University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2019   HW1: Hello World, Remember C   Wednesday, August 28

Due: Friday, September 6, 5:00PM

This first assignment is designed to refresh your C knowledge and introduce you to the software and hardware that will be used throughout the course. You will use the Xilinx SDSoC (Software-Defined System on Chip) software to run a "Hello, World!" application on one of the ARM Cortex-A53 cores on a Ultra96-v2. Recently, SDSoC has been combined with SDAccel, so we will typically refer to the software as SDx.

This assignment is more scripted than future assignments. The goal is get you started quickly on the platform and toolset. You should take the time to explore and understand the tools. While the assignment is fairly detailed, it does not go so far as to tell you every button to press. You will need to consult other documentation identified and experiment to complete many steps.

Like most modern tools you will use, these tools are complicated and their behavior at times will be obscure. While we have given you instructions here, you will likely find them incomplete or do something not quite as intended and find yourself trying to figure out where to go. We expect it will take some reading, searching, and experimentation to get back on track (it took us a bit of all of these to get this far).

You probably want to read through the entire assignment (including the "Homework Submission" section at the end) before you start to work the assignment.

Collaboration

In this assignment, you work with partners that we assigned. You can find the assignment on Canvas in the *Partners* map under the *Files* section. In the event that the partner assignment does not work out (e.g., your assigned partner has already dropped the course), contact the instructor or TA as soon as possible.

You can certainly do this assignment alone. Everyone should be learning how to use all the tools, so don't take this as an opportunity to compartmentalize who knows how to do what. Our rationale for pairs for this first assigment is so that you can reason together on how to get the tools to work. For future assignments, you will have a different partner that we will assign—all the more reason to make sure everyone knows how to use all the tools.

Each student should submit a complete report. You are free to share information about how to perform certain activities using the Xilinx software with other students. In fact, we encourage you to share any difficulties that you have with the software and general

solutions or workarounds for them on Piazza. However, **you are not allowed to share the results** (e.g. output files, numbers, graphs) that you obtain using the software outside of the pair working together on the assignment. Moreover, **final results should be analyzed and conclusions should be drawn individually**.

All students must follow the Code of Academic Integrity. Infringement of the code can have **severe consequences** for you and your partner, such as **failing the course or cancellation of your student visa** if you are an international student, so please familiarize yourself with them. See the course policies on the course web page `http://www.seas.upenn.edu/~ese532` for full details of our policies for this course.

Ultra96 Boards

Our original hope was to distribute boards in class on Day 1 (August 28th). It looks like the boards may not be available then. If they are not, watch piazza for updates on when we will distributed boards. You can work the C Refresher section without the boards, so we encourage you to get started on that immediately.

SDx Software

We have installed SDx on the PCs in the Detkin and Ketterer Labs. Currently [August 28, 2019] Windows PCs have the software. The Linux PCs will also have SDx installed in the near future.

If you would like to run the software on your own system, you can download it from the Downloads page on the Xilinx website. We require SDx 2018.3 because we test our instructions on that version. We cannot support or accept homework based on other versions. Be aware that SDx has strict operating system requirements, and that backward compatibility is not guaranteed. However, we have tried SDSoC 2018.3 on Ubuntu 18.04.1 LTS, and it works well. The software requirements can be found in the SDx Environments Release Notes, Installation, and Licensing Guide. We advise against running SDx in a virtual machine because we have not seen any successful cases. Typically, communication through JTAG and UART is the issue. Should you manage to make this work anyway, please share how you did this with us.

Be aware that the complete installation of SDx needs a lot of disk space (around 79 GB). If you do not install the full device packages, you cannot create VIVADO_HLS project. We recommend you install the full device packages.

Windows

If you install SDx on your own Windows-based system, you can use a license from the license server. In order to use it, you should set the environment variable `LM_LICENSE_FILE` or

`XILINXD_LICENSE_FILE` to `2100@potato.cis.upenn.edu;1709@potato.cis.upenn.edu;`
`1717@potato.cis.upenn.edu;27010@potato.cis.upenn.edu;27009@potato.cis.upenn.edu`.
You can also set the variable in the Xilinx License Configuration Manager (XLCM). In the
past, we have seen an issue where SDx complains that you have to install the Visual Studio
2012 Redistributable. However, even after installing it, SDx keeps complaining. To solve
this, you can remove the file `Vivado\tps\win64\xvcredist.exe` in your SDx installation
root. More information can be found here.

