

p149064 CritRealAppProg

by Robert McIntyre

FILE	2783_ROBERT_MCINTYRE_P149064_CRITREALAPPPROG_168540_1946 243705.DOCX (337.3K)		
TIME SUBMITTED	22-MAR-2019 01:40PM (UTC+0000)	WORD COUNT	3455
SUBMISSION ID	103008063	CHARACTER COUNT	32630

Critical and Real-Time Application Programming

Part 1

Robert McIntyre

M6X01090

Mike Dacey

Table of Contents

Introduction	3
Real-time Systems	3
Hard Real Time System	3
Soft Real Time Systems	3
Sat navigation Example	3
Alpha Blending	4
Testing.....	4
Test 1 – Test of one byte.....	4
Test 2 – Testing the whole ARGB of one pixel.	5
Test 3 – Calculation on whole image	7
Micro Controller steps for the washing machine cycle	8
Safety feature steps for washing machine.....	11
Appendices.....	12
FunctionExample.s	12
main.c.....	16

Introduction

This report demonstrates the roles of critical systems and some safety aspects needed in the equipment. It describes different types of real-time systems available and where they are used. The report includes programming a microcontroller board in assembler to alpha blend two images together. The final section includes the steps in a washing machine cycle.

Real-time Systems

There are two main types of real-time systems in computer systems. These are hard and soft real-time systems. Real Time Operating Systems (RTOS) are usually found in the embedded systems. Time is the constraint in these systems as there is a need to produce accurate outputs at a particular time. This differs from a General Purpose Operating system (GPOS) which produces accurate results with no time limit. The main difference between the hard and soft real time systems is how a failure could be tolerated.

Hard Real Time System

A hard real time system has software which must operate on a time deadline. There is no delay allowed and a missed deadline will result in a critical failure. This type of system is used when there cannot be any failures. Examples of this type of system are aircraft control systems, airbags in vehicles and pacemaker.

Soft Real Time Systems

Soft real-time systems can miss a couple of instructions without it affecting the overall outcome. The critical times in these systems may be delayed slightly but this will result in a reduced performance of the software output. Examples of these are mobile phones, digital cameras and virtual reality.

Sat navigation Example



A sat navigation system is a soft real-time system as it is not a critical failure when the updating and recalculating of the sat nav is slightly late. In the sat nav there is a GPS system that locates the correct position of the car and compares the GPS coordinates to a map that is stored in memory. The CPU calculates the route from the location to the destination and displays the result on the screen. There is a latency of 1.5 seconds in updating to the user. Therefore, the deadline for acquiring new data can be missed slightly without the route. Even if the data is missed for a short time because of loss of signal due to a tunnel for example the system does not fail. When the signal is reestablished it will update a new GPS location.



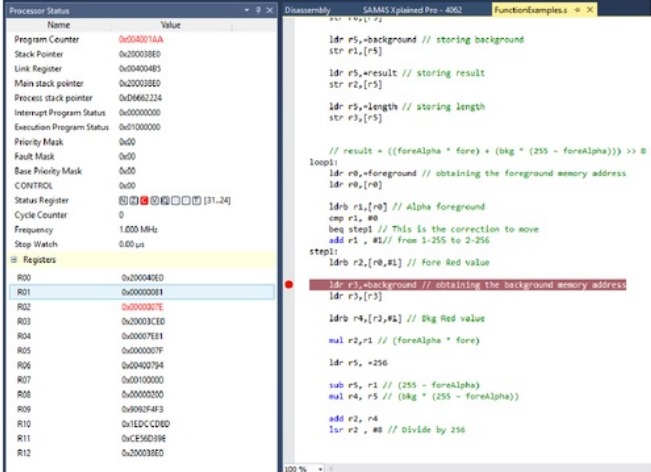
Alpha Blending

In order to complete the alpha blending task first the code was written to combine the first red byte of information from the foreground and background array. In this stage the correction value was also used changing it from 1 – 255 to 2 – 256. This is shown in test one. Next the calculation of a whole pixel is done including a loop and a counter to count 4 bytes see test two. Finally the calculation on the whole image is done see test three.

Testing

Test 1 – Test of one byte

This test involves changing the first value in the ARGB. The first value being Red value which is 0x00 and the alpha value being 0x80. The correction has been added in where



The screenshot shows a debugger window with the following components:

- Processor Status:** A table showing various system registers and their values.
- Registers:** A list of registers R00 through R12 with their current values.
- Disassembly:** A window showing the assembly code for the function `FunctionExamples`. The code includes comments and instructions for storing background and foreground data, calculating the result, and performing a loop for alpha blending.
- Blend:** A section of code showing the final blending calculation.

