AN-1050
应用笔记

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

ADV74xx/ADV78xx压缩I²C脚本的方法

作者：Witold Kaczurba

简介

本应用笔记介绍一种用于压缩微控制器平台所用的大量I²C脚本的方法。文中信息所针对的应用是用户需要将50个以上脚本放入单个微控制器的存储器。对于包含200个以上脚本，用于一条I²C总线上的6个以上器件的脚本集，本方法可提供出色的结果。

利用ADI公司的多格式解码器，用户可以解码各种各样的视频标准。由于支持多种标准，因此这些视频解码器提供许多不同的设置，每种设置均包含针对各种模式的批量I²C写操作。这些写操作的集合就是脚本。

某些情况下，用户可能使用数百个脚本来配置许多I²C器件。为了保存所有这些脚本，用户需要微控制器提供大量存储空间。请注意，这些脚本除了具有一些独特的写操作之外，还可能具有一些相似的写操作。本应用笔记将详细说明如何在PC端压缩脚本，以及如何在微控制器端编写有效的解压缩方法。使用解压缩算法效率越高，则所需的RAM越少。

本应用笔记包括一个Octave程序的脚本。Octave是用于数值计算的免费（GNU/GPL许可）计算机程序。该Octave脚本可以压缩脚本，并将结果导出为含解压缩程序的C代码。所产生的C代码易于使用，可方便地移植到任何微控制器。

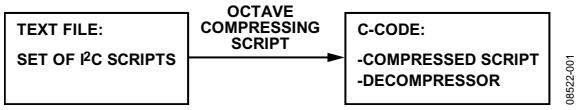

图1. 压缩算法的原理

目录

关于脚本

基础知识

本应用笔记所介绍的压缩方法旨在用于按照“脚本结构”章节规定而构建的脚本集。一个I²C写操作由三个值组成：

- 器件地址(用于寻址I²C总线上的器件)
- 寄存器地址
- 需写入寄存器的值

ADI公司随评估板提供的脚本通常将I²C写操作按升序排列，也就是对同一器件的写操作按照寄存器地址大小升序排列。例如：

- 42 00 AB
- 42 01 CD
- 42 02 EF

这并非偶然，写操作按顺序排列将使得在不同脚本中更有可能找到相同的规律，从而使压缩更有效。请注意，对同一器件的同一寄存器执行同样的写操作是不太可能的，例如：

- 42 03 04
- 42 03 08

这就提高了不同脚本中出现相同写操作序列的概率。本应用笔记所描述的算法之所以检查脚本是否含有相同的四个写操作序列(如图2所示)，其原因正在于此。

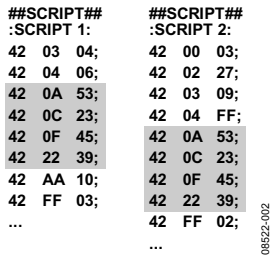

图2. 两个不同脚本中出现相同的四个写操作序列

存储许多不同脚本模块共有的大量序列(由四个写操作组成)对微控制器很有利。在C程序中，这些相同的模块(也称为“键”)可以存储为一个连续阵列。这样，每个键都很容易进行寻址，解压缩算法可以省去许多指针。要知道，每个指针常量都需要占用内存。

脚本结构

用于压缩的Octave脚本要求原始脚本集以特殊方式存储在一个文件中。每个原始脚本均以一个表头(第一行)开始，表头的起始和结束部分均为字符##。下一行是一个小表头，必须以冒号起始和结束。接下来的第三行包含适当的I²C写操作，例如：

