



FPGA LIBRES CON ICESTUDIO

Alhambra II

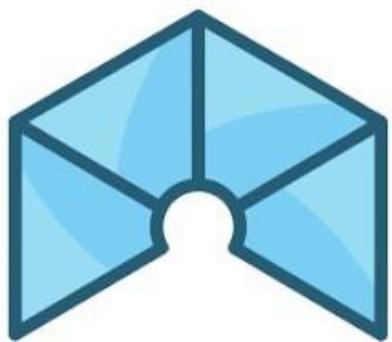


Tabla de contenido

Índice de ilustraciones.....	4
1. FPGA (“Field Programmable Gate Array”)	8
1.1. ¿Qué son las FPGAs?	8
1.2. Proyecto ICESTORM	8
1.3. Comunidad FPGA Wars	9
2. Instalación del Icestudio.....	10
2.1. Descargar el instalador de Icestudio desde GitHub:	10
2.2. Ejecutar el instalador.....	11
2.3. Configurar el idioma	12
2.4. Instalar el toolchain.....	13
2.5. Instalar los drivers	14
2.5.1. Conectar la placa de la FPGA al USB.....	14
2.5.2. Seleccionar el driver	15
3. Software Icestudio	16
3.1. Seleccionar	16
3.2. Herramientas.....	16
4. Placa Alhambra II.....	17
4.1. Pinout Alhambra II.....	18
5. Colecciones en Icestudio	19
6. Circuitos digitales en Icestudio.....	20
6.1. Construcción de circuitos básicos	20
6.2. Trabajo en paralelo.....	21
6.3. Entrada variable	21
7. Pines Externos de Alhambra II.....	23
8. Manipulación de Bits. Puertas lógicas.....	24
8.1. Puerta NOT	24
8.2. Puerta AND/NAND	24
8.3. Puerta OR/NOR.....	24
8.4. Puerta XOR/XNOR	24
9. Bombeo de Bits	25
10. Servos	26
10.1. Introducción a los servos.....	26
10.2. Señal PWM de control.....	26
10.3. Conexión Servo a la Alhambra II.....	27
11. Multiplexor	29
11.1. Multiplexor 2:1	29
11.2. Multiplexor 4:1	31

12. Servos de rotación continua.....	32
13. Sensores de infrarrojos	34
14. Bloques con parámetros	36
15. Circuitos combinacionales.....	37
16. Buses y números	40
16.1. Truncado.....	41
16.2. Nomenclatura.....	42
16.3. Separadores y agregadores de bus	43
17. Creando bloques	44
17.1. Diseño jerárquico	45
17.2. Creando el primer bloque	45
17.3. Bloques con puertos.....	46
17.4. Bloque con entradas de reloj	48
17.5. Bloque con parámetros	49
18. Circuitos combinacionales con varias salidas.....	50
18.1. Circuitos combinacionales con 1 entrada	50
18.2. Circuitos combinacionales con 2 entradas.....	52
18.3. Circuitos combinacionales con 3 entradas.....	52
18.4. Generador de tablas: IceFactory	53
19. La colección	54
20. Display 7 segmentos.....	55
21. Biestables y notificaciones	59
21.1. Biestables Set-Reset	59
22. Tics, tiempo y temporizadores.....	61
22.1. Reloj del sistema.....	61
22.2. Pulso.....	62
22.3. ¿Qué son los Tics?	62
22.4. Temporizadores.....	64
22.5. Generando pulsos periódicos.....	65
23. Contadores	66
23.1. Encadenando contadores.....	67
23.2. Contando tiempo.....	67
23.3. Recorriendo tablas	69
24. Biestables de datos y cambio	70
24.1. Biestable de cambio (T)	70
24.2. Biestables de datos.....	71
24.3. Desplazamiento de Bits	72
24.4. Conversión serie-paralelo.....	73

25. Registros y comparadores	74
25.1. Carga paralela.....	74
25.2. Registros de desplazamiento	75
25.3. Comparadores	76
25.3.1. Comparadores de un operando	77
25.4. Comunicando FPGA con Arduino	78
25.4.1. Escritura.....	79
25.4.2. Lectura	82
26. Puerto serie	85
26.1. Terminal de comunicaciones.....	85
26.3. Transmisor Serie	87
26.4. Receptor Serie	90
26.5. Combinando Transmisor y Receptor	92
26.6. Comunicación Arduino-FPGA	93
26.7. Comunicación Bluetooth-Serie.....	94

Índice de ilustraciones

<i>Ilustración 1. Interior de la FPGA</i>	8
<i>Ilustración 2. Github de FPGAwars</i>	10
<i>Ilustración 3. Opciones de instalación</i>	10
<i>Ilustración 4. Versiones de icesstudio disponibles</i>	11
<i>Ilustración 5. Proceso de instalación de icesstudio</i>	11
<i>Ilustración 6. Instalación completada</i>	12
<i>Ilustración 7. Pantalla de inicio de icesstudio</i>	12
<i>Ilustración 8. Instalación del toolchain</i>	13
<i>Ilustración 9. Instalación finalizada</i>	13
<i>Ilustración 10. Instalación de los drivers</i>	14
<i>Ilustración 11. Pasos a seguir para la instalación</i>	14
<i>Ilustración 12. Aplicación Zadig</i>	15
<i>Ilustración 13. Circuito de ejemplo</i>	15
<i>Ilustración 14. Menú Herramientas</i>	16
<i>Ilustración 15. Alhambra II</i>	17
<i>Ilustración 16. Bit 1</i>	20
<i>Ilustración 17. Bit 1 y Salida</i>	20
<i>Ilustración 18. Unión de Bit 1 y Salida</i>	20
<i>Ilustración 19. Mensaje recibido una vez realizada la carga del programa</i>	21
<i>Ilustración 20. Dos funciones independientes en paralelo</i>	21
<i>Ilustración 21. Esquema de funcionamiento de los pulsadores</i>	21
<i>Ilustración 22. Funciones paralelas independientes con entradas variables</i>	22
<i>Ilustración 23. Esquema de conexión de los LEDs</i>	23
<i>Ilustración 24. Esquema de conexión de los pulsadores</i>	23
<i>Ilustración 25. Conexión de pulsador y LED externos</i>	23
<i>Ilustración 26. Puerta NOT</i>	24
<i>Ilustración 27. Puerta AND</i>	24
<i>Ilustración 28. Puerta NAND</i>	24
<i>Ilustración 29. Puerta OR</i>	24
<i>Ilustración 30. Puerta NOR</i>	24
<i>Ilustración 31. Puerta XOR</i>	24
<i>Ilustración 32. Puerta XNOR</i>	24
<i>Ilustración 33. Bombeo de Bits al LED7</i>	25
<i>Ilustración 34. Bombeo de Bits a varios LEDs al mismo tiempo</i>	25
<i>Ilustración 35. Conexión del servo</i>	26
<i>Ilustración 36. Movimiento del servo</i>	26
<i>Ilustración 37. Señal PWM</i>	26
<i>Ilustración 38. Conexión del servo a la FPGA</i>	27
<i>Ilustración 39. Bloque Servo de icesstudio</i>	27
<i>Ilustración 40. Bloque Dato_A_8Bit</i>	27
<i>Ilustración 41. Bloque ServoDosPosiciones</i>	27
<i>Ilustración 42. Movimiento del servo a dos posiciones</i>	28
<i>Ilustración 43. Movimiento automático del servo</i>	28
<i>Ilustración 44. Función del multiplexor</i>	29
<i>Ilustración 45. Funcionamiento del multiplexor</i>	29
<i>Ilustración 46. Esquema del funcionamiento del multiplexor</i>	29
<i>Ilustración 47. Ejemplo de funcionamiento del multiplexor</i>	30
<i>Ilustración 48. Ejemplo de funcionamiento automático del multiplexor</i>	30
<i>Ilustración 49. Esquema del Mux 4:1</i>	31
<i>Ilustración 50. Ejemplo mux 4-1</i>	31

<i>Ilustración 51. Esquema funcionamiento MotorBit</i>	32
<i>Ilustración 52. Movimientos posibles del servo</i>	32
<i>Ilustración 53. Prueba de funcionamiento de bloque MotorBit</i>	32
<i>Ilustración 54. Programación del movimiento de dos servos en el mismo robot</i>	33
<i>Ilustración 55. Funcionamiento de un sensor IR</i>	34
<i>Ilustración 56. Programación para visualizar estado del sensor mediante un LED</i>	34
<i>Ilustración 57. Programación sigue líneas</i>	34
<i>Ilustración 58. Programación para usar el sensor IR en el movimiento del servo</i>	35
<i>Ilustración 59. Bloques formados por otros bloques</i>	36
<i>Ilustración 60. Programación interna del bloque ServoDosPosiciones</i>	36
<i>Ilustración 61. Tabla de la verdad de un circuito combinacional</i>	37
<i>Ilustración 62. Puerta NOT mediante tabla</i>	37
<i>Ilustración 63. Puerta AND mediante tabla</i>	38
<i>Ilustración 64. Puerta no existente creada mediante tabla</i>	38
<i>Ilustración 65. Multiplexor 2:1 creado mediante tabla de la verdad</i>	39
<i>Ilustración 66. Bus</i>	40
<i>Ilustración 67. Parámetros del bus</i>	40
<i>Ilustración 68. Constante 3 visualizada en 2 LEDs mediante bus de 2 bits</i>	40
<i>Ilustración 69. Constante 178 visualizada en 8 LEDs mediante un bus de 8 bits</i>	41
<i>Ilustración 70. Dato truncado</i>	41
<i>Ilustración 71. Nomenclatura</i>	42
<i>Ilustración 72. Misma función con distinta nomenclatura</i>	42
<i>Ilustración 73. Separador y agregador de 4 bits</i>	43
<i>Ilustración 74. Encriptador de 4 bits</i>	43
<i>Ilustración 75. Bloque comecocos</i>	44
<i>Ilustración 76. Bloque comecocos con pines de entrada y salida conectados</i>	44
<i>Ilustración 77. Jerarquía del bloque ServoDosPosiciones</i>	45
<i>Ilustración 78. Parámetros del bloque</i>	45
<i>Ilustración 79. Bloque creado</i>	45
<i>Ilustración 80. Puerta AND de 3 entradas</i>	46
<i>Ilustración 81. Puertos de entrada y salida añadidos</i>	46
<i>Ilustración 82. Programación interna de puerta AND de 3 entradas</i>	46
<i>Ilustración 83. Información del bloque</i>	46
<i>Ilustración 84. Entradas y salidas conectadas al bloque</i>	47
<i>Ilustración 85. Programación interna del bloque 2?</i>	47
<i>Ilustración 86. Conexiones del bloque 2?</i>	47
<i>Ilustración 87. Bloque corazón con entrada de reloj</i>	48
<i>Ilustración 88. Programación interna de bloque Corazón AND</i>	48
<i>Ilustración 89. Bloque Corazón AND con entradas y salidas conectadas</i>	48
<i>Ilustración 90. Parámetros del bloque ServoDosPosiciones</i>	49
<i>Ilustración 91. Circuito combinacional de 1 entrada con tabla</i>	50
<i>Ilustración 92. Circuito combinacional para control de un servo</i>	50
<i>Ilustración 93. Circuito combinacional de 4 salidas doble</i>	51
<i>Ilustración 94. Circuito combinacional con 8 salidas y multiplexor</i>	51
<i>Ilustración 95. Circuito combinacional con 2 entradas</i>	52
<i>Ilustración 96. Circuito combinacional con 3 entradas</i>	52
<i>Ilustración 97. Portada de IceFactory</i>	53
<i>Ilustración 98. Parámetros de la tabla</i>	53
<i>Ilustración 99. Bloque creado con IceFactory</i>	53
<i>Ilustración 100. LEDs del Display 7 segmentos</i>	55
<i>Ilustración 101. Encendido de 2 LEDs del display</i>	55
<i>Ilustración 102. Encendido de todos los LEDs del display</i>	55

Ilustración 103. Valores necesarios para visualizar dígitos en el display	56
Ilustración 104. Visualización de 2 dígitos en el display.....	56
Ilustración 105. Visualización de digito implementado mediante tabla de la verdad	57
Ilustración 106. Cuenta cíclica de 0 a 3 usando una tabla de la verdad	57
Ilustración 107. Traducción binario-decimal mediante una tabla de la verdad.....	58
Ilustración 108. Funcionamiento Biestable Set-Reset	59
Ilustración 109. Pulsador de entrada y LED de salida	59
Ilustración 110. Encendido de un LED mediante Biestable	60
Ilustración 111. Esquema del ejercicio	60
Ilustración 112. Programación del ejercicio usando biestable.....	60
Ilustración 113. Reloj del sistema en distintos bloques	61
Ilustración 114. Conexión automático y manual del reloj	61
Ilustración 115. Señal del reloj del sistema	61
Ilustración 116. Ancho de pulso	62
Ilustración 117. Señal PWM	62
Ilustración 118. Señal Tic.....	62
Ilustración 119. Bloque señal Tic con pulsador	63
Ilustración 120. Funcionamiento del bloque Tic-Pulsador	63
Ilustración 121. Encendido de LED	63
Ilustración 122. Bloque temporizador	64
Ilustración 123. Programación de encendido de LED durante 3 segundos	64
Ilustración 124. Encendido de LEDs usando temporizadores	64
Ilustración 125. Generando pulsos periódicos	65
Ilustración 126. Control de brillo con señal PWM	65
Ilustración 127. Control de brillo de un LED	65
Ilustración 128. Bloque contador	66
Ilustración 129. Contador de 0 a 8 usando un contador y un display	66
Ilustración 130. Contador de 2 cifras encadenando contadores.....	67
Ilustración 131. Contador de tiempo.....	67
Ilustración 132. Combinaciones posibles.....	68
Ilustración 133. Programación del reloj	68
Ilustración 134. Recorriendo una tabla usando un contador	69
Ilustración 135. Biestable T	70
Ilustración 136. Encadenando biestables T	70
Ilustración 137. Biestable de datos	71
Ilustración 138. Captura de una constante	71
Ilustración 139. Capturando el estado de un Switch	71
Ilustración 140. Captura de varios estados	72
Ilustración 141. Desplazamiento de bit 0	72
Ilustración 142. Conversión serie paralelo	73
Ilustración 143. Bloque de carga paralela y registro de desplazamiento	74
Ilustración 144. Registro de 8 bits	74
Ilustración 145. Carga de 3 bits en binario.....	74
Ilustración 146. Bloque Registro de desplazamiento	75
Ilustración 147. Carga y desplazamiento de 3 bits.....	75
Ilustración 148. Comparadores	76
Ilustración 149. Comparación de dos datos	76
Ilustración 150. Bloques de comparación con una constante	77
Ilustración 151. Programación de la caja fuerte	77
Ilustración 152. Conexión ARDUINO-ALHAMBRA	78
Ilustración 153. Transferencia de datos	78
Ilustración 154. Conexión ARDUINO-ALHAMBRA	79

<i>Ilustración 155. Bloque usado para la recepción de datos.....</i>	79
<i>Ilustración 156. Conexión de la señal de control.....</i>	79
<i>Ilustración 157. Circuito sincronizador</i>	80
<i>Ilustración 158. Receptor serie</i>	80
<i>Ilustración 159. Función ShiftOut de Arduino.....</i>	81
<i>Ilustración 160. Recepción de datos.....</i>	81
<i>Ilustración 161. Conexión ARDUINO-ALHAMBRA</i>	82
<i>Ilustración 162. Registro de desplazamiento</i>	82
<i>Ilustración 163. Sincronización de datos</i>	83
<i>Ilustración 164. Circuito de lectura.....</i>	84
<i>Ilustración 165. Conexión puerto serie</i>	85
<i>Ilustración 166. Instalación de Arduino IDE</i>	85
<i>Ilustración 167. Permisos en la instalación de Arduino IDE</i>	86
<i>Ilustración 168. Circuito de comprobación.....</i>	86
<i>Ilustración 169. Bloque TX.....</i>	87
<i>Ilustración 170. Envío de una constante</i>	88
<i>Ilustración 171. Envío de un digito de 3 bits.....</i>	88
<i>Ilustración 172. Bloque TX String</i>	89
<i>Ilustración 173. Envío de cadena.....</i>	89
<i>Ilustración 174. Bloque RX.....</i>	90
<i>Ilustración 175. Visualización del dato recibido</i>	90
<i>Ilustración 176. Control de servo mediante dato recibido</i>	91
<i>Ilustración 177. Bloque control de posición del servo</i>	91
<i>Ilustración 178. Control de servo mediante dato recibido desde PC.....</i>	92
<i>Ilustración 179. Combinando RX y TX.....</i>	92
<i>Ilustración 180. Conexión Arduino-FPGA</i>	93
<i>Ilustración 181. Transmisión Arduino-FPGA.....</i>	93
<i>Ilustración 182. Prueba Eco con la conexión Bluetooth</i>	94

1. FPGA (“Field Programmable Gate Array”)

1.1. ¿Qué son las FPGAs?

Las FPGAs son como **chips en blanco**. Chips constituidos por puertas lógicas, cables y biestables, elementos básicos de los circuitos digitales, pero estos elementos están sin conectar.

Las FPGAs se programan normalmente usando un lenguaje específico (los dos más comunes son VHDL y Verilog) y que, tras cargarlo en el integrado, es creado físicamente en el chip, logrando construir cualquier circuito digital.

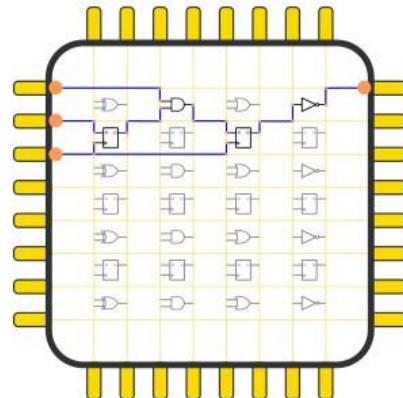


Ilustración 1. Interior de la FPGA

Ventajas:

- Son más rápidos que los sistemas programables.
- Pueden trabajar en frecuencias altas.
- Las FPGAs se ejecutan de forma paralelo como circuitos independientes, no como los sistemas microprocesadores que ejecutan un programa de forma secuencial.
- En el año 2015 se liberó el hardware (Clifford Wolf)

1.2. Proyecto ICESTORM

Proyecto creado por Clifford Wolfek (2015) para liberar la FPGA Lattice. Gracias a las herramientas libres que existen en este proyecto, se han creado otras herramientas, como, **Icestudio**, Software que deja accesible para muchas personas la síntesis y el diseño de FPGAs.

En España, la comunidad FPGA Wars, con la intención de desarrollar software y hardware libre, ha desarrollado el software **Icestudio** y la tarjeta electrónica **Alhambra II**.

1.3. Comunidad FPGA Wars

En España la comunidad FPGA Wars ha sido creada para promover el desarrollo de software y hardware libre.

<http://fpgawars.github.io/>

Participantes:

Juan González (Más conocido como Obijuan)

Jesús Arroyo Torrens

Eladio Delgado

Andrés Prieto-Moreno

Proyectos:

Entre los proyectos que nos ayudaran a trabajar con las FPGAs destacamos los siguientes dos proyectos:

- **Icezum Alhambra II:** Placa de desarrollo de FPGA (FPGA de Lattice iCE40HX1K-TQ144), placa diseñada por Eladio González en colaboración con Juan González.
- **Icestudio:** Software creado por Juan Arroyo que nos permitirá diseñar gráficamente los circuitos necesarios para programar la FPGA.

2. Instalación del Icestudio

El software Icestudio, nos permite crear circuitos digitales, sintetizarlo y descargarlo a la placa Alhambra II donde esta nuestra FPGA. Este software es multiplataforma y software libre.

Paso a paso para instalación en Windows:

2.1. Descargar el instalador de Icestudio desde GitHub:

<https://github.com/FPGAwars/icestudio>

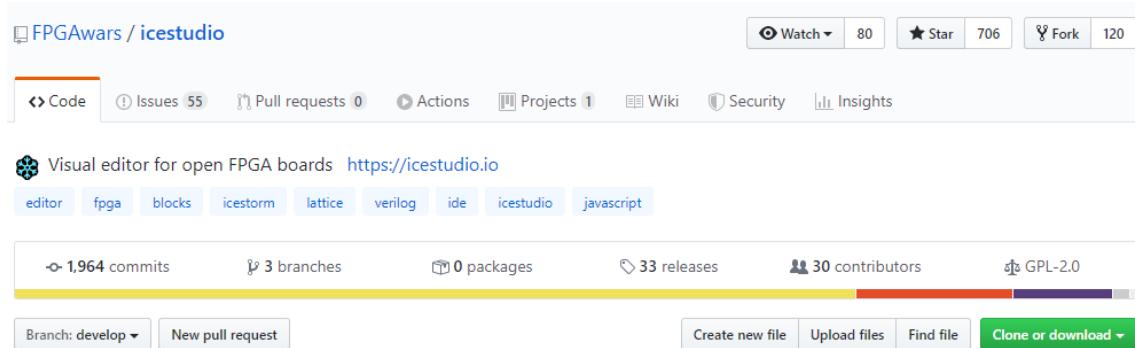


Ilustración 2. Github de FPGAwars

Bajamos hasta el apartado de **Installation** y hacemos clic en **Windows Installer**:

Installation

You can install **stable** or **nightly** Icestudio versions.

Because Icestudio is in development process, until it rises to 1.0 version, we are recommending to install **nightly** Icestudio versions, that have the latest features.

- GNU/Linux
 - i. Install [Python >= 3.5](#) and [xclip](#)
 - ii. For **stable** version, download and execute the [AppImage](#)
 - iii. For **nightly** version, download it from [icestudio.io](#)
- Windows
 - i. For **stable** version, download and execute the [Windows installer](#)
 - ii. For **nightly** version, download it from [icestudio.io](#)
- Mac OS
 - i. Install [Python >= 3.5](#) and [Homebrew](#)
 - ii. For **stable** version, download and execute the [DMG package](#)
 - iii. For **nightly** version, download it from [icestudio.io](#)

Ilustración 3. Opciones de instalación

Vemos las diferentes opciones de instalación, elegimos la que corresponda con nuestro sistema operativo, en nuestro caso la 0.5.0-win64.exe.

 icestudio-0.5.0-linux32.AppImage	111 MB
 icestudio-0.5.0-linux32.zip	118 MB
 icestudio-0.5.0-linux64.AppImage	110 MB
 icestudio-0.5.0-linux64.zip	117 MB
 icestudio-0.5.0-osx32.zip	107 MB
 icestudio-0.5.0-osx64.dmg	102 MB
 icestudio-0.5.0-osx64.zip	109 MB
 icestudio-0.5.0-win32.exe	122 MB
 icestudio-0.5.0-win32.zip	110 MB
 icestudio-0.5.0-win64.exe	131 MB
 icestudio-0.5.0-win64.zip	119 MB
 Source code (zip)	
 Source code (tar.gz)	

Ilustración 4. Versiones de icestudio disponibles

2.2. Ejecutar el instalador

Una vez descargado el instalador, lo ejecutamos haciendo doble clic. Dependiendo el antivirus que tengamos instalados nos pueden aparecer distintos mensajes que tenemos que obviar. También nos puede preguntar si permitimos que el instalador realice cambios, en ese caso le indicamos que sí.

Se abre el instalador y hacemos clic en **Siguiente**. El instalador detecta automáticamente si tenemos instalado **Python 2.7**. Si no lo está, procede a su instalación, y luego a la instalación de Icestudio.

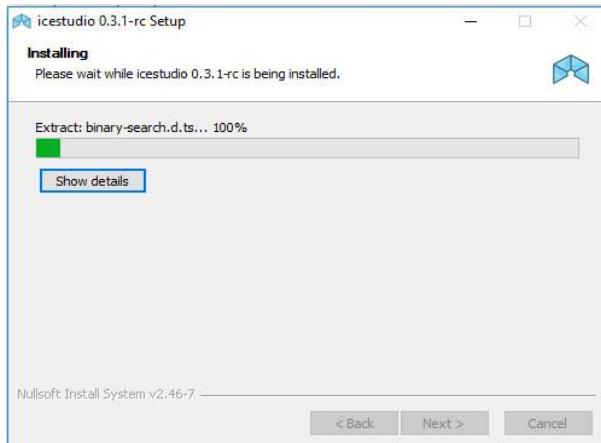


Ilustración 5. Proceso de instalación de icestudio

Al terminar la instalación nos sale un mensaje como el siguiente, donde hacemos clic en Terminar:

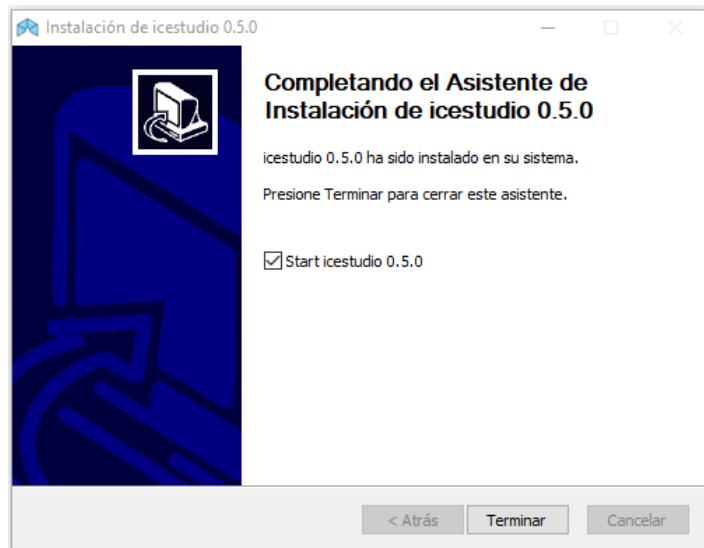


Ilustración 6. Instalación completada

Hacemos clic en Finalizar y se abrirá el software Icestudio.

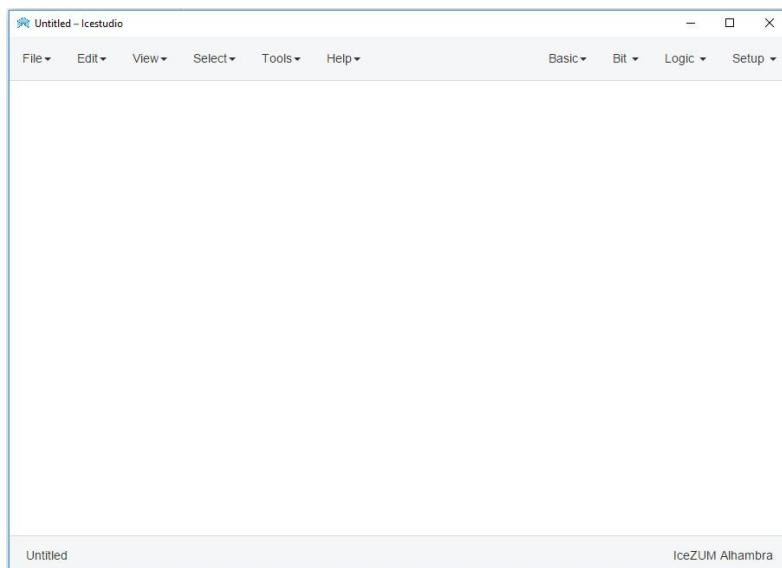


Ilustración 7. Pantalla de inicio de icesstudio

2.3. Configurar el idioma

Lo primero que debemos hacer al arrancar el icesstudio es configurar el idioma. Vamos al menú **Edit/Preferences/Language** y seleccionamos el idioma deseado.

2.4. Instalar el toolchain

Con lo que tenemos hasta ahora podemos abrir ejemplos y modificarlos, pero no podemos sintetizarlos en la FPGA, para ello tenemos que instalar las herramientas necesarias (toolchain). Para ello, vamos a **Herramientas/Toolchain/Instalar**

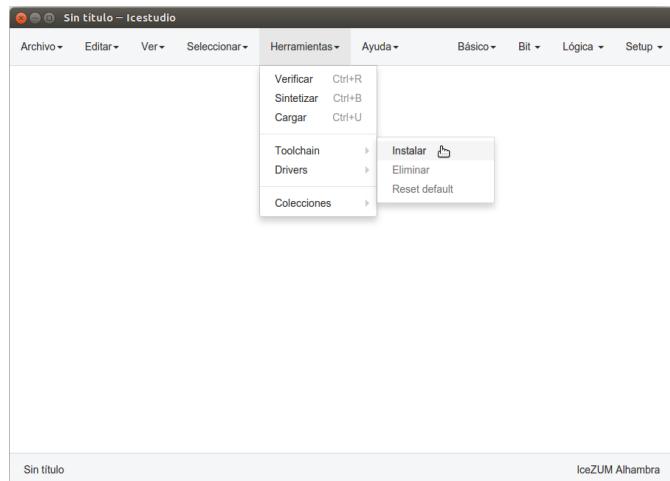


Ilustración 8. Instalación del toolchain

Comienza su instalación. Nos aparece una ventana con una **barra de progreso**. Este proceso puede durar unos minutos. Al terminar, nos aparece una **notificación** verde en la parte inferior derecha de la pantalla indicando que la **toolchain** está instalada.

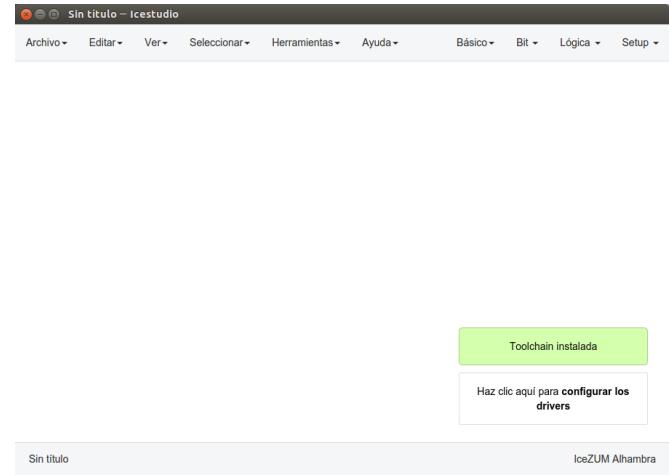


Ilustración 9. Instalación finalizada

Si tienes algún problema al instalar el toolchain, puede ser por culpa del WIFI. Intenta hacerlo conectado vía cable.

2.5. Instalar los drivers

En el punto en el que estamos ya podemos sintetizar el circuito (se generara el bitstream) pero no podemos cargarlo en la placa con la FPGA libres. Para poder hacerlo deberemos habilitar los drivers: **Herramientas/Drivers/Habilitar**.

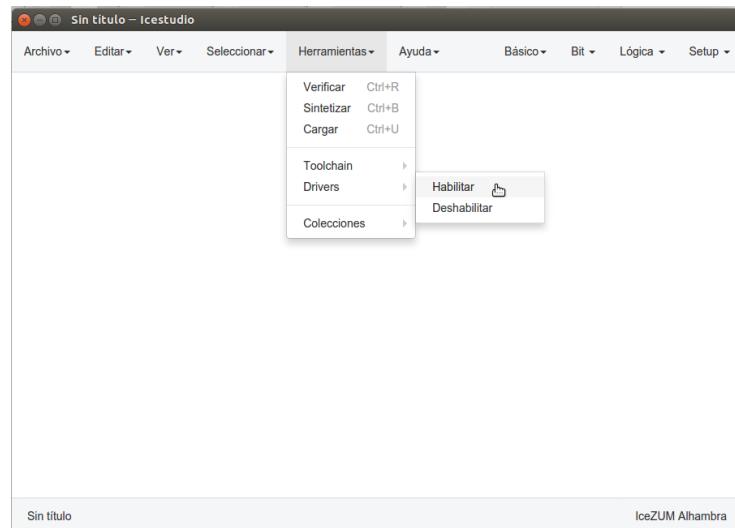


Ilustración 10. Instalación de los drivers

Hay que tener la placa de FPGA conectada y nos aparecerán los pasos a seguir.

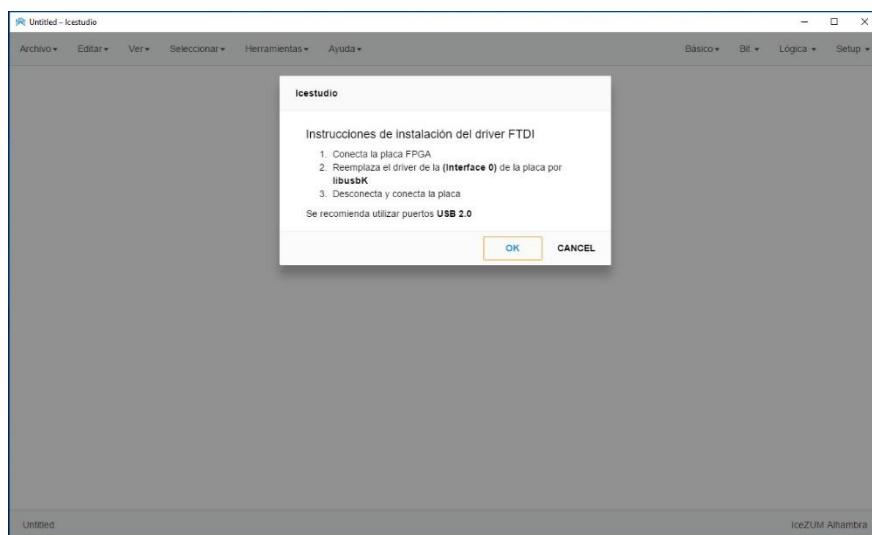


Ilustración 11. Pasos a seguir para la instalación

Una vez en este punto los pasos a seguir para instalar los drivers son los que a continuación enumeramos:

2.5.1. Conectar la placa de la FPGA al USB

Se recomienda conectar la placa de la FPGA a un **USB 2.0** del PC. Una vez instalado el driver, la placa de FPGA siempre deberá de conectarse al USB donde teníamos conectada la placa a la hora de instalar los drivers.