Processor Status Table:

Name	Value
Program Counter	0x000016A
Stack Pointer	0x200038E0
Link Register	0x0040485
Main stack pointer	0x200038E0
Process stack pointer	0x04662224
Interrupt Program Status	0x00000000
Execution Program Status	0x01000000
Priority Mask	0x00
Fault Mask	0x00
Base Priority Mask	0x00
CONTROL	0x00
Status Register	0x00000000 [31-24]
Cycle Counter	0
Frequency	1.000 MHz
Stop Watch	0.00 µs

Registers Table:

Register	Value
R00	0x200040E0
R01	0x00000001
R02	0x0000000E
R03	0x20003CE0
R04	0x00007E81
R05	0x0000007F
R06	0x00400794
R07	0x01000000
R08	0x00000000
R09	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x200038E0

Disassembly Code:

```
ldr r5,=background // storing background
str r1,[r5]

ldr r5,=result // storing result
str r2,[r5]

ldr r5,=length // storing length
str r3,[r5]

// result = ((foreAlpha * fore) + (bkg * (255 - foreAlpha))) >> 8
loop1:
ldr r0,=foreground // obtaining the foreground memory address
ldr r0,[r0]

ldrb r1,[r0] // Alpha foreground
cmp r1, #0
beq step1 // This is the correction to move
add r1, #1 // from 1-255 to 2-256

step1:
ldrb r2,[r0,#1] // Fore Red value

ldr r3,=background // obtaining the background memory address
ldr r3,[r3]

ldrb r4,[r3,#1] // Bkg Red value

mul r2,r1 // (foreAlpha * fore)

ldr r5, =256

sub r5, r1 // (255 - foreAlpha)
mul r4, r5 // (bkg * (255 - foreAlpha))

add r2, r4
lsr r2, #8 // Divide by 256
```

Blend Code:

```
//r0 // foreground
//r1 // Background
//r2 // Result
//r3 // length

// Storing the memory address to flash to free some of the registers
ldr r5,=foreground // storing foreground
str r0,[r5]

ldr r5,=background // storing background
str r1,[r5]

ldr r5,=result // storing result
str r2,[r5]

ldr r5,=length // storing length
str r3,[r5]

// result = ((foreAlpha * fore) + (bkg * (255 - foreAlpha))) >> 8
loop1:
ldr r0,=foreground // obtaining the foreground memory address
ldr r0,[r0]

ldrb r1,[r0] // Alpha foreground
cmp r1, #0
beq step1 // This is the correction to move
add r1, #1 // from 1-255 to 2-256

step1:
ldrb r2,[r0,#1] // Fore Red value

ldr r3,=background // obtaining the background memory address
ldr r3,[r3]

ldrb r4,[r3,#1] // Bkg Red value

mul r2,r1 // (foreAlpha * fore)

ldr r5, =256

sub r5, r1 // (255 - foreAlpha)
mul r4, r5 // (bkg * (255 - foreAlpha))

add r2, r4
lsr r2, #8 // Divide by 256
```

With the calculations placed in with the included offsets in the memory address for both the foreground and the background. This tests to see if this has worked with one value. The next step is to see if I can do it with the red and the green value.

Test 2 – Testing the whole ARGB of one pixel.

```

FunctionExamples  SAMMS Vplanned Pro - 4352  main.c
obtainBackground:
    ldr r1,[r0] // Getting the Alpha foreground
    ldr r5,=alphaforeground
    strb r1,[r5] // Storing the alpha value in memory to use for later

    ldrb r4,[r5,#1] // Getting the blue fore value from memory
    ldrb r5,[r1,#0] // Getting the blue back value from memory
    push {r1,r4} //Pushing the blue values on the stack. Background first

    ldrb r4,[r5,#2] // Getting the green fore value from memory
    ldrb r5,[r1,#1] // Getting the green back value from memory
    push {r1,r4} //Pushing the Green values on the stack. Background first

    ldrb r4,[r5,#3] // Getting the red fore value from memory
    ldrb r5,[r1,#2] // Getting the red back value from memory
    push {r1,r4} // Pushing the Red values on the stack. Background first

loop:
    ldr r1,=counter // obtaining the counter memory address
    ldr r4,[r1] // storing the counter into the register.
    cmp r4, #4
    bne Resetting
    ldr r2, #0
    strb r2,[r1] // change r4 to r3
Resetting:
    ldrb r3,[r1]
    add r1,#1 // Incrementing the counter
    strb r3,[r1] // Storing the counter in memory

    ldrb r1,[r0] // Alpha foreground
    ldr r1,=alphaforeground
    ldr r4,[r1]
    cmp r1, #0
    bne steps // This is the correction to move
    add r1, #1 // from 1-255 to 2-256
steps:
    pop {r2,r4} //Fore Red, Back Red
    mul r2,r1 // (foreAlpha * fore)

    ldr r5,=256
    sub r5,r1 // (255 - foreAlpha)
    mul r4,r5 // (bkg * (255 - foreAlpha))

    add r1,r4 // (foreAlpha * fore) + (bkg * (255 - foreAlpha))
    lsr r2,r2, #8 // Divide by 256

    // R2 CONTAINS RESULTS FOR THE BYTE
    ldrb r1,[r0] // Alpha foreground
    ldr r4,=blendResultsAddr
    ldr r4,[r4]
    ldr r3,=counter
    ldrb r3,[r3]
    cmp r3, #1
    bne notAlpha
    strb r1,[r4] // Storing the the alpha value in the BlendResults array
    add r4,r4, #1
notAlpha:
    strb r2,[r4] // Storing the result of the either the RR GG BB values
    add r4,r4, #1 // incrementing the pointer
    ldr r5,=blendResultsAddr
    str r5,[r5]
    add r3,r3, #1 // counter
    cmp r3, #4
    bne loop

    ldr r1,=foreground
    ldr r0,[r1]
    add r0,r0, #4
    str r0,[r1]
    ldr r1,=background
    ldr r3,[r1]
    add r3,r3, #4

