UART DMA İçin FIFO Fonksiyonları

Tüm fonksiyonları *"fifo.c"* ve *"fifo.h"* dosyaları içermektedir.

```c
#ifndef FIFO_H_
#define FIFO_H_

#include "main.h"

#define BUF_SIZE 10 //FIFO Number
#define DATA_SIZE 32 //Data Size

//FIFO Struct
typedef struct {
	uint8_t buf[DATA_SIZE*BUF_SIZE]; //FIFO Buffer
	unsigned short head; //FIFO Write Index
	unsigned short tail; //FIFO Read Index
	unsigned short size; //FIFO Size (Byte)
	unsigned short FIFOindex; //FIFO Buffer's Write Index
	unsigned short UARTindex; //FIFO Buffer's Read  Index
}FIFO;

void fifo_init();
void fifo_init_control();
unsigned short fifo_read(uint8_t * buf);
unsigned short fifo_write(uint8_t * buf);
unsigned short UART_fifo_write(UART_HandleTypeDef *huart);

#endif //FIFO_H_

```

Figure 1 - fifo.h Dosyasının Kesiti

Figür 1'de tüm fonksiyonlar gösterilmektedir.

1. FIFO Struct Değişkenleri

"BUF_SIZE" tanımlaması FIFO buffer'ının kaç adet *"DATA_SIZE"* boyutunda veri saklayacağını belirtmektedir. *"DATA_SIZE"* tanımlaması ise saklanacak verilerin boyutunu belirtmektedir.

FIFO değişkenleri bir struct yapısı altında toplanmıştır. *"buf"* değişkeni direct olarak FIFO buffer'ının kendisidir. Uint8_t tipinde tanımlanmasının sebebi gönderilecek verilerin bu tipte olacağının belirlenmesi üzerinedir.

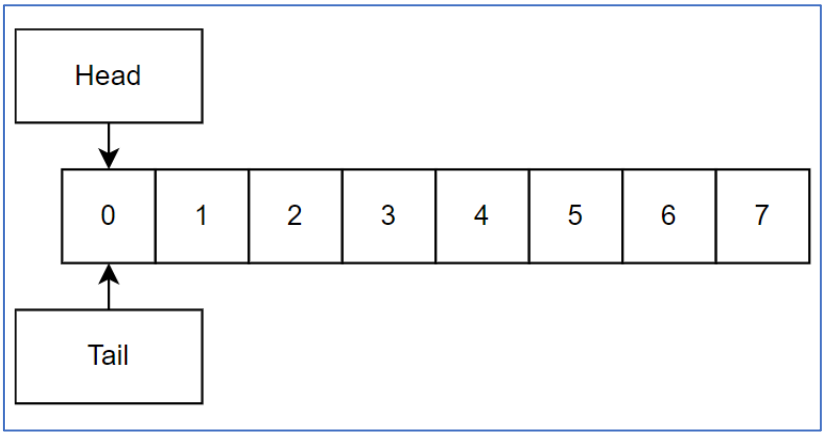

Figure 2 - FIFO Diyagramı

Figür 2'de FIFO yapısı gösterilmiştir. *"head"* değişkeni FIFO'ya veri yazılırken buffer'ın hangi hücresinde olunduğunu belirtmektedir. *"tail"* değişkeni ise FIFO'dan veri okunurken hangi hücre üzerinde olduğunu belirtmektedir. *"size"* değişkeni FIFO'nun kaç byte olduğunu belirtmektedir. *"FIFOindex"* değişkeni FIFO'ya veri yazarken sıranın kaçıncı 32 byte'lık datada olduğunu gösterirken, *"UARTindex"* değişkeni FIFO'dan UART'a veri yazarken sıranın yazılacak olan kaçıncı 32 byte'lık datada olduğunu göstermektedir. 10 adet data figür 2'de gösterildiği biçimde arka arkaya sıralanacak şekilde FIFO'ya doldurulmaktadır.

