Getting Started with the Hardware and Software Platform

OBJECTIVES

- Learn about the DE2i-150 board and its components.
- Learn about the operating system: Ubuntu.
- Write, compile and execute simple C applications.

FUNDAMENTALS

REPOSITORY EXAMPLES

- Refer to the Tutorial: Embedded Intel for a comprehensive list of examples.

TERASIC DE2I-150 BOARD

- Refer to the board website or the Tutorial: Embedded Intel for User Manuals and Guides. We mention the information that is relevant for the Microprocessor system (these documents heavily focus on the FPGA system):
  - ✓ DE2i-150 Quick Start Guide: To quickly connect the power, mouse, keyboard, display for the Intel® Atom™.
  - ✓ DE2i-150 Getting Started Guide: Details on powering the Board.
  - ✓ DE2i-150 FPGA System User Manual: Installation of WiFi Module and Antenna on DE2i-150.
  - ✓ DE2i-150 Windows 7 User Manual: Boot DE2i-150 with a Bootable USB Flash Drive
  - ✓ Installing Ubuntu OS on the DE2i-150: Some tips for Ubuntu OS installation. * We use a USB flash drive with an image.

COMPONENTS

- This board includes an Intel® Atom™ processor and an FPGA. Each device includes a range of peripherals connected to them. These two devices are connected via PCI Express interface. Table I gives a brief overview of the platform's features.

TABLE I. BRIEF OVERVIEW OF THE FEATURES OF THE TERASIC DE2I-150 FPGA DEVELOPMENT KIT.

Microprocessor	**FPGA System**
Intel® Atom™ Dual Core Processor N2600, 1.6 GHz	Cyclone IV EP4CGX150DF31 FPGA
1 MB L2 cache (512 KB per core)	Three 50 MHz oscillator clock inputs
Intel® NM10 Express Chipset	128 MB SDRAM
2 GB DDR3 SDRAM	64 MB Flash Memory
64 GB SSD	4 MB SSRAM
HDMI, VGA output	SD Card: SPI and 4-bit SD mode
4 USB ports	Display: VGA, LCD
Gigabit Ethernet, 802.11 b/g/n	Ethernet port

OPERATING SYSTEM AND PROGRAMMING LANGUAGE SUPPORT

- The SSD (hard drive) enclosed with the Development Kit has been pre-loaded with Yocto Operating System; this is from the manufacturer (Terasic). Thus, Yocto Linux is the natural choice for the DE2i-150 FPGA Development Kit. However, as we spent more time on the Yocto Linux platform, we ran into several issues:
  - ✓ Yocto Linux slowdown: After working on it for a little while (~40 mins), the OS becomes unacceptable slow (even on the command line) to the point that it was impractical to work on it. We suspect that there is an issue with the generation of this specific Yocto Linux for the DE2i-150 Development Kit.
  - ✓ Yocto Linux corrupted downloadable image: You might try to re-install the Yocto image provided by the manufacturer (Terasic). However, after extensive tryouts (installation on different boards, independent image downloads), we suspect the downloadable Yocto image to be corrupted. This was made evident in various ways: i) installation would not complete, ii) completed installation that resulted in unacceptable outcomes: system would not start, GUI would load up incorrectly, system would start and then freeze within minutes. By installing this image, any targeted board will be rendered unusable.

- Faced with this situation, we explored two alternatives:
  - ✓ **Windows 7**: This is the first operating system we tried as the manufacturer (Terasic) already provides extensive documentation (*DE2i-150 Windows 7 User Manual*). However, Microsoft has recently (Jan. 2020) discontinued Windows 7 and it is no longer possible to download the installation image.
  - ✓ **Ubuntu Linux**: The Intel® document (*Installing Ubuntu OS on the DE2i-150*) provides details regarding Ubuntu Linux installation on the Embedded Kit. Thus, we installed Ubuntu Linux 12.04.4 on the DE2i-150 Development Kits. The

installation turned out to be way smoother than the document predicted. It has worked almost flawlessly: drivers were working fine, our application run fine, and we had no issues installing specific packages (like TBB which provide difficult under Yocto Linux). Moreover, so far, we have not experienced any slowdowns.
  - We note that you might need to create your own FPGA PCIe driver to work with the FPGA and OS on the DE2i-150 Board (drivers are provided for Yocto).

