

# Hello World from Impulse C inside an RTOS

## Impulse C Integration Tutorial

*Using the Impulse C and Xilinx Software Development Kits to integrate existing Impulse C applications and custom hardware accelerators into the context of a real-time operating system*

---

### Contents

[Introduction](#)

[Resources](#)

[Building a Baseline Impulse C Application](#)

[Adding Custom IP-Cores to an Existing System Configuration](#)

[Integrating Impulse C Code into the Xilkernel](#)

[Notes](#)

### Introduction

This tutorial will guide you through the process of manually integrating an Impulse C application into the Xilinx SDK environment running on top of the Xilkernel RTOS.

After completing this tutorial you will be able to:

1. Create a basic Impulse C application and generate the hardware and software files necessary to target the ML507 platform.
2. Add custom user IP cores to an existing system configuration using the Xilinx Platform Studio.
3. Translate Impulse C generated software tasks (C source code) into POSIX equivalent threads that are run inside of the Xilkernel.

### Resources

To complete this tutorial you will need the following resources:

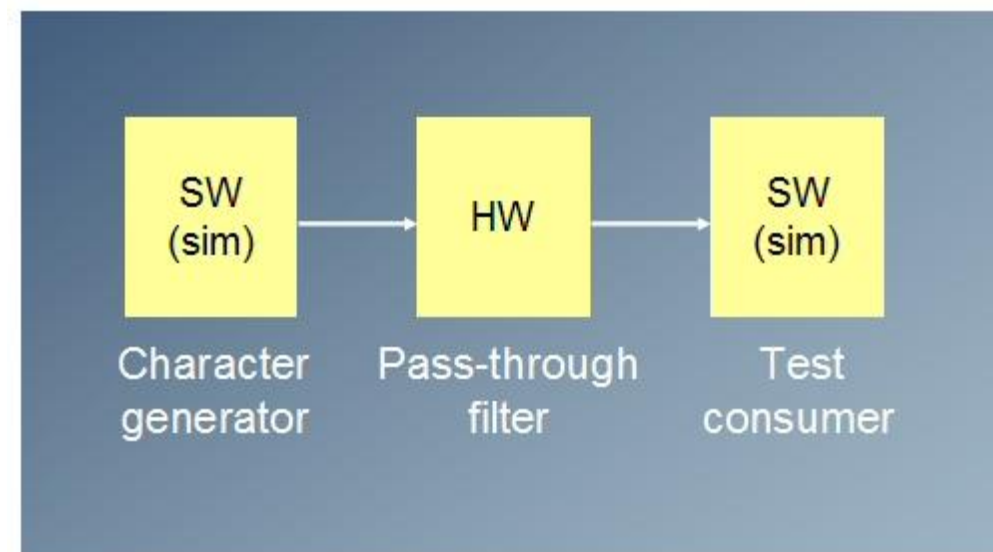
- Xilinx ML507 development board with a Virtex-5 FPGA.
- Platform cable USB II adapter (USB-to-JTAG).
- Licensed copy of version 12.3 of the Xilinx tool suite, including the Xilinx Platform Studio (XPS) and Embedded Development Kit (EDK).
- Licensed copy of version 3.70 of the Impulse CoDeveloper Application Manager and compiler tools.

## Building a Baseline Impulse C Application

1. Open the Impulse CoDeveloper Application Manager (3.70).
2. On the start page, select **“Sample Projects”**, and then click on the **“HelloWorld”** application template under the **“Impulse C Step-by-Step Tutorials”** table section, as shown below.

Impulse C Step-by-Step Tutorials	Description
<a href="#">HelloWorld</a>	Tutorial example #1: Learn the basics of stream communications and desktop simulation.
<a href="#">EdgeDetect</a>	Tutorial example #2: Learn about desktop software simulation using a simple image filter.
<a href="#">EdgeDetect_2</a>	Tutorial example #2, continued: Learn how to use multiple hardware processes for increased parallelism.
<a href="#">Mandelbrot</a>	Tutorial example #3: Learn how to use pipelining and optimization for increased throughput.




1. Examine the block diagram for this application. It is a very simple stream-based application that has three stages: one for producing characters in software, one for reading and forwarding characters in hardware, and then one for reading and displaying characters in software. This stream can be seen below.