2. FIFO Fonksiyonları

a. fifo_init

"fifo_init" fonksiyonu fonksiyonların içerisinde eğer kod ilk defa çalıştırılıyorsa FIFO'nun değişkenlerine atama yaparak FIFO'yu init etmektedir. Main kod içerisinde çağrılmasına gerek yoktur.

b. fifo_init_control

"fifo_init_control" fonksiyonu diğer fonksiyonlar içerisinde FIFO'nun daha önce init edilip edilmediğini kontrol eden fonksiyondur. Main kod içerisinde çağırılmasına gerek yoktur.

c. fifo_read

"fifo_read" fonksiyonu FIFO'daki veriyi okuyan fonksiyondur. Her çağrıldığında FIFO'dan *"DATA_SIZE"* kadar veri okur. UART üzerinden sadece yazma işlemi yapılacaksa main kod içerisinde çağrılmasına gerek yoktur.

d. fifo_write

"fifo_write" fonksiyonu FIFO'ya veri yazan fonksiyondur. Her çağrıldığında, verilen veriyi *"DATA_SIZE"* kadar FIFO'ya yazar. Eğer verilen veri *"DATA_SIZE"*dan daha küçük bir veri ise kalan byte'ları 0 ile doldurur. Eğer daha büyük bir veri ise sadece *"DATA_SIZE"* kadar yazma yapar. Main kod içerisinde FIFO'ya veri yazılması gereken yerlerde çağırılabilir.

e. UART_fifo_write

"UART_fifo_write" fonksiyonu FIFO'daki verileri DMA ile UART üzerinden yazan fonksiyonudur. Main kod içerisinde FIFO'dan okunacak sıradaki verinin DMA ile UART üzerinden yazılması gereken yerlerde çağrılabilir.

Örnek proje içerisinde bu fonksiyon timer ile birlikte kullanılmıştır. Bu yüzden main kod içerisinde çağrılmasına gerek kalmamıştır.

3. FIFO Fonksiyonları – Detaylı Anlatım

a. Değişkenler

"DMA_WRITE_BUF" buffer'ı FIFO'daki veriyi DMA ile UART'a basarken, FIFO'dan çekilen verinin DMA ile UART'a basılmadan önce aktarıldığı ara bir bufferdır. Bu buffer DMA fonksiyonuna verilerek aktarım sağlanır.

```
uint8_t DMA_WRITE_BUF[DATA_SIZE]; //Buffer is filling by FIFO to write data to DMA
unsigned short fifoInit = 0; //Checking is fifo initialized
unsigned short STATUS_DMA;

//Extern this object
//Extern FIFO fifo;
//Use this object in fifo functions
FIFO fifo;
```

Figure 3 - Değişkenler

"fifoInit" değişkeni FIFO'nun init edilip edilmediğini belirten değişkendir. FIFO init edilmediği durumda *"fifo_init_control"* fonksiyonu *"fifo_init"* fonksiyonunu çağırır ve *"fifoInit"* değişkenini *"1"* yapar.

"STATUS_DMA" değişkeni veri DMA ile UART'a yazıldıktan sonra DMA fonksiyonun return ettiği değeri saklamaktadır. Eğer return edilen değer *"0x02U"* ise DMA meşgul anlamına gelmektedir ve veri atlaması olmaması için FIFO'dan yeni veri okuması yapılmaz.

"FIFO fifo" tanımlanan struct yapısından bir obje oluşturmak için kullanılmıştır. Bütün FIFO değişkeni işlemleri bu obje üzerinden yapılmaktadır.

b. fifo_init

```
//FIFO variables initialized
//User does not need to use this function
void fifo_init(){
	fifo.head = 0;
	fifo.tail = 0;
	fifo.size = DATA_SIZE;
	fifo.buf[DATA_SIZE] = 0;
	fifo.FIFOindex = 0;
	fifo.UARTindex = 0;
}
```

Figure 4 - fifo_init Fonksiyonu

FIFO'ya ait değişkenlere initial atama yapılan fonksiyondur.

