Rajbir S Kahlon

9/22/19

CMPE 146 Real-Time Embedded System Co-design

Professor Wu

**Lab 2**

**Exercise 1.1**

static uint32\_t myData[10000];

uint32\_t jjj;

for(jjj=0; jjj<10000; jjj++) {

myData[jjj] = rand()|1;

}

printf("%d\n", myData[0]);

printf("%d\n", myData[1]);

printf("%d\n", myData[2]);

printf(" %d\n", myData[3]);

printf("%d\n", myData[4]);

printf("%d\n", myData[5]);

printf("%d\n", myData[6]);

**Exercise 1.2**

static uint32\_t compute\_simple\_checksum(uint8\_t\* data, uint32\_t length) {

uint32\_t crc,LL;

crc = 0;

for(LL=0;LL<length;LL++){

crc += data[LL];

}

return crc;

}

**Exercise 1.3**

A screenshot of a cell phone

Description automatically generated

Hardware was the fastest. The software was a lot slower and custom-written was a little slower than software method.

**Exercise 1.4**

The results are the same as before even with flipping the least significant bit of myData[20] and then myData[21]. All 3 checksum always result in the same, but the hardware accelerated method is way faster than the other 2. Software was slightly faster than custom method that we wrote in exercise 1.2. Code is in the appendix.

**Exercise 2**

Was not able to figure out how to set baud rate and other settings in Putty.

A screenshot of a computer

Description automatically generated

**Appendix:**

/\* --COPYRIGHT--,BSD

\* Copyright (c) 2017, Texas Instruments Incorporated

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* \* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* \* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\*

\* \* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\* --/COPYRIGHT--\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* MSP432 CRC32 - CRC32 Calculation

\*

\* Description: In this example, the CRC32 module is used to calculate a CRC32

\* checksum in CRC32 mode. This value is compared versus a software calculated

\* checksum. The idea here is to have the user use the debugger to step through

\* the code and observe the CRC calculation in the debugger. Note that this

\* code example was made to use the standard CRC32 polynomial value of

\* 0xCBF43926.

\*

\* MSP432P401

\* ------------------

\* /|\| |

\* | | |

\* --|RST |

\* | |

\* | |

\* | |

\* | |

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

#include <time.h>

#define CRC32\_POLY 0xEDB88320

#define CRC32\_INIT 0xFFFFFFFF

volatile uint32\_t ii;

static uint32\_t calculateCRC32(uint8\_t\* data, uint32\_t length);

volatile uint32\_t hwCalculatedCRC, swCalculatedCRC, customCRC;

//![Simple CRC32 Example]

int main(void)

{

static uint32\_t myData[10000];

uint32\_t jjj;

for(jjj=0; jjj<10000; jjj++) {

myData[jjj] = rand()|1;

}

printf("%d\n", myData[0]);

printf("%d\n", myData[1]);

printf("%d\n", myData[2]);

printf("%d\n", myData[3]);

printf("%d\n", myData[4]);

printf("%d\n", myData[5]);

printf("%d\n", myData[6]);

uint32\_t ii;

/\* Stop WDT \*/

MAP\_WDT\_A\_holdTimer();

MAP\_CRC32\_setSeed(CRC32\_INIT, CRC32\_MODE);

for (ii = 0; ii < 9; ii++)

MAP\_CRC32\_set8BitData(myData[ii], CRC32\_MODE);

/\* Getting the result from the hardware module \*/

uint32\_t hw\_time;

Timer32\_startTimer(TIMER32\_BASE, true);

hwCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

hw\_time = Timer32\_getValue(TIMER32\_BASE);

printf("hw is %x\n", hwCalculatedCRC);

printf("\n time it took was %d\n", hw\_time);

/\* Calculating the CRC32 checksum through software \*/

uint32\_t sw\_time;

Timer32\_startTimer(TIMER32\_BASE, true);

swCalculatedCRC = calculateCRC32((uint8\_t\*) myData, 9);

printf("\n ignore above");

swCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

sw\_time = Timer32\_getValue(TIMER32\_BASE)-hw\_time;

printf("\n sw is %x\n", swCalculatedCRC);

printf("\n time it took was %d\n", sw\_time);

/\*Calculating CRC32 custom simple checksum\*/

uint32\_t custom\_time;

