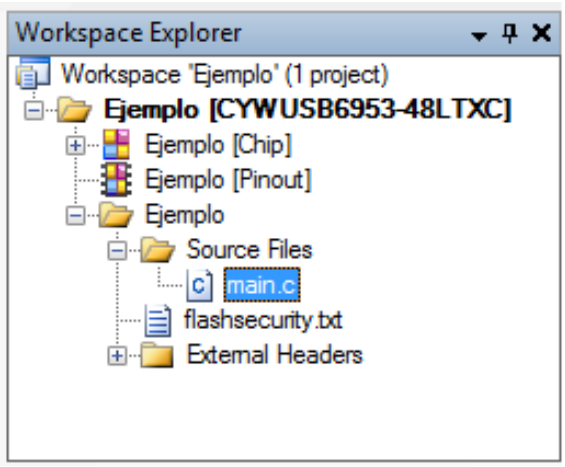
De la misma forma se pueden modificar los puertos del microcontrolador, bien sean de entrada, salida o alta impedancia, así como el tipo de interrupción usada en cada uno de los puertos, como se indica en la Figura A.13.

Figura A.10. Pines de Entrada y Salida del Microcontrolador



Una vez configurados los módulos necesarios para el desarrollo del proyecto se puede continuar con la creación de las rutinas de código que contienen las instrucciones que se desean ejecutar con el microcontrolador. En el *Workspace* es posible encontrar los archivos principales del proyecto y las librerías utilizadas, como se muestra en la Figura A.14.

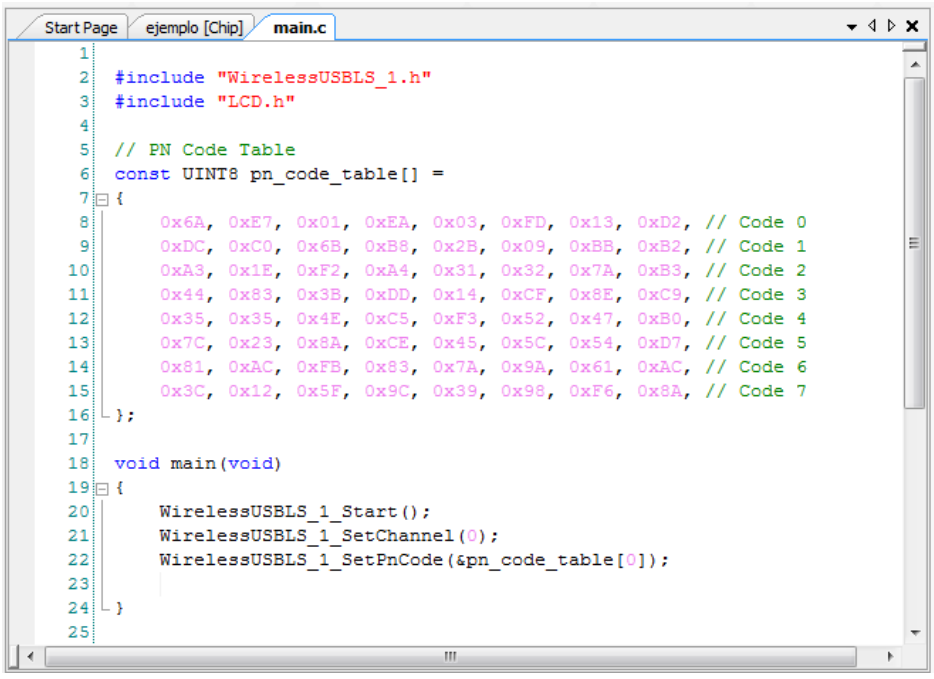
Figura A.11. Archivos Fuente del Proyecto



Por ahora solo se encuentra el archivo principal (*main*) dentro del proyecto. Sobre este archivo se escriben las líneas de código con las instrucciones que el microcontrolador debe realizar. Si se desea se pueden agregar más archivos fuente al proyecto, en el caso de librerías o componentes de código que requieren una ejecución externa del archivo principal.

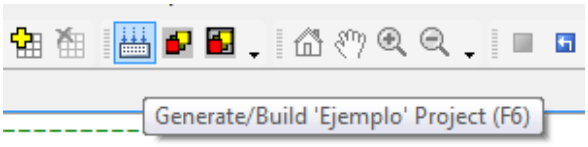
La configuración del módulo radio con que cuenta el CYWUSB6953 se realiza en su mayoría mediante líneas de comando, ya que se cuenta con rutinas de selección de la secuencia pseudo-aleatoria, la frecuencia de operación y el envío y recepción de la información. Todas las rutinas se especifican en el *datasheet* del dispositivo, así como los parámetros que cada una de ellas necesita para su ejecución.

Figura A.12. Editor de Codigo



La compilación del proyecto se realiza con F6 o directamente sobre el botón *Generate/Build*, que se muestra en la Figura A.16. Esta opción se encarga de cargar las librerías usadas por cada uno de los módulos de usuarios que pertenecen al proyecto, y también de crear los archivos necesarios para la programación física del microcontrolador.

Figura A.13. Compilación del Proyecto

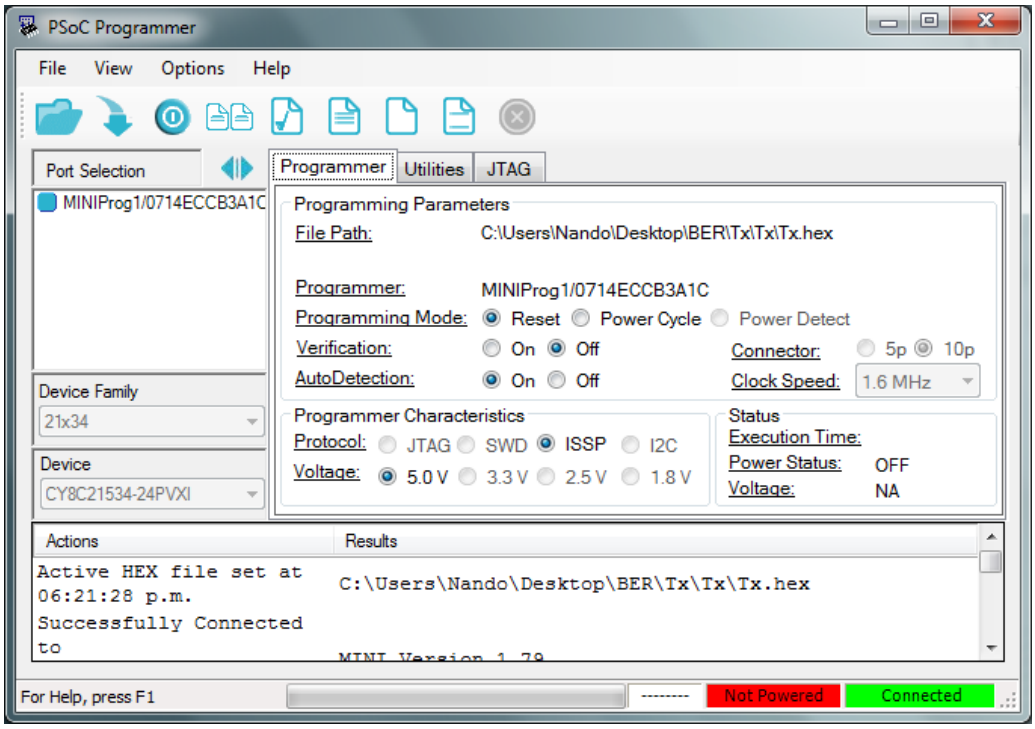


A.2. MANEJO DE LA HERRAMIENTA PSoC PROGRAMMER

La programación del microcontrolador se realiza con el *PSoC Programmer*. Este programa es responsable de cargar la configuración realizada mediante código (resumida en el archivo de extensión *.hex*) directamente sobre el microcontrolador.