- As a result, we have decided to use **Ubuntu Linux 12.04.4** on the DE2i-150 Development board as our hardware/software platform in this course.

- The Terasic Board you receive comes pre-installed with Ubuntu 12.04.4 and the proper version of the *gcc* and *g++* compilers (5.4.1). However, we recognize that during board usage, you might run into unexpected problems. In this case, the following is the list of steps you need to complete to return the Terasic Board to the original Ubuntu settings.

OPERATING SYSTEM (UBUNTU) INSTALLATION

- Install Ubuntu 12.04.4
  - ✓ Create a bootable USB stick with Rufus (or unibootd). The Ubuntu OS image (.iso) is available at the Ubuntu repository. Select the PC (Intel x86) desktop CD image: `ubuntu-12.04.4-desktop-i386.iso`.
    - You can also download the .iso file from here.
  - ✓ The following steps are very similar to the ones mentioned in the *DE2i-150 Windows 7 User Manual* (page 10).
    - Plug in your Ubuntu bootable USB flash drive to one of the USB ports on DE2i-150 Board.
    - When the BIOS logo is shown, press F10 on the keyboard to enter the boot menu.
    - Select the USB flash drive that is plugged into the board (e.g.: *Sandisk Cruzer Spark*) and press Enter to set the BIOS of DE2i-150 to boot from the USB device.
      - DO NOT select the EFI USB drive options.
    - The DE2i-150 will reboot and load the Ubuntu OS loading screen.
  - ✓ Follow the instructions to install Ubuntu (if Yocto is installed, select 'Replace unknown Linux Distribution with Ubuntu').
  - ✓ You can set up any user and password as desired (username: student, computer name: ece4900@atom).
- Connect to Wifi. This should be automatic. If you run into problems, use the Wired Ethernet connection.
  - ✓ It might ask to upgrade to 14.04 (due to driver issue with display, select NO if possible). Note that it is unclear whether Ubuntu 14.04 would work fine in the Terasic Board.

PROGRAMMING LANGUAGES SUPPORT

- In these tutorials, we use the GCC (GNU Compiler Collection) that supports C and C++. We also use the Intel® TBB library and POSIX threads (pthreads).
  - ✓ By default, Ubuntu 12.04.4 comes pre-installed with *gcc* 4.6.3 version.

- Install the *g++* compiler:
  - ✓ `sudo apt-get update` → You should always run it before installing or upgrading packages on the command-line (unless you've run it very recently). It fetches information about what packages are available in what versions from where.
  - ✓ `sudo apt-get install g++`
    - Note that this will install the 4.6.3 version (which is the newest that ubuntu 12.04 supports automatically)

- Install Intel® Threading Building Blocks (TBB) Library:
  - ✓ This will allow the C++ compiler to work with TBB.
  - ✓ To install TBB, go to the Terminal and type:
    - `sudo apt-get install libtbb-dev`

- Some TBB features (e.g.: lambda expressions for *parallel_for, parallel_reduce, parallel_pipeline*) are not supported by the *g++* 4.6.3 version. We require the modifier `std=c++11` (newer version of C++) at compilation. We thus need to install a newer version of the C/C++ compilers.
  - ✓ To see the compiler version: `g++ --version , or gcc ---version` (it should say 4.6.3)
  - ✓ We can install newer versions (4.7, 4.8, 5.4.1). As this is not supported automatically by Ubuntu 12.04.4, the following steps need to be followed to install 5.4.1 version:
    - `sudo apt-get purge g++ → (optional)`
    - `sudo add-apt-repository ppa:ubuntu-toolchain-r/test`
    - `sudo apt-get update`
    - `sudo apt-get install gcc-5 g++-5`
    - `sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-5 60 --slave /usr/bin/g++ g++ /usr/bin/g++-5`
  - ✓ After these steps are completed, TBB will work when the `std=c++11` modifier is used at compilation.
  - ✓ To verify the new version, type: `g++ --version`. It should list the version as 5.4.1.

