Design of electronic instruments using reconfigurable logic blocks and free tools

Miguel Risco-Castillo
Universidad Nacional de Ingeniería, Facultad de Ciencias, Perú, mriscoc@uni.pe

Abstract– ***This paper presents the development and use of free libraries and software tools that can be employed to construct different basic electronics laboratory instruments, as an oscilloscope, wave generator, data recorder, custom instruments, etc. using the flexibility and power of an advance reconfigurable logic device as a field programmable gate array (FPGA). The libraries and tools have a modular design approach so that it is easily adaptable to different kinds and manufacturers of hardware and FPGA, the PC open source software and the hardware description language (HDL) used to code the FPGA are easily extensible to add functionalities or applications. The use of the SBA architecture allows shortening development time and as low resources usage.***

Keywords- Virtual Instrument, Reconfigurable System, FPGA, VHDL, SBA

Digital Object Identifier (DOI):http://dx.doi.org/10.18687/LACCEI2018.1.1.374
ISBN: 978-0-9993443-1-6
ISSN: 2414-6390

Diseño de instrumentos electrónicos mediante bloques de lógica reconfigurable y herramientas libres

Miguel Risco-Castillo
Universidad Nacional de Ingeniería, Facultad de Ciencias, Perú, mriscoc@uni.pe

***Resumen*– Este artículo presenta el desarrollo y uso de librerías y herramientas de software que pueden emplearse para construir diferentes instrumentos electrónicos básicos, como osciloscopios, generador de ondas, registrador de datos, instrumentos personalizados, etc. utilizando la flexibilidad y el poder de un dispositivo avanzado de lógica reconfigurable (FPGA). Las bibliotecas y herramientas tienen un enfoque de diseño modular para que sea fácilmente adaptable a diferentes fabricantes de hardware y tipos de FPGA, El software de código abierto de PC y el lenguaje de descripción de hardware (HDL) utilizado para configurar el FPGA son fácilmente extensibles para agregar funcionalidades o aplicaciones. El uso de la arquitectura SBA permite tiempos cortos de desarrollo y un bajo consumo de recursos.**

Palabras Clave* -- *Instrumentos virtuales, sistemas reconfigurables, FPGA, VHDL, SBA.

Abstract*– *This paper presents the development and use of free libraries and software tools that can be employed to construct different basic electronics laboratory instruments, as an oscilloscope, wave generator, data recorder, custom instruments, etc. using the flexibility and power of an advance reconfigurable logic device as a field programmable gate array (FPGA). The libraries and tools have a modular design approach so that it is easily adaptable to different kinds and manufacturers of hardware and FPGA, the PC open source software and the hardware description language (HDL) used to code the FPGA are easily extensible to add functionalities or applications. The use of the SBA architecture allows shortening development time and as low resources usage.

Keywords*-- *Virtual Instrument, Reconfigurable System, FPGA, VHDL, SBA.

I. Introducción

El desarrollo de las ciencias experimentales y la ingeniería se beneficia de la capacidad de obtener datos confiables de situaciones y procesos controlados, como mediciones y comparaciones. Las herramientas que nos permiten obtener esta información son los instrumentos [1].

Un equipo que permite medir o adquirir información para un experimento o trabajo específico, es un requisito permanente en universidades y laboratorios de investigación, las herramientas convencionales de bajo costo no siempre permiten una personalización adecuada, y solo los equipos más costosos permiten la programación, configuración o personalización. Por lo tanto, a menudo, los investigadores de centros de bajos recursos no cuentan con las herramientas adecuadas y necesitan hacerse pequeños sistemas electrónicos de aplicación específica. La mayoría de las veces, el desarrollo de hardware y software para tales tipos de dispositivos no se comparte y tampoco se suelen volver a usar.

El desarrollo de un instrumento personalizado es una tarea que requiere mucho tiempo y recursos, no solo en el diseño y construcción de estos, sino en el desarrollo de firmware y software de soporte que, a menudo, es una tarea de mayor complejidad que distrae al investigador de objetivos reales. Los instrumentos virtuales o sintéticos [2] son una buena alternativa porque son más baratos que los instrumentos convencionales y también más flexibles.

El concepto de instrumentación virtual [3] no es nuevo, el aumento de la disponibilidad y el poder computacional de la computadora personal (PC) y también la disminución de sus costos, permiten trasladar algunos aspectos del instrumento convencional al software en una PC. La interactividad y el cálculo de datos más complejos de los instrumentos son migrados a la computadora. Haciendo posible el uso de hardware genérico o más básico, y por lo tanto más económico.

Con el incremento exponencial en la complejidad de los dispositivos lógicos reconfigurables, el arreglo de puertas programables en campo, o FPGA por sus siglas en inglés, se convierte en un elemento atractivo para la construcción de instrumentos reconfigurables, debido a su inherente flexibilidad [4].

