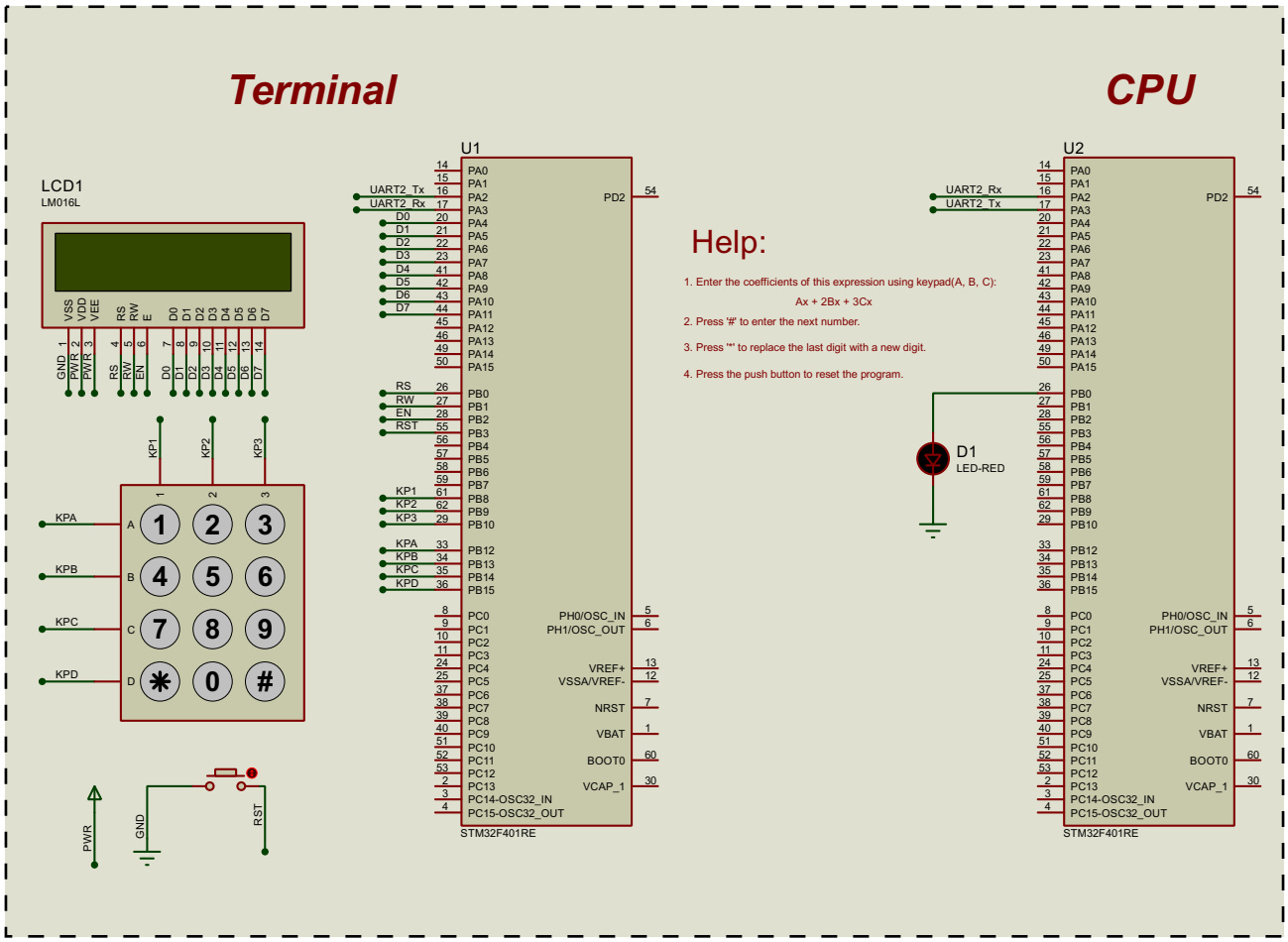
Mohammad Hashemi | 97243073
Instructor's Name: Dr. Dara Rahmati
May 30, 2021

