CESE 2018 –

PROYECTO FINAL DE LA MATERIA PROTOCOLOS EN S.E.

Título “Ensayos en Buses I2C y Serial en placa CL3 con ARM STM32F411”

CESE2018 Rafael Oliva v21-08- 2018

1. Introducción:

El presente trabajo pretende realizar un ensayo básico de los buses I2C, y Serial de la placa CL3 que utilizaré para el Proyecto Final de la Especialización. Pendiente: Ensayo de SPI/FFS y de un ESP8266 (que la placa incorpora) conectado a uno de los puertos serie – presentar un ensayo de comunicación básica con comandos AT.

2. Antecedentes de la placa a utilizar:

Se encaró el desarrollo del Firmware de una placa denominada CL3 que migra desde una versión previa AVR de 8 bits (CL2, desarrollada en 2009 de la firma L&R Ingeniería) a una versión ARM-Cortex M4F de 32 bits manteniendo características del formato físico anterior. Se busca que dicho Firmware tenga características de facilidad de uso con un enfoque similar a la sAPI desarrollada para la CIAA. Según su propia definición, la sAPI es una biblioteca de funciones de código abierto, que funciona como HAL (Hardware Abstraction Layer, o Capa de Abstracción de Hardware) y resulta portable a diferentes placas con distintos microcontroladores. Dichas funciones proporcionan acceso a periféricos internos del microcontrolador (GPIO, UART, ADC, etc), módulos de alto nivel como Delay (retardo), PWM, Buffer Circular o Print (impresión a través de UART), y acceso a módulos externos específicos.

La estructura de diagrama en bloques y la foto de la placa original CL2 se aprecia en la Figura 1. Dado que la mayoría de las aplicaciones para las que se utilizó involucran el registro de variables eléctricas y/o meteorológicas de sistemas de energía renovable (fundamentalmente generadores fotovoltaicos y eólicos de baja potencia, en sistemas aislados) se la dotó de una interfaz para tarjetas SD industriales y de un reloj de tiempo real con oscilador compensado por temperatura (TCXO), para reducir significativamente la variación en los registros temporales.

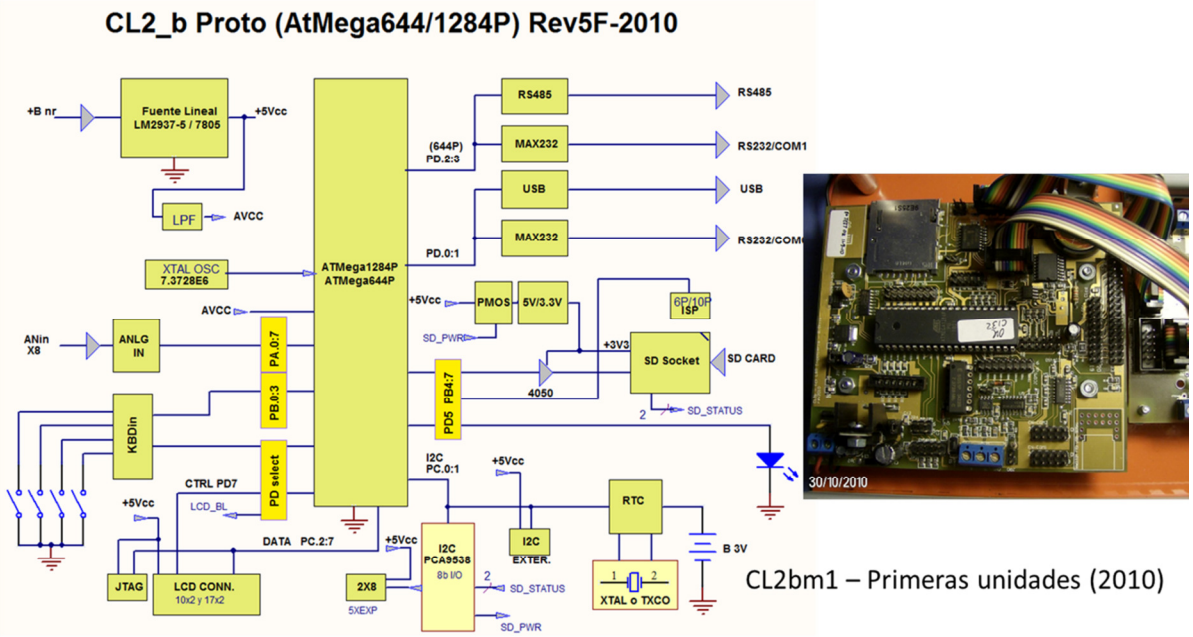


Fig. 1. Diagrama de la versión inicial CL2 de 2010 y foto de las primeras unidades

3. Descripción de la placa CL3 a utilizar:

En la placa CL3 se mantuvieron las características mecánicas, el oscilador compensado por temperatura y la interfaz SD industrial. El desarrollo se realiza actualmente en cooperación con el Ing. Leonardo Garberoglio (GADIB-UTN/FRSN), y su diagrama en bloques se muestra en la Figura 2. La foto de la placa prototipo actualmente en ensayo se aprecia en la Figura 3.

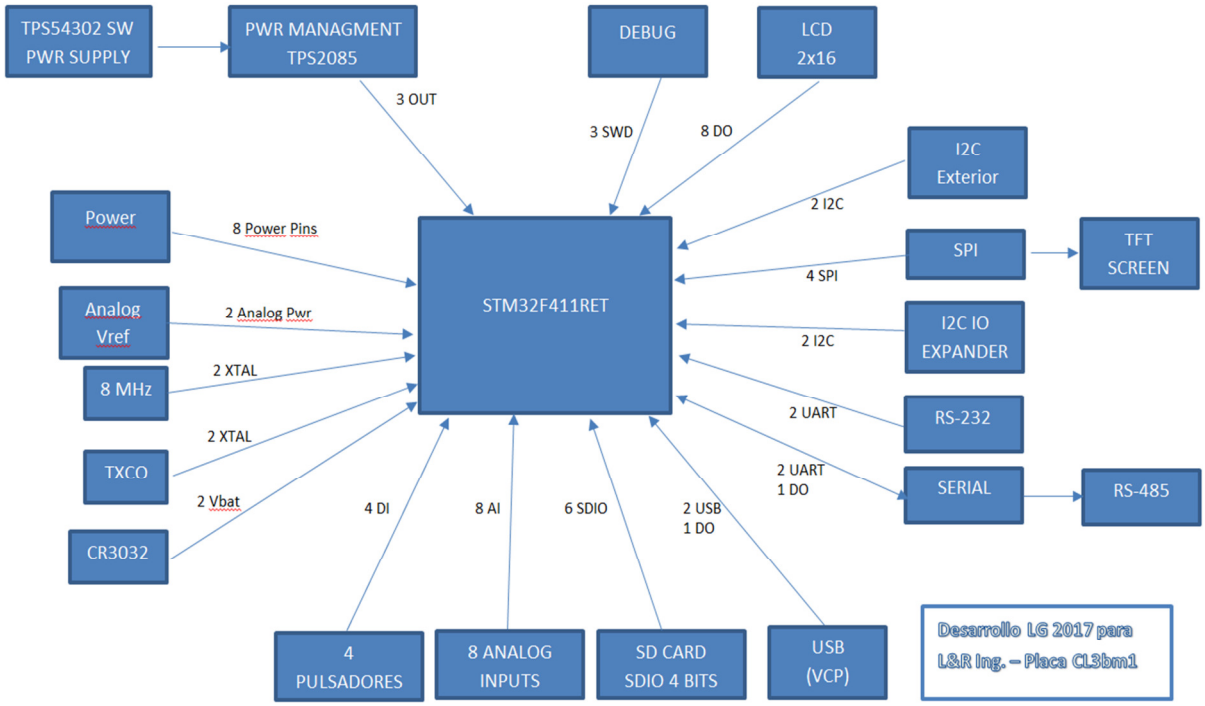


Fig. 2. Diagrama en bloques del CL3 (08-2017)