c. fifo_init_control

```
//Checking is FIFO initialized at fifo_read, fifo_write and UART_fifo_write functions
//User does not need to use this function
void fifo_init_control(){
    if(fifoInit == 0){
        fifo_init();
        fifoInit = 1;
    }
}
```

Figure 5 - fifo_init_control Fonksiyonu

"fifo_init_control" fonksiyonu, diğer fonksiyonlar içinde FIFO'nun init edilip edilmediğini kontrol eden fonksiyondur.

d. fifo_read

```
//Read FIFO data and write to another buffer
//User does not need to use this function if just want to transmit data
unsigned short fifo_read(uint8_t * buf){
    unsigned short i;
    unsigned short bufIndex = 0;
    fifo_init_control();
    for(i = fifo.UARTindex*DATA_SIZE; i < (fifo.UARTindex+1)*DATA_SIZE; i++){
        if( fifo.tail != fifo.head ){
            buf[bufIndex] = fifo.buf[fifo.tail];
            fifo.tail++;
            bufIndex++;
            if( fifo.tail == DATA_SIZE*BUF_SIZE )
                fifo.tail = 0;
        }
        else
            return i;
     }

    if(fifo.UARTindex + 1 == BUF_SIZE)
        fifo.UARTindex = 0;
//  else
//      fifo.UARTindex++;
    return DATA_SIZE;
}
```

Figure 6 - fifo_read Fonksiyonu

"i" değişkeni for içerisinde kullanılmak için oluşturulmuş dummy bir değişkendir. *"bufIndex"* değişkeni, *"i"* değişkeninden bağımsız olarak buffer'ın mevcut hücresini tanımlamak için kullanılmıştır. FIFO'dan okuma veya yazma yaparken 32 byte'lık veriler halinde yapılabilmesi adına *"i"* değişkeni *"UARTindex"* ve *"DATA_SIZE"*'ın çarpımına eşitlenmiştir. Bir sonraki veri paketine kadar işlem yapılması adına *"UARTindex+1"* ile *"DATA_SIZE"*'ın çarpımına kadar döngü devam etmektedir.

"tail != head" ifadesi FIFO'ya veri yazıldığı anlamına gelmektedir. Eğer eşitlik var ise FIFO'da okunacak veri kalmamıştır demektir. Bu yüzden *"tail != head"* olduğu durumda FIFO'dan okuma işlemi yapılmıştır. FIFO'nun boyutu *"BUF_SIZE"* değişkeni ile sınırlandırılmıştır. Buffer dolduktan sonra yeni veriler tekrar başa yazılacaktır. Bu yüzden UARTindex son veri'ye eşit olduğunda sıfırlanmıştır.

e. fifo_write

"i" ve *"bufIndex"* değişkenleri için *"fifo_read"* fonksiyonunda yazılanların aynısı geçerlidir. Aynı şekilde for döngüsü parametreleri için de *"fifo_read"* fonksiyonunda yazılanlar geçerlidir. *"fifo_read"* fonksiyonundan farklı olarak yapılan ilerlemeler *"head"* değişkeni üzerinden yapılmaktadır. Bu sayede okuma yapılacağı zaman *"head"* ve *"tail"* değişkenlerinin eşit olup olmama durumu üzerinden ilerlenebilir.

```
//Write bytes to FIFO from given buffer
unsigned short fifo_write(uint8_t * buf){
    unsigned short i;
    unsigned short bufIndex = 0;
    fifo_init_control();
    for(i = fifo.FIFOindex*DATA_SIZE; i < (fifo.FIFOindex+1)*DATA_SIZE; i++){
        if( (fifo.head + 1 == fifo.tail) || ( (fifo.head + 1 == (fifo.size)*fifo.FIFOindex) && (fifo.tail == 0) )
            return i;
        else {
            fifo.buf[fifo.head] = buf[bufIndex];
            fifo.head++;
            bufIndex++;
            if( fifo.head == DATA_SIZE*BUF_SIZE )
                fifo.head = 0;
        }
    }
    if(fifo.FIFOindex + 1 == BUF_SIZE)
        fifo.FIFOindex = 0;
    else
        fifo.FIFOindex++;
    return DATA_SIZE;
}
```