Antecedentes y trabajos previos

En el mercado actualmente existen diversos instrumentos virtuales comerciales, por ejemplo, National Instruments [5] tiene equipos y tableros integrados con su software LabView. Pico Technology [6] tiene un conjunto de osciloscopios para PC y sistemas de adquisición de datos basados en instrumentación virtual (ver Fig. 1).

Fig. 1 Ejemplo del instrumento virtual desarrollado por Pico Technology.

Digital Object Identifier (DOI): http://dx.doi.org/10.18687/LACCEI2018.1.1.374
ISBN: 978-0-9993443-1-6
ISSN: 2414-6390

TiePie Engineering [7] tiene muchos dispositivos especializados como osciloscopios y generadores arbitrarios de funciones (ver Fig. 2), también proporcionan varias librerías multiplataforma pre-compiladas para software de terceros. BitScope [8] tiene diversos productos de instrumentación virtual con software avanzado.

Fig. 2 Instrumento virtual desarrollado por TiePie Engineering.

La mayoría de los equipos comerciales, son de hardware y software propietarios y cerrados, de modo que el usuario generalmente no puede extenderlos o personalizarlos.

Esfuerzos comunitarios para proporcionar hardware abierto y alternativas de software pueden encontrarse fácilmente en sitios de financiación colectiva (crowdfunding), como por ejemplo la iniciativa Omniboard en Kickstarter [9] (ver Fig. 3) o el proyecto Red Pitaya [10] que posee un hardware interesante (ver Fig. 4) y moderno con múltiples capacidades, y software de instrumentos básicos en desarrollo para esta plataforma.

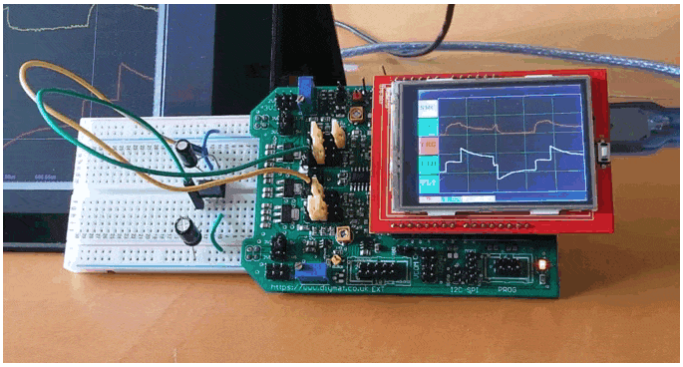


Fig. 3 Omniboard en Kickstarter.

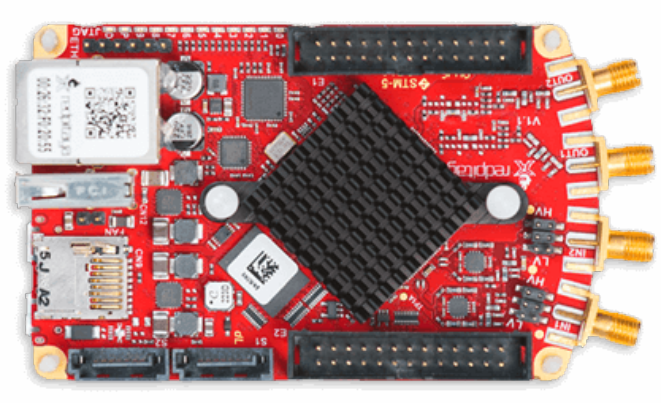


Fig. 4 Tarjeta desarrollada por el proyecto Red Pitaya.

Casi la totalidad de las propuestas de instrumentación electrónica antes mencionadas están basadas en un hardware en particular y el software es altamente optimizado para dispositivos específicos, lo cual limita la capacidad de aplicar lo desarrollado a módulos de hardware más genéricos.

Existen tarjetas electrónicas genéricas con sistemas embebidos basados en procesadores ARM o FPGA y herramientas que se pueden utilizar para construir algún tipo de instrumentación virtual en diferentes niveles. Estas son provistas por diferentes fabricantes (Xilinx, Altera, Microsemi, Lattice, etc.). Digilent [11], Maxim Integrated [12] y Analog Devices (Linear Technology) [13] venden varios módulos periféricos para expandir fácilmente las capacidades de esas tarjetas, proporcionando el hardware necesario para la adquisición y generación de señales.

Descripción y Objetivos

Este documento presenta herramientas y librerías desarrolladas para la construcción de algunos instrumentos básicos como, un generador arbitrario de formas de onda y un osciloscopio, implementados en una placa con FPGA utilizando herramientas gratuitas.

En el diseño de los distintos bloques funcionales del hardware y del software, se ha considerado la implementación en hardware genérico, por lo cual, no se hace uso de las librerías y núcleos especializados proporcionados por los fabricantes de FPGA.