Como se observa en la Figura A.17, en primer lugar es necesario establecer la ubicación del archivo *.hex*, el cual se encuentra dentro de la carpeta que contiene el proyecto. Por otra parte se selecciona el dispositivo a programar manualmente, o si se desea es posible seleccionar la opción de auto detección, para seleccionarlo automáticamente. Es necesario seleccionar el Modo de Programación (*Programming Mode*) en *Reset*, ya que el microcontrolador debe estar conectado a una fuente de voltaje externo de 3.3 V para realizar la programación exitosamente, por lo cual es obligatorio retirar el jumper de su posición. El programador en este caso es el *MiniProg* con el que cuenta el kit de desarrollo.

Figura A.1. PSoC Programmer



ANEXO B

CARACTERIZACIÓN DE LA POTENCIA DE RECEPCIÓN

Los valores de potencia de recepción se midieron con los módulos PRoC en términos de RSSI, que corresponden a valores enteros entre 0 y 31. Estos valores son el resultado de la conversión analógica a digital de la intensidad de la potencia recibida en un determinado canal. Con el objetivo de facilitar la manipulación de los datos y obtener los resultados en términos comúnmente conocidos, es necesario transformar los valores de RSSI a dBm. Debido a que la resolución del ADC es de 5 bits, no se cuenta con absoluta precisión del RSSI medido, así como tampoco es posible obtener un nivel de RSSI para todos los niveles de potencia.

La transformación de RSSI a dBm se realiza haciendo uso de los datos disponibles en la hoja de especificaciones del CYWUSB6935 [13], mediante una interpolación lineal entre los valores medidos de RSSI y los valores en dBm, para el caso más sencillo, que corresponde a las medidas de potencia de recepción en el escenario exterior, pues como se mencionó en el Capitulo 4, el comportamiento de las pérdidas de propagación en términos de RSSI en este escenario es similar al descrito por el modelo de propagación de la Ecuación 3.2.

En la Tabla B.1 se muestran los valores de RSSI medidos en el escenario exterior, con los cuales fue posible realizar la interpolación, pues el comportamiento en este escenario es intuitivo.

Tabla B.1 RSSI en Términos de dBm

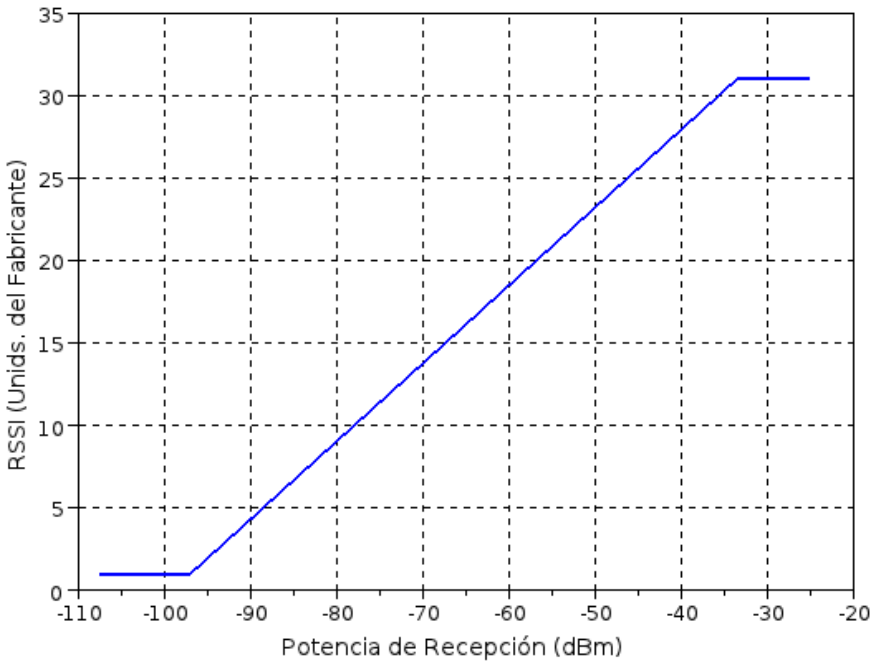
RSSI (Unidades del Fabricante)	Potencia (dBm)
1	-97.115385
2	-95
3	-92.884615
4	-90.769231
5	-88.653846
6	-86.538462
7	-84.423077
8	-82.307692
9	-80.192308
10	-78.076923
11	-75.961538
12	-73.846154
13	-71.730769
14	-69.615385
15	-67.5
16	-65.384615
17	-63.269231
18	-61.153846
19	-59.038462
20	-56.923077

Tabla B.1. (Continuación)

21	**-54.807692**
22	-52.692308
23	-50.576923
24	-48.461538
25	-46.346154
26	-44.230769
27	-42.115385
28	-40
29	-37.884615
30	-35.769231
31	-33.653846

La Figura B.1 representa la relación gráfica entre RSSI y la potencia de recepción en dBm.

Figura B.1. Relacion entre RSSI y Potencia de Recepcion



En las Tablas B.2 y B.3 se resumen los valores de RSSI, potencia de recepción y pérdidas de propagación en función de la distancia, cuyas graficas se muestran en sección 3 del Capítulo 4.

En la Tabla B.2 se muestran los valores de RSSI medidos en el escenario exterior y su correspondiente valor de potencia en dBm. Además se muestran los valores de las pérdidas de propagación, calculados mediante la Ecuación 4.15.

Tabla B.2. Potencia Recibida y Pérdidas de Propagación en Escenario Exterior

Distancia (m)	RSSI (Unidades del Fabricante)	Potencia de Recepción (dBm)	Pérdidas de Propagación (dB)
0.1	31	-29.423077	19.723077
0.2	31	-33.653846	23.953846
0.3	30	-35.769231	26.069231
0.4	28	-40	30.3
0.5	26	-44.230769	34.530769
1	21	-54.807692	45.107692
1.5	20	-56.923077	47.223077
2	18	-61.153846	51.453846
2.5	18	-61.153846	51.453846
3	17	-63.269231	53.569231
3.5	16	-65.384615	55.684615
4	15	-67.5	57.8
4.5	15	-67.5	57.8
5	14	-69.615385	59.915385
5.5	14	-69.615385	59.915385
6	13	-71.730769	62.030769
6.5	13	-71.730769	62.030769
7	12	-73.846154	64.146154
7.5	12	-73.846154	64.146154
8	12	-73.846154	64.146154
8.5	12	-73.846154	64.146154
9	11	-75.961538	66.261538
9.5	11	-75.961538	66.261538
10	11	-75.961538	66.261538
11	10	-78.076923	68.376923
12	10	-78.076923	68.376923
13	9	-80.192308	70.492308
14	9	-80.192308	70.492308
15	8	-82.307692	72.607692
16	8	-82.307692	72.607692
17	7	-84.423077	74.723077
18	7	-84.423077	74.723077
19	7	-84.423077	74.723077
20	6	-86.538462	76.838462
21	6	-86.538462	76.838462
22	5	-88.653846	78.953846
23	5	-88.653846	78.953846
24	5	-88.653846	78.953846
25	5	-88.653846	78.953846
26	5	-88.653846	78.953846
27	4	-90.769231	81.069231
28	4	-90.769231	81.069231
29	4	-90.769231	81.069231
30	4	-90.769231	81.069231

En la Tabla B.3 se muestran los valores de RSSI medidos en el escenario interior y su correspondiente valor de potencia en dBm. Además se muestran los valores de las pérdidas de propagación, calculados mediante la Ecuación 4.15.