```

To do the first pixel I have place the RR GG BB values on to the stack in reverse order as the stack is a first in last out operation. The Alpha is stored on the memory to enable it to be used throughout the loop. The answers are stored in an array. The Answer for the first pixel is 80 7E 7E FF.

```

mul r2,r1 // (foreAlpha * fore)

ldr r5,=256
sub r5,r1 // (255 - foreAlpha)
mul r4,r5 // (bkg * (255 - foreAlpha))

add r1,r4 // (foreAlpha * fore) + (bkg * (255 - foreAlpha))
lsr r2,r2, #8 // Divide by 256

// R2 CONTAINS RESULTS FOR THE BYTE

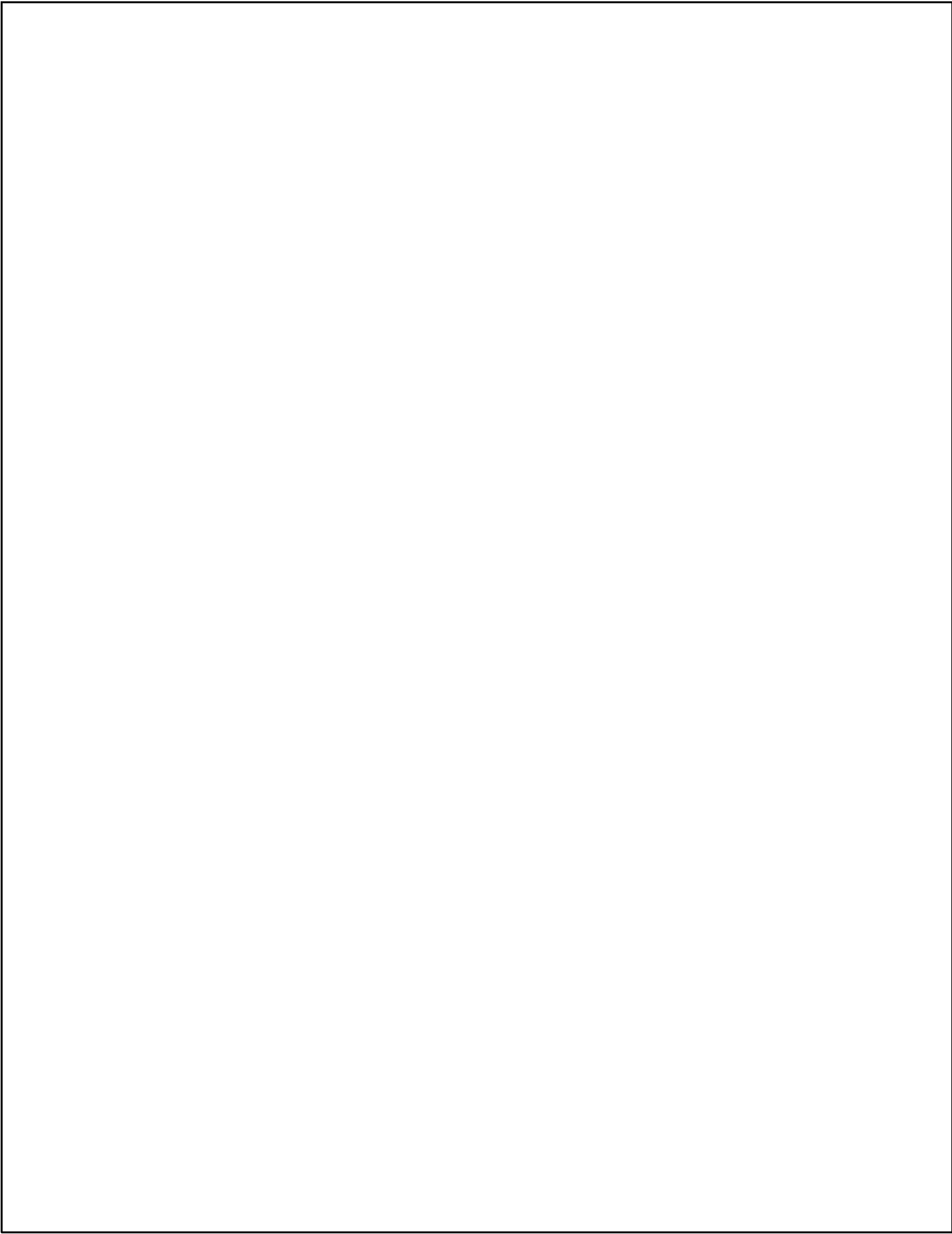
ldrb r1,[r0] // Alpha foreground
ldr r4,=blendResultsAddr
ldr r4,[r4]
ldr r3,=counter
ldrb r3,[r3]
cmp r3, #1
bne notAlpha
strb r1,[r4] // Storing the the alpha value in the BlendResults array
add r4,r4, #1
notAlpha:
strb r2,[r4] // Storing the result of the either the RR GG BB values
add r4,r4, #1 // incrementing the pointer
ldr r5,=blendResultsAddr
str r5,[r5]
add r3,r3, #1 // counter
cmp r3, #4
bne loop

ldr r1,=foreground
ldr r0,[r1]
add r0,r0, #4
str r0,[r1]
ldr r1,=background
ldr r3,[r1]
add r3,r3, #4