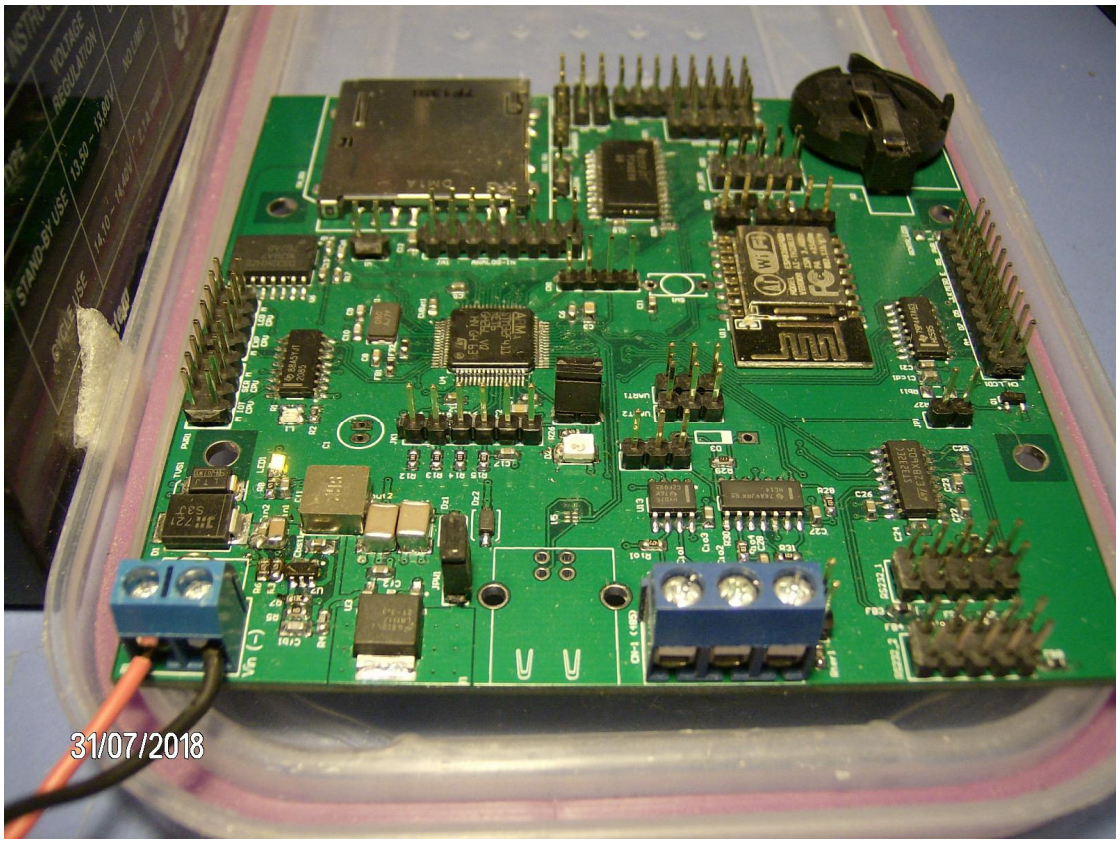


Fig. 3. Foto de la placa CL3 - primer prototipo en ensayos 07-2018

4. Descripción de los periféricos a ensayar en la placa CL3:

En la placa CL3 todavía no se ha desarrollado un BSP (Board Support Package) completo, por ahora se van configurando y ensayando por partes los periféricos. Para ello se utiliza el programa CubeMX de ST, que facilita la inicialización de los módulos internos. Los periféricos ensayados son la UART6 -> Puerto Serial 2 y el Expansor PCA9539 conectado via I2C1 al controlador principal

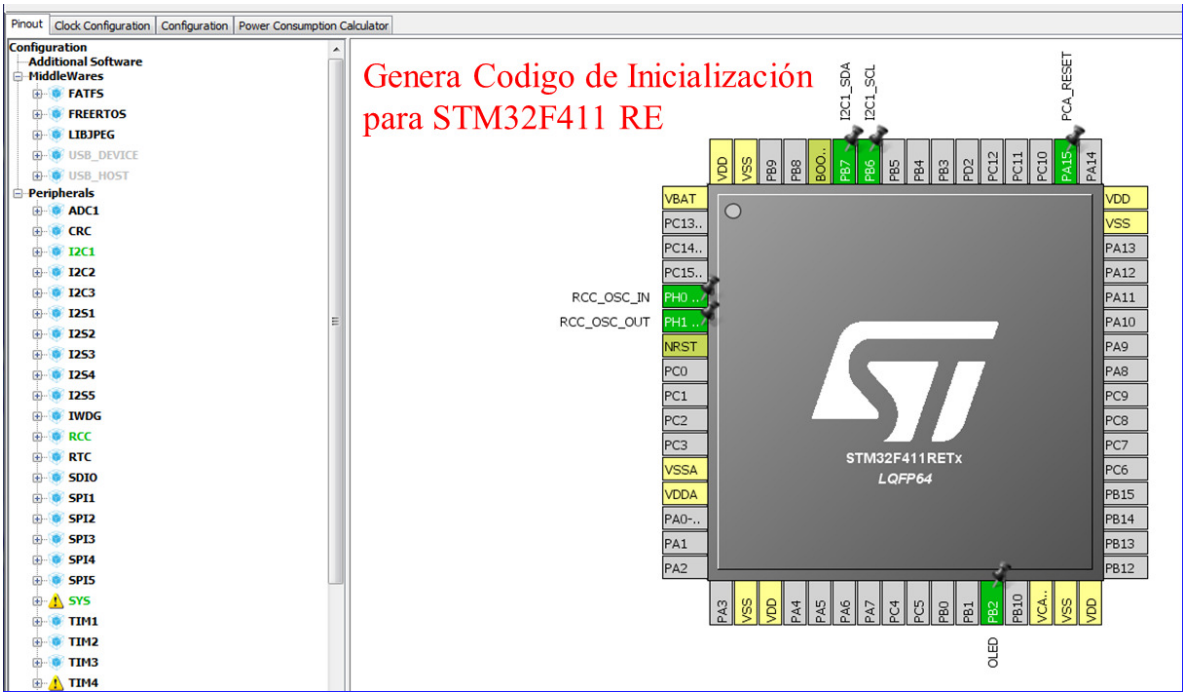


Fig. 4. Inicializacion con CubeMX de la placa CL3

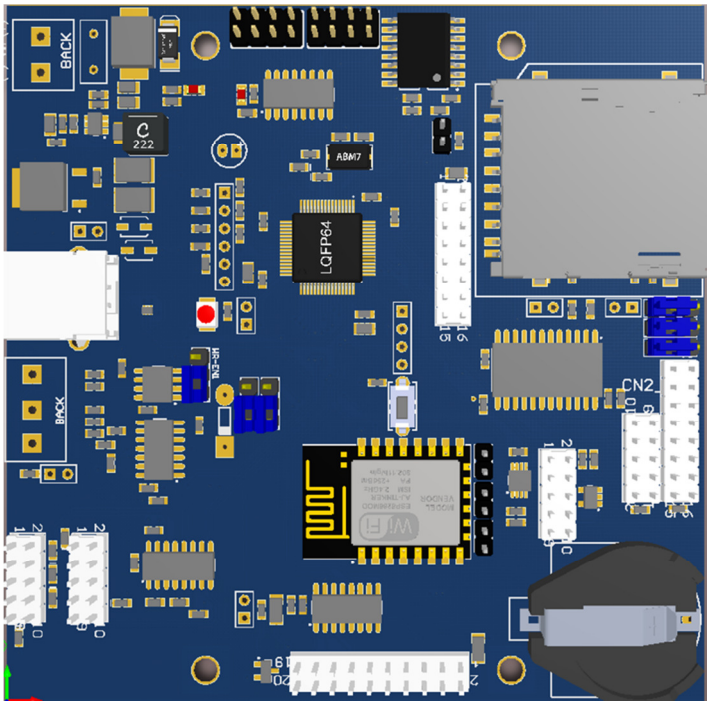


Fig. 5. CL3 - Primer render del circuito impreso (08-2017)

5. Descripción de los ENSAYOS REALIZADOS la placa CL3:

5.1 Ensayos Serial: En la placa CL3 el circuito de los puertos serie se muestra en la Figura 6. Se usó la conexión RS232_2 a la UART6 para los ensayos. No se conectó el ESP finalmente.