Tabla B.3. Potencia Recibida y Pérdidas de Propagación en Escenario Interior

Distancia (m)	RSSI (Unidades del Fabricante)	Potencia (dBm)	Pérdidas de Propagación (dB)
1	24	-48.461538	38.761538
2	18	-61.153846	51.453846
3	19	-59.038462	49.338462
4	16	-65.384615	55.684615
5	14	-69.615385	59.915385
6	15	-67.5	57.8
7	14	-69.615385	59.915385
8	13	-71.730769	62.030769
9	12	-73.846154	64.146154
10	10	-78.076923	68.376923
11	11	-75.961538	66.261538
12	10	-78.076923	68.376923
13	9	-80.192308	70.492308
14	10	-78.076923	68.376923
15	11	-75.961538	66.261538
16	10	-78.076923	68.376923
17	9	-80.192308	70.492308
18	7	-84.423077	74.723077
19	9	-80.192308	70.492308
20	9	-80.192308	70.492308
21	10	-78.076923	68.376923
22	10	-78.076923	68.376923
23	9	-80.192308	70.492308
24	9	-80.192308	70.492308
25	8	-82.307692	72.607692
26	8	-82.307692	72.607692
27	9	-80.192308	70.492308
28	10	-78.076923	68.376923
29	9	-80.192308	70.492308
30	9	-80.192308	70.492308
31	8	-82.307692	72.607692
32	7	-84.423077	74.723077
33	9	-80.192308	70.492308
34	10	-78.076923	68.376923
35	9	-80.192308	70.492308
36	10	-78.076923	68.376923
37	9	-80.192308	70.492308
38	8	-82.307692	72.607692
39	8	-82.307692	72.607692
40	7	-84.423077	74.723077

Tabla B.3. (Continuación)

41	**7**	**-84.423077**	**74.723077**
42	7	-84.423077	74.723077
43	6	-86.538462	76.838462
44	6	-86.538462	76.838462
45	6	-86.538462	76.838462
46	5	-88.653846	78.953846

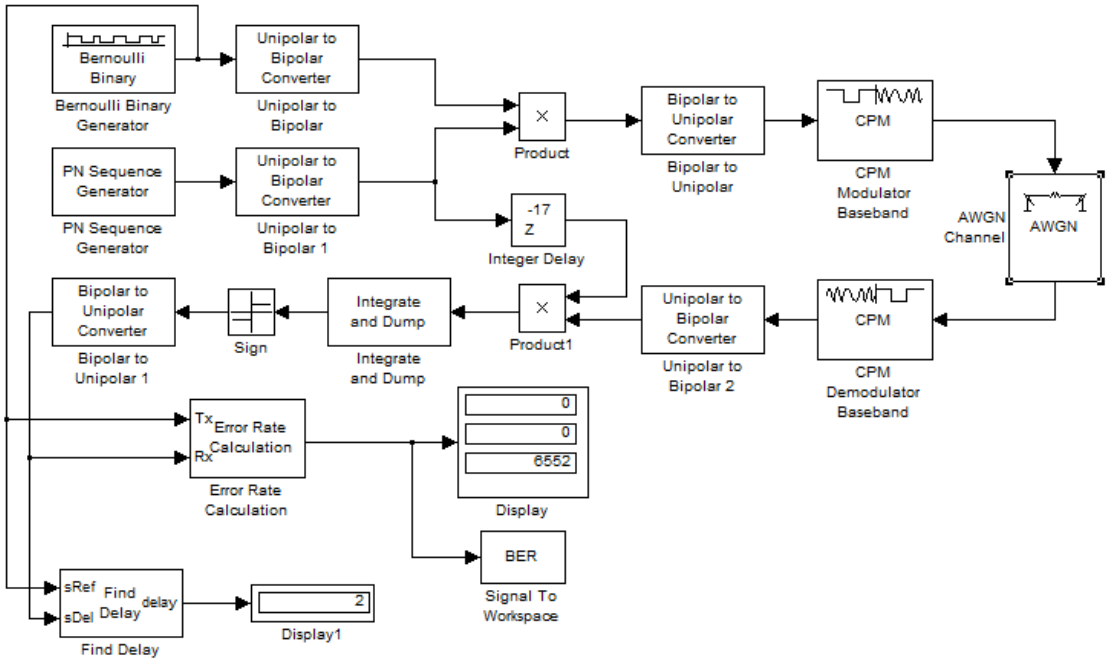
Con los valores de potencia de recepción calculados en los dos escenarios es posible encontrar los valores de E_b necesarios para realizar el análisis del desempeño del sistema descrito en la Sección 4.4.

ANEXO C

DESCRIPCIÓN DE LA SIMULACIÓN DEL SISTEMA DE COMUNICACIÓN

Para tener un referente teórico acerca del comportamiento del sistema de comunicación implementado mediante los dispositivos PRoC, se realizó una simulación en la herramienta Simulink®, en donde se represento el conjunto de elementos del sistema mediante bloques que simulan su comportamiento. El diagrama en bloques del sistema realizado en Simulink se muestra en la Figura C.1, en donde se han agregado los bloques que ejecutan las funciones de ensanchamiento, desensanchamiento, modulación y demodulación del sistema de comunicación. El sistema incluye bloques que se encargan de realizar la adecuación de las señales digitales entre cada uno de los procesos del sistema. También se ha agregado un bloque que simula el comportamiento de un canal AWGN, necesario para realizar la simulación básica del canal de comunicaciones. Por último, para visualizar los resultados de desempeño del sistema, la simulación cuenta con bloques de despliegue para visualizar los resultados obtenidos al finalizar la simulación.

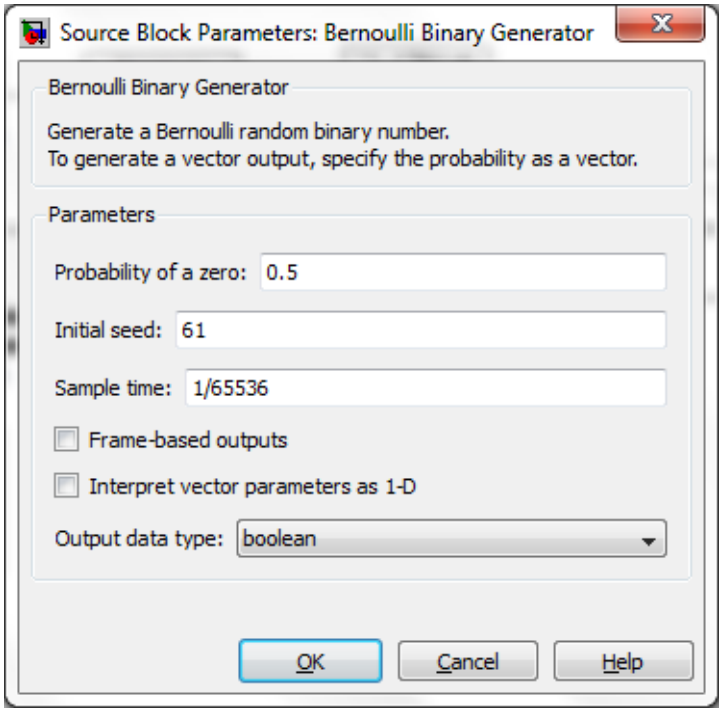
Figura C.1. Diagrama en Bloques del Sistema Simulado



- Generador Binario de Bernoulli (*Bernoulli Binary Generator*): Este bloque es el encargado de generar la información que se desea transmitir. Se encarga de generar aleatoriamente los dígitos binarios que se desean transmitir, de acuerdo al tiempo de muestro (*sample time*) que se ingrese. En el caso de la Figura C.2, se genera una señal aleatoria binaria a una velocidad de 64 kbps, que corresponde a un tiempo de muestreo igual a 1/65536, como se muestra en la Figura C.2, y, con la misma probabilidad de generar símbolos de 0 y 1. Para generar señales coyas velocidades de transmisión de datos sean 16 y 32 kbps, este valor debe establecerse en 1/16384 y