```

Memory 4

Type	Memory	base PERIPHERALS
0x20000000	80 7E 7E FF	End
0x20000004	00 00 00 00	...



Test 3 – Calculation on whole image

There is another counter and loop added from test two. The results of the alpha blending go from the memory space 0x20000000 - 0x200003FF. Memory address for the foreground and background are increased by 4 to move to the next pixel to continue the calculation.

```

0x20000000 80 7e 7e ff 0x200003A0 80 ff 7e 7e
0x20000004 80 7e 7e ff 0x200003A4 80 ff 7e 7e
0x20000008 80 7e 7e ff 0x200003A8 80 ff 7e 7e
0x2000000C 80 7e 7e ff 0x200003AC 80 ff 7e 7e
0x20000010 80 7e 7e ff 0x200003B0 80 ff 7e 7e
0x20000014 80 7e 7e ff 0x200003B4 80 ff 7e 7e
0x20000018 80 7e 7e ff 0x200003B8 80 ff 7e 7e
0x2000001C 80 7e 7e ff 0x200003BC 80 ff 7e 7e
0x20000020 ff 00 00 ff 0x200003C0 ff ff 00 00
0x20000024 ff 00 00 ff 0x200003C4 ff ff 00 00
0x20000028 ff 00 00 ff 0x200003C8 ff ff 00 00
0x2000002C ff 00 00 ff 0x200003CC ff ff 00 00
0x20000030 ff 00 00 ff 0x200003D0 ff ff 00 00
0x20000034 ff 00 00 ff 0x200003D4 ff ff 00 00
0x20000038 ff 00 00 ff 0x200003D8 ff ff 00 00
0x2000003C ff 00 00 ff 0x200003DC ff ff 00 00
0x20000040 80 7e 7e ff 0x200003E0 80 ff 7e 7e
0x20000044 80 7e 7e ff 0x200003E4 80 ff 7e 7e
0x20000048 80 7e 7e ff 0x200003E8 80 ff 7e 7e
0x2000004C 80 7e 7e ff 0x200003EC 80 ff 7e 7e
0x20000050 ff 00 00 ff 0x200003F0 80 ff 7e 7e
0x20000054 ff 00 00 ff 0x200003F4 80 ff 7e 7e
0x20000058 ff 00 00 ff 0x200003F8 80 ff 7e 7e
0x2000005C ff 00 00 ff 0x200003FC 80 ff 7e 7e
0x20000060 ff 00 00 ff 0x20000400 00 00 00 00
0x20000064 ff 00 00 ff 0x20000404 00 00 00 00
0x20000068 ff 00 00 ff 0x20000408 00 00 00 00
0x2000006C ff 00 00 ff 0x2000040C 00 00 00 00
0x20000070 ff 00 00 ff 0x20000410 00 00 00 00
0x20000074 ff 00 00 ff 0x20000414 00 00 00 00
0x20000078 ff 00 00 ff 0x20000418 00 00 00 00
0x2000007C ff 00 00 ff 0x2000041C 00 00 00 00
----- -- -- -- 0x20000420 00 00 00 00

```

Processor Status

Name	Value
Program Counter	0x00400230
Stack Pointer	0x20003CF0
Link Register	0x00400543
Main stack pointer	0x20003CF0
Process stack pointer	0x07662224
Interrupt Program Status	0x00000000
Execution Program Status	0x01000000
Priority Mask	0x00
Fault Mask	0x00
Base Priority Mask	0x00
CONTROL	0x00
Status Register	0x00000000 [31..24]
Cycle Counter	0
Frequency	1.000 MHz
Step Watch	0.00 µs

Registers

R00	0x200044F0
R01	0x2000101C
R02	0x00000400
R03	0x200040F0
R04	0x00000400
R05	0x00000000
R06	0x2000100C
R07	0x00000000
R08	0x00000200
R09	0x002F4F3