2.5.2. Seleccionar el driver

Cuando pulsamos Ok, Icestudio lanza la aplicación Zadig (que se ha instalado al instalar Icestudio) que nos permite instalar el driver de la FPGA.

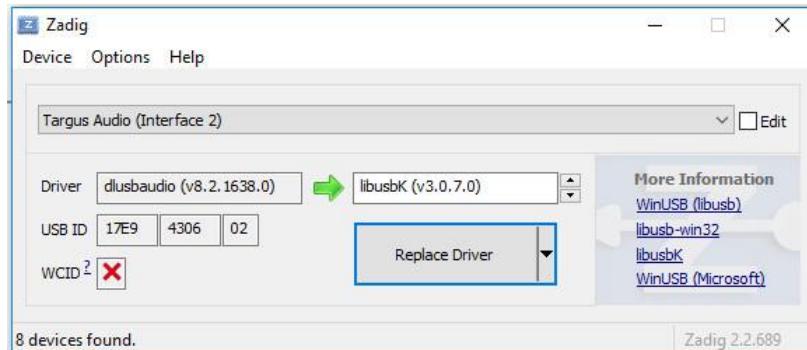


Ilustración 12. Aplicación Zadig

Al pulsar en el desplegable, debemos elegir la tarjeta de FPGA que utilizaremos, en nuestro caso la Alhambra II y dentro de las dos opciones que nos aparecen relacionadas con la Alhambra II elegimos la que pone **Interface 0**.

Entonces el desplegable se cierra y nos aparecen las opciones seleccionadas. En la casilla que está encima del botón de "Replace driver" debemos seleccionar el **driver libusK** (es muy probable que sea el que aparece por defecto), pero si no, lo seleccionamos con las flechas. Ahora hacemos clic **Replace driver** y comienza la instalación del driver.

En unos segundos aparece el mensaje de que se ha **instalado correctamente**. Vamos a comprobarlo.

a. Desconectar y conectar la placa

Una vez instalado el driver, debemos desconectar y volver a conectar la placa de FPGA para que el driver funcione correctamente.

b. Cargar el circuito

Ya podemos cargar el circuito sintetizado, para ello dibujamos un circuito o cargamos uno de los ejemplos existentes.

Una vez tenemos el circuito que queremos cargar, debemos elegir la opción **Herramientas/Cargar o Ctrl+U** y nos aparece una notificación indicando que comienza la carga.

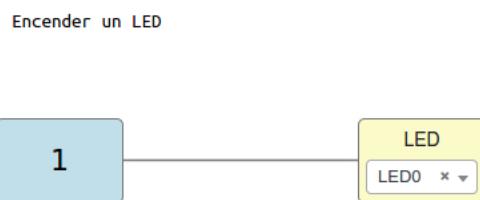


Ilustración 13. Circuito de ejemplo

Una vez terminada la carga nos aparece una notificación verde indicándonos que la carga se ha realizado correctamente y podemos verificar que el circuito funciona perfectamente.

3. Software Icestudio

Vamos a ver para que sirven los menús **seleccionar** y **herramientas** del software Icestudio que tenemos en la barra de opciones superior.

3.1. Seleccionar

En esta opción seleccionamos la placa con la que trabajamos, en nuestro caso, la Alhambra II y la colección que queremos cargar (Ver apartado 5. *Colecciones en icesstudio*).

A la hora de programar la FPGA nos basamos en distintos elementos gráficos que al conectarse entre ellos crean el circuito deseado. Estos elementos están disponibles en colecciones que podemos crear o cargar. Esta colección aparece en la parte derecha del menú. En la colección por defecto, aparecen: Básico, Bit, Lógica y Setup, pero a medida que creamos o carguemos otras colecciones estas opciones varían.

3.2. Herramientas

En la opción Herramientas hay tres grupos de opciones:



Ilustración 14. Menú Herramientas

- Opciones relacionadas con verificar el circuito que hemos dibujado, sintetizar el circuito y cargarlo en la FPGA.
- Opción de cargar, actualizar o eliminar el toolchain, así como habilitar o deshabilitar el driver.
- Y, por último, la opción para añadir o borrar colecciones.

4. Placa Alhambra II

La placa Alhambra II ha sido desarrollada por Eladio Delgado Mingorance, y es una de las placas que nos permitirá probar nuestros circuitos diseñados con Icestudio.

La placa ha sido diseñada con herramientas de *código abierto* (FreeCad, KiCad, Inkscape y LibreOffice) y es compatible con la cadena de herramientas (toolchain) de fuente abierta de Clifford Wolf.

Las características de la placa Alhambra son las que a continuación enumeramos:

- Placa de desarrollo FPGA (iCE40HX4K-TQ144 de Lattice)
- Hardware abierto
- Compatible con opensource icesstorm toolchain e Icestudio
- Pinout similar a Arduino Uno
- Oscilador MEMS de 12 MHZ
- Interruptor ON / OFF (apaga tu robot móvil fácilmente)
- Fuente de alimentación USB. Dos conectores. Hasta 4.8A
- Pines analógicos (aunque el convertidor A / D incrustado i2c)
- 20 pines de entrada / salida 3.3v (5v tolerante)
- El dispositivo USB FTDI 2232H permite la programación FPGA y la interfaz UART a un PC
- Botón de reinicio
- 8 leds de propósito general (leds de usuario)
- 2 pulsadores de uso general
- Memoria flash de 4MB



Ilustración 15. Alhambra II

Desde la siguiente dirección se puede comprar la placa y/o consultar la información sobre la placa Alhambra II:

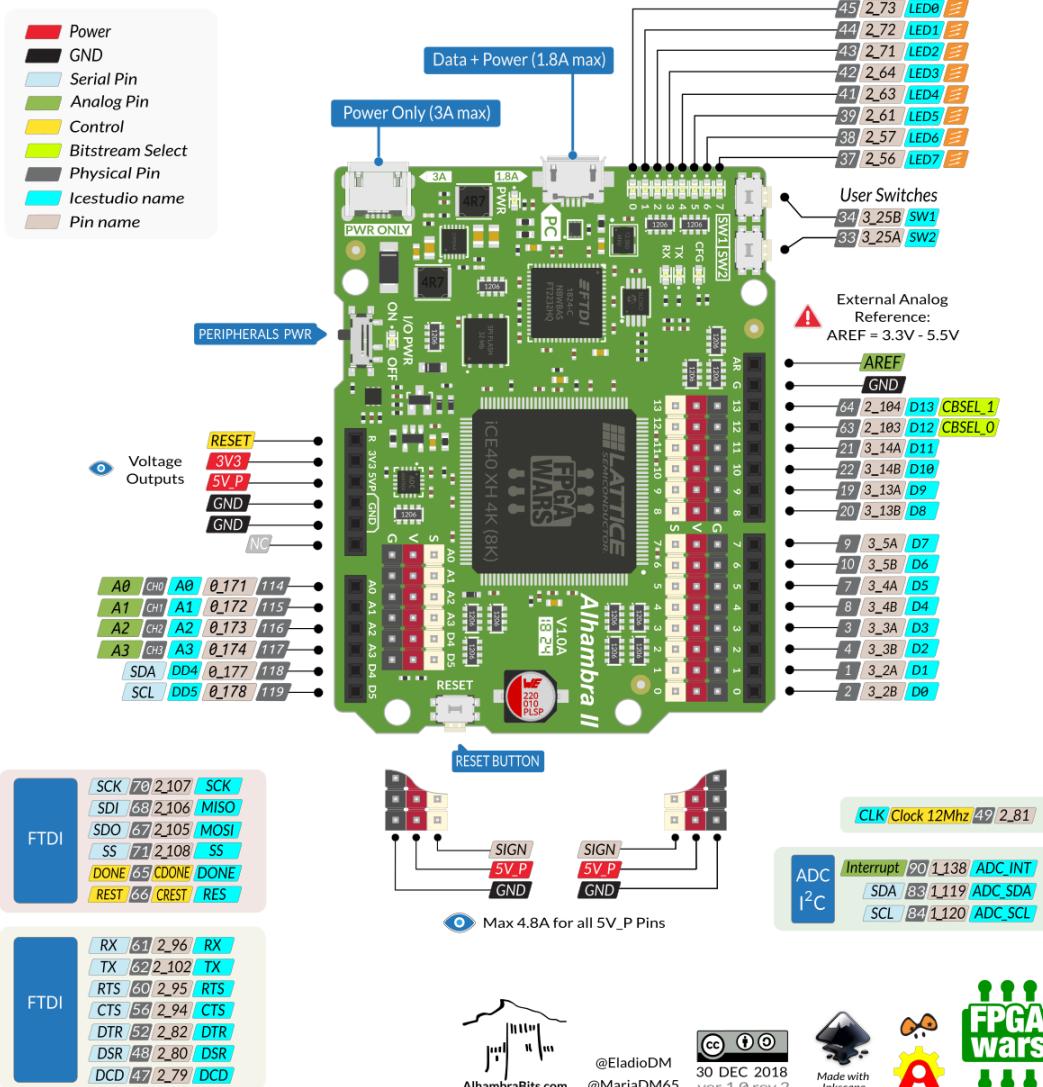
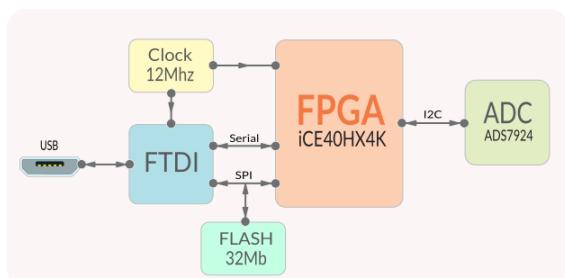
<https://alhambrabits.com/alhambra/>

4.1. Pinout Alhambra II

Alhambra II V1.0A

Open FPGA Board

- ⚠ Maximum Supply Voltage: 5.5V
- ⠑ Operating Supply Voltage: 3.5 - 5.5V
- ✓ 3.3V GPIO, 5V Compatible
- ⠑ Each GPIO Includes a 200 ohm Serial Resistor
- ✓ ADC Internal to External Ref. Automatic Switching



5. Colecciones en Icestudio

Las colecciones contienen bloques y ejemplos hechos por otros. Son muy útiles ya que puede que los bloques necesarios para nuestro programa ya estén creados con el consiguiente ahorro de tiempo y trabajo. Como ya dijimos, la colección por defecto tiene los menús Bit, Lógica y Setup y varios ejemplos donde se usan estos bloques. En **Ver/Información** tenemos información sobre la colección: Descripción, autores, traducciones...

Nosotros vamos a la colección creada expresamente para este curso, llamada "CursoFPGA". Todos los ejemplos y ejercicios de este curso están implementados en esa colección.

Descargamos la colección deseada y **NO** la descomprimimos. Vamos al menú **Herramientas/Colecciones/Añadir** y seleccionemos el .zip correspondiente. Una vez añadida vamos al menú **Seleccionar/Colección** y vemos que tenemos la colección por defecto seleccionada y las que hayamos añadido en la parte inferior. Seleccionamos la deseada y nos cambia la barra de opciones superior. Ahora tenemos los menús correspondientes a esa colección.

6. Circuitos digitales en Icestudio

Vamos a empezar a programar nuestra FPGA. Para ello realizaremos diversos ejercicios que permitirán al alumnado y profesor poder realizar distintos circuitos digitales que probaremos en nuestra FPGA. Estos ejercicios nos servirán para practicar y conocer los distintos elementos que se pueden utilizar en circuitos digitales y a su vez aprenderemos a programar nuestra FPGA.

Los circuitos digitales trabajan con BITS. Cada BIT es un dato que puede tener dos estados diferentes: **uno o cero**. Con la combinación de estos 1 y 0 conseguimos realizar cualquier programa. Estos circuitos sirven para almacenar, transportar y manipular conjuntos de bits (dígitos binarios).

6.1. Construcción de circuitos básicos

En la barra de opciones de Icestudio, tenemos el menú **Const/Bits**, y dentro tenemos los dos estados posibles del Bit. Clicamos en el 1, y los arrastramos al centro del programa.

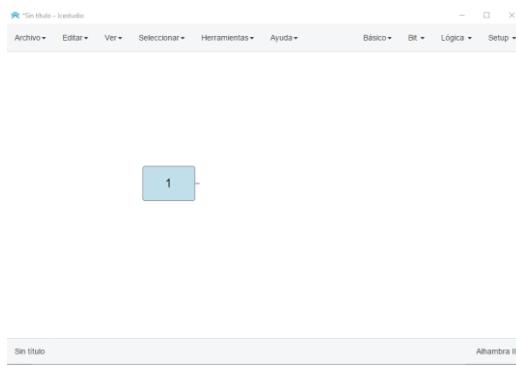


Ilustración 16. Bit 1

Ahora colocamos el bloque que se corresponde con la pata de la FPGA donde queremos visualizar ese BIT 1. En el menú Basic tenemos la opción Output (salida). Al igual que con el BIT, clicamos Output y lo arrastramos cerca del BIT.



Ilustración 17. Bit 1 y Salida

Tiramos el cable desde el BIT hasta la salida para conectarlos. Y seleccionamos la Salida en el desplegable. Por ahora vamos a trabajar con LEDs y botones integrados en la propia placa, así que seleccionamos uno de los LED.



Ilustración 18. Unión de Bit 1 y Salida

Una vez construido el circuito, lo probamos. Conectamos la Alhambra al ordenador y vamos a **Herramientas/Cargar** o hacemos **CTRL+U**.

Una vez realizada la carga, se nos notifica con un mensaje en la esquina inferior derecha. Y vemos como se enciende el LED de la placa seleccionado.

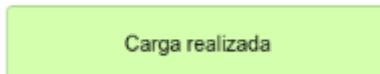


Ilustración 19. Mensaje recibido una vez realizada la carga del programa

6.2. Trabajo en paralelo

A diferencia de lo que ocurre en Arduino o en otros lenguajes de programación, en la programación Hardware trabajamos en paralelo, lo que quiere decir, que, si creamos dos circuitos independientes en el mismo programa, funcionarán correctamente mientras no se pisen entre ellos, esto es, mientras no utilicen la misma entrada/salida. Para probar esto, repetimos el ejercicio anterior. Copiamos el circuito ya creado y cambiamos la salida del segundo circuito.

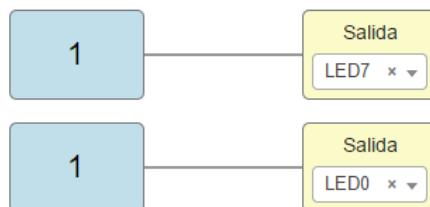


Ilustración 20. Dos funciones independientes en paralelo

Cargamos el programa y vemos como se encienden ambos LEDs a la vez.

6.3. Entrada variable

Además de trabajar con los BITS 0 y 1, también podemos trabajar con los pulsadores de la propia Alhambra. Gracias al pulsador podemos conseguir un 0 o un 1 cuando deseemos.

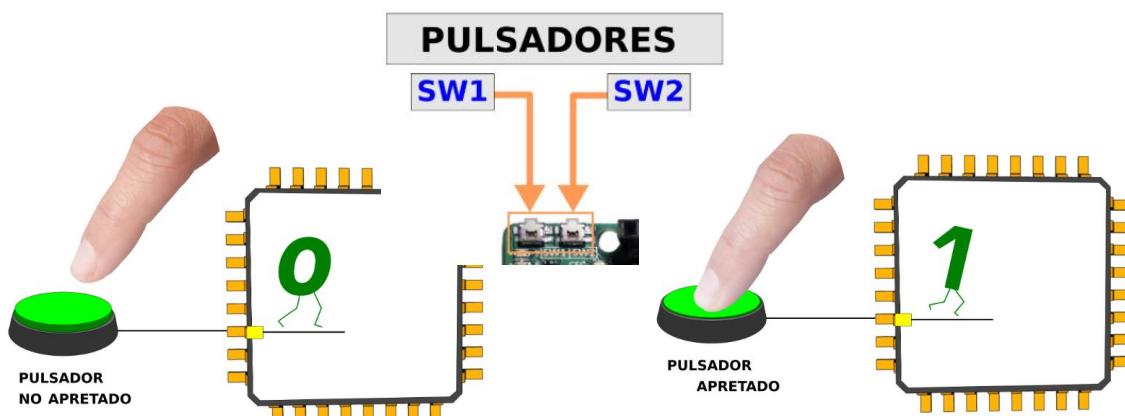


Ilustración 21. Esquema de funcionamiento de los pulsadores

Para probar estos pulsadores, sustituimos el BIT del anterior ejemplo por una entrada y elegimos SW1 y SW2. Cargamos y probamos.

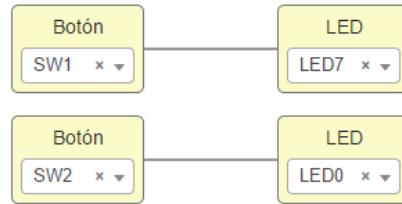


Ilustración 22. Funciones paralelas independientes con entradas variables

Vemos como al pulsar el Botón 1 (SW1) se enciende 7 y al pulsar el Botón 2 (SW2) se enciende el LED 0

7. Pines Externos de Alhambra II

Hasta ahora hemos trabajado con los LEDs y los botones de la propia Alhambra, pero habrá ejercicios o proyectos para los que necesitaremos conectar más pulsadores o incluso otros dispositivos diferentes. La Alhambra II dispone de **20 pines de E/S de 5V**, de los cuales 14 son Digitales y 6 son analógicos y los tenemos tanto machos como en hembras.

Los **pines de 5v** están conectados a los de la FPGA a través de unos **conversores de nivel** (3.3V \leftrightarrow 5V). El valor de la **resistencia R** que usamos, determina la **configuración** del conversor: entrada o salida. Para conectar un LED el pin debe ser de **salida** y para ello hay que colocar una resistencia **mayor o igual a 2K2**.

El inconveniente de esto es que el **LED se iluminará poco** porque la resistencia es grande y deja pasar **poca corriente**. Para que el LED luzca con **más intensidad** hay dos opciones: utilizar un **LED de alta luminosidad** o bien realizar la conexión a través de un **transistor**

Por otro lado, tanto los pulsadores como los interruptores pueden estar en **dos posiciones**: en una de ellas envían un **bit a 0** y en la otra un **bit a 1**. El **pulsador** cuando está en **reposo** se encuentra en su **posición estable**. Nosotros usaremos esta posición para transmitir el **bit 0**: que significa pulsador **NO apretado**. Al **apretarlo** con el dedo envía un **1**, y permanece en esa posición de forma **temporal**, mientras mantengamos el dedo **haciendo fuerza**. En el momento en que quitamos el dedo, el pulsador volverá **automáticamente** a su posición de reposo.

Vamos a probar el mismo circuito que en el apartado anterior. Pero esta vez usaremos un botón externo como entrada y un led externo como salida. El esquema de conexión es el siguiente:

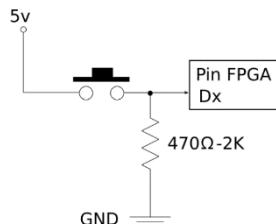


Ilustración 24. Esquema de conexión de los pulsadores

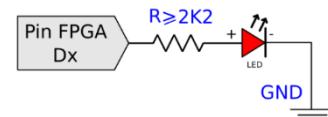


Ilustración 23. Esquema de conexión de los LEDs

El circuito del Icestudio es muy simple, tan solo necesitamos una entrada y una salida. Y elegiremos los pines digitales en los que hayamos conectado el botón y el LED:

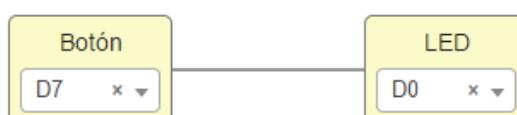


Ilustración 25. Conexión de pulsador y LED externos

Cargamos el programa y lo probamos.

8. Manipulación de Bits. Puertas lógicas

Para modificar o manipular los bits a nuestro gusto, usamos puertas lógicas. A continuación, veremos las principales.

8.1. Puerta NOT

La puerta **NOT** es la más sencilla, simplemente cambia el valor del bit. Transforma el 1 en 0 y el 0 en 1.

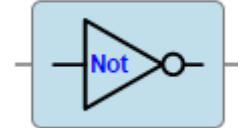


Ilustración 26. Puerta NOT

8.2. Puerta AND/NAND

Esta puerta **AND** cuenta con dos entradas y una salida. Para que la salida sea un 1, ambas entradas tienen que estar activadas, sino tendremos un 0 en la salida. Por ello el nombre, si se activan la entrada 1 **Y** la entrada 2, se activa la salida.

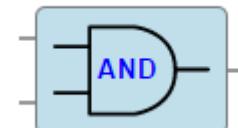


Ilustración 27. Puerta AND

La puerta **NAND** es la negada de AND, esto es, es como una puerta NOT en serie con una AND. La salida del **NAND** será la contraria que si fuese una AND. Siempre estará a 1, menos cuando ambas salidas estén activas, que la salida pasará a ser 0.

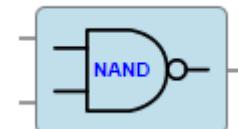


Ilustración 28. Puerta NAND

8.3. Puerta OR/NOR

Al contrario que con la puerta AND, en la puerta **OR** solo es necesario que una de las dos entradas esté activada para tener la salida activada, esto es, mínimo tiene que hacer una entrada a 1 para que la salida

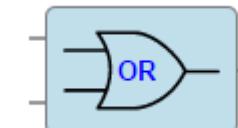


Ilustración 29. Puerta OR

Al igual que pasa con la AND y la NAND, la puerta **NOR**, es lo contrario que la OR. En cuanto haya alguna entrada activa, la salida será un 0.

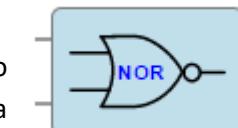


Ilustración 30. Puerta NOR

8.4. Puerta XOR/XNOR

En la salida de la puerta **XOR** tendremos un 1 **SOLO** cuando una de las dos entradas sea un 1. Si ambas entradas están activas, la salida será un 0.

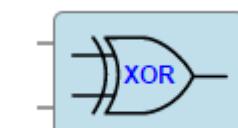


Ilustración 31. Puerta XOR

En la puerta **XNOR** ocurrirá lo contrario: Cuando una entrada sea un 1, la salida será un 0.

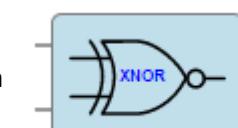


Ilustración 32. Puerta XNOR

SE RECOMIENDA HACER LOS EJERCICIOS 8.1, 8.2 y 8.3 DE LA COLECCIÓN

9. Bombeo de Bits

Como hemos visto en apartados anterior, podemos encender o apagar uno o varios LEDs a la vez, usando constantes o botones. En muchos de los proyectos se necesitan señales que cambien de estado con cierta frecuencia, esto es, un bit que “parpadee”. Para ello, vamos a ver un bloque llamado Corazón, creado por Obijuan y que podemos encontrar en nuestra colección.

Tenemos el menú **Varios/Bombeo**, donde tenemos varios corazones que bombean bits a distintas frecuencias. Elegimos uno de ellos y lo llevamos al centro del programa. Como podemos ver, este bloque tiene una entrada y una salida. La entrada tiene conectado el reloj de la Alhambra, que usa para bombear los bits a la frecuencia deseada, mientras que la salida es la que conectamos al LED de la Alhambra para poder visualizarlo.

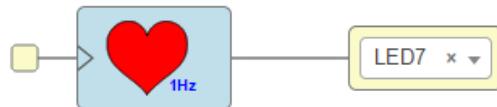


Ilustración 33. Bombeo de Bits al LED7

Si hacemos doble clic en el bloque, podemos ver la programación **Verilog** que lleva este bloque para su correcto funcionamiento.

Si queremos varios LEDs parpadeando a la misma frecuencia, podemos simplemente, conectar varias salidas al mismo corazón.

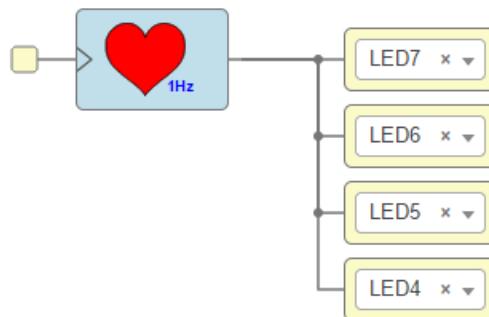


Ilustración 34. Bombeo de Bits a varios LEDs al mismo tiempo

SE RECOMIENDA HACER LOS EJERCICIOS 9.1, 9.2, 9.3 y 9.4 DE LA COLECCIÓN

10. Servos

10.1. Introducción a los servos

Los servos son actuadores o motores cuyo eje se puede fijar a una posición concreta dentro de un rango de unos 180º. Se suelen usar mucho en los proyectos robóticos para implementar articulaciones o pinzas, por ejemplo. El cable de conexión está formado por 3 cables: Masa (GND), alimentación (+5V) y señal de control.

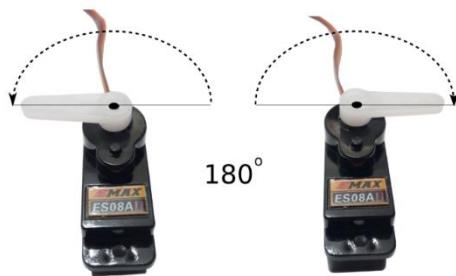


Ilustración 36. Movimiento del servo

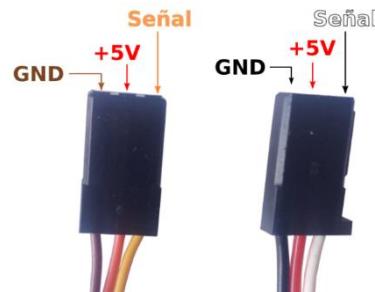


Ilustración 35. Conexión del servo

10.2. Señal PWM de control

Para mover la cabeza del servo, necesitamos una señal PWM. Se trata de un pulso de 5V, de anchura variable dependiendo de la posición deseada, que se debe enviar cada 20ms.

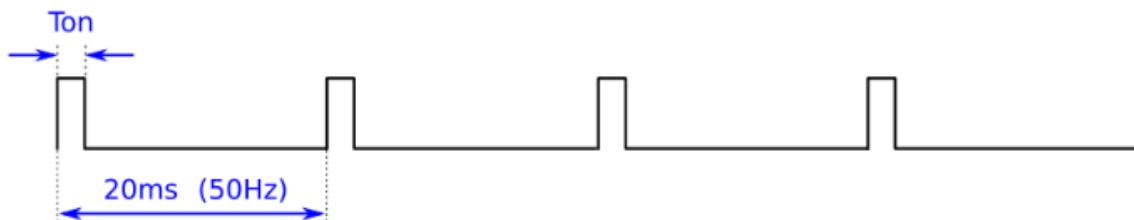


Ilustración 37. Señal PWM

Los valores exactos dependen del modelo del servo. El ángulo en el que se posiciona el servo es directamente proporcional al tiempo del ancho de pulso y se tiene que repetir periódicamente. En caso de que no se repita, ya no es una posición fija y puede ser movido con la mano a cualquier posición.

10.3. Conexión Servo a la Alhambra II

El conector hembra del servo lo podemos conectar directamente a cualquiera de los pines machos de la placa, siempre respetando la polaridad.

Para programar el movimiento del servo, tenemos el bloque servo. Consta de 2 entradas y una salida. Una de las entradas es el reloj que viene conectado por defecto, mientras que la otra entrada es el dato de 8 bits que marca el ancho de pulso deseado. La salida tenemos que conectarla al pin digital al que hayamos conectado el servo.

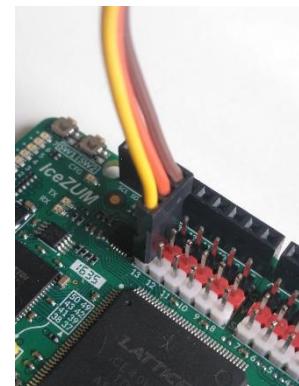


Ilustración 38. Conexión del servo a la FPGA

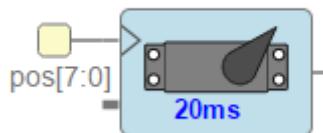


Ilustración 39. Bloque Servo de icesstudio

Con el fin de meter un dato numérico al servo, contamos con el bloque Dato_A_8bit, gracias al cual podremos meter la posición del servo deseada con el formato 8dXXX.

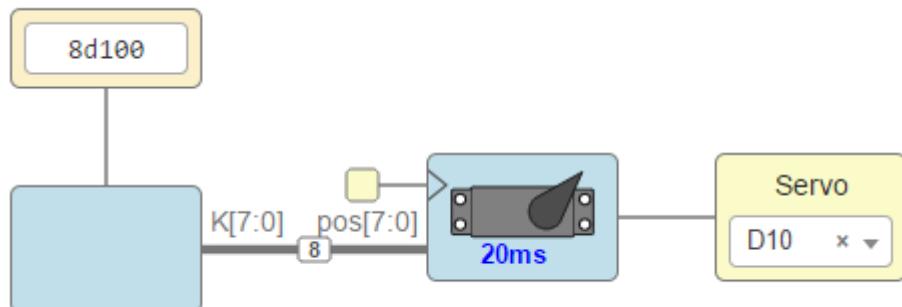


Ilustración 40. Bloque Dato_A_8Bit

Tomando este circuito como base, tenemos el bloque **ServoDosPosiciones** que nos servirá para controlar un servo mediante un pulsador, esto es, podremos mover el servo a dos posiciones.

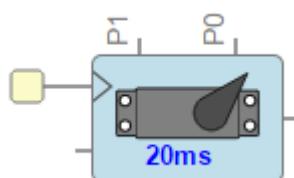


Ilustración 41. Bloque ServoDosPosiciones

Por lo tanto, conectamos un pulsador de la Alhambra al servo. Elegimos las posiciones deseadas (en microsegundos) usando **Básico/Constante** para escribir el dato deseado. Y conectamos como salida el pin digital donde tenemos conectado el servo. Una vez cargado el programa, vemos como el servo se mueve a la posición 0 y cuando pulsamos el SW1, se mueve a la posición 1.

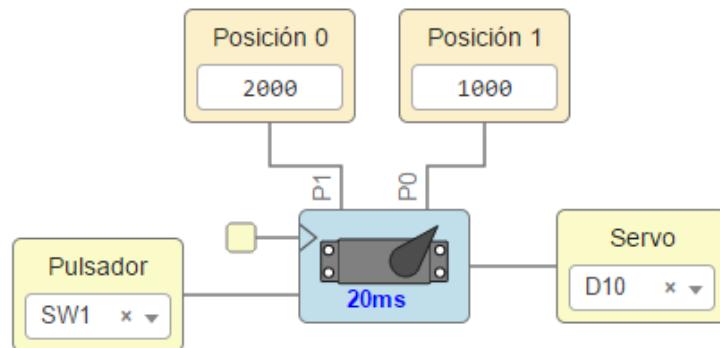


Ilustración 42. Movimiento del servo a dos posiciones

Podemos lograr este movimiento automáticamente en lugar del pulsador, conectamos un bloque Corazon. Además, conectamos los LEDs 7 y 0 al Corazón para ver el cambio mediante luces.

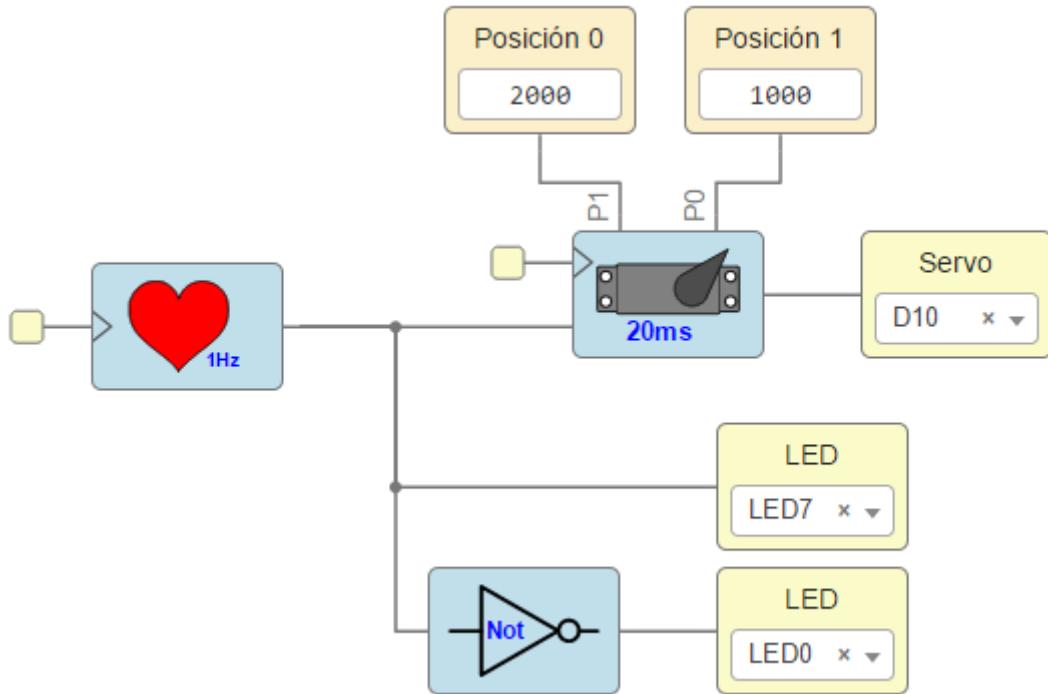


Ilustración 43. Movimiento automático del servo

SE RECOMIENDA HACER LOS EJERCICIOS 10.1, 10.2, 10.3 y 10.4 DE LA COLECCIÓN

11. Multiplexor

Hemos visto que se pueden implementar dos circuitos independientes en el mismo programa. Los multiplexores nos permiten combinar circuitos que usan las mismas salidas.