Figure 7 - fifo_write Fonksiyonu

f. UART_fifo_write

```
//Write FIFO data to UART with DMA
unsigned short UART_fifo_write(UART_HandleTypeDef *huart){
    unsigned short STATUS_DMA = 0x00U;
    fifo_init_control();
    if((fifo.UARTindex != 0 || fifo.UARTindex != fifo.FIFOindex) && (fifo.head != fifo.tail)){
        if(STATUS_DMA == 0x02U){
            STATUS_DMA = HAL_UART_Transmit_DMA(huart,DMA_WRITE_BUF,DATA_SIZE);
        }
        else if(STATUS_DMA == 0x00U){
            fifo_read(DMA_WRITE_BUF);
            STATUS_DMA = HAL_UART_Transmit_DMA(huart,DMA_WRITE_BUF,DATA_SIZE);
        }
    }
    return STATUS_DMA;
}
```

Figure 8 – UART_fifo_write Fonksiyonu

"STATUS_DMA" değişkeni *"HAL_UART_Transmit_DMA"* fonksiyonun geri döndürdüğü değeri kaydetmektedir. Eğer DMA veri göndermek için müsaitse *"HAL_OK"* veya değilse *"HAL_BUSY"* gibi değerleri geri döndürmektedir. *"HAL_OK"* için 0x00U ve *"HAL_BUSY"* için 0x02U değerleri karşılık gelmektedir.

İlk if bloğunun içinde çift taraflı controller yapılmaktadır. Eğer *"head"* ve *"tail"* eşit değilse sırada UART'a basılacak veri vardır, veya *"UARTindex"* ile *"FIFOindex"* eşit değilse ve *"UARTindex"* 0 değilse gönderilmemiş veri paketi vardır gibi. Bu bloğun içerisinde *"STATUS_DMA"* kontrolü yapılmaktadır. Eğer ilk defa veri gönderilecekse DMA meşgul değildir. Bu yüzden initial değer olarak *"STATUS_DMA"* değişkenine 0x00U değeri atanmıştır. Yani ilk kez veri UART'a basılacağı zaman kod else if bloğuna girer ve FIFO'dan veri okuyarak ara buffer olan *"DMA_WRITE_BUF"* bufferına verileri alır. Sonrasında *"HAL_UART_Transmit_DMA"* fonksiyonunu kullanarak DMA ile verileri UART'a basar. Return olarak DMA'in durumunu da alarak *"STATUS_DMA"* değişkenine atar. Eğer DMA meşgulse *"HAL_UART_Transmit_DMA"* fonksiyonu geri 0x02U değerini döndürecektir. Bu durumda kod ilk if

bloğuna girer. Veri kaybı olmaması adına FIFO'dan yeni veri çekilmez ve DMA'in mevcut durumunu öğrenmek adına *"HAL_UART_Transmit_DMA"* fonksiyonu tekrar çağrılır. Eğer bu süre zarfında *"HAL_UART_Transmit_DMA"* fonksiyonu müsait duruma geçmişse veri DMA ile basılır. Değilse DMA müsait olana kadar bu döngü tekrarlanır.

g. Timer

Kod timer ile birlikte çalışmaktadır. Örnek kod içerisinde TIM3 kullanılmıştır. Timer fonksiyonu içerisinde *"UART_fifo_write"* fonksiyonu çağırılmaktadır. Örnek kod içerisinde UART2 kullanılmıştır. Bu yüzden *"UART_fifo_write"* fonksiyonuna parameter olarak *"&huart2"* verilmiştir.

```
void TIM3_IRQHandler(void)
{
  /* USER CODE BEGIN TIM3_IRQn 0 */

  /* USER CODE END TIM3_IRQn 0 */
  HAL_TIM_IRQHandler(&htim3);
  /* USER CODE BEGIN TIM3_IRQn 1 */

  HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin);

  UART_fifo_write(&huart2);

  /* USER CODE END TIM3_IRQn 1 */
}
```

