AN204459

S6J3200 Series Evaluation for Png Image Decoding

Target Products: Refer to Section 2

This application note describes the evaluation of the performance for the decoding PNG data with S6J3200 Series MCU and describes how to use and how to implement PNG library functions for the decoding software.

Contents

1 Introduction

1.1 About Document

This application note describes the evaluation of the performance for the decoding PNG data with S6J3200 Series MCU. S6J3200 series has (2D/3D) Graphic subsystem function for the cluster application system, but the image data to input Graphic subsystem, is decoded by the software, for example, in case of PNG file. Many users might concern MCU performance for this process with our S6J3200 Series MCU. For this sake, the MCU performance for PNG decoding process is evaluated with S6J326C MCU.

This process is executed using APIs which are provided open source PNG library (libpng).

Then this application note also describes how to use and how to implement PNG library functions for this evaluation software

We disclose our sample project which was used for this evaluation. It will be good for the user when user check this document with our sample project.

1.2 About PNG library

“libpng” is the open source library for the reference usage of PNG Image file. It can be downloaded from the following website.

http://www.libpng.org/pub/png/libpng.html

This time we use “libpng” version 1.6.14.

2 Target Products

Products of what is described in this operation manual are as follows.

Series	Breed Type Rating (Package Suffix Except)
S6J3200	S6J323A, S6J323C, S6J324A, S6J324C, S6J325A, S6J325C, S6J326A, S6J326C, S6J327A, S6J327C, S6J328A, S6J328C,

