

**ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA Y DISEÑO INDUSTRIAL**

TRABAJO DE MICROS:

MÁQUINA DESPACHADORA DE REFRESCOS

MATERIA:

SISTEMAS ELÉCTRONICOS DIGITALES

INTEGRANTES:

- GUTIÉRREZ LÓPEZ JOSUE
- OCARANZA MORALES JENNIFER
- TELLEZ QUINTANA RICARDO



ÍNDICE

INTRODUCCIÓN	3
Implementaciones en VHDL	3
Implementaciones para el Micro	4
DESARROLLO	6
CONTROLADOR DE LA MAQUINA DESPACHADORA DE REFRESCOS.....	6
DIAGRAMA DE BLOQUES DE LA MAQUINA DESPACHADORA DE REFRESCOS.....	7
PRUEBAS DE MICROCONTROLADOR	8
Sensor HC-SR04.....	8
LCD 16 X 2.....	8
Interruptores y comunicación.....	9
ESTRATEGIA Y ALGORITMOS DESARROLLADOS	10
TIM 1	10
I2C1.....	10
SPI1	11
Librerías.....	13
Código	14
• Librerías	14
• Variables.....	14
• Funciones.....	15
• Main	15
• While.....	16
RESULTADOS.....	20

INTRODUCCIÓN

Durante este proyecto se va a diseñar una máquina expendedora de refrescos con las siguientes características:

Admite monedas de 10c, 20c, 50c y 1€. Sólo admite el importe exacto, de forma que si introducimos dinero de más da un error y “devuelve” todo el dinero. Cuando se llega al importado exacto del refresco (1€) se activará una señal para dar el producto. Como entradas tendrán señales indicadoras de la moneda, señales indicadoras de producto y como salidas la señal de error y la de producto.

Implementaciones en VHDL

Uno de los aspectos más destacables es la capacidad de operar en dos modos diferentes mediante un interruptor (sw_c). En el modo de cambio (sw_c = '1'), la máquina permite ingresar una cantidad superior al precio exacto del producto, calcula el cambio correspondiente, lo muestra en el display de 7 segmentos y posteriormente entrega el producto. Por otro lado, en el modo sin cambio (sw_c = '0'), la máquina no permite excedentes y solo acumula dinero hasta alcanzar el importe exacto, mostrando el monto acumulado en el display y emitiendo una señal de error si se intenta introducir más dinero del necesario. Esta funcionalidad proporciona flexibilidad y mejora la experiencia del usuario al adaptarse a distintas situaciones.

Además, la máquina permite seleccionar entre tres diferentes precios de refresco, lo que amplía las opciones disponibles para los usuarios. Cada precio está asociado a un producto específico, identificado mediante caracteres en el display de 7 segmentos: d para el producto 1, E para el producto 2 y F para el producto 3. Esta información se muestra junto con el precio correspondiente, permitiendo al usuario visualizar de manera clara el producto seleccionado y su costo.

El diseño incluye un controlador lógico encargado de gestionar todas las señales de entrada y salida del sistema. Este controlador valida las monedas ingresadas, determina si se debe devolver cambio y activa la señal para la entrega del producto una vez que se alcanza el importe necesario. En caso de errores, como introducir dinero de más en el modo sin cambio, el sistema genera una señal de error y devuelve automáticamente todo el dinero ingresado.

Implementaciones para el Micro

El microcontrolador STM32F411E-DISCO desempeñó un papel fundamental en la comunicación en serie SPI entre él mismo, actuando como Máster, y la FPGA, que actúa como Slave. Este sistema de comunicación permite dividir las funciones de la máquina despachadora de refrescos en dos componentes principales:

1. Cálculos y procesamiento de datos: Realizados por la FPGA. La FPGA procesa todas las señales provenientes de los sensores y botones, ejecutando las operaciones necesarias para el funcionamiento de la máquina.
2. Interacción con el usuario: Este detecta las acciones del usuario mediante botones e interruptores, los cuales están configurados para enviar datos en tiempo real a la FPGA a través de comunicación SPI. Además, el microcontrolador se encarga de proyectar información relevante en el display y controlar los LEDs indicativos.

El microcontrolador se utiliza específicamente para las funciones de interfaz y botones para facilitar el uso del usuario final. Aunque el procesamiento central lo realiza la FPGA, el microcontrolador actúa como un puente entre las acciones del usuario y el sistema principal, gestionando las respuestas de la FPGA para mostrarlas al usuario.

Funciones específicas del microcontrolador:

- Proyección de información: El microcontrolador recibe datos desde la FPGA y los transforma en salidas visibles para el usuario, como información en el display de 7 segmentos o señales visuales a través de LEDs. Por ejemplo:
 - Cuando la máquina entra en estado de error, los LEDs parpadean para indicar el problema.
 - Durante el despacho de un producto, los LEDs cambian a otro patrón distintivo para indicar que el proceso está en curso y parará hasta el momento en que se toma la bebida.
- Botones: Los botones están conectados al microcontrolador y envían datos a la FPGA dependiendo del estado actual del sistema. Cada botón representa una función específica, como seleccionar un producto o reiniciar el proceso.