42 05 02 ; Prim_Mode = 010b for GR

其中：

42表示器件的8位I²C地址(0x42)。

05表示8位寄存器地址(0x05)。

02表示8位值(0x02)。

Prim_Mode = 010b for GR是用户注释，为可选项

请注意表达式中的空格和分号。脚本最后一行是一个单词End，没有任何空格。脚本之间用空行分开。

作为例子，下面几页显示了ADV7401评估板(EVAL-ADV7401EBZ)的脚本集的一部分。

```
##CP VGA 640x480##
:640x480 _@ 60 Autodetecting sync source  25.175 MHz out through DAC:
42 05 02 ; Prim_Mode = 010b for GR
42 06 08 ; VID_STD = 1000b for 640 × 480 @ 60
42 1D 47 ; Enable 28 MHz crystal
42 3A 11 ; Set latch clock settings to 001b, Power down ADC3
42 3B 80 ; Enable external bias
42 3C 5C ; PLL_QPUMP to 100b
42 6A 00 ; DLL phase adjust
42 6B 82 ; Enable DE output, swap Pr& Pb
42 73 90 ; Set man_gain
42 7B 1D ; TURN OFF EAV & SAV CODES Set BLANK_RGB_SEL
42 85 03 ; Enable DS_OUT
42 86 0B ; Enable stdi_line_count_mode
42 8A 90 ; VCO range to 00b
42 F4 3F ; Max drive strength
42 0E 80 ; Analog Devices recommended setting
42 52 46 ; Analog Devices recommended setting
42 54 00 ; Analog Devices recommended setting
42 0E 00 ; Analog Devices recommended setting
54 00 13 ; Power-down encoder
74 EE EE ; Power-down HDMI
End

##CP VGA 640x480##
:640x480 _@ 72 Autodetecting sync source  31.5 MHz out through DAC:
42 05 02 ; Prim_Mode = 010b for GR
42 06 09 ; VID_STD = 1001b for 640 × 480 @ 72
42 1D 47 ; Enable 28 MHz crystal
42 3A 11 ; set latch clock settings to 001b, Power down ADC3
42 3B 80 ; Enable external bias
42 3C 5C ; PLL_QPUMP to 100b
42 6A 00 ; DLL phase adjust
42 6B 82 ; Enable DE output, swap Pr& Pb
42 73 90 ; Set man_gain
42 7B 1D ; TURN OFF EAV & SAV CODES Set BLANK_RGB_SEL
42 85 03 ; Enable DS_OUT
42 86 0B ; Enable stdi_line_count_mode
42 F4 3F ; Max drive strength
42 0E 80 ; Analog Devices recommended setting
42 52 46 ; Analog Devices recommended setting
42 54 00 ; Analog Devices recommended setting
42 0E 00 ; Analog Devices recommended setting
54 00 13 ; Power down encoder
74 EE EE ; Power down HDMI
End
```

```
##CP VGA 640x480##
:640x480 _@ 75  Autodetecting sync source  31.5 MHz Out through DAC:
42 05 02 ; Prim_Mode = 010b for GR
42 06 0A ; VID_STD =1 010b for 640 × 480 @ 75
42 1D 47 ; Enable 28 MHz crystal
42 3A 11 ; set latch clock settings to 001b, Power down ADC3
42 3B 80 ; Enable external bias
42 3C 5C ; PLL_QPUMP to 100b
42 6A 00 ; DLL phase adjust
42 6B 82 ; Enable DE output, swap Pr& Pb
42 73 90 ; Set man_gain
42 7B 1D ; TURN OFF EAV & SAV CODES Set BLANK_RGB_SEL
42 85 03 ; Enable DS_OUT
42 86 0B ; Enable stdi_line_count_mode
42 F4 3F ; Max drive strength
42 0E 80 ; Analog Devices recommended setting
42 52 46 ; Analog Devices recommended setting
42 54 00 ; Analog Devices recommended setting
42 0E 00 ; Analog Devices recommended setting
54 00 13 ; Power down encoder
74 EE EE ; Power down HDMI
End
```

请注意，脚本中使用的器件地址始终是一个大于0的偶数。奇数地址用于回读器件，这是不会出现的。因此，地址0x00用作转义码，一种用于解压缩的特殊代码。

压缩和解压缩

“Octave源代码”中的Octave脚本在PC端压缩脚本。Octave脚本输出包含压缩数据和解压缩算法的C代码。

Octave脚本算法由下列步骤组成：

1. 将脚本载入不同维数的矩阵(大小取决于脚本的长度)。
2. 搜索矩阵，寻找相同模块或键，其中一个键由四个写操作组成。
3. 如果找到一个相同模块，则将其写入一个单独的块，称为键，具有key_number索引。
4. 只要有一个相同模块，就用单一写操作(0x00, A, B)代替该块，其中A × 256 + B等于值key_number。
5. 将key_number索引递增1。
6. 重复执行寻找和替换相同模块的步骤，直到一次自始至终的搜索没有获得任何结果为止。