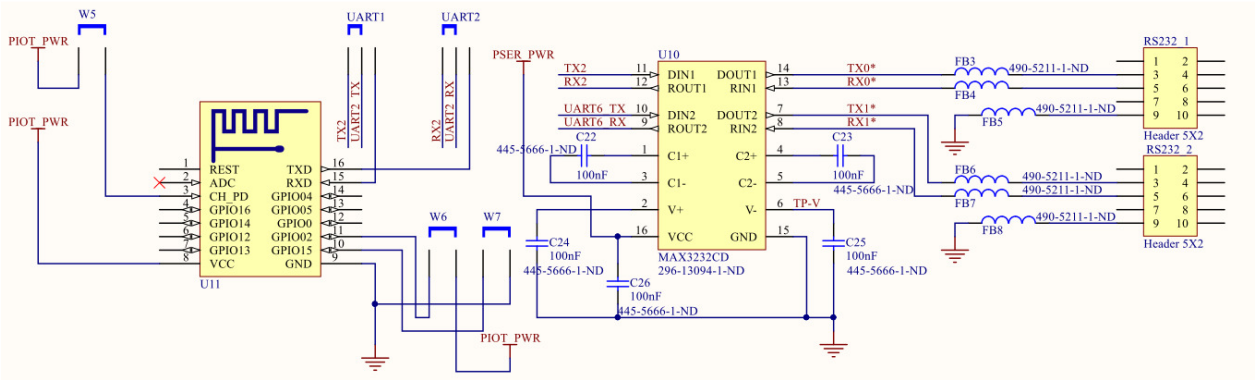


Fig. 6. CL3 – Puerto Serie dual y conexión a ESP8266

5.2 Ensayos I2C: En la placa CL3 el expansor de I/O 16 salidas PCA9539 está conectado al bus I2C1, según se muestra en la Figura 7. Se conectaron dos LEDs a los puertos IO4, IO5 para demostrar su uso.

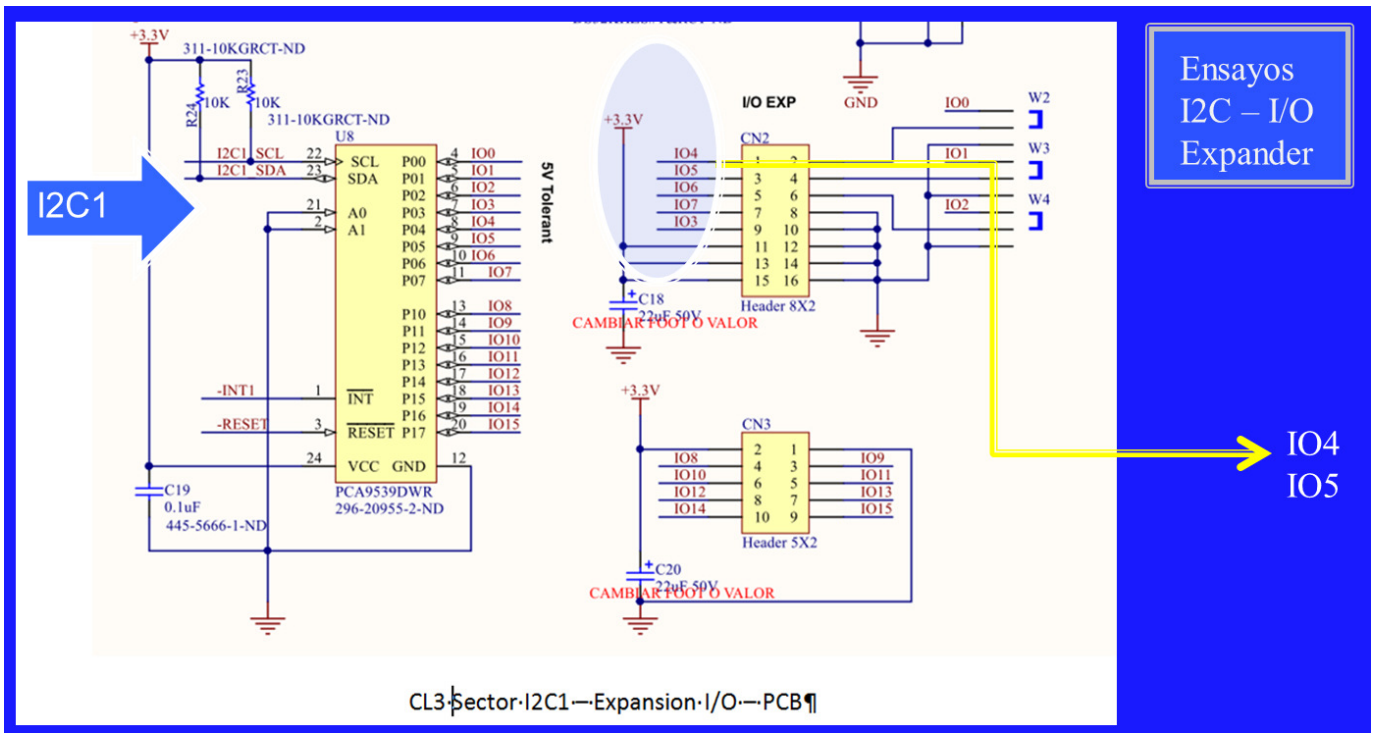


Fig. 7. CL3 – Puerto I2C, expansor PCA9539

6. DIAGRAMA DE ENSAYOS REALIZADOS la placa CL3:

6.1 Ensayos Serial: En la figura 8 se aprecia el conexionado básico de los ensayos y en la Figura 9 se aprecia el ensayo realizado.