Figure 9 - Timer Fonksiyonu

Kodun timer'a girdiğinden görsel olarak da emin olunması adına fonksiyon içerisinde LED3 yakılmıştır.

h. DMA RX Complete Callback

DMA ile UART üzerinden veri aktarımı tamamlandığı zaman kod *"HAL_UART_TxCpltCallback"* fonksiyonuna girer. Veri aktarımının tamamlandığından emin olunduktan sonra *"UARTindex"* değişkeni artırılmaktadır. Bu yüzden bu işlem *"HAL_UART_TxCpltCallback"* fonksiyonu içinde yapılmıştır.

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
    HAL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
    fifo.UARTindex++;
}
```

Figure 10 - HAL_UART_TxCpltCallback Fonksiyonu

"HAL_UART_TxCpltCallback" fonksiyonuna kodun girdiğinden görsel olarak emin olunması adına LED1 yakılmıştır.

i. Main Kod

Örnek projede 9 adet dummy veri hazırlanmıştır. Her bir veri 32byte'dır. Ayrıca *"fifo.c"* dosyasında oluşturulan *"FIFO"* tipindeki *"fifo"* objesinin de main kod içerisinde extern edilmesi gerekmektedir (*"HAL_UART_TxCpltCallback"* fonksiyonunda kullanıldığı için).

```
/* USER CODE BEGIN PV */

uint8_t data1[DATA_SIZE] = {0x1,0x1,0x1,0x1,0x1,0x1,
uint8_t data2[DATA_SIZE] = {0x2,0x2,0x2,0x2,0x2,0x2,
uint8_t data3[DATA_SIZE] = {0x3,0x3,0x3,0x3,0x3,0x3,
uint8_t data4[DATA_SIZE] = {0x4,0x4,0x4,0x4,0x4,0x4,
uint8_t data5[DATA_SIZE] = {0x5,0x5,0x5,0x5,0x5,0x5,
uint8_t data6[DATA_SIZE] = {0x6,0x6,0x6,0x6,0x6,0x6,
uint8_t data7[DATA_SIZE] = {0x7,0x7,0x7,0x7,0x7,0x7,
uint8_t data8[DATA_SIZE] = {0x8,0x8,0x8,0x8,0x8,0x8,
uint8_t data9[DATA_SIZE] = {0x9,0x9,0x9,0x9,0x9,0x9,

extern FIFO fifo;

/* USER CODE END PV */
```

Figure 11 - Main Koddaki Veriler ve Extern Değişkenler

Figür 10'daki görselde verilerin tamamı gösterilmemiştir. Örneğin *"data1"* bufferında 32 adet 0x1 verisi bulunmaktadır fakat görselin fazla yer kaplamaması adına sadece birkaç tanesi gösterilmiştir.

```
/* USER CODE BEGIN WHILE */

while (1)
{
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

    fifo_write(data1);
    HAL_Delay(2);
    fifo_write(data2);
    HAL_Delay(2);
    fifo_write(data3);
    HAL_Delay(2);
    fifo_write(data4);
    HAL_Delay(2);
    fifo_write(data5);
    HAL_Delay(2);
    fifo_write(data6);
    HAL_Delay(2);
    fifo_write(data7);
    HAL_Delay(2);
    fifo_write(data8);
    HAL_Delay(2);
    fifo_write(data9);
    HAL_Delay(2000);

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

Figure 12 - Main Kod İçerisindeki While Döngüsü

Main kod içerisindeki while içinde veriler 2 milisaniye aralıklarla FIFO'ya yazılmıştır. Kodu test etmek amaçlı böyle bir şey yapılmıştır. Kodun while içerisinde döndüğünün görsel olarak da anlaşılabilmesi adına LED2 yakılmıştır.