Disassembly

SAMMS Xplained Pro -4062

FunctionExamples.s

main.c

```

ldr r3,[r0] // alpha foreground
ldr r4,=blendResultsAddr
ldr r4,[r4]
ldr r3,=counter
ldr r3,[r3]
cmp r3,#1
bne notAlpha
strb r3,[r4] // storing the alpha value in the blendResults array
add r4,#1
notAlpha:
strb r2,[r4] // storing the result of the either the RR GG BB values
add r4,#1 // incrementing the pointer
ldr r5,=blendResultsAddr
str r4,[r5]
add r3,#1 // counter
cmp r3,#4
bgt loop2
ldr r1,=foreground
ldr r0,[r1]
add r0,#1
str r0,[r1]
ldr r1,=background
ldr r3,[r1]
add r3,#1
str r3,[r1]
//str r0,
// LAST PROGRESS
ldr r2,=1824
ldr r4,=counter
ldr r5,#0
str r5,[r4]
ldr r2,=counter2
ldr r4,[r1]
add r4,#1
str r4,[r1]
cmp r4,r2 //
bgt obtainingValues

```

Micro Controller steps for the washing machine cycle

No	Action	Duration	Delay
1.	Starting the Wash - User selects a wash cycle on the machine. The values of this cycle are sent from the interface module and stored on the EEPROM module on the main board. - Button to start the process is pressed.	1 sec	1 sec
2.	Locking the Door - When the button is pressed the When the door is shut there is a latch to check if door is properly shut (Bit check of an off or on state) - Once the latch is in the on state the electromagnet will turn on with an on state so the door cannot be opened. - When the door is not properly locked the buzzer alerts the user. - The user re-attempts to shut the door	2 sec	1s
3.	Turning of the drum sequence - The main CPU sends a signal to through the output buffer to the communication bus and to the power driver - Once the signal reaches the power driver the CPU that is on the power board uses a sequence to turn on the motor module clockwise for 1min stop for 3 seconds anti-clockwise for a 1min stop for 3 seconds and then repeat on a loop. Carries on sequence until it is given a signal to stop.	2 min per loop	6 sec
4.	Adding the Water to the Drum - The main CPU sends the signal through the communication bus to the power driver to open the valves on the pump to allow water to enter the drum	10 min	2 min
5.	Close Water pump - Power driver board sends the signal to close the valve in the pump	1 sec	10 min

6.	Turn on the heater - The main CPU sends the signal through the communication bus. The CPU on the power driver will send a signal to the heater module to turn on the module	Until temp reached ~30 min	5 min
7.	Measure Temperature - Gather the information from the EEPROM memory to see what temperature the user wanted their clothes at - The main CPU sends a signal to the input module to measure the temperature to the required temperature from the user	Until temp reached ~30 min	5 min
8.	Turn off heater - Signal sent from input sensor to the CPU that the temperature has been reached. - The main CPU sends the signal through the communication bus. The CPU on the power driver will send a signal to the heater module to turn off the module	1 sec	Until temp reach
9.	Open drain valve - The main CPU sends the signal through the communication bus to open the drain valve to drain the water that is in the drum	1 sec	1 hour 40 min
10	Increase spinning speed - The main CPU sends the signal through the communication bus to the power driver to change the value to increase the speed of the motor. The CPU increases the speed of the drum	5 min	3 sec
11	Stop spinning - The main CPU sends the signal through the communication bus to the power driver to turn off the power module.	30 sec	5 min
12	Close drain valve - The main CPU sends the signal through the communication bus to close the drain valve	1 sec	30 sec
13	Rinse Cycle - The main CPU sends a signal through the communication bus to the interface module and changes the output interface to a rinse cycle.	1 sec	1 sec
14	Open water valve - The main CPU sends the signal through the communication bus to the power driver to open the valves on the pump to allow water to enter the drum	10 min	2 min
15	Turn drum	2 min per loop	6 sec

	- The main CPU sends a signal to through the output buffer to the communication bus and to the power driver - Once the signal reaches the power driver the CPU that is on the power board uses a sequence to turn on the motor module clockwise for 1min stop for 3 seconds anti-clockwise for a 1min stop for 3 seconds and then repeat on a loop. Carries on sequence until it is given a signal to stop.		
16	Close Water pump - Power driver board sends the signal to close the valve in the pump	1 sec	10 min
17	Open drain valve - The main CPU sends the signal through the communication bus to open the drain valve to drain the water that is in the drum	5 min	1s
18	Increase spinning speed - The main CPU sends the signal through the communication bus to the power driver to change the value to increase the speed of the motor. The CPU increases the speed of the drum -	5 min	3 sec
19	Close drain valve - The main CPU sends the signal through the communication bus to close the drain valve	1 sec	30 sec
20	Turn drum - The main CPU sends a signal to through the output buffer to the communication bus and to the power driver - Once the signal reaches the power driver the CPU that is on the power board uses a sequence to turn on the motor module clockwise for 1min stop for 3 seconds anti-clockwise for a 1min stop for 3 seconds and then repeat on a loop. Carries on sequence until it is given a signal to stop.	2 min per loop	6 sec
21	Repeat from step 14 x3 - This is repeating from step 14 three times		
22	Open drain valve - The main CPU sends the signal through the communication bus to open the drain valve to drain the water that is in the drum	5 min	1s
23	Spin Cycle - The main CPU sends a signal through the communication bus to the interface module and changes the output interface to a rinse cycle.	1 sec	1 sec
24	Increase spin speed	10 min	3 sec