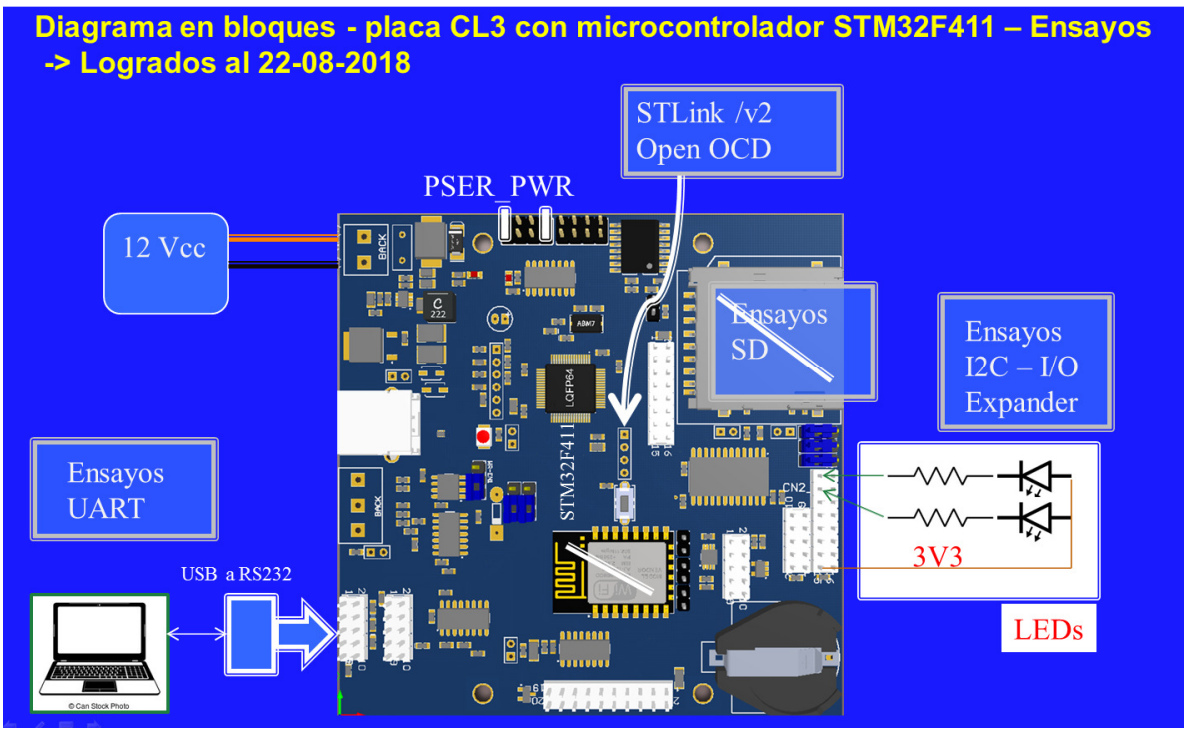


Fig. 8. CL3 – Puerto I2C, expansor PCA9539

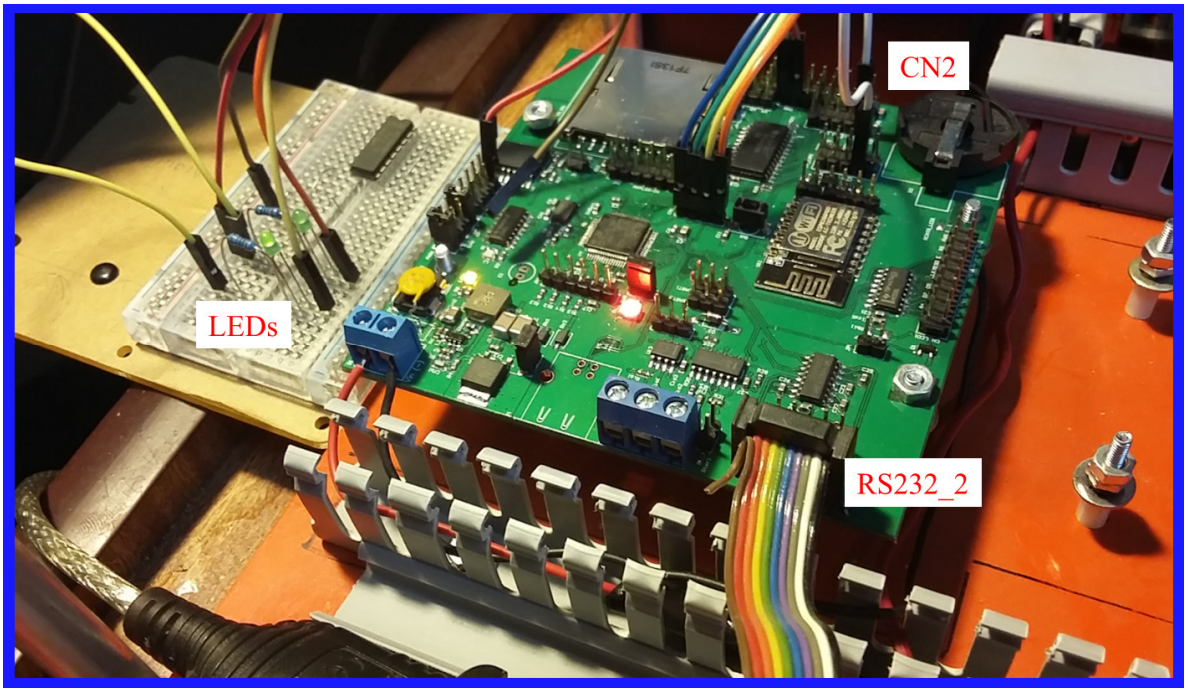
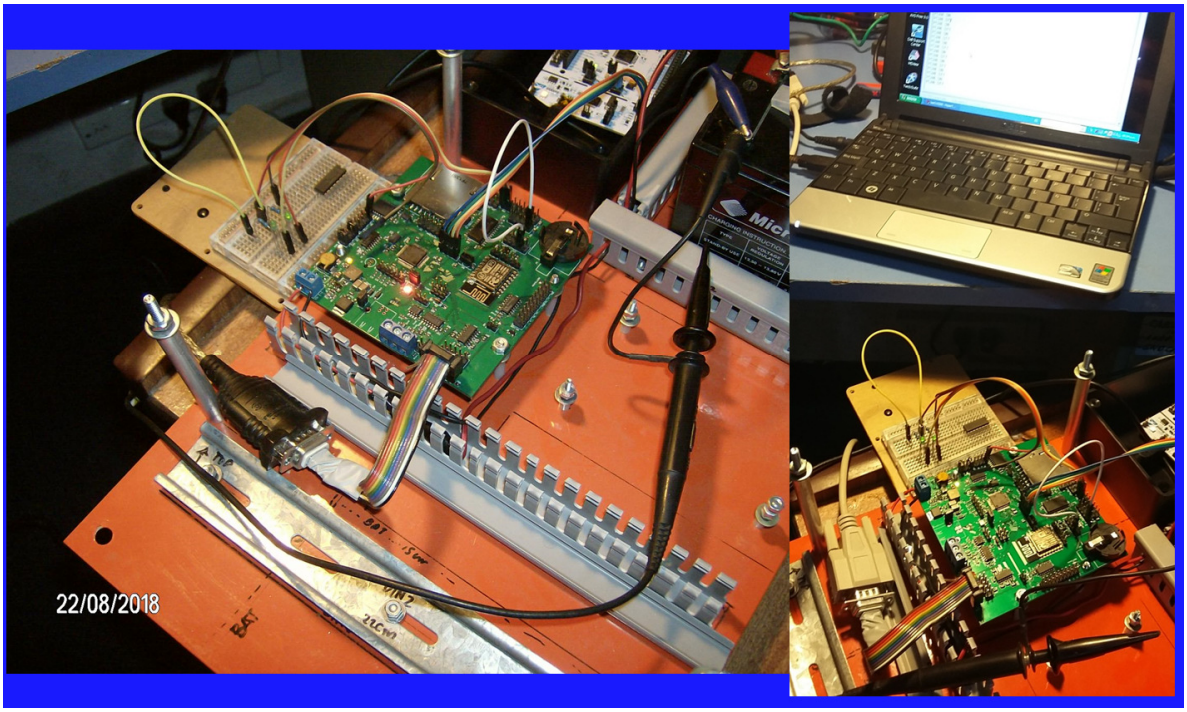


Fig. 9. Ensayo sobre placa

El ensayo consistió en la inicialización del sistema, preparación de puertos serie e I2C utilizando funciones HAL de STM, configuración del chip PCA via comandos I2C y encendido secuencial de LEDs, alterable por el usuario a través de un getchar() insertado en el lazo principal.