1. Before continuing, go to the directory where these project files are located (it should be **C:\Impulse\CoDeveloper3\Examples\Tutorials\HelloWorld\**), and copy the entire directory into a **clean directory with read/write permissions**. This will allow you to build the project files without any problems. Once this step is complete, close the Impulse CoDeveloper Application Manager, navigate to the directory where the project is now located, and re-open the **HelloWorld.icProj** project.
2. The next step is to build and export this application. To do so, first go to **"Project->Options"** from the top menu.
3. Under the **"Generate"** tab in the Options dialog menu, change the Platform Support Package from "Generic (VHDL)" to **"Xilinx Virtex-5 PLB v4.6 (VHDL)"**. This will ensure that the generated hardware modules and software drivers are compatible with the PLB communication interface defined by Xilinx. When finished, click "Apply" and then "OK".
4. To build and export this application, simply go to **"Project->Export Generated Hardware (HDL)"**, followed by **"Project->Export Generated Software"**.
5. Once the compilation process is complete, follow the steps in the **Xilkernel Tutorial** to generate a PowerPC-based system configuration inside Xilinx Platform Studio (XPS), but do not export the project to the Xilinx SDK just yet.

## Adding Custom IP-Cores to an Existing System Configuration

1. Once the base system has been configured we will need to add our new Impulse C IP core to the project. To do this, copy the contents of the **export** directory under the Impulse C project location into the same directory where the XPS project is located. For example, if the Impulse C export code is located at **C:\Test\export\** and the XPS project directory is **C:\Test\System\**, then you must copy the contents from **C:\Test\export\** to **C:\Test\System\**. The contents of the export directory include the custom pcores, drivers, and software files needed to run the Impulse C application.
2. Once this is complete, go to **"Project->Rescan User Repositories"** from the top menu in XPS. This will search for and include new IP cores found within the directory of the project. Once it is finished, you should see the new **plb\_helloworldarch** IP core under the USER directory in the IP catalog, as shown below.

Description	IP Version
[-]  EDK Install	
[-] Analog	
[-] Arithmetic	
[-] Bus and Bridge	
[-] Clock, Reset and Interrupt	
[-] Communication High-Speed	
[-] Communication Low-Speed	
[-] DMA and Timer	
[-] Debug	
[-] FPGA Reconfiguration	
[-] General Purpose IO	
[-] IO Modules	
[-] Interprocessor Communication	
[-] Memory and Memory Controller	
[-] PCI	
[-] Peripheral Controller	
[-] Processor	
[-] Utility	
[-] Project Local PCores	
[-] USER	
[-]  plb_basicstream_arch	1.00.a
[-]  plb_helloworldarch	1.00.a

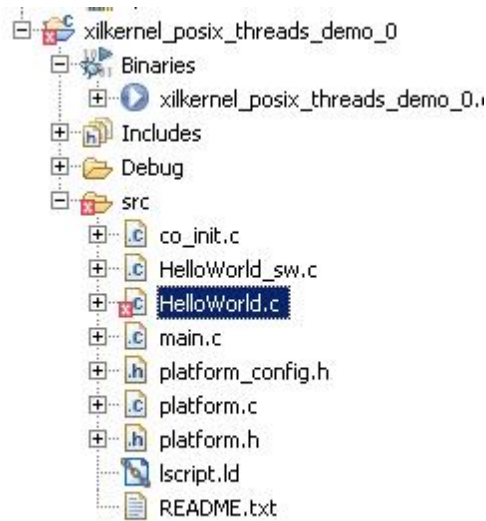
3. Right click on the **plb\_helloworldarch** item and click “Add IP”, followed by “OK” in the dialog menu that appears.
4. Once an instance of this core has been added to the project you need to interface it with the PLB so you can communicate with it from the PowerPC. To do this, go to the “Bus Interfaces” window, expand the **plb\_helloworldarch\_0** instance, and change the connection to **plb\_v46\_0** (the single PLB instance we have on our FPGA), as shown below.

