



Embedded gcov

GCC/gcov code coverage data extraction from the actual embedded system, without requiring a file system, or an operating system, or standard C libraries

Ken Peters

1/18/2022



Gcov overview: Basic description

- gcov basic description
 - Compile your code with gcc options for coverage tracking.
 - -ftest-coverage -fprofile-arcs
 - Do not have to compile every source file for coverage.
 - gcc inserts counting code around every function call and branch.
 - Program timing is slowed.
 - Image file size is increased.
 - Run the program.
 - Get the coverage count data.
 - A “normal” program will just create files containing the data for each source code file.
 - An embedded program may not be able to do that. There is another way (this talk!)
 - Run gcov to decode the coverage count data to annotated copies of source.
 - Or other tools to display the data.

Gcov overview: Gcov output



- Example output (from Wikipedia <https://en.wikipedia.org/wiki/Gcov>)

```
#include <stdio.h>
int main (void)
{
1   int i;
10  for (i = 1; i < 10; i++)
    {
9     if (i % 3 == 0)
3         printf ("%d is divisible by 3\n", i);
9     if (i % 11 == 0)
#####        printf ("%d is divisible by 11\n", i);
9     }
1   return 0;
1 }
```

- There are tools that can process this into interactive HTML form
 - Such as lcov <http://ltp.sourceforge.net/coverage/lcov.php>
 - And/or coveralls <https://coveralls.io/features>

Gcov overview: GCC adds coverage functions



"Hello world" without coverage:

```
$ grep ">:" hello.dump | grep -v plt
00001000 <_init>:
00001060 <_start>:
00001090 <deregister_tm_clones>:
000010c0 <register_tm_clones>:
00001100 <__do_global_ctors_aux>:
00001140 <frame_dummy>:
00001149 <main>:
00001170 <__libc_csu_init>:
000011e0 <__libc_csu_fini>:
000011e8 <_fini>:
```

Basically, we will provide our own versions (based on gcc code) of `__gcov_init` and `__gcov_exit` that dump out the count data similar to `__gcov_dump_one`, but in a way that works for our embedded system instead of writing files.

Though if you have a luxurious embedded filesystem that could be used.

"Hello world" with coverage:

```
$ grep ">:" hello_gcov.dump | grep -v plt
00001000 <_init>:
000013e0 <gcov_do_dump.cold>:
000013f0 <_start>:
00001420 <deregister_tm_clones>:
00001450 <register_tm_clones>:
00001490 <__do_global_ctors_aux>:
000014d0 <frame_dummy>:
000014d9 <main>:
0000151f <_sub_I_00100_0>:
00001535 <_sub_D_00100_1>:
00001550 <__gcov_merge_add>:
...
00001e50 <gcov_do_dump>:
00002d00 <__gcov_dump_one>:
00002d30 <__gcov_exit>:
00002dc0 <__gcov_init>:
00002e80 <__libc_csu_init>:
00002ef0 <__libc_csu_fini>:
00002ef8 <_fini>:
```

Gcov overview: GCC/gcov files



- From the GNU gcc documentation (<https://gcc.gnu.org/onlinedocs/gcc/Gcov-Data-Files.html#Gcov-Data-Files>)
 - The .gcno notes file is generated when the source file is compiled with the GCC -ftest-coverage option. It contains information to reconstruct the basic block graphs and assign source line numbers to blocks.
 - The .gcda count data file is generated when a program containing object files built with the GCC -fprofile-arcs option is executed. A separate .gcda file is created for each object file compiled with this option. It contains arc transition counts, value profile counts, and some summary information.
- We need to get the .gcda info out of the embedded system.

Embedded gcov: Credits



- Many gcc developers who created the real gcov data system.
- Thanassis Tsiodras (April 2016)
(<https://www.thanassis.space/coverage.html>)
 - “This first post is about coverage - and besides showcasing the basics, moves on to deeper challenges that arise when you need to work with embedded devices that have no filesystems - offering a solution that is portable across all GCC versions, for all embedded platforms that are supported by GCC.”
- Alexander Tarasikov (<http://allsoftwaresucks.blogspot.com/2015/05/gcov-is-amazing-yet-undocumented.html>)
 - “Since we're running in kernel or "bare-metal", we don't have neither libgcov nor file system to dump the ".gcda" files. But one should not fear GCOV! Adding it to your kernel is just a matter of adding one C file (which is mostly shamelessly copy-pasted from linux kernel and gcc sources) and a couple CFLAGS.”
- I did find there was a little more to do than that, at least in my system.
 - Especially if the executable has to be fairly small (such as in PROM).