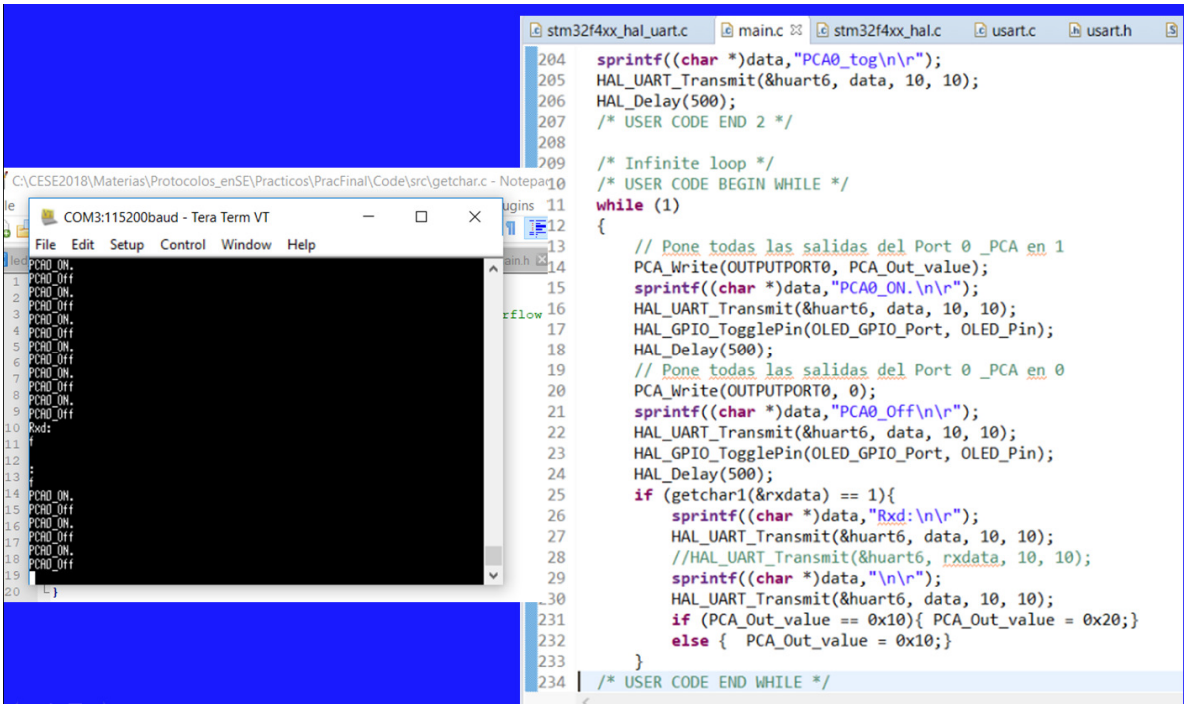


Fig. 10. EnsayoS sobre placa 2,3

7. CODIGO FUENTE – DEPURACION EN ECLIPSE/AC6 STM

7.1 SOURCE

```
7.2
7.3 /**
7.4   ******************************************************************************
7.5   * TP FINAL PCSE R. OLIVA 08-2018
7.6   * @file           : main.c
7.7   * @brief          : Main program body
7.8   ******************************************************************************
7.9   ** This notice applies to any and all portions of this file
7.10  * that are not between comment pairs USER CODE BEGIN and
7.11  * USER CODE END. Other portions of this file, whether
7.12  * inserted by the user or by software development tools
7.13  * are owned by their respective copyright owners.
7.14  *
7.15  * COPYRIGHT(c) 2018 STMicroelectronics
```

```
7.16  *
7.17  * Redistribution and use in source and binary forms, with or without modification,
7.18  * are permitted provided that the following conditions are met:
7.19  *   1. Redistributions of source code must retain the above copyright notice,
7.20  *      this list of conditions and the following disclaimer.
7.21  *   2. Redistributions in binary form must reproduce the above copyright notice,
7.22  *      this list of conditions and the following disclaimer in the documentation
7.23  *      and/or other materials provided with the distribution.
7.24  *   3. Neither the name of STMicroelectronics nor the names of its contributors
7.25  *      may be used to endorse or promote products derived from this software
7.26  *      without specific prior written permission.
7.27  *
7.28  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
7.29  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
7.30  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
7.31  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
7.32  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
7.33  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
7.34  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
7.35  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
7.36  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
7.37  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
7.38  *
7.39  ******************************************************************************
7.40  */
7.41/* Includes ------------------------------------------------------------------*/
7.42#include "main.h"
7.43#include "stm32f4xx_hal.h"
7.44#include "i2c.h"
7.45#include "usart.h"
7.46#include "gpio.h"
7.47
7.48/* USER CODE BEGIN Includes */
7.49#define INPUTPORT0  0x00
7.50#define INPUTPORT1  0x01
7.51#define OUTPUTPORT0 0x02
7.52#define OUTPUTPORT1 0x03
7.53#define POLINVPORT0 0x04
7.54#define POLINVPORT1 0x05
7.55#define CONFIGPORT0 0x06
7.56#define CONFIGPORT1 0x07
7.57
7.58#define ALLOUTPUT   0x00
7.59#define ALLINPUT    0xFF
7.60
7.61#define PCA_ADDR    0xE8
7.62/* USER CODE END Includes */
7.63
7.64/* Private variables ---------------------------------------------------------*/
7.65
7.66/* USER CODE BEGIN PV */
7.67/* Private variables ---------------------------------------------------------*/
7.68
7.69/* USER CODE END PV */
7.70
7.71/* Private function prototypes -----------------------------------------------*/
7.72void SystemClock_Config(void);
7.73uint8_t getchar1(uint8_t * ch);
7.74
7.75/* USER CODE BEGIN PFP */
7.76/* Private function prototypes -----------------------------------------------*/
7.77
7.78/* USER CODE END PFP */
7.79
7.80/* USER CODE BEGIN 0 */
7.81uint8_t PCA_Read(uint8_t registerAddress)
7.82{
7.83  uint8_t data[3] = {0,0,0};
7.84
7.85  //Simple ID reg reading
7.86  data[0]=registerAddress;
7.87  data[1]=0;
7.88  HAL_I2C_Master_Transmit(&hi2c1, PCA_ADDR, data, 1, 100);
7.89
7.90  //
7.91  HAL_I2C_Master_Receive(&hi2c1, PCA_ADDR+1, data, 1, 100);
7.92
7.93  return data[0];
7.94}
7.95
```

```
7.96void PCA_Write(uint8_t registerAddress, uint8_t dataWrite)
7.97{
7.98  uint8_t data[2] = {0,0};
7.99
7.100   //Simple ID reg reading
7.101   data[0]=registerAddress;
7.102   data[1]=dataWrite;
7.103   HAL_I2C_Master_Transmit(&hi2c1, PCA_ADDR, data, 2, 100);
7.104
7.105 }
7.106
7.107 // Configura el pin "pin" del puerto "port" como entrada (direccion=1) o como salida (direccion=0)
7.108 void PCA_ConfigPort(uint8_t port, uint8_t pin, uint8_t direction)
7.109 {
7.110   uint8_t configActual=0;
7.111
7.112   configActual = PCA_Read(CONFIGPORT0+port);
7.113   configActual &= ~(1<<pin);              // Pongo el 0 el bit correspondiente el pin que quiero modificar
7.114   configActual |= direction<<pin;                 //  Dejo  en  0  o  pongo  en  1  el  bit  correspondiente  al  pin
    dependiendo del valor direction
7.115   PCA_Write(CONFIGPORT0+port, configActual);      //EscriBo el nuevo valor en el registro correspondiente
7.116 }
7.117
7.118 /*
7.119 Posted by MarkV6 on 2017-10-24 11:54
7.120 Here my non blocking getchar. You need to clear the overflow otherwise the rx available will not rise again
7.121 http://www.openstm32.org/forumthread5015
7.122 */
7.123
7.124 uint8_t getchar1(uint8_t * ch)
7.125 {
7.126        // check for overflow and clear
7.127   if (__HAL_UART_GET_FLAG(&huart6, UART_FLAG_ORE))
7.128           __HAL_UART_CLEAR_FLAG(&huart6, UART_FLAG_ORE);
7.129
7.130   if (__HAL_UART_GET_FLAG(&huart6, UART_FLAG_RXNE))
7.131   {
7.132           *ch = huart6.Instance->DR & 0x1FF;
7.133           return 1;
7.134   }
7.135
7.136   return 0;
7.137 }
7.138
7.139 /* USER CODE END 0 */
7.140
7.141 /**
7.142   * @brief  The application entry point.
7.143   *
7.144   * @retval None
7.145   */
7.146 int main(void)
7.147 {
7.148   /* USER CODE BEGIN 1 */
7.149
7.150   /* USER CODE END 1 */
7.151
7.152   /* MCU Configuration--------------------------------------------------------*/
7.153
7.154   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
7.155   HAL_Init();
7.156
7.157   /* USER CODE BEGIN Init */
7.158
7.159   /* USER CODE END Init */
7.160
7.161   /* Configure the system clock */
7.162   SystemClock_Config();
7.163
7.164   /* USER CODE BEGIN SysInit */
7.165
7.166   /* USER CODE END SysInit */
7.167
7.168   /* Initialize all configured peripherals */
7.169   MX_GPIO_Init();
7.170   MX_I2C1_Init();
7.171   MX_USART6_UART_Init();
7.172
7.173   /* USER CODE BEGIN 2 */
7.174   uint8_t valor;
```