Bus Interfaces			
Ports			
Addresses			
Name	Bus Name	IP Type	IP Version
plb_v46_0		★ plb_v46	1.05.a
ppc440_0		★ ppc440_virt...	1.01.a
xps_bram_if_cntlr_1_bram		★ bram_block	1.00.a
DDR2_SDRAM		★ ppc440mc_d...	3.00.c
xps_bram_if_cntlr_1		★ xps_bram_if...	1.00.b
xps_intc_0		★ xps_intc	2.01.a
jtagppc_cntlr_inst		★ jtagppc_cntlr	2.01.c
plb_helloworldarch_0		★ plb_helloworl...	1.00.a
SPLB	No Connection		
proc_sys_reset_0	No Connection	★ proc_sys_re...	3.00.a
Ethernet_MAC	New Connection	★ xps_etherne...	4.00.a
xps_timer_0	plb_v46_0	★ xps_timer	1.02.a
R5232_Uart_1		★ xps_uartlite	1.01.a
clock_generator_0		★ clock_gener...	4.00.a

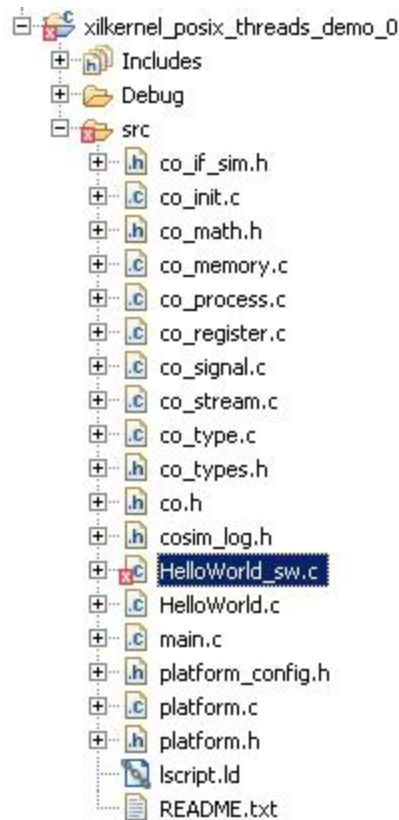
- Now we need to generate the addresses for the system so this new pcore can be referenced from the source code. To do this, go to the “Addresses” tab in the main window and click on “**Generate Addresses**”. You should see the **plb\_helloworldarch\_0** instance added to the **ppc440’s Address Map**.
- Generate the software libraries and BSPs by clicking on “**Software->Generate Libraries and BSPs**”.
- Now you can generate the hardware and export the design to the Xilinx SDK, as done in the Xilkernel tutorial.

## Integrating Impulse C Code into the Xilkernel

- Once the system design has been generated and exported to the Xilinx SDK, follow the steps outlined in the Xilkernel tutorial to generate a Xilkernel BSP image and sample multi-threaded application. We will work from this code to integrate our Impulse C application.
- Copy the three source files **HelloWorld.c**, **HelloWorld\_sw.c**, and **co\_init.c** from the **code** directory under the XPS project directory into the **xilkernel\_posix\_threads\_demo\_0** application instance that you have just generated. This can be done by dragging and dropping these files from Windows Explorer into the **src** directory under the project in the Eclipse side bar.
- Once you copy the files over you will notice many compiler warnings that occur, all of them being originally localized to the **HelloWorld.c** file, as shown below.



4. Before fixing any of these compiler warnings, we need to include the Impulse C driver code so we can actually communicate with the IP core we added to the system. To do this, simply copy all of the C source and header files from the `..\drivers\plb_HelloWorldArch_v1_00_a\src\` directory (where `..` is the absolute path to the XPS project we are using) into the `src` directory under the `xilkernel_posix_threads_demo_0`, as shown below.



5. To fix the compiler errors inside **HelloWorld.c** we need to delete all lines of code that deal with the Impulse C logging mechanism (since we are no longer in the Impulse C environment). In other words, simply delete any line that uses a **cosim** macro. The resulting source code for **HelloWorld.c** should read as follows:

```

////////////////////////////////////
// Copyright (c) 2003, Impulse Accelerated Technologies, Inc.
// All Rights Reserved.
//
// HelloWorld_sw.c: Process to be executed on the target CPU.
//
#define MONITOR

#include "co.h"
#include "cosim_log.h"
#include <stdio.h>

void Producer(co_stream output_stream, co_parameter iparam)
{
    int iterations=(int)iparam;
    int32 i;
    static char HelloWorldString[] = "Hello World!";
    char *p;

    co_stream_open(output_stream, O_WRONLY, CHAR_TYPE);

    for(i=0; i<iterations; i++) {