Embedded gcov: Insert in your code



```
#include "gcov_public.h"
```

```
...
```

```
    // want this as early as possible,  
    // but cannot call this until after  
    // the trap table and system stuff are set up  
    // may not be needed in all systems, depending on startup code  
    __gcov_call_constructors();
```

```
...
```

```
        case 9: // a command in my system  
                __gcov_exit(); // dumps the data  
                result = RET_NO_ERROR;  
                break;
```

```
...
```

- Add the embedded gcov source files `gcov_public.c` and `gcov_gcc.c` to your build.
 - You likely want a separate gcov build target, with preprocessor flags.
 - May need a separate linker file for gnu ld, defining symbols for `__gcov_call_constructors()` (see backup).
- Then compile with gcc and usual coverage flags `-ftest-coverage -fprofile-arcs`

Embedded gcov: Customize to your system



- gcov_public.h has preprocessor defs for features, uncomment the ones you need.
 - #define GCOV_OPT_OUTPUT_SERIAL_HEXDUMP
 - #define GCOV_OPT_OUTPUT_BINARY_MEMORY
 - #define GCOV_OPT_USE_MALLOC
 - Etc.
- gcov_public.h has preprocessor defs for necessary functions, edit as needed
 - #define GCOV_PRINT_STR(str) puts((str))
 - #define GCOV_PRINT_NUM(num) print_num((num))
 - In my system, these are pre-existing custom UART output functions, not stdio library
- gcov_public.c might need editing to adjust values or some behavior
 - Array sizes if not using malloc
 - Memory address if using GCOV_OPT_OUTPUT_BINARY_MEMORY
 - Print statements if using GCOV_OPT_OUTPUT_SERIAL_HEXDUMP without printf or imitation
 - Etc.

Embedded gcov: Output

No RTOS
needed!



Burn into
PROM!

- Serial port hexdump: Basically concatenated .gcda files

```
gcov_exit
Emitting 0x00000038 bytes for /home/kjpeters/aa/gcov/objs/jump.gcda
00000000: 67 63 64 61 41 37 35 52 a8 c9 80 94 01 00 00 00
00000010: 00 00 00 03 1a 6f d1 de b7 0a cb b5 7c b3 4a f9
00000020: 01 a1 00 00 00 00 00 04 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 00 00 00 00
/home/kjpeters/aa/gcov/objs/jump.gcda
...
Emitting 0x00000040 bytes for /home/kjpeters/aa/gcov/objs/sleepreset.gcda
00000000: 67 63 64 61 41 37 35 52 a8 c9 7c 99 01 00 00 00
00000010: 00 00 00 03 07 0d aa 51 ac fd 10 6f 49 ba a8 52
00000020: 01 a1 00 00 00 00 00 06 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
/home/kjpeters/aa/gcov/objs/sleepreset.gcda
Emitting 0x00000078 bytes for /home/kjpeters/aa/gcov/objs/autoboot.gcda
00000000: 67 63 64 61 41 37 35 52 a8 c9 7c 76 01 00 00 00
00000010: 00 00 00 03 46 6b 8c 1c e8 40 5e e7 f0 4a a9 37
00000020: 01 a1 00 00 00 00 00 14 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070: 00 00 00 00 00 00 00 00
/home/kjpeters/aa/gcov/objs/autoboot.gcda
Gcov End
```