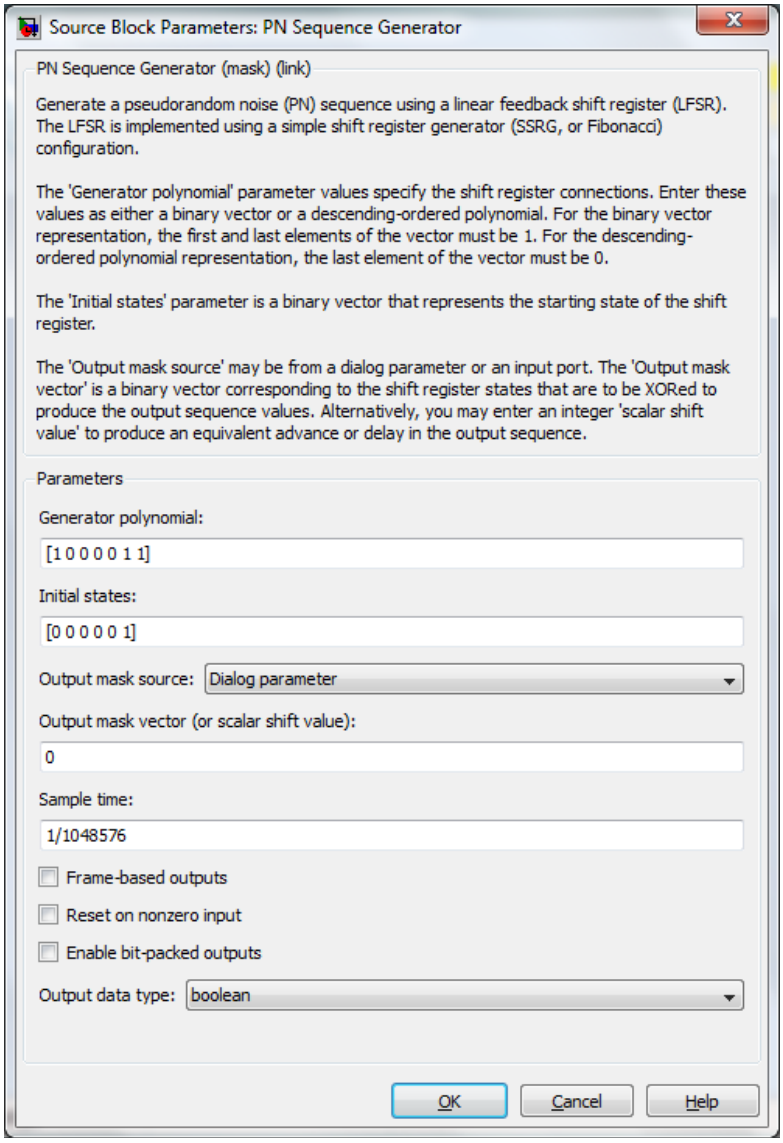
1/32768, respectivamente. Debido a que se desean obtener símbolos binarios, el tipo de datos de salida (*output data type*) de este bloque es booleano (*boolean*).

Figura C.1. Generador Binario de Bernoulli



- Generador de Secuencias Pseudo-aleatorias (*PN Sequence Generator*): Este bloque es el encargado de generar la secuencia pseudo-aleatoria usada en el proceso de ensanchamiento y de desensanchamiento de la información. El tiempo de muestreo para la generación de la secuencia pseudo-aleatoria debe ser menor que el usado para la generación de la información, ya que de esta forma, al realizar la combinación entre ellas se obtenga una señal de alta velocidad. En el caso de la Figura C.3 se ha generado una secuencia pseudo-aleatoria de 1 Mbps. El polinomio generador (*generator polynomial*) corresponde al asignado por defecto al agregar el bloque. Nuevamente el tipo de datos de salida de este bloque es booleano (*boolean*).

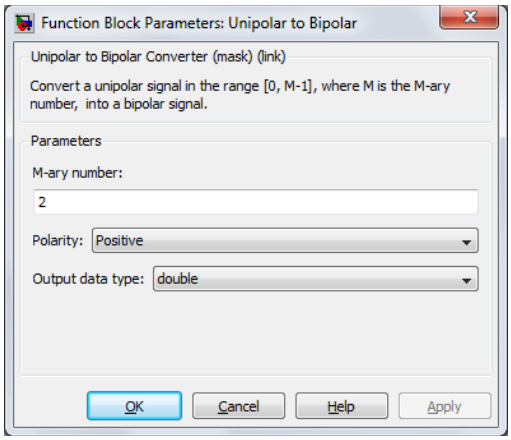
Figura C.2. Generador de Secuencias Pseudo-aleatorias



- Conversor de Unipolar a Bipolar (*Unipolar to Bipolar Converter*): Este bloque se encarga de asignar valores positivos y negativos normalizados a las señales binarias con el objetivo de facilitar el proceso de multiplicación entre la señal de datos y la secuencia pseudo-aleatoria. El bloque se encarga de asignar el valor de -1 a su salida si en su entrada se encuentra un valor 0 lógico, y un valor de +1 a su salida si en su entrada detecta un valor de 1 lógico. En este caso el tipo de salida es un valor de punto flotante (*double*), o si se desea puede ser un entero (*integer*).

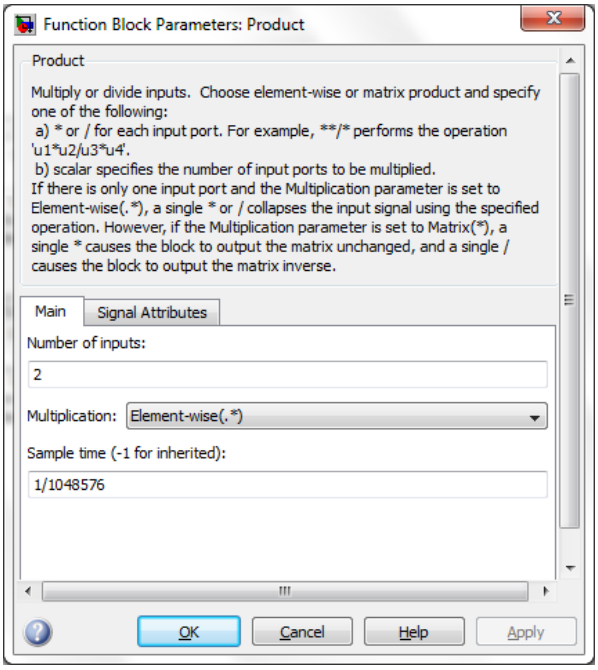
  El bloque encargado de realizar el proceso contrario se denomina Conversor de Bipolar a Unipolar (*Bipolar to Unipolar Converter*). La diferencia entre estos dos bloques se encuentra en el tipo de datos de salida, pues en este último se deben obtener valores binarios a la salida del bloque.

Figura C.3. Conversor de Unipolar a Bipolar



- Producto (*Product*): Este bloque es el encargado de realizar la multiplicación directa entre las señales correspondientes a los datos y a la secuencia pseudo-aleatoria. Por esta razón este bloque se configura para que reciba 2 entradas (*inputs*) como se muestra en la Figura C.5. El producto se realiza elemento-a-elemento (*element-wise*), y el tiempo de muestreo corresponde al mismo usado para la generación de las secuencias pseudo-aleatorias. De esta forma se garantiza que la señal de salida posea la misma velocidad que la secuencia de ensanchamiento.

Figura C.4. Producto



- Modulador CPM de Banda Base (*CPM Modulator Baseband*): Este bloque se encarga de realizar la modulación GFSK basado en un esquema de modulación de fase

continua. Los parámetros necesarios para producir la modulación GFSK se muestran en la Figura C.6, en donde se ha seleccionado el valor de 2 como número M-ario (*M-ary number*) ya que se trata de un esquema de la familia FSK, en donde cada símbolo binario se representa por una frecuencia diferente. El valor de índice de modulación se calculó a partir de la Ecuación C.1, usando las características físicas del radiotransmisor.