	- The main CPU sends the signal through the communication bus to the power driver to change the value to increase the speed of the motor. The CPU increases the speed of the drum		
25	Stop Spinning - The main CPU sends the signal through the communication bus to the power driver to turn off the power module.	1 sec	10 min
26	Turn Drum - The main CPU sends a signal to through the output buffer to the communication bus and to the power driver - Once the signal reaches the power driver the CPU that is on the power board uses a sequence to turn on the motor module clockwise for 1min stop for 3 seconds anti-clockwise for a 1min stop for 3 seconds and then repeat on a loop. Carries on sequence until it is given a signal to stop.	2 min per loop	6 sec
27	Close drain valve - The main CPU sends the signal through the communication bus to close the drain valve	1 sec	30 sec
28	Stop the drum - The main CPU sends the signal through the communication bus to the power driver to turn off the power module.	30 sec	1 sec
29	Release the door lock - The main CPU sends a signal to turn off the electro magnet to open the door. There needs to be a delay	1 sec	1 min
30	Sound the buzzer - The main CPU sends a signal to the speaker to making a beeping sound to alert the user the washing machine is finished.	1 min	1 min

Safety feature steps for washing machine

A washing machine cycle has critical steps that need safety features which may require delays in the system. The first of these is locking the door. This is required to prevent the door being open when either the drum is turning, or the machine is full of water. If the temperature of the system is above the required value the system needs to automatically turn off the heater. There needs to be a safety feature where the pump will not start if water valve is not open. When the cycle has finished there needs to be a delay to check the drum has stopped turning and there is no water in the drum before the door lock is released so that the door can be opened safely. This also happens if the user cycle has been interrupted.

Appendices

FunctionExample.s

```
/*
 * FunctionExamples.s
 *
 * Created: 26/09/2016 12:43:18
 * Author: sc-md
 */
.section .text
.global myFunction, BlendByPixel, Blend, ClearRegister
.type myFunction, %function
.type Blend, %function
.type ClearRegister, %function
.type BlendByPixel, %function

BlendByPixel:

    /*
        //result = foreAlpha * fore + (1 - foreAlpha) * bkg
        result = 1 * fore + (1 - 1) * bkg = fore           (the foreground
image is opaque/fully visible)
        result = 0 * fore + (1 - 0) * bkg = bkg           (the
foreground image is invisible)
        result = 0.5 * fore + (1 - 0.5) * bkg = 50% of both (foreground is 50%
transparent)
    */
myFunction:

Blend:
    // Order of arguments being passed to function
    // foreground
    // Background
    // Result
    // length

    // Storing the ememory address to flash to free some of the registers
    ldr r5,=foreground // storing foreground str r0,[r5]

    ldr r5,=background // storing background
    str r1,[r5]

    ldr r5,=length // storing length
    str r2,[r5]

    ldr r2,=blendResultsAddr // loads the memory address for blend results
    ldr r5,=blendResults // storing length
    str r5,[r2]

    // ldr r6, =0 // Counter for the number of 4 x 1 bytes
    ldr r7, =0 // Counter for ARGB

    // Maths equation to calculate the pixel
```

```

    // result = ((foreAlpha * fore) + (bkg * (255 - foreAlpha))) >> 8

    ldr r0,=foreground // obtaining the foreground memory address
    ldr r0,[r0]

    ldr r3,=background // obtaining the background memory address
    ldr r3,[r3]

    // This is placing the AARRGGBB Values into the stack pointer
obtainingValues:
    ldrb r1,[r0] // Getting the Alpha foreground
    ldr r6,=alphaForeground
    strb r1,[r6] // Storing the alpha value in memory to use for later

    ldrb r4,[r0,#3] // Getting the blue fore value from memory
    ldrb r5,[r3,#3] // Getting the blue Back value from memory
    push {r5,r4} //Pushing the blue values on the stack. Background first

    ldrb r4,[r0,#2] // Getting the Green fore value from memory
    ldrb r5,[r3,#2] // Getting the Green Back value from memory
    push {r5,r4} //Pushing the Green values on the stack. Background first

    ldrb r4,[r0,#1] // Getting the Red fore value from memory
    ldrb r5,[r3,#1] // Getting the Red Back value from memory
    push {r5,r4} // Pushing the Red values on the stack. Background first

loop1:
    ldr r3,=counter // obtaining the Counter memory address
    ldr r4,[r3] // storing the counter into the register.
    cmp r4, #4
    ble Resetting
    ldr r2, =0 // resetting counter
    strb r2,[r3]
Resetting:
    ldrb r5,[r3]
    add r5,#1 // Incrementing the counter
    strb r5,[r3] // Storing the counter in memory

    ldrb r1,[r0] // Alpha foreground

    ldr r1,=alphaForeground // Loading the address for the alpha foreground
    ldr r1,[r1]
    cmp r1, #0
    beq step1 // This is the correction to move
    add r1 , #1// from 1-255 to 2-256
step1:

    pop {r2,r4} // loading the pixels of Foreground Red, Background Red

    mul r2,r1 // (foreAlpha * fore)

    ldr r5, =256
    sub r5, r1 // (256 - foreAlpha)
    mul r4, r5 // (bkg * (256 - foreAlpha))

    add r2, r4 // (foreAlpha * fore) + (bkg * (255 - foreAlpha))