此过程创建的矩阵结构如图3所示。

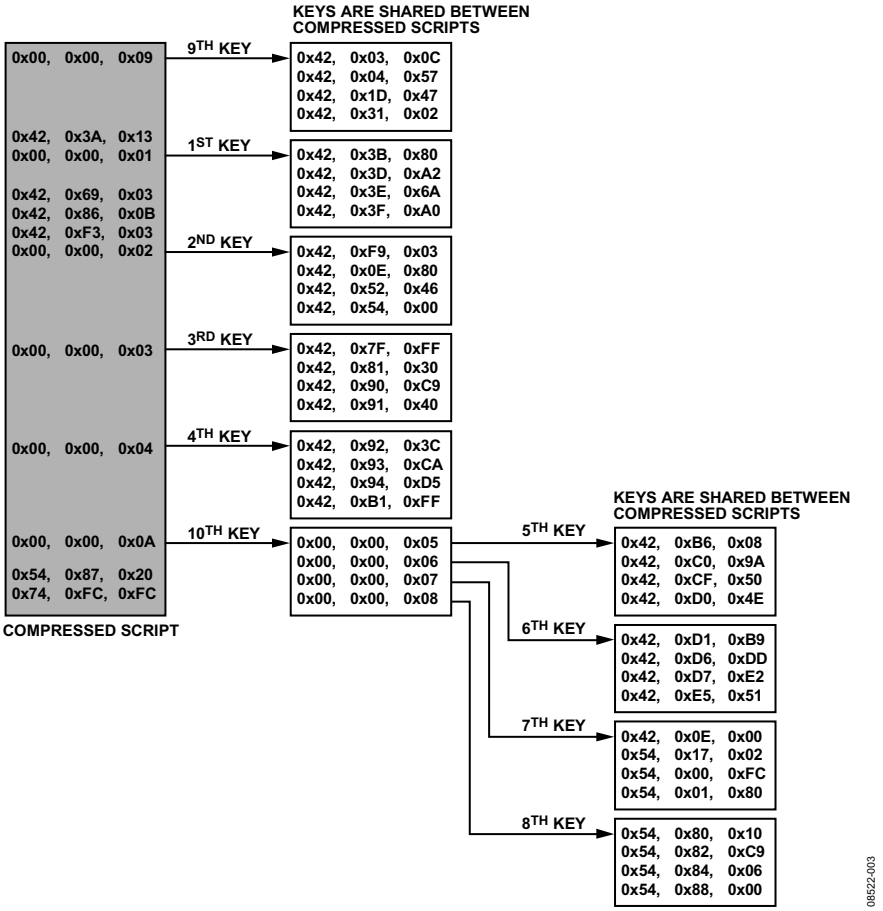

图3. 压缩脚本的结构

一旦完成矩阵格式化，并确定相同键之后，就可以开始创建C代码，并执行少量的后优化处理(后压缩)。后压缩处理将执行下列步骤：

- 写入脚本代码的结尾表示(0xFF)，以便解压缩程序能够找到脚本的结尾。
- 判断key_number是否小于256。如果是，其表示代码(0x00, A, B)可以缩短为(0x00, B)，因为A始终为0。
- 将解压缩程序的C代码添加到该C代码的末尾。

最后生成的C代码如图4所示。

使用

用户可以使用“Octave源代码”部分所提供的代码。它由6个文件组成，这些文件应位于同一个文件夹。将脚本集文件script.txtfile放入此文件夹，该文件的格式见下一部分的说明。运行该脚本后，创建output.c文件，它包含用于解压缩的压缩脚本算法和主代码的示例。