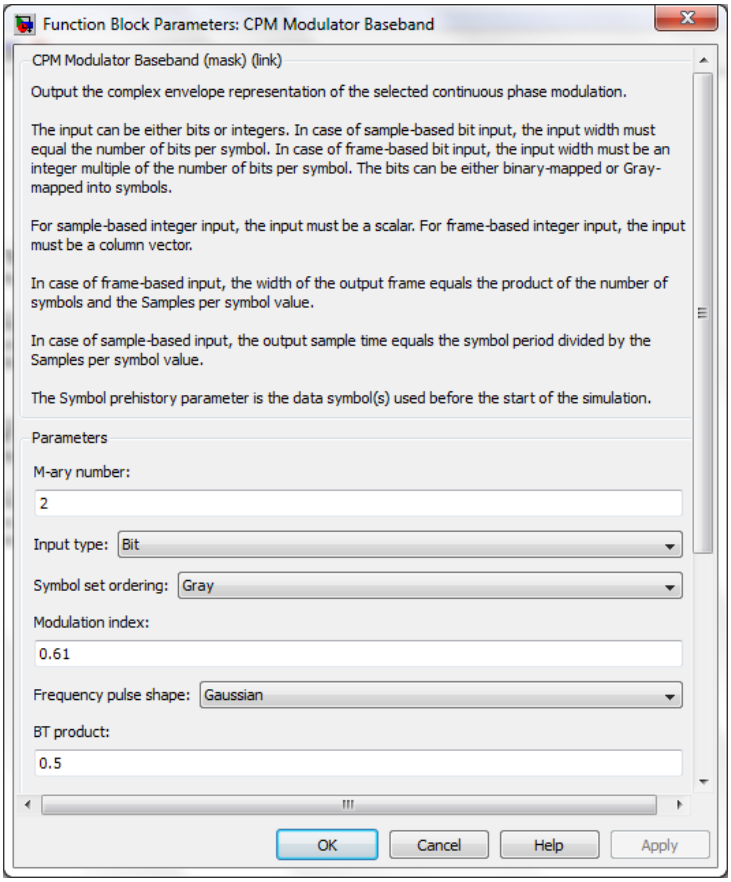
$$Indice\ de\ Modulación = \frac{2\Delta f}{R_c} \tag{C.1}$$

De acuerdo a la Ecuación C.1, el índice de modulación de los dispositivos PRoC corresponde a:

$$Indice\ de\ Modulación = \frac{640\ kHz}{1Mcps} = 0.61$$

Necesariamente se debe seleccionar un filtro para la conformación de pulsos (*pulse shaping*) del tipo Gaussiano (*Gaussian*), y para la situación real, con un valor del parámetro del filtro *BT* igual a 0.5, según el fabricante.

Figura C.5. Modulador de Fase Contínua en Banda Base



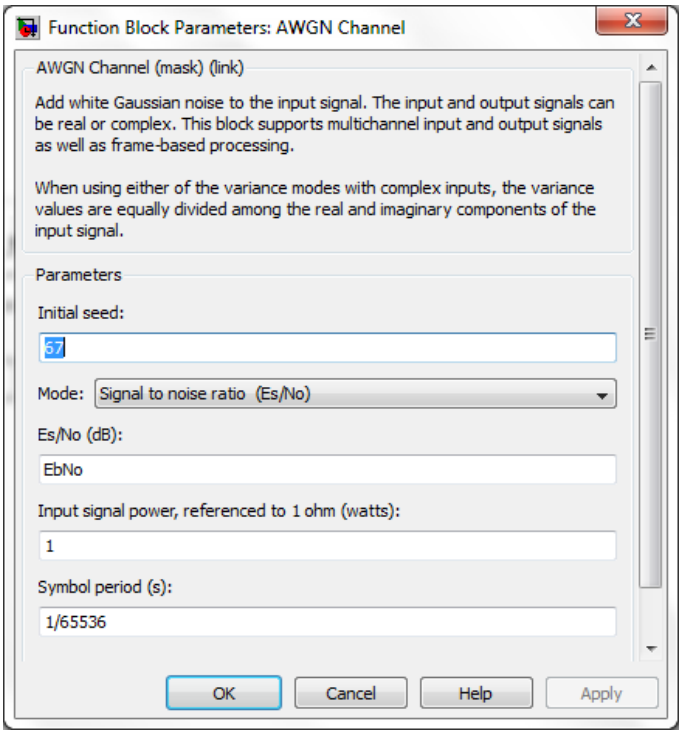
Los parámetros del bloque de modulación de la Figura C.6 se configuran de manera similar en el bloque de demodulación.

El proceso de modulación y demodulación implica un retardo de la señal recibida con respecto a la señal transmitida. Por esta razón, antes de realizar el desensnchamiento de la información es necesario sincronizar la información con la secuencia pseudo-aleatoria. Para ello se introdujo un bloque que adiciona un retardo igual a 17 muestras de la secuencia pseudo-aleatoria, como se muestra en el diagrama en bloques de la Figura C.1, calculado mediante un bloque Calculador de Retardos (*Find Delay*).

- Canal AWGN (*AWGN Channel*): Este bloque es el encargado de representar el comportamiento de un canal AWGN, a partir de las variaciones de la relación E_b/N_0, E_s/N_0 o SNR. Para el caso particular, en donde la señal de entrada posee una velocidad de 1 Mbps (velocidad de *chips*), se ha definido un periodo de sñimbolo igual a 1/65536, correspondiente a una velocidad de bits igual a 64 kbps. Este parámetro varía de acuerdo a la velocidad de datos del sistema, pues a partir de estos valores es posible definir adecuadamente los valores de E_b/N_0.

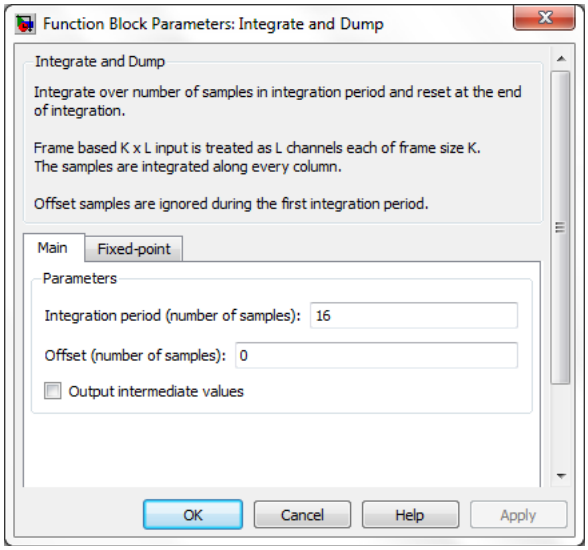
  Cabe aclarar que el módulo AWGN asume los datos de entrada como bits, a pesar de que a la entrada de este bloque se encuentran los *chips* producto del ensanchamiento. Por esta razón es necesario aclarar que, cuando el bloque se refiere a símbolos, se está hablando de bit, y cuando este mismo hace referencia a bits, se toman en cuenta los *chips*. Esto indica que, las gráficas obtenidas de desempeño en la Sección 4.31 corresponden en términos reales a la relación E_b/N_0.

Figura C.6. Canal AWGN



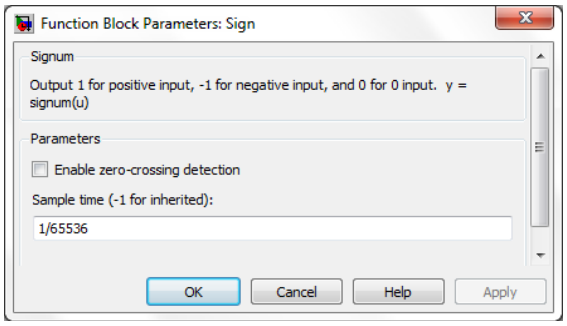
- Integración y Descarga (*Integrate and Dump*): De acuerdo al periodo de integración (*Integration period*) este bloque se encarga de acumular los valores muestreados en este tiempo, y finalmente restablece el valor. En la Figura C.8 el periodo de integración de 16 corresponde a una velocidad de transmisisón de datos igual a 64 kbps, mientrs que para las velocidades iguales a 16 y 32 kbps, estos valores equivalen a 64 y 32, respectivamente, y corresponden al Factor de Ensanchamiento. En la simulación este es el bloque que hace el papel de correlacionar la información recibida de acuerdo a la secuencia pseudo-aleatoria.