Timer32\_startTimer(TIMER32\_BASE, true);

customCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

custom\_time = Timer32\_getValue(TIMER32\_BASE);

printf("\n\n custom is %x\n", customCRC);

printf("\n time it took was %d\n", custom\_time);

printf("\n \n");

/\*Take 20th value of array and flip it\*/

uint32\_t og = ~myData[20];

og ^= 1;

printf("\n current myData[20] is %d\n", myData[20]);

myData[20] = og;

printf("\n new flipped myData[20] is %d\n", myData[20]);

uint32\_t hw\_time1;

uint32\_t sw\_time1;

uint32\_t custom\_time1;

/\* Getting the new result from the hardware module \*/

Timer32\_startTimer(TIMER32\_BASE, true);

hwCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

hw\_time1 = Timer32\_getValue(TIMER32\_BASE);

printf("hw 20th bit is %x\n", hwCalculatedCRC);

printf("\n time it took was %d\n", hw\_time1);

/\* Calculating the new CRC32 checksum through software \*/

Timer32\_startTimer(TIMER32\_BASE, true);

swCalculatedCRC = calculateCRC32((uint8\_t\*) myData, 9);

printf("\n ignore above");

swCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

sw\_time1 = Timer32\_getValue(TIMER32\_BASE)-hw\_time1;

printf("\n sw 20th bit is %x\n", swCalculatedCRC);

printf("\n time it took was %d\n", sw\_time1);

/\*Calculating new CRC32 custom simple checksum\*/

Timer32\_startTimer(TIMER32\_BASE, true);

customCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

custom\_time1 = Timer32\_getValue(TIMER32\_BASE)-sw\_time1;

printf("\n\n new 20th bit custom is %x\n", customCRC);

printf("\n time it took was %d\n", custom\_time1);

printf("\n \n");

/\*Take 21st value of array and flip it\*/

uint32\_t og2 = myData[21];

printf("\n current myData[21] is %d\n", myData[21]);

og2 ^= 1;

myData[21] = og2;

printf("\n new flipped myData[21] is %d\n", myData[21]);

uint32\_t hw\_time2;

uint32\_t sw\_time2;

uint32\_t custom\_time2;

/\* Getting the new result from the hardware module \*/

Timer32\_startTimer(TIMER32\_BASE, true);

hwCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

hw\_time2 = Timer32\_getValue(TIMER32\_BASE);

printf("hw 21stth bit is %x\n", hwCalculatedCRC);

printf("\n time it took was %d\n", hw\_time2);

/\* Calculating the new CRC32 checksum through software \*/

Timer32\_startTimer(TIMER32\_BASE, true);

swCalculatedCRC = calculateCRC32((uint8\_t\*) myData, 9);

printf("\n ignore above");

swCalculatedCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

sw\_time2 = Timer32\_getValue(TIMER32\_BASE)-hw\_time2;

printf("\n sw 21st bit is %x\n", swCalculatedCRC);

printf("\n time it took was %d\n", sw\_time2);

/\*Calculating new CRC32 custom simple checksum\*/

Timer32\_startTimer(TIMER32\_BASE, true);

customCRC = MAP\_CRC32\_getResultReversed(CRC32\_MODE) ^ 0xFFFFFFFF;

Timer32\_haltTimer(TIMER32\_BASE);

custom\_time2 = Timer32\_getValue(TIMER32\_BASE)-sw\_time2;

printf("\n\n new 21st bit custom is %x\n", customCRC);

printf("\n time it took was %d\n", custom\_time2);

/\* Pause for the debugger \*/

\_\_no\_operation();

}

//![Simple CRC32 Example]

/\* Standard software calculation of CRC32 \*/

static uint32\_t calculateCRC32(uint8\_t\* data, uint32\_t length)

{

uint32\_t ii, jj, byte, crc, mask;;

crc = 0xFFFFFFFF;

for(ii=0;ii<length;ii++)

{

byte = data[ii];

crc = crc ^ byte;

for (jj = 0; jj < 8; jj++)

{

mask = -(crc & 1);

crc = (crc >> 1) ^ (CRC32\_POLY & mask);

}

}

printf("%d", crc);

return ~crc;

}

static uint32\_t compute\_simple\_checksum(uint8\_t\* data, uint32\_t length) {

uint32\_t crc,LL;

crc = 0;

for(LL=0;LL<length;LL++){

crc += data[LL];

}

return crc;

}