Configuración de datos entre Nexys y el microcontrolador:

- La FPGA Nexys cuenta con 8 entradas digitales conectadas al microcontrolador. Estos 8 bits se envían desde la FPGA al microcontrolador para proporcionar información esencial sobre el estado del sistema.
- Por su parte, la FPGA envía al microcontrolador un paquete de datos de 24 bits, que este utiliza para tomar decisiones y ejecutar las tareas correspondientes. La comunicación es bidireccional y se realiza mediante 3 paquetes de 8 bits, garantizando la sincronización de las operaciones.

DESARROLLO

CONTROLADOR DE LA MAQUINA DESPACHADORA DE REFRESCOS.

Para el diseño de nuestra máquina expendedora de refrescos, se optó por implementar una máquina de estados finitos (FSM) debido a la naturaleza secuencial de las operaciones que debe realizar. La FSM permite modelar de manera eficiente los diferentes pasos necesarios para el correcto funcionamiento del sistema, garantizando una transición clara entre las etapas según las entradas y condiciones establecidas. En este proyecto, cada estado de la FSM representa una etapa clave del funcionamiento de la máquina. La descripción del funcionamiento del controlador se hizo en el documento del trabajo de VHDL, por lo que en este documento solo se comentara de forma general.

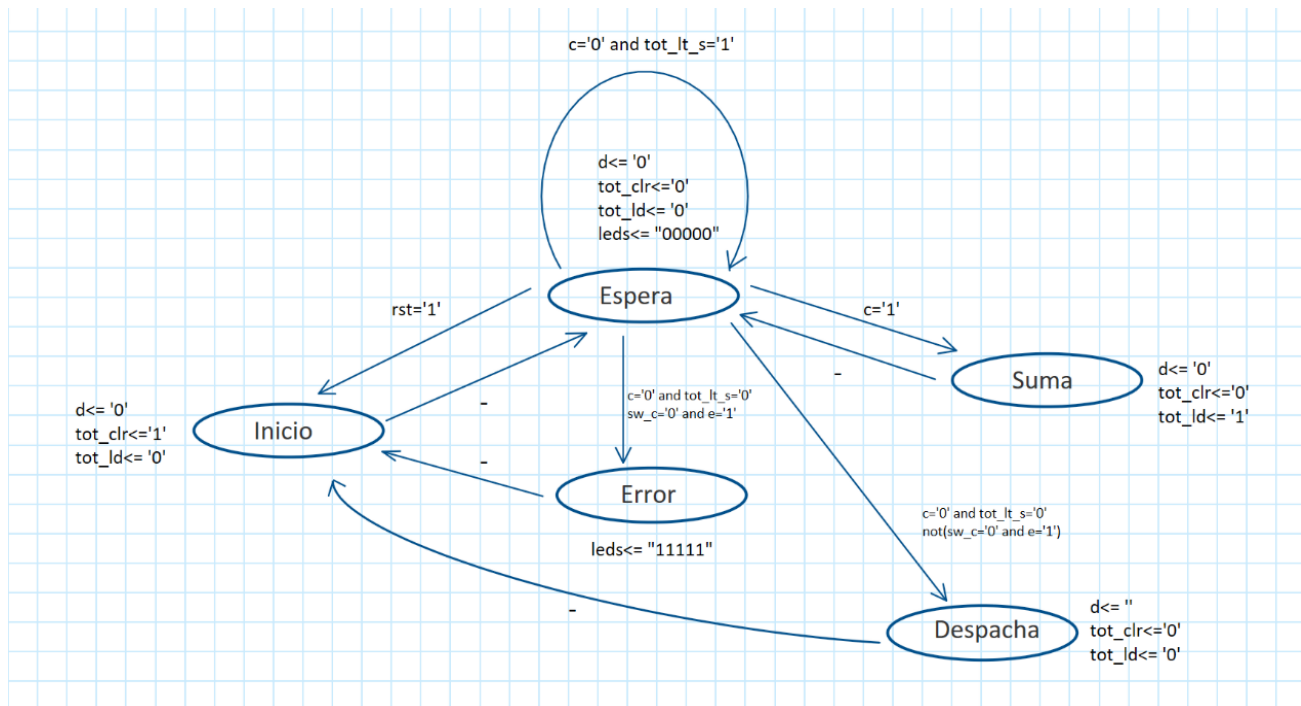


Figura 1. Máquina de estados para el controlador de la Máquina de refrescos despachadora.

DIAGRAMA DE BLOQUES DE LA MAQUINA DESPACHADORA DE REFRESCOS.