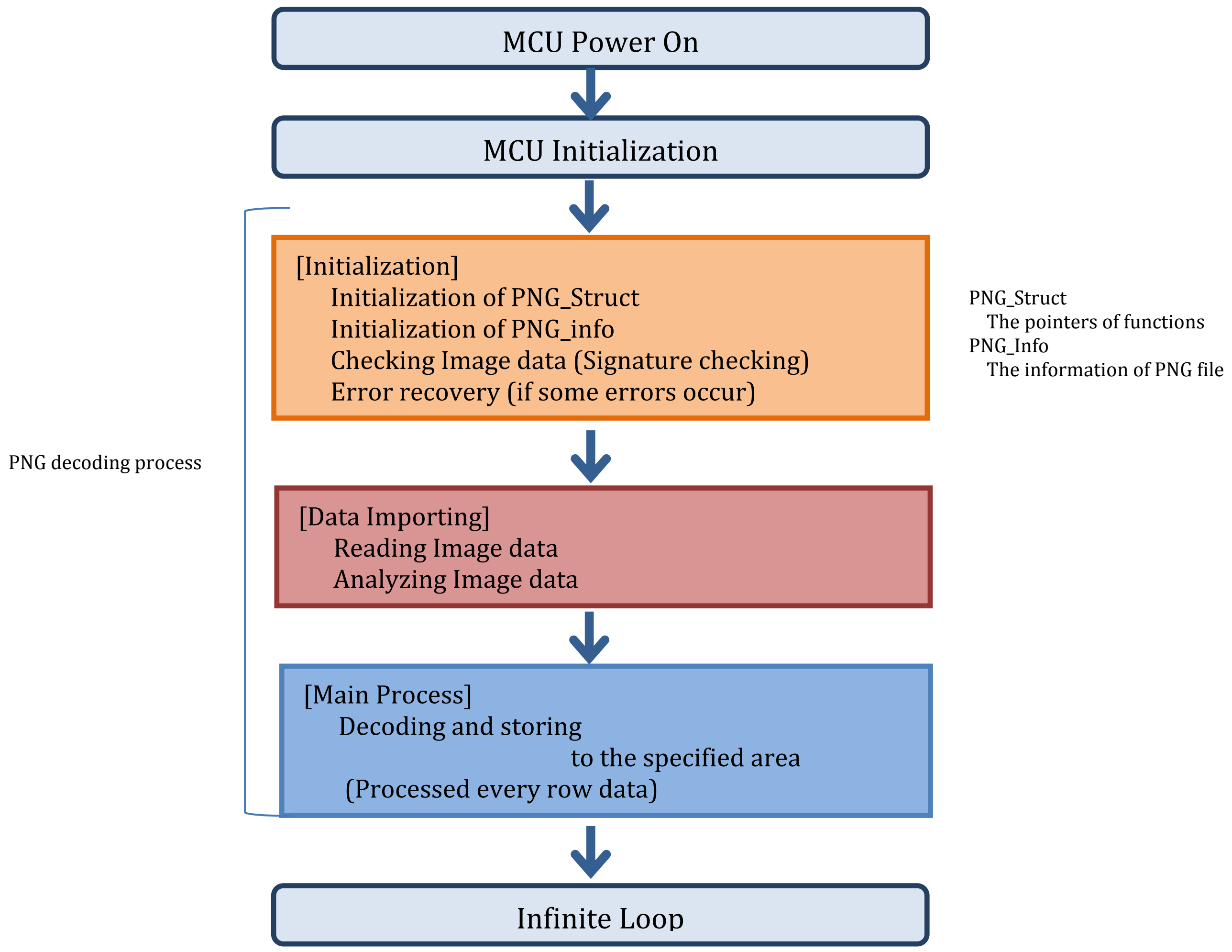
3 Evaluation

3.1 Overview of the software process

The flow of Application software to evaluate the performance for PNG decoding process is shown below. The decoding process is shown in the dark colored blocks in the below flow.

This process uses APIs which are provided by PNG library

Table 1. Software flow


MCU Power On
MCU Initialization
[Initialization]
Initialization of PNG_Struct
Initialization of PNG_info
Checking Image data (Signature checking)
Error recovery (if some errors occur)
[Data Importing]
Reading Image data
Analyzing Image data
[Main Process]
Decoding and storing
to the specified area
(Processed every row data)
Infinite Loop
PNG decoding process
PNG_Struct
The pointers of functions
PNG_Info
The information of PNG file

This evaluation software initializes the work area and the necessary parts for PNG library, and then reads the image data.

After reading data, the software decodes it and stored the data to the specified area.

(In this case it is VRAM area: 0x50000000).

In the actual drawing process, this data becomes input of Graphic Design subsystem of MCU.

3.2 Reference Software

This chapter describes the evaluation software, which we prepared, this time.
For the target, 3 kinds of 256 x256 PNG Image file are evaluated.

3.2.1 Software Environment

For the development, the evaluation, GHS (Green Hills) Multi and GHS probe are used for this evaluation.

IDE	: GHS Multi version : 6.1.4
Compiler version	: Comp2013.5.4

The compiler/Linker option settings are shown below. They are influence to the performance.

```
-preprocess_assembly_files -thumb -align8
-cpu=cortexr5 -littleendian
--long_long
-fhard -C99 -passsource -list
-needprototype -G -v
-uvfd -delete -strip
--no_keep_static_symbols
--no_link_filter -lnk=-v
-Mu -Ml -gsize
-nostartfiles --preprocess_linker_directive_full
-Olink -DRELEASE -pnone --no_coverage_analysis
-Onoinline -Omax -Onoipa -nothumb -Ospeed
```

3.2.2 Base Software

The evaluation software is used

- S6J3200 Series sample program : SampleSW_S6J3200_20150522
- PNG library : libpng1.6.14_zlib_1.2.8

Sample project should be used to enable Free run timer (Ch0), this evaluating software uses Freerun timer to monitor the processing time, using the self-made timer API. (see 32BitTimer.c)

3.2.3 Implementation

The whole of process which is shown in Figure 1, is implemented in main.c with the following code image.

The colored rectangular parts are corresponded to each process shown in the same color in the flow of Figure 1.