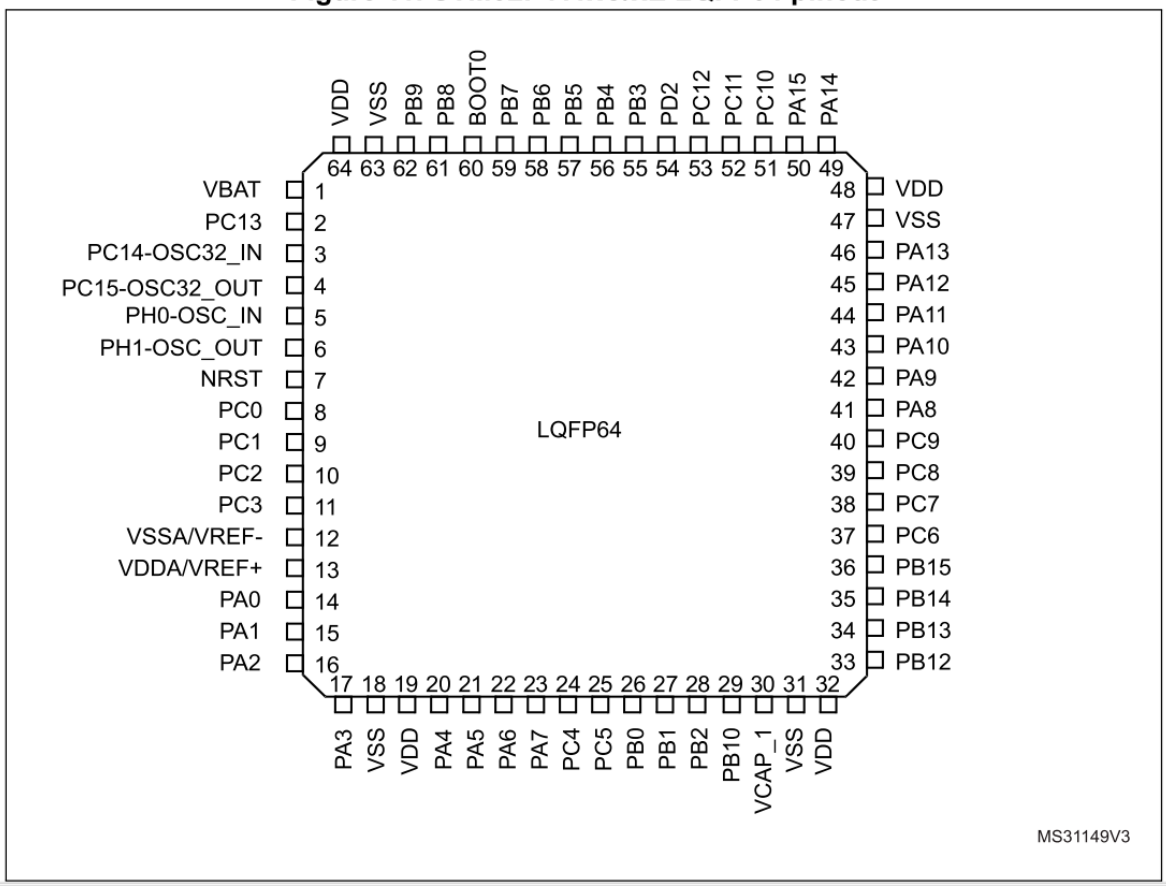
```
7.175   uint8_t data[10];
7.176   uint8_t rxdata = 0;
7.177   uint8_t PCA_Out_value = 0x10;
7.178
7.179   sprintf((char *)data,"CESE\n\r");
7.180   HAL_UART_Transmit(&huart6, data, 10, 10);
7.181   HAL_Delay(100);
7.182   sprintf((char *)data,"PCSE TPf\n\r");
7.183   HAL_UART_Transmit(&huart6, data, 10, 10);
7.184   sprintf((char *)data,"R.Oliva.\n\r");
7.185   HAL_UART_Transmit(&huart6, data, 10, 10);
7.186
7.187   HAL_GPIO_TogglePin(OLED_GPIO_Port, OLED_Pin);
7.188   HAL_Delay(500);
7.189
7.190   sprintf((char *)data,"I2C\n\r");
7.191   HAL_UART_Transmit(&huart6, data, 10, 10);
7.192
7.193   valor = PCA_Read(CONFIGPORT0);
7.194   valor = PCA_Read(CONFIGPORT1);
7.195   // Tentativamente esto es Todo el PORT0 = Salidas
7.196   PCA_Write(CONFIGPORT0, 0);
7.197   //Configuración de a un pin..
7.198   //Configuro el Pin1 del puerto 1 como salida
7.199   PCA_ConfigPort(1, 1, 0);
7.200   HAL_GPIO_TogglePin(OLED_GPIO_Port, OLED_Pin);
7.201   HAL_Delay(500);
7.202
7.203   // Verifico que el valor se grabó correctamente en el registro
7.204   valor = PCA_Read(CONFIGPORT1);
7.205
7.206   sprintf((char *)data,"PCA0_tog\n\r");
7.207   HAL_UART_Transmit(&huart6, data, 10, 10);
7.208   HAL_Delay(500);
7.209   /* USER CODE END 2 */
7.210
7.211   /* Infinite loop */
7.212   /* USER CODE BEGIN WHILE */
7.213   while (1)
7.214   {
7.215       // Pone todas las salidas del Port 0 _PCA en 1
7.216     PCA_Write(OUTPUTPORT0, PCA_Out_value);
7.217     sprintf((char *)data,"PCA0_ON.\n\r");
7.218     HAL_UART_Transmit(&huart6, data, 10, 10);
7.219     HAL_GPIO_TogglePin(OLED_GPIO_Port, OLED_Pin);
7.220       HAL_Delay(500);
7.221       // Pone todas las salidas del Port 0 _PCA en 0
7.222       PCA_Write(OUTPUTPORT0, 0);
7.223     sprintf((char *)data,"PCA0_Off\n\r");
7.224     HAL_UART_Transmit(&huart6, data, 10, 10);
7.225     HAL_GPIO_TogglePin(OLED_GPIO_Port, OLED_Pin);
7.226       HAL_Delay(500);
7.227       if (getchar1(&rxdata) == 1){
7.228             sprintf((char *)data,"Rxd:\n\r");
7.229             HAL_UART_Transmit(&huart6, data, 10, 10);
7.230             //HAL_UART_Transmit(&huart6, rxdata, 10, 10);
7.231             sprintf((char *)data,"\n\r");
7.232             HAL_UART_Transmit(&huart6, data, 10, 10);
7.233             if (PCA_Out_value == 0x10){ PCA_Out_value = 0x20;}
7.234             else {  PCA_Out_value = 0x10;}
7.235       }
7.236   /* USER CODE END WHILE */
7.237
7.238   /* USER CODE BEGIN 3 */
7.239
7.240   }
7.241   /* USER CODE END 3 */
7.242
7.243 }
7.244
7.245 /**
7.246   * @brief System Clock Configuration
7.247   * @retval None
7.248   */
7.249 void SystemClock_Config(void)
7.250 {
7.251
7.252   RCC_OscInitTypeDef RCC_OscInitStruct;
7.253   RCC_ClkInitTypeDef RCC_ClkInitStruct;
7.254
```

```
7.255     /**Configure the main internal regulator output voltage
7.256     */
7.257   __HAL_RCC_PWR_CLK_ENABLE();
7.258
7.259   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
7.260
7.261     /**Initializes the CPU, AHB and APB busses clocks
7.262     */
7.263   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
7.264   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
7.265   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
7.266   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
7.267   RCC_OscInitStruct.PLL.PLLM = 4;
7.268   RCC_OscInitStruct.PLL.PLLN = 84;
7.269   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
7.270   RCC_OscInitStruct.PLL.PLLQ = 4;
7.271   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
7.272   {
7.273     _Error_Handler(__FILE__, __LINE__);
7.274   }
7.275
7.276     /**Initializes the CPU, AHB and APB busses clocks
7.277     */
7.278   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
7.279                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
7.280   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
7.281   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
7.282   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
7.283   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
7.284
7.285   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
7.286   {
7.287     _Error_Handler(__FILE__, __LINE__);
7.288   }
7.289
7.290     /**Configure the Systick interrupt time
7.291     */
7.292   HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
7.293
7.294     /**Configure the Systick
7.295     */
7.296   HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
7.297
7.298   /* SysTick_IRQn interrupt configuration */
7.299   HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
7.300 }
7.301
7.302 /* USER CODE BEGIN 4 */
7.303
7.304 /* USER CODE END 4 */
7.305
7.306 /**
7.307   * @brief  This function is executed in case of error occurrence.
7.308   * @param  file: The file name as string.
7.309   * @param  line: The line in file as a number.
7.310   * @retval None
7.311   */
7.312 void _Error_Handler(char *file, int line)
7.313 {
7.314   /* USER CODE BEGIN Error_Handler_Debug */
7.315   /* User can add his own implementation to report the HAL error return state */
7.316   while(1)
7.317   {
7.318   }
7.319   /* USER CODE END Error_Handler_Debug */
7.320 }
7.321
7.322 #ifdef  USE_FULL_ASSERT
7.323 /**
7.324   * @brief  Reports the name of the source file and the source line number
7.325   *         where the assert_param error has occurred.
7.326   * @param  file: pointer to the source file name
7.327   * @param  line: assert_param error line source number
7.328   * @retval None
7.329   */
7.330 void assert_failed(uint8_t* file, uint32_t line)
7.331 {
7.332   /* USER CODE BEGIN 6 */
7.333   /* User can add his own implementation to report the file name and line number,
7.334      tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
```

