

Critical Real-Time Systems

Assignment 1 JEvans P151713

by Jonathan Ieuan Evans

FILE	4766_JONATHAN_IEUAN_EVANS_CRITICAL_REAL-TIME_SYSTEMS_ASSIGNMENT_1_JEVANS_P151713_168540_440165117.DOCX (147.14K)		
TIME SUBMITTED	15-MAR-2019 06:28PM (UTC+0000)	WORD COUNT	1570
SUBMISSION ID	102487049	CHARACTER COUNT	12067

Critical and Real-Time Application Programming Assignment 1

Jonathan Evans

P151713

15th March 2019

Contents

1. Introduction.....	5
2. Real-time Systems	6
2.1 Hard Real-time Systems.....	6
2.2 Soft Real-time Systems.....	6
2.3 Car Satellite Navigation Example.....	6
3. Alpha Blending in Assembly	7
4. Washing Machine timing constraints	9
5. Appendices.....	13

1. Introduction

This report will consist of an explanation of soft real-time systems and hard real-time systems, an example of an alpha blending program written in Assembly, and the identification of timing constraints of a washing machine system.

2. Real-time Systems

A real-time system is a computer application that must complete a given process in a timely manner.

These systems are desired in scenarios where the execution of a process hopefully finishes by a given time (also known as deadlines in real-time system terms). The outcomes of such systems could be fatal in terms of safety or in case of non-safety-critical systems, provide a degradation of application performance.

Real-time systems can be categorised as two different types of system, hard real-time systems, and soft real-time systems.

2.1 Hard Real-time Systems



A hard real-time system is a system that must execute processes before a given deadline. This can be a system where a deadline being missed could be a threat to people's safety (also known as a safety-critical system).

An example of such a system could be a system to control a nuclear power plant. This would be a safety critical application that demands deadlines to be met. If a deadline is not met, the consequences can cause death, radiation poisoning, and other dangerous outcomes.

2.2 Soft Real-time Systems

In contrast to a hard-real-time system, a soft-real time system can afford to miss deadlines. However, this may come at the cost of poor performance as more deadlines are missed.

An example of soft real-time system could be video decoding application.

When someone tries to watch a live streaming video, the video may be decoded using their CPU. Depending on the video format, this may be a difficult task, however, if the user's computer cannot decode the video in a timely manner, dropped frames will appear and the video will appear to be stuttering, resulting in a degradation of video quality due to the lowered frame rate.

2.3 Car Satellite Navigation Example

A car satnav is an example of a soft real-time system. This is the case because the mechanisms in which the satnav receives satellite communication does not happen in real-time. However, a slight latency of 1.5 seconds is acceptable as it is at the driver's discretion to drive responsibly, and cater for the 1.5 seconds delay. As the driver does not need an update instantaneously of their route, it does not make a satnav suitable for a hard real-time system, even if the satellite communication time constraint was removed, as it is the driver's fault if they cause an accident, not the system.



3. Alpha Blending in Assembly

Assembly language is used to program low level functionalities of a computing system.

Below is an example of an alpha blending program that has been created for execution on ARM platforms which utilise the thumb-16 instruction set.

The functionality of the alpha blend is written in assembly, and called from C. With this, the parameters of the function can be accessed through the values in their registers.

Below is a snippet that shows the creation of two arrays that represent bitmap images in the ARGB colour format. These arrays have been passed into the alpha blend function. Each array consists of values that make up two pixels horizontally, and 5 vertically, resulting in a 2x5 resolution image.

```
unsigned char backgroundImage [] = {
    0x00, 0xff, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
    0x00, 0xff, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
    0x00, 0xff, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
    0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
    0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00
};
unsigned char foregroundImage [] = {
    0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
    0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
    0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
    0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
    0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00
};

// This will point to the destination of the image that has been created in a
// Note: This has been multiplied by two to make it easier to see where the
unsigned char destinationImage[10*4*2] = { 0 };

board_init();

blendAlpha(backgroundImage, foregroundImage, destinationImage, 2, 5);
```

Figure 1 C file snippet (full code available in appendix 1)

The formula to perform an Alpha Blend is shown below

$$\text{result} = ((\text{foreAlpha} * \text{fore}) + (\text{bkg} * (255 - \text{foreAlpha}))) \gg 8$$

Figure 2 Alpha Blend formula (as seen in assignment brief)

The corresponding code to the function in the C file is shown in appendix 2. This ultimately consists of two loops that load each pixel horizontally in the array, performs an alpha blend, stores the output in a new array, and continues vertically to do this for every line in the array.

The results of the program are shown below.

0x20003438	80 fe 00 00 80 fe 00 00
0x20003440	80 fe 00 00 80 fe 00 00
0x20003448	80 fe 00 00 80 fe 00 00
0x20003450	80 fe 00 00 80 fe 00 00
0x20003458	80 fe 00 00 80 fe 00 00

Figure 3 New alpha blended image

Verification for the above results is shown below. For demonstration purposes, the red value of the foreground and background image will be alpha blended, and should yield 0xFE:

0xff == 255 (red)

```
(0x80 * 0xff) + (0xff * (255-0x80)) >> 8
```

Converted to decimal for convenience:

```
(128 * 255) + (255 * (255-128))
```

```
(32640) + (255 * 127)
```

```
32640 + 32385
```

```
65025
```

```
65025 in hex: 0xFE01
```

```
Shifted right by 8: 0xfe
```

Figure 4 Alpha Blending formula example

It is important to acknowledge that 0xFE is only one number away from 0xFF (which is the desired result). This has occurred as a result of shifting the value right by 8 (which effectively divides a number by 256). This has been done as it is computationally cheaper than a division of 255.