HW07 Report

Microprocessor and Assembly Language Course - Spring 2021

Dual microprocessor serial communication using UART and STM32F4



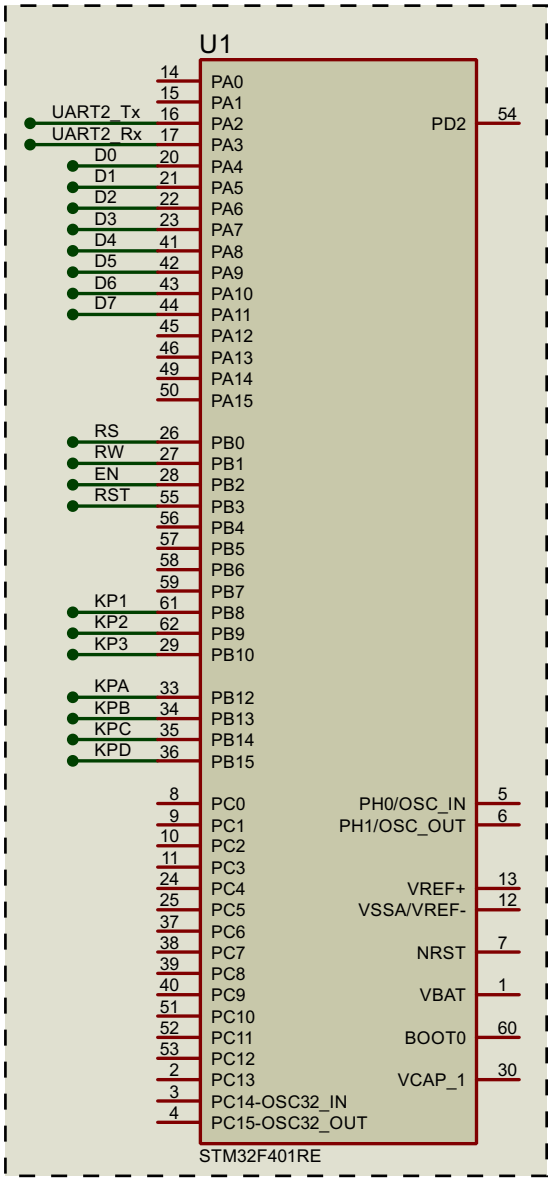
As we know from the project description, we have two distinct parts: **Terminal** and **CPU**. We will explain each of them in detail in the following:

1. Terminal

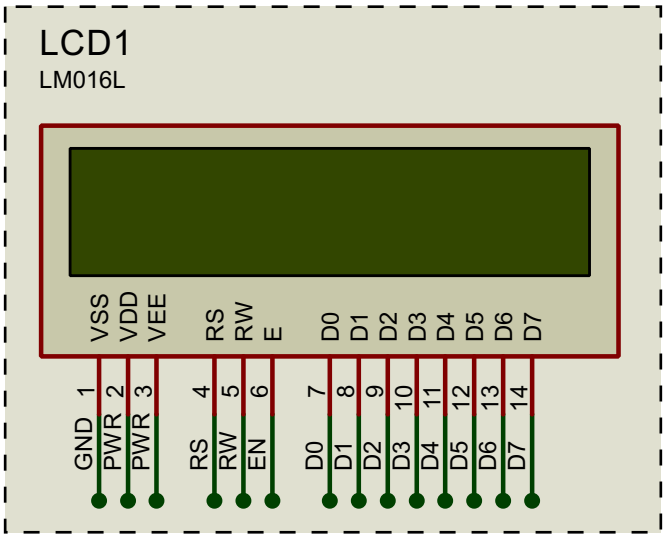
In terminal, we have a STM32F4 micro-controller, a LCD, a phone-keypad and a push button.