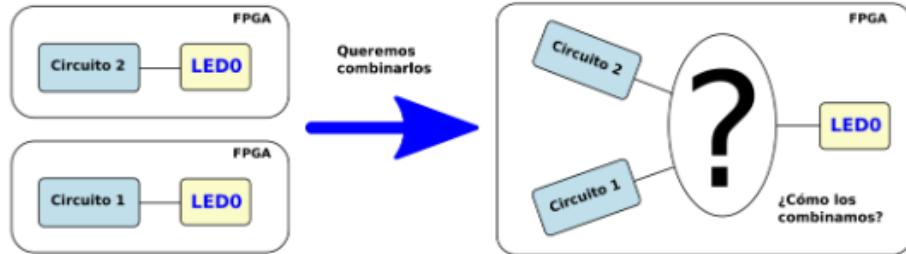


Ilustración 44. Función del multiplexor

Por separado funcionan perfectamente, pero al unirlos en el mismo programa tenemos el problema de que ambos acceden al mismo LED. Gracias al multiplexor podemos implementar ambos circuitos, siempre que lo hagan por turnos, nunca a la vez. Usamos otro bit para seleccionar cuál de los dos circuitos queremos usar en cada momento.

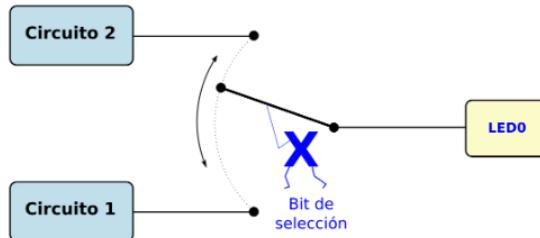


Ilustración 45. Funcionamiento del multiplexor

11.1. Multiplexor 2:1

Estos multiplexores tienen 3 entradas, 2 canales y una de selección, y una salida.

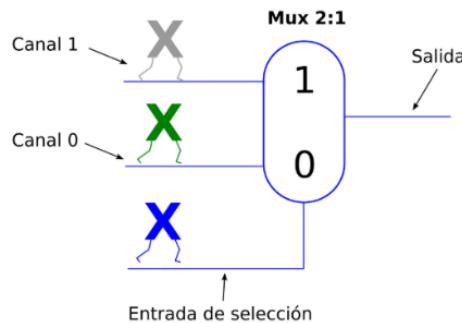


Ilustración 46. Esquema del funcionamiento del multiplexor

Tenemos estos bloques en el menú **Varios/Mux/1-bit**. Además del 2-1 normal, también contamos con un 2-1 con las entradas cambiadas de orden (flip), pero ambos funcionan de la misma forma. Como primer ejemplo, tenemos dos corazones de distinta frecuencia conectados a los canales de entrada y usamos un pulsador de la propia placa como bit de selección. Visualizamos el resultado usando un LED. Al cargarlo vemos como el parpadeo cambia de frecuencia al accionar el pulsador de la Alhambra.

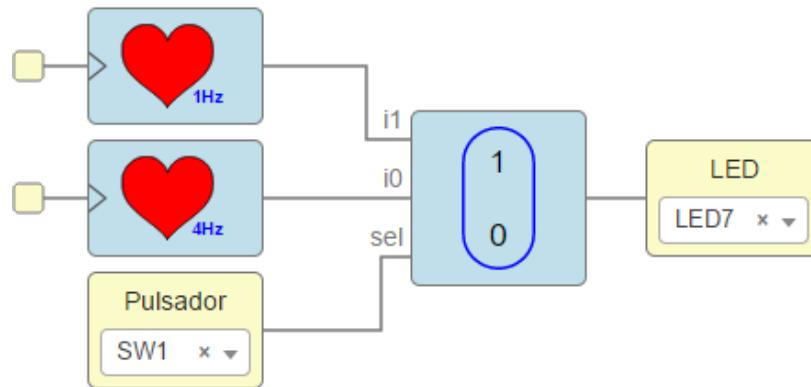


Ilustración 47. Ejemplo de funcionamiento del multiplexor

También podemos hacer este cambio automáticamente si conectamos otro corazón en la entrada de selección. Para este ejemplo, conectamos un corazón de 7Hz al primer canal de entrada y el corazón de 1Hz en la entrada de selección.

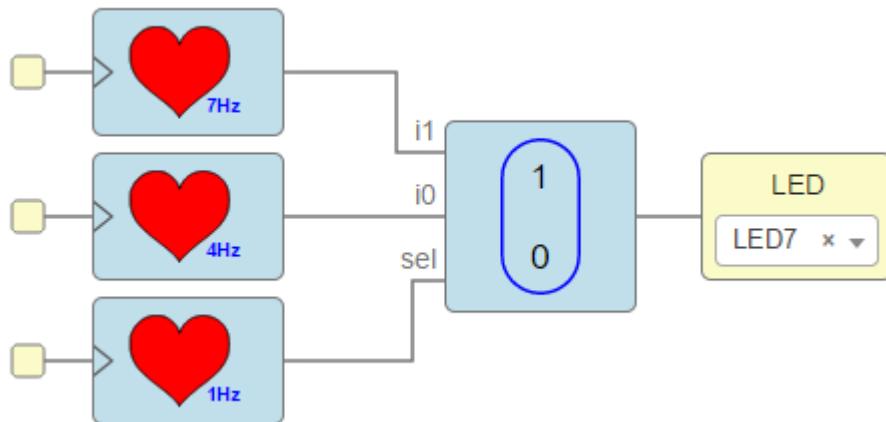


Ilustración 48. Ejemplo de funcionamiento automático del multiplexor

11.2. Multiplexor 4:1

La única diferencia con el 2:1 es que este tiene 4 canales en vez de 2, por lo tanto, para poder elegir entre 4 canales, necesitamos 2 bits de selección. Mediante la combinación de estos dos bits conseguiremos elegir uno de los canales. Por lo demás, el funcionamiento es el mismo. Tenemos este multiplexor en **Varios/Mux/1-bit**, al igual que con el 2:1, tenemos el mux 4:1 normal y el invertido. Para este ejemplo vamos a usar el normal y vamos a programar 4 estados diferentes para el mismo led.

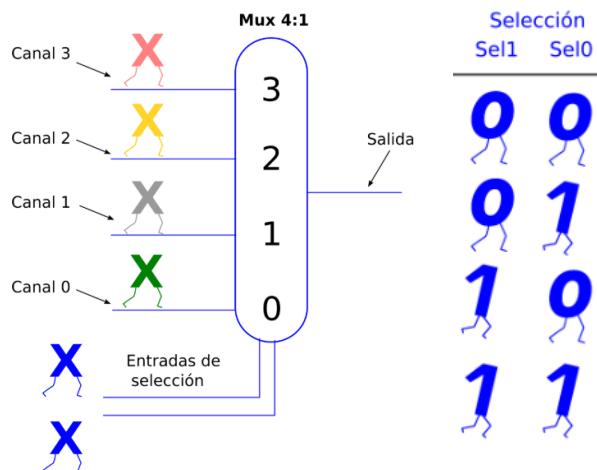


Ilustración 49. Esquema del Mux 4:1

Tenemos 4 canales: En dos de ellos, haremos que el LED parpadee a distintas frecuencias; en otro mantendremos el LED siempre encendido y en el último, siempre apagado. Controlaremos el multiplexor con los dos pulsadores de la placa.

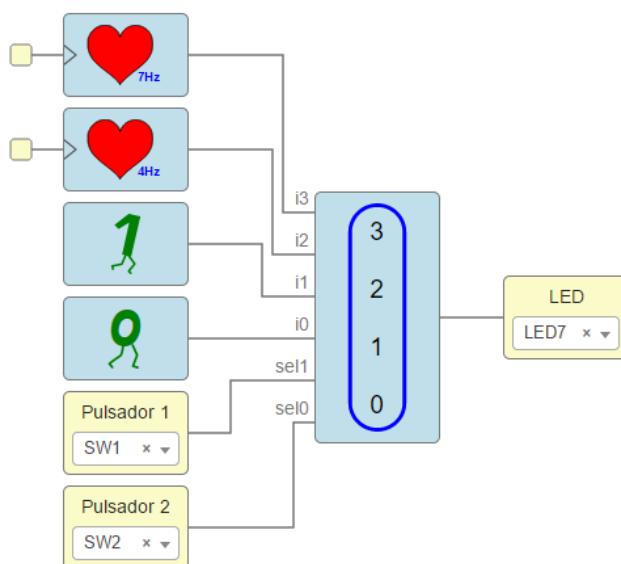


Ilustración 50. Ejemplo mux 4-1

SE RECOMIENDA HACER LOS EJERCICIOS 11.1 DE LA COLECCIÓN

12. Servos de rotación continua

Los servos utilizados hasta ahora nos permitían fijar un ángulo, pero tenían un tope mecánico que los impide girar más de 180 grados. Los servos de rotación continua, en cambio, se controlan por velocidad y al no tener topes mecánicos, puede girar continuamente. Son como motores de corriente continua que se pueden conseguir trucando servos normales.

Estos motores pueden girar en cualquier dirección y podemos controlar el sentido con el bit de entrada de giro. Además, tenemos otra entrada para controlar el encendido/apagado del servo. Con la combinación de estos 2 bits controlaremos el servo.

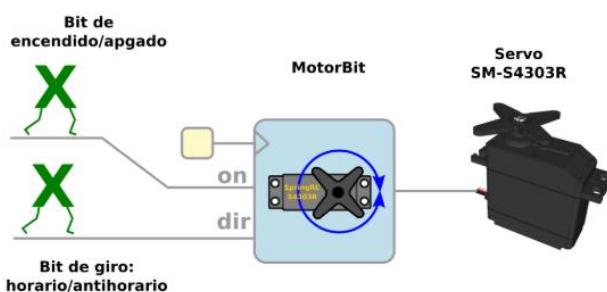


Ilustración 51. Esquema funcionamiento MotorBit

Entradas	Salidas
Bit de encendido/apagado (on)	MOTOR
Bit de giro: horario/antihorario (dir)	
0 0	Parado
1 0	Giro antihorario
1 1	Giro horario

Ilustración 52. Movimientos posibles del servo

Probamos este bloque con un circuito simple. Añadimos los LEDs para ver cómo actúan los bits.

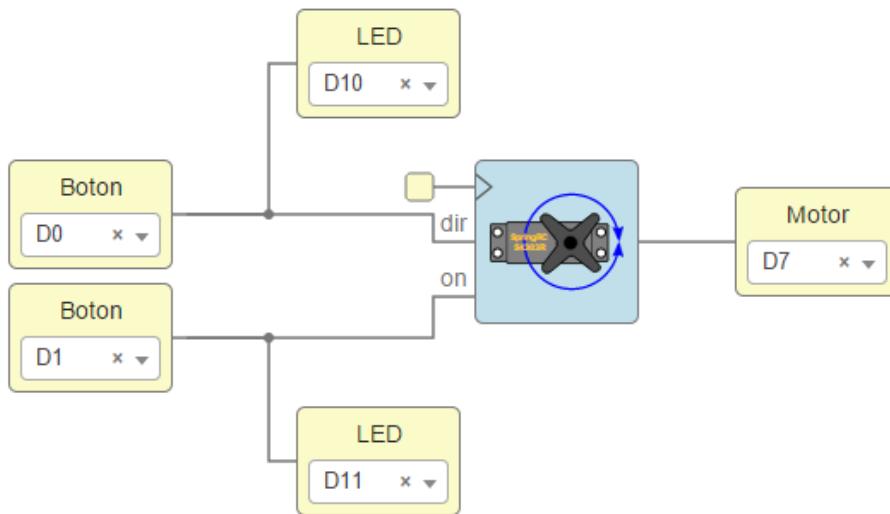


Ilustración 53. Prueba de funcionamiento de bloque MotorBit

El uso más común de estos servos es el de pequeños robots con ruedas. Por lo tanto, vamos a probar un programa con dos servos (uno para cada rueda) que pueda manejar el robot con cuatro botones. Debido a la colocación de las ruedas en el robot, para conseguir un movimiento rectilíneo cada una gira en sentido distinto, por lo que habrá que tener esto en cuenta a la hora de programa.

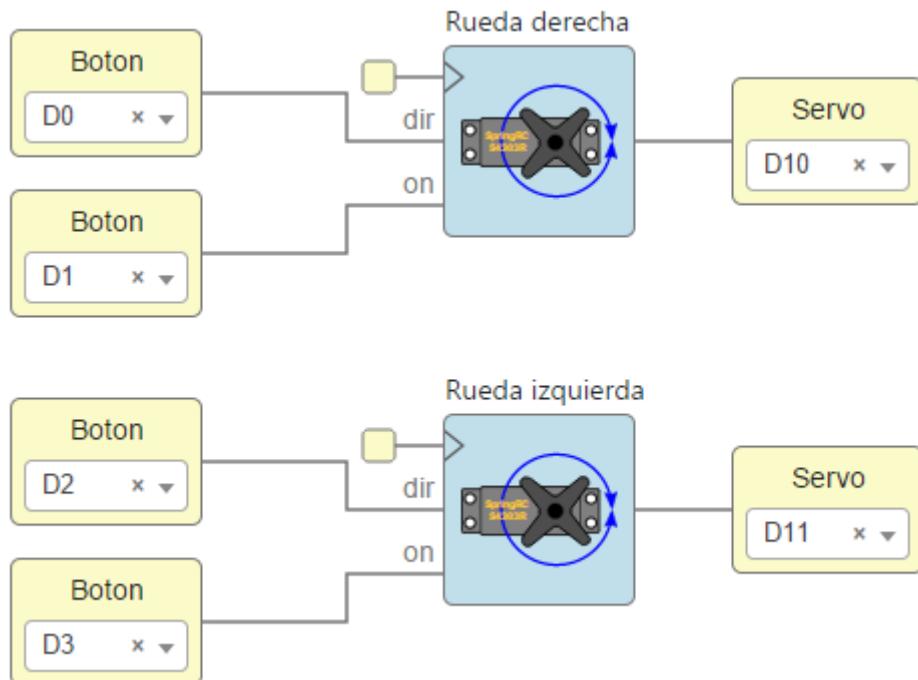


Ilustración 54. Programación del movimiento de dos servos en el mismo robot

SE RECOMIENDA HACER LOS EJERCICIOS 12.1 y 12.2 DE LA COLECCIÓN

13. Sensores de infrarrojos

Los sensores de infrarrojos nos permiten detectar objetos a corta distancia, detectando el color negro en superficies planas. Estos sensores emiten una luz constantemente que recibe el propio dispositivo cuando ésta rebota en un objeto que está lo suficientemente cerca. Cuando el receptor detecta la luz, el sensor nos devuelve un 1. En caso de no detectar, devuelve un 0. Lo mismo ocurre con líneas de colores, al detectar blanco, la luz se refleja y nos devuelve un 1, pero al detectar el negro, la luz no refleja y, por lo tanto, nos devuelve un 0.

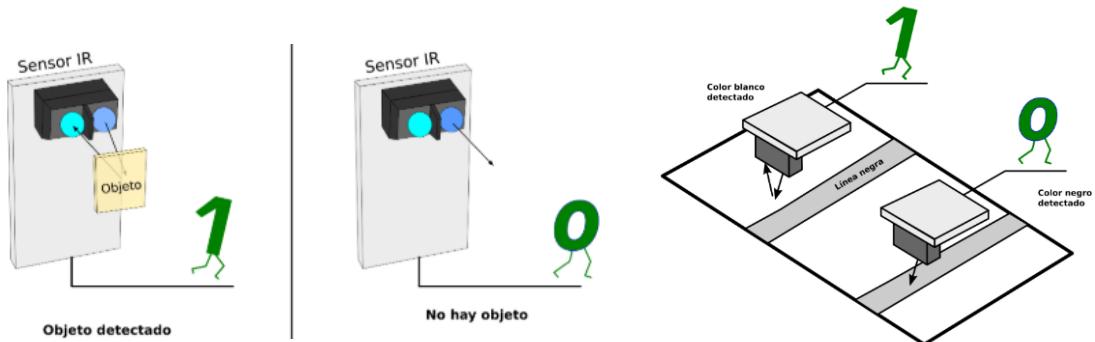


Ilustración 55. Funcionamiento de un sensor IR

Estos sensores tienen 3 cables de conexión al igual que los servos, señal, 5V y GND, por lo que lo conectamos igual que los servos, a cualquier pin digital de la placa Alhambra.

Simplificando, el sensor actúa como un pulsador: tiene dos estados, “encendido” y “apagado”. Por lo tanto, la programación en Icestudio será la misma. Una entrada para el IR, y una salida para un LED con el que visualizaremos el funcionamiento.



Ilustración 56. Programación para visualizar estado del sensor mediante un LED

Veremos cómo al acercar algún objeto, se enciende el LED y al alejarlo, se apaga.

Como hemos dicho, estos sensores son muy útiles para hacer robots seguir líneas, esto es, robots que detectan una línea negra y la siguen, siempre que sea en una superficie plana. Hacemos un circuito muy parecido al anterior.

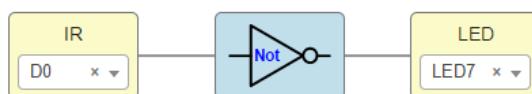


Ilustración 57. Programación sigue líneas

Le añadimos el NOT porque queremos que el LED se encienda cuando detecte el color negro.

Este dato se suele usar para mover o parar un motor. Ya hemos dicho que el IR funciona como un botón, por lo que la programación del servo rd la misma que uno de los ejemplos del apartado anterior.

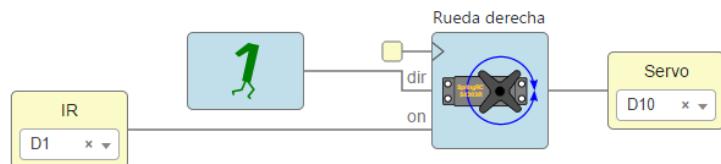


Ilustración 58. Programación para usar el sensor IR en el movimiento del servo

SE RECOMIENDA HACER LOS EJERCICIOS 13.1 DE LA COLECCIÓN

14. Bloques con parámetros

En este apartado aprenderemos a usar bloques genéricos que nos permiten crear bloques específicos asignando valores a sus parámetros. Mediante la composición, construimos bloques concretos a partir de otros más sencillos. El bloque de mayor entidad, llamado TOP, tiene las entradas y salidas de la FPGA.

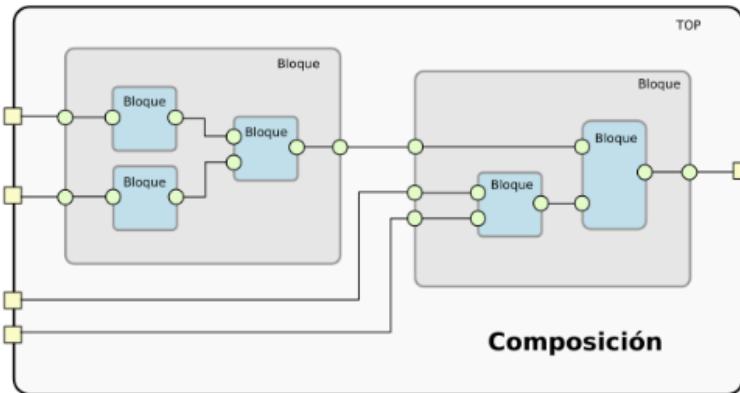


Ilustración 59. Bloques formados por otros bloques

Si por ejemplo cogemos el bloque “ServoDosPosiciones” y hacemos doble clic en él, se nos abre el circuito que lo forma.

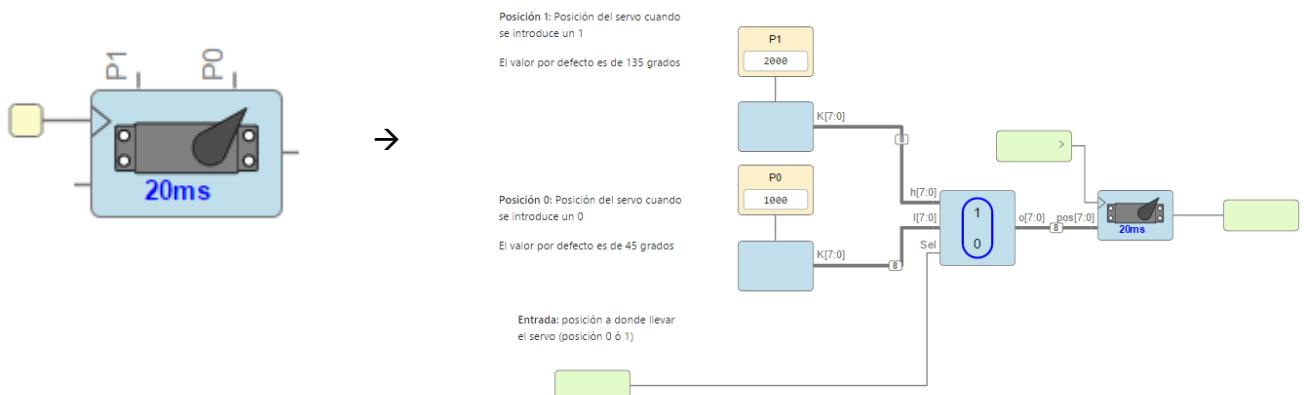


Ilustración 60. Programación interna del bloque ServoDosPosiciones

Y si continuamos haciendo doble clic en bloques como el multiplexor o el servo, siguen abriéndose otros circuitos más básicos.

Algunos bloques necesitan de ciertos parámetros para su correcto funcionamiento como el ServoDosPosiciones visto anteriormente y otros puede que no necesiten ni entrada o ni salida, como el Corazón visto en apartados anteriores.

15. Circuitos combinacionales

Los circuitos combinaciones son los que nos permite manipular los bits, pero no los almacenan, esto es, no dependen de datos recibidos en el pasado, solo los del presente. Ya hemos visto varios de estos circuitos, como las puertas lógicas (AND, OR...) o los multiplexores. Estos circuitos se caracterizan por estar totalmente definidos por su tabla de la verdad, ya que en ella se especifican todas las combinaciones posibles para las posibles entradas. Por ejemplo, para un circuito combinacional de 2 entradas y 1 salida, como las puertas lógicas, tenemos:

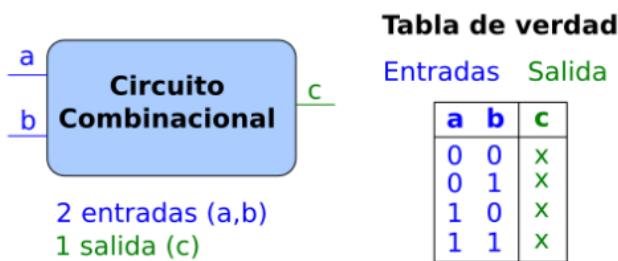


Ilustración 61. Tabla de la verdad de un circuito combinacional

Las combinaciones de **a** y **b** son las de la parte izquierda de la tabla mientras que a la derecha se pone la salida asociada a cada combinación, que puede ser 0 o 1. Cuantas más entradas tenga el circuito, más combinaciones posibles tendremos y, por lo tanto, mayor será la tabla de la verdad asociada.

Estos circuitos se pueden crear desde cero en el caso en el que no tengamos un bloque específico para el funcionamiento deseado. Para ello usamos las tablas de **Combi/Tablas** dependiendo de las entradas y salidas deseadas y seleccionamos HEX para hacerlo en hexadecimal o BIN si deseamos crearla con números binario.

Vamos a probar a crear la puerta NOT, por lo que seleccionamos una tabla BIN de 1 entrada y 1 salida y le añadimos los datos con **Basic/Memory**.

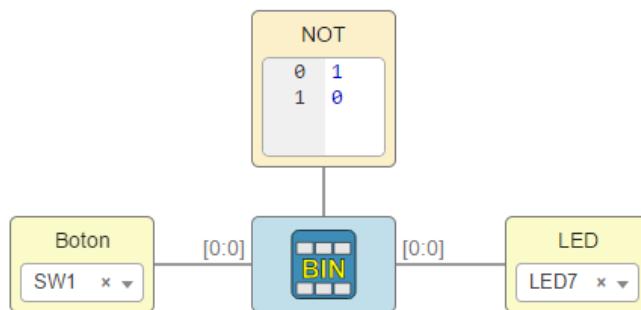


Ilustración 62. Puerta NOT mediante tabla

Podemos hacer lo mismo mediante puertas con más entradas, la única diferencia es que habrá que añadir más filas a la memoria. Probamos con una puerta AND, a cuya tabla de la verdad le hemos añadido una columna a la izquierda (Nº Fila) que nos da el valor en decimal correspondiente a la entrada en binario.

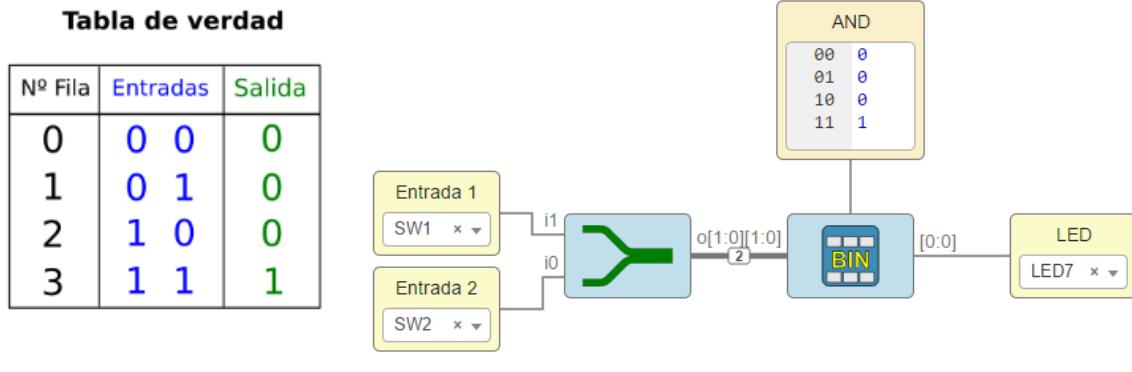


Ilustración 63. Puerta AND mediante tabla

Al tener dos entradas, la tabla nos pide 1 solo cable con ambos datos. Veremos cómo funcionan estos bloques separadores/agregadores en el siguiente apartado.

Además de puertas ya creadas, podemos crear circuitos para los que no haya un bloque ya creado. Por ejemplo, si queremos un comparador de bits que se active cuando ambas entradas tienen el mismo valor, simplemente tenemos que cambiar la memoria del ejemplo anterior y hacer que sea una cuando ambas entradas tienen el mismo valor.

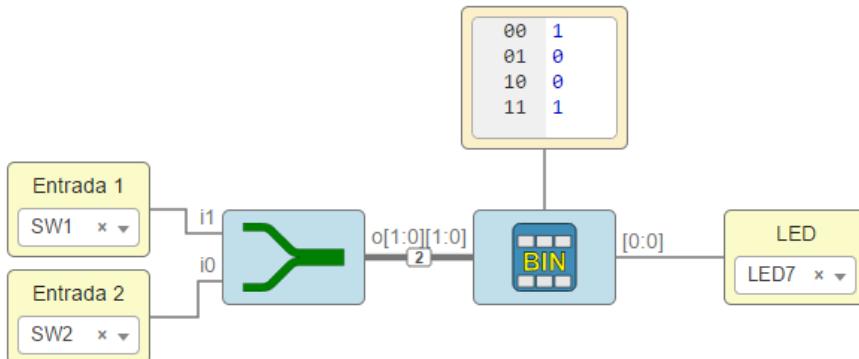


Ilustración 64. Puerta no existente creada mediante tabla

Los multiplexores también están formados por tablas, por lo que podemos hacer uno usándolas. Probaremos uno muy simple, dos entradas conectadas a corazones que bombearan bits y otra entrada para seleccionar cuál de los dos canales utilizar. El resultado lo visualizaremos en un LED. Es igual que uno de los ejemplos utilizados en el apartado de multiplexores, pero esta vez, en vez de usar el bloque ya creado, usamos una tabla de tres bits de entrada.

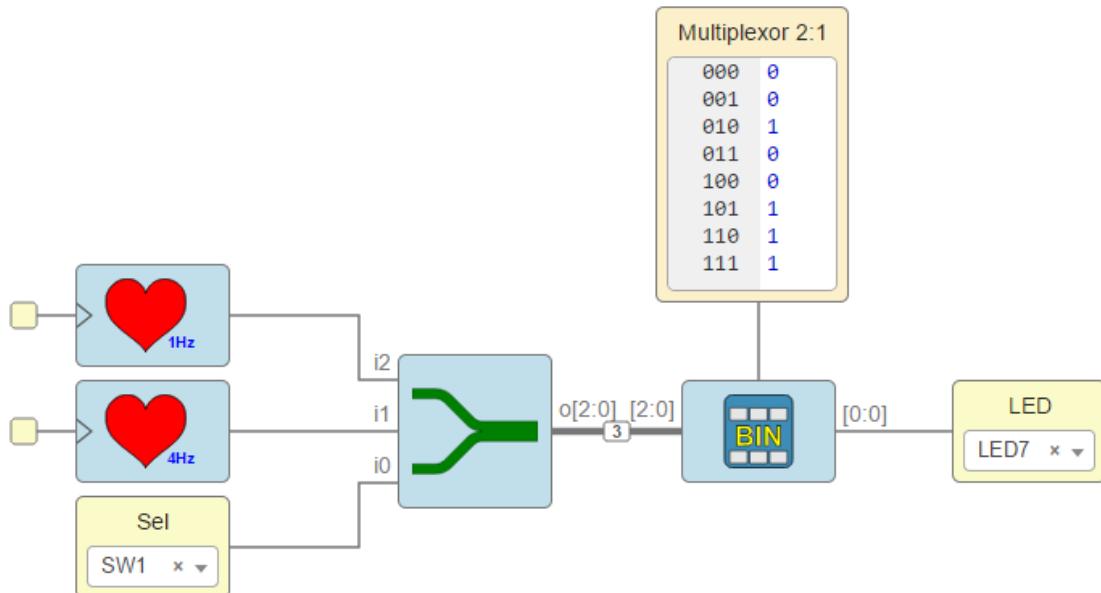


Ilustración 65. Multiplexor 2:1 creado mediante tabla de la verdad

Podemos seguir añadiendo entradas al circuito y solo cambiará el número de filas de la tabla de la verdad.

SE RECOMIENDA HACER LOS EJERCICIOS 15.1, 15.2 Y 15.3 DE LA COLECCIÓN

16. Buses y números

Hasta ahora hemos hecho circuitos que usaban bits de manera aislada y tenían pocos cables, pero con circuitos más complejos es recomendable agrupar los cables en buses y tratar la información como números en vez de como bits sueltos.

Tenemos buses de distintos tamaños en **Const/Bus**, dentro de cada carpeta tendremos un bloque genérico al cual le pasaremos el numero decimal que queremos tener en la salida. Si, por ejemplo, ponemos un tres, a la salida tendremos un 0011. Vemos que la salida del bloque tiene un $k[1:0]$, que significa lo siguiente:

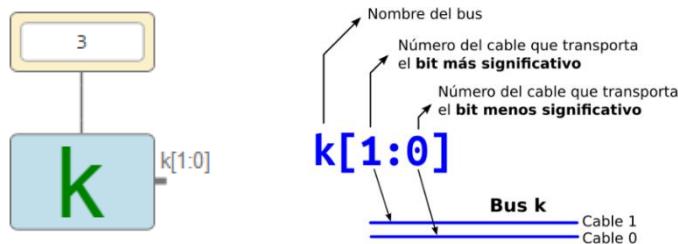


Ilustración 66. Bus

A la hora de agregar una salida, tambien deberemos decirle que se trata de un bus de 2 bits. Para ello, colocamos el bloque de salida como siempre, y como nombre le ponemos: **NOMBRE[1:0]**. Si fuera de 3 bits, pondriamos **NOMBRE[2:0]**, para 4 bits, **NOMBRE[3:0]** y así sucesivamente.