```

```

        p = HelloWorldString;

        while (*p) {
            co_stream_write(output_stream, p, sizeof(char));
            p++;
        }
        co_stream_close(output_stream);
    }
}

void Consumer(co_stream input_stream)
{
    char c;

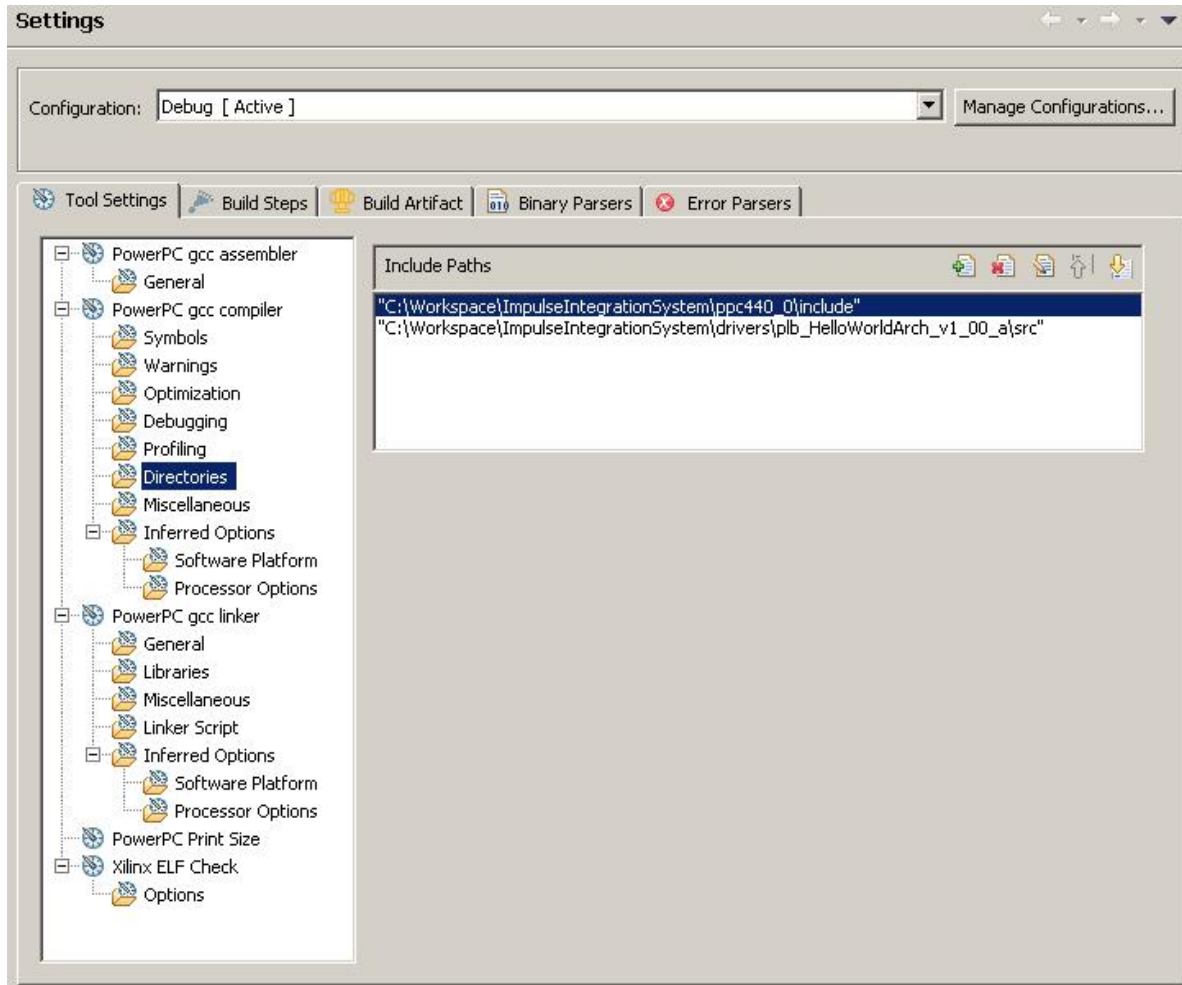