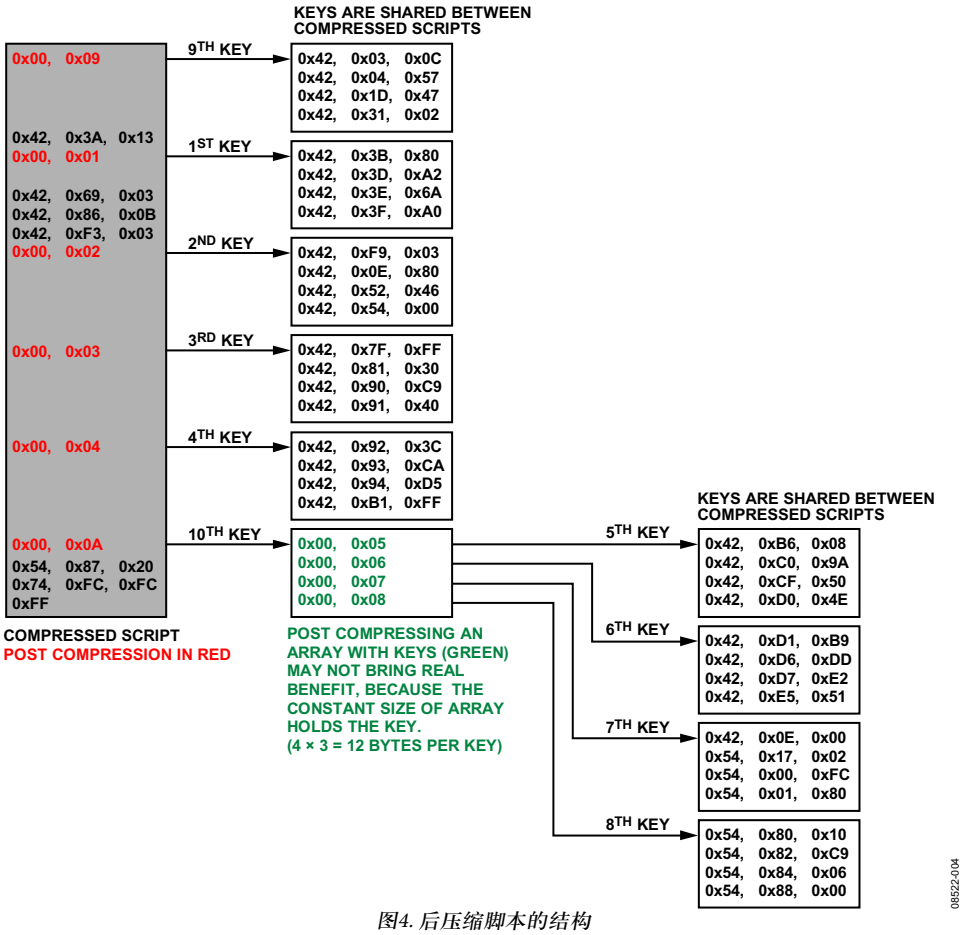

图4. 后压缩脚本的结构

OCTAVE SOURCE CODE

All listings should be saved to one folder together with the script.txt file containing the script to be compressed. This code was tested using Octave Version 3.0.3.

MAIN.M

```
clear;

filename = 'script.txt';

%%%%%% loading the script:  %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  number_of_scripts = number_of_scripts_in_file(filename);

  %%% load all scripts - script by script  %%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  for i=1:number_of_scripts
    cmd = sprintf("[script_%d] = load_script(\"%s\", %d);", i, filename, i);
    eval(cmd);
  endfor
  printf("Scripts loaded...\r\n");
  fflush(stdout);

  %%%%%%% find same occurances in different scripts %%%%%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  NUMBER_OF_LINES = 4;
  key_number = 0;

  do % master loop - runs the optimization until no improvement is done

    % global_found is a variable to show that any improvement was done in full run
    global_found = 0;

    for i=1:number_of_scripts
      i                % print the iteration
      fflush(stdout);  % update user's screen

      %% load variable current_script with matrix: %%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      eval(sprintf("current_script_a = script_%d;", i));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% check if the script is empty or not: %%
dim = size(current_script_a);
if ((dim==[0,0]) || (dim(1) < NUMBER_OF_LINES))
  found = 0;
  continue;
endif
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[dim1, dim2] = size(current_script_a);  % check get size
start_line = 0;

%%%% looking for the same pattern across matrix %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while (start_line + NUMBER_OF_LINES - 1 < dim1)
  start_line++;  % move the pointer
  found = 0;     % this pattern (KEY) is not found yet
  key_added = 0; % KEY: variable for maintaining keys (KEY not FOUND - so not added yet)

  % get part of matrix as a pattern to find in other scripts: %
  part_of_script_a = current_script_a(start_line:start_line+NUMBER_OF_LINES-1, :);

  %%%% search in other scripts for pattern %%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  for j=i+1:number_of_scripts

    eval(sprintf("current_script_b = script_%d;", j));

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% check if the script is empty or not: %%
    dim = size(current_script_b);
    if ((dim==[0,0]) || (dim(1) < NUMBER_OF_LINES))
      continue;
    endif
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    val = find_mat2(current_script_b, part_of_script_a);
    if (val != 0)

      %% AS PATTERN WAS FOUND IN THE OTHER SCRIPT - %
      %%%%%%%  - ADDING THE KEY TO KEY LIST: %%%%%%%%
      if (key_added == 0)
        key_added = 1; % KEY_ADDED
        key(++key_number, :, :) = part_of_script_a';
        this_key = [0, floor(key_number / 256), mod(key_number, 256)]; % MAX: 65535 KEYS!
```

```
            current_script_a = fsubst(current_script_a, this_key, start_line, \
                                  start_line+NUMBER_OF_LINES-1);

            eval(sprintf("script_%d = current_script_a;", i));
            [dim1, dim2] = size(current_script_a);
          end

          %% MODIFICATION OF THE SCRIPT %%
          current_script_b = fsubst(current_script_b, this_key, val, val+NUMBER_OF_LINES-1);
          eval(sprintf("script_%d = current_script_b;", j));

          %% ADDING COUNTER OF FOUND LINES
          found = found + 1;
          global_found = 1;
        endif
      endfor
    endwhile
  endfor
until (!global_found)

%%% export file to c-file %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
script_to_c;
```