Figura C.7. Integración y Descarga



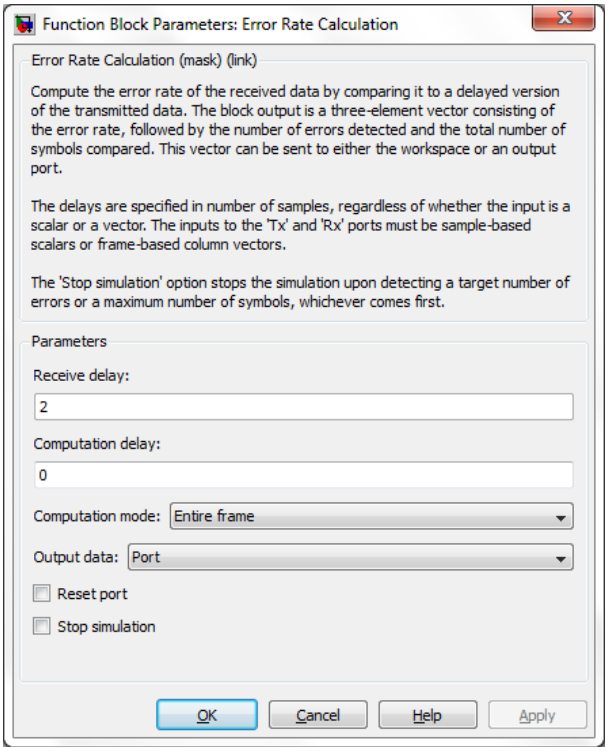
- Signo (*Sign*): Este bloque se encarga de generar una señal normalizada de acuerdo a los valores de entrada del bloque, a un determinado tiempo de muestreo, que en este caso corresponde al tiempo de cada bit. Este bloque da como resultado un valor igual a 1 si a la entrada se presenta un valor positivo, mientras que si se tiene en la entrada un valor negativo, la salida es -1. En el caso de la Figura C.9 el tiempo de muestreo corresponde a 64 kbps. Para las demás velocidades de transmisión del sistema este parámetro cambia de la misma forma que en el Generador Binario de Bernoulli, tomando los mismos valores de muestreo.

Figura C.8. Signo



- Cálculo de Tasa de Errores (*Error Rate Calculation*): Una vez que se ha recuperado la información en el extremo final del sistema de comunicación es pertinente verificar el estado de los datos recibidos, mediante una comparación realizada elemento a elemento entre cada uno de los bits recibidos y los bits originales que se transmitieron. Este bloque permite realizar esta comparación, teniendo en cuenta que las señales deben estar perfectamente sincronizadas, para que no existan errores en la comparación. Dentro de la configuración de este bloque que se muestra en la Figura C.10 se configura el Retardo de transmisión (*Receive delay*) con un valor de 2 muestras, lo que indica que el bloque se encarga de retardar las señales recibidas para sincronizarlas con las señales originales, y así realizar con éxito dicha comparación. Este retardo se debe a que las señales han sufrido un proceso de modulación y demodulación, lo cual hace que aparezcan tiempos de latencia en la transmisión de la información.

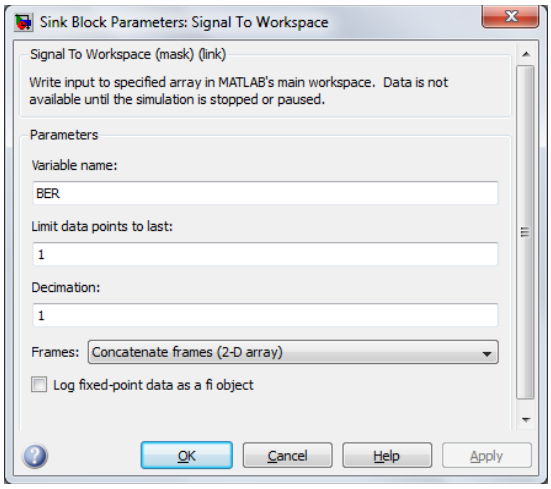
Figura C.9. Cálculo de Tasa de Errores



- Señal al Espacio de Trabajo (*Signal to Workspace*): Este bloque es el encargado de enviar los datos correspondientes a BER hacia un espacio de trabajo exterior, para que sean tomadas por BERTool[2], obtenidos como resultado de la simulación al modificar la variable E_b/N_0 descrita en el canal AWGN. En el bloque de la Figura C.11 es necesario solamente establecer el nombre de la variable, para que el *Workspace* la interprete para visualizar el desempeño del sistema.

[2] BERTool es una herramienta de interfaz grafica para el análisis del desempeño a nivel de BER y E_b/N_0 en MATLAB.

Figura C.10. Señal al Espacio de Trabajo



ANEXO D

ARCHIVOS DE CONFIGURACIÓN DE LOS DISPOSITIVOS PRoC

Para la evaluación de sistema de comunicación vía radio de corto alcance en 2.4 GHz propuesto en el trabajo de grado, fue necesario implementar rutinas de código en lenguaje de programación C en cada uno de los módulos transceptores PRoC con el fin obtener los valores de RSSI, y número de errores de bits que permiten calcular la BER, para luego hacer el análisis y desempeño según los resultados obtenidos en las diferentes pruebas realizadas.

El envío de la información se realizó haciendo uso de un protocolo de comunicación de datos básico, donde se conformaron las tramas de acuerdo a las especificaciones descritas en la Sección 3 del Capítulo 3.

D.1. PROTOCOLO DE INFORMACIÓN

Debido a que fue necesario la implementación de un protocolo de comunicación básico en el que simplemente el modulo transmisor PRoC enviara cierta cantidad de tramas con determinado numero de bits cada una sin que se manejara algún tipo de retransmisiones, se desarrolló una rutina de código que se encarga de conformar las tramas de información. A continuación se muestran las rutinas que se ejecutan el proceso de conformación de los campos de la trama de información:

```
#ifndef _PROTOCOL_H_
#define _PROTOCOL_H_

#define LENGTH  25

typedef struct _FRAME
{
//! Parity bit of data packet
UINT8 preamble[3];

//! Type of device part of data packet
UINT8 synchro;

//! Data toggle information of data part of packet
 UINT8 length;

//! Buffer to hold data packet
UINT8 payload[LENGTH];

//! To hold calculated checksum of data packet being received
UINT8 checksum;

//! To hold calculated checksum of data packet being received
UINT8 finished;

} MY_FRAME;

//! LS packet for transmitting data
#define MY_TX_FRAME MY_FRAME
```

```
//! Packet for transmitting
extern MY_TX_FRAME  tx_frame;

#endif // _PROTOCOL_H_
```