Kod main içindeki while'da 2 milisaniye aralıklarla verileri sürekli olarak FIFO'ya yazarken, yaklaşık 1 milisaniye aralıklara timer çalışmaktadır. Her 1 milisaniyede bir timer fonksiyonuna girerek sırada DMA üzerinden UART'a yazılacak verinin olup olmadığı kontrol edilmektedir. Timer süresi 10 milisaniyede de denenmiştir ve kod yine düzgün bir şekilde çalışmıştır.

4. İnceleme

Kodun düzgün çalışıp çalışmadığı hem RealTerm ile hem de logic analyzer ile test edilmiştir.

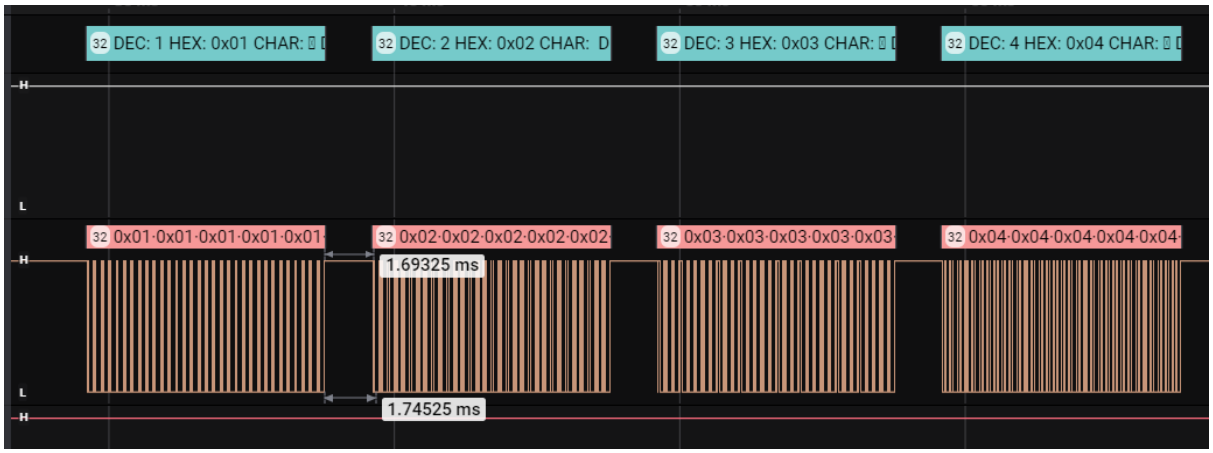

Figure 13 - Logic Analyzer Sonucu

Figür 13'de görüldüğü gibi veriler kodda gönderildiği sıra ile UART'a basılmıştır. Timer süresi yaklaşık 1 ms ayarlanmıştır. Figür 13'de de veri aktarımı arasındaki mesafenin 1ms'ye yakın olduğu görülebilir.

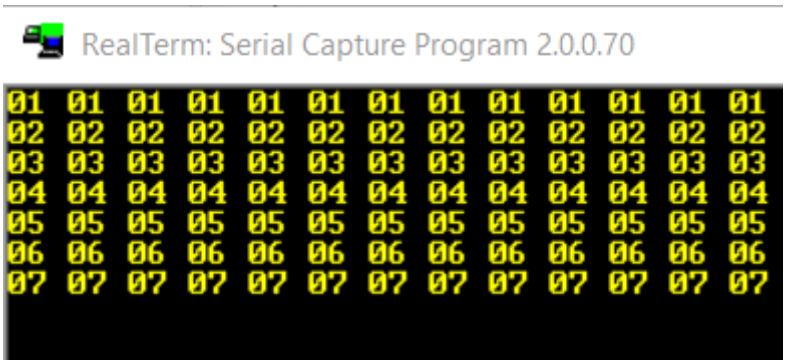

Figure 14 - RealTerm Sonucu

Figür 14'de de görüleceği üzere veriler gönderildiği şekilde UART'a basılmıştır.