LOAD_SCRIPT.M

```
function [matrix] = load_script(filename, script_number)
  % script_number - number of script (starts from 1) to get
  current_script_number = 0;
  matrix = [];

  if ((fhandle = fopen(filename, "r")) == -1)
    return;

  else
    started_script_line = 0;

    while (!feof(fhandle))
      text_line = fgetl(fhandle);

      if (strncmpi(text_line, "end", 3))
        started_script_line = 0;

        if (current_script_number == script_number)
          return; % needed script has been extracted
        endif
```

```
      elseif (started_script_line > 0)
            started_script_line = started_script_line + 1;
      endif

      %%% Is it first line of new script?: %%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      if ((length(text_line) > 1) && (text_line(1) == "#") && (text_line(2) == "#"))

        started_script_line = 1;
        current_script_number = current_script_number + 1;
      endif

      %%% Is it next line of the same script?: %%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      if ((started_script_line > 2) && (script_number == current_script_number))
        new_line = (sscanf(text_line, "%x %x %x"))'; % don't forget about '

        if (started_script_line == 3)
          matrix = new_line;       % creates matrix
        else
          matrix = [matrix; new_line];  % appends to already created matrix
        endif
      endif


    endwhile

    fclose (fhandle);
    return;


  endif
endfunction
```

NUMBER_OF_SCRIPTS_IN_FILE.M

```
function [number_of_scripts] = number_of_scripts_in_file(filename)

  if ((fhandle = fopen(filename, "r")) == -1)
    number_of_scripts = -1;
    fprintf(stdout, "Error while opening file!");
    return;
  else
    number_of_scripts = 0;

    while (!feof(fhandle))
```

```
      text_line = fgetl(fhandle);

      if (length(text_line > 1) && (text_line(1) == "#") && (text_line(2) == "#"))
        number_of_scripts = number_of_scripts + 1;
      endif
    endwhile

    fclose (fhandle);
    return;
  endif
endfunction
```

FSUBST.M

```
function [ret]=fsubst(matrix, sub, first_line, last_line)

  last_line = last_line + 1;
  [dim_a1, dim_a2] = size(matrix);

  ret = matrix(1:first_line-1, :);
  ret = [ret; sub];
  ret = [ret; matrix(last_line:dim_a1, :)];

endfunction
```

FIND_MAT2.M

```
function [res] = find_mat2(mat_a, mat_b)
%find_mat2 returns the first line where mat_b is located in mat_a

occurances_found = 0;

[dim_a1, dim_a2] = size(mat_a);
[dim_b1, dim_b2] = size(mat_b);

  res = 0;
% dim_a2
% dim_b2

  mat_a_expanded = mat_a'(:);
  mat_b_expanded = mat_b'(:);

len_b = length(mat_b_expanded);

% which one is bigger?

for (i=1:dim_a2:(dim_a1*dim_a2-dim_b1*dim_b2+1))

  difference = mat_a_expanded(i:i+len_b-1) - mat_b_expanded;
```

```
  % count number of zero columns
  number_of_matches = length(find(difference == 0));

  if (number_of_matches == len_b)
    %fprintf(stdout, "Offset = %d\r\n", i);
    res(++occurances_found) = ceil(i/dim_a2);
  endif

endfor


endfunction
```