Figure 1

```
/****************** START: LOGIC FOR PNG CONVERSION FUNCTION *********************/

/* Registration functions */
png_ptr = png_create_read_struct_2(PNG_LIBPNG_VER_STRING,
                                   NULL, (png_error_ptr)user_error_fn,
                                   NULL,(png_voidp)work_area, NULL,
                                   NULL);
info_ptr = png_create_info_struct(png_ptr);
if (!info_ptr)
{
   /* Error, stop */
}
end_info = png_create_info_struct(png_ptr);
if (!end_info)
{
   /* Error, stop */
}
/* check data */
if( png_sig_cmp(png_address, 0, 8 ) )
{
   /* Error not PNG data, stop */
}
/* for error recovery */
if (setjmp(png_jmpbuf(png_ptr)))
{
   png_destroy_read_struct(&png_ptr, &info_ptr, &end_info);
   /* Error, stop */
}
```

```
/* register read data function */
png_set_read_fn(png_ptr, PngPointer,(png_rw_ptr)user_read_fn);

/* get image information */
png_read_info(png_ptr, info_ptr);
png_get_IHDR(png_ptr, info_ptr, &width, &height, &bit_depth, &color_type,
            &interlace_type, NULL, NULL);

/* Expand paletted colors into true RGB triplets */
if (color_type == PNG_COLOR_TYPE_PALETTE) png_set_palette_to_rgb(png_ptr);

/* Expand paletted or RGB images with transparency to full alpha channels */
if (png_get_valid(png_ptr, info_ptr, PNG_INFO_tRNS)) png_set_tRNS_to_alpha(png_ptr);

/* Flip the RGB pixels to BGR (or RGBA to BGRA) */
if (color_type & PNG_COLOR_MASK_COLOR) png_set_bgr(png_ptr);

/* Swap bytes of 16 bit files to least significant byte first */
png_set_swap(png_ptr);

/* Add filler (or alpha) byte (before/after each RGB triplet) */
png_set_filler(png_ptr, 0xff, PNG_FILLER_AFTER);

/* Turn on interlace handling. */
number_passes = png_set_interlace_handling(png_ptr);

/* Correct gamma and add background */
png_read_update_info(png_ptr, info_ptr);
```

```
/* Main processing loop */
for (pass = 0; pass < number_passes; pass++)
{
   for (counter = 0; counter < height; counter++)
   {
      row_pp = &row_pointers[counter * row_stride];
      png_read_rows(png_ptr, (png_bytepp)(&row_pp), NULL, 1);

   }
}
/* Finish */
png_read_end(png_ptr, info_ptr);
/****************** END: LOGIC FOR PNG CONVERSION FUNCTION **************************/
```

The followings are the points of the implementation for this evaluation.
When you try your own evaluation, please refer them for your modifications.

The first function "png_create_read_struct_2()" is used to make the PNG structure, and it needs to resist the following generic functions, which user has to prepare as user own functions.

user_read_fn	- reads the next N bytes from the PNG array
user_malloc_fn	- function that returns a pointer to a buffer for libpng to use
user_error_callback	- callback function called by library in case of error, can just put an infinite loop in this function
user_error_fn	- function called by library in case of error, can just put an infinite loop in this function
user_free_fn	- called by libpng to return buffer

For these functions, libpng has pre-prepared functions, and they are useful. When user uses theose pre-prepared functions, user can set "NULL" as the parameter of "png_create_read_struct_2()".

But user should need to make "user_read_fn" and "user_error_fn".The pre-prepared function of "user_read_fn" is not for the embedded system. And also, there is not any pre-prepared function for "user_error_fn" but it is necessary for the debugging. The followings are the example of "user_read_fn" and "user_error_fn".

When user makes user_malloc_fn, the heap size has to be taken care. The heap size is defined in Linker Directive file (in the case of GHS Multi). this software is for evaluation, then it does not care to release work_area (heap area), which is allocated by Malloc process. using "user_free_fn", after the main process. But it is necessary to release in the actual programming.

```
void user_read_fn(png_structrp png_ptr, png_bytep data, png_size_t length)
{
        static long i = 0;
        long loop = 0;

        while(loop < length)
        {
                *(data + loop) = *((char*)png_address + i);
                loop++;
                i++;
        }
}
```

```
char user_error_fn(void)
{
		WPREGBIT_GPIO(GPIO_PODR3, 13, 1); // for test
		WPREGBIT_GPIO(GPIO_PODR3, 15, 1); // for test
		while(1);
		return 0;
}

** This function sets IO flag, and process stays here, when libpng APIs fail.
     These ports are configured at the head of main.c
  This port setting is for S6J3260 Series208 pin Evaluation board (S6T3J200261A208A2).
```

Libpng needs some of definitions, for example the pointer of PNG structure, work space for libpng.

Furthermore, some of parameter value (address) should be defined by the user, for example, the address of workspace.

The followings are necessary definitions.

```
static void *PngPointer;                          /* pointer to PNG image data */
uint8_t* row_pointers = (void *)0x50000000;  /* frame buffer pointer */
uint32_t row_stride;                              /* frame buffer stride*/
void * work_area = (void *)0x02000000;            /* work ram pointer(4byte align)*/
size_t work_size = 64*1024;                       /* work ram size(>64kByte)*/
png_structp png_ptr;                              /* Pointer of PNG structure */
png_infop info_ptr;                               /* Pointer of Info structure */
png_infop end_info;                               /* Pointer of Info structure */

png_address = (unsigned char *) _56_hapy_png;     /* _56_hapy_png; is name of table */
```

The values of the following definition are defined by the user.