Enter the output blocks

LED[1:0]

FPGA pin

Ilustración 67. Parámetros del bus

Al hacer clic en OK, aparece un bloque de salida con dos cajetillas de selección. Lo unimos al bloque K y seleccionamos dos LEDs para probar. Estos buses funcionan igual que los cables por lo que podemos usarlos igual que lo haciamos con los cables para encender mas LEDs siempre que la salida tenga el mismo numero de datos posibles

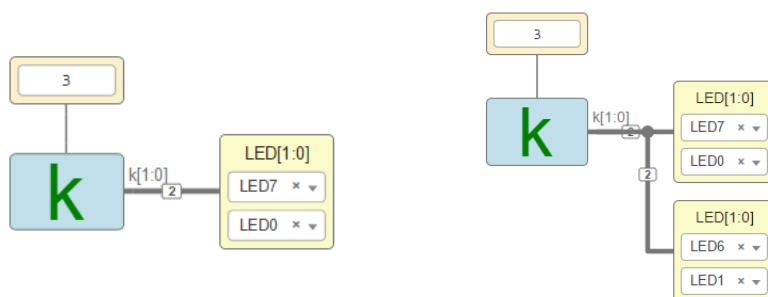


Ilustración 68. Constante 3 visualizada en 2 LEDs mediante bus de 2 bits

Los buses de 8 bits se utilizan bastante, por lo que vamos a probarlo. Tendremos que hacer exactamente lo mismo que en el ejemplo anterior, solo que esta vez la salida es $k[7:0]$, para que acepte los 8 bits. Al disponer de 8 bits, podremos visualizar números del 0 al 255, ya que este último es 1111 1111 en binario.

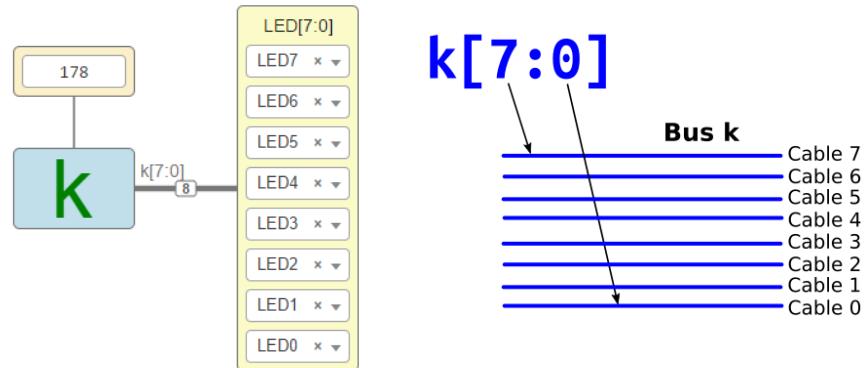


Ilustración 69. Constante 178 visualizada en 8 LEDs mediante un bus de 8 bits

16.1. Truncado

Al asignar un valor a la constante hay que tener en cuenta el número de bits que vamos a usar, si, por ejemplo, tenemos un bus de 4 bits, el mayor número decimal que se podrá enviar correctamente será el 15 ya que cualquier número superior se truncará. Si intentamos mandar un 16, que en binario es 10000, solo mandará los 4 bits **menos** significativos, esto es, el 1 desaparecerá (se truncará) y tendremos un 0000.

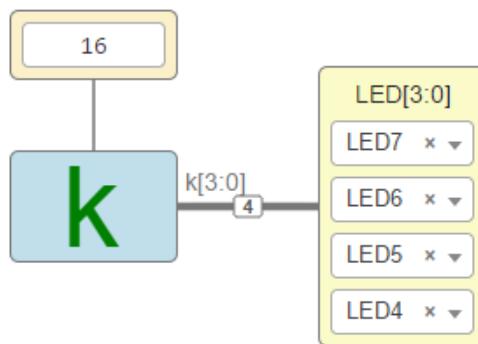


Ilustración 70. Dato truncado

16.2. Nomenclatura

Cuando introducimos los valores como parámetros sin indicar específicamente, por defecto están en decimal, pero también podemos introducirlos en binario o hexadecimal de la siguiente forma:

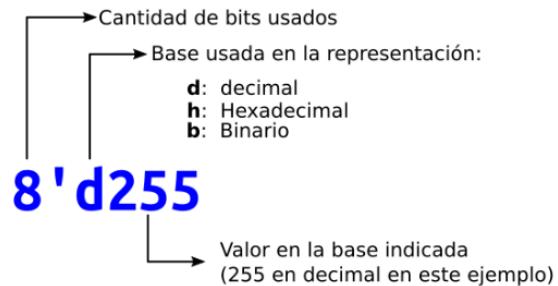


Ilustración 71. Nomenclatura

En el siguiente ejemplo, vemos 4 formas diferentes de encender los tres LEDs superiores de la plaza Alhambra II.

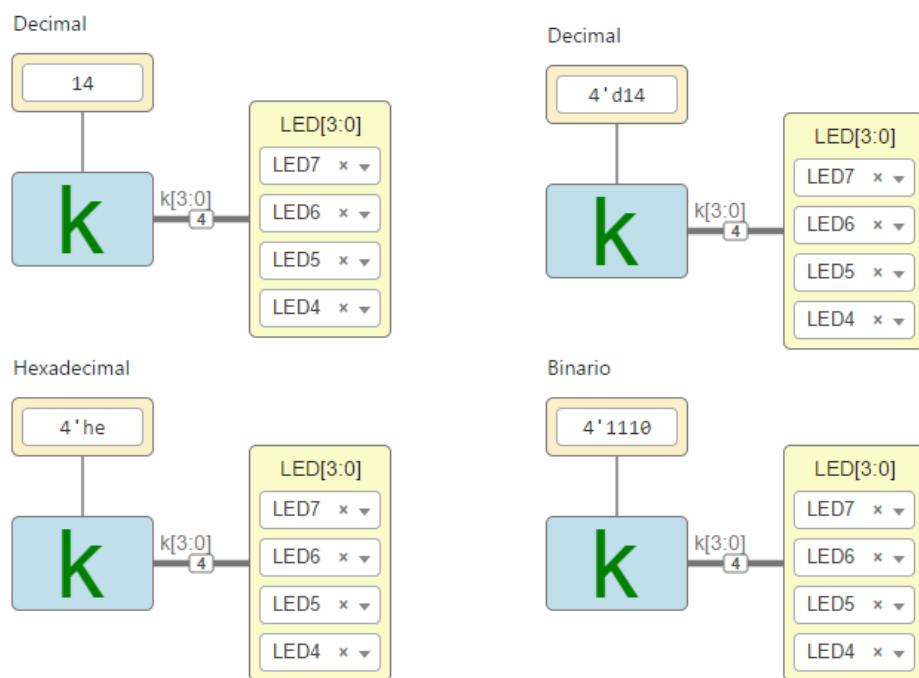


Ilustración 72. Misma función con distinta nomenclatura

16.3. Separadores y agregadores de bus

Como ya hemos visto en algún ejemplo de apartados anteriores, es posible que en algún momento nos haga falta unir dos o varios cables en un mismo bus o, al contrario, separar un bus en diferentes cables. Para ello, contamos con los bloques separadores y agregadores en **Varios/Bus**. La colección dispone de los agregadores y separadores más comunes, en caso de necesitar algún otro, se pueden combinar varios de los disponibles o crear uno diferente modificando alguno ya existente.

Vamos a probar ambos bloques separando un bus cualquiera y volviendo a unirlo, como en el siguiente ejemplo:

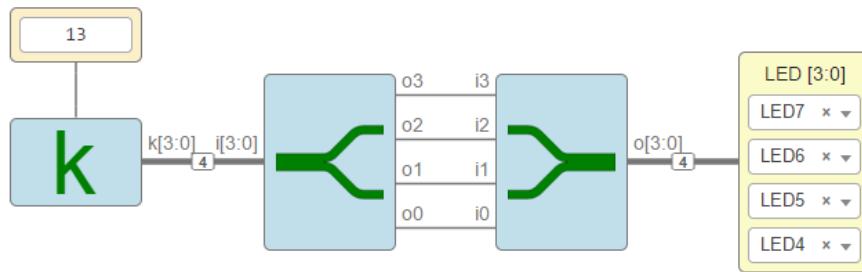


Ilustración 73. Separador y agregador de 4 bits

Una de las aplicaciones más usadas de separadores y agregadores es la de encriptación de datos cambiando el orden de los bits de un bus. Por ejemplo, por un lado, enviamos un valor de los LEDs 7, 6, 5 y 4 de la placa y, por otro lado, mediante estos bloques, intercambiamos los dos bits de mayor y menos peso y sacamos el valor por los LEDs 3, 2, 1 y 0.

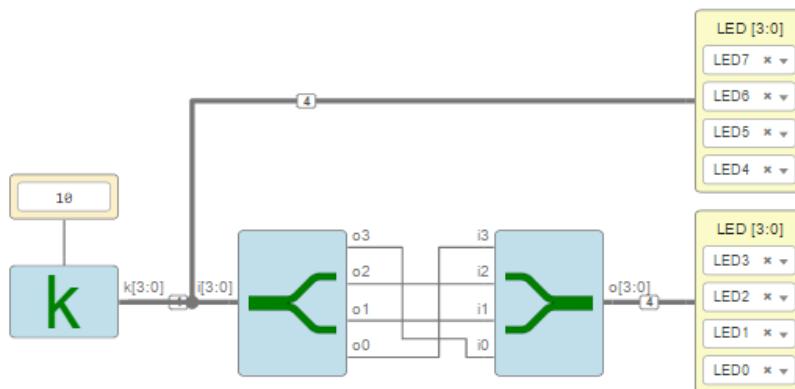


Ilustración 74. Encriptador de 4 bits

SE RECOMIENDA HACER LOS EJERCICIOS 16.1, 16.2 y 16.3 DE LA COLECCIÓN

17. Creando bloques

En este apartado aprenderemos a crear bloques a partir de los bloques que ya conocemos. Las entradas y salidas de los bloques son conocidas como puertos y cada bloque puede tener uno o varios. Cada puerto tiene un nombre (opcional cuando es solo 1 bit) y un tamaño en bits. Además de los puertos, los bloques también pueden tener parámetros en la parte superior.

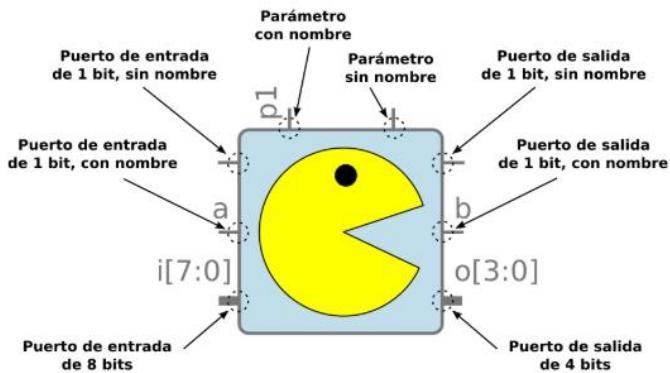


Ilustración 75. Bloque comecocos

Los pines son los puntos físicos por donde entra la información y por donde se mandan las respuestas. Cada uno de estos pines está asociado a una pata de la FPGA. Estos pines pueden ser de un bit, que se conectan al bloque mediante cables, o puede ser de más de un bit, entonces, los conectamos al bloque mediante un bus.

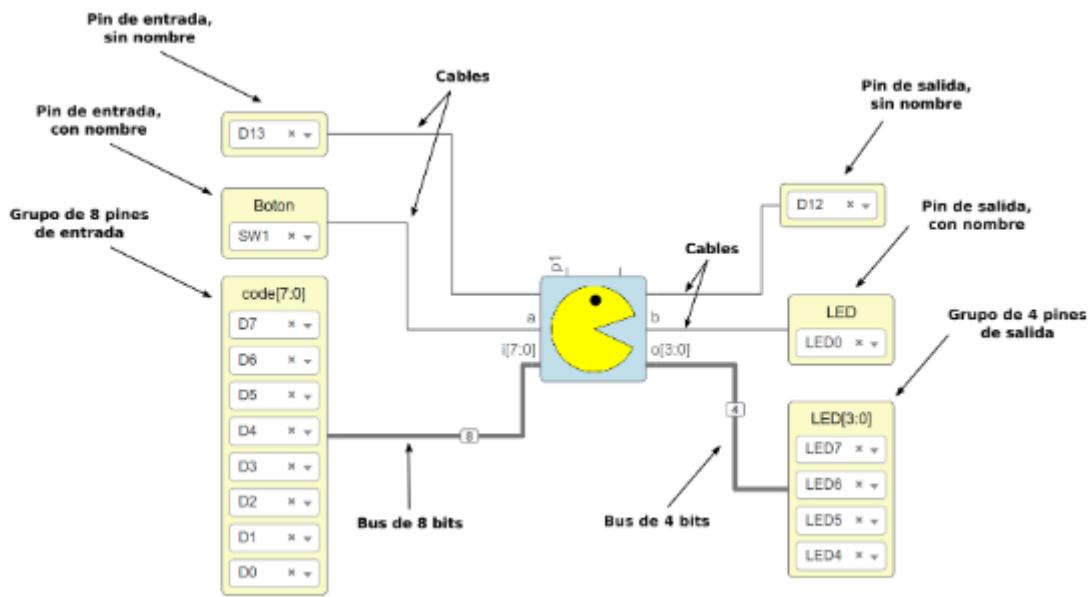


Ilustración 76. Bloque comecocos con pines de entrada y salida conectados

17.1. Diseño jerárquico

Combinando bloques y asignando valores a sus parámetros, construimos nuevos bloques, creando jerarquías. Si, por ejemplo, hacemos doble clic en el bloque “ServoDosPosiciones”, podremos ver por qué bloques está formado.

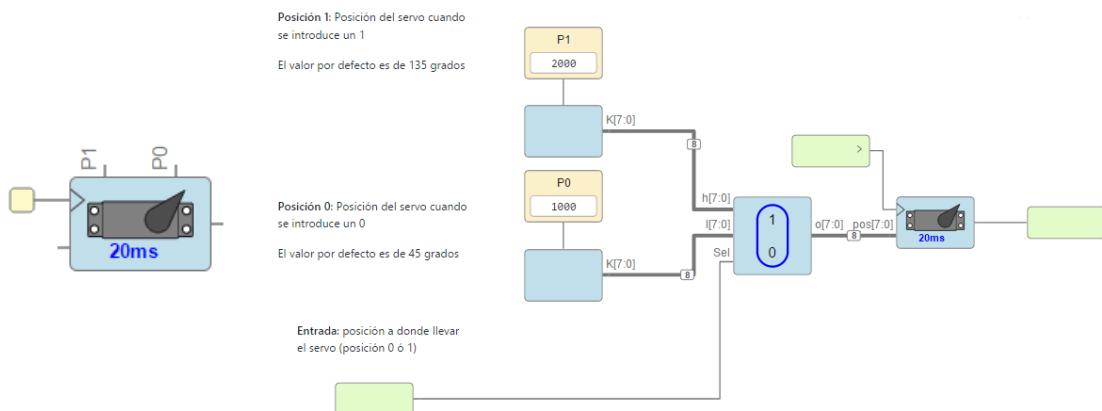


Ilustración 77. Jerarquía del bloque ServoDosPosiciones

17.2. Creando el primer bloque

Vamos a crear el primer bloque, que solo contendrá texto:

1. Abrimos Icestudio en blanco.
2. Vamos a **Editar/Información del proyecto**
3. Le damos un nombre al bloque, podemos también añadir una pequeña descripción, nombre del autor e incluso una imagen en .SVG para que el bloque sea más visual.
4. Guardamos el bloque en **Archivo/Guardar como...**
5. Abrimos un Icestudio en blanco de nuevo y vamos a **Archivo/Añadir como bloque...**
6. Seleccionamos el bloque guardado en el paso 4 y lo colocamos

Name	<input type="text" value="Primer bloque"/>
Version	<input type="text"/>
Description	<input type="text" value="Primer bloque creado en el curso La Salle"/>
Author	<input type="text" value="La Salle Andoain"/>
Image	<input type="text"/>

Open SVG Save SVG Reset SVG

Ilustración 78. Parámetros del bloque



Ilustración 79. Bloque creado

Si hacemos doble clic en este bloque, se abre un documento en blanco ya que no le hemos añadido nada.

17.3. Bloques con puertos

Hasta ahora los bloques de 1 bit son los que más hemos estado usando así que empezaremos creando uno de estos. Vamos a crear la puerta AND de 3 entradas siguiendo los pasos que hemos realizado en el anterior ejemplo.

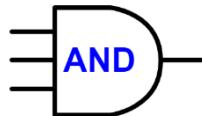


Ilustración 80. Puerta AND de 3 entradas

Esta vez añadiremos puertos en el Icestudio en blanco. Vamos a **Básico/Entradas** y seleccionamos una entrada, esta vez desmarcaremos la opción **FPGA pin** y le damos a **OK**. Aparecerá el siguiente bloque verde que representa un puerto de entrada. Añadimos otros dos para tener las 3 entradas. Hacemos lo mismo con el puerto de salida.



Ilustración 81. Puertos de entrada y salida añadidos

Hay varias maneras de implementar una puerta AND de 3 entradas: Circuito combinacional, con una tabla de la verdad... Nosotros la implementaremos usando puertas AND de 2 entradas que ya hemos visto:

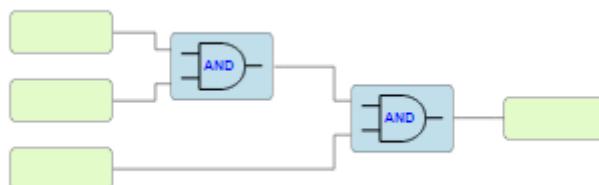


Ilustración 82. Programación interna de puerta AND de 3 entradas

Ahora, editamos la información del bloque en **Editar/Información del proyecto**. Podemos encontrar la imagen en la carpeta imágenes de la colección.

Name	<input type="text" value="AND3"/>
Version	<input type="text"/>
Description	<input type="text" value="Puerta AND de 3 entradas"/>
Author	<input type="text" value="La Salle Andoain"/>
Image	

Ilustración 83. Información del bloque

Guardamos el proyecto y lo añadimos como bloque en otra página en blanco.

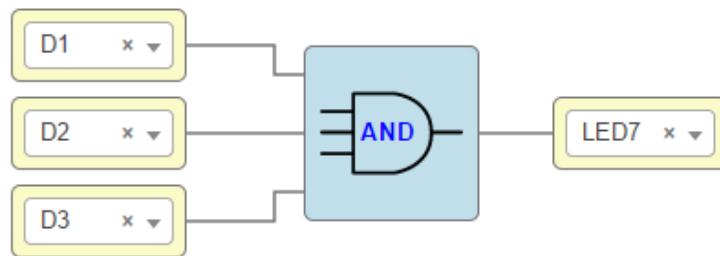


Ilustración 84. Entradas y salidas conectadas al bloque

Para realizar la prueba añadimos 3 interruptores externos y un LED de la Alhambra que no se encenderá hasta que los 3 interruptores estén activados.

De la misma manera, podemos crear bloques de más de 1 bit, por ejemplo, vamos a crear un bloque que detecte el numero 2 (en binario 10) y encienda un LED. Tenemos que tener en cuenta que, al crear la entrada, esta tiene que ser de 2 bits, por lo que hay que añadirle un nombre y [1:0].

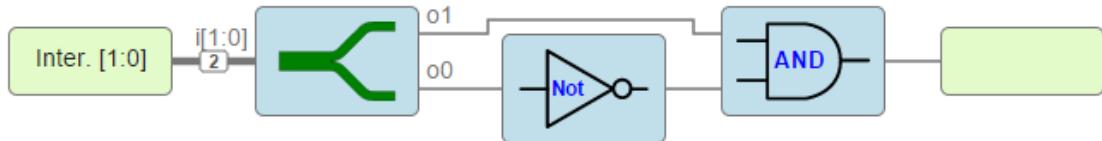


Ilustración 85. Programación interna del bloque 2?

Editamos la información, guardamos el bloque y lo añadimos en otro proyecto en blanco.

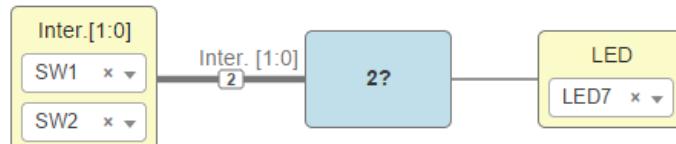


Ilustración 86. Conexiones del bloque 2?

17.4. Bloque con entradas de reloj

Algunos bloques tienen una entrada de reloj que más adelante veremos para que sirve, pero ahora vamos a aprender cómo hacer bloques con entrada de reloj. Usaremos como ejemplo el bloque Corazón. Como podemos ver, a la izquierda del bloque hay un puerto especial, el reloj del sistema:

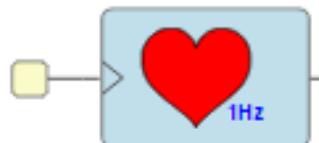


Ilustración 87. Bloque corazón con entrada de reloj

Vamos a crear un bloque usando este como base. Para ello, creamos una entrada con la opción “FGPA pin” desmarcada y con la opción “Show Clock” marcada y conectamos esta entrada mediante un cable a la entrada de reloj del corazón. Añadimos otra entrada para un pulsador, una puerta AND y una salida de la siguiente forma para probar el bloque:

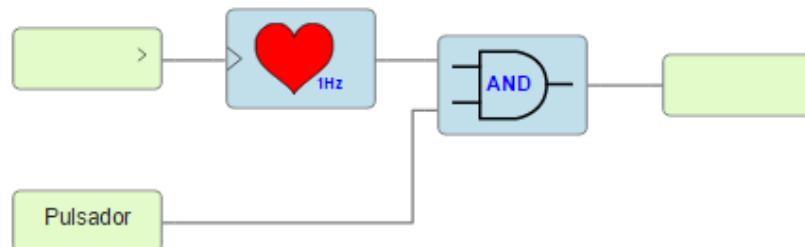


Ilustración 88. Programación interna de bloque Corazón AND

Editamos la información, guardamos y añadimos como un bloque, seleccionamos la entrada y la salida y probamos.



Ilustración 89. Bloque Corazón AND con entradas y salidas conectadas

17.5. Bloque con parámetros

Como ya hemos comentado anteriormente, además de entrada y salidas, los bloques pueden tener parámetros. Para crear un bloque con parámetros, simplemente hay que usar otros bloques paramétricos y colocar el contenedor (Constante o Memory) correspondiente.

Por ejemplo, si hacemos doble clic en el ServoDosPosiciones, podemos observar cómo está formado por el propio servo y un multiplexor al que se le introducen dos posibles parámetros mediante el bloque Constante. Al tener esos dos bloques constantes dentro (con sus valores por defecto), el bloque final también tiene dos pines para parámetros.

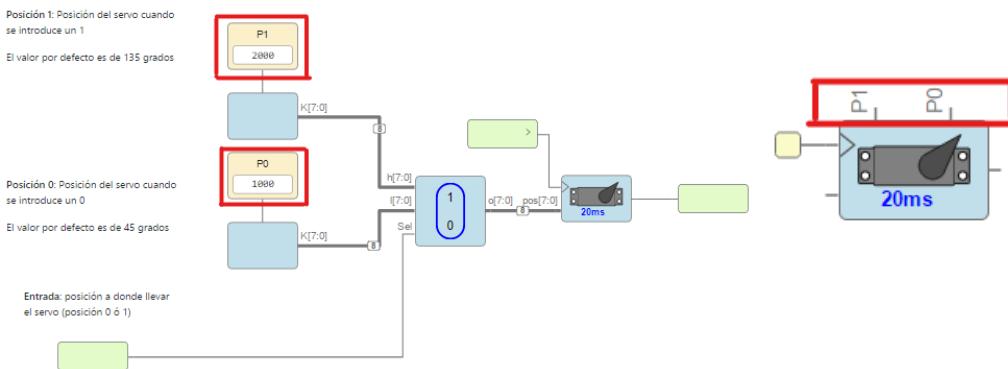


Ilustración 90. Parámetros del bloque ServoDosPosiciones

SE RECOMIENDA HACER LOS EJERCICIOS 17.1 Y 17.2 DE LA COLECCIÓN

18. Circuitos combinacionales con varias salidas

Ya hemos visto que son los circuitos combinacionales, pero todos los ejemplos/ejercicios que hemos visto constan de una sola salida. En este apartado veremos algunos ejemplos de circuitos combinacionales con varias salidas. Las tablas necesarias para este apartado las tenemos en **Combi/Tablas** junto con las vistas anteriormente.

18.1. Circuitos combinacionales con 1 entrada

Empezamos por lo más sencillo, un circuito de 1 entrada y 2 salidas. Al tener solo 1 entrada, tenemos dos posibles datos, 1 y 0, por lo tanto, no podremos tener en la salida todas las combinaciones posibles de 2 bits, habrá que elegir solo las dos deseadas.

Comenzamos diseñando un circuito que haga parpadear dos LEDs de forma alterna, mientras uno está apagado, el otro está encendido.

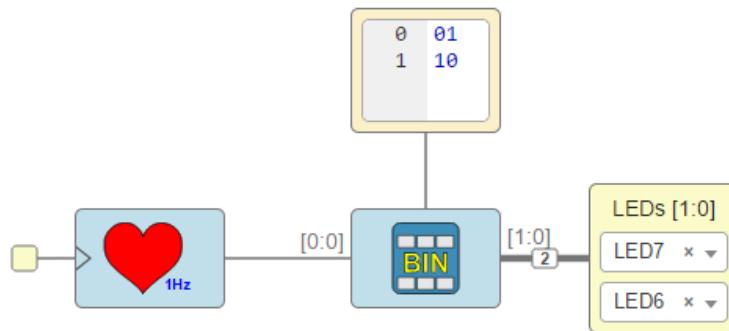


Ilustración 91. Circuito combinacional de 1 entrada con tabla

Modificando los valores de la memoria podemos conseguir distintas aplicaciones, por ejemplo, si queremos controlar los servos de un robot:

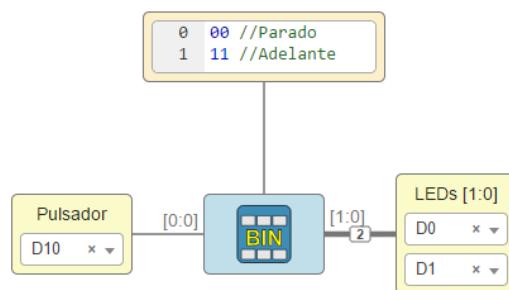


Ilustración 92. Circuito combinacional para control de un servo

Podemos hacer lo mismo para salidas de más bits. Esta vez, en vez de encender 2 bits, hacemos que parpadeen alternadamente los 8 LEDs de la placa Alhambra.

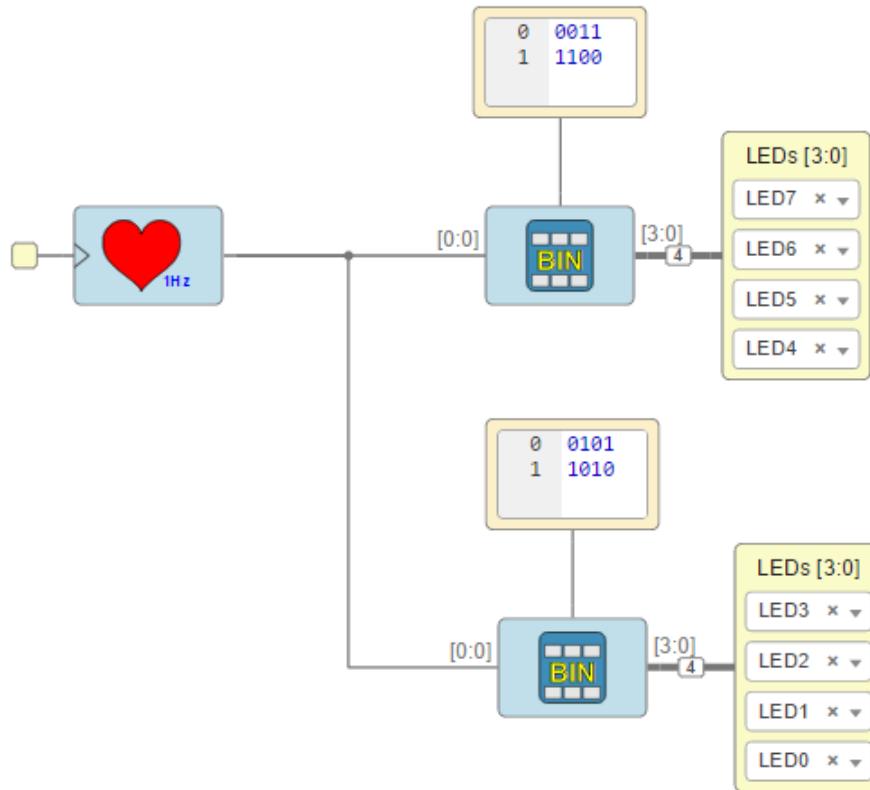


Ilustración 93. Circuito combinacional de 4 salidas doble

Podemos conseguir exactamente lo mismo usando una sola tabla de 8 bits de salida. O incluso poder seleccionar entre dos estados de los LEDs diferentes mediante un multiplexor.

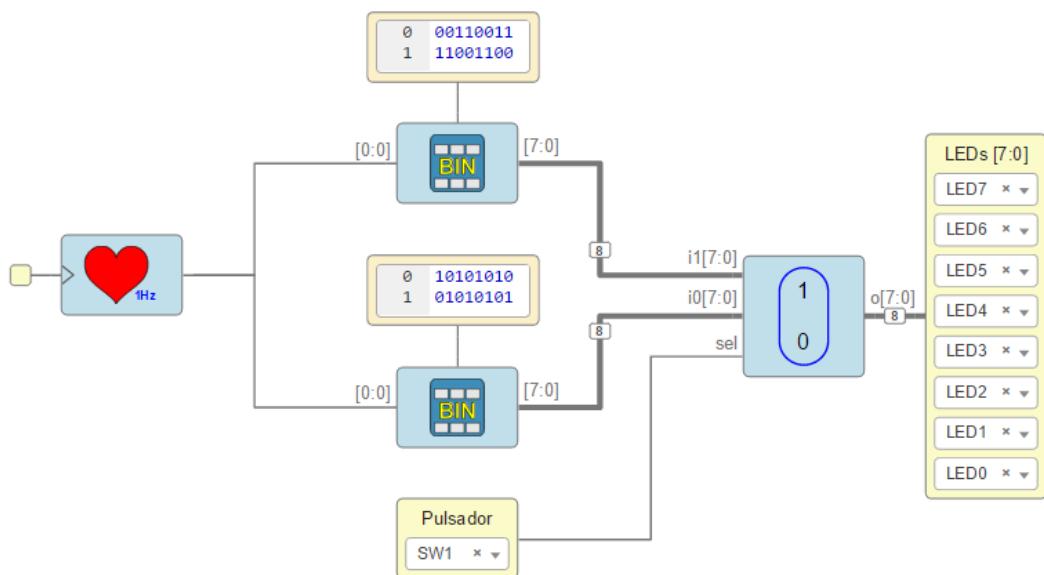


Ilustración 94. Circuito combinacional con 8 salidas y multiplexor

18.2. Circuitos combinacionales con 2 entradas

A más entradas, más combinaciones, esta vez tenemos una tabla de la verdad de 4 filas. Y, por ejemplo, se puede mejorar el control del robot visto anteriormente.

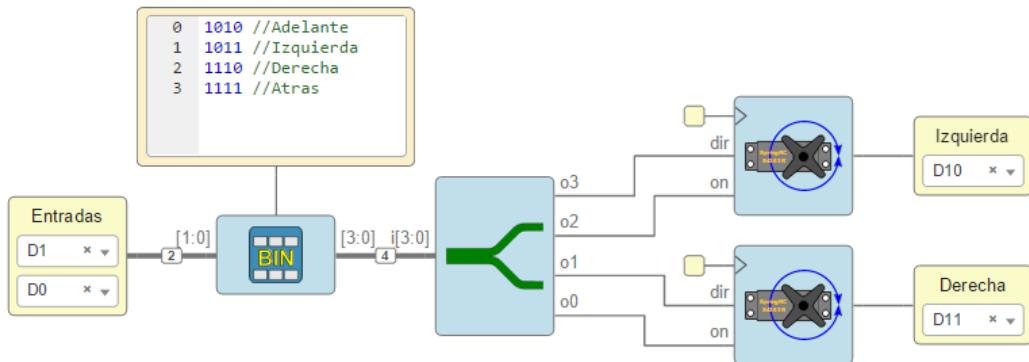


Ilustración 95. Circuito combinacional con 2 entradas

18.3. Circuitos combinacionales con 3 entradas

Como has visto, se repite el formato, solo hay que tener en cuenta el número de entradas y salidas que necesitas. Para este último ejemplo vamos a crear un circuito combinacional de 3 entradas y 8 salidas que usamos como decodificador. La entrada recibe un numero entre el 0 y 7, y activará exactamente esa salida, y solo esa.