Si bien este enfoque genérico limita la optimización, permite que los diseños posean una alta portabilidad entre distintas placas de desarrollo, amplía sus capacidades de mejora y diversifica sus aplicaciones. En especial se presenta el uso de la arquitectura de bus simple (SBA) [14], la cual se basa en el estándar abierto de interconexión Wishbone [15] y herramientas asociadas que facilitan en gran medida la implementación en el FPGA.

II. Materiales y Métodos

Para el diseño de los bloques o núcleos de propiedad intelectual (IP Cores) se eligió el lenguaje VHDL [16] (combinación de los acrónimos VHSIC "Very High Speed Integrated Circuit" y HDL "Hardware Description Lenguage"). Este lenguaje se usa como base en la herramienta SBACreator [17] que permite el diseño de arquitecturas en FPGA basadas en SBA. El SBACreator es una herramienta gratuita que permite también acceder a una librería de "IP Cores" entre las que se encuentran núcleos de comunicación y para el manejo de Conversores Analógico-Digital y Digital-Analógico.

Para el interfaz gráfico de usuario (GUI por sus siglas en inglés) se eligió el entorno integrado de desarrollo (IDE) Lazarus [18], una herramienta gratuita y libre de desarrollo rápido de aplicaciones basada en el lenguaje de programación Object Pascal, disponible para diversos sistemas operativos como Windows, GNU/Linux y Mac OS X.

Se toma como ejemplo el desarrollo de un instrumento personalizado que tiene la capacidad de generar formas de ondas (Generador de funciones) de baja frecuencia (en el rango de las señales de audio) y un osciloscopio básico, todas las funcionalidades integradas en un solo GUI. Este instrumento simple puede ser usado para demostrar el funcionamiento de un filtro (ver Fig. 5). Las especificaciones generales del instrumento se muestran en la Tabla I.

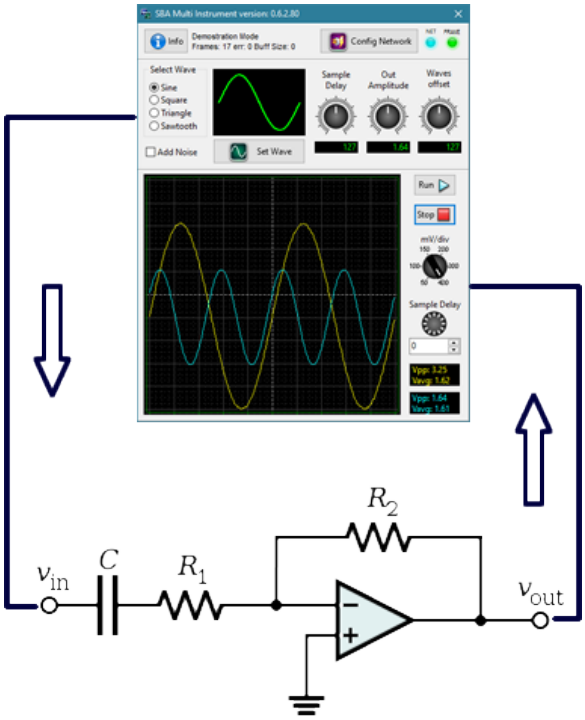


Fig. 5 Imagen descriptiva del uso del instrumento.

TABLA I
ESPECIFICACIONES GENERALES DEL INSTRUMENTO

Generador de funciones	
Formas de onda	Senoidal, Cuadrada, Triangular y Diente de sierra
Resolución	8 bits
Voltaje de salida	0 a 3.3V
Rango de frecuencias	20 a 20KHz
Osciloscopio básico	
Resolución vertical	8 bits
Canales	2 canales
Buffer por canal	256 muestras
Voltaje de entrada	0 a 3.3V
Ancho de banda	20 a 20KHz

La base del hardware del instrumento ha sido implementada en dos tarjetas con FPGA diferentes de los actualmente principales fabricantes de estos dispositivos, la tarjeta Nexys2 (ver Fig. 6) que cuenta con una FPGA Xilinx Spartan-3E FPGA 500K y la tarjeta DE0 (ver Fig. 7) que posee una FPGA Intel/Altera Cyclone III EP3C16F484.

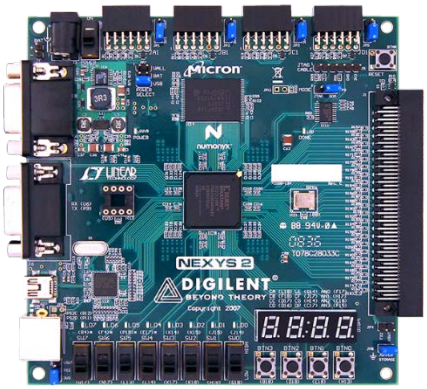


Fig. 6 Tarjeta Nexys2 fabricada por Digilent.

Fig. 7 Tarjeta DE0 fabricada por TerasIC.