“row pointers” is the area where the decoded data is stored, it is assigned to VRAM area (from 0x50000000) in this case.

“work_area” is the work space for libpng functions, it is assigned to RAM area (from 0x02000000), in this case.

“work_size” is the size of work_area, and this size is related to Heap size. In this case, heap size is also extended from 0x1000 (the default value of the sample project) to 0x10000 in Linker directive file.

“png_address” should be set the address of PNG Image data table. PNG Image data should be converted to C-header file format previously.

In the test case, user can find the free converter tool (Ex Binary-C converter tool) from website.

And also, the user should define other variables, depending on the own process.

In this case, “row_stride” which is used for main processing loop. was defined to become “Height” x “row_sride” = “total data size”. “row_stride” is consists of the multiplex of “the number of Row data” and”The number of colors” of PNG Image data.

Note: When user changes the target PNG file, the value might be modified depending on target Image data.

Whole of them are defined in main.c.

3.2.4 Other modifications

To implement the libpng, some parts were detected as the linking error. But those parts were not important parts for this evaluation. Then the following includes are changed or commented out.

```
zlib.h      One including header file is changed.   (Line 421)
            /*#    include <types.h>*/      /* for off_t */
            #    include <base_types.h>      /* for off_t */

zutil.c     Unnecessay including file is commented (Line9 - )
            #if 0
            #ifndef Z_SOLO
            #  include "gzguts.h"
            #endif
            #endif
```

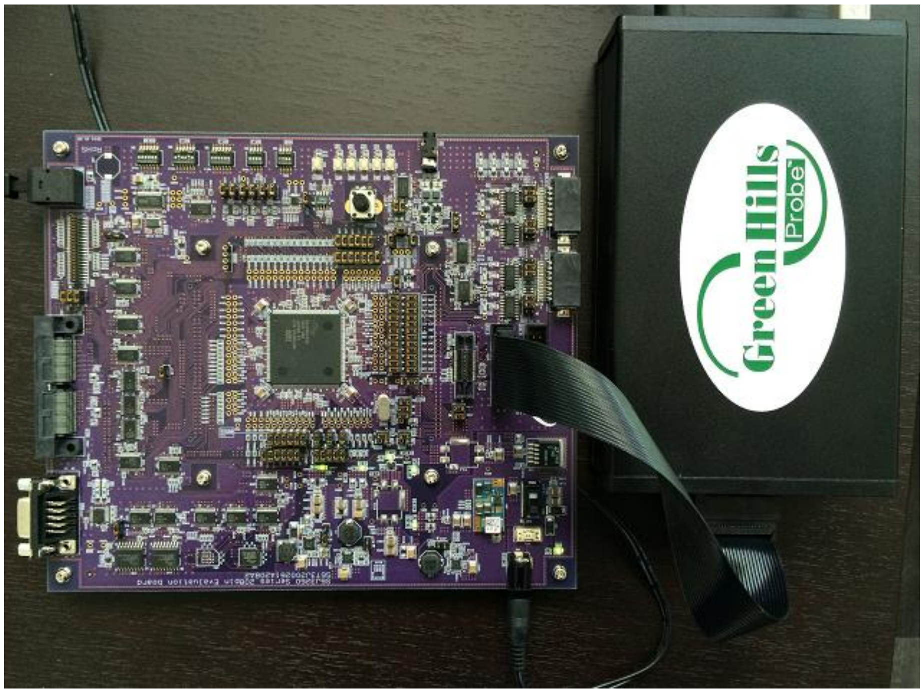
3.3 Hardware Environment

This Evaluation is executed with MCU S6J326CK (208 pin) and the evaluation board

S6T3J200A261A2082.

CPU clock setting is 240 MHz

Figure 2. Evaluation Board and GHS probe



3.3.1 Reference Project

For this evaluation, the following contents are prepared as the reference kit

- Software Project S6J3200_DecompressionPNGType.zip
- PNG Image data (3 Type of 256 x 256 pixel)
- Binary to C converter tool
- bin2c-1.1.zip
- Application Note (This document)

4 Evaluation Result

For the evaluation, we used 3 kinds of 256 x256 pixel PNG files, and target processing time was about 50msec for the smallest PNG Image. This target was achieved from our evaluation as it is shown in the following table.