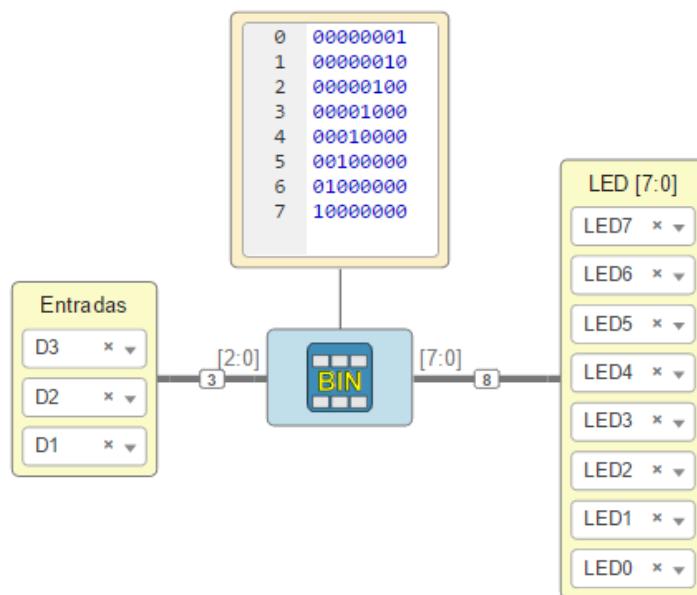
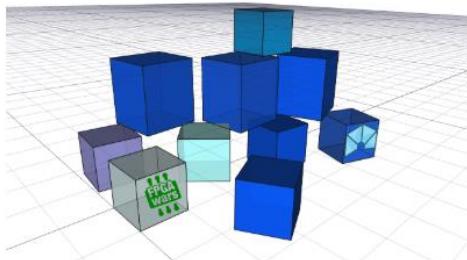


Ilustración 96. Circuito combinacional con 3 entradas

18.4. Generador de tablas: IceFactory

¿Y qué pasa si necesitamos una tabla que no esté ya creada en la colección? Pues la podemos crear fácilmente gracias a [IceFactory](#).



Generación de bloques para IceStudio

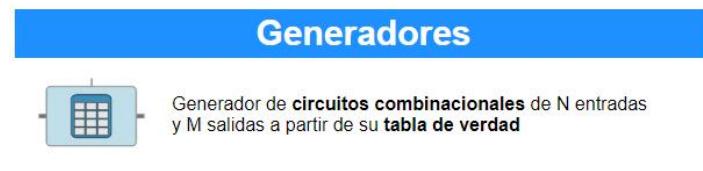


Ilustración 97. Portada de IceFactory

Hacemos clic en el icono de la tabla que nos introduce al paso a paso para crear la tabla. Seleccionamos el número de bits de entrada y de salida deseados y hacemos clic en Generar.

Paso 1: Introduce los Bits de entrada y salida

Bits de entrada: Bits de salida:

Paso 2: Introduce el formato de los datos

Binario Hexadecimal

Paso 3: Genera el componente

Generar

Paso 4: Descarga el componente

[Descargar fichero: tabla-bin-3-5.ice](#)

Ilustración 98. Parámetros de la tabla

Hacemos clic en Descargar fichero. Lo guardamos como los bloques que creamos en apartados anteriores y ya estaría disponible para añadir como

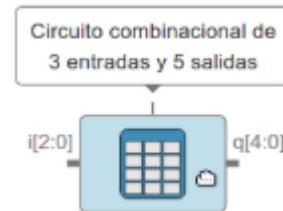


Ilustración 99. Bloque creado con IceFactory

SE RECOMIENDA HACER LOS EJERCICIOS 18.1 Y 18.2 DE LA COLECCIÓN

19. La colección

La colección con la que llevamos trabajando todo el curso es “CursoFPGA”, pero hay muchas otras creadas por otras personas, o incluso cada uno puede crear la suya propia. Pero también se pueden modificar o añadir bloques, ejemplos o ejercicios a las ya existentes. En este apartado veremos cómo realizar dichos cambios.

Las colecciones añadidas a Icestudio se guardan en una carpeta específica del disco local. Si vamos a la carpeta de usuario del PC, vemos cómo hay una carpeta llamada `.icestudio`, dentro de ella vemos las distintas carpetas del programa. Si hacemos doble clic en `collections`, vemos las distintas colecciones que tenemos añadidas al Icestudio. Nos centraremos en los archivos que hay dentro de la colección “La Salle Andoain”:

- **package.json**: Es un elemento obligatorio que posee información sobre la colección.
- **Blocks**: Carpeta con los bloques de la colección. Su contenido es el que se muestra en la esquina superior derecha del icestudio.
- **Examples**: Carpeta con los ejemplos y ejercicios del curso.
- **Locale**: Carpeta con las traducciones a diferentes idiomas.
- **Readme.md**: Información de la colección (para humanos). Su contenido se muestra en [Ver/Información de la colección](#).

Para modificar los bloques, lo haremos desde **Archivo/Bloques** para que el bloque se guarde correctamente una vez modificado.

Se pueden añadir bloques en nuevas carpetas o subcarpetas, modificar ejemplos, resolver nuevos ejercicios... pero en todo caso, habrá que recargar la colección en **Herramientas/Colecciones/Recargar** para que los cambios se hagan efectivos.

20. Display 7 segmentos

Los displays de 7 segmentos permiten que, trabajando en binario, veamos la información en decimales. Están formados por circuitos combinacionales creados a partir de tablas de la verdad y se dividen en 8 segmentos, 7 de forma alargada y 1 para representar el punto. Cada uno de ellos es un LED independiente y tiene un nombre diferente para poder diferenciarlos. Con un solo display podemos visualizar un dígito del 0 al 9, así que añadiendo más displays podemos formar números mayores.

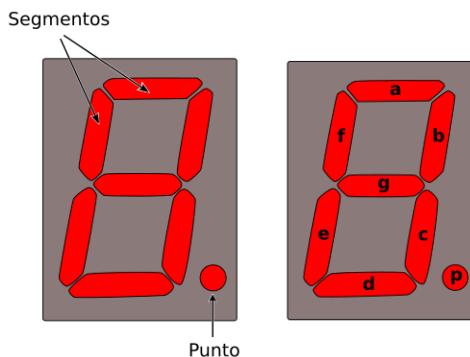


Ilustración 100. LEDs del Display 7 segmentos

Cada uno de estos segmentos debe ir conectado a un pin de la Alhambra para que pueda ser controlado, y el pin común del display (Vcc o GND), lo conectamos también a la Alhambra.

Vamos a probar a encender alguno de estos segmentos para probar su funcionamiento. Mandamos un bit a los segmentos a y d y observamos cómo se encienden el segmento inferior y superior del display.

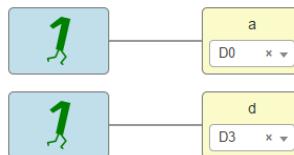


Ilustración 101. Encendido de 2 LEDs del display

A continuación, hacemos lo mismo con todos los segmentos para probar que funciona correctamente.

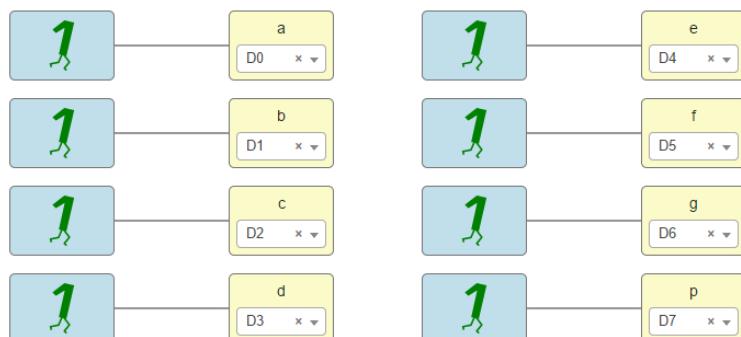


Ilustración 102. Encendido de todos los LEDs del display

Estos serían los valores en binario necesarios para cada dígito:

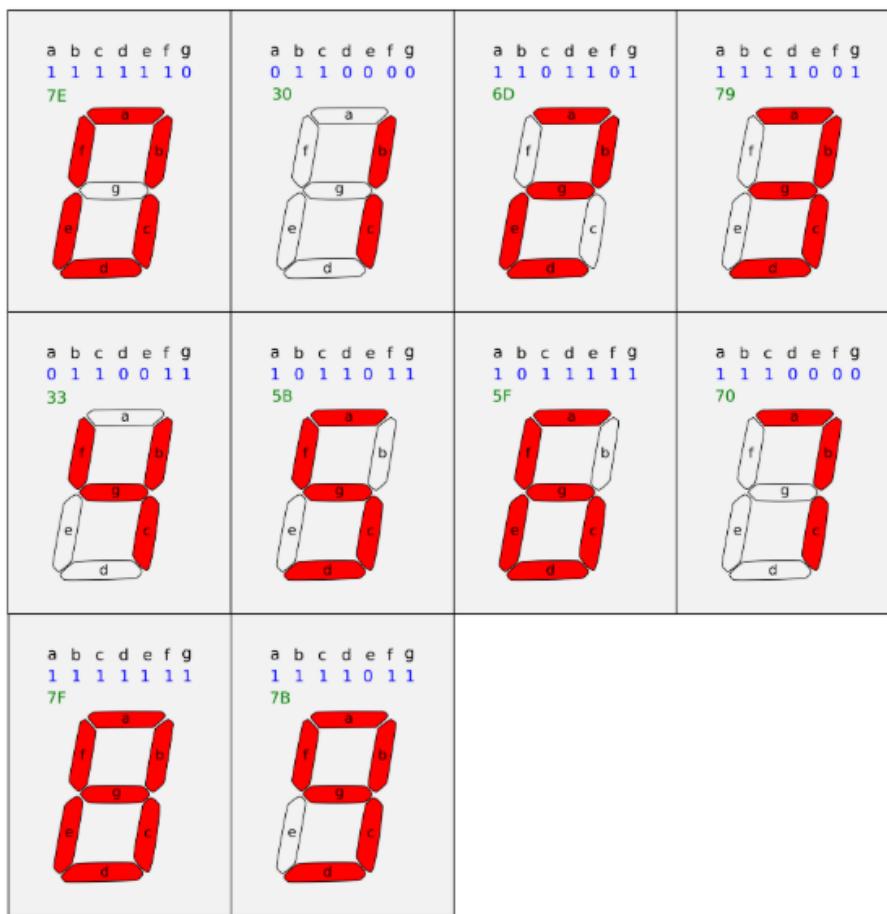


Ilustración 103. Valores necesarios para visualizar dígitos en el display

Tenemos bloques constantes creados en **Const/7Seg/AnodoComun** o **CatodoComun** para facilitar el trabajo, también tenemos letras. Vamos a probar alguno de ellos. Usamos un multiplexor para elegir entre dos constantes diferentes.

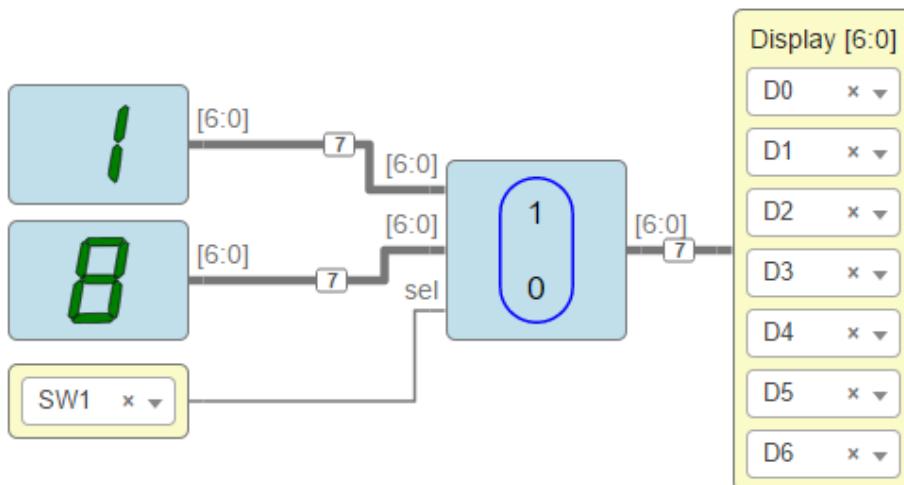


Ilustración 104. Visualización de 2 dígitos en el display

Este mismo circuito se puede realizar con una tabla de la verdad. Esta vez, la entrada será un sensor infrarrojo, por lo tanto, en el display visualizaremos un 0 cuando el IR no detecte nada mientras que veremos un 1 cuando se detecte un objeto.

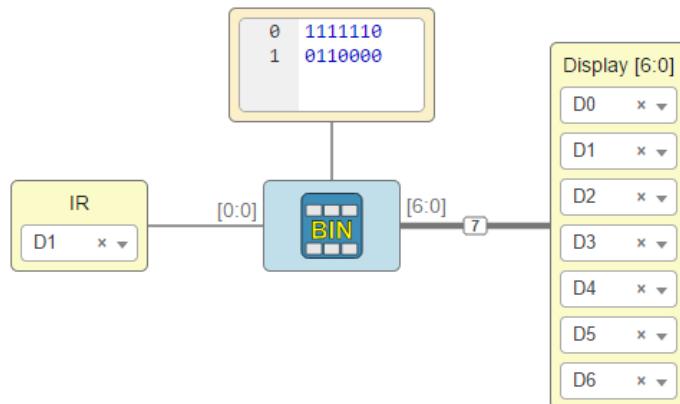


Ilustración 105. Visualización de dígito implementado mediante tabla de la verdad

Estos displays se suelen usar para contar, por lo que vamos a montar una cuenta cíclica de 0 a 3. Usamos una tabla de 2 entradas para ello haciendo que la cuenta sea automática mediante 2 corazones.

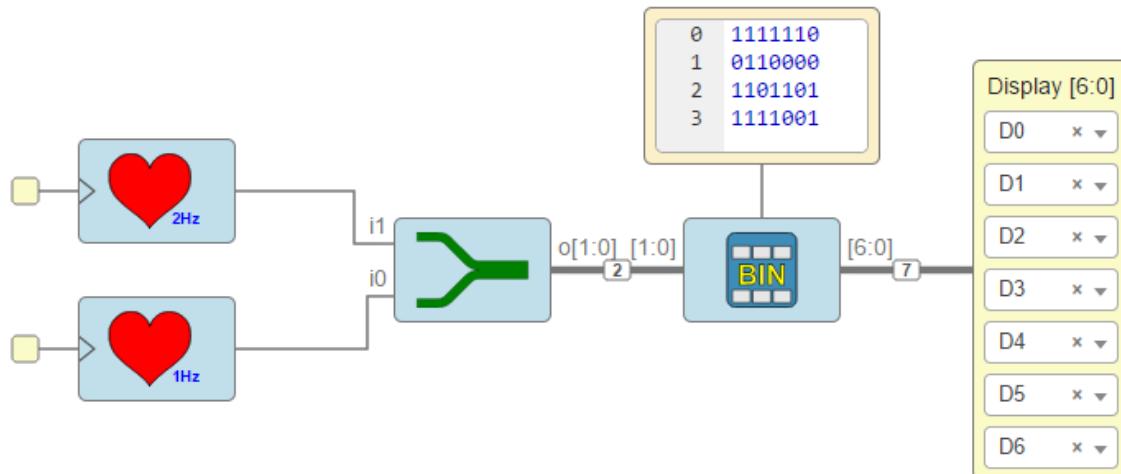


Ilustración 106. Cuenta cíclica de 0 a 3 usando una tabla de la verdad

Otro de los usos que le podemos dar al display es el de conversor de binario a decimal. Para este ejemplo, usamos 3 interruptores para introducir un dato en binario y mediante una tabla lo podremos ver en decimal gracias al display 7 segmentos.

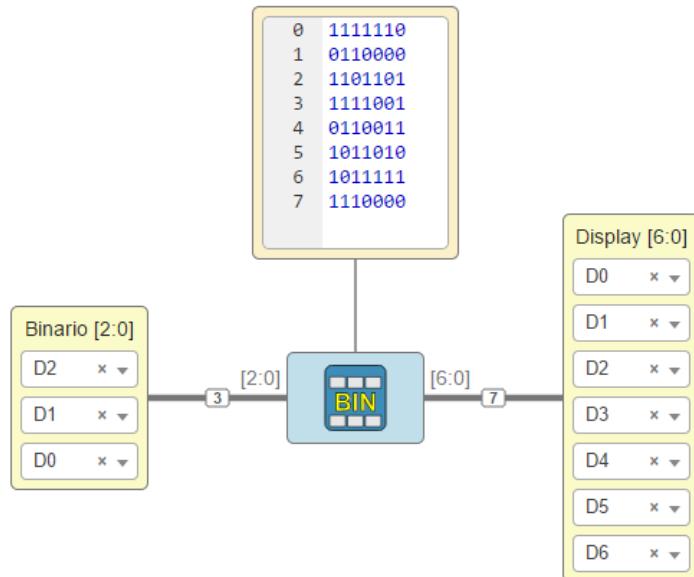


Ilustración 107. Traducción binario decimal mediante una tabla de la verdad

SE RECOMIENDA HACER LOS EJERCICIOS 20.1 Y 20.2 DE LA COLECCIÓN

21. Biestables y notificaciones

En este apartado vamos a conocer el componente usado para almacenar los bits: el biestable. Estos elementos son capaces de recordar eventos ocurridos en el pasado mediante los cuales podemos tomar decisiones en el presente o en el futuro.

Los biestables son los encargados de guardar las notificaciones de lo que ocurre y cómo solamente es capaz de almacenar 1 bit, los usamos para registrar sólo 1 notificación. Cualquier cambio que ocurre en nuestro circuito es llamado evento, y tenemos dos tipos: flanco de subida cuando el bit cambia de 0 a 1 y flanco de bajada cuando pasa de 1 a 0.

21.1. Biestables Set-Reset

Para recordar cosas lo primero que hay que hacer es registrarlas, y la operación SET es la que se encarga de ello. La segunda operación es el borrado de las notificaciones, el RESET. Además, tenemos la salida que nos indica la notificación. Si está a 0 no ha ocurrido el evento, pero si está a 1 es que hay una notificación pendiente.

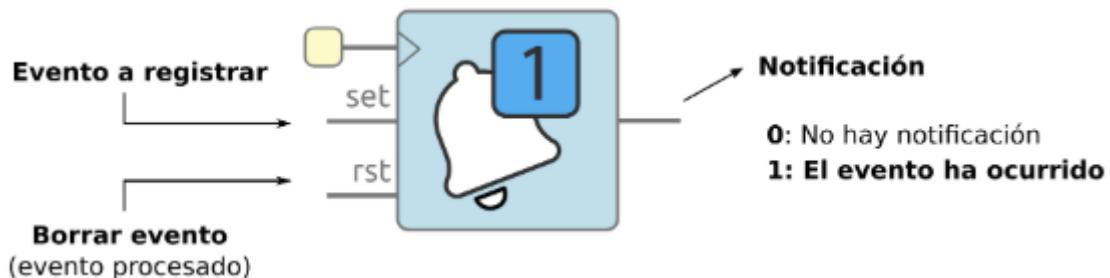


Ilustración 108. Funcionamiento Biestable Set-Reset

Vamos a ver la diferencia entre usar un biesstable y no con un ejercicio simple: el encendido de un LED. Con este primer circuito, observamos como el LED se mantiene encendido solamente cuando tenemos pulsado el botón de la Alhambra. No afecta el estado anterior, solo al presente.



Ilustración 109. Pulsador de entrada y LED de salida

Si, en cambio, añadimos un biestable, conseguimos registrar que el botón (SW1) ha sido pulsado y se enciende el LED. Si el LED está apagado, significa que el botón no ha sido pulsado desde el ultimo borrado de la memoria.

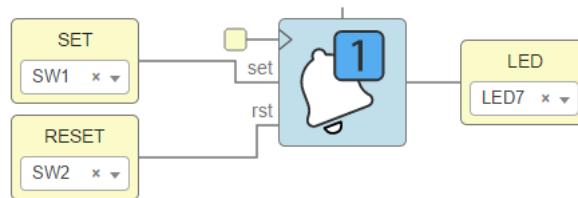


Ilustración 110. Encendido de un LED mediante Biestable

Vamos a darle un uso más práctico al biestable programando la subida de la barrera al llegar un coche a la entrada de un aparcamiento. Tenemos dos sensores, uno que notifica la llegada del vehículo y otro que el vehículo ya está dentro del recinto. Usamos estos datos para modificar la posición de un servo que simula la barrera. Cuando el primer sensor detecta la llegada del vehículo, se abre la barrera y no se cierra hasta que el segundo sensor detecta el vehículo.

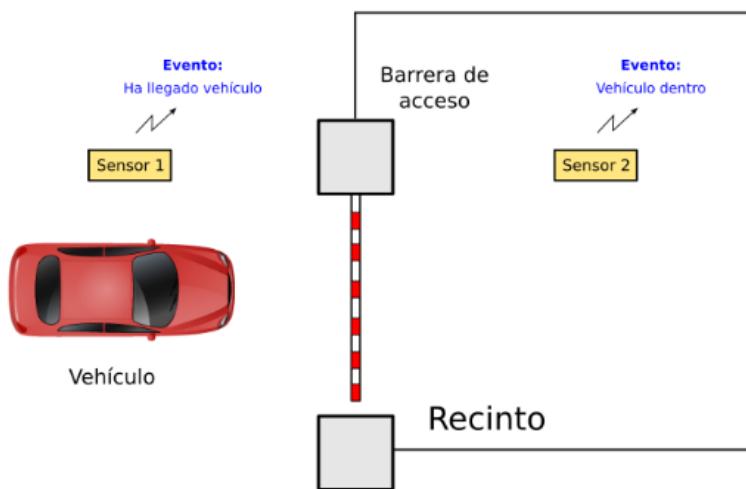


Ilustración 111. Esquema del ejercicio

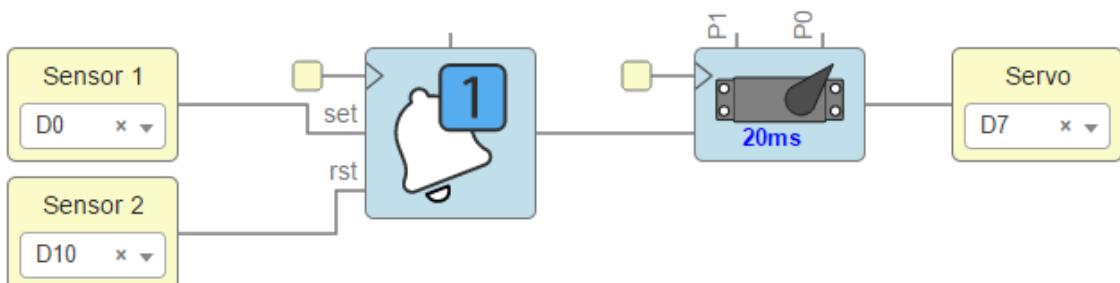


Ilustración 112. Programación del ejercicio usando biestable

SE RECOMIENDA HACER LOS EJERCICIOS 21.1 Y 21.2 DE LA COLECCIÓN

22. Tics, tiempo y temporizadores

Los circuitos tienen noción del tiempo gracias al reloj del sistema, el cual podemos usar para crear temporizadores que miden el tiempo, realizan acciones o propagan eventos.

22.1. Reloj del sistema

La Alhambra posee un reloj situado fuera de la FPGA que se conecta a uno de sus pines. En Icestudio, los componentes que tienen una entrada de reloj se conectan automáticamente a ese reloj de la placa.

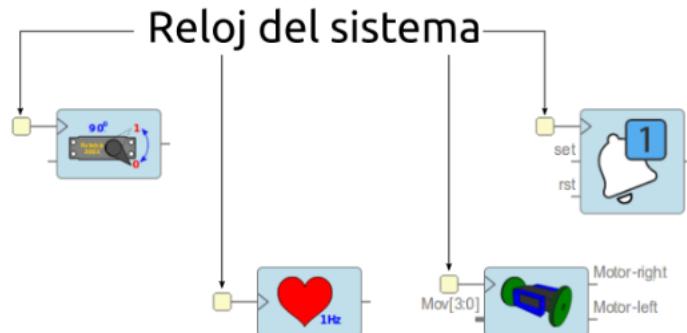


Ilustración 113. Reloj del sistema en distintos bloques

A pesar de que estos bloques se conectan automáticamente al reloj, este sigue siendo una entrada como cualquier otra, por lo que podemos conectarlo manualmente.

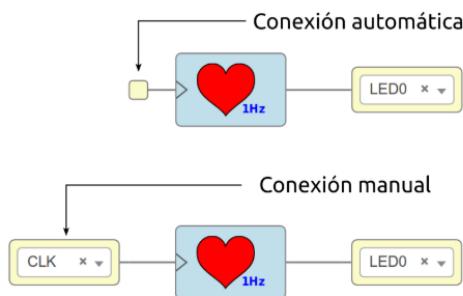


Ilustración 114. Conexión automático y manual del reloj

La señal del reloj la representamos mediante una señal cuadrada. Como ya hemos mencionado en otro apartado, el paso de 0 a 1 se llama flanco de subida y es el evento que se usa como referencia.

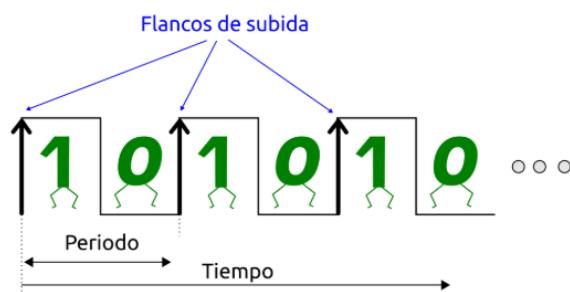


Ilustración 115. Señal del reloj del sistema

22.2. Pulso

Cuando una señal pasa de estar a 0 a estar activa durante un tiempo y vuelve al reposo, decimos que se ha producido un pulso y el tiempo que permanece a 1 es el ancho de pulso.



Ilustración 116. Ancho de pulso

Cuando un pulso se repite de forma periódica, aunque sea de diferente ancho, se dice que es un pulso periódico o PWM (Pulse-width modulation).

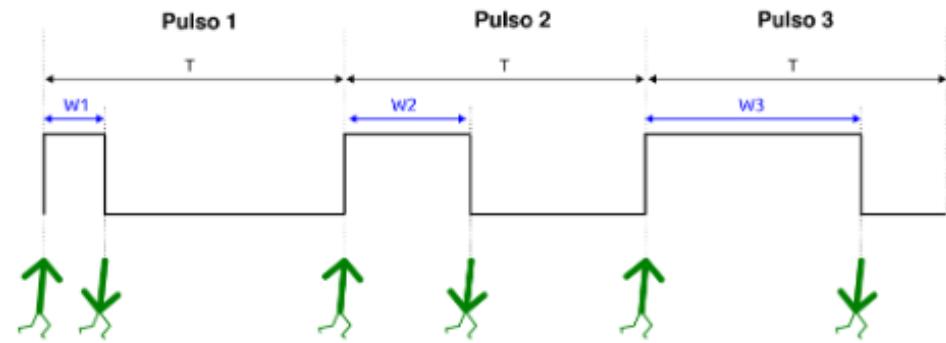


Ilustración 117. Señal PWM

22.3. ¿Qué son los Tics?

El reloj produce pulsos de periodo T y ese periodo es nuestro periodo mínimo, nuestra resolución, esto es, no podremos trabajar con tiempos menores a T . Es como intentar medir milisegundos con un cronómetro que solo marca segundos.

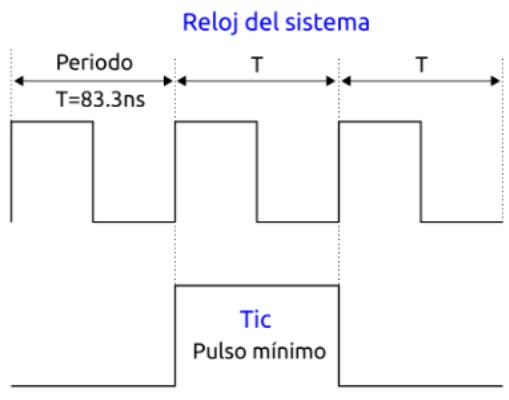


Ilustración 118. Señal Tic

Los tics, por definición, tienen una anchura de 1. En Icestudio tenemos un componente que nos servirá para crear tics en **Varios/Pulsadores**:

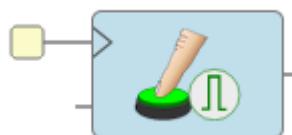


Ilustración 119. Bloque señal Tic con pulsador

Cuando se active la entrada de este bloque, mandará solamente un tic por la salida:

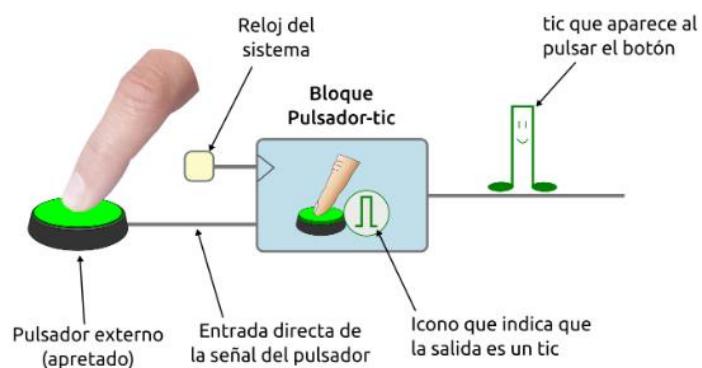


Ilustración 120. Funcionamiento del bloque Tic-Pulsador

Si intentamos encender un LED directamente conectándolo a la salida de este bloque, nunca lo veremos encendido, porque, aunque el LED si se haya encendido, lo hace tan rápidamente que nuestro ojo no lo aprecia. Para poder verlo, añadimos un biestable, ya que este se encarga de guardar el dato que le hayamos mandado.

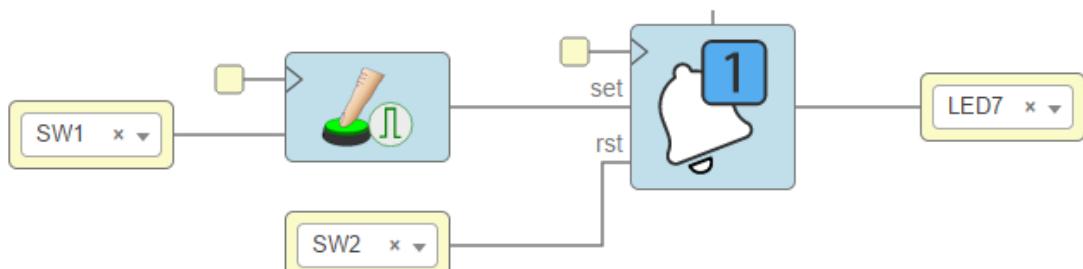


Ilustración 121. Encendido de LED

22.4. Temporizadores

Los temporizadores son circuitos que nos informan cuando ha transcurrido un intervalo de tiempo. El que nosotros vamos a usar tiene 1 entrada y 2 salidas. La entrada start sirve para ponerlo en marcha al enviarle un tic. Conseguimos dos salidas, en la primera conseguimos un pulso de anchura igual al intervalo de tiempo de espera y en la otra, un tic que nos indica que el tiempo ha expirado. En **Varios/Timers** contamos con este temporizador tanto en segundos como en milisegundos y microsegundos.

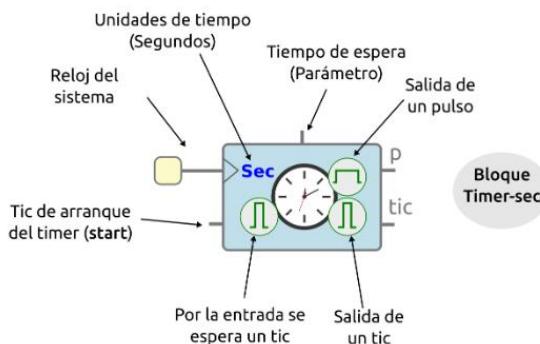


Ilustración 122. Bloque temporizador

Para probar este bloque, vamos a hacer un circuito que encienda un LED durante 3 segundos y luego se apague.

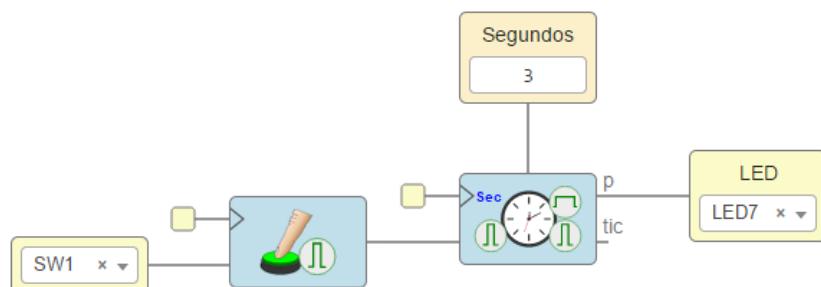


Ilustración 123. Programación de encendido de LED durante 3 segundos

Estos temporizadores se pueden encadenar gracias al tic de salida, vamos a probar esta utilidad haciendo que se encienda un LED durante 400ms y en cuanto acabe este, se encienda el siguiente durante otros 400ms.