Para la etapa de generación y adquisición de señales analógicas se utilizaron los módulos de Digilent PMODAD1 (ver Fig. 8) y PMODDA1 (ver Fig. 9), conversores analógico-digital y digital-analógico respectivamente.

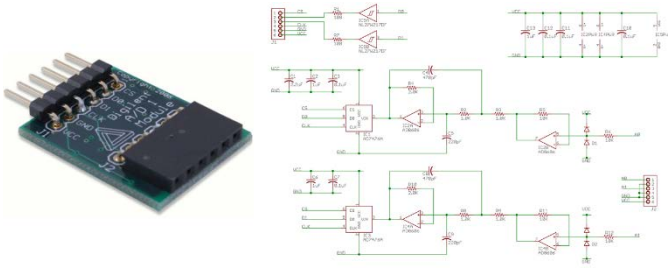


Fig. 8 Módulo Conversor Digital-Analógico PMOD AD1.

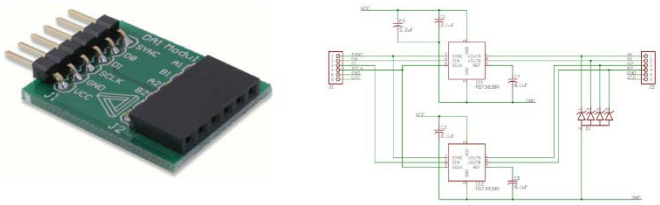


Fig. 9 Módulo Conversor Analógico-Digital PMOD DA1.

La librería SBA tiene disponibles los núcleos para manejar ambos módulos.

Para la descripción del hardware del instrumento en VHDL se utilizaron 3 proyectos SBA (un módulo maestro y dos esclavos), un proyecto describe el comportamiento del generador de funciones e implementa una memoria de doble puerto, de dónde se obtienen las muestras que son enviadas al conversor digital-analógico a una velocidad establecida por parámetros grabados en un registro de configuración. El segundo proyecto, describe el funcionamiento de un osciloscopio básico, las muestras son obtenidas del conversor analógico-digital a una velocidad establecida por un registro de configuración y grabadas en una memoria de doble puerto.

El tercer y último proyecto, describe el controlador maestro y etapa de comunicación, que se encarga de establecer la comunicación con el GUI en la PC. Escribe los parámetros de funcionamiento en los registros de configuración, graba y lee en las memorias de doble puerto de los proyectos anteriores.

Del mismo modo, para la elaboración del software, se realizó una separación modular por funciones, un módulo principal encargado de realizar las operaciones de gestión y comunicación de los datos, un módulo encargado de generar las muestras según la forma de onda y parámetros escogidos por el usuario, y finalmente un módulo de visualización que muestra mediante una gráfica, el valor de la amplitud de las muestras adquiridas por el instrumento respecto al tiempo en forma continua. En la Fig. 10 se muestra una representación del diagrama de bloques del instrumento. El software también se encarga de realizar algunos cálculos menores como los valores promedio, máximos y mínimos.

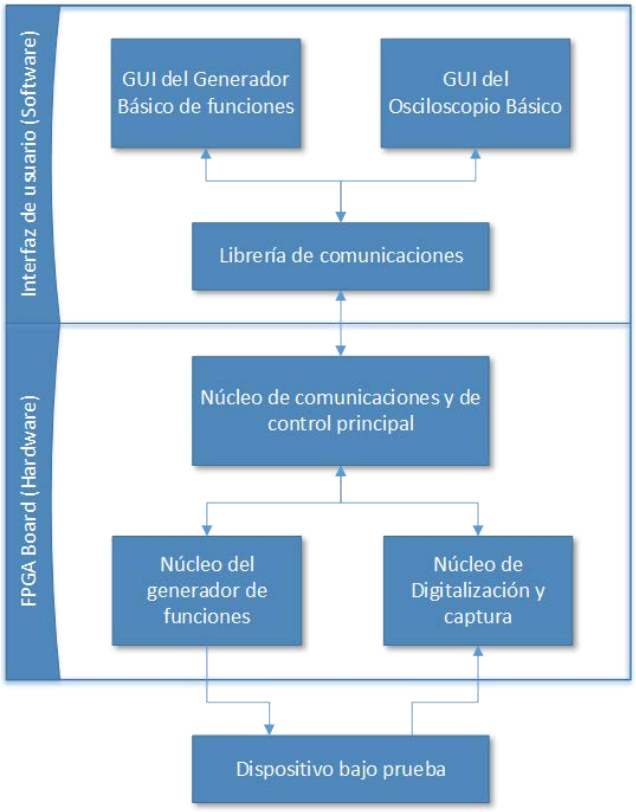


Fig. 10 Diagrama de bloques de la arquitectura del instrumento.

III. Resultados

En la Tabla II se puede apreciar los resultados de la síntesis del hardware para la tarjeta Digilent Nexys2, más precisamente para la FPGA Xilinx Spartan-3E.