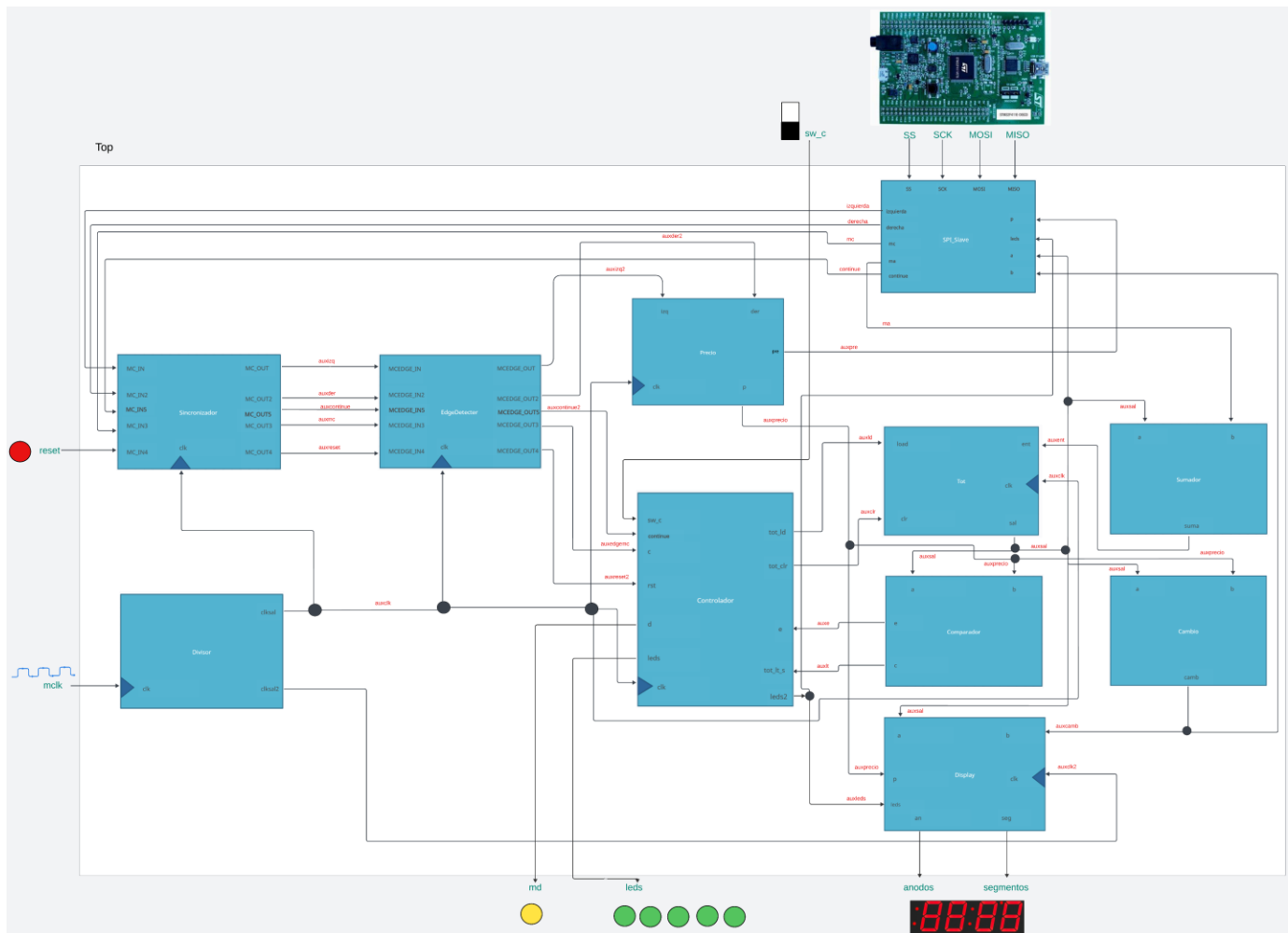


Figura 2. Diagrama de bloques de la Maquina Despachadora de Refrescos adaptada para la comunicación en serie.

Al igual que la máquina de estados anteriormente mencionada, la descripción a detalle de este diagrama de bloques se realizó en el documento del trabajo VHDL, por lo que en este reporte se dará a conocer a detalle cómo se logró la comunicación en serie entre el micro y la FPGA misma que fue posible gracias a la entidad SPI_SLAVE (bloque utilizado en la descripción en VHDL).

PRUEBAS DE MICROCONTROLADOR

En las pruebas realizadas con el microcontrolador se integraron diferentes dispositivos clave para asegurar el correcto funcionamiento de la máquina despachadora de refrescos. A continuación, se detallan las pruebas y resultados obtenidos con cada componente.

Sensor HC-SR04

Una de las pruebas realizadas para el correcto funcionamiento del proyecto final fue el uso del sensor HC-SR04, el cual lo usaremos para detectar cuando el refresco despachado sea tomado por el cliente.

Este se programará mediante interrupciones y solo cuando queramos conocer el valor de la distancia a la que se encuentra el refresco se llamará a la interrupción para que le diga al micro el valor deseado. Posteriormente cuando supera cierta distancia indica que fue tomado el refresco y finalmente toma la indicación deseada.



Figura 3. Sensor HC-SR04.

Para este se configuró un temporizador para medir el tiempo de retorno de la señal del sensor, lo que permite calcular la distancia al objeto.