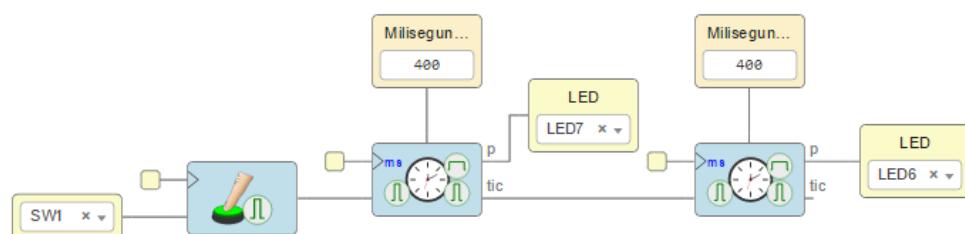


Ilustración 124. Encendido de LEDs usando temporizadores

22.5. Generando pulsos periódicos

Los temporizadores nos permiten crear pulsos periódicos muy fácilmente, solo hay que conectar un corazón de tics a su entrada.

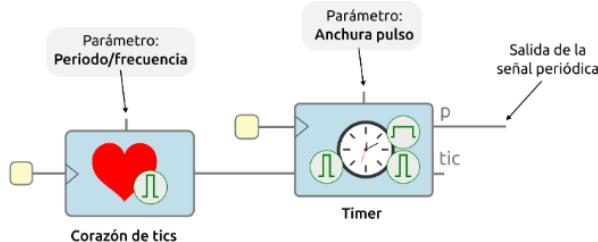


Ilustración 125. Generando pulsos periódicos

El corazón se encarga del periodo de la señal mientras que el temporizador se ocupa del ancho de pulso. Para probar este funcionamiento, vamos a jugar con la intensidad de brillo de un LED.

Al introducir por un LED una señal lo suficientemente rápida, no notamos que parpadea porque nuestro cerebro se queda con el valor medio y lo veremos encendido. Cuando mayor sea la anchura de pulso, más tiempo pasa encendido el LED frente al tiempo que pasa apagado, entonces veremos cómo brilla con más intensidad.

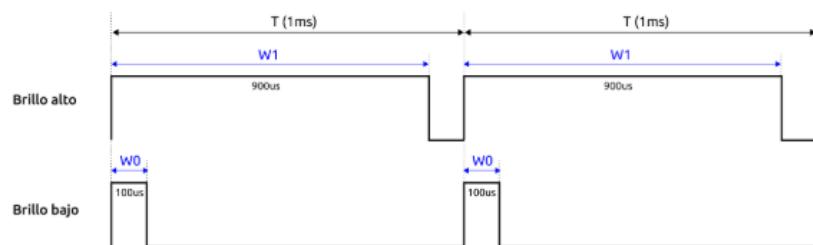


Ilustración 126. Control de brillo con señal PWM

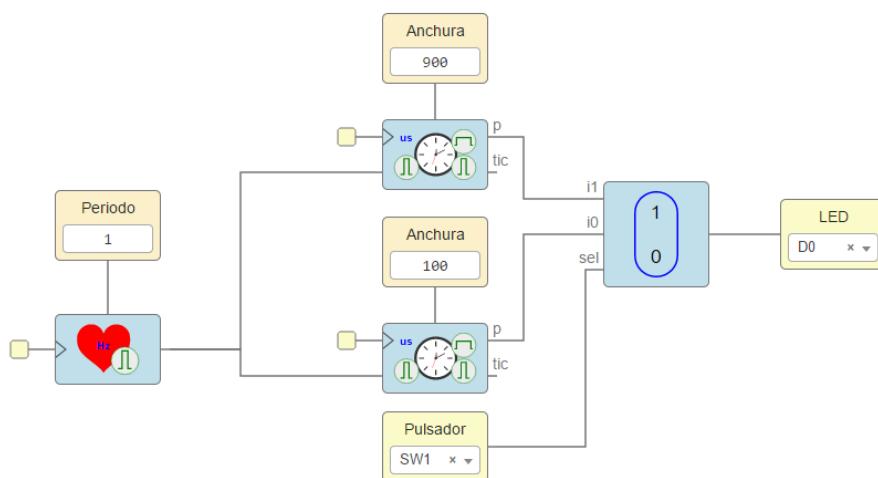


Ilustración 127. Control de brillo de un LED

SE RECOMIENDA HACER LOS EJERCICIOS 22.1, 22.2 Y 22.3 DE LA COLECCIÓN

23. Contadores

Los contadores son unos componentes fundamentales en la electrónica digital. Los usamos para contar eventos, medir tiempo y recorrer tablas. Nos vamos a centrar en los contadores ascendentes. Estos componentes cuentan tics y suman una unidad cada vez que llega un tic. Los tenemos en **Varios/Contadores**

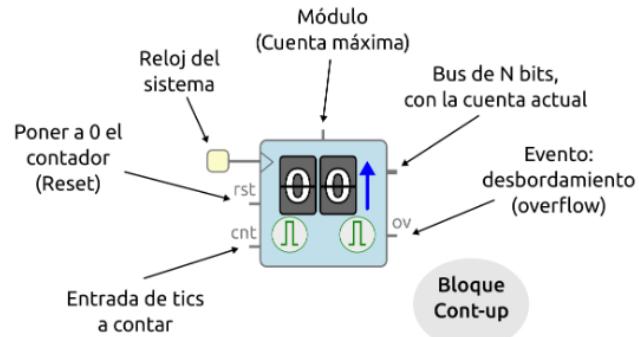


Ilustración 128. Bloque contador

Tenemos 2 entradas además del reloj del sistema, una por donde llegan los tics a contar y otra para resetear el contador; 2 salidas, una con un bus de N bits con la cuenta actual y otra para el **overflow** (cuando el contador pasa del número máximo a 0); y un parámetro donde introducimos el módulo del contador: el número máximo, que una vez alcanzado hace que vuelva a comenzar desde 0 (Overflow).

Vamos a probar un circuito simple con el que contamos las veces que apretamos un pulsador. Vemos la cuenta en un display 7 segmentos y añadimos otro LED para visualizar cuando ocurre el Overflow.

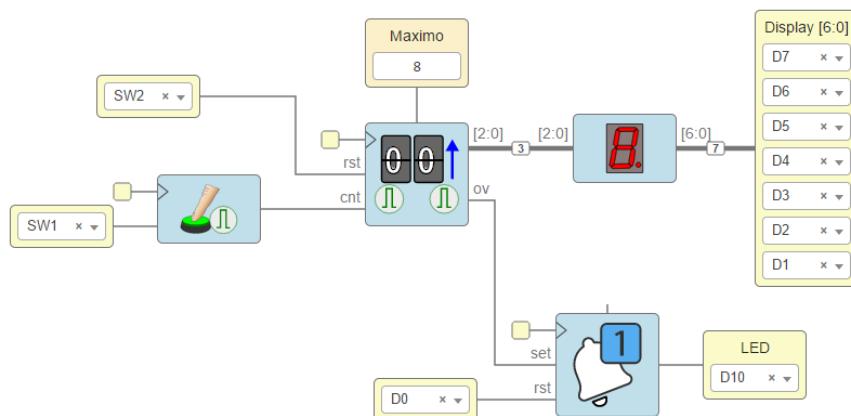


Ilustración 129. Contador de 0 a 8 usando un contador y un display

23.1. Encadenando contadores

Al ocurrir el overflow, se emite un tic por su salida, por lo que esa misma salida puede usarse para la entrada de un contador y así contar las veces que haga overflow el anterior contador. De esta forma podemos añadir cifras a la cuenta.

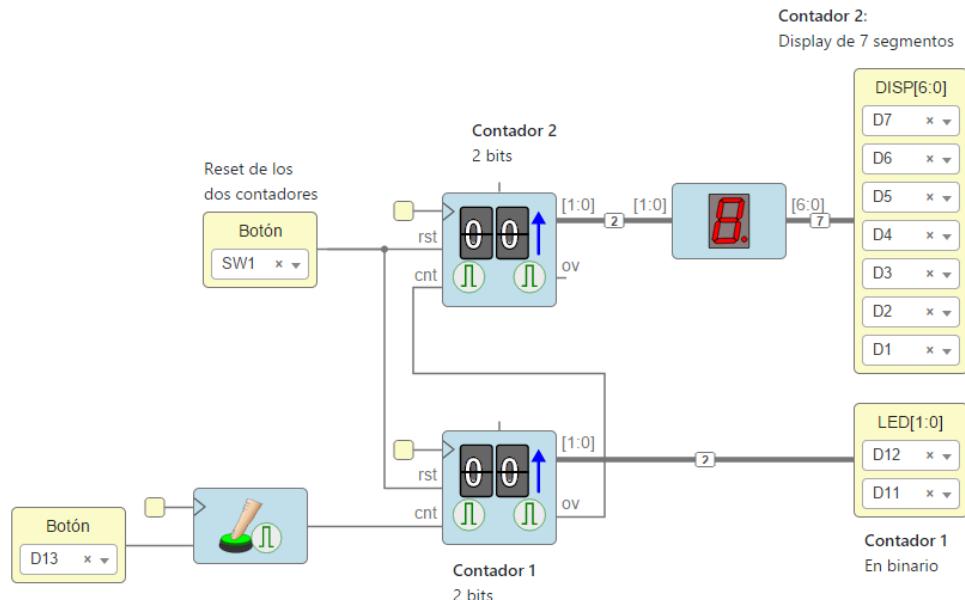


Ilustración 130. Contador de 2 cifras encadenando contadores

Usamos dos LEDs para mostrar en binario la cuenta del primer contador y un display para la cuenta del segundo contador.

23.2. Contando tiempo

Los generadores de tics como los corazones, emiten un tic cada vez que haya pasado un tiempo T: Mediante un contador podemos contar cuantos tics han pasado y así medir el tiempo transcurrido. Vamos a probar a crear un circuito simple que cronometre los segundos transcurridos.

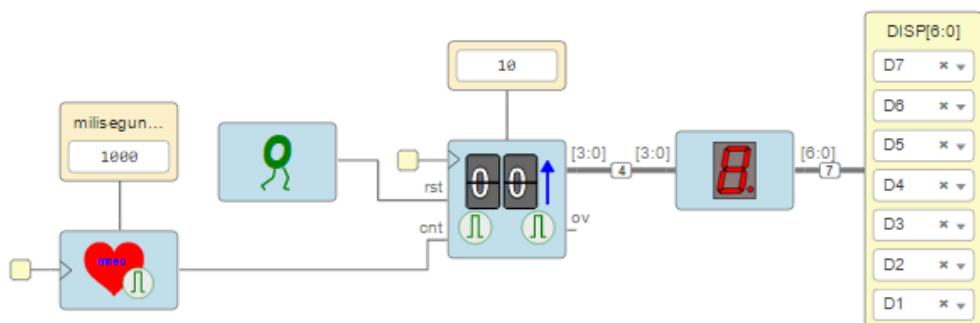


Ilustración 131. Contador de tiempo

Cambiando el parámetro del corazón podemos hacer que se midan décimas, segundos, minutos...

Usando el segundero como base, se puede crear un cronómetro básico, encadenando contadores. El único problema que tendemos es que no podemos visualizar todos los dígitos a la vez ya que no contamos con tantas salidas en la Alhambra. Aun así, vamos a realizar un cronómetro que cuente hasta 10 minutos, con decimas incluidas. Podremos seleccionar que dígito visualizar gracias a la combinación de 2 interruptores.

Switches	Dígito
00	Décimas
01	Unidades de segundo
10	Decenas de segundo
11	Unidades de minuto

Ilustración 132. Combinaciones posibles

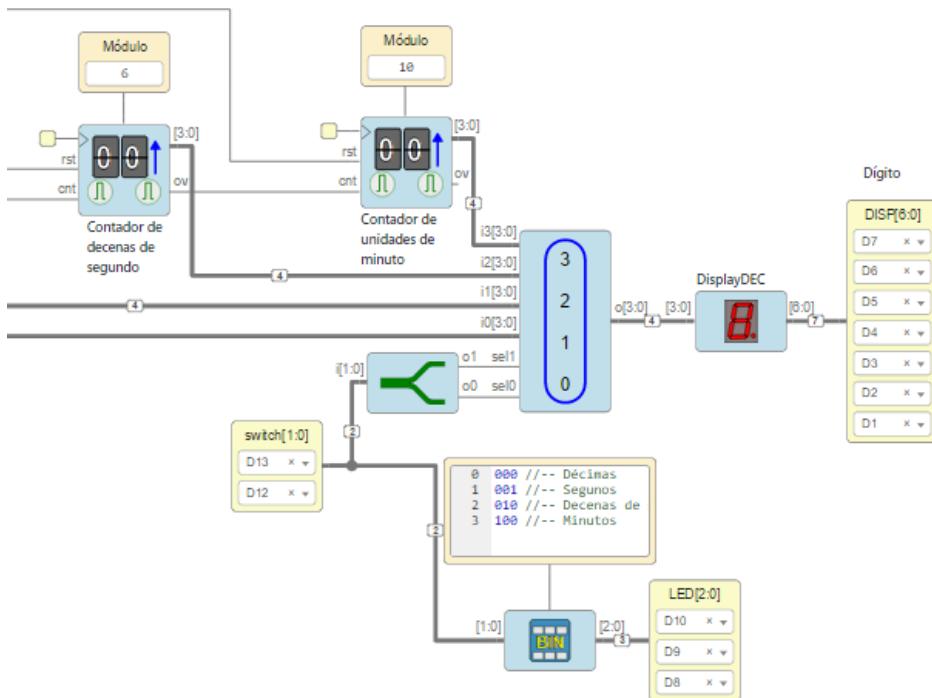
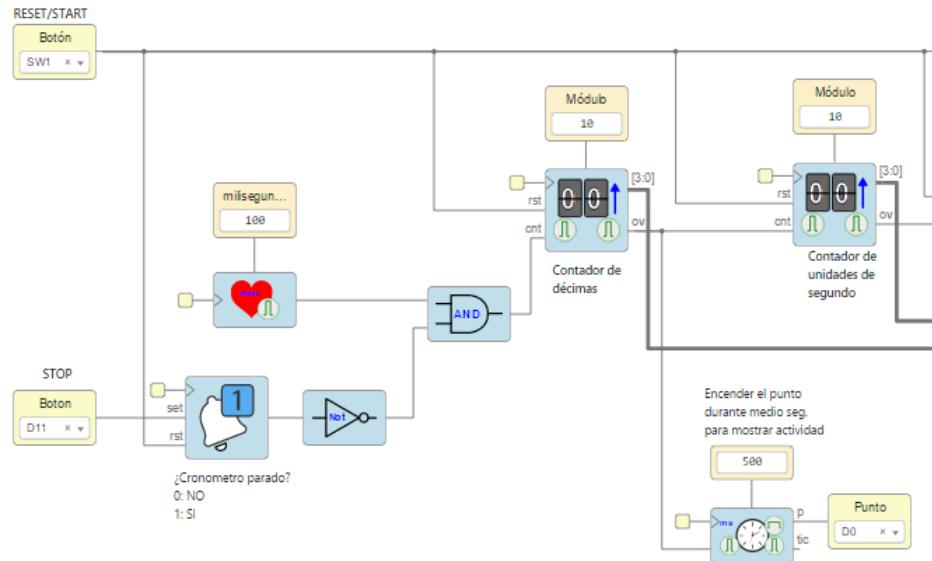


Ilustración 133. Programación del reloj

23.3. Recorriendo tablas

Unos de los usos más extendidos de los contadores es el de recorrer tablas secuencialmente, ya que, gracias a ellos, conseguimos que los circuitos accedan a datos o instrucciones consecutivas. Pueden generar secuencias en LEDs o displays, así como implementar máquinas secuenciales sencillas.

Vamos a generar una secuencia en los LEDs de la placa: La conocida secuencia del coche fantástico, en la que se encienden los LEDs consecutivamente haciendo un efecto como si una luz se desplazase de un lado a otro.

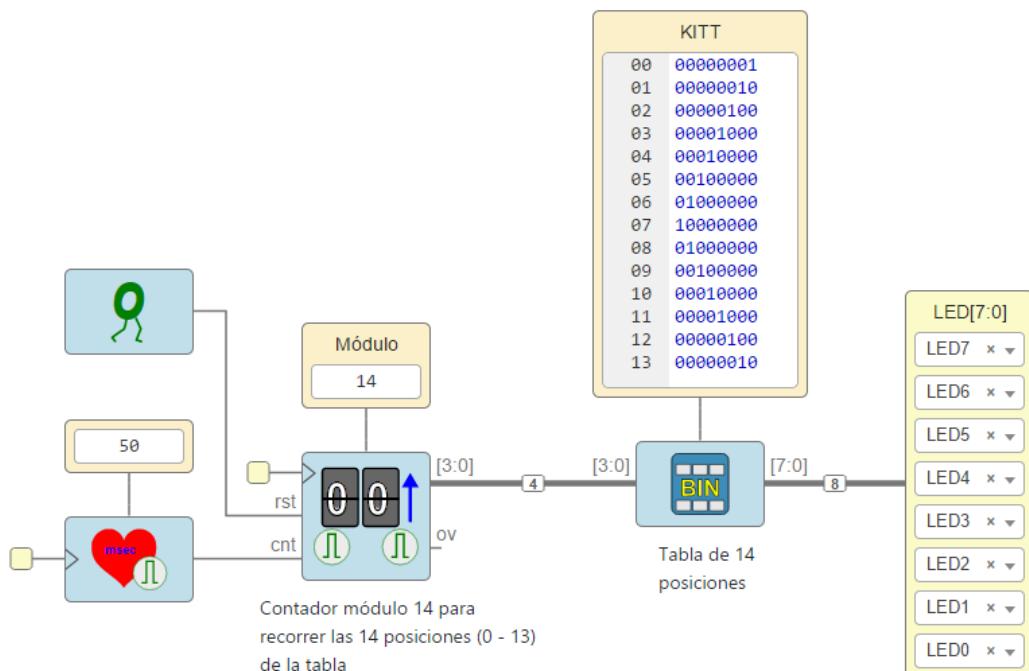


Ilustración 134. Recorriendo una tabla usando un contador

SE RECOMIENDA HACER LOS EJERCICIOS 23.1, 23.2 Y 23.3 DE LA COLECCIÓN

24. Biestables de datos y cambio

Nuestros circuitos manipulan datos que normalmente se almacenan en los biestables tipo D, para hacer operaciones aritméticas o conversiones. También aprenderemos a usar los biestables tipo T, también llamados de cambio.

24.1. Biestable de cambio (T)

Los biestables de cambio, cambian su valor cada vez que reciben un tic por su entrada, esto es, si su valor es 0, al recibir un tic, cambia a 1, y si recibe otro tic vuelve a 0. Probamos su funcionamiento con un pulsador.

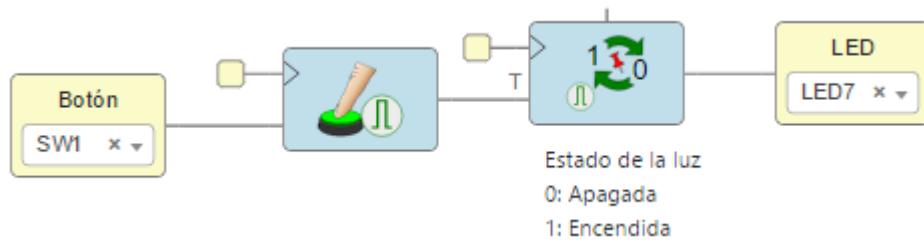


Ilustración 135. Biestable T

La entrada de estos biestable tiene que ser un tic porque si recibe una señal con un ancho de pulso mayor, cambia varias veces de estado. Por ello, no se pueden encadenar directamente. Para poder encadenarlos y así poder conseguir una señal con el doble de periodo hay que usar los bloques de flancos que podemos encontrar en **Varios/Flancos**, este bloque manda un tic al recibir un flanco de bajado/subida.

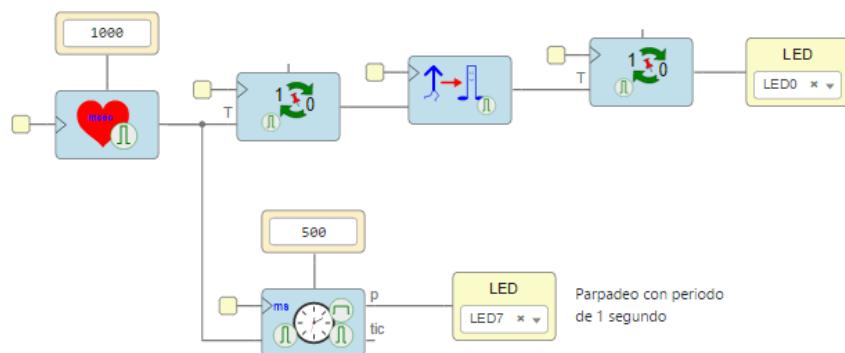


Ilustración 136. Encadenando biestables T

Por ejemplo, gracias a este circuito conseguimos un parpadeo con periodo 4000ms en el LED0, ya que cada vez que sale de un biestable T, se dobla el periodo inicial de 1000ms.

24.2. Biestables de datos

Los biestables de datos se usan para almacenar datos y retenerlos hasta que lleguen los siguientes.

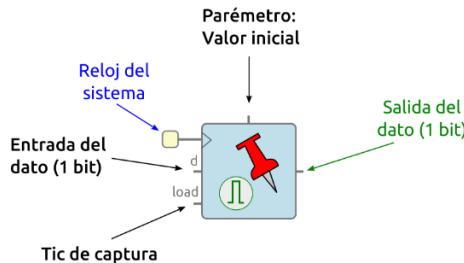


Ilustración 137. Biestable de datos

Además del reloj, el biestable tiene una entrada de un bit por donde llega el dato a almacenar (0 o 1), pero este dato no se guarda en el biestable hasta que llegue un tic por la entrada de captura. Entonces ese dato se guarda y es visible por su salida.

Vamos a probar a capturar un simple 1 cuando pulsemos un pulsador de la Alhambra.

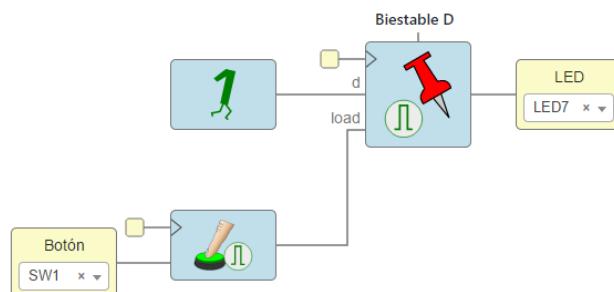


Ilustración 138. Captura de una constante

Pero no tiene que ser siempre una constante lo que se captura, puede ser un dato genérico que llegue desde un interruptor o sensor. Por ejemplo:

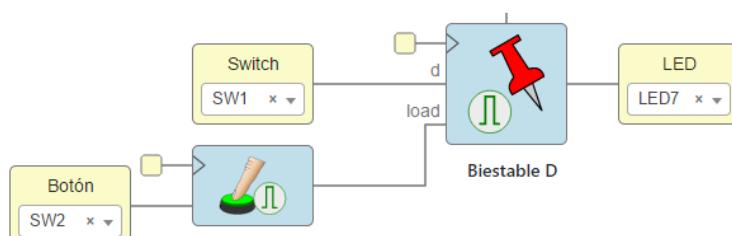


Ilustración 139. Capturando el estado de un Switch

Si queremos capturar más de un bit, podemos hacerlo usando biestables D en paralelo. Cada uno tiene su propia entrada de dato, pero tienen una entrada de captura común, así capturan el dato al mismo tiempo.

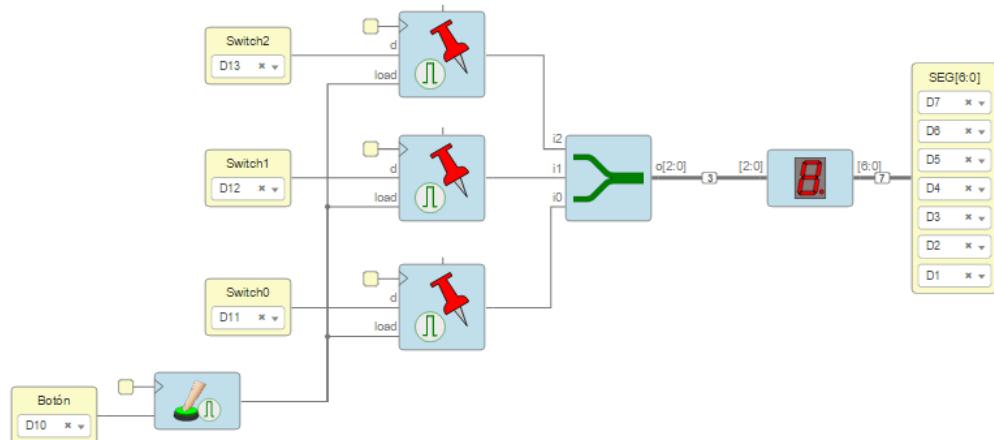


Ilustración 140. Captura de varios estados

24.3. Desplazamiento de Bits

Los biestables D nos permiten realizar desplazamiento, que es una operación muy importante que usemos para operaciones aritméticas o conversión de serie a paralelo. Esto se consigue encadenando biestables D, ya que el bit pasa al siguiente biestable cuando reciba el tic de captura.

Vamos a probar esta función con el siguiente ejemplo. Inicialmente hay un 1 en el primer biestable e iremos introduciendo ceros para ver cómo se desplaza el 1. Visualicemos esto mediante LEDs.

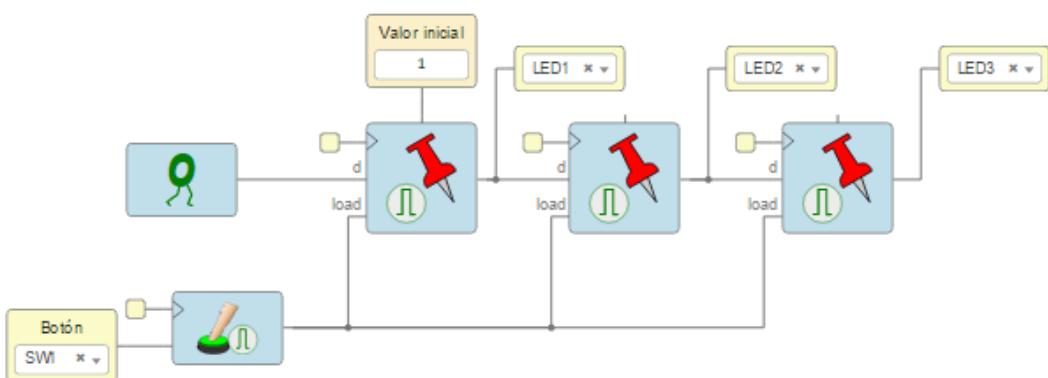


Ilustración 141. Desplazamiento de bit 0

En el sistema de numeración decimal, la operación más sencilla es la multiplicación por 10, ya que basta con añadir un cero por la derecha. En el sistema binario, la operación más sencilla es la multiplicación por dos, ya que solo hay que desplazar los bits hacia la izquierda. Por ejemplo, 0001 es 1, 0010 es 2, 0100 es 4 y 1000 es 8.

24.4. Conversión serie-paralelo

En muchos periféricos, el envío de bits se hace en serie, uno detrás de otro, pero para procesarlos, los necesitamos en paralelo. Para ello existen los conversores serie paralelo basados en biestables D. [Primero se introduce el dato en serie colocando biestables encadenados y luego se carga el dato en biestables D en paralelo.](#)

Vamos a realizar un circuito para ver cómo funciona:

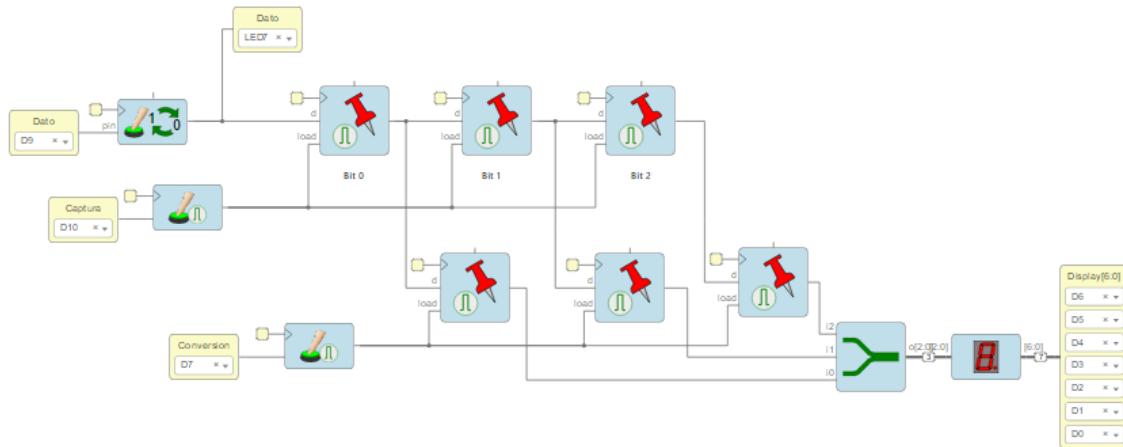


Ilustración 142. Conversión serie paralelo

La parte superior del programa es la vista anteriormente, un desplazamiento de bits. Podemos seleccionar el dato a guardar con el botón Dato y el biestable T. Una vez introducido el dato de 3 bits en serie en los biestables D superiores, al pulsar el botón conversor, esos tres datos se guardan en los 3 biestables inferiores para poder visualizar el dato en el display.

SE RECOMIENDA HACER LOS EJERCICIOS 24.1 Y 24.2 DE LA COLECCIÓN

25. Registros y comparadores

Los registros nos permiten almacenar números de varios bits y trasmisirlos de un circuito a otro en serie o en paralelo. Están compuestos de varios biestables D, tanto en paralelo como en cadena. En este apartado veremos los de carga paralela y los registros de desplazamiento. Encontramos estos bloques en **Varios/Registros**

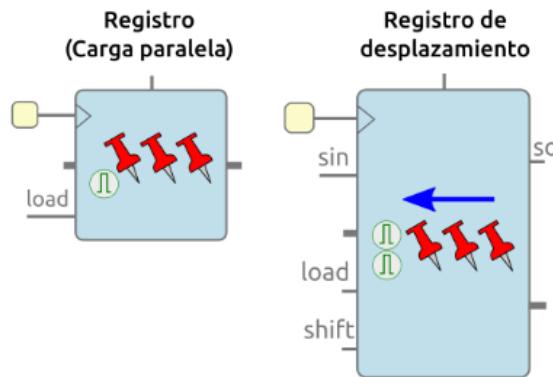


Ilustración 143. Bloque de carga paralela y registro de desplazamiento

25.1. Carga paralela

Como ya hemos dicho, este bloque es la unión de varios biestables D en paralelo y nos permite cargar y almacenar un numero de N bits. Al igual que en los biestables D, tenemos una entrada de dato y un tic de captura.

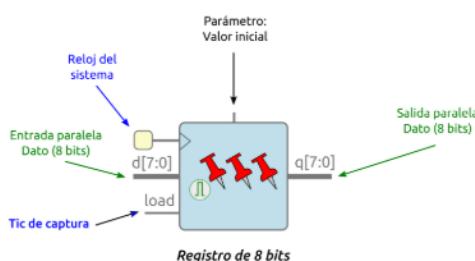


Ilustración 144. Registro de 8 bits

Vamos a observar cómo funciona con un circuito simple, con el cual introducimos un dato de 3 bits en el registro y lo visualizamos en un display.

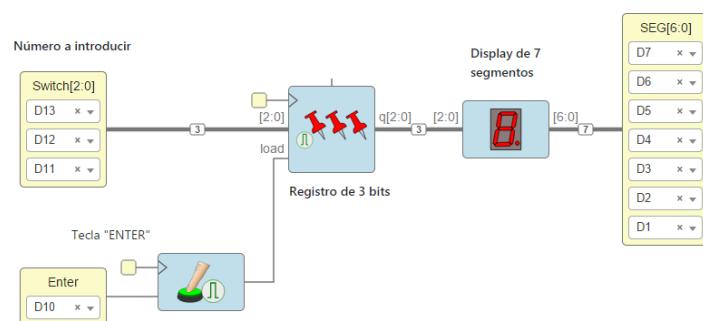


Ilustración 145. Carga de 3 bits en binario

25.2. Registros de desplazamiento

Los registros de desplazamiento son iguales a los de carga paralela, con una entrada adicional para desplazar los bits hacia la izquierda.

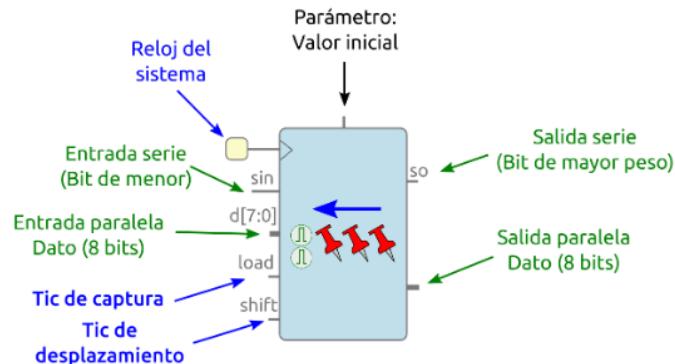


Ilustración 146. Bloque Registro de desplazamiento

Cuando este registro recibe un tic de carga, captura el dato que entra en paralelo. Cuando recibe un tic de desplazamiento, mueves los bits una posición hacia la izquierda, añadiendo el bit de la entrada serie como bit de menor peso y sacando el de mayor peso por su salida serie.