SCRIPT_TO_C.M

```
  filename_output = "output.c"
  termination_characters = "        0xFF }; \r\n\r\n";

  %%% opening file to save %%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%
  if ((fhandle = fopen(filename_output, "w")) == -1)
    return;
  endif

  %%% SIZE OF KEY %%%
  %%%%%%%%%%%%%%%%%%%
  if (key_number > 0)
    [key_dim_a, key_dim_b, key_dim_c] = size(key(1, :, :));
    size_of_key = key_dim_a * key_dim_b * key_dim_c;
  else
    size_of_key = 0;
  end

  %%/* DEVCPP - INCLUDES */       %%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  fprintf(fhandle, "#include <cstdlib>\r\n");
  fprintf(fhandle, "#include <iostream>\r\n");
  fprintf(fhandle, "using namespace std;\r\n");


  fprintf(fhandle, "#define DECOMPR_SIZE_OF_KEY    %d\r\n", size_of_key);
  fprintf(fhandle, "#define DECOMPR_NUMBER_OF_SCRIPTS %d\r\n", number_of_scripts);
  fprintf(fhandle, "#define DECOMPR_NUMBER_OF_KEYS    %d\r\n\r\n", key_number);

  fprintf(fhandle, "#define DECOMPR_ADDRESS_OR_ESCAPE   0\r\n");
  fprintf(fhandle, "#define DECOMPR_REGADDR             1\r\n");
  fprintf(fhandle, "#define DECOMPR_VALUE               2\r\n\r\n");
```

```
fprintf(fhandle, "#define DECOMPR_ESCAPE_CODE         0\r\n");
fprintf(fhandle, "#define DECOMPR_TERMINATE_VALUE     0xFF\r\n");
fprintf(fhandle, "#define DECOMPR_USES_2_BYTE_CODES   (DECOMPR_NUMBER_OF_KEYS > 254)\r\n");

for (i=1:number_of_scripts)
  eval(sprintf("current_script = script_%d;", i));

  fprintf(fhandle, "const unsigned char script_%d[] = { \r\n", i);
  [dim1, dim2] = size(current_script);

  for (line = 1:dim1)
    %% POST COMPRESSION / OPTIMIZATION: %%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if ( (current_script(line, 1) == 0) && (key_number < 255) )
      fprintf(fhandle, "        0x%.2X, 0x%.2X,\r\n", \
              current_script(line, 1),                \
              current_script(line, 3));

    else
      fprintf(fhandle, "        0x%.2X, 0x%.2X, 0x%.2X,\r\n",
              current_script(line, 1),                \
              current_script(line, 2),                \
              current_script(line, 3));               \

    endif
  endfor

  fprintf(fhandle, termination_characters);
endfor

%%% LISTING THE KEYS  %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%
if (key_number > 0)
  fprintf(fhandle,                                     \
          "const unsigned char keys[DECOMPR_NUMBER_OF_KEYS][DECOMPR_SIZE_OF_KEY] = { \r\n");

endif
for (n=1:key_number)

  %%%% listing the key   %%%%
  %%%%%%%%%%%%%%%%%%%%%%%%%%
  if ((key_number < 255) && (key(n, 1, 1) == 0))
    fprintf(fhandle, "        { 0x%.2X, 0x%.2X, \r\n", \
            key(n, 1, 1),                              \
            key(n, 3, 1));
```

```
  else
    fprintf(fhandle, "{ 0x%.2X, 0x%.2X, 0x%.2X, \r\n",  \
            key(n, 1, 1),                               \
            key(n, 2, 1),                               \
            key(n, 3, 1));

  endif

  for line=2:key_dim_c-1
    if ((key_number < 255) && (key(n, 1, line) == 0))
      fprintf(fhandle, "          0x%.2X, 0x%.2X, \r\n", \
              key(n, 1, line),                           \
              key(n, 3, line));
    else
      fprintf(fhandle, "          0x%.2X, 0x%.2X, 0x%.2X, \r\n", \
              key(n, 1, line),                                    \
              key(n, 2, line),                                    \
              key(n, 3, line));
    endif
  endfor

  if ((key_number < 255) && (key(n, 1, key_dim_c) == 0))
    fprintf(fhandle, "          0x%.2X, 0x%.2X }", \
            key(n, 1, key_dim_c),                  \
            key(n, 3, key_dim_c));
  else
    fprintf(fhandle, "          0x%.2X, 0x%.2X, 0x%.2X }", \
            key(n, 1, key_dim_c),                          \
            key(n, 2, key_dim_c),                          \
            key(n, 3, key_dim_c));
  endif

  if (n != key_number) % this is not the last key:
    fprintf(fhandle, ",\r\n",                          \
            key(n, line, 1),                           \
            key(n, line, 1),                           \
            key(n, line, 1));
  else
    fprintf(fhandle, "};\r\n\r\n",
            key(n, line, 1),                           \
            key(n, line, 1),                           \
            key(n, line, 1));
  endif
endfor


%%% LIST OF POINTERS TO SCRIPTS %%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(fhandle,"const unsigned char *script_list[DECOMPR_NUMBER_OF_SCRIPTS] = {\r\n");

for (i=1:number_of_scripts-1)
  fprintf(fhandle, sprintf("      script_%d,\r\n", i));
endfor
i++;

fprintf(fhandle, sprintf("      script_%d\r\n", i)); % last script
fprintf(fhandle,"};\r\n\r\n");
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% EXAMPLE SEND_I2c_COMMAND: %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(fhandle, "void send_i2c_command(unsigned char dev_addr, ");