Linux

If you install SDx on Ubuntu, you should pay special attention to the instructions pro-
vided in the manual about installing certain 32-bit libraries and creating a symbolic link for
`gmake`. A license can be obtained from the license server. In order to use the license server,
you should set the environment variable `LM_LICENSE_FILE` or `XILINXD_LICENSE_FILE` to
`2100@potato.cis.upenn.edu:1709@potato.cis.upenn.edu:1717@potato.cis.upenn.edu:`
`27010@potato.cis.upenn.edu:27009@potato.cis.upenn.edu`.

A hint from past years: you can install drivers by running:
`SDK/data/xicom/cable_drivers/lin64/install_script/install_drivers/install_drivers`

C Refresher

For some of these questions you will need to compile and run C code. (For others, you may find it useful to experiment with C code.) For this section, you may use any editor and C compiler. If you don't have a favorite C compiler on your personal laptop or PC, you can use the machines in Detkin and Ketterer. On the Linux machines, you can compile a simple, single C file application in `program.c` using:

```
 gcc -o program program.c
```

1. How do you declare a pointer that points to the following two-dimensional array, which you can use to address the elements as `a[0][0]`, `a[0][1]`, etc.? (1 line)

   ```
   int a[2][3];
   ```

2. Considering the following code, give an expression to obtain the address of `b` that can be accessed via the third element of x. (1 line)

   ```
   struct s2
   {
     float a;
     int b;
   };

   struct s1
   {
     int c;
     struct s2 **d;
   };

   struct s1 x[5];
   ```

3. The following array will be stored as a sequence of bits in memory. We could also consider these bits as a sequence of bytes (unsigned char). Show code that prints those bytes. Avoid needless copying or losing information. Note that an IEEE Double-precision floating-point value is stored in 64 bits. You can see https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/ for more information on IEEE Double-precision floating-point format, but understanding this is not necessary to your solution, only to understanding what it is your solution is reporting.

   ```
   double a[] = {3.14, 2.71};
   ```

4. Put together code to print the address associated with each of the arrays, a, b, c, and d. Include the code and the results of running it and reporting the addresses.

```
void temp(int i) {

    int a[2];
    int b[3];
    int *c;
    int *d;
    c=(int *)malloc(sizeof(int)*4);
    d=(int *)malloc(sizeof(int)*5);

    // print addresses for arrays here....

    return;
}
```

5. What might happen with the following code?

```
int a[3];
int b[4];
int c[5];

 // intervening code omitted

b[4]=13;
```

Many different things could happen. Give multiple answers. For each identified case, what happens and why (2 lines max for each case).

6. Compile and run the following code.

```c
#include "stdio.h"
#include "stdlib.h"

int main (int argc, char **argv)
{
  unsigned char a[3]={0xFF,0x01,73};
  unsigned char sum;
  unsigned int intsum;

  signed char sa[3]={127,1,33};
  signed char ssum;
  unsigned int sintsum;

  fprintf(stdout,"Unsigned:\n");
  for (int i=0;i<3;i++)
    for (int j=0;j<3;j++)
      {
sum=a[i]+a[j];
intsum=a[i]+a[j];
fprintf(stdout,"in decimal: %d+%d=%d (intsum=%d)\t",a[i],a[j],sum,intsum);
fprintf(stdout,"in hexadecimal: %x+%x=%x (intsum=%x)\n",
                      a[i],a[j],sum,intsum);
      }

  fprintf(stdout,"Signed:\n");
    for (int i=0;i<3;i++)
    for (int j=0;j<3;j++)
      {
ssum=sa[i]+sa[j];
sintsum=sa[i]+sa[j];
fprintf(stdout,"in decimal: %d+%d=%d (intsum=%d)\t",sa[i],sa[j],ssum,sintsum);
fprintf(stdout,"in hexadecimal: %x+%x=%x (intsum=%x)\n",
                        sa[i],sa[j],ssum,sintsum);
      }

}
```

Explain the results you get. Why do the `char` and `unsigned char` sums differ from the integer sums? (2-3 lines each) Why does the computer produce the specific values that it does for each of the `char` and `unsigned char` sums? (1-3 lines each)

7. Below is a simple example of a linked list and a test program. Your job is to complete insert_in_order function. This function should add the new element into the linked list in ascending order. For example, when given the values 20, 5, 10 our list should end up sorted as 5, 10, 20. In this case the head of the list is 5 and the "tail" is 20.