Para experimentar haremos un circuito para cargar un dato de 3 bits y desplazarlo. Introducimos el dato mediante 3 interruptores y se muestra en decimal en un display de 7 segmentos.

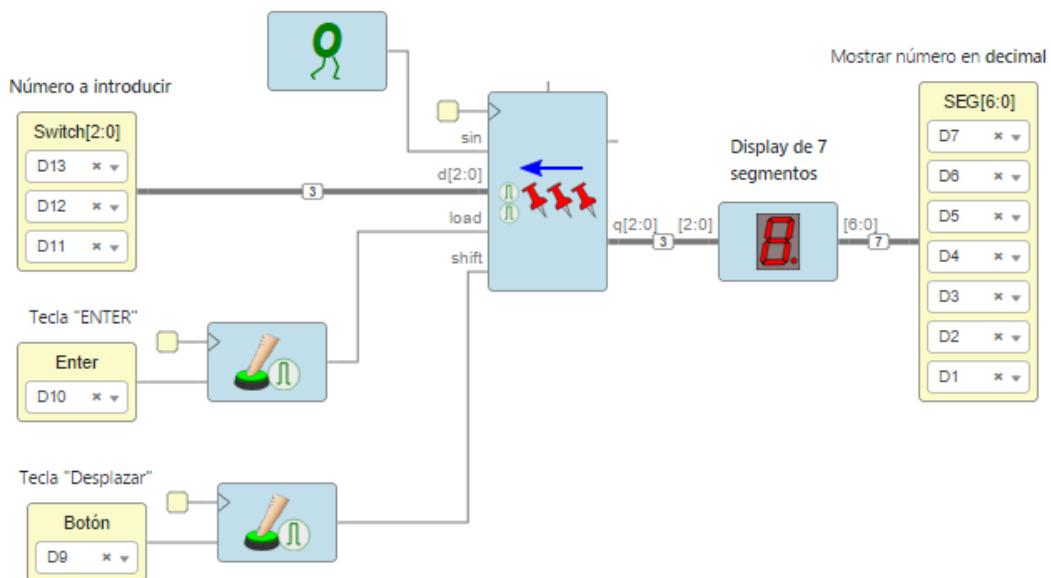


Ilustración 147. Carga y desplazamiento de 3 bits

25.3. Comparadores

Los comparadores sirven para determinar la relación entre dos números, en nuestro caso usaremos el de igualdad y el de menor que. Ambos bloques reciben dos datos de n bits, y en caso de que se cumpla la condición, producen un 1 en la salida (un 0 en caso de que no se cumpla la condición).

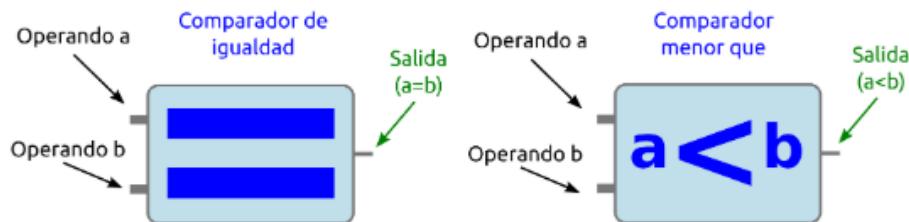


Ilustración 148. Comparadores

Para probar el funcionamiento de estos bloques vamos a usar ambos en el mismo ejemplo. Obtenemos el primer operando de un registro de 3 bits, que introducimos a través de 3 interruptores externos al pulsar la tecla Enter. El operando b lo obtenemos de un contador de 3 bits, el cual podemos ir incrementando mediante un pulsador. Visualizamos las salidas mediante LEDs.

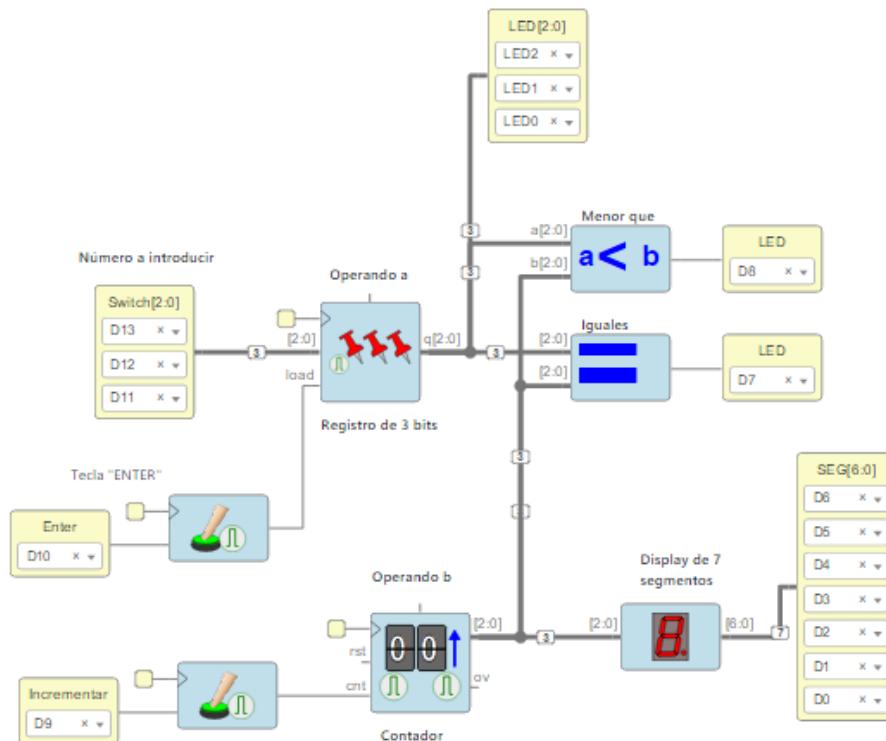


Ilustración 149. Comparación de dos datos

25.3.1. Comparadores de un operando

En caso de querer realizar una comparación con una constante, disponemos de los comparadores de un solo operando, que se introduce y se compara con el parámetro constante. Al igual que con los comparadores de 2 operandos, obtenemos un 0 o un 1 en la salida dependiendo del resultado de la comparación.

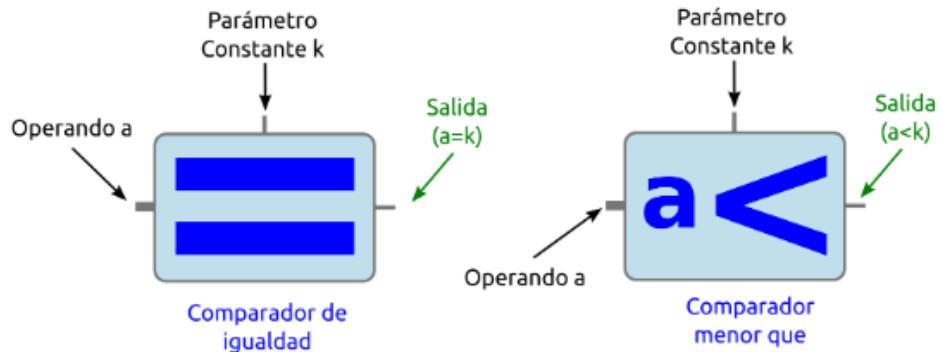


Ilustración 150. Bloques de comparación con una constante

Volvemos a usar el ejercicio de la caja fuerte. Esta vez, comparamos un dato de 3 bits introducido mediante interruptores con el código constante. En caso de ser correcto el servo se mueve de posición.

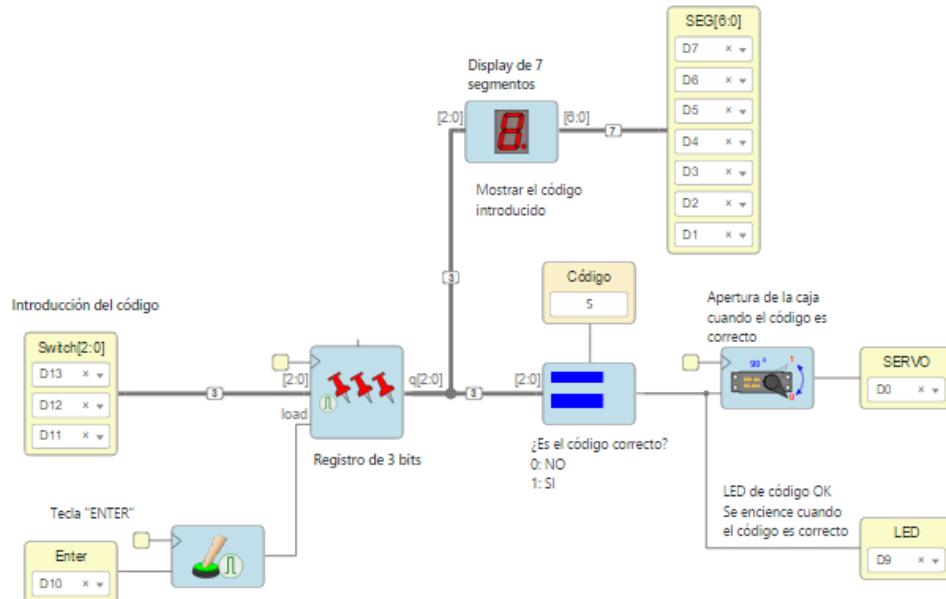


Ilustración 151. Programación de la caja fuerte

25.4. Comunicando FPGA con Arduino

Otra de las aplicaciones de las FPGAs es crear periféricos o controladores para usarlos desde un microprocesador, como por ejemplo Arduino. En la FPGA hacemos la parte física, con pensamiento hardware, mientras que en el Arduino la parte de programación, con pensamiento computacional, mezclando de esta forma el diseño hardware con el software

Una forma sencilla de empezar es usando comunicaciones serie síncronas entre el Arduino y la FPGA, donde los bits de transmiten uno detrás de otro el cable de datos (un cable por sentido), en los instantes indicados por una señal de reloj. Usamos además un cable de control para indicar el fin de la transferencia.

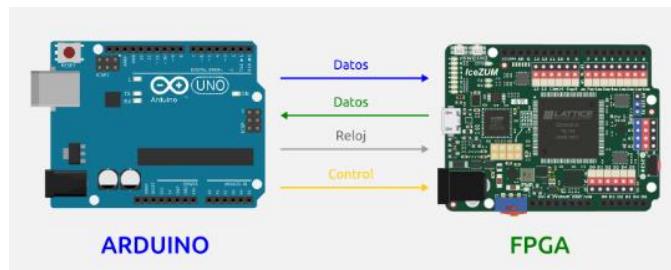


Ilustración 152. Conexión ARDUINO-ALHAMBRA

El Arduino es el Máster: es el que genera la señal de reloj y el que toma la iniciativa en la comunicación, tanto para enviar como para recibir. La FPGA es la esclava. Este tipo de comunicación se llama maestro-esclavo, y es el esquema usado en los buses como el SPI o i2c

Aunque el Arduino incorpora hardware específico para realizar esta comunicación, lo hacemos por todo por software, para comprender mejor el funcionamiento. Primero aprenderemos a enviar datos desde el Arduino a la FPGA (escritura), y luego de la FPGA al Arduino (lectura).

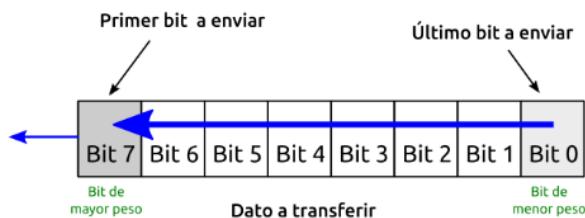


Ilustración 153. Transferencia de datos

Usamos números de 8 bits (bytes) como unidades de comunicación, aunque podríamos usarlos de cualquier otra longitud. También establecemos el convenio de enviar primero el bit más significativo y el último el de menor peso, en ambos sentidos de comunicación.

25.4.1. Escritura

La operación de enviar datos desde el Arduino a la FPGA lo llamamos escritura. Utilizamos tres cables: el que lleva los datos del Arduino a la FPGA (verde), el que lleva la señal de reloj (gris) y la señal de control (amarillo). Además, hay que conectar las masas (GND) de ambas placas (negro).

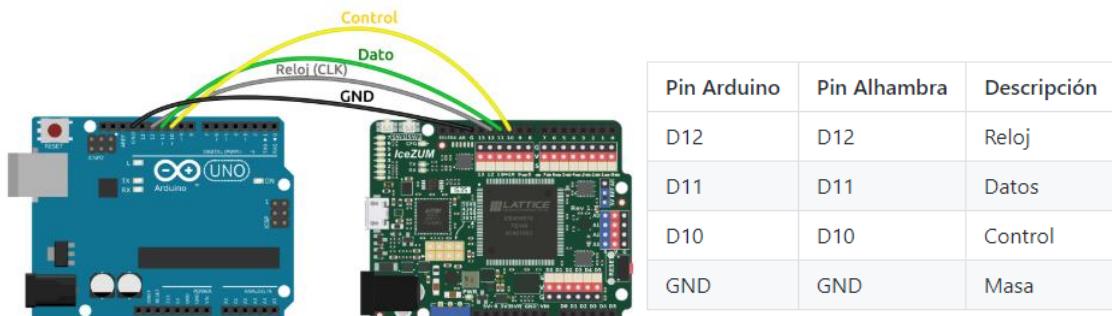


Ilustración 154. Conexión ARDUINO-ALHAMBRA

La recepción de los datos serie se hace usando un registro de desplazamiento de 8 bits. El cable de datos se conecta a la entrada sin y la señal de reloj (clk) a la de desplazamiento.

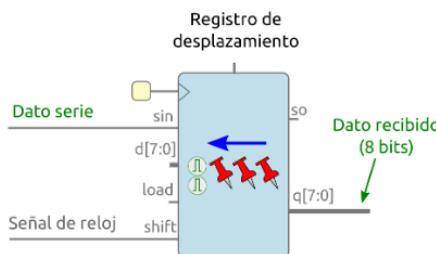
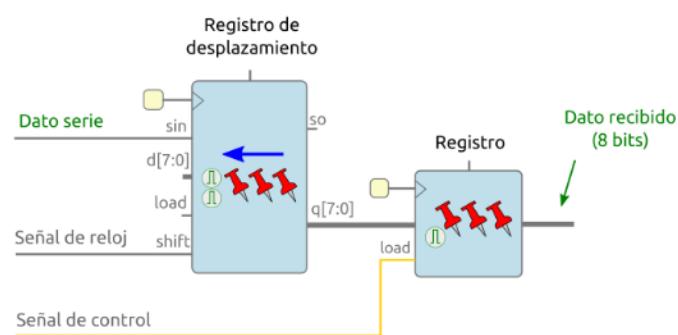


Ilustración 155. Bloque usado para la recepción de datos

Durante la recepción del dato, el registro se va desplazando y en su salida aparecen valores temporales que NO son el dato final. Por ello, colocamos un registro adicional que capture el dato final, cuando se haya recibido completamente. La orden de captura es la que llega por el cable de control.



25.4.1.1. Circuito

Todas las llegan desde el

Ilustración 156. Conexión de la señal de control

sincronizador

señales que exterior de la

FPGA tiene que pasar por un circuito sincronizador, que tiene este aspecto en Icestudio, y que se encuentra en el menú **Varios/Input/Sync**.

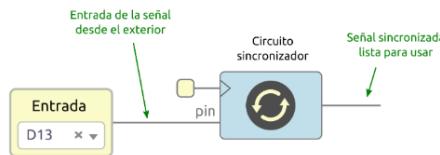


Ilustración 157. Circuito sincronizador

El sincronizador está formado por dos biestables D en cascada cuya misión es minimizar los problemas de metaestabilidad que suceden cuando la señal exterior cambia en el mismo instante que el flanco de reloj del sistema de la FPGA. En esos casos, el dato capturado no es ni uno ni cero.

25.4.1.2. Receptor serie síncrono

Nuestro receptor serie síncrono genérico, está formado por las tres señales de entrada: datos, reloj y control, que pasan por su respectivos sincronizadores. Además, tenemos el registro de desplazamiento para convertir el dato serie a paralelo y el registro de datos que contiene el dato final recibido.

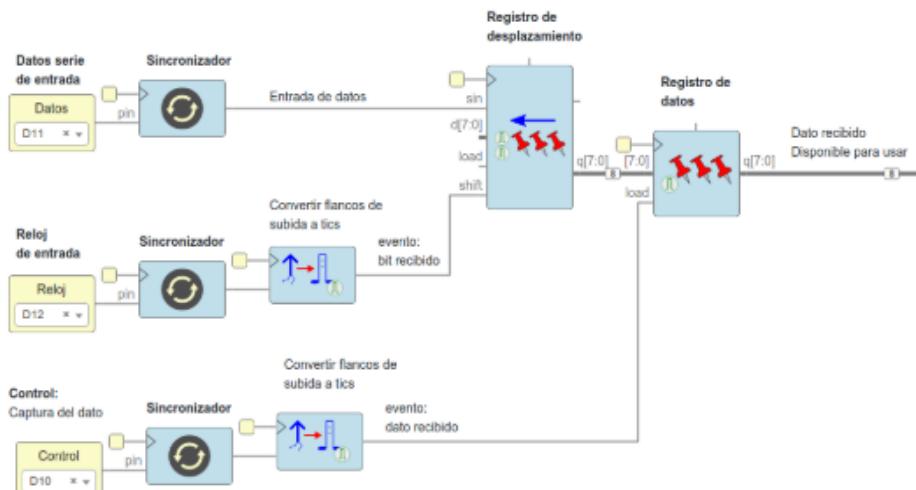


Ilustración 158. Receptor serie

Las señales de reloj y control las convertimos en tics, mediante un conversor de flancos de subida en tics, para poder introducirlas en las entradas de shift y load de los registros de desplazamiento y datos respectivamente.

El Arduino cada vez que realiza una operación de escritura sobre la FPGA está almacenando un dato de 8 bits en este registro de datos. Según el circuito que se esté implementando, este dato recibido se usa con un fin u otro.

Para realizar esta escritura, el Arduino (software) tiene que realizar estas operaciones:

- Inicialmente las señales de reloj y control deben estar a 0
- Enviar el dato en serie, empezando por el bit más significativo. Por cada bit enviado debe generar un pulso en la señal de reloj (escritura de un 1, seguida de un 0)
- Una vez enviado el octavo bit, se debe generar otro pulso en la señal de control (un 1 y luego un 0) para que el dato se capture en el registro de datos.

25.4.1.3. Software para escritura

Para realizar la escritura desde el Arduino a la FPGA implementamos la función `fpga_write()`, que envía el dato pasado como parámetro. Se implementa de forma muy fácil usando la función `shiftOut` de la biblioteca de Arduino.

```
void fpga_write(int value) {
    shiftOut(DAT, CLK, MSBFIRST, value);
    digitalWrite(CTRL, HIGH);
    digitalWrite(CTRL, LOW);
}
```

Ilustración 159. Función ShiftOut de Arduino

La función `shiftOut()` envía datos de 8 bits de una vez. Con la constante `MSBFIRST` se indica que envíe primero el bit de mayor peso (que es como lo tenemos configurado en el hardware). Las constantes `CLK`, `DAT` y `CTRL` son los pines de Arduino por donde sacar las señales de reloj, datos y control respectivamente.

Las señales de reloj y datos las controla la propia función `shiftOut`. Una vez que se ha enviado el dato, se emite el pulso en la señal de control para que se capture en el registro de datos. Este pulso lo emitimos nosotros, desde la función `fpga_write()`.

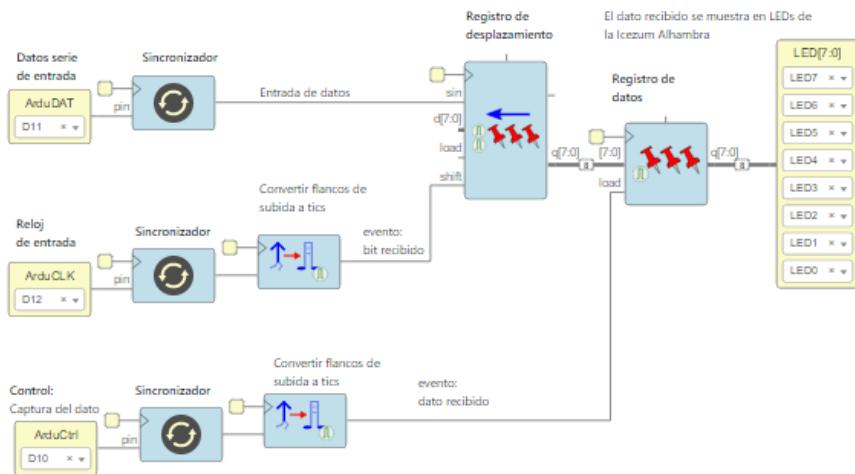


Ilustración 160. Recepción de datos

Vamos a aumentar las salidas del Arduino usando un puerto de 8 bits de la FPGA, en el que usaremos la conexión FPGA-Arduino anteriormente vista. La implementación del circuito

usa el receptor serie síncrono genérico, al que se le ha conectado la salida del registro de datos a los LEDs de la Alhambra, para ver el dato recibido.

En el Arduino cargamos un programa de pruebas, que por ejemplo escriba los datos 0x55 y 0xAA cada medio segundo. Primero se configuran los pines a usar. El pin de control se pone a 0. En el bucle principal se envían los dos valores. El programa se puede descargar de aquí: [Puerto-secuencia.ino](#)

Para probarlo, cargamos el hardware en la Alhambra, desde Icestudio, y el software en Arduino, desde el IDE de Arduino. Los valores escritos por el Arduino aparecen en los LEDs. La secuencia se detiene en cualquiera de los siguientes casos: reset de la FPGA, de Arduino o si desconectamos uno de los cables.

25.4.2. Lectura

La operación de recibir datos en el Arduino desde la FPGA lo llamamos lectura. Igual que para la escritura, utilizamos tres cables: reloj, control y datos. La diferencia está en que el cable de datos transmite la información en el sentido FPGA → Arduino. Para diferenciarlo del otro sentido (Arduino → FPGA) usaremos el color azul.

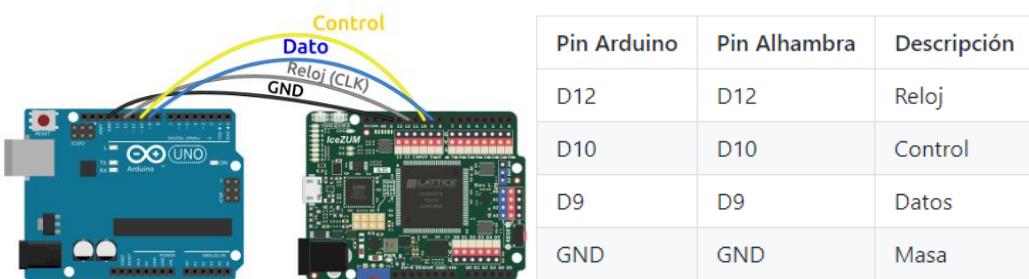


Ilustración 161. Conexión ARDUINO-ALHAMBRA

La transmisión de los datos desde la FPGA se hace usando un registro de desplazamiento de 8 bits. El cable de datos se conecta a la salida serie “so”, la señal de reloj (clk) a la de desplazamiento (shift) y la señal de control a la de carga paralela. El dato de 8 bits se introduce por la entrada paralela d del registro.

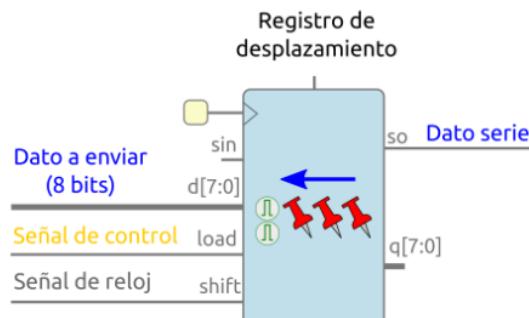


Ilustración 162. Registro de desplazamiento

25.4.2.1. Transmisor serie síncrono

Nuestro transmisor serie síncrono genérico está formado por dos señales de entrada: reloj y control, que pasan por su respectivos sincronizadores y llegan al registro de desplazamiento, que convierte el dato a enviar de paralelo a serie. El dato se envía por la salida serie y pasa por un biestable D del sistema para eliminar los glitches y [cumplir con las reglas del diseño síncrono](#).

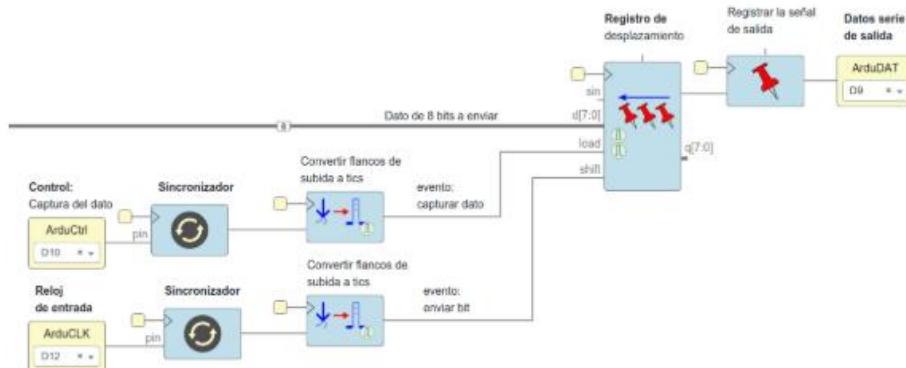


Ilustración 163. Sincronización de datos

El dato a enviar normalmente está almacenado en un registro, al que llamamos registro de datos, pero podría venir de cualquier otro lugar: una memoria, un contador, etc, por eso no se ha incluido en el esquema genérico.

Para realizar la lectura, el Arduino (software) tiene que realizar estas operaciones:

- Inicialmente las señales de reloj y control deben estar a 0
- Enviar un pulso por la señal de control, para que la FPGA capture el dato a enviar
- Recibir el dato en serie, empezando por el bit más significativo. Cada vez que lee un bit debe generar un pulso en la señal de reloj
- Al cabo de 8 pulsos de reloj ya tiene el dato completo

25.4.2.2. Software para lectura

Para realizar la lectura de la FPGA desde el Arduino implementamos la función `fpga_read()`, que devuelve el dato recibido. Se implementa de forma muy fácil usando la función `shiftIn` de la biblioteca de Arduino.

La función `shiftIn()` recibe un dato de 8 bits de una vez. Con la constante `MSBFIRST` se indica que recibe primero el bit de mayor peso (que es como lo tenemos configurado en el hardware). Las constantes `CLK`, `CTRL` y `DAT` son los pines de Arduino por donde sacar las señales de reloj y control, y el pin por donde se reciben los datos respectivamente.

La señal de reloj la controla la propia función `shiftIn`. Antes de invocarla es necesario generar un pulso en la señal de control para que la FPGA capture el dato. Lo emitimos nosotros al comienzo de la función `fpga_read()`. Como ejemplo de lectura de un dato de la FPGA desde el Arduino, vamos a implementar un puerto de entrada adicional de 8 bits. En los bits menos significativos de este puerto conectamos tres interruptores, y desde el Arduino leemos sus valores

Por 3 pulsadores se introduce un valor y se pulsa el botón de ENTER para capturarlo. Su valor se muestra en el display de 7 segmentos. Ese es el valor que lee el Arduino, que muestra por la consola serie. La implementación del circuito usa el transmisor serie síncrono genérico, al que se le ha conectado por su entrada paralela el registro de datos que contiene el número introducido por los interruptores. Este registro es el que se muestra por el display de 7 segmentos

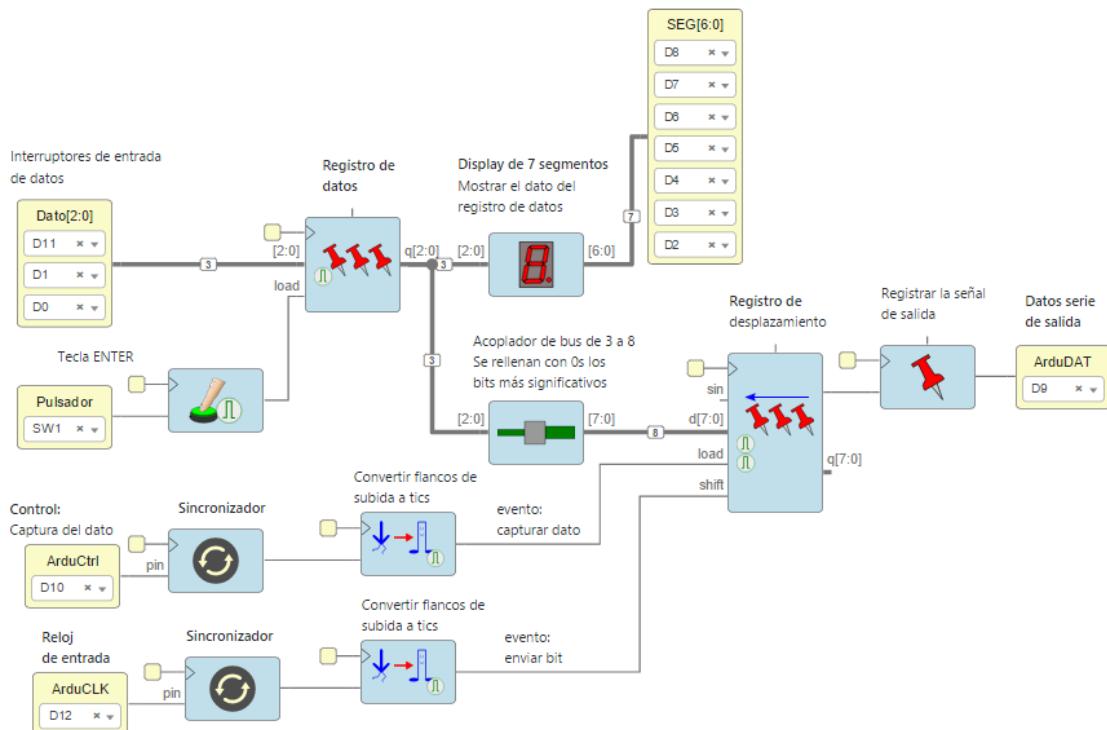


Ilustración 164. Circuito de lectura

El registro de desplazamiento es de 8 bits, mientras que el registro de datos es de 3 bits. Por eso se usa un elemento acoplador para conectarlos, que simplemente añade ceros en los bits de mayor peso del registro de datos. Este elemento se encuentra en el menú **Varios/Bus/3-bits/Acoplador-3-8**

En el Arduino cargamos un programa de pruebas que lee constantemente el registro de datos, y si detecta algún cambio, muestra el número por la consola serie. El programa completo se puede descargar de aquí: Puerto-[entrada.ino](#).

En el bucle principal se lee el dato en la variable dat, y en old se encuentra el valor leído anteriormente. Si son diferentes es que ha habido un cambio con respecto a la lectura anterior, por lo que se imprime el número. Para pasar de binario a ASCII sólo hay que sumar el carácter '0': Serial.write (dat+'0');

Para probarlo, cargamos el hardware en la Alhambra, desde Icestudio, y el software en Arduino, desde el IDE de Arduino. En este vídeo lo vemos en acción. Los números introducidos en los switches se muestra en la consola serie del Arduino al apretar el botón de Enter.

SE RECOMIENDA HACER LOS EJERCICIOS 25.1 DE LA COLECCIÓN

26. Puerto serie

Una forma sencilla de comunicar nuestros circuitos con el PC es usando el puerto serie. Son comunicaciones asíncronas y están muy extendidas en los dispositivos como BT, Arduino, microcontroladores... Se utiliza un solo cable de datos para cada sentido ya que la velocidad se acuerda antes de la comunicación.

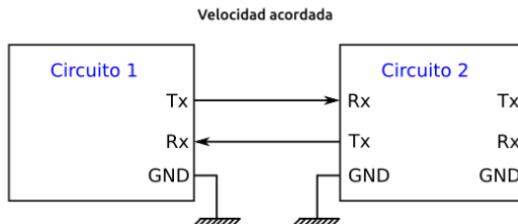


Ilustración 165. Conexión puerto serie

Los ordenadores modernos no tienen conectores del puerto serie, pero como se sigue usando mucho, han creado los chips USB-serie para poder realizar esta conexión mediante el puerto USB. Estos puertos, además de los cables de datos Tx y Rx, cuentan con otros 6 cables de control opcionales: DTR, RTS, CTS, DSR, DCD y RI. Algunos dispositivos como Arduino usan el DTR o RTS para resetearla placa, activar el bootloader o cargar el programa.

La FGPA cuenta con las señales de control DTR, RTS, CTS, DSR y DCD, además de las señales TX y RX de trasmisión de datos.

26.1. Terminal de comunicaciones

Se puede utilizar cualquier terminal de los que existen, en cada sistema operativo. En este tutorial usaremos dos: el [Arduino-IDE](#) y el [ScriptCommunicator](#), que son libres y multiplataforma.