    co_stream_open(input_stream, O_RDONLY, CHAR_TYPE);

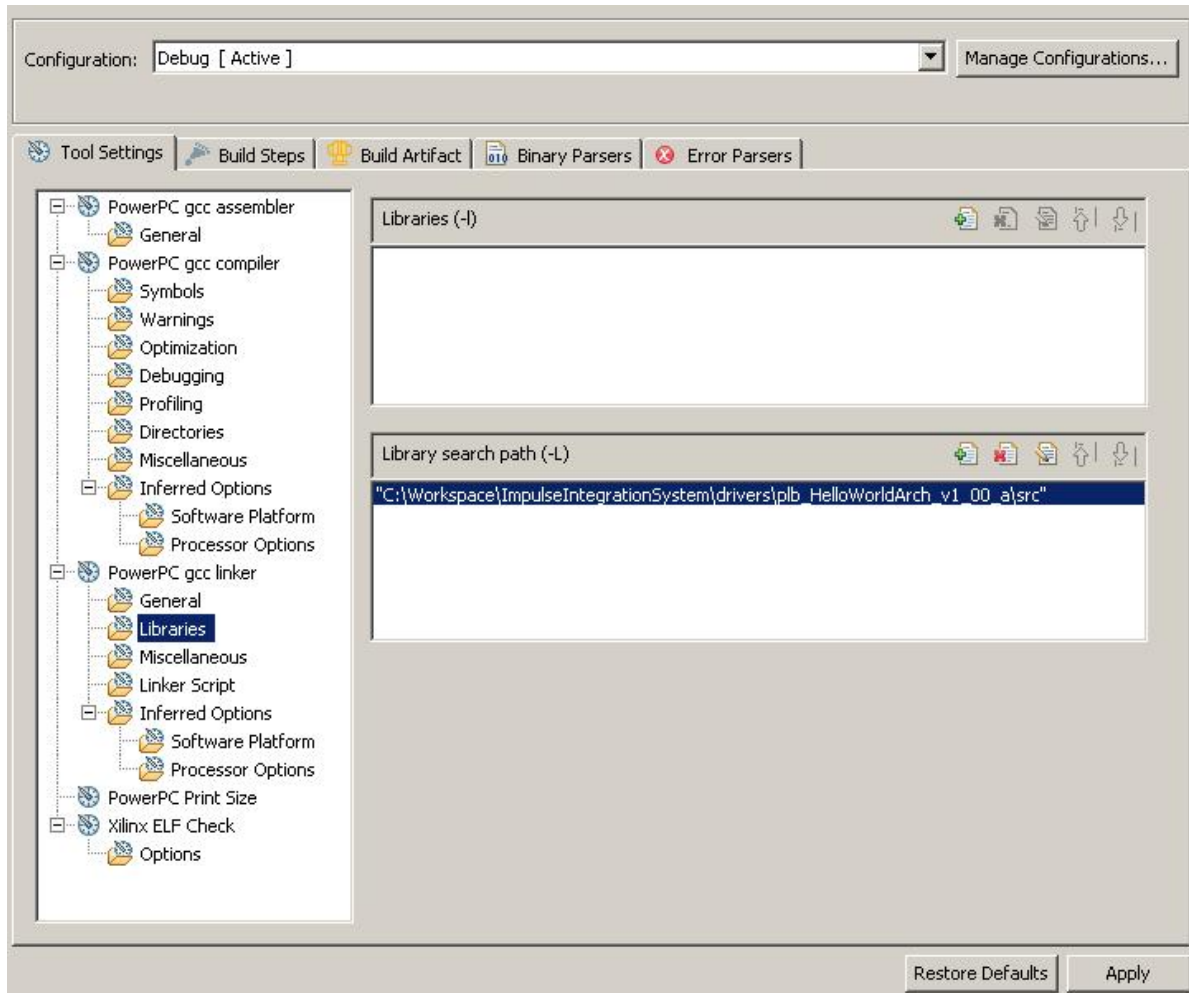
    while ( co_stream_read(input_stream, &c, sizeof(char)) == co_err_none ) {
        printf("Consumer read %c from stream: input_stream\n", c);
    }
    co_stream_close(input_stream);
}