The purpose of this problem is to help prepare you to write and debug C code with pointers. A traditional linked list contains some arbitrary data as well as a pointer to the next element in the list.

Hint: What is the typical case behavior for insert_in_order?
What special cases do you need to handle within insert_in_order function?

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

/*
 * definition of linked list node
 * the value is the element in the list
 * the pointer points to the next element in the list
 * if it exists
 */
typedef struct node_struct
{
    int8_t value;
    struct node_struct* next;
} node_struct;

// when the list is not empty, this variable will always contain
// the element in the first position in the list (the head of the list)
static node_struct* head;

// This function will perform the insertion sort and maintain the linked list
// you will need to maintain the links and properly place the new element
void insert_in_order( node_struct* new_element )
{
  // YOUR CODE HERE
}
```

```
// this function creates a new entry into the list to be inserted
void add_element( int8_t value )
{
   // create a new element in our list
   node_struct* new_element = (node_struct*)malloc( sizeof(node_struct) );
    if(new_element == NULL)
    {
        printf("malloc failed \n");
        return;
    }
    // assign our values
    new_element->value = value;
    new_element->next = NULL;

    insert_in_order(new_element);
}

// prints the entirety of our list
void print_list(  )
{
    if( head == NULL )
    {
        printf("list is empty \n");
        return;
    }

    node_struct* element = head;

    while( element != NULL )
    {
        printf("value in list %d \n",element->value);
        element = element->next;
    }
}
```

```
int main()
{
    int8_t a = 20;
    int8_t b = 5;
    int8_t c = 10;
    int8_t d = 21;
    int8_t e = 41;
    int8_t f = 2;

    head = NULL;

    add_element( a );
    add_element( b );
    add_element( c );
    add_element( d );
    add_element( e );
    add_element( f );

    print_list();