LCD 16 X 2

Para que tengamos una interfaz amigable para el cliente final se colocará un LCD para crear el menú en el cual aparecerá toda la información sobre el funcionamiento de la máquina y que el cliente pueda seleccionar el refresco.

Para este se usará una librería para facilitar el funcionamiento del LCD

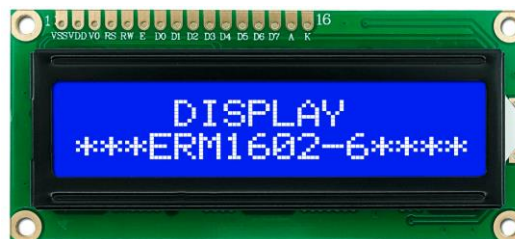


Figura 4. LCD 16 X 2.

Para este configuramos una comunicación con este periférico mediante I2C y se configuró de la siguiente forma:

Pin Na...	Signal on ...	GPIO outp...	GPIO mode	GPIO Pull...	Maximum ...	User Label	Modified
PB8	I2C1_SCL	n/a	Alternate ...	No pull-up ...	Very High		<input checked="" type="checkbox"/>
PB9	I2C1_SDA	n/a	Alternate ...	No pull-up ...	Very High		<input type="checkbox"/>

Figura 5. Configuración puertos LCD por comunicación I2C.

Interruptores y comunicación

Antes de que este funcionando por completo el microcontrolador se realizaron pruebas de interruptores para ver si en efecto se mandaba correctamente la información a la nexys, dentro de esta prueba de igual de comprobó que la comunicación entre estos se realizara de forma correcta con la configuración deseada la cual es Full Duplex ya que siempre que el micro que es el maestro mande la señal de lectura y escritura, haga estos dos de forma simultánea.

Esta prueba es de las mas importantes ya que si la comunicación no funciona de forma correcta, todo lo demás podría no funcionar correctamente.

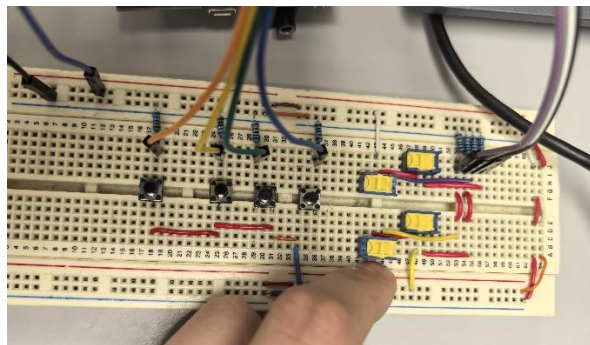


Figura 6. Configuración de interruptores.

Todas las conexiones tienen resistencias en configuración de Pull-Down para que por defecto tenga valores de 0 y le envíe 5V cuando se pulse el botón.



Figura 7. Conexión Pull-Down ocupada para mandar datos a la Nexys .

Aparte de esto anteriormente se realizaron pruebas de comunicación con Arduino para la programación del microcontrolador y así poder tener todo listo para el momento de la prueba con la Nexys, este fue mediante comunicación I2C.

ESTRATEGIA Y ALGORITMOS DESARROLLADOS

TIM 1

▼ Counter Settings	
Prescaler (PSC - 16 bits value)	5
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits v.	65535
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits value)	0
auto-reload preload	Disable
▼ Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
▼ Input Capture Channel 1	
Polarity Selection	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter (4 bits value)	0

Figura 8. Configuración TIM1.

Este se ocupó como anteriormente se mencionó para el correcto funcionamiento del sensor ultrasónico. Estas fueron las configuraciones utilizadas para que funcionara de forma adecuada.

I2C1

✔ Parameter Settings

✔ User Constants

✔ NVIC Settings

✔ DMA Settings

✔ GPIO Settings

Configure the below parameters :

🔍

Search (Ctrl+F)

⏪

⏩

ⓘ

▼ Master Features

I2C Speed Mode

Standard Mode

I2C Clock Speed (Hz)

100000

▼ Slave Features

Clock No Stretch Mode

Disabled

Primary Address Length selection

7-bit

Dual Address Acknowledged

Disabled

Primary slave address

0

General Call address detection

Disabled

Figura 9. Configuración I2C1.

Estas son las configuraciones establecidas para que el Display pudiera proyectar de forma adecuada el menú de opciones que explicaremos un poco más adelante.

SPI1

Parameter Settings	User Constants	NVIC Settings	DMA Settings	GPIO Settings
Configure the below parameters :				
<input type="text" value="Search (Ctrl+F)"/> <input type="button" value="←"/> <input type="button" value="→"/> <input type="button" value="i"/>				
Basic Parameters				
Frame Format	Motorola			
Data Size	8 Bits			
First Bit	MSB First			
Clock Parameters				
Prescaler (for Baud Rate)	2			
Baud Rate	1.5 MBits/s			
Clock Polarity (CPOL)	Low			
Clock Phase (CPHA)	2 Edge			
Advanced Parameters				
CRC Calculation	Disabled			

Figura 10. Configuración SPI1.