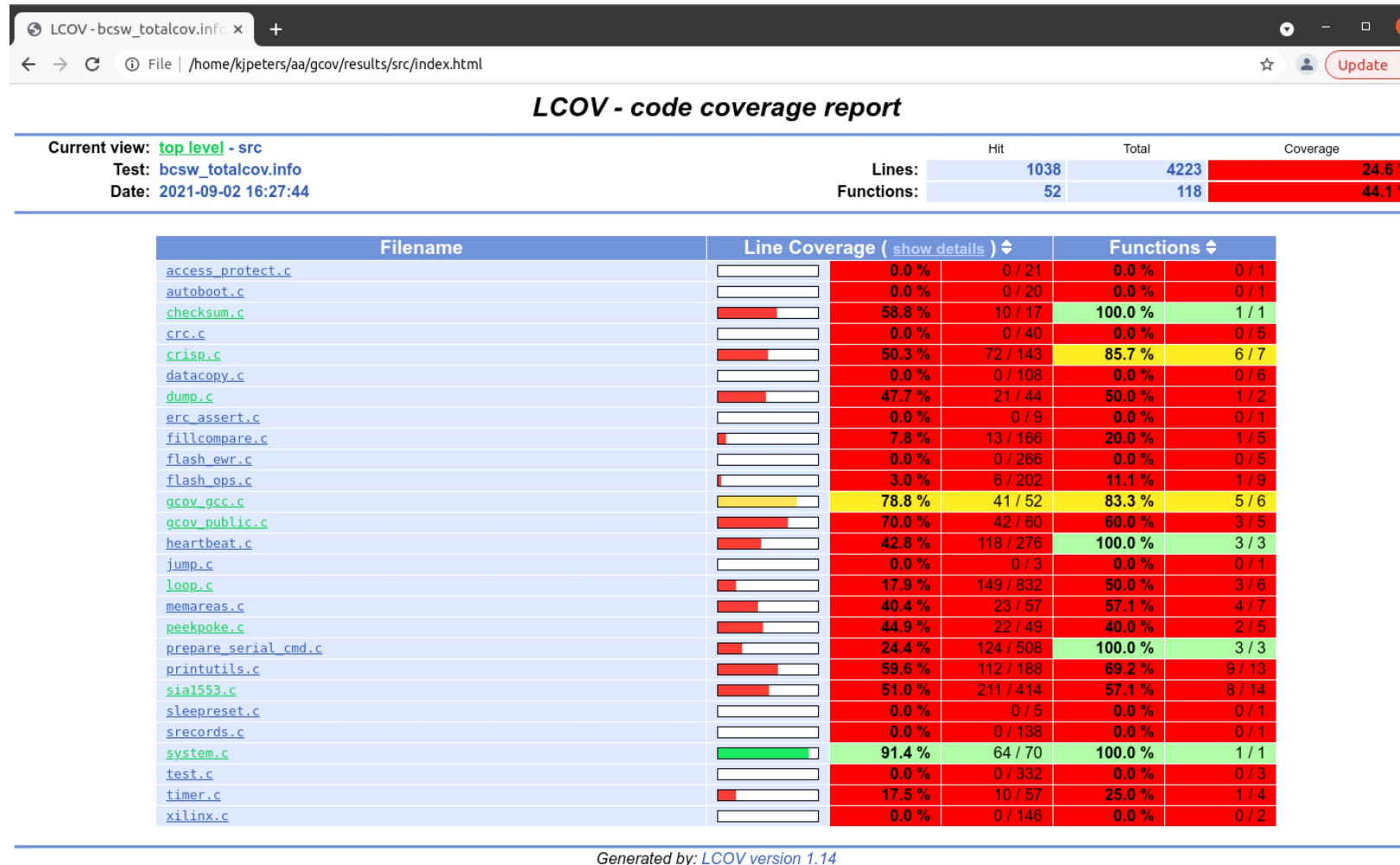
- Binary memory or file: Basically same data in binary form. Extract however your system allows
 - Debugger
 - S/C data interface

Embedded gcov: Postprocessing



- Convert concatenated .gcda into separate files in object directory.
 - I use shell script and awk script on the serial port log file (see backup).
 - Have not produced a tool yet for binary memory dump, or binary file.
- Now can process as any non-embedded gcov data.
 - I use shell scripts with lcov and genhtml (comes with lcov).
 - Have pushed results to a coveralls server (via python tool).
 - With help from JPLers James Mertz (512) via Michael Vanover (512)
 - Worked, not exercised much.
 - Example lcov html output on next 2 slides (final 2 slides!)

Example lcov html output (file list)



Example lcov html output (source code)



```
LCOV - bcsw_totalcov.info x +
File | /home/kjpeters/aa/gcov/results/src/peekpoke.c.gcov.html
17 :
18 : /******
19 : /** @brief Reads the contents of an address and prints to uart a
20 : *
21 : * @author 2002-07-11 dsorozco
22 : * @author 2008-05-02 cyamamot
23 : *
24 : * @param [in] address
25 : *
26 : * @return contents of address
27 : *****/
28 : UINT32
29 : 2 : peek(UINT32 address)
30 : {
31 :     UINT32 addr;
32 :     UINT32 temp;
33 :
34 : 2 : addr = address;
35 : 2 : temp = MEMORY_BASE_PTR(addr);
36 :
37 : 2 : puts("address ");
38 : 2 : print_num(address);
39 : 2 : puts(" = ");
40 : 2 : print_num(temp);
41 : 2 : puts("\n");
42 :
43 : 2 : return (temp);
44 : } /* end peek() */
45 :
46 :
47 :
48 : /******
49 : /** @brief Writes a value to an address
50 : *
51 : * @author 2002-07-11 dsorozco
52 : * @author 2008-05-02 cyamamot
53 : *
54 : * @param [in] address - address to write
55 : * @param [in] value - value to set
56 : *****/
57 : void
58 : 0 : poke(UINT32 address,
59 :     UINT32 value)
60 : {
61 :     UINT32 addr;
62 :
63 : 0 : addr = address;
64 : 0 : MEMORY_BASE_PTR(addr) = value;
65 :
66 : 0 : puts("set address ");
67 : 0 : print_num(address);
68 : 0 : puts(" to ");
69 : 0 : print_num(value);
70 : 0 : puts("\n");
71 :
72 : 0 : return;
73 : } /* end poke() */
74 :
```