    return 0;
}
```

8. What is the purpose of the preprocessor, compiler, and linker? (each 3 lines max.)
   Potentially useful:

   - http://lampwww.epfl.ch/~fsalvi/docs/gcc/www.network-theory.co.uk/docs/gccintro/gccintro_68.html
   - https://www.tenouk.com/ModuleW.html

   [We point you at the gcc documentation because it is easily available online. Many of the options and concepts are the same across other C compilers. In some cases, SDSoC is using gcc internally. For some of your development (including, perhaps, for this section of this assignment) you may find it useful to get your C working on a workstation or laptop using gcc before moving to SDSoC.]

9. If the preprocessor cannot find a file that is included with #include, give at least three different ways you could resolve the problem so that the preprocessor can find the file? (1–2 lines each)

   Possibly useful:

   - https://gcc.gnu.org/onlinedocs/gcc/Directory-Options.html#Directory-Options
   - http://gcc.gnu.org/onlinedocs/gcc/Preprocessor-Options.html

10. If the linker gives you an error like undefined reference to ..., identify three reasons this could occur and at least one way to resolve each. (1–2 lines each)

    Possibly useful:

    - https://gcc.gnu.org/onlinedocs/gcc/Directory-Options.html#Directory-Options
    - https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html

11. How does SDx (or gdb) know where functions and data are located in the executable when you are debugging? (3 lines max.)

    Possibly useful:

    - run objdump on an executable you compiled. Run objdump -help to see what options it offers. Experiment with the options to see what information you can get it to display.
    - http://lampwww.epfl.ch/~fsalvi/docs/gcc/www.network-theory.co.uk/docs/gccintro/gccintro_74.html

First SDx Application

1. Create a new *Workspace* directory in which you would like to store all your SDx project files. In the labs, you can make a directory under `S:\`, your Eniac home directory. Choose a directory with a short path name because SDx cannot handle paths with more than 260 characters, and it needs many of the characters for the file hierarchy that it generates in your directory. In Windows, make sure that you are not using UNC addresses, such as `\\system\dir\dir\`, when you are using network resources, but use network drives, such as `S:\`.

2. If you are using Linux, source `<SDx directory>/2018.3/settings64.sh` to prepare your shell for running SDx.

3. Launch SDx and switch to the workspace directory that you created. In the labs, you can start SDx via *All Programs → ESE Lab Software → Xilinx Design Tools → SDx 2018.3 → SDx IDE 2018.3*. You can find instructions to perform this and the next step in the SDx Environment User Guide: Getting Started.

4. The freshly installed SDx software may not include the board platform for ultra96-v2. You can download the SDx platform SDSoC Baremetal Platform - Xilinx Matrix Multiply Example/SDSoC_Platform_v2018.3. Unzip the downloaded files and install the platform as SDSoC_Platform_v2018p3.pdf.

5. (only for your own installation on non-CETS computers) As SDx is on top of VIVADO HLS and VIVADO, we should also download the platform from here, and copy the directory `./ultra96v2` to
`<Vivado directory>/Vivado/2018.3/data/boards/board_files/`.

6. Create an empty *SDx project* for the Ultra96-v2 with a *Standalone* operating system. Start with "Create SDx Project", choose "Application". The Ultra96-v2 is indicated as the *u96v2_avnet[custom]* platform in SDx. Under "System Configuration" select "u96v2_avnet". Under "Domain" select "'Standalone". When asked for a Template, select "Empty Application" An operating system, e.g., Linux, provides more drivers and libraries to the application than the standalone configuration. A drawback is that an operating system will also consume more resources. In this assignment, we will only use the serial port, for which the standalone configuration is sufficient.

7. Create a new file called `hello_world.c` in the `src` directory using SDx (not another text editor). Students familiar with the Eclipse IDE should have no trouble using SDx as it is based on Eclipse. Other students may find the Eclipse Documentation useful.

8. Write C source code for printing a cheerful message such as `Hello World` in the `hello_world.c` file, and save it. To verify whether your code works, you could try to compile it using a regular C compiler such as Visual C or gcc.

9. Build the project. You can refer again to the earlier mentioned SDx Environment User Guide. This will take much longer than a regular C application because SDx will also generate a hardware configuration for the FPGA.
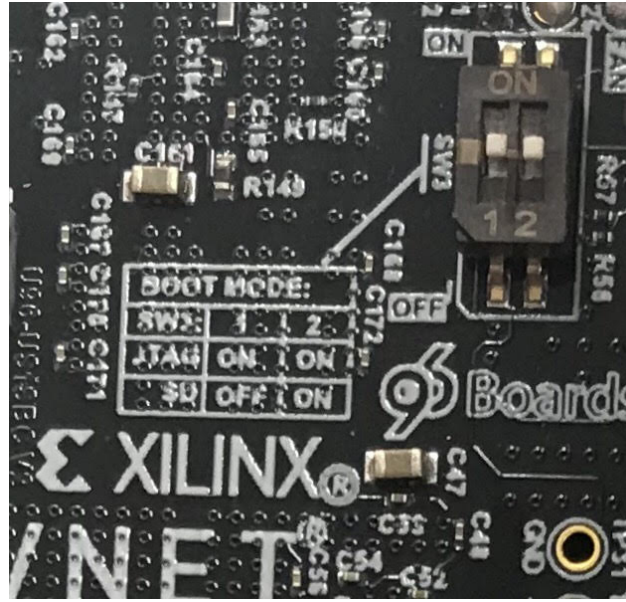
Figure 1: SW3 in JTAG Mode

Note, at the end of this section we will ask you for the build log. It is generated here. There are things you do after this (e.g., Step 17) that will overwrite it, so you may want to capture it here, or at least, note what it looks like here so that when you regenerate it, you will know what Step 22 is asking you to supply.

10. Set the configuration mode of the Ultra96-v2 to JTAG mode while the board is powered off. The mode is set by turning on the switches of SW3. This is illustrated in Figure 1. In JTAG mode, the FPGA configuration is transmitted to the FPGA via a USB cable. In SD card mode, the user copies the configuration to an SD card, which must be transferred physically to the Ultra96-v2. JTAG mode is better fit for short debug cycles because the configuration can be performed automatically when the application is started. More information on the different configurations can be found in the Ultra96-V2 Getting Started Guide.