En la Tabla III se muestran los resultados de las síntesis para el caso de la tarjeta TerasIC DE0, que cuenta con una FPGA Altera/Intel Cyclone III EP3C16F484. En ambos casos el consumo de recursos porcentual es bajo, lo cual es una característica de haber implementado la arquitectura en SBA.

TABLA II
CONSUMO DE RECURSOS DEL INSTRUMENTO EN UNA FPGA XILINX SPARTAN-3E

Celdas de núcleo	929 de 8672	10%
Flip flops / registros	741 de 17344	4%
Bloques de memoria	3 de 28	11%

TABLA III
CONSUMO DE RECURSOS DEL INSTRUMENTO EN UNA FPGA ALTERA/INTEL CYCLONE III EP3C16F484

Elementos lógicos	1674 de 15408	11%
Flip flops / registros	813 de 15408	5%
Bloques de memoria	3 de 56	5%

En el apéndice A, se muestra el diagrama RTL esquemático del instrumento, en el apéndice B el uso aproximado de los recursos en una FPGA Xilinx Spartan-3E. En el apéndice C, se encuentra el código VHDL de la entidad principal que corresponde al proyecto maestro.

En la Fig. 11 se muestra el interfaz de usuario del instrumento tal y como se visualiza en el sistema operativo Windows 10, aunque gracias a Lazarus es posible compilar el mismo GUI para sistemas operativos diferentes como Linux y OSX.

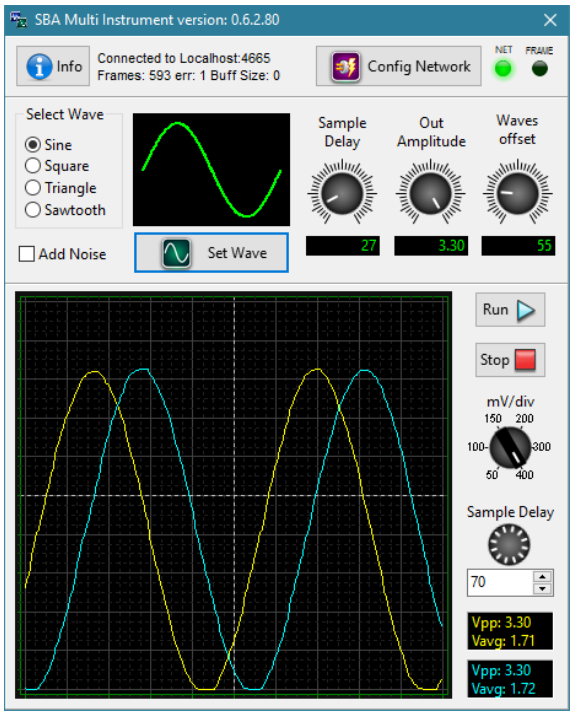


Fig. 11 Interfaz de usuario del instrumento.

IV. CONCLUSIONES

Usar el FPGA para el desarrollo de un instrumento permite que el hardware sea reconfigurado fácilmente y sea lo suficientemente flexible para permitir la adición y mejora de sus funciones.

Gracias al uso de la arquitectura de bus simple (SBA), el tiempo de desarrollo se reduce considerablemente y dado que el código para el FPGA se describe sólo en VHDL, permite que los instrumentos desarrollados se puedan transferir fácilmente a diversas plataformas. Los costos de desarrollo son relativamente bajos debido a que no se utilizan núcleos IP comerciales ni es necesario adquirir licencias, lo que podría beneficiar a las universidades e instituciones de investigación que no cuentan con muchos recursos financieros.

Con tan solo un 10% de los recursos consumidos por el instrumento en el FPGA de Xilinx, muestra una gran capacidad de crecimiento, no solo aumentando las capacidades del dispositivo en sí, sino haciendo posible que el instrumento coexista con otros implementados al mismo tiempo en el FPGA.

La flexibilidad proporcionada por el software en la computadora personal para crear instrumentos virtuales permite la generación de nuevas herramientas cada vez más potentes. El uso de software libre y multiplataforma facilita el intercambio y reutilización de las aplicaciones y librerías desarrolladas.

V. REFERENCIAS

1. **Bakshi, U. A., Bakshi, A. V. and Bakshi, K. A.** *Electrical Measurements and Instrumentation.* Pune, India : Technical Publications Pune, 2009.

2. **Nadovich, Chris.** *Synthetic Instruments: Concepts and Applications.* s.l. : Elsevier, 2005.

3. **National Instruments.** *Virtual Instrumentation.* [Online] 2013. http://www.ni.com/white-paper/4752/en/.

4. *A Virtual Instrumentation Laboratory Based on a Reconfigurable Coprocessor.* **Quintáns, Camilo, y otros.** 2006. IEEE Transactions on Insrumentation and Measurement, vol. 55, no. 2. págs. 635-645.

5. **National Instruments.** *National Instruments home page.* [Online] http://www.ni.com.