fprintf(fhandle, "unsigned char reg_addr, unsigned char value) {\r\n");

fprintf(fhandle, "     printf(\"%%.2X %%.2X %%.2X ;\\r\\n\", dev_addr, reg_addr, value);\r\n");
fprintf(fhandle, "     return;\r\n");
fprintf(fhandle, "}\r\n\r\n");

%% DECOMPRESSING ALGORITHM: %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf(fhandle, "void decompress(const unsigned char *scr_addr, unsigned char ");
fprintf(fhandle, "is_key=0)\r\n{\r\n");

fprintf(fhandle, "     unsigned char dev_addr, reg_addr, value; // information ");
fprintf(fhandle, "that has to be passed to I2C function\r\n\r\n");

fprintf(fhandle, "     unsigned int char_type = DECOMPR_ADDRESS_OR_ESCAPE; ");
fprintf(fhandle, " // state machine var.\r\n");

fprintf(fhandle, "     unsigned char *script = (unsigned char *) scr_addr; // pointer\r\n");
fprintf(fhandle, "     unsigned int line_counter = 0; // used to terminate when ");
fprintf(fhandle, "processing of keys (as they do not have END sequence as 0xFF;\r\n\r\n");

fprintf(fhandle, "     do {\r\n");
fprintf(fhandle, "         switch (char_type) {\r\n");
fprintf(fhandle, "                case DECOMPR_ADDRESS_OR_ESCAPE: {\r\n");
fprintf(fhandle, "                       if ((*script == DECOMPR_TERMINATE_VALUE))\r\n");
fprintf(fhandle, "                          return;\r\n\r\n");

if (key_number > 0)
  fprintf(fhandle, "                        if ((*script) == DECOMPR_ESCAPE_CODE) {\r\n\r\n");
  fprintf(fhandle, "                           // two versions of code ");
  fprintf(fhandle, "(below) requires different offset of pointer:\r\n");
```

```
  fprintf(fhandle, "                          //    (0x00, A, B = 16-bit)\r\n");
  fprintf(fhandle, "                          //    (0x00, B = 8-bit code)\r\n");
  fprintf(fhandle, "                          if (DECOMPR_USES_2_BYTE_CODES) {\r\n");
  fprintf(fhandle, "                             decompress(keys[(*(script+1))*256 + ");
  fprintf(fhandle, " (*(script+2))-1], 1);\r\n");

  fprintf(fhandle, "                             script+=3; // shift the pointer\r\n");
  fprintf(fhandle, "                          } else {\r\n");
  fprintf(fhandle, "                             decompress(keys[(*(script+1))-1], 1);\r\n");
  fprintf(fhandle, "                             script+=2; // shift the pointer\r\n");
  fprintf(fhandle, "                          }\r\n\r\n");
  fprintf(fhandle, "                          line_counter++;\r\n");
  fprintf(fhandle, "                       } else {\r\n");
endif

fprintf(fhandle, "                          dev_addr = *script;\r\n");
fprintf(fhandle, "                          script++;\r\n");
fprintf(fhandle, "                          char_type = DECOMPR_REGADDR;\r\n");

if (key_number > 0)
  fprintf(fhandle, "                       }\r\n");
endif

fprintf(fhandle, "                       break;\r\n");
fprintf(fhandle, "                    }\r\n");
fprintf(fhandle, "                case DECOMPR_REGADDR: {\r\n");
fprintf(fhandle, "                          reg_addr = *script;\r\n");
fprintf(fhandle, "                          script++;\r\n");
fprintf(fhandle, "                          char_type = DECOMPR_VALUE;\r\n");
fprintf(fhandle, "                          break;\r\n");
fprintf(fhandle, "                    }\r\n\r\n");
fprintf(fhandle, "                case DECOMPR_VALUE: {\r\n");
fprintf(fhandle, "                          value = *script;\r\n");
fprintf(fhandle, "                          script++;\r\n");
fprintf(fhandle, "                          char_type = DECOMPR_ADDRESS_OR_ESCAPE;\r\n");
fprintf(fhandle, "                          \r\n");
fprintf(fhandle, "                          // all data should be ready to send:\r\n");
fprintf(fhandle, "                          send_i2c_command(dev_addr, reg_addr, \r\n");
fprintf(fhandle, "                                           value);\r\n");
fprintf(fhandle, "                          line_counter++;\r\n");
fprintf(fhandle, "                          break;\r\n");
fprintf(fhandle, "                    }\r\n\r\n");
fprintf(fhandle, "                default: break;\r\n");
fprintf(fhandle, "             }\r\n");
fprintf(fhandle, "     } while (!(is_key && line_counter==4));\r\n}\r\n\r\n");

%% EXAMPLE MAIN %%
```

```
%%%%%%%%%%%%%%%%%%
fprintf(fhandle, "int main(int argc, char *argv[])\r\n{\r\n");
fprintf(fhandle, "    int i;\r\n\r\n");
fprintf(fhandle, "    for (i=0; i<DECOMPR_NUMBER_OF_SCRIPTS ; i++) {\r\n");
fprintf(fhandle, "        printf(\"## Scripts ##\\n:Script no.%%d:\\n\", i);\r\n");
fprintf(fhandle, "        decompress(script_list[i]);\r\n");
fprintf(fhandle, "        printf(\"End\\n\\n\");\r\n");
fprintf(fhandle, "    }\r\n\r\n");
fprintf(fhandle, "    system(\"PAUSE\");\r\n");
fprintf(fhandle, "    return EXIT_SUCCESS;\r\n}\r\n");


fclose(fhandle);
```