D.2. DETECCIÓN DE BITS Y TRAMAS ERRADAS

La siguiente rutina de código se implementó para el envío de un núumero fijo de tramas en el módulo transmisor PRoC, en donde se habilita un pin del microcontrolador para que reciba la señal proveniente de un interruptor (*switch*) para que, al ser presionado, se envíe la información por medio de tramas. En este modulo también se elige la secuencia pseudo-aleatoria, la potencia de transmisión y la frecuencia de operación del modulo PRoC. Por otra parte, antes de realizar el proceso de transmisión, se calcula el valor de suma de verificación o *ckecksum*, para la detección de errores en el receptor, y se ingresa este valor en el campo correspondiente en la trama.

```
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------

#include "WirelessUSBLS.h"
#include "cypdef.h"
#include "psocgpioint.h"
#include "lcd.h"
#include "stdlib.h"
#include "delay.h"
#include "protocol.h"

UINT8 button_pressed (void);
UINT8 calc_checksum(UINT8 *data);

UINT8   button_up = TRUE;

//*****************************
// Códigos PN
//*****************************

const UINT8 pn_code_table[] =
{
    0x6A, 0xE7, 0x01, 0xEA, 0x03, 0xFD, 0x13, 0xD2, // Código 1
    0xDC, 0xC0, 0x6B, 0xB8, 0x2B, 0x09, 0xBB, 0xB2, // Código 2
    0xA3, 0x1E, 0xF2, 0xA4, 0x31, 0x32, 0x7A, 0xB3, // Código 3
    0x44, 0x83, 0x3B, 0xDD, 0x14, 0xCF, 0x8E, 0xC9, // Código 4
    0x35, 0x35, 0x4E, 0xC5, 0xF3, 0x52, 0x47, 0xB0, // Código 5
    0x7C, 0x23, 0x8A, 0xCE, 0x45, 0x5C, 0x54, 0xD7, // Código 6
    0x81, 0xAC, 0xFB, 0x83, 0x7A, 0x9A, 0x61, 0xAC, // Código 7
    0x3C, 0x12, 0x5F, 0x9C, 0x39, 0x98, 0xF6, 0x8A, // Código 8
};

//*****************************
// Bytes a Transmitir
//*****************************
UINT8 data[] =
 {
       0xAE, 0x35, 0x6B, 0xB5, 0x38,
       0xA9, 0x1C, 0x48, 0x25, 0x1C,
       0x91, 0x5F, 0xA4, 0x55, 0xF7,
```

```
       0xB6, 0x3E, 0x8B, 0x5D, 0xC2,
       0x4E, 0x05, 0xF2, 0xDA, 0x23
};

//Checksum F3
UINT8 frame_preamble[3] = {0xAA, 0xAA, 0xAA};

//Tamano del Vector a Transmitir
UINT8 size = sizeof(data);
//Canal
//UINT8 channel = 76;
UINT8 channel = 60;
//Secuencia Pseudoaleatoria
UINT8 code = 0;
//Potencia de Transmisión
UINT8 power = 4;
//Semilla de Checksum
UINT8 seed = 0x5E;

MY_FRAME   tx_frame;


//*****************************
// Ciclo Principal
//*****************************
void main(void)
{
// Definición de Variables
UINT8   i=0;
UINT8   k=0;
UINT16  tx_count=0;
UINT8 store[4];

//Inicio del Radio
WirelessUSBLS_Start();
WirelessUSBLS_SetChannel(channel);
WirelessUSBLS_SetPnCode(&pn_code_table[code]);
WirelessUSBLS_SetTxPowerLevel(power);

//Para 32 chips
//WirelessUSBLS_WriteRegister(REG_DATA_RATE,0x05);

//Inicio de LCD
LCD_Start();
LCD_Position(0,0);
LCD_PrCString("Ch: ");
itoa((char *)&store, (channel+2402), 10);
LCD_PrString((char *)&store);
LCD_Position(1,0);
LCD_PrCString("Power: ");
itoa((char *)&store, power, 10);
LCD_PrString((char *)&store);

while(1)
{
//Radio Transmite 100 Veces el Vector al presionar el boton
if (button_pressed())
{
for(k=0; k<100; k++)
{
```

```
for(i=0; i<3; i++)
{
tx_frame.preamble[i]=frame_preamble[i];
}

tx_frame.synchro=0xC7;
tx_frame.length=size;

for(i=0; i<size; i++)
{
tx_frame.payload[i]=data[i];
}

tx_frame.checksum=calc_checksum((UINT8 *)&data);

tx_frame.finished=0xFF;

LCD_Position(0,14);
LCD_PrHexByte(tx_frame.checksum);

WirelessUSBLS_SendData(size+7, (UINT8 *)&tx_frame);

Delay50uTimes(2000);
tx_count++;
LCD_Position(1,13);
itoa((char *)&store, tx_count, 10);
LCD_PrString((char *)&store);
}
}
}
}

//*****************************
// Cálculo de Checksum
//*****************************
UINT8 calc_checksum(UINT8 *data)
{
int n;
UINT8 checksum = seed;

for(n=0; n<size; n++)
{
checksum ^= data[n];
}
return(checksum);
}

//*****************************
// Detección de Botón
//*****************************
UINT8 button_pressed(void)
{
UINT8  port_shadow;

port_shadow = SW_Data_ADDR;
if(port_shadow &= SW_MASK)
{
if (button_up == FALSE)
{
return 0;
```

```
}
else
{
button_up = FALSE;
return 1;
}
}
else
{
button_up = TRUE;
return 0;
}
}
```