Adicionalmente tenemos estas configuraciones para que pusiera existir la comunicación entre la nexys y el micro, siendo el micro el maestro. Esta comunicación es Full-Duplex ya que se lee y se escribe al mismo tiempo en el micro mediante dos buses de datos. La señal de reloj es impuesta por el micro, al igual que la señal de inicio y fin del envío y recepción de datos.

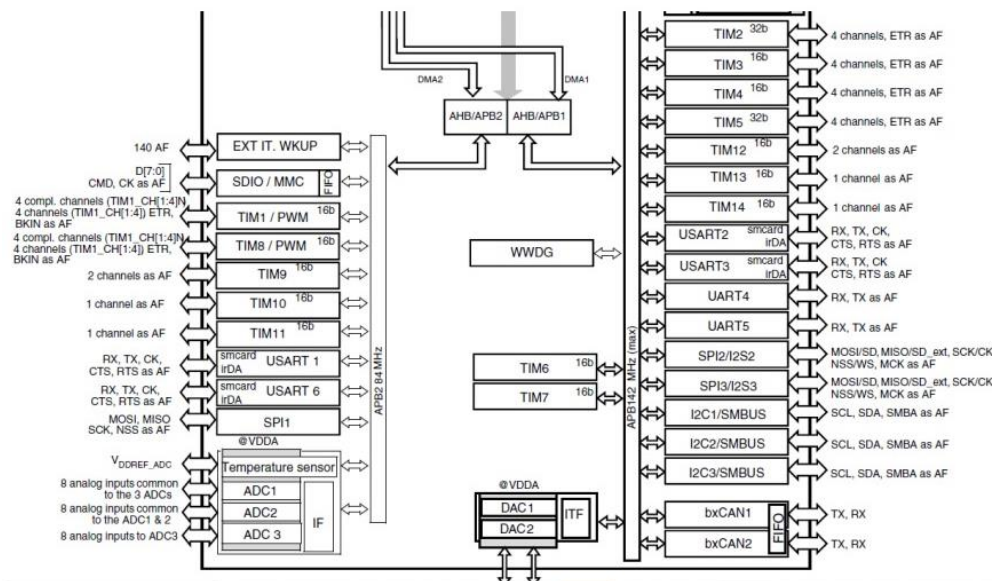


Figura 11. Diagrama buses señales de reloj STM32F411 E-DISCO.

Todos estos dispositivos al ocupar diferentes velocidades de la señal de reloj se tuvieron que establecer de este modo ya que utilizaremos los buses principales de las señales de reloj en el cual nuestra comunicación y el funcionamiento del sensor estarán ligados al bus APB2 y el uso de nuestra interfaz estarán conectados al APB1, para que de este modo poder trabajar a velocidades deseadas.

Por último, para terminar la parte de las configuraciones establecidas en el micro podemos ver cómo es que quedarán las conexiones que se realizarán en este, combinando todo lo realizado en las distintas pruebas.

Librerías.

Dentro de la librería del Sensor HC-SR04 podemos ver que como se mencionó anteriormente, este está trabajando mediante interrupciones ya que solo cuando lo desea el programador solicitara que le mande el valor de la distancia a la que se encuentra el objeto, pero este se encuentra trabajando todo el tiempo.

```
10 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
11 {
12     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
13     {
14         if(flag_captured == 0)
15         {
16             t_ini = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
17             flag_captured = 1;
18             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);
19         }
20
21         else if(flag_captured == 1)
22         {
23             t_end = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
24             __HAL_TIM_SetCounter(htim, 0);
25
26             if(t_end > t_ini){
27                 t_time = t_end - t_ini;
28             }
29             else if(t_ini > t_end){
30                 t_time = (0xFFFF - t_ini) + t_end;
31             }
32
33             dist = (uint16_t)((t_time*0.034)/2);
34             flag_captured = 0;
35
36             __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
37             __HAL_TIM_DISABLE_IT(&htim1, TIM_IT_CC1);
38         }
39     }
40 }
41 void HCSR04_Init(void)
42 {
43     HAL_GPIO_WritePin(Trigger_GPIO_Port, Trigger_Pin, GPIO_PIN_RESET);
44     HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_1);
45 }
```

Figura 14. Librería utilizada para el sensor HC-SR04.

Código.

Ahora analizaremos el código parte por parte para comprender su funcionamiento.

- *Librerías*

```
18 /* USER CODE END Header */
19 /* Includes -----
20 #include "main.h"
21 #include "i2c.h"
22 #include "spi.h"
23 #include "tim.h"
24 #include "gpio.h"
25
26 /* Private includes -----
27 /* USER CODE BEGIN Includes */
28 #include "i2c-lcd.h"
29 #include "hc_sr04.h"
30 #include <stdio.h>
31 /* USER CODE END Includes */
```

Figura 15. Código - librerías.

Primero es necesario usar todas las librerías necesarias para que pueda funcionar nuestro código.

Aquí podemos observar que ya estamos incluyendo las librerías necesarias para nuestro lcd y para el sensor HC-SR04, aparte de las necesarias para la programación hall.