6. **Pico Technology.** *Pico Technology - PC Oscilloscope & Data Acquisition Products.* [Online] http://www.picotech.com/.

7. **TiePie Engineering.** *Osciloscopios USB y generadores de forma de onda arbitraria.* [Online] https://www.tiepie.com/es.

8. **BitScope.** Test, measurement & data acquisition. [En línea] http://www.bitscope.com/.

9. **Omniboard.** Oscilloscope, generator, debugger, multimeter and much more. [En línea] https://www.kickstarter.com/projects/1673888854/oscilloscope-generator-debugger-multimeter-and-muc.

10. **Red Pitaya.** Red Pitaya home page. *Red Pitaya home page.* [Online] https://www.redpitaya.com/.

11. **Digilent Inc.** Peripheral Modules (Pmods™). *Flexible peripheral modules for all your designs.* [Online] https://store.digilentinc.com/pmod-modules/.

12. **Maxim Integrated.** Maxim Pmod-Compatible Plug-In Peripheral Modules. *Maxim Pmod-Compatible Plug-In Peripheral Modules.* [Online] https://www.maximintegrated.com/en/design/partners-and-technology/design-technology/fpga-design-resources/pmod-compatible-plug-in-peripheral-modules.html.

13. **Linear Technology.** Linear Technology - Home page. [En línea] http://www.linear.com/.

14. **Simple Bus Architecture.** *SBA Home page.* [En línea] http://sba.accesus.com/.

15. **OpenCores.** SoC Interconnection: WISHBONE. [En línea] https://opencores.org/howto/wishbone.

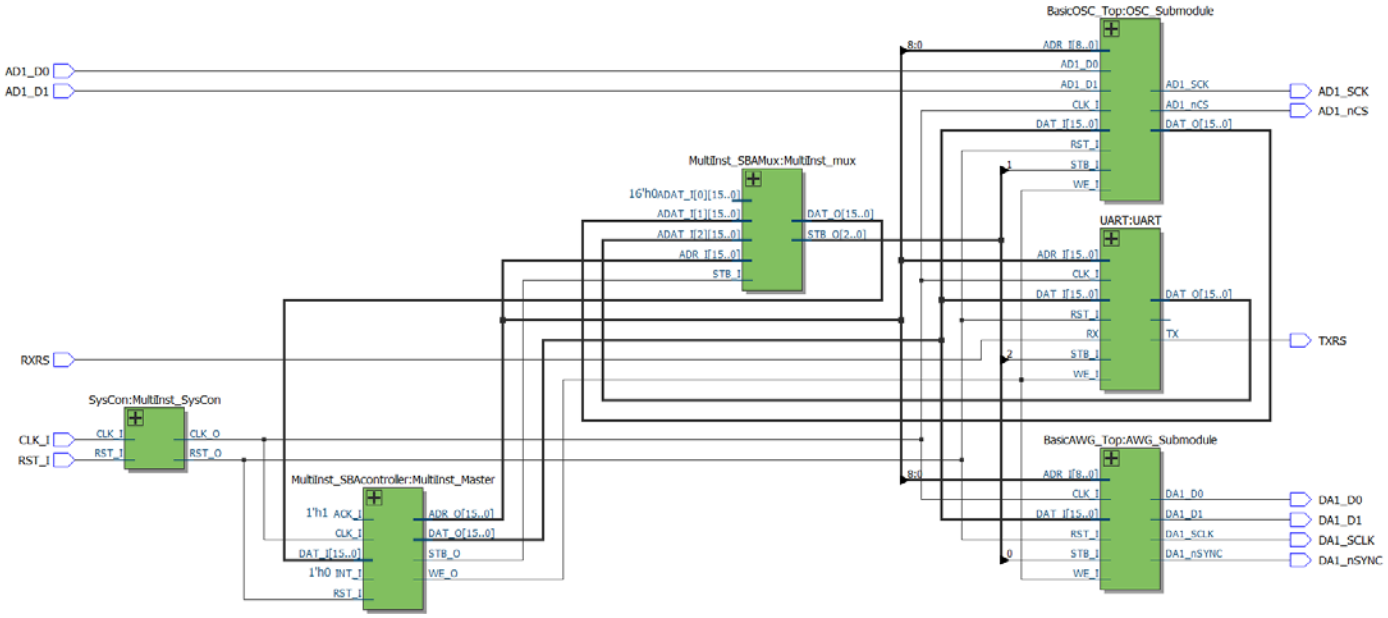
16. **Ashenden, Peter J.** *The Designer's Guide to VHDL.* s.l. : Morgan Kaufmann Publishers, 2008. ISBN: 978-0-12-088785-9.

17. **SBA.** SBA Creator. [En línea] http://sba.accesus.com/software-tools/sba-creator.