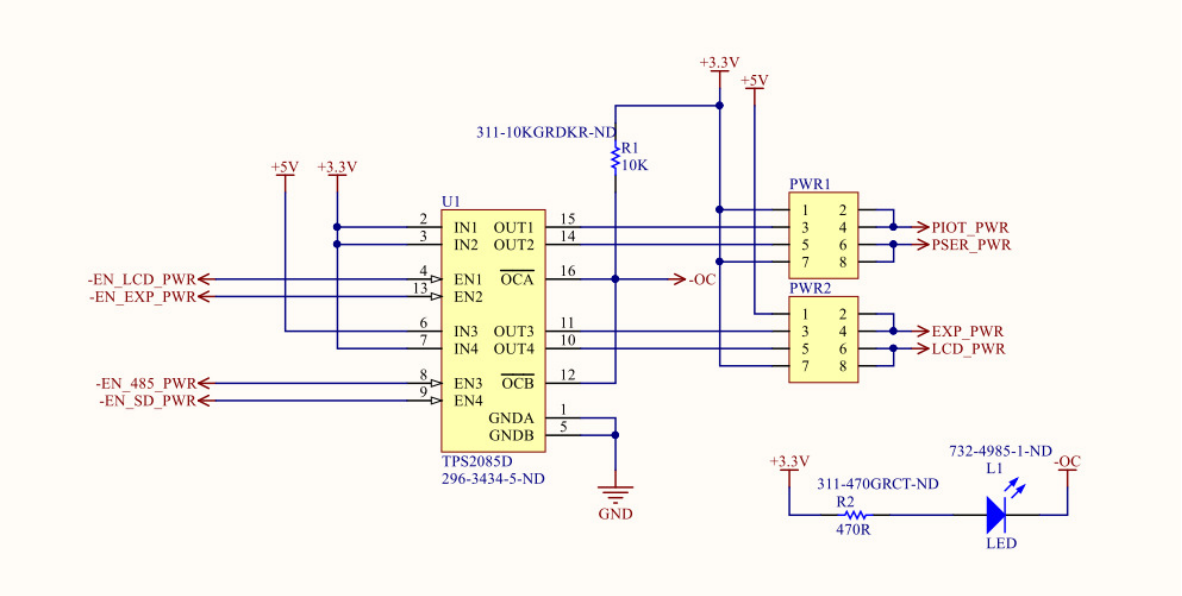
```
7.335   /* USER CODE END 6 */
7.336 }
7.337 #endif /* USE_FULL_ASSERT */
7.338
7.339 /**
7.340   * @}
7.341   */
7.342
7.343 /**
7.344   * @}
7.345   */
7.346
7.347 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

ANEXO

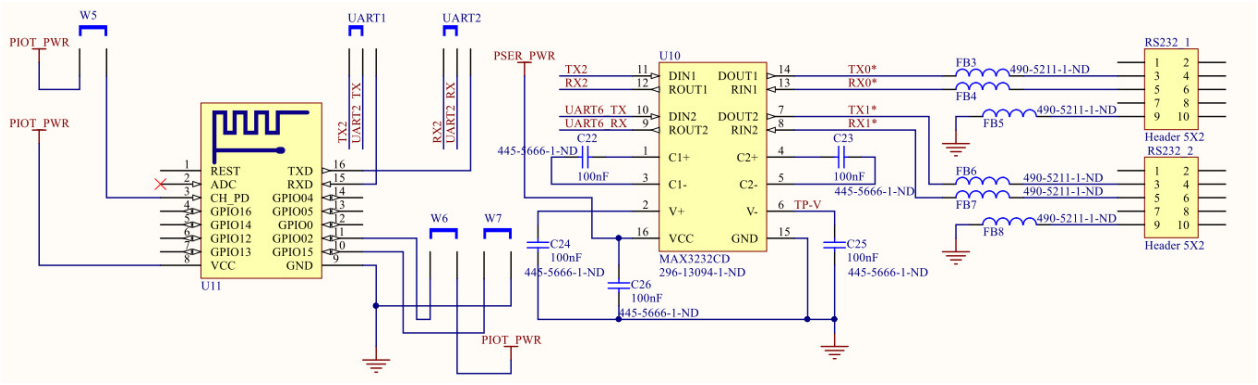
Figure 11. STM32F411xC/xE LQFP64 pinout



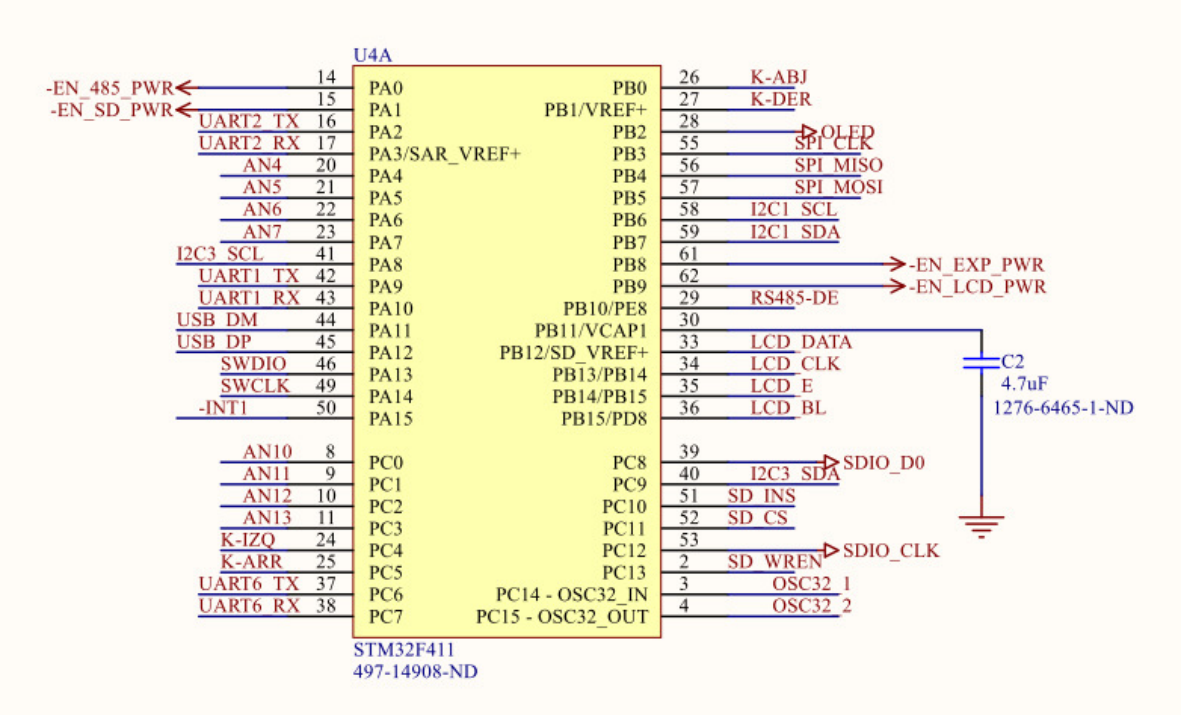
Alimentación Periféricos



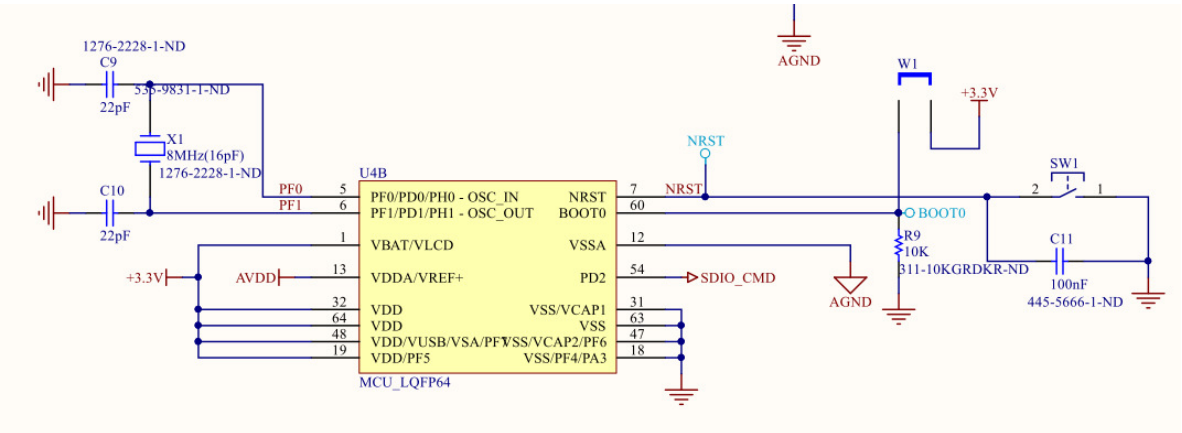
Circuito UARTs - Esquemático



Cpu-logic

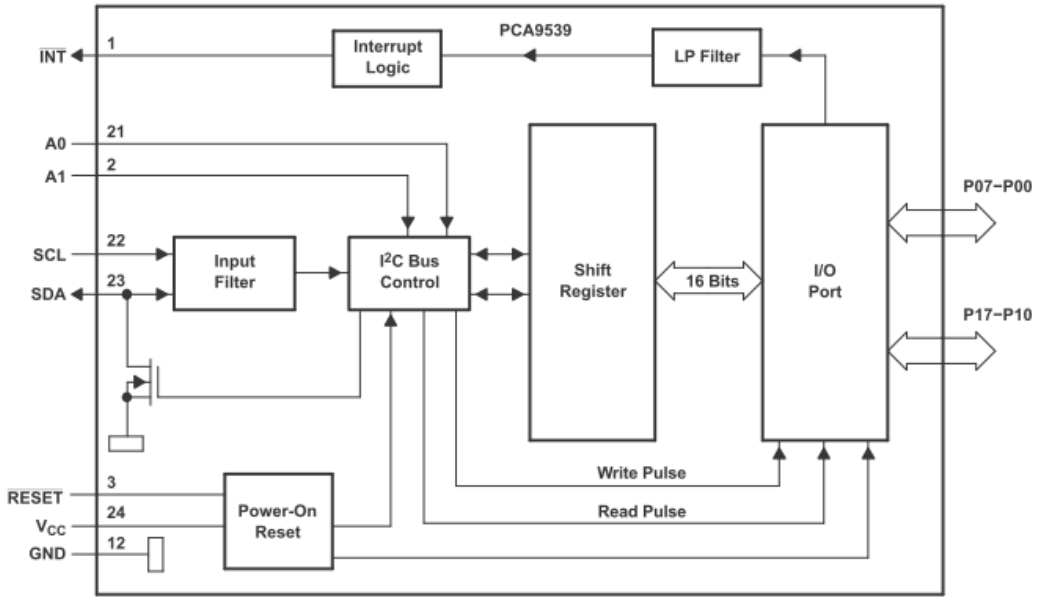


Cpu-Pwr+Clks



8 Detailed Description

8.1 Functional Block Diagram



A. Pin numbers shown are for DB, DBQ, DGV, DW, and PW packages.
B. All I/Os are set to inputs at reset.

Figure 17. Logic Diagram (Positive Logic)

8.3.2.1 Device Address

Figure 22 shows the address byte of the PCA9539.

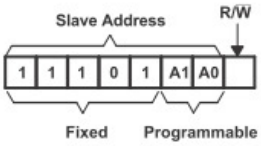


Figure 22. Pca9539 Address

Table 2. Address Reference

INPUTS		I²C BUS SLAVE ADDRESS
A1	A0	
L	L	116 (decimal), 74 (hexadecimal)
L	H	117 (decimal), 75 (hexadecimal)
H	L	118 (decimal), 76 (hexadecimal)
H	H	119 (decimal), 77 (hexadecimal)

The last bit of the slave address defines the operation (read or write) to be performed. A high (1) selects a read operation, while a low (0) selects a write operation.