- *Variables*

```
54 /* Private function prototypes -----
55 void SystemClock_Config(void);
56 /* USER CODE BEGIN PFP */
57 uint8_t p;
58 uint16_t b;
59 uint16_t a;
60 uint16_t error;
61
62 uint8_t spiRxBuf[3];
63 uint8_t spiTxBuf[3];
64 uint8_t spiTxBuf_D;
65 uint32_t dato;
66
67 uint8_t unidades_a, decenas_a, centenas_a;
68 uint8_t unidades_b, decenas_b, centenas_b;
69 char unidad_str[2], decena_str[2], centena_str[2];
70 uint8_t distancia;
71 volatile uint8_t despachado = 0;
72 /* USER CODE END PFP */
```

Figura 16. Código - Variables.

Posteriormente tenemos nuestras variables necesarias en todo el código para el almacenado de datos y para la toma de decisiones del micro.

- *Funciones*

```

77 void descomponer_numero(uint16_t a, uint8_t *unidades, uint8_t *decenas, uint8_t *centenas) {
78     *unidades = a % 10; // Unidades de 'a'
79     *decenas = (a / 10) % 10; // Decenas de 'a'
80     *centenas = (a / 100) % 10; // Centenas de 'a'
81 }
82
83 void itoa_manual(uint8_t num, char *str) {
84     str[0] = num + '0'; // Convertir el número a carácter ASCII
85     str[1] = '\0'; // Terminar la cadena con el carácter nulo
86 }
87
88 void waitthalget(uint32_t tickstart, uint16_t tiempo_espera){
89     while((HAL_GetTick() - tickstart) < tiempo_espera){
90     }
91 }
92
93 int valor = 0;
94 /* USER CODE END 0 */

```

Figura 17. Código - Funciones.

Tenemos una función para descomponer números y poder proyectarlos en el display, ya que la nexys nos mandará los números completos, pero para poder proyectarlos necesitamos dividirlo en unidades, decenas y centenas.

También tenemos una función para convertir estas unidades en un string ya que la librería del LCD ocupa que los datos que se quieran proyectar se encuentren en este formato.

Por último, tenemos la función del GetTick que ocuparemos en caso de que queramos esperar, pero así nuestro micro sigue procesando información y no detiene todo para poder esperar a que pase el tiempo deseado.

- *Main*

```

123 /* Initialize all configured peripherals */
124 MX_GPIO_Init();
125 MX_I2C1_Init();
126 MX_SPI1_Init();
127 MX_TIM1_Init();
128 /* USER CODE BEGIN 2 */
129 lcd_init();
130 HCSR04_Init();
131 //Entradas de la comunicacion en serie.
132
133 //Variables necesarias para el codigo
134 uint8_t unidades_a, decenas_a, centenas_a;
135 uint8_t unidades_b, decenas_b, centenas_b;
136 char unidad_str[2], decena_str[2], centena_str[2];
137 uint8_t distancia;
138 uint8_t despachado = 0;
139
140 //Comunicacion
141 spiRxBuf [0] = 0x00;
142 spiRxBuf [1] = 0x00;
143 spiRxBuf [2] = 0x00;
144
145 /* USER CODE END 2 */

```

Figura 18. Código - Main.

Dentro de nuestro main inicializamos sobre todo el sensor y el display para que empiecen a funcionar, y se inicializan algunas variables, sobre todo se establecen a 0 las variables encargadas a la comunicación con la nexys, las cuales son spiRxBuf, la

cual es una cadena de 3 bytes ya que el tamaño de los datos recibidos por parte de la nexys requiere que sea de este tamaño nuestra variable.

- *While*

```
149 while (1)
150 {
151
152     if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_0)) { // 10e
153         spiTxBuf_D |= (1 << 2); // Establece el bit 2 a 1
154     } else {
155         spiTxBuf_D &= ~(1 << 2); // Establece el bit 2 a 0
156     }
157
158     if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1)) { // 20e
159         spiTxBuf_D |= (1 << 3); // Establece el bit 3 a 1
160     } else {
161         spiTxBuf_D &= ~(1 << 3); // Establece el bit 3 a 0
162     }
163
164     if (HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_2)) { // 50 e
165         spiTxBuf_D |= (1 << 4); // Establece el bit 4 a 1
166     } else {
167         spiTxBuf_D &= ~(1 << 4); // Establece el bit 4 a 0
168     }
169
170     if (HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_7)) { //100 e
171         spiTxBuf_D |= (1 << 5); // Establece el bit 1 a 1
172     } else {
173         spiTxBuf_D &= ~(1 << 5); // Establece el bit 1 a 0
174     }
175
176     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_6)) { // Derecha
177         spiTxBuf_D |= (1 << 7); // Establece el bit 7 a 1
178     } else {
179         spiTxBuf_D &= ~(1 << 7); // Establece el bit 7 a 0
180     }
181
182     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_7)) { // Mc
183         spiTxBuf_D |= (1 << 6); // Establece el bit 6 a 1
184     } else {
185         spiTxBuf_D &= ~(1 << 6); // Establece el bit 6 a 0
186     }
187
188     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_5)) { // Continue
189         spiTxBuf_D |= (1 << 1); // Establece el bit 5 a 1
190     } else {
191         spiTxBuf_D &= ~(1 << 1); // Establece el bit 5 a 0
192     }
193
194     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_4)) { // Izquierda
195         spiTxBuf_D |= (1 << 0); // Establece el bit 0 a 1
196     } else {
197         spiTxBuf_D &= ~(1 << 0); // Establece el bit 0 a 0
198     }
199
200     spiTxBuf[0] = spiTxBuf_D;
201     spiTxBuf[1] = spiTxBuf_D;
202     spiTxBuf[2] = spiTxBuf_D;
```