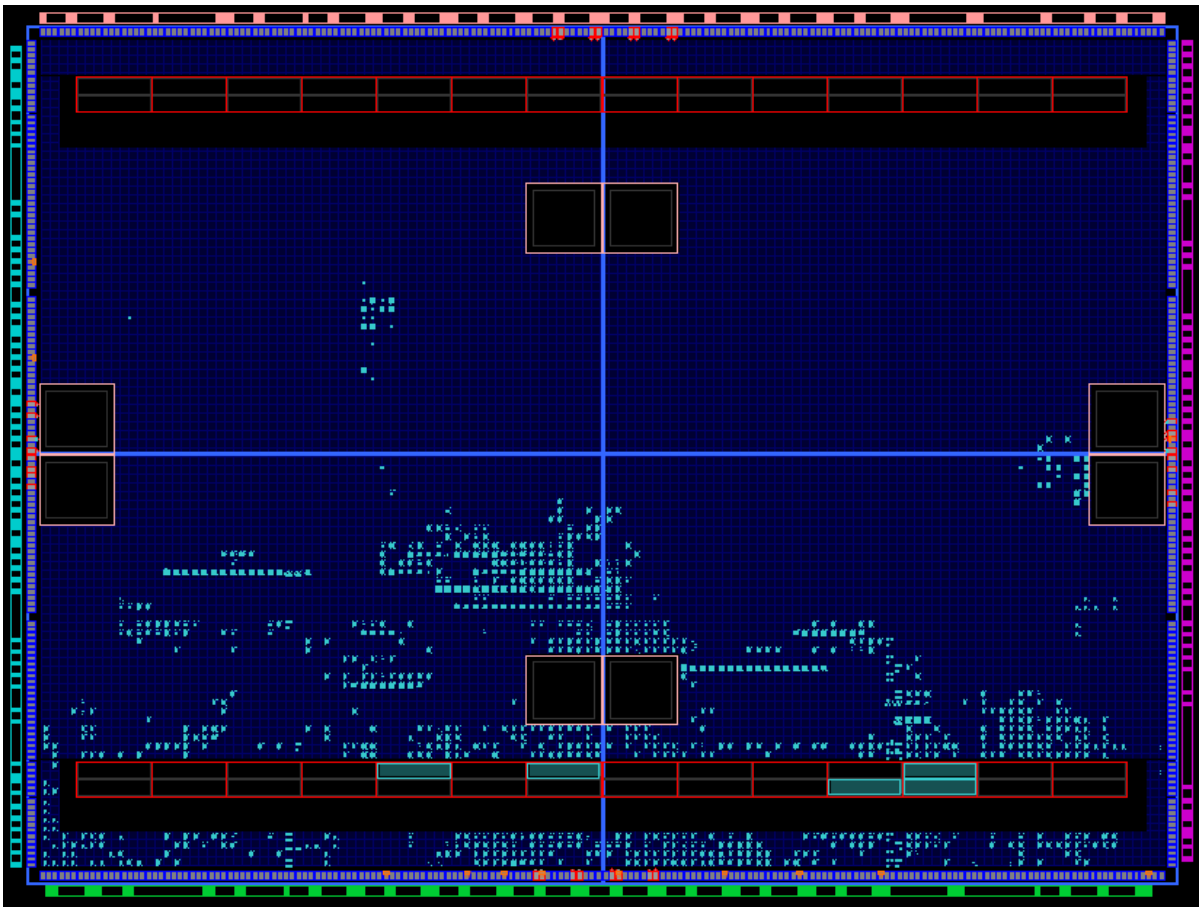
18. **Lazarus RAD IDE.** *Lazarus Homepage.* [Online] http://www.lazarus-ide.org/.

VI. APÉNDICES

Apéndice A: Diagrama esquemático RTL del instrumento de prueba.



Apéndice B: Diagrama de uso aproximado del FPGA Xilinx Spartan-3E, los bloques en color cian muestran las celdas en uso.