```

```

lsr r2 , #8 // Divide by 256

// R2 CONTAINS RESULTS FOR THE BYTE

ldrb r1,[r0] // loading the Alpha foreground
ldr r4, =blendResultsAddr // loading the alpha blend address
ldr r4,[r4]
ldr r3, =counter // loads the counter for AARRGGBB
ldrb r3,[r3]
cmp r3, #1 // checks to see if the counter is equal to one to load the alpha value
into memory
bne notAlpha
strb r1,[r4] // Storing the the alpha value in the BlendResults array
add r4, #1
notAlpha:
strb r2,[r4] // Storing the result of the either the RR GG BB values
add r4, #1 // Incrementing the pointer
ldr r5, =blendResultsAddr //loading the blend result address
str r4,[r5]
add r3, #1 // incrementing the counter
cmp r3, #4 // checks to see if it the end of the loop
blt loop1

// loads the foreground and background into the registers ready for the next loop
ldr r1,=foreground
ldr r0,[r1]
add r0,#4
str r0,[r1]
ldr r1,=background
ldr r3,[r1]
add r3,#4
str r3,[r1]

ldr r2,=1024 // length of the array // this could retrieve the value from memory
instead

ldr r4,=counter
ldr r5, =0 // resets the counter to 0
str r5,[r4]
ldr r1, =counter2
ldr r4,[r1]
add r4, #4 // add the memory address by 4
str r4,[r1] // Stores the new value of the counter

cmp r4,r2 //
blt obtainingValues
pop {pc} // Takes off the program counter from the stack
add r1, #1 // This line needs to be taken out

ClearRegister: // This is to set all the registers to 0
push {lr}
ldr r0, = #0
ldr r1, = #0
ldr r2, = #0
ldr r3, = #0
ldr r4, = #0
ldr r5, = #0
ldr r6, = #0

```

```

    ldr r7, = #0
    pop {pc}

.section .data
    blendResults: .space 4 * 1024 // This is the array to store the blended results
    blendResultsAddr: .space 4 // Stores the address of the Blended Results
    background: .space 4 // Stores the address for th background when passed as an
argument
    foreground: .space 4 // Stores the address for th foreground when passed as an
argument
    alphaForeground: .space 4 // Stores the alpha value when looped each AARRGGBB
    result: .space 4 // Stores the result
    length: .space 4 // Stores the length of the array when passed as an argument
    counter: .space 4 // Stores the counter for each pixel
    counter2: .space 4 // Stores the counter for the whole picture
.end
.LTORG