Embedded gcov: Availability



- In process of getting permission to release as open source
 - Expected to public github as <https://github.com/nasa-jpl/embedded-gcov>
 - See other JPL open source releases under <https://github.com/nasa-jpl>



Backup



__gcov_call_constructors support

- Needed if your system startup code does not internally call constructors.
 - Unix systems ordinarily do this automatically internally. Embedded systems might not (especially if plain C, not C++).
- Uncomment GCOV_OPT_PROVIDE_CALL_CONSTRUCTORS in gcov_public.h
- Linker file needs to produce a .ctors section containing gcov-generated constructors and defining symbols __ctor_list and __ctor_end:

```
/* for loadable gcov image, need constructors */
/* (even for C code, needed for __gcov_init) */
/* if not gcov, does not hurt */
.ctors : {
    __ctor_list = . ;
    *(SORT(.ctors.*))
    *(.ctors)
    __ctor_end = . ;
    . = ALIGN(16);
} > ram
```

- Call in your code embedded-gcov-provided __gcov_call_constructors()
 - Uses these symbols to call all the gcov-generated constructors to initialize.



Serial hexdump separation

- Shell script (with awk script) to separate into .gcda file for each source file

```
#!/bin/bash
```

```
# Typical usage: ./gcov_convert.sh ../test01_serial_log.txt
```

```
# Serial log file can have null characters in it
```

```
# if the system rebooted during the log.
```

```
# Also remove any other non-ASCII chars (only allow specific octal character values through)
```

```
# (thanks to https://alvinalexander.com/blog/post/linux-unix/how-remove-non-printable-ascii-characters-file-unix/)
```

```
tr -cd '\11\12\15\40-\176' < $1 > ${1%.*}_nonulls.txt
```

```
# Convert from DOS test file
```

```
dos2unix ${1%.*}_nonulls.txt
```

```
# Create separate .gcda.xxd files from the serial log
```

```
# The files can be created at the full pathname specified in the log
```

```
# or can be created in the current directory, see serial_split.awk.
```

```
# Current directory is more convenient for us here.
```

```
cat ${1%.*}_nonulls.txt | awk -f serial_split.awk
```

```
# Move the .gcda.xxd files from here to ../objs
```

```
# which is where the object files and .gcno files
```

```
# should already be
```

```
mv *.gcda.xxd ../objs
```

```
# Convert the separate .gcda.xxd files to separate binary .gcda files
```

```
# And remove the .xxd files
```

```
for i in `find ../objs -name '*.gcda.xxd';do
```

```
    cat "$i" | xxd -r > "${i%.*}.gcda"
```

```
    rm "$i"
```

```
done
```

```
/Emit/ {  
    print;  
    init = 1;  
    tstr="";  
    next;  
}
```

```
!/gcda/ {  
    if (!init || !NF) {  
        next;  
    }  
    tstr = tstr""$0"\n";  
    next;  
}
```

```
/gcda/ {  
    if (!init) {  
        next;  
    }  
    # to create file at full path location  
    #print tstr > $0".xxd";
```

```
    # to create file in current directory  
    cmd = sprintf("basename %s", $0);  
    cmd | getline fname;  
    print tstr > fname".xxd";
```

```
    # on to next file  
    init = 0;  
    tstr="";  
    next;  
}
```

```
# Copyright 2021, by the California Institute of Technology. ALL  
# RIGHTS RESERVED. United States Government Sponsorship  
# acknowledged.
```

```
# Any commercial use must be negotiated with the Office of Technology  
# Transfer at the California Institute of Technology.
```

```
#
```

```
# This software may be subject to U.S. export control laws and  
# regulations. By accepting this document, the user agrees to comply  
# with all applicable U.S. export laws and regulations. User has the  
# responsibility to obtain export licenses, or other export authority  
# as may be required before exporting such information to foreign  
# countries or providing access to foreign persons.
```