Apéndice C: Código VHDL de la entidad principal del proyecto maestro.

```
1.  ----------------------------------------------------------------------------
2.  -- Project Name: MultiInst
3.  -- Title: SBA Multi-Instrument Sample
4.  -- Version: 0.2
5.  -- Date: 2018/03/31
6.  -- Project Author: Miguel A. Risco-Castillo
7.  -- Description: SBA Multi-Instrument Sample, uses BasicAWG and BasicOSC projects,
8.  -- This version does not include the Seven Segment display, to make the design
9.  -- wide compatible with more FPGA boards.
10. ----------------------------------------------------------------------------
11.
12. Library IEEE;
13. use IEEE.std_logic_1164.all;
14. use work.MultiInst_SBAconfig.all;
15.
16. entity MultiInst_Top is
17. port (
18.   CLK_I     : in  std_logic;
19.   RST_I     : in  std_logic;
20.   -- Interface for PMODAD1
21.   AD1_nCS   : out std_logic;
22.   AD1_D0    : in  std_logic;
23.   AD1_D1    : in  std_logic;
24.   AD1_SCK   : out std_logic;
25.   -- Interface for PMODDA1
26.   DA1_nSYNC : out std_logic;
27.   DA1_D0    : out std_logic;
28.   DA1_D1    : out std_logic;
29.   DA1_SCLK  : out std_logic;
30.   -- UART;
31.   RXRS      : in  std_logic;
32.   TXRS      : out std_logic
33. );
34. end MultiInst_Top;
35.
36. ----------------------------------------------------------------------------
37.
38. architecture MultiInst_structural of MultiInst_Top is
39.
40. -- SBA internal signals
41.   Signal RSTi  : Std_Logic;
42.   Signal CLKi  : Std_Logic;
43.   Signal ADRi  : ADDR_type;
44.   Signal DATOi : DATA_type;
45.   Signal DATIi : DATA_type;
46.   Signal ADATi : ADAT_type;
47.   Signal STBEi : std_logic;
48.   Signal STBi  : std_logic_vector(Stb_width-1 downto 0);
49.   Signal WEi   : Std_Logic;
50.   Signal ACKi  : Std_Logic;
51.   Signal INTi  : Std_Logic;
52.
53. -- Auxiliary external to internal signals
54.   Signal CLKe  : std_logic;
55.   Signal RSTe  : std_logic;
56.
57. -- Auxiliary IPCores signals
58.
59. ----------------------------------------------------------------------------
```

```
60.
61. begin
62.
63.   MultiInst_SysCon: entity work.SysCon
64.   port Map(
65.     CLK_I => CLKe,
66.     CLK_O => CLKi,
67.     RST_I => RSTe,
68.     RST_O => RSTi
69.   );
70.
71.   MultiInst_Master: entity work.MultiInst_SBAcontroller
72.   port Map(
73.     RST_I => RSTi,
74.     CLK_I => CLKi,
75.     DAT_I => DATIi,
76.     DAT_O => DATOi,
77.     ADR_O => ADRi,
78.     STB_O => STBEi,
79.     WE_O  => WEi,
80.     ACK_I => ACKi,
81.     INT_I => INTi
82.   );
83.
84.   MultiInst_mux: entity work.MultiInst_SBAMux
85.   port Map(
86.     STB_I => STBEi,
87.     ADR_I => ADRi,
88.     STB_O => STBi,
89.     ADAT_I=> ADATi,
90.     DAT_O => DATIi
91.   );
92.
93.   UART: entity work.UART
94.   generic map(
95.     debug   => debug,
96.     sysfreq => sysfreq,
97.     baud    => 115200,
98.     rxbuff  => 518
99.   )
100.    port map(
101.      -------------
102.      RST_I => RSTi,
103.      CLK_I => CLKi,
104.      STB_I => STBi(STB_UART),
105.      ADR_I => ADRi,
106.      WE_I  => WEi,
107.      DAT_I => DATOi,
108.      DAT_O => ADATi(STB_UART),
109.      -------------
110.      RX    => RXRS,
111.      TX    => TXRS
112.    );
113.
114.    AWG_Submodule: entity work.BasicAWG_Top
115.    port Map(
116.      CLK_I => CLKi,
117.      RST_I => RSTi,
118.      -- RAM and Parameters Write Access;
119.      STB_I => STBi(STB_AWG),
120.      WE_I  => WEi,
```

```
121.       ADR_I => ADRi(8 downto 0),
122.       DAT_I => DATOi,
123.       -- Interface for PMODDA1
124.       DA1_nSYNC => DA1_nSYNC,
125.       DA1_D0    => DA1_D0,
126.       DA1_D1    => DA1_D1,
127.       DA1_SCLK  => DA1_SCLK
128.     );
129.
130.     OSC_Submodule:entity work.BasicOSC_Top
131.     port Map(
132.       CLK_I => CLKi,
133.       RST_I => RSTi,
134.       -- RAM and Parameters Write Access;
135.       STB_I => STBi(STB_OSC),
136.       WE_I  => WEi,
137.       ADR_I => ADRi(8 downto 0),
138.       DAT_I => DATOi,
139.       DAT_O => ADATi(STB_OSC),
140.       -- Interface for PMODAD1
141.       AD1_nCS => AD1_nCS,
142.       AD1_D0  => AD1_D0,
143.       AD1_D1  => AD1_D1,
144.       AD1_SCK => AD1_SCK
145.     );
146.
147.   -- External Signals Assignments
148.   --------------------------------
149.    RSTe  <= RST_I;                -- SBA reset is active high
150.    CLKe  <= CLK_I;
151.
152.   -- Internal Signals Assignments
153.   --------------------------------
154.    ACKi  <= '1';                  -- If None Slave IPCore use ACK then ACKi must be '1'
155.    INTi  <= '0';                  -- No interrupts support;
156.
157.   end MultiInst_structural;
158.
159.   ---------------------------------------------------------------------------------
```