11. Insert the AES-ACC-U96-JTAG board to Ultra96-v2 board as Figure 2 and connect AES-ACC-U06-JTAG to PC by a USB wire. Be aware that there have been USB cables in the labs that only convey power. We will use the cables for transmitting data, so power-only USB cables are not suitable. You can check whether a cable can transmit data by measuring the resistance of the middle two pins of a connector from one side of the cable to the other. The resistance should be low.

12. Connect the board to an outlet using the power adapter, and power on the board. Note that the board cannot be powered via USB.

    Ketterer: the Ultra96-v2 power supplies will not fit into the power strips directly attached to the bench. Note that there is a Y-extender at each station (one side used for monitor, one side free) that can be used.
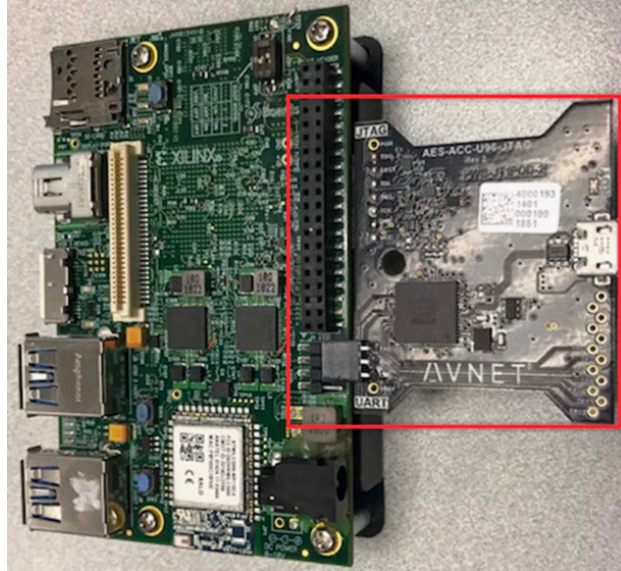
Figure 2: USB Connectors to be Connected

13. If you are using Windows, it may ask you whether you would like to install the USB-UART and Digilent device drivers. In that case, give Windows permission to install the drivers. If the drivers are not installed automatically, you can install the driver manually. Go to *Devices and Printers*. Right-click on the *FTDI USB2UART* device, select *Update device drivers*. In the following dialogs, make sure that you search for the drivers online.

14. If you are using Linux, make sure that `hw_server` is running with root permissions. If that is not the case, kill the process and start it again as root. Without root permission, the application cannot be launched on the FPGA.

15. If you are using Linux, you will have to change the permissions of the serial device each time you connect the Ultra96-v2 to your PC. To avoid this on Ubuntu, you have to add your user to the `dialout` user group with the command `sudo adduser $USER dialout`. You will have to log out and back in again before this takes effect.

16. You haven't turned on the Ultra96-v2 yet. Press the power button as shown in Figure 3, then you can hear the fan is running.

17. Launch the application in debug mode. Right-click on your project in the *Project Explorer*, and choose *Debug As → Launch on Hardware (SDx Debugger)*. The configuration will now be uploaded to the FPGA. Occasionally, SDx warns `No ARM device found on the board`. This is often resolved by power cycling the board and relaunching.

18. A dialog will pop up requesting your approval to switch to the *Debug* perspective, which you should confirm. The perspective affects the menus, views, and toolbars of
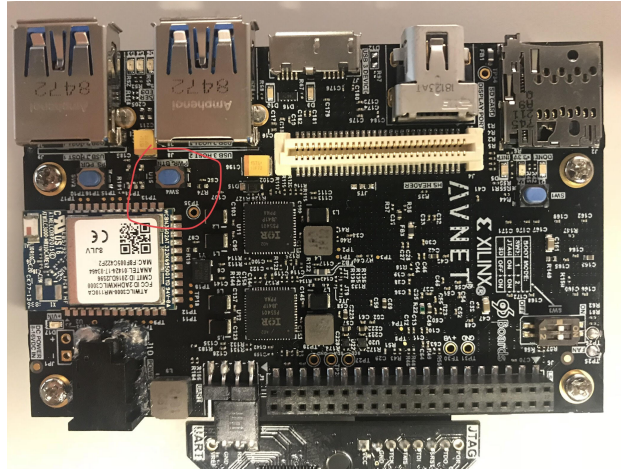
Figure 3: Press the botton to power on

SDx. To switch to another perspective, one presses the buttons in the top-right corner of SDx. The *SDx* button will return you to the perspective that we used so far. For now, stay in the *Debug* perspective. The first executable line of your code should be highlighted in the view in the center.

On windows, a Windows Firewall box may also appear for `hw_server.exe`. Cancel out of this. You do not need this functionality at present.

19. Switch to the *SDx Terminal* tab in the view at the bottom center of the window. Press the plus button in the top right of the view to add a new serial connection to the board. Enter the name of the port and keep the default baud rate of 115200 symbols/s. In Windows, the port name will be `COMXXX`, where `XXX` is a number. You can determine the number by opening the device manager (under Control Panel) to see all available ports and observing which port disappears when the UART connector on the board is disconnected. In Linux, the port name is usually `/dev/ttyUSBXXX`. Here, you can also disconnect and reconnect the cable to see which device file is associated with the serial cable. Once you have completed the dialog, dismiss it by clicking *OK*.

20. Press the *Resume* button in the main SDx toolbar, or select *Run → Resume* from the menu.

21. The application will continue to run. The message that you wrote will be displayed in the SDx terminal.

22. Include the build log in the *Console* window and the program output from the *SDx Terminal* window in your report. Note that neither of these should be empty. As mentioned in Step 9 some of the later steps (like Step 17) overwrite the build log.

Should you decide to change your code after running it, note that SDx does not support hot swapping of code similar to Java in Eclipse. You will have to build your entire application

again and relaunch your debug session. Further note that resuming the application when it has finished will not restart it. Instead, it will keep running in an infinite loop. To restart it you will have to *relaunch* it.

As you probably have noticed by now, building a project can consume much time. That is because SDx builds both hardware and software. Building hardware is notoriously slow, and we must plan for the time required. That said, there are a few things that may reduce the time somewhat. If you run your code on a network drive (such as S: in the labs), you should consider moving your code to a local drive. Moreover, make sure that you build only the projects that you want to build. In other words, select your project and choose *Build Project* instead of building the entire workspace. You can also close projects to avoid building them.

Debug An Application

1. Create a new project with the same settings as the previous one.

2. Create a new source C file and paste the following code in it. The code should print another message, but due to a bug, it doesn't.