1.1. STM32F4

Ports PA4 to PA11 are connected to D0 to D7 of LCD ports(output). PB0 to PB2 are also connected to RS, RW and E of LCD respectively. PB3(RST) is wired to push button and PB8 to PB15 are connected to the keypad pins. PA2 and PA3 are the pins corresponding to USART2 in the micro controller. PA2 is for transmitting the data and PA3 is for receiving.



1.2. LCD

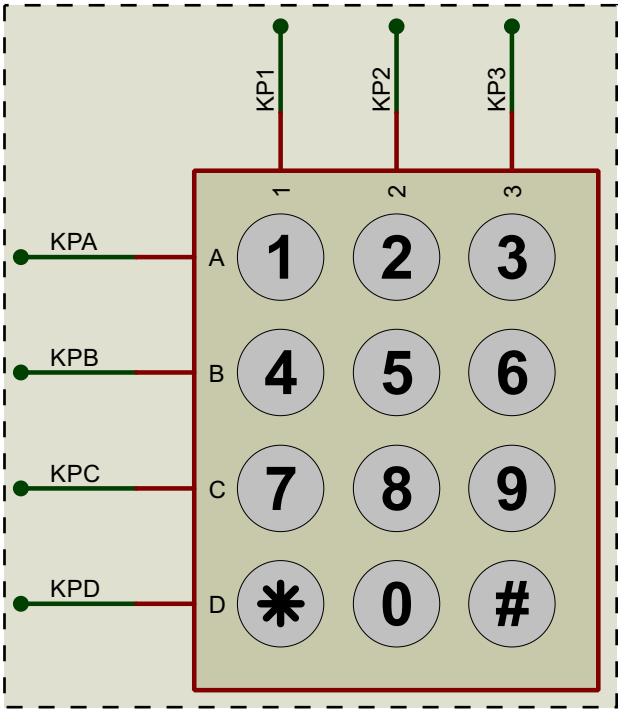


Pins D0-D7 are used for both data and commands. In the source code there are two functions *LCD_commands(char command)* and *LCD_data(char data):*

```
void LCD_command(unsigned char command) {
     GPIOB -> ODR &= ~(RS | RW);              /* RS = 0, R/W = 0 */
     GPIOA -> ODR = command << 4;             /* put command on data bus */
     GPIOB -> ODR |= EN;                      /* pulse EN high */
     delayMs(0);
     GPIOB -> ODR &= ~EN;
     if (command < 4)
          delayMs(4);          /* command 1 and 2 needs up to 1.64ms */
     else
          delayMs(1);          /* all others 40 us */
```

```
void LCD_data(char data) {
    GPIOB -> ODR |= RS;        /* RS = 1 */
    GPIOB -> ODR &= ~RW;       /* R/W = 0 */
    GPIOA -> ODR = data << 4; /* put data on data bus */
    GPIOB -> ODR |= EN;        /* pulse EN high */
    delayMs(0);                /* Do not change this line! */
     GPIOB -> ODR &= ~EN;
     delayMs(1);
}
```

1.3. phone-Keypad



The functionality of the keypad is so clear! The are four pins(rows) which act as output for the micro controller and we must enable them to read the columns(KP1, KP2 and KP3).