ACTIVITIES

FIRST ACTIVITY: BOARD SETUP AND BASIC UTILITIES

- The DE2i-150 FPGA Development Kit is a full-featured computer system that fuses together the world of high-performance processing (Intel® N2600 processor) and unbelievably high configurability (Altera Cyclone IV GX FPGA). Fig. 1 and Fig. 2 depict this board (front side and back side) indicating the key components.
- The board (as delivered to you) comes pre-installed with Ubuntu 12.04.4 version as well as with the 5.4.1 version of the *gcc* and *g++* compilers.

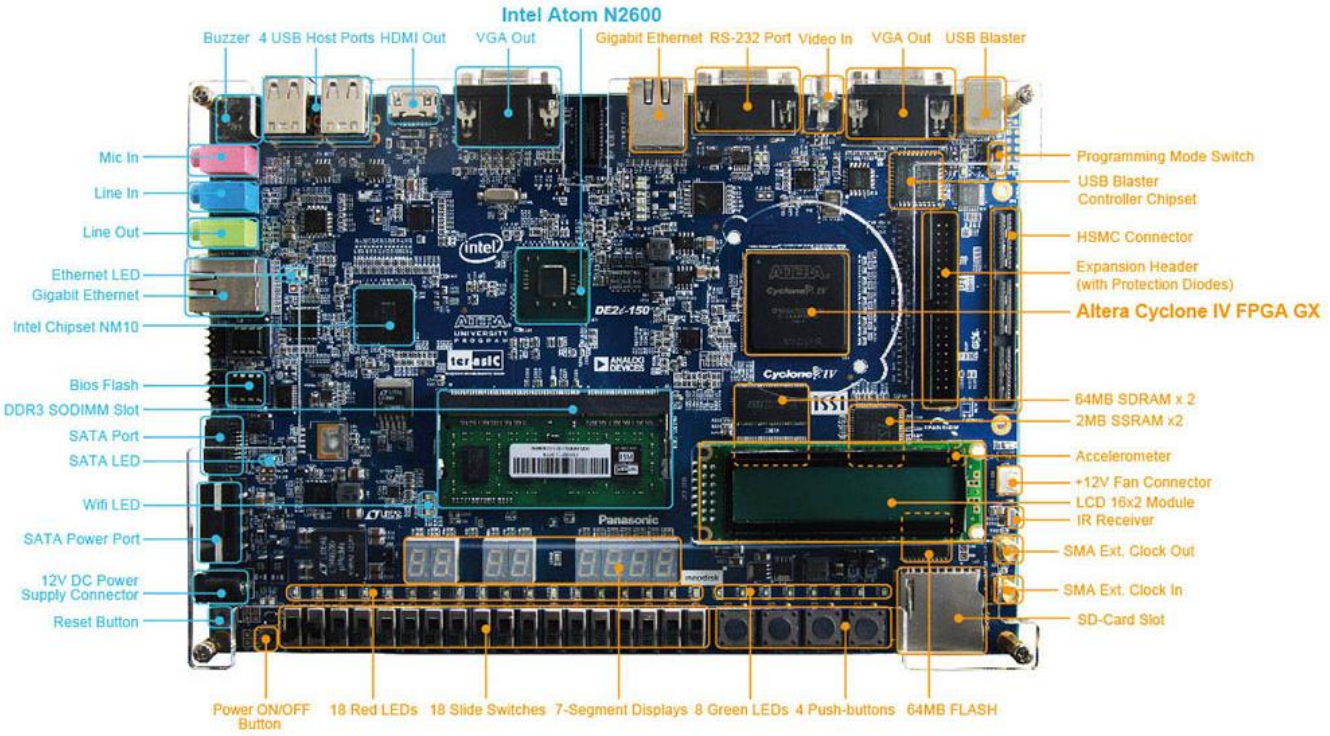

Figure 1. Terasic DE2i-150 FPGA Development Kit: Front Picture with components indicated. Source: *Terasic website*