Figura 19. Código – While (Asignación a variable de estado de botonería).

Dentro de nuestro while empezamos con la parte de los botones, los cuales le asignaran un 1 o un 0 dependiendo de si esta activado o desactivado nuestro pulsador para posteriormente enviarlo a la nexys.


```

241 //Enviar datos
242 HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET); //Recibir datos de la FPGA
243 // Transmitir 1 byte y recibir 3 bytes simultáneamente (full-duplex)
244 if (HAL_SPI_TransmitReceive(&hspi1, spiTxBuf, spiRxBuf, 3, HAL_MAX_DELAY) == HAL_OK) {
245 // Procesar los datos recibidos
246 dato = (spiRxBuf[0] << 16) | (spiRxBuf[1] << 8) | spiRxBuf[2];
247 }
248 HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET); //Dejar de recibir datos de la FPGA
249
250 error = (dato>>1) & 0x1;
251 p = (dato>>2) & 0x3;
252 b = (dato>>4) & 0x3FF;
253 a = (dato>>14) & 0x3FF;

```

Figura 20. Código – While (Comunicación).

Aquí podemos ver que empieza la comunicación, inicializamos el pin GPIOE_2 el cual le avisará a la nexys que comenzará con la comunicación.

Posteriormente empieza a recibir ya a enviar, almacenando los datos recibidos por parte de la nexys en sus variables respectivas desglosando el dato obtenido.

Y finalmente debemos especificar el final de la comunicación desactivando el pin GPIOE_2 para que la nexys deje de mandar datos y que sepa que terminaron de mandarse los datos.

```

255 if (error == 0) {
256     lcd_enviar("Precio: ", 0, 0);
257     switch (p) {
258         case 0b00: // 00
259             lcd_enviar("1,00e", 0, 8);
260             break;
261         case 0b10: // 10
262             lcd_enviar("5,30e", 0, 8);
263             break;
264         case 0b01: // 01
265             lcd_enviar("2,80e", 0, 8);
266             break;
267         default:
268             lcd_clear();
269             break;
270     }
271 }
272 else {
273     lcd_clear();
274     lcd_enviar("Devolucion", 0, 2);
275     descomponer_numero(a, &unidades_a, &decenas_a, &centenas_a);
276     itoa_manual(unidades_a, unidad_str); // Convierte unidades a string
277     itoa_manual(decenas_a, decena_str); // Convierte decenas a string
278     itoa_manual(centenas_a, centena_str); // Convierte centenas a string
279     lcd_enviar(centena_str, 1, 5);
280     lcd_enviar(",", 1, 6);
281     lcd_enviar(decena_str, 1, 7);
282     lcd_enviar(unidad_str, 1, 8);
283     lcd_enviar("e", 1, 9);
284 }

```

Figura 21. Código – While (Menú de precios en interfaz en LCD).

Después tenemos el inicio del código de la interfaz, en este caso podemos observar inicialmente solo la parte que proyecta el precio del refresco dependiendo si la persona presiona o no un botón el cual le manda la señal a la nexys y dependiendo lo que reciba de vuelta desplegará el precio del refresco.


```

393     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET); //Recibir datos de la FPGA
394     // Transmitir 1 byte y recibir 3 bytes simultaneamente (full-duplex)
395     if (HAL_SPI_TransmitReceive(&hspi1, spiTxBuf, spiRxBuf, 3, HAL_MAX_DELAY) == HAL_OK) {
396         // Procesar los datos recibidos
397         dato = (spiRxBuf[0] << 16) | (spiRxBuf[1] << 8) | spiRxBuf[2];
398     }
399     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET); //Dejar de recibir datos de la FPGA
400 }else{
401     spiTxBuf_D &= ~(1 << 1); //Continue a 0
402
403     spiTxBuf[0] = spiTxBuf_D;
404     spiTxBuf[1] = spiTxBuf_D;
405     spiTxBuf[2] = spiTxBuf_D;
406
407     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET); //Recibir datos de la FPGA
408     // Transmitir 1 byte y recibir 3 bytes simultaneamente (full-duplex)
409     if (HAL_SPI_TransmitReceive(&hspi1, spiTxBuf, spiRxBuf, 3, HAL_MAX_DELAY) == HAL_OK) {
410         // Procesar los datos recibidos
411         dato = (spiRxBuf[0] << 16) | (spiRxBuf[1] << 8) | spiRxBuf[2];
412     }
413     HAL_GPIO_WritePin(GPIOE, GPIO_PIN_2, GPIO_PIN_RESET); //Dejar de recibir datos de la FPGA
414 }
415 }
416 }
417
418 lcd_clear();
419

```

Figura 22. Código – While (Texto proyectado en la interfaz del LCD).