Meanwhile this may seem undesirable, the difference may be unnoticeable for such a small change in colour values.

4. Washing Machine timing constraints

This section will analyse the timing constraints involved in a wash and dry cycle on a washing machine system.

The programs that run on washing machines are responsible for a lot of actions. The table below outlines the functions involved in a wash and dry cycle. It includes the time taken for these tasks to be completed and their delays. Explanation of the delays and durations have been included where necessary. Safety critical measures have also been included detailed.

Action	Duration	Delay
1. Set Door Lock The time that the washing machine takes to lock the door is relatively short. This time takes into account the time it takes for the locking mechanism to physically lock the door. The lock protects the user by preventing them from opening the washing machine during a wash and dry cycle. As a result, the user will not be able to touch the drum if it starts moving.		< 1s
2. Opening Valve This will open a valve to allow water to flow into or out of the washing machine. The delay value is determined by the mechanism in which the valve is opened.		1s
3. Start pump This will start the pump that will drain water from the washing machine. This will take approximately 20 seconds as this will need to drain the water in the washing machine.	~20s	1s

4. Stop pump This will stop the pump when the water has been drained.		1s
5. Close valve This will close a valve, preventing more water from entering or leaving the washing machine. The water level measurements will provide the washing machine with information that will allow it to determine when the valve must be closed. If the valve did not close, the user is at risk of flooding their kitchen.		1s
6. Start turning drum The drum will begin turning near the beginning of the wash cycle. The delay of 3 seconds is mechanical as the drum will take time to speed up.	100s	3s
7. Stop turning drum As with when the drum started turning, slowing down the drum is a mechanical function that takes the time specified.		3s
8. Start Spinning drum (for rinse) Unlike turning the drum, spinning the drum happens when the washing machine needs to rinse the clothes. The duration consists of the time it takes to rinse the clothes. However, this setting may depend on timings specified by the user. The 10 seconds delay accounts for the delay for the drum to reach maximum speed.	600s	10s

9. Stop Spinning drum The drum will stop spinning after the duration of the rinse has been reached, or the user has manually interrupted the washing machine. The delay accounts for the time it takes for the drum to stop spinning.		10s
10. Sounding buzzer After a wash and dry cycle has been completed (and after the wash portion has been finished), the buzzer will be sounded. This will sound for approximately 10 seconds and will take less than 1 second to trigger.	10s	< 1s
11. Check temperature Before the drying portion of the wash and dry cycle has finished, a temperature reading will be taken to ensure that the washing machine is safe to open. This will protect the user from burning their hand on the inside of the machine.		< 1s
12. Slow/Speed-up/ or Stop drum The drums speed will vary depending on if it is rinsing clothes, turning them, re-filling with more water, or if the machine has over-heated. The drum must stop whenever the cycle before the door lock has been released, preventing the user from touching the moving drum.		1s
13. Start heater The heater will help dry the clothes quickly. This will take approximately 20 seconds to heat-up the washing machine to a desired temperature.	~20s	1s
14. Stop heater The heater can be stopped for the user to remove their washing, or for the machine to		1s

cool down in the situation where it is at the temperature where it may overheat.		
15. Measure water levels Measurements of water levels will take one second. This measurement is crucial as it will provide information to the washing machine that will allow it to close the valves when the water level is at the desired level, preventing the user from flooding their room.		1s
16. Release lock The lock will be released when it is safe to do so (and only after the user has requested the machine to stop, or a wash and dry cycle has finished).		< 1s

5. Appendices

(available on following pages)

Appendix 1

```
/*
 * Alpha blends background image with foreground
 */
extern void blendAlpha (unsigned char foreground[], unsigned char background[],
unsigned char destination[], int width, int height);

#include <asf.h>

int main (void)
{
    /*
     * Below is a pixel that is solid red:
     * 0xff == 255
     * When alpha blended, the result should be:
     * (0x80 * 0xff) + (0xff * (255-0x80)) >> 8
     * in decimal (128 * 255) + (255 * (255-128))
     * (32640) + (255 * 127)
     * 32640 + 32385
     * 65025
     * in hex: 0xFE01
     * Shifted right by 8: 0xfe
     */
    unsigned char backgroundImage [] = {
        0x00, 0xff, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
        0x00, 0xff, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
        0x00, 0xff, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
        0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
        0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00
    };
    unsigned char foregroundImage [] = {
        0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
        0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
        0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
        0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00,
        0x80, 0xff, 0x00, 0x00, 0x80, 0xff, 0x00, 0x00
    };

    // This will point to the destination of the image that has been created in
    assembly.
    // Note: This has been multiplied by two to make it easier to see where the
    memory has been filled
    unsigned char destinationImage[10*4*2] = { 0 } ;

    board_init();

    blendAlpha(backgroundImage, foregroundImage, destinationImage, 2, 5);

    return 0;
}
```