Here you can see the C source code of its function:

```
char keypad_getkey(void){
      int row, col;
      const int row_select[] = {0x1000, 0x2000, 0x4000, 0x8000};
      /*check to see any key pressed*/
      GPIOB -> ODR |= 0xF000;       /*enable all rows*/
      GPIOB -> ODR &= ~(0xF000);    /*load all rows with zero output*/
      delayUs(5);                   /*wait for signal return*/
      col = GPIOB ->IDR & 0x700;    /*read all columns*/
      GPIOB -> ODR |= 0xF000;       /*disable all rows*/
      if (col == 0x700) return 10;  /*no key pressed*/
      /*if a key is pressed, it gets here to find out which key.*/
      /*It activates one row at a time and read the input to see which
column is active*/
      for (row = 0; row < 4; row++){
            GPIOB -> ODR |= 0xF000; /*disable all rows*/
            GPIOB -> ODR |= row_select[row];    /*enable one row*/
            GPIOB -> ODR &= ~row_select[row];   /*drive the active row low*/
            delayUs(5);
            col = GPIOB -> IDR & 0x700;         /*read all columns*/
            if (col != 0x700) break;
            /*if one of the input is low, some key is pressed*/
            }
            GPIOB -> ODR |= 0xF000;             /*disable all Mows00*/
            delayMs(200);
            if (row == 4) return 10;
            if (row == 3 && col == 0x600) return '*';
            if (row == 3 && col == 0x500) return '0';
            if (row == 3 && col == 0x300) return '#';
            if (col == 0x300) return (row * 3 + 3) + '0';
            /*0000 0011 0000 0000 key in column 0*/
            if (col == 0x500) return (row * 3 + 2) + '0';
            /*0000 0101 0000 0000 key in column 1*/
            if (col == 0x600) return (row * 3 + 1) + '0';
            /*0000 0110 0000 0000 key in column 2*/
}
```

2. UART

For this system, the USART2 of both micro-controller are used(so the source code of their configuration and their functions implementation are the same).

For the usage of UARTs we must first initialize them. The detailed configuration can be found in the STM32F4 manual reference:

```
void USART2_init(void){
	/* 1. Enable the UART CLOCK and GPIO CLOCK */
	RCC -> AHB1ENR |= RCC_AHB1ENR_GPIOAEN;/* turn on the GPIOC clk */
	RCC -> APB1ENR |= (1<<17);          /* Enable USART2 clock */
	/* 2. Configure the UART PINs for ALternate Functions */
	GPIOA -> MODER |= (2<<4); /* Bits (5:4)= 1:0 --> Alternate Function
for Pin PA2 */
	GPIOA -> MODER |= (2<<6);  /* Bits (7:6)= 1:0 --> Alternate Function
for Pin PA3 */
	GPIOA -> OSPEEDR |= (3<<4) | (3<<6); /* Bits (5:4)= 1:1 and Bits
(7:6)= 1:1 --> High Speed for PIN PA2 and PA3 */
	GPIOA -> AFR[0] |= (7<<8);    /* Bytes (11:10:9:8) = 0:1:1:1  --> AF7
Alternate function for USART2 at Pin PA2 */
	GPIOA -> AFR[0] |= (7<<12);   /* Bytes (15:14:13:12) = 0:1:1:1  -->
AF7 Alternate function for USART2 at Pin PA3 *
	/* 3. Enable the USART by writing the UE bit in USART_CR1 register to
1. */
	USART2 -> CR1 = 0x00;         /* clear all */
	USART2 -> CR1 |= (1<<13);     /* UE = 1... Enable USART */
	/* 4. Program the M bit in USART_CR1 to define the word length. */
	USART2 -> CR1 &= ~ (1<<12);   /* M =0; 8 bit word length */
	/* 5. Select the desired baud rate using the USART_BRR register. */
	USART2 -> BRR = 0x0683;       /* Baud rate of 9600, @ 16MHz */
	/* 6. Enable the Transmitter/Receiver by Setting the TE and RE bits in
USART_CR1 Register */
	USART2 -> CR1 |= (1<<2);      /* RE=1.. Enable the Receiver */
	USART2 -> CR1 |= (1<<3);      /* TE=1.. Enable Transmitter */
}
```

And two function for writing and reading intro UARTs:

```
uint8_t UART2_recieve(void) {
	uint8_t c;
	while (!(USART2 -> SR & (1<<5)));  /* wait for RXNE bit to set */
	c = USART2 -> DR;  /* Read the data. This clears the RXNE also */
	return c;
}

uint8_t UART2_recieve_number(void) {
	uint8_t c;
	c = UART2_recieve();
	return c;
}
```