Para descargar el Arduino IDE, buscamos la versión más actual en la página web y descargamos. Ejecutamos la descarga como administrador y se iniciará la instalación.

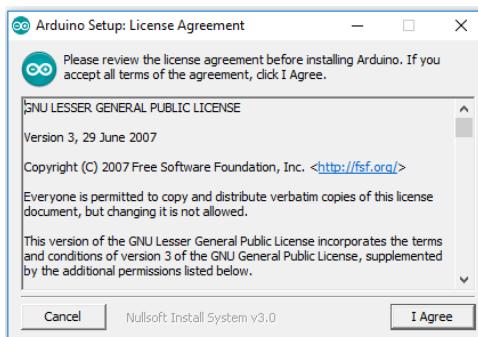


Ilustración 166. Instalación de Arduino IDE

Hacemos clic en **I Agree** y seleccionamos todas las opciones. Pinchamos en **Next**. Dejamos el entorno por defecto y comienza la instalación. Una vez acabe, hacemos clic en **Close** y ya estará listo para usar.

Para el ScriptCommunicator, descargamos la última versión desde el Github. Al hacer clic hay que esperar unos segundos y comenzará la descarga. Descomprimir el fichero y ejecutar el instalador. En caso de que aparezca el siguiente mensaje de advertencia hacemos clic en Más información y tras eso, a Ejecutar e todas formas

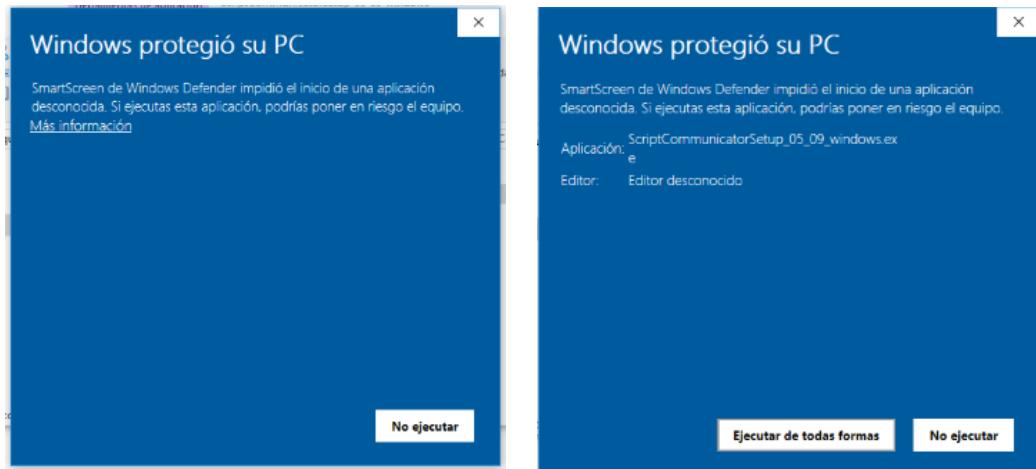


Ilustración 167. Permisos en la instalación de Arduino IDE

Dejamos todas las opciones por defecto y hacemos clic en Install. Al cabo de unos segundos habrá terminado, así que hacemos clic en Finish.

Haremos una prueba para comprobar que todo funciona correctamente. Simplemente uniremos cables y los montaremos en los LEDs:

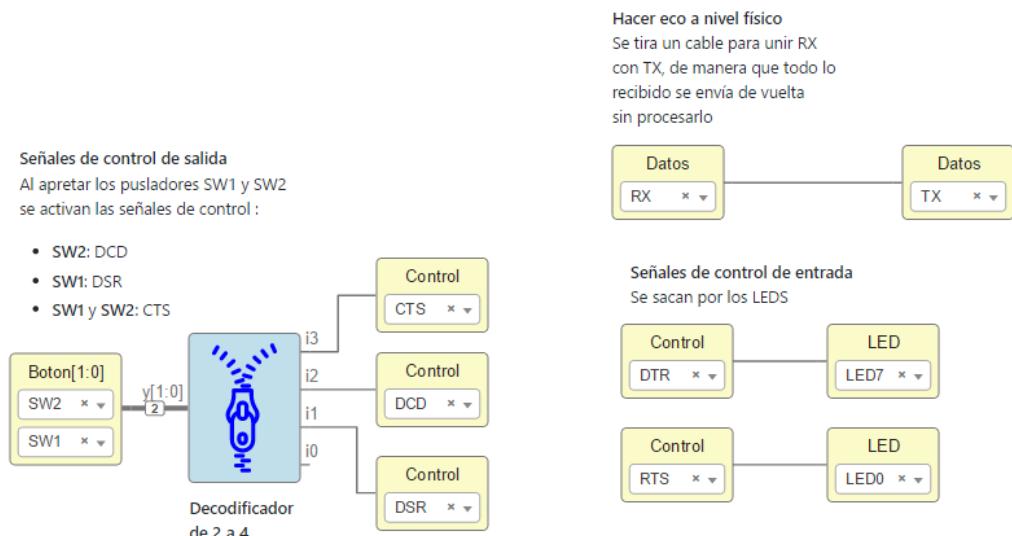


Ilustración 168. Circuito de comprobación

Lo cargamos en la placa. Primero probaremos el funcionamiento con el IDE Arduino. Vamos a **Herramientas/Puerto** y seleccionamos el COMX, donde la X varía de un ordenador a otro.

Abrimos el terminal serie y hacemos una prueba enviando diferentes cadenas. Deberíamos recibir las mismas. Es indiferente la velocidad a la que esté configurado el terminal. Los LEDs de TX y RX de la Alhambra se encienden unos instantes al intercambiar datos con el PC:

Desde Arduino no tenemos acceso a las señales de control así que seguimos las pruebas con ScriptCommunicator. En este entorno también debemos seleccionar el puerto serie, desde **Settings/SerialPort/Port**.

Para comenzar las pruebas nos conectamos a la placa haciendo clic en **Connect**. Comprobamos que el eco funciona (igual que en Arduino). En la parte inferior del ScriptCommunicator vemos las señales CTS, DSR y DCD a 1. Al apretar los pulsadores se pondrán a 0, ya que tienen la lógica invertida. Los LEDs 7 y 0 estarán encendidos y pinchando en DTR y RTS cambiaremos su estado.

26.3. Transmisor Serie

El envío de datos de la FPGA al PC lo realizamos a través del transmisor serie que encontramos en **Varios/Serial**.

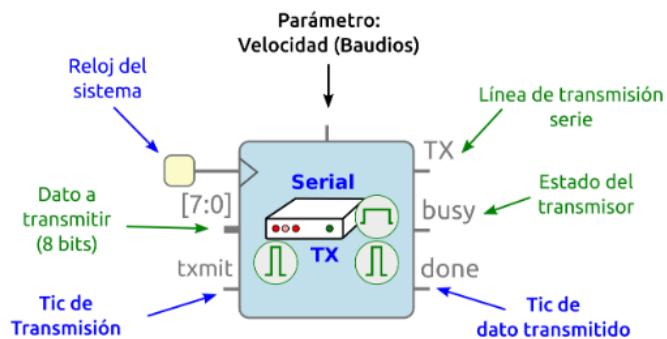


Ilustración 169. Bloque TX

El dato a transmitir se introduce por la entrada de datos, de 8 bits. Al generar el tic de transmisión, se captura el dato y se envía por la línea serie (TX). Es un funcionamiento similar a la captura de los registros.

Hasta que el dato no se haya transmitido completamente, NO se puede enviar el siguiente. El estado del transmisor lo sabemos por su salida Busy: que indica si está ocupado o no. Cuando se ha enviado el dato actual, emite el tic de dato transmitido.

La velocidad de transmisión la fijamos mediante el parámetro. Tenemos que introducir alguna de las velocidades estándares (en baudios): 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 o 115200. Por defecto se usan 115200 baudios.

Empezamos probando a enviar un carácter constante cada vez que se pulse el botón SW1 de la Alhambra. Cuando el dato se haya enviado, encenderemos un LED durante 100ms.

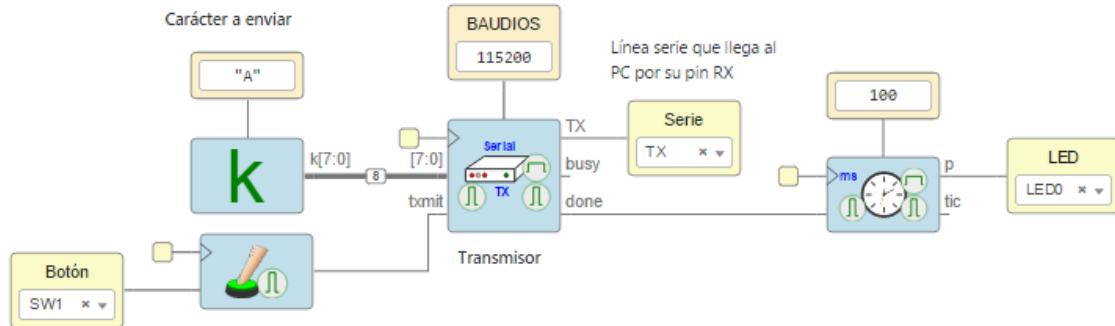


Ilustración 170. Envío de una constante

Cargamos el circuito y lo probamos. Abrimos el ScriptCommunicator, configurando la velocidad a 115200 baudios. Una vez conectados, observamos cómo llega una A con cada pulsación y parpadea el LED.

También podemos enviar números en binario de la misma forma que con caracteres. Añadimos un display para ver el dígito enviado.

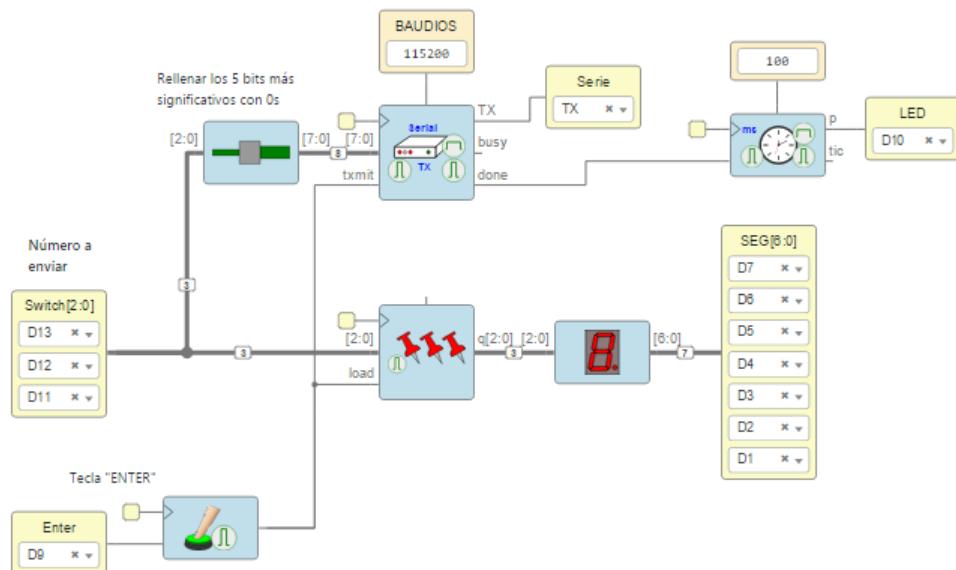


Ilustración 171. Envío de un dígito de 3 bits

Lo que se está enviando son números crudos. El ScriptCommunicator tiene un par de pestañas activables que nos permiten visualizar el dato recibido en hexadecimal y en binario. Si lo visualizamos en ASCII, veremos "Basura", ya que lo que se envían no se corresponde con caracteres ASCII visibles. Estas pestañas se activan desde **Settings/ConsoleOptions**.

También podemos mandar cadenas gracias al bloque “serial-tx-str32”.

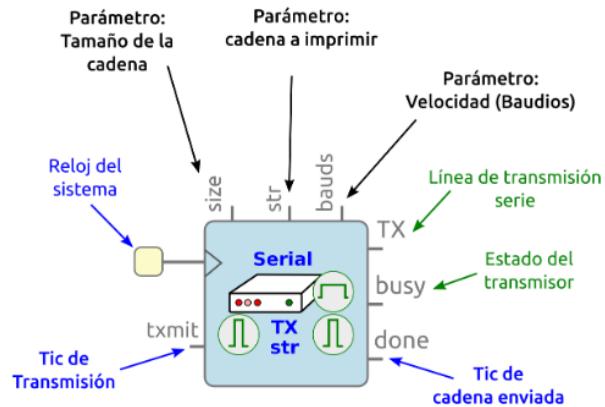


Ilustración 172. Bloque TX String

Este bloque, no tiene entrada de datos, se enviará la cadena escrita como parámetro. Por lo demás funciona como el bloque anterior.

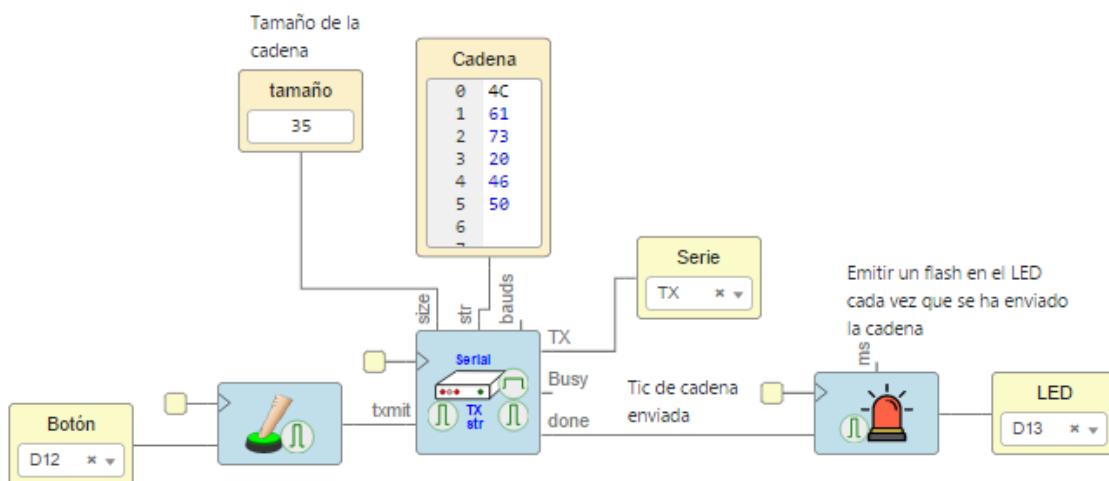


Ilustración 173. Envío de cadena

Lo cargamos y ya podemos ver el mensaje en el terminal cada vez que se pulsa el botón.

26.4. Receptor Serie

Al igual que para transmitir, disponemos de un bloque para recibir en **Varios/Serial**.

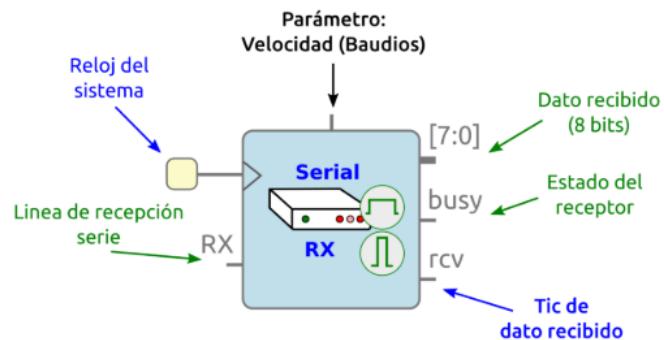


Ilustración 174. Bloque RX

Los datos en serie llegan por el pin RX y salen por un bus de 8 bits en paralelo. Cada vez que llega un dato nuevo, el receptor emite un tic de recibido y la señal de busy se pone a 0. La velocidad parámetro tiene que ser la misma que hemos puesto en el transmisor del PC o recibiremos mensajes basura.

Como primer ejemplo, visualizaremos el dato recibido en binario en los LEDs. Además, cada vez que llegue un mensaje, un LED parpadeará.

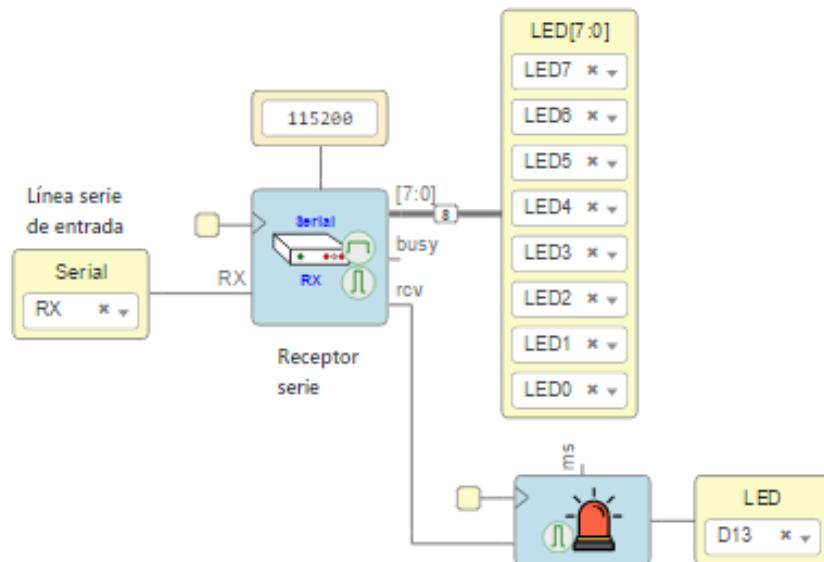


Ilustración 175. Visualización del dato recibido

Para poder enviar datos pulsado teclas y no tener que darle al Enter desde el ScriptCommunicator, hay que activar la opción send input en la parte inferior.

Tras esta prueba simple, vamos a programar un control de un servo con dos teclas desde el PC. Para ello, el circuito deberá comparar el dato recibido con el predefinido para cada movimiento.

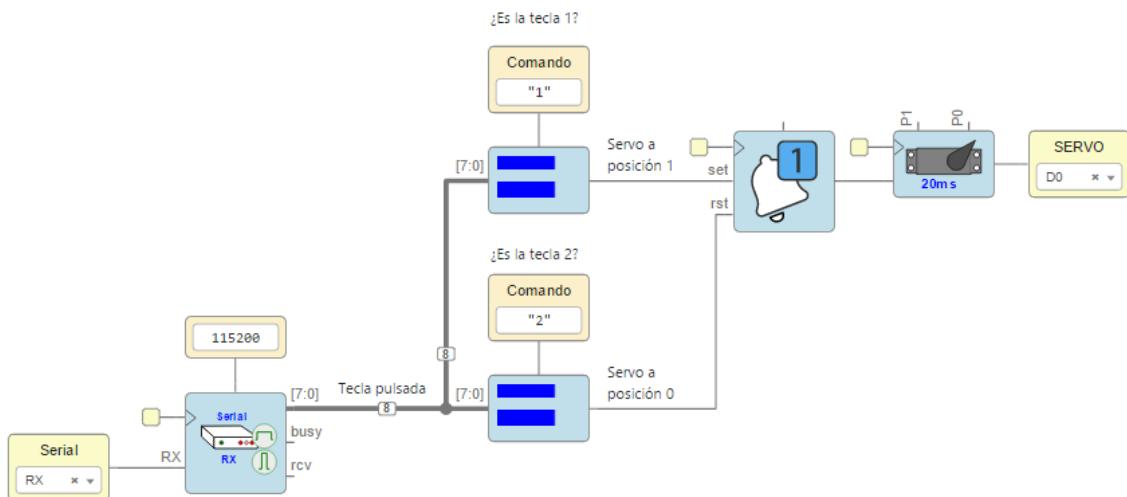


Ilustración 176. Control de servo mediante dato recibido

Una vez ha llegado el dato, lo comparamos con los parámetros prefijados para cada movimiento y usando un biestable seleccionamos la posición deseada del servo.

En caso de querer poder elegir la posición del servo, contamos con el siguiente bloque:

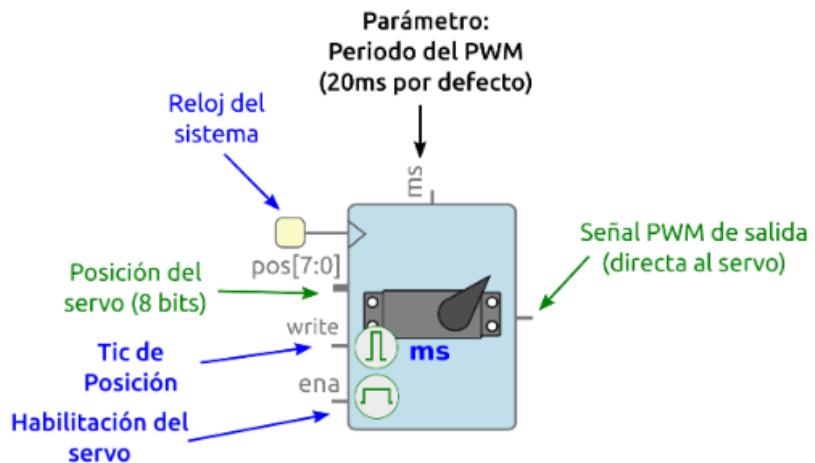


Ilustración 177. Bloque control de posición del servo

Controlaremos la posición mediante su entrada de 8 bits, el tic de posición y la habilitación. Y mandaremos la señal PWM directamente al servo. Como parámetro, indicaremos el periodo del PWM del servo.

Vamos a ver un ejemplo de cómo controlar un servo con los datos enviados desde PC.

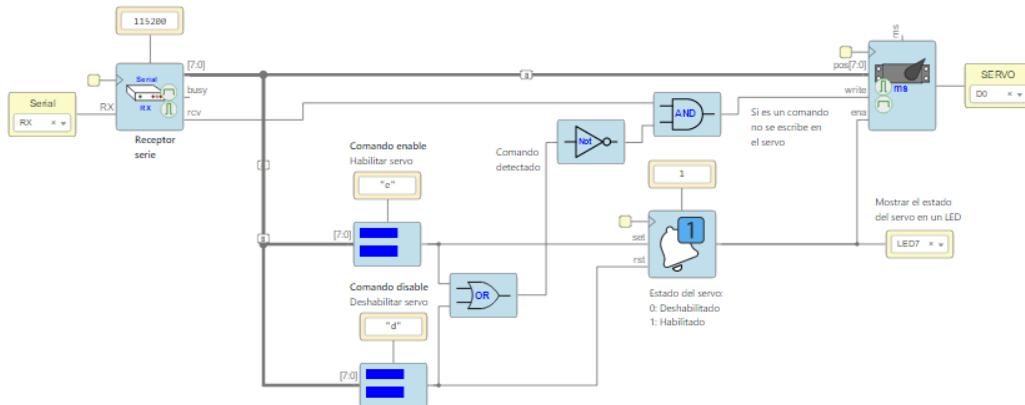


Ilustración 178. Control de servo mediante dato recibido desde PC

Primero deberemos saber si el dato recibido es para habilitar o deshabilitar el servo o un dato de posición. Esto lo conseguimos mediante dos comparadores y un biestable que controlar la habilitación. En caso de que no sea un dato de control, se le da la orden al servo para moverse a la posición recibida.

Desde el ScriptCommunicator podemos enviar datos en decimal seleccionado en el desplegable uint8. Se introduce el valor en decimal y se pincha en send. Desde la pestaña ASCII también podemos usar las teclas para posicionar el servo y enviar los comandos 'e' y 'd'

26.5. Combinando Transmisor y Receptor

Normalmente tendremos que integrar tanto emisor como receptor en el mismo circuito por lo que vamos a ver un ejemplo implementándolos.

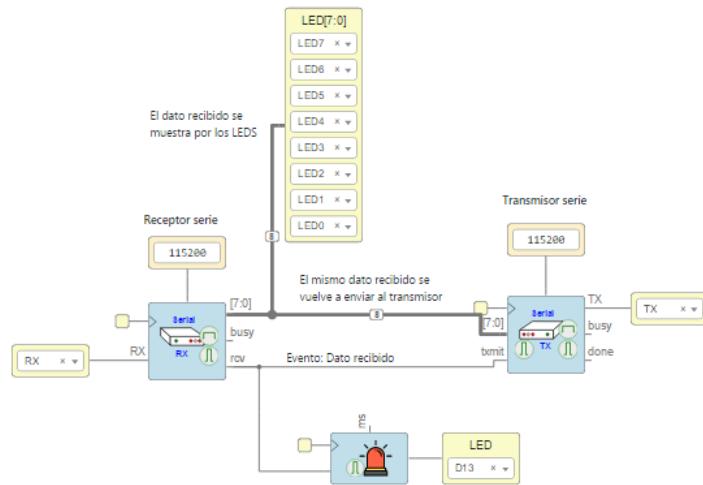


Ilustración 179. Combinando RX y TX

En este circuito se vuelve a transmitir todo lo que se recibe. Esto se denomina hacer eco. Además de enviar cada dato de vuelta, se muestra en los LEDs, y se emite un parpadeo en el LED. La salida de datos del receptor se conecta directamente a la entrada de la transmisión, y también la señal de rcv a la de txmit. El mismo tic de dato recibido se usa para transmitirlo de vuelta.

26.6. Comunicación Arduino-FPGA

Otra forma de comunicarnos con Arduino es usando el puerto serie. En vez de enviar comandos desde el PC a la FPGA, lo haremos desde el Arduino. Utilizaremos el puerto serie proporcionado por la biblioteca SoftwareSerial.h, y dejar el puerto estándar para la comunicación con el PC y poder así depurar los programas.

La conexión entre Arduino y la FPGA la haremos mediante 3 cables: GND, TX y RX. La FPGA actúa como un Servidor, ofreciendo servicios al Arduino, que es el Cliente

Pin Arduino	Pin Alhambra	Descripción
D11	D11	TX. Datos: Arduino->FPGA
D10	D10	RX. Datos: FPGA->Arduino
GND	GND	Masa

Ilustración 180. Conexión Arduino-FPGA

La FPGA ofrece dos servicios al Arduino a través del puerto serie: incremento de un contador y lectura de un interruptor. Se implementan con los comandos 'i' y 'r'. Cuando el Arduino envía una 'i', la FPGA incrementa un contador módulo 10, que se muestra en el display de 7 segmentos. Cuando envía el comando 'r', la FPGA devuelve el estado del interruptor: dígito '0' cuando si está a 0, y dígito '1' si está a 1.

Mediante un comparador y una puerta AND se obtiene el tic de comando, que aparece al llegar cada comando. El tic del comando 'i' incrementa el contador. El tic del comando 'r' se usa para transmitir el estado del interruptor. El interruptor tiene sólo 1 bit, y para enviarlo hay que llenar hasta los 8 bits. Pero si se rellena colocando el número 3 en los 4 bits de mayor peso, conseguimos convertirlo al dígito '0' o '1', ya que sus códigos ASCII son 0x30 y 0x31 respectivamente

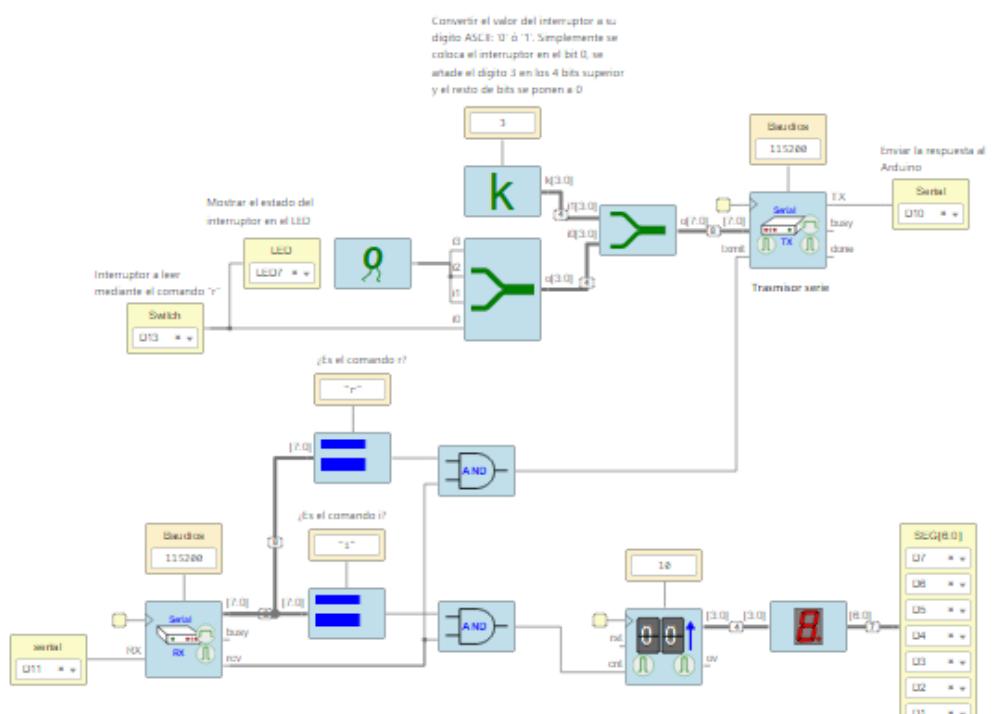


Ilustración 181. Transmisión Arduino-FPGA

El Arduino es el cliente, que solicita los servicios al hardware. Haremos un programa que incremente el contador cada 500 ms, y que lea el estado del interruptor de la FPGA para actualizar su LED y enviar los cambios al PC para verlos en un terminal

En el Arduino se usan dos puertos serie. El implementado con la biblioteca SoftwareSerial.h es el que usaremos para comunicarnos con la FPGA, dejando el otro para enviar mensajes al PC y depurar el código. El programa es [serial_client.ino](#).

Para probar la aplicación cargamos primero el hardware en la FPGA, y luego el software desde el entorno de Arduino.

Primero se carga el hardware y luego el software. Al hacerlo vemos cómo el contador se empieza a incrementar, y escuchamos los pitidos. Si se aprieta el pulsador de reset del Arduino, el contador dejará de incrementarse. Luego abrimos un terminal y cambiamos de posición el interruptor. Cada vez que hay un cambio aparece un mensaje en la consola. Si nos fijamos en el LED de Arduino veremos que cambia su estado según el valor de este interruptor.

26.7. Comunicación Bluetooth-Serie

Conectando un adaptador bluetooth-serie, podemos comunicarnos con nuestros circuitos de forma inalámbrica, por ejemplo, desde un teléfono móvil. En este curso, usamos el HC-05.

El módulo tiene un LED rojo que nos indica su estado: conectado o no conectado. La alimentación es entre 3.3v - 5v. En total tiene 6 pines, pero sólo utilizamos 4: VCC, GND, TX y RX.

Repetiremos el ejemplo del eco, pero esta vez usamos el móvil como terminal. Hay infinidad de aplicaciones para hacer estas pruebas en Google Play.

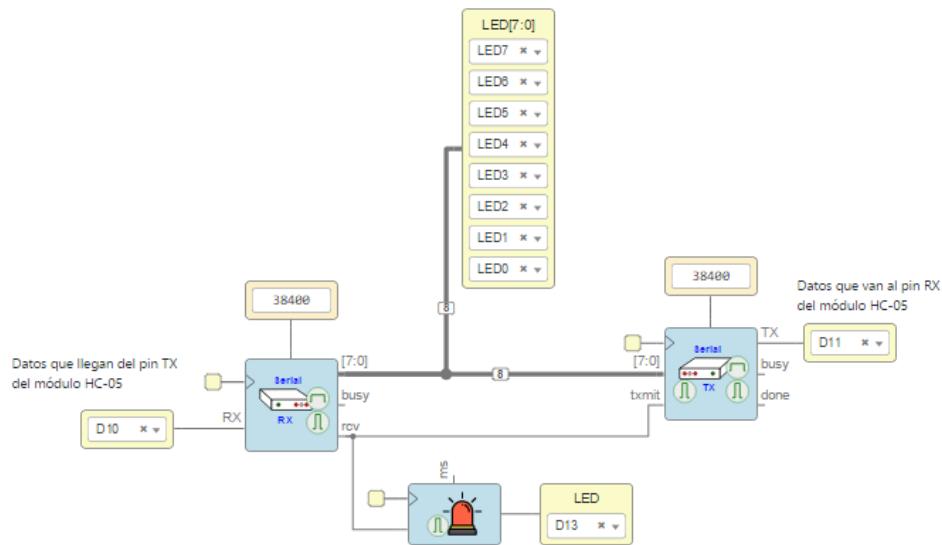


Ilustración 182. Prueba Eco con la conexión Bluetooth

Cargamos el circuito en la FPGA y arrancamos la APP. El LED rojo del módulo HC-05 estará parpadeando, el mismo tiempo encendido que apagado, indicando que NO hay conexión. Seleccionamos el modulo al que queremos conectarnos desde la APP. El LED del módulo HC-05 ahora parpadeará de forma diferente: dos parpadeos rápidos y un segundo apagado (se repite esta secuencia).

Al apretar una tecla, el carácter correspondiente se envía, y la FPGA lo recibe y se muestra en binario en los LEDs. Vemos el LED parpadear y escuchamos el pitido corto. La FPGA re-envía el carácter y lo vemos en el terminal.

SE RECOMIENDA HACER LOS EJERCICIOS 26.1 Y 26.2 DE LA COLECCIÓN