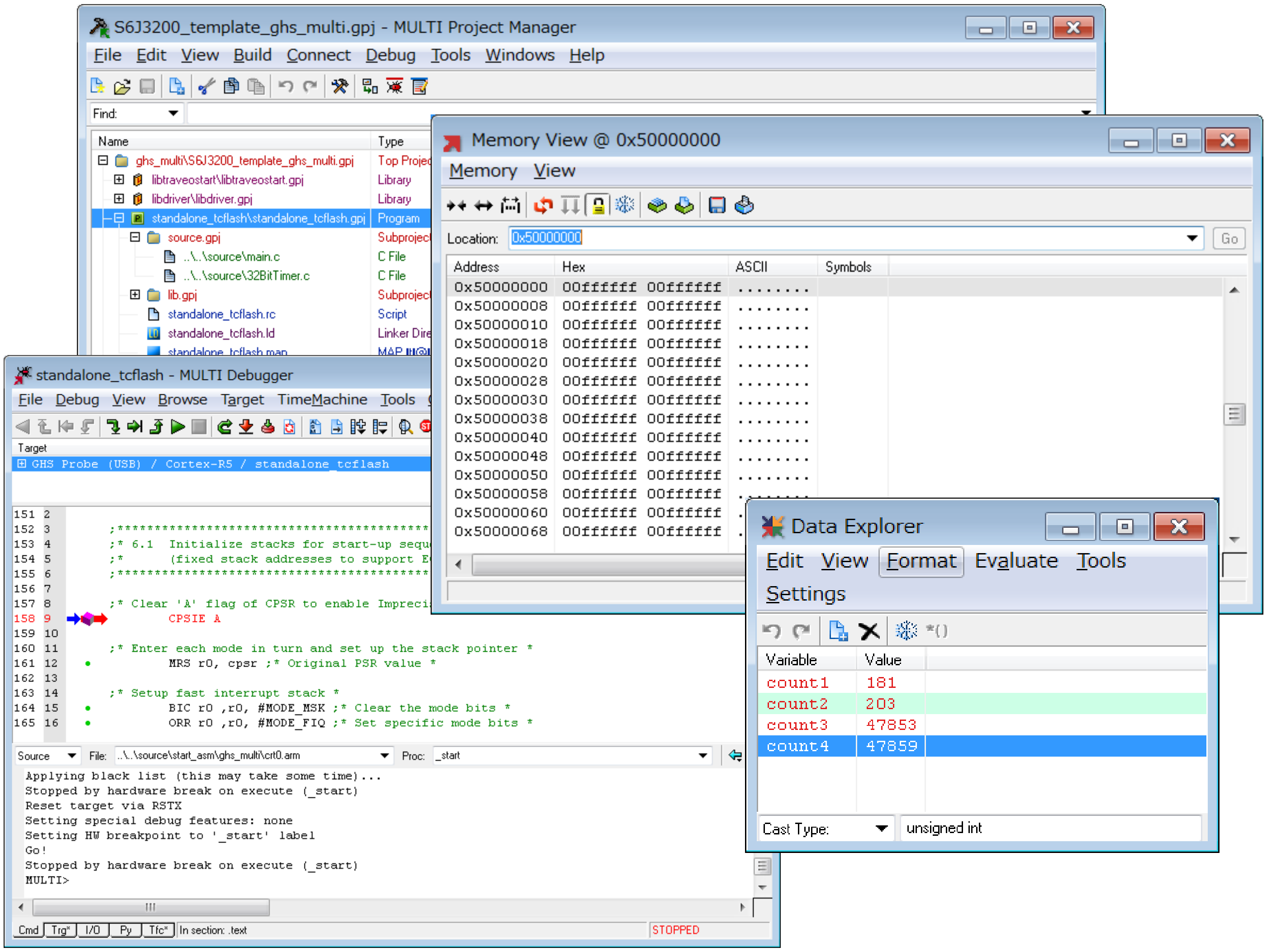
This data was evaluated with CPU clock 240MHz.

