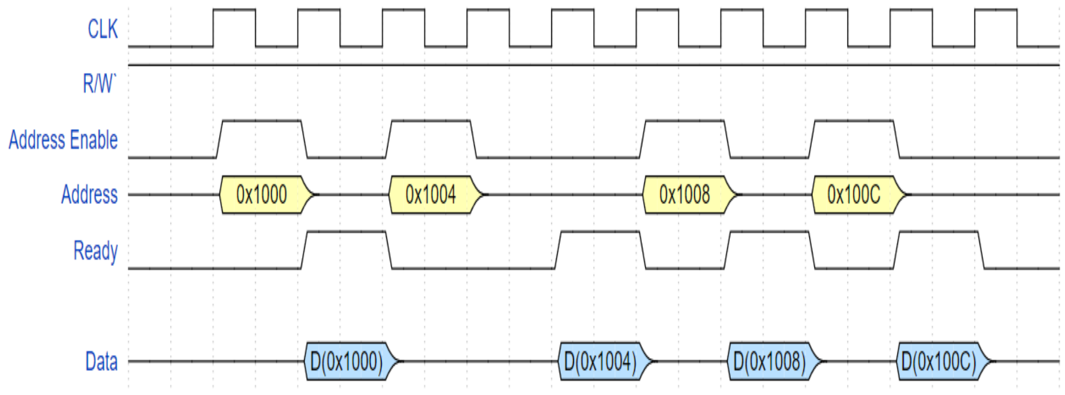
COE 306, Term 171

Introduction to Embedded Systems

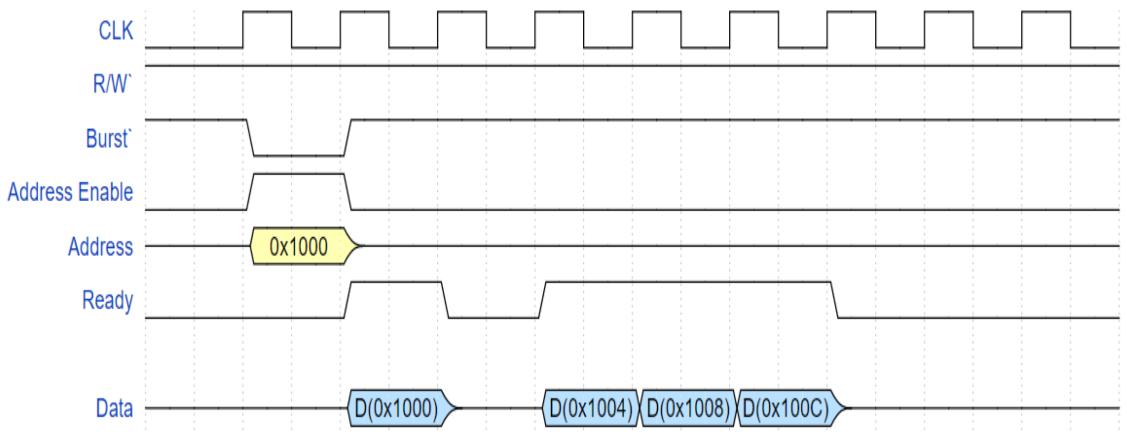
Assignment# 3 Solution
Due date: Saturday, Nov. 18, 2017

Q.1. A memory device is read by putting a read request on the bus for a single bus cycle using a **R/W'** signal. The memory device then puts the read data on the bus on the next clock cycle. The memory asserts a **Ready** signal to indicate that the data is ready. We would like to read four consecutive 32-bit words at memory addresses 0x1000, 0x1004, 0x1008, and 0x100C. Assume that the data bus is 32-bit wide.

(a) Use a timing diagram to show the bus signals required to read the four locations using non-burst requests. Show the bus clock, the address, the read data, and any other required signals. Assume that while reading the address 0x1004 the memory will not be ready for one clock cycle.