```

main.c

```
/**
 * \mainpage User Application template doxygen documentation
 *
 * \par Empty user application template
 *
 * This is a bare minimum user application template.
 *
 * For documentation of the board, go \ref group_common_boards "here" for a link
 * to the board-specific documentation.
 *
 * \par Content
 *
 * -# Include the ASF header files (through asf.h)
 * -# Minimal main function that starts with a call to board_init()
 * -# Basic usage of on-board LED and button
 * -# "Insert application code here" comment
 */

/*
 * Include header files for all drivers that have been imported from
 * Atmel Software Framework (ASF).
 */
/*
 * Support and FAQ: visit <a href="http://www.atmel.com/design-support/">Atmel
Support</a>
 */

#include <asf.h>
#include <stdio.h>

extern int myFunction(int i);

extern void Blend(unsigned char fore[], unsigned char back[], int length); // If width in
pixels then multiply by 4
extern void ClearRegister();
extern void BlendByPixel(unsigned char fore[], unsigned char back[], int length); // If
width in pixels then multiply by 4

int main (void)
{
    unsigned char foregroundImage[] = {

        0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00
        ,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0xFF,0x00,0x00,0xF
        F,0xFF,0x00,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x
        00,0x00,0xFF,0xFF,0x00,0x00,0xFF,0xFF,0x00,0x00,0xFF,

        0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00
        ,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0xFF,0x00,0x00,0xF
        F,0xFF,0x00,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x00,0xFF,0xFF,0x
        00,0x00,0xFF,0xFF,0x00,0x00,0xFF,0xFF,0x00,0x00,0xFF,

        0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00,0x00,0xFF,0x80,0x00
    }
}
```

```
0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF,
0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF, 0x00, 0x00, 0x80, 0xFF, 0x00, 0x00,
0, 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,
```



```
int foregroundImageLength = sizeof(foregroundImage)/sizeof(char);
foregroundImageLength = foregroundImageLength /4;
int backgroundImageLength = sizeof(backgroundImage)/sizeof(char);
board init();
```

```
int height = 16;
```

```

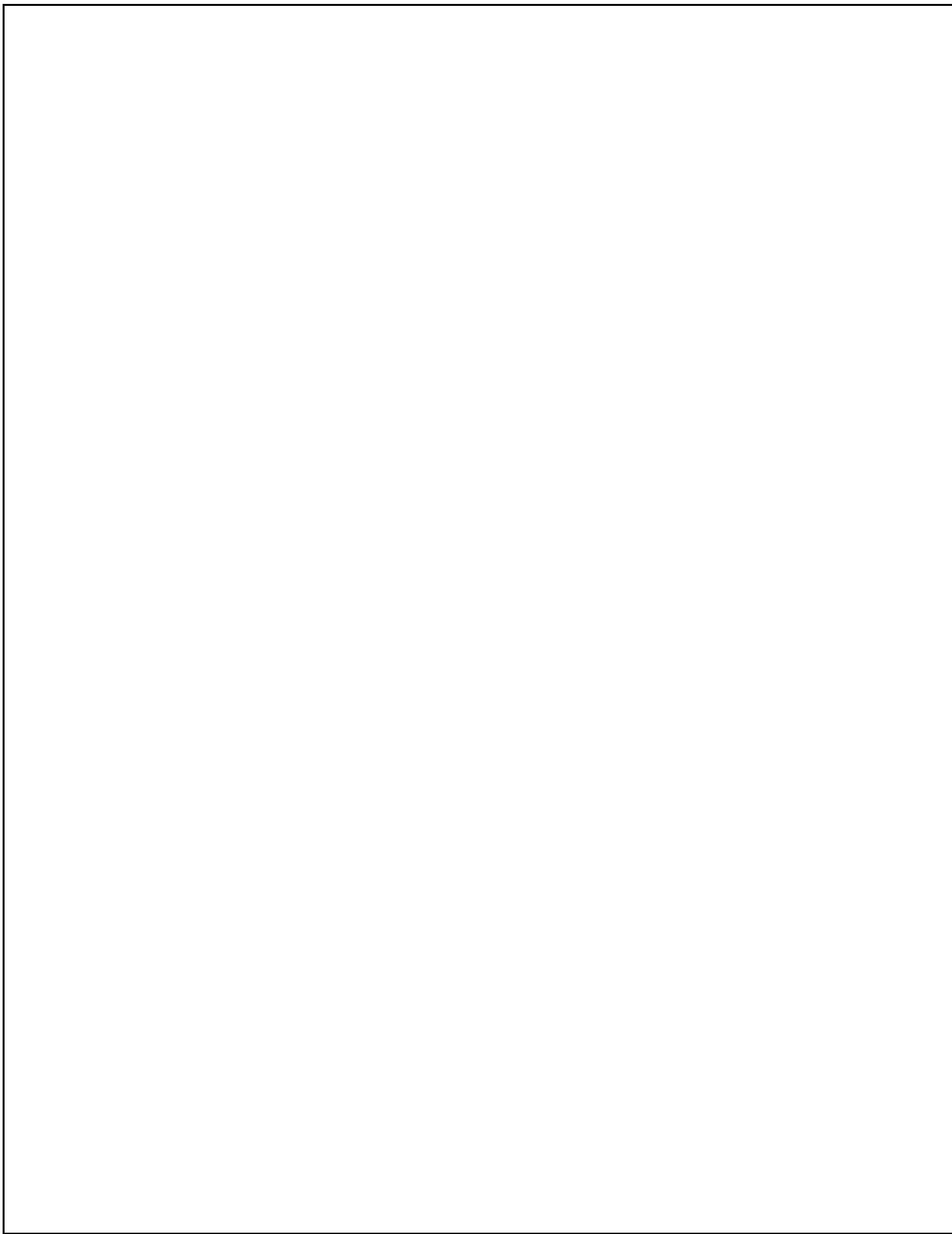
    int width = 16;
    //ClearRegister();
    //BlendByPixel(foregroundImage, backgroundImage, foregroundImageLength); //
Testing
    Blend(foregroundImage, backgroundImage, foregroundImageLength);
    int result = myFunction(2); // Test values
    result += myFunction(result);
    //Blend(*foregroundImage,*backgroundImage,*resultsImage,16,16);

    /* Insert application code here, after the board has been initialized. */

    /* This skeleton code simply sets the LED to the state of the button. */
    while (1) {

        /* Is button pressed? */
        if (ioport_get_pin_level(BUTTON_0_PIN) == BUTTON_0_ACTIVE) {
            /* Yes, so turn LED on. */
            ioport_set_pin_level(LED_0_PIN, LED_0_ACTIVE);
        } else {
            /* No, so turn LED off. */
            ioport_set_pin_level(LED_0_PIN, !LED_0_ACTIVE);
        }
    }
}

```



FINAL GRADE

80/100

GENERAL COMMENTS

Instructor

Knowledge (20): 15

Cognitive (40): 33

Practical (40): 32

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

The requirements analysis provides quite a detailed table of actions and timings. However, there should have been more analysis of the sequence of actions and features required,

C code

Arrays for images provided for background and foreground, with the blended result stored in destination array created in assembler. Testing was performed by visual inspection.

Assembler

ARGB components of pixel extracted and blended using alpha blending.

No assembler functions were used and code could have been better structured.

No bitmask code provided.

PAGE 1

PAGE 2

PAGE 3



Comment 1

References missing



Comment 2

Reasonable arguments provided, I would liked to have seen more analysis of soft/hard systems

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