Appendix 2 (shown on proceeding pages)

```

.section .text
.global blendAlpha
.type blendAlpha, %function

/*
 * Alpha blends background image with foreground
 * Arguments passed in with function:
 * R0 = pointer to background image
 * R1 = pointer to foreground image
 * R2 = pointer to destination image
 * R3 = width
 * R4 = height
 */
blendAlpha:
    PUSH {r4, lr}
    LDR r4, [sp, #0x08]

    // Incrementer
    LDR r5, = #0

    PUSH {r5}

    // Array offset
    LDR r7, = #0
    B loadImageHorizontalEnd

/*
 * A nested loop is used to fill in arrays
 * Elements are filled in width and then height.
 */
loadImageHorizontal:
    // Save register r3 to store foreAlpha
    PUSH {r3-r4}

    /*
     * Alpha blend happens here. Each pixel is loaded by byte and stored in a
     * new array once done.
     *
     * Equation for alpha blending:
     * newImage = ((foreAlpha * fore) + (background * 255- foreAlpha))) >> 8
     *
     * Note: This will not provide full opaqueness or transparency, as a result
     * + 1 must be added to the resulting pixel.
     */

    // Load Alpha
    // *****
    LDRB r3, [r1, r7] // Alpha

    LDRB r6, [r0, r7] // Background

    STRB r3, [r2, r7] // Store alpha (no operations need to be performed here)

    ADD r7, #1 // Array offset

    // Load R
    // *****
    LDRB r4, [r1, r7] // Foreground

    LDRB r6, [r0, r7] // Background

```

```

// (foreAlpha * foreR)
PUSH {r0}
MOV r0, r3
MUL r0, r4

// (255 - foreAlpha)
PUSH {r5}
LDR r5, = #255
SUB r5, r3

// MULTIply above SUBtraction by background pixel e.g: (255 - foreAlpha) * bkgR
MUL r5, r6

// ADD above results, store in r4
ADD r4, r5, r0
POP {r5}
POP {r0}

// Shift right 8 times
LSR r4, #8

STRB r4, [r2, r7]
ADD r7, #1 // Array offset

// Load G
// *****
LDRB r4, [r1, r7] // Foreground

LDRB r6, [r0, r7] // Background

// (foreAlpha * foreR)
PUSH {r0}
MOV r0, r3
MUL r0, r4

// (255 - foreAlpha)
PUSH {r5}
LDR r5, = #255
SUB r5, r3

// MULTIply above SUBtraction by background pixel e.g: (255 - foreAlpha) * bkgR
MUL r5, r6

// ADD above results, store in r4
ADD r4, r5, r0
POP {r5}
POP {r0}

// Shift right 8 times
LSR r4, #8

STRB r4, [r2, r7]
ADD r7, #1 // Array offset

// Load B
// *****
LDRB r4, [r1, r7] // Foreground

LDRB r6, [r0, r7] // Background

// (foreAlpha * foreR)
PUSH {r0}

```

```

MOV r0, r3
MUL r0, r4

// (255 - foreAlpha)
PUSH {r5}
LDR r5, = #255
SUB r5, r3

// MULtiPLY above SUBtraction by background pixel e.g: (255 - foreAlpha) * bkgR
MUL r5, r6

// ADD above results, store in r4
ADD r4, r5, r0
POP {r5}
POP {r0}

// Shift right 8 times
LSR r4, #8

STRB r4, [r2, r7]
ADD r7, #1 // Array offset

// 1 ARGB pixel will be blended and stored in memory at this point
POP {r3-r4}

ADD r5, #1 // Increment index

loadImageHorizontalEnd:
CMP r5, r3
BLT loadImageHorizontal

/*
 * Save and reset incrementer
 */
POP {r5}
ADD r5, #1
PUSH {r5}

CMP r5, r4

LDR r5, = 0

BLT loadImageHorizontal

POP {r4, pc}

b skip
/*
 * Subroutines
 */
addOne:
ADD r4, #1
ADD r1, #3 // Advance PC by 3
MOV r15, r1

skip:
.end
.LTORG

```

Critical Real-Time Systems Assignment 1 JEvans P151713

GRADEMARK REPORT

FINAL GRADE

77/100

GENERAL COMMENTS

Instructor

Knowledge (20): 15

Cognitive (40): 30

Practical (40): 32

Car sat nav scenario discussion provides quite a nice analysis of both hard/software. References missing.

Requirements analysis provides table of actions and times. But what are the sequences of actions required? More analysis was required.

C Code

Uses image arrays but results were checked by visual inspection. ARGB pixel values obtained and alpha blending performed but the code was duplicated 3 times

Blending of arrays did row and column calculations which was a bit complicated

Bitmask handling not provided.

In general the assembler was fine but the code structure could have been improved by using functions to split the assembler.

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

References



Comment 2

Reasonable conclusion would like to have seen a bit more debate

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16