```c
#include <stdio.h>

int len(char * s)
{
  int l = 0;
  while (*s) s++;
  return l;
}

int rot13(int l)
{
  if (l >= 'A' && l <= 'Z')
    l = (l - 'A' + 13) % 26 + 'A';
  if (l >= 'a' && l <= 'z')
    l = (l - 'a' + 13) % 26 + 'a';
  return l;
}

char * msg = "Jryy Qbar!!!\n";

int main()
{
  int i = 0;
  printf("The secret message is: ");
  while (i < len(msg))
    printf("%c", rot13(msg[i++]));

  return 0;
}
```

3. Build and run the new project.

4. You will notice that the output is not correct. Use the debugger to locate the bug in the code. Specifically, we are asking you to **not** simply perform `printf` debugging where you insert code and recompile. If you haven't used a debugger before, take the time now, during this easy assignment, to learn the basic tricks of the debugger. Learn how to set breakpoints, step through code, and inspect variables. Instructions

on how to set breakpoints, view variable values, and more can be found in the Eclipse documentation.

5. In your report, describe how you found the bug, how you changed the code, and show the message that should have been displayed.

Homework Submission

Your writeup should follow http://www.seas.upenn.edu/~ese532/fall2019/handouts/writeup_guidelines.pdf

Your writeup should include the following:

1. **Tools**

   (a) Provide the code for your "First Application"

   (b) Provide build log from your successful build of your "First Application"

   (c) Provide the SDx Terminal log from your successful execution of your "First Application"

   (d) Describe how you do the following in SDx (each 3 lines max.):

       i. Add a breakpoint.

       ii. Remove a breakpoint.

       iii. Inspect a variable value.

       iv. Step through program execution without a breakpoint.

2. **Debug**

   (a) Provide the SDx Terminal window for the correct output.

   (b) Provide code for the corrected function.

   (c) Include a description of how you used the tools to identify the bug. (Or confirm a hypothesis you formed about the location of the bug) (5 lines max.)

3. **C** – answers the questions about C pointers, compilers, and executables from the "C Refresher" section.