Ya finalmente observamos el resto del código necesario principalmente para lo que se quiere que se proyecte en la interfaz con las decisiones tomadas en la nexys, por lo que este código relativamente es más sencillo ya que toda la toma de decisiones es procesada por la FPGA.

Podemos observar que volvemos a activar la comunicación posteriormente con la nexys, esto es debido a que en este punto nos encontramos en un estado de error y debemos asegurarnos a que salgamos de este estado y por ello se queda en este bucle hasta que realicemos la acción necesaria para esto y por ello es necesario volver a inicializar la comunicación en este punto.

RESULTADOS

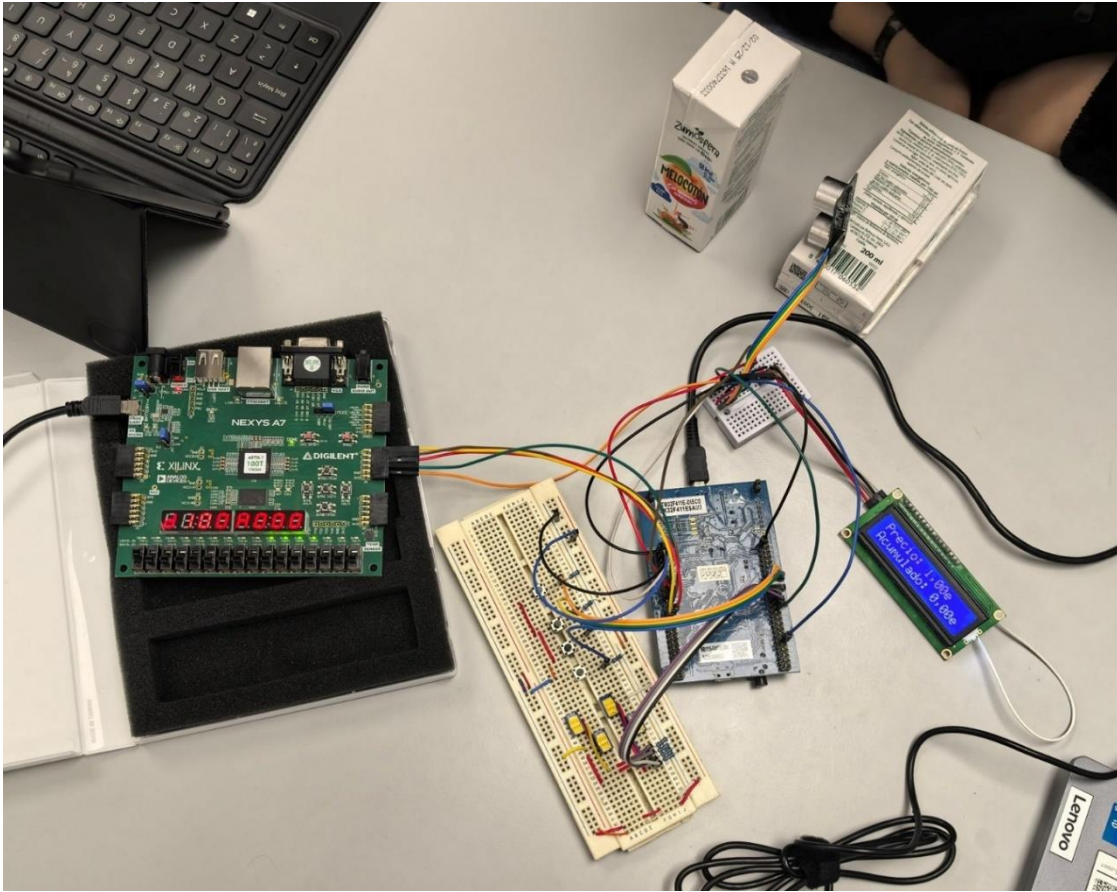


Figura 23. Proyecto funcionando.

La integración de dispositivos como la FPGA y el microcontrolador demuestra la importancia de combinar tecnologías complementarias para desarrollar sistemas funcionales y eficientes. Mientras que la FPGA sobresale en tareas de procesamiento paralelo y toma de decisiones complejas, el microcontrolador se especializa en la recopilación y transmisión de información desde sensores y periféricos, actuando como puente entre el usuario y el sistema principal.

La correcta implementación de protocolos de comunicación como I2C y SPI asegura la sincronización y fiabilidad en la transferencia de datos, siendo muy importante para proyectos de este tipo.

En resumen, la colaboración entre dispositivos con capacidades específicas demuestra ser esencial para abordar problemas complejos, maximizando las fortalezas de cada componente y garantizando un sistema robusto y funcional como el que logramos desarrollar en este proyecto.