```

6. Once this is complete, you will notice another compiler error in **co\_init.c** file, referring to the fact that the **XPAR\_PLB\_HELLOWORLDARCH\_0\_BASEADDR** macro is not found. This is an address that is automatically generated by XPS when we export the hardware, and is located in the **xparameters.h** file. To fix this error, we need to include the appropriate Xilinx library files into the project. Use the following images to see which directories need to be added to the project build path (note that these screens are reached from the **C/C++ Build Settings** menu for the SDK project).







When you are finished, click on “Apply” to rebuild the project with these new settings.

7. The next step is to remove the second main function that was originally included in the Impulse C application. Inside **HelloWorld.c**, comment out the entire main routine. There are three lines of code that are important to us, however, and we need to relocate them to the actual main routine. These lines are shown below.

```
co_architecture my_arch;
my_arch = co_initialize(); // Removed the iterations parameter!
co_execute(my_arch);
```

This code needs to be moved into the **main.c** file so that the main function now reads as:

```
#include "co.h" // Needed for the co_architecture macro!

/* Functions */
```

```

int main()
{
    int i;

    init_platform();

    /* Taken from the old Impulse C main routine */
    co_architecture my_arch;
    my_arch = co_initialize(); // The iterations parameter was removed!
    // co_execute() was removed because we don't want to initialize the threads here

    /* Assign random data to the input array */

    for (i = 0; i < DATA_SIZE; i++)
        input_data[i] = i + 1;

    /* Initialize xilkernel */
    xilkernel_init();

    /* Create the master thread */
    xmk_add_static_thread(master_thread, 0);

    /* Start the kernel */
    xilkernel_start();

    //....

```

8. The next step is to wrap the Impulse C processes by POSIX threads, which means we need to expose these function prototypes to the main thread function, create structs to wrap their parameters, and then create the threads to run these processes. The first step is to add a header file for the HelloWorld.c sample application named **HelloWorld.h**. The header file should have the following code:

```

void Producer(co_stream output_stream); // note that the co_parameter was removed
void Consumer(co_stream input_stream);

```

Note that the `co_parameter` parameter for the Producer routine was removed. This was done to simplify the conversion process. It could have been left in if both of the parameters were wrapped in a single structure that was passed to the POSIX thread.

Now, make sure **HelloWorld.h** is included in the **HelloWorld\_sw.c** file, and also that the **co\_parameter** **iparam** parameter is removed from the Producer function parameter list. Since we removed this parameter, the first line within this function also needs to change from:

```
int iterations = (int)iparam;
```

to

```
int iterations = 10;
```

9. Now go to the master thread function (`master_thread`) in **main.c** and change the following remove the following block of code:

```

/* This is how we can join with a thread and reclaim its return value */
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

for (i = 0; i < N_THREADS; i++) {
    args[i] = i;
    ret = pthread_create (&worker[i], &attr, (void*)partial_sum_worker,
                        &args[i]);

    if (ret != 0) {
        xil_printf("Xilkernel Demo: ERROR (%d) launching worker thread" \
                    "%d.\r\n", ret, i);
        break;
    }
}

if (ret) {
    xil_printf("Xilkernel Demo: Master Thread Terminating.\r\n");
    return (void*)-1;
}

final_result = 0;
for (i = 0; i < N_THREADS; i++) {
    ret = pthread_join(worker[i], (void*)&result);
    final_result += (int)result;
    xil_printf("Xilkernel Demo: Collected result (%d) from worker: %d.\r\n",
                (int)result, i);
}

xil_printf("Xilkernel Demo: Result computed by worker threads = %d.\r\n",
            final_result);

```

Replace this code with the following:

```

// Variable declaration at the beginning of the function
pthread_t pid;
pthread_t cid;

// Old code block location
pthread_create(&pid, NULL, (void*)Producer, NULL);
pthread_create(&cid, NULL, (void*)Consumer, NULL);
pthread_join(pid, NULL);
pthread_join(cid, NULL);

```

With the addition of this code we also need to **include the HelloWorld.h** header file.

10. Now, build and run the application on the ML507. If you re-routed the UART output to the SDK console, you should see the following output:

```

Consumer read H from stream: input_stream
Consumer read e from stream: input_stream
Consumer read l from stream: input_stream
Consumer read l from stream: input_stream
Consumer read o from stream: input_stream
Consumer read  from stream: input_stream
Consumer read W from stream: input_stream
//...

```

## **Issues and Troubleshooting**

All issues, questions, and concerns can be directed to Christopher Wood at [caw4567@rit.edu](mailto:caw4567@rit.edu).