En módulo receptor se implementó una rutina para que el dispositivo se encuentre constantemente en modo de escucha, de tal forma que se encuentre en capacidades de recibir la información en el momento en que se realice la transmisión de la información. Como se mencionó anteriormente, los datos transmitidos se almacenan la memoria del microcontrolador, con el objetivo de realizar la comparación bit a bit entre la información almacenada y la información recibida, para finalmente calcular la cantidad de bits errados que llegan. Cabe aclarar que en el modulo receptor se configura la misma secuencia pseudo-aleatoria y frecuencia de operación que en el modulo transmisor para garantizar la comunicación del sistema. El siguiente archivo muestras las líneas de código empleadas en el modulo receptor PRoC.

```
//-------------------------------------------------------------------- //-------------
-------------------------------------------------------------
// C main line
//--------------------------------------------------------------------------------

//#include <m8c.h>        // part specific constants and macros
//#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include "cypdef.h"
#include "psocgpioint.h"
#include "WirelessUSBLS.h"
#include "LCD.h"
#include "LED_1.h"
#include "stdlib.h"
#include "delay.h"
#include "protocol.h"

// Definicion de Funciones
UINT8 check_error(UINT8);
UINT8 calc_checksum(UINT8);
UINT8 verificar(UINT8);

// Tabla de Códigos PN
const UINT8 pn_code_table[] =
{
    0x6A, 0xE7, 0x01, 0xEA, 0x03, 0xFD, 0x13, 0xD2, // Código 1
    0xDC, 0xC0, 0x6B, 0xB8, 0x2B, 0x09, 0xBB, 0xB2, // Código 2
    0xA3, 0x1E, 0xF2, 0xA4, 0x31, 0x32, 0x7A, 0xB3, // Código 3
    0x44, 0x83, 0x3B, 0xDD, 0x14, 0xCF, 0x8E, 0xC9, // Código 4
    0x35, 0x35, 0x4E, 0xC5, 0xF3, 0x52, 0x47, 0xB0, // Código 5
    0x7C, 0x23, 0x8A, 0xCE, 0x45, 0x5C, 0x54, 0xD7, // Código 6
    0x81, 0xAC, 0xFB, 0x83, 0x7A, 0x9A, 0x61, 0xAC, // Código 7
    0x3C, 0x12, 0x5F, 0x9C, 0x39, 0x98, 0xF6, 0x8A, // Código 8
};
```

```
// Vector de Bytes a Transmitir
const UINT8 data[] =
 {
       0xAE, 0x35, 0x6B, 0xB5, 0x38,
       0xA9, 0x1C, 0x48, 0x25, 0x1C,
       0x91, 0x5F, 0xA4, 0x55, 0xF7,
       0xB6, 0x3E, 0x8B, 0x5D, 0xC2,
       0x4E, 0x05, 0xF2, 0xDA, 0x23,
};

// Canal
//UINT8   channel = 76;
UINT8   channel = 60;
// Secuencia Pseudoaleatoria
UINT8   code = 0;
// Semilla de Checksum
UINT8 seed = 0x5E;

#define size 32
#define RX_TIMEOUT 1000

UINT8   rx_buf[size];
UINT8   valid_buf[size];
UINT8   value;
UINT8   errors;
UINT8   ones;
UINT8    bits_ok;
UINT8   store[10];
UINT8   i;
UINT8   j;
BOOL   antes;
BOOL   actual;

void main(void)
{
    // Definición de Variables
BOOL    flag_frame_errors = 0;
UINT8   counter = 0;
UINT8   length = 0;
UINT8   fr_lost = 0;
UINT8   shift = 0;
UINT8   k = 0;
UINT8   m = 0;
UINT8   correr = 0;
UINT16  frame_rx = 0;
UINT16  bit_errors = 0;
UINT16  byte_errors = 0;
UINT16  frame_errors = 0;
UINT16  errors = 0;

// Inicio de LCD
LCD_Start();
LCD_Position(0,0);
LCD_PrCString("FRx");
LCD_Position(0,7);
LCD_PrCString("Er");
LCD_Position(1,0);
LCD_PrCString("ByE");
LCD_Position(1,7);
```

```
LCD_PrCString("BiE");

// Inicio del Radio

WirelessUSBLS_Start();
WirelessUSBLS_SetChannel(channel);
WirelessUSBLS_SetPnCode(&pn_code_table[code]);

//Para 32 chips
//WirelessUSBLS_WriteRegister(REG_DATA_RATE,0x05);

while(1)
{
// Radio en Constante Escucha
length = WirelessUSBLS_bReadData(size, (UINT8 *)&rx_buf, (UINT8 *)&valid_buf,
RX_TIMEOUT);

for(j = 0; j < length; j ++)
{
LCD_Position(0,k);
LCD_PrCString("  ");
LCD_Position(0,k);
LCD_PrHexByte(rx_buf[j]);
k=k+2;
if(k==16)
{
k=0;
break;
}
}

if(length > 1 && length < 33)
{
bits_ok = 0;
antes = 0;
actual = 0;

for(i = 0; i < 5; i ++)
{
shift = verificar(rx_buf[i]);
//0xBB se recibe cuando el preambulo va bien
if(shift != 0xBB)
{
//Se detectó el fin del preambulo
if((bits_ok > 7 && bits_ok < 26))
{
if(shift != 0)
{
//Desplazamiento de los bits
for(correr = i-1; correr < length; correr ++)
{
rx_buf[correr] <<= shift;
if(correr != length-1)
{
rx_buf[correr] |= (rx_buf[correr+1]>>(8-shift)) & (0xFF>>(8-shift));
}
}
}

if(shift == 7)
```

```
{
i++;
}

//Hay sincronismo
LCD_Position(0,14);
LCD_PrHexByte(rx_buf[i]);
if(check_error(rx_buf[i]^0xC7) < 3)
{
if((check_error(rx_buf[length-1] ^ 0xFF)) < 2)
{
length = rx_buf[i+1];
frame_rx++;
k=0;

//Desplazar los bytes utiles al inicio
for(correr = (i+2); correr < (i+2)+(length+1); correr++)
{
rx_buf[k] = rx_buf[correr];
k++;
}

for(correr = length+1; correr < length+10; correr++)
{
rx_buf[correr] = 0x00;
}

//Verificar el campo Checksum con el Checksum de Payload
if(rx_buf[length] != calc_checksum(length))
{
//Hay errores en los datos
for(m = 0; m < length; m ++)
{
value = (rx_buf[m] ^ data[m]);

if(value != 0)
{
flag_frame_errors=1;
byte_errors++;
bit_errors += check_error(value);
}
}

if(flag_frame_errors == 1)
{
frame_errors++;
}
}
}
}

else
{
//Se descarta la trama, se toma como perdida
fr_lost++;
}

i=11;
}
bits_ok = 0;
```

```
}
}

LCD_Position(0,4);
itoa((char *)&store, frame_rx, 10);
LCD_PrString((char *)&store);
LCD_Position(0,10);
itoa((char *)&store, frame_errors, 10);
LCD_PrString((char *)&store);
LCD_Position(1,4);
itoa((char *)&store, byte_errors, 10);
LCD_PrString((char *)&store);
LCD_Position(1,12);
itoa((char *)&store, bit_errors , 10);
LCD_PrString((char *)&store);

length = 0;
flag_frame_errors=0;

}
}
}

//*****************************
// Verificación de Errores de Bit
//*****************************
UINT8 check_error(UINT8 xor)
{
errors = 0;

for(j = 0; j < 8; j++)
{
if(((xor << j) & 0x80) != 0x00)
{
errors++;
}
}
return errors;
}

//*****************************
// Cálculo de Checksum
//*****************************
UINT8 calc_checksum(UINT8 tamano)
{
int n;
UINT8 checksum = seed;

for(n = 0; n < tamano; n++)
{
checksum ^= rx_buf[n];
}
return(checksum);
}

//*****************************
// Verificacion de Preambulo
//*****************************
UINT8 verificar(UINT8 secuencia)
{
```

```
UINT8 a;
for(a = 0; a < 8; a ++)
{
actual = ((secuencia << a) & 0x80);

if(antes != actual)
{
bits_ok ++;
}

if((antes == actual) && bits_ok > 8)
{
if(a == 0)
{
a = 8;
}

return a-1;
}

antes=actual;
}
return 0xBB;
}
```

D.3. LECTURA DE RSSI

La rutina de código implementada para el cálculo del valor de RSSI se implementa en el modulo receptor PRoC, y consiste en la lectura del registro REG_RSSI conformado por 5 bits, que permiten obtener valores enteros entre 0 y 31. El archivo de configuración se muestra a continuación.

```
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------

//#include <m8c.h>        // part specific constants and macros
//#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include "cypdef.h"
#include "psocgpioint.h"
#include "WirelessUSBLS.h"
#include "LCD.h"
#include "LED_1.h"
#include "stdlib.h"
#include "delay.h"

// Tabla de Códigos PN
const UINT8 pn_code_table[] =
{
    0x6A, 0xE7, 0x01, 0xEA, 0x03, 0xFD, 0x13, 0xD2, // Código 1
    0xDC, 0xC0, 0x6B, 0xB8, 0x2B, 0x09, 0xBB, 0xB2, // Código 2
    0xA3, 0x1E, 0xF2, 0xA4, 0x31, 0x32, 0x7A, 0xB3, // Código 3
    0x44, 0x83, 0x3B, 0xDD, 0x14, 0xCF, 0x8E, 0xC9, // Código 4
    0x35, 0x35, 0x4E, 0xC5, 0xF3, 0x52, 0x47, 0xB0, // Código 5
    0x7C, 0x23, 0x8A, 0xCE, 0x45, 0x5C, 0x54, 0xD7, // Código 6
    0x81, 0xAC, 0xFB, 0x83, 0x7A, 0x9A, 0x61, 0xAC, // Código 7
    0x3C, 0x12, 0x5F, 0x9C, 0x39, 0x98, 0xF6, 0x8A, // Código 8
};
```

```
// Canal
//UINT8   channel = 76;
UINT8   channel = 60;
// Secuencia Pseudoaleatoria
UINT8   code = 0;

#define RX_TIMEOUT 100
#define size 3

UINT8   rx_buf[size];
UINT8   valid_buf[size];
UINT8   value;

void main(void)
{
// Definición de Variables
UINT8   counter = 0;
UINT8   length = 0;
UINT8   rssi = 0;
UINT8   store[10];

LCD_Start();
LCD_Position(0,0);
LCD_PrCString("RSSI:");

// Inicio del Radio
WirelessUSBLS_Start();
WirelessUSBLS_SetChannel(channel);
WirelessUSBLS_SetPnCode(&pn_code_table[code]);

while(1)
{
// Radio en Constante Escucha
length  =  WirelessUSBLS_bReadData(size,  (UINT8  *)&rx_buf,  (UINT8  *)&valid_buf,
RX_TIMEOUT);
WirelessUSBLS_WriteRegister(REG_CARRIER_DETECT,0x80);
Delay50u();
rssi = WirelessUSBLS_bGetRssi();
WirelessUSBLS_WriteRegister(REG_CARRIER_DETECT, 0);
LCD_Position(1,0);
LCD_PrHexByte(WirelessUSBLS_bReadRegister(REG_THRESHOLD_LO));
LCD_Position(1,8);
LCD_PrHexByte(WirelessUSBLS_bReadRegister(REG_THRESHOLD_HI));

if(rssi!= 0x00)
{
LCD_Position(0,6);
LCD_PrCString("   ");
LCD_Position(0,6);
itoa((char *)&store, rssi, 10);
LCD_PrString((char *)&store);
LCD_Position(1,0);
LCD_PrHexByte(WirelessUSBLS_bReadRegister(REG_DATA_RATE));
Delay10msTimes(19);
}
}
}
```