Table 2. Evaluation Data

	Image 1 256_hapy.png (60kB)		Image 2 256x256_album1.png (87kB)		Image 2 256x256_album1.png (117kB)	
Check Pint	Counter	Time (msec)	Counter	Time (msec)	Counter	Time (msec)
Initialization / Registration	191	0.20246	191	0.20246	191	0.20246
Reading Image Info	23	0.02438	22	0.02332	22	0.02332
Decode and Save Image in VRAM	47293	50.13058	54715	57.9979	69027	73.16862
Total Time	**48144**	**51.03264**	**54934**	**58.23004**	**69247**	**73.40182**

And also, we checked the following points to confirm that this performance works collect.

1. Error function (This case"user_error_fn") has never been called
2. Decoded PNG data is copied to VRAM area (0x5000_0000) with the memory window of Debugger


S6J3200_template_ghs_multi.gpj - MULTI Project Manager
File Edit View Build Connect Debug Tools Windows Help
Find:
Memory View @ 0x50000000
Memory View
Location: 0x50000000
Go
standalone_tcflash - MULTI Debugger
File Debug View Browse Target TimeMachine Tools
Target
Data Explorer
Edit View Format Evaluate Tools
Settings

Name	Type
ghs_multi\S6J3200_template_ghs_multi.gpj	Top Projec
libtraveostart\libtraveostart.gpj	Library
libdriver\libdriver.gpj	Library
standalone_tcflash\standalone_tcflash.gpj	Program
source.gpj	Subprojec
..\..\source\main.c	C File
..\..\source\32BitTimer.c	C File
lib.gpj	Subprojec
standalone_tcflash.rc	Script
standalone_tcflash.ld	Linker Dire
standalone_tcflash.map	MAP

```
Address     Hex                ASCII     Symbols
0x50000000  00ffffff 00ffffff  ........
0x50000008  00ffffff 00ffffff  ........
0x50000010  00ffffff 00ffffff  ........
0x50000018  00ffffff 00ffffff  ........
0x50000020  00ffffff 00ffffff  ........
0x50000028  00ffffff 00ffffff  ........
0x50000030  00ffffff 00ffffff  ........
0x50000038  00ffffff 00ffffff  ........
0x50000040  00ffffff 00ffffff  ........
0x50000048  00ffffff 00ffffff  ........
0x50000050  00ffffff 00ffffff  ........
0x50000058  00ffffff 00ffffff  ........
0x50000060  00ffffff 00ffffff  .
0x50000068  00ffffff 00ffffff  .
```

```
GHS Probe (USB) / Cortex-R5 / standalone_tcflash
```

```

        ;*******************************************
        ;* 6.1  Initialize stacks for start-up sequ
        ;*      (fixed stack addresses to support E
        ;*******************************************

        ;* Clear 'A' flag of CPSR to enable Impreci
                CPSIE A

        ;* Enter each mode in turn and set up the stack pointer *
                MRS r0, cpsr ;* Original PSR value *

        ;* Setup fast interrupt stack *
                BIC r0 ,r0, #MODE_MSK ;* Clear the mode bits *
                ORR r0 ,r0, #MODE_FIQ ;* Set specific mode bits *
```

Source File: ..\..\source\start_asm\ghs_multi\crt0.arm Proc: _start

```
Applying black list (this may take some time)...
Stopped by hardware break on execute (_start)
Reset target via RSTX
Setting special debug features: none
Setting HW breakpoint to '_start' label
Go!
Stopped by hardware break on execute (_start)
MULTI>
```

Cmd Trg* I/O Py Tfc* In section: .text STOPPED

Variable	Value
count1	181
count2	203
count3	47853
count4	47859

Cast Type: unsigned int

5 Reference

1. Libpng Website : http://www.libpng.org/pub/png/libpng.html
2. Libpng Manual : http://www.libpng.org/pub/png/libpng-1.4.0-manual.pdf
3. 32-BIT MICROCONTROLLER Spansion Traveo Family S6J3200 series HARDWARE MANUAL
4. MN708-00006-1v0-E_dn.pdf

Document History

Document Title: AN204459 - S6J3200 Series Evaluation for Png Image Decoding

Document Number: 002-04459

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	—	KHAS	07/22/2015	Initial release
*A	5058663	KHAS	02/15/2016	Converted Spansion Application Note “S6J3200_AN708-00016” to Cypress format
*B	5876104	AESATMP8	09/07/2017	Updated logo and Copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709