(b) Assuming a fixed burst size of 4 using incrementing address, use a timing diagram to show the required bus signals to read the two locations using burst requests. Assume that an active-low burst signal must be asserted for a single cycle only for each burst read request. Assume that the starting burst address is put only at the request of the burst and other addresses do not need to be sent during a burst transfer. Assume that each consecutive data in a burst will be ready in the consecutive cycle after the previous data as long as memory is ready. Assume that while reading the address 0x1004 the memory will not be ready for one clock cycle.



Q.2. We would like to investigate the performance of a bus system in relation to a digital audio application. Digital audio is specified by three main parameters:

1. Number of channels, e.g. stereo audio uses two channels.
2. Sampling rate: number of digital samples per second.
3. Sample size (or bit depth): number of bits per sample.

Assume that a system bus that runs at 1 MHz, and requires a total of 4 cycles to complete a single 16-bit transfer. Assuming uncompressed 6-channel audio (5.1 speaker configuration), what is the best combination of sampling rate and sample size that can be handled by this system bus? Justify your choice.
Typical sampling rates: 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz (Audio CD), 48 kHz (DVD).
Typical sample sizes: 8-bit, 16-bit (Audio CD), 20-bit, 24-bit, 32-bit.

First, we find the number of bits that the described system bus can transfer per second, N_{bus}:

$$T = (D + O)\frac{N}{W} \; => \; Nbus = \frac{WT}{(D + O)} = \frac{1 \times 10^6 \times 16}{4} = 4 \times 10^6 \; bits$$

Considering Audio CD quality (sampling rate: 44.1 kHz, sample size: 16 bits), the required bandwidth is:
N_{audio} = 6 x 44,100 x 16 = 4,233,600 bit/second > 4 x 10^6

We can either reduce the sampling rate, or the sample size, to lower the required bandwidth to match the system bus capacity:

N_{audio1} = 6 x 32,000 x 16 = 3, 072, 000 bit/second < 4 x 10^6

N_{audio2} = 6 x 44,100 x 8 = 2, 116, 800 bit/second < 4 x 10^6

Since Naudio2 halves the sample size, it is expected to have a greater impact on audio quality. Hence, reducing the sampling rate is preferred, resulting in a sampling rate of 32 kHz and a sample size of 16 bits.

Furthermore, we can achieve better audio quality by increasing the sample size of Naudio1 from 16 to 20 bits:
$N_{audio3} = 6 \times 32{,}000 \times 20 = 3{,}840{,}000$ bit/second $< 4 \times 10^6$

Therefore, the best combination that the system can handle is 32,000 sampling rate and 20 bit samples.

Q.3. A real-time system receives data through an I/O device, the CPU processes the data, then the results of the processing are transferred to system memory. The I/O device, the CPU, and the memory controller are all on the same system bus, which runs at 1MHz. The CPU runs at 5 MHz. Each bus transaction (transfer) between any two devices on the bus takes 4 bus cycles, 1 of which is used to transfer data, and the remaining cycles are used by the bus protocol. The bus has 32 data lines, transferring 32 bits per data-transfer cycle.

The I/O device receives 1024 bytes at a time. While processing the received data, for each received byte, the CPU generates 4 bytes. Only generated data is transferred from the CPU to system memory.

If the I/O device receives new data at a rate of 100 times per second (1024 bytes each), how many CPU cycles can be spent processing each byte without violating the real-time requirements?

Assume that the memory is fast enough to handle any requests received by the memory controller.

$N_{I/O} = 1024$ B x $100 = 102400$ B

$T_{I/O}(N) = (D + O)\ N/W = 4 \times 102400/4 = 102400$ cycles

$N_{mem} = 1024$ B x $100 \times 4 = 409600$ B

$T_{mem}(N) = 4 \times 409600/4 = 409600$ cycles

$T_{bus} = 102400 + 409600 = 512000$ cycles

$t_{bus} = T_{bus}\ P = 512000 \times 10^{-6} = 0.512$ s

$t_{CPU} = 1 - 0.512 = 0.488$ s

$T_{CPU} = t_{CPU} \times f_{CPU} = 0.488 \times 5 \times 10^6 = 2440000$ cycles

Number of CPU cycles can be spent processing each byte without violating the real-time requirements = 2440000 / 102400 = 23.83 => 23 cycles per Byte.