结论

结果

本应用笔记所讨论的压缩算法实现的结果如下：

- 对于ADV7401评估板所用的51个脚本，压缩比为1.858(4748字节压缩至2556字节)。
- 对于ADV7840评估板所用的251个脚本，压缩比为5.056(52,088字节压缩至10,268字节)。

压缩比使用编译程序的ROM大小进行计算。

测试使用了KEIL编译器和ADI公司ADuC7024微控制器。

进一步优化

用户可能希望对给定的算法进行优化。下面介绍一些优化技巧。

通常而言，非常大的脚本使用200到700个键。因此，可以不用三个字节来解码地址(0x00,A,B)，而是简单地用ESCAPE码来代替，从而实现优化：

- 0x00, A表示(0 + A)之间的键的地址
- 0x01, A表示(256 + A)之间的键的地址
- 0x03, A表示(512 + A)之间的键的地址
- 0x05, A表示(768 + A)之间的键的地址

这些值(0x01,0x03,0x05)是奇数，不能用来存储器件的地址。

用户也可以用直方图来检查值。

键的长度也可以不固定，不过在这种情况下，解压缩将需要更多的微控制器资源。

参考文献和授权信息

Octave可以免费下载，请访问GNU操作系统官方网站。该网站还含有Octave授权信息。

用户可能希望利用C/C++集成开发环境（如Dev-C++）检查所生成的代码。欲了解相关信息及授权信息，请访问Bloodshed Dev-C++集成开发环境(IDE)官方网站。

注释