8.3.2.2 Control Register And Command Byte

Following the successful acknowledgment of the address byte, the bus master sends a command byte that is stored in the control register in the PCA9539. Three bits of this data byte state the operation (read or write) and the internal register (input, output, Polarity Inversion or Configuration) that will be affected. This register can be written or read through the I^2C bus. The command byte is sent only during a write transmission.

Once a command byte has been sent, the register that was addressed continues to be accessed by reads until a new command byte has been sent.

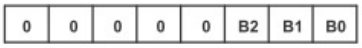
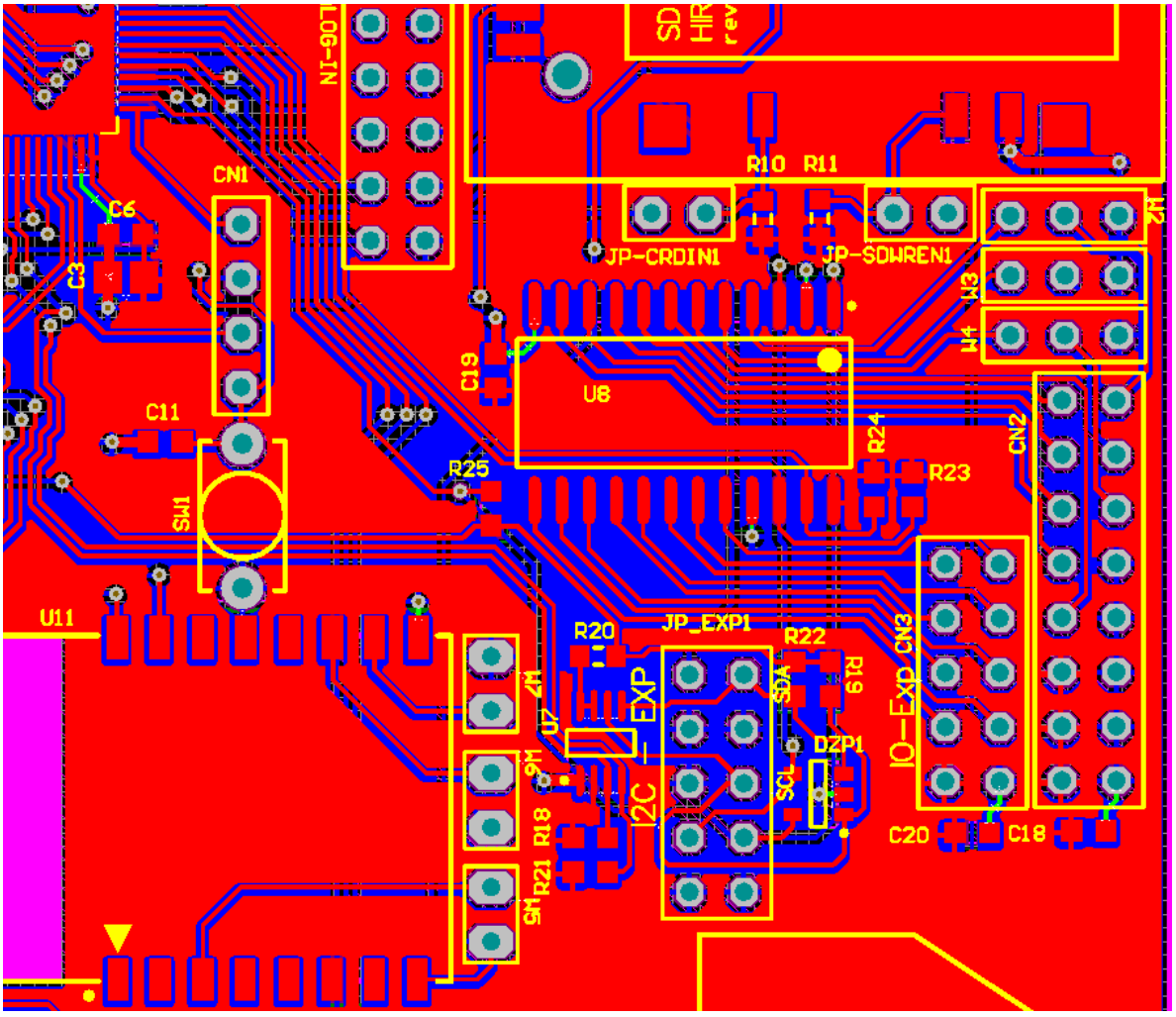


Figure 23. Control Register Bits

Table 3. Command Byte

CONTROL REGISTER BITS			COMMAND BYTE (HEX)	REGISTER	PROTOCOL	POWER-UP DEFAULT
B2	B1	B0				
0	0	0	0x00	Input Port 0	Read byte	xxxx xxxx
0	0	1	0x01	Input Port 1	Read byte	xxxx xxxx
0	1	0	0x02	Output Port 0	Read/write byte	1111 1111
0	1	1	0x03	Output Port 1	Read/write byte	1111 1111
1	0	0	0x04	Polarity Inversion Port 0	Read/write byte	0000 0000
1	0	1	0x05	Polarity Inversion Port 1	Read/write byte	0000 0000
1	1	0	0x06	Configuration Port 0	Read/write byte	1111 1111
1	1	1	0x07	Configuration Port 1	Read/write byte	1111 1111



PERIFERICOS EN PLACA CL3 (I)