Q.4. Use PWM with 3 channels connected to an RGB-LED to display the basic colors: Red, Green, Blue, Yellow, Cyan, Magenta and White with intensity varying form low to high. Consult the datasheet of the RGB-LED to identify the needed resistor values to be connected to the RGB pins to get proper colors displayed. Include the code of your solution along with a video link demonstrating desired functionality.

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>

int main(void) {

    // TODO: insert code here

            LPC_GPIO0->FIODIR = (1<<11);
            LPC_GPIO0->FIOPIN = (1<<11);

            LPC_PINCON->PINSEL4 |= 1<<0 | 1<<2 | 1<<4; //Configure Pin 2.0 as PWM 1.1,
Configure Pin 2.1 as PWM 1.2, Configure Pin 2.2 as PWM 1.2
            LPC_PWM1->MR0 = 255;
            LPC_PWM1->LER |= 1;


            LPC_PWM1->PCR |= (1 << 9) | (1<<10) | (1<<11); // enable the outputs of PWM1.1,
PWM1.2, PWM1.3

            LPC_PWM1->TCR |= (1 << 0) | (1 << 3); // Enable Counter & PWM

    volatile static int i, l;

    while(1) {

            LPC_PWM1->MR1 = 255;
            LPC_PWM1->LER |= (1 << 1);
            LPC_PWM1->MR2 = 255;
            LPC_PWM1->LER |= (1 << 2);
            LPC_PWM1->MR3 = 255;
            LPC_PWM1->LER |= (1 << 3);
            for (l = 0; l < 1000000; l++);

            printf("printing red \n");

            for (i=255; i>=25; i=i-25){
                        LPC_PWM1->MR1 = i;
```

```
		LPC_PWM1->LER |= (1 << 1);
		for (l = 0; l < 1000000; l++);
	}

	LPC_PWM1->MR1 = 255;
	LPC_PWM1->LER |= (1 << 1);
	LPC_PWM1->MR3 = 255;
	LPC_PWM1->LER |= (1 << 3);

	printf("printing green \n");

	for (i=255; i>=25; i=i-25){
		LPC_PWM1->MR2 = i;
		LPC_PWM1->LER |= (1 << 2);
		for (l = 0; l < 1000000; l++);
	}

	LPC_PWM1->MR1 = 255;
	LPC_PWM1->LER |= (1 << 1);
	LPC_PWM1->MR2 = 255;
	LPC_PWM1->LER |= (1 << 2);

	printf("printing blue \n");

	for (i=255; i>=25; i=i-25){
		LPC_PWM1->MR3 = i;
		LPC_PWM1->LER |= (1 << 3);
		for (l = 0; l < 1000000; l++);
	}

	printf("printing yellow \n");

	LPC_PWM1->MR3 = 255;
	LPC_PWM1->LER |= (1 << 3);

	for (i=255; i>=25; i=i-25){
		LPC_PWM1->MR1 = i;
		LPC_PWM1->LER |= (1 << 1);
		LPC_PWM1->MR2 = i;
		LPC_PWM1->LER |= (1 << 2);
		for (l = 0; l < 1000000; l++);
	}

	printf("printing magenta \n");

	LPC_PWM1->MR2 = 255;
	LPC_PWM1->LER |= (1 << 2);

	for (i=255; i>=25; i=i-25){
```

```
            LPC_PWM1->MR1 = i;
            LPC_PWM1->LER |= (1 << 1);
            LPC_PWM1->MR3 = i;
            LPC_PWM1->LER |= (1 << 3);
            for (l = 0; l < 1000000; l++);
}

printf("printing cyan \n");

LPC_PWM1->MR1 = 255;
LPC_PWM1->LER |= (1 << 1);

for (i=255; i>=25; i=i-25){
            LPC_PWM1->MR2 = i;
            LPC_PWM1->LER |= (1 << 2);
            LPC_PWM1->MR3 = i;
            LPC_PWM1->LER |= (1 << 3);
            for (l = 0; l < 1000000; l++);
}

printf("printing white \n");

for (i=255; i>=25; i=i-25){
            LPC_PWM1->MR1 = i;
            LPC_PWM1->LER |= (1 << 1);
            LPC_PWM1->MR2 = i;
            LPC_PWM1->LER |= (1 << 2);
            LPC_PWM1->MR3 = i;
            LPC_PWM1->LER |= (1 << 3);
            for (l = 0; l < 1000000; l++);
}

printf("printing other colors \n");

    LPC_PWM1->MR2 = 125;
    LPC_PWM1->LER |= (1 << 2);
    LPC_PWM1->MR3 = 255;
    LPC_PWM1->LER |= (1 << 3);

    for (i=125; i>=25; i=i-25){
               LPC_PWM1->MR1 = i;
               LPC_PWM1->LER |= (1 << 1);
               for (l = 0; l < 1000000; l++);
    }

            LPC_PWM1->MR2 = 255;
            LPC_PWM1->LER |= (1 << 2);
            LPC_PWM1->MR3 = 125;
            LPC_PWM1->LER |= (1 << 3);
```

```
for (i=125; i>=25; i=i-25){
            LPC_PWM1->MR1 = i;
            LPC_PWM1->LER |= (1 << 1);
            for (l = 0; l < 1000000; l++);
}


LPC_PWM1->MR1 = 125;
LPC_PWM1->LER |= (1 << 1);
LPC_PWM1->MR3 = 255;
LPC_PWM1->LER |= (1 << 3);

for (i=125; i>=25; i=i-25){
           LPC_PWM1->MR2 = i;
           LPC_PWM1->LER |= (1 << 2);
           for (l = 0; l < 1000000; l++);
}

LPC_PWM1->MR1 = 255;
LPC_PWM1->LER |= (1 << 1);
LPC_PWM1->MR3 = 125;
LPC_PWM1->LER |= (1 << 3);

for (i=125; i>=25; i=i-25){
           LPC_PWM1->MR2 = i;
           LPC_PWM1->LER |= (1 << 2);
           for (l = 0; l < 1000000; l++);
}

LPC_PWM1->MR1 = 125;
LPC_PWM1->LER |= (1 << 1);
LPC_PWM1->MR2 = 255;
LPC_PWM1->LER |= (1 << 2);

for (i=125; i>=25; i=i-25){
           LPC_PWM1->MR3 = i;
           LPC_PWM1->LER |= (1 << 3);
           for (l = 0; l < 1000000; l++);
}

LPC_PWM1->MR1 = 255;
LPC_PWM1->LER |= (1 << 1);
LPC_PWM1->MR2 = 125;
LPC_PWM1->LER |= (1 << 2);

for (i=125; i>=25; i=i-25){
           LPC_PWM1->MR3 = i;
           LPC_PWM1->LER |= (1 << 3);
```

```
                for (l = 0; l < 1000000; l++);
        }

        LPC_PWM1->MR2 = 125;
        LPC_PWM1->LER |= (1 << 2);
        LPC_PWM1->MR3 = 125;
        LPC_PWM1->LER |= (1 << 3);

        for (i=125; i>=25; i=i-25){
                LPC_PWM1->MR1 = i;
                LPC_PWM1->LER |= (1 << 1);
                for (l = 0; l < 1000000; l++);
        }


        LPC_PWM1->MR1 = 125;
        LPC_PWM1->LER |= (1 << 1);
        LPC_PWM1->MR3 = 125;
        LPC_PWM1->LER |= (1 << 3);

        for (i=125; i>=25; i=i-25){
                LPC_PWM1->MR2 = i;
                LPC_PWM1->LER |= (1 << 2);
                for (l = 0; l < 1000000; l++);
        }

        LPC_PWM1->MR1 = 125;
        LPC_PWM1->LER |= (1 << 1);
        LPC_PWM1->MR2 = 125;
        LPC_PWM1->LER |= (1 << 2);

        for (i=125; i>=25; i=i-25){
                LPC_PWM1->MR3 = i;
                LPC_PWM1->LER |= (1 << 3);
                for (l = 0; l < 1000000; l++);
        }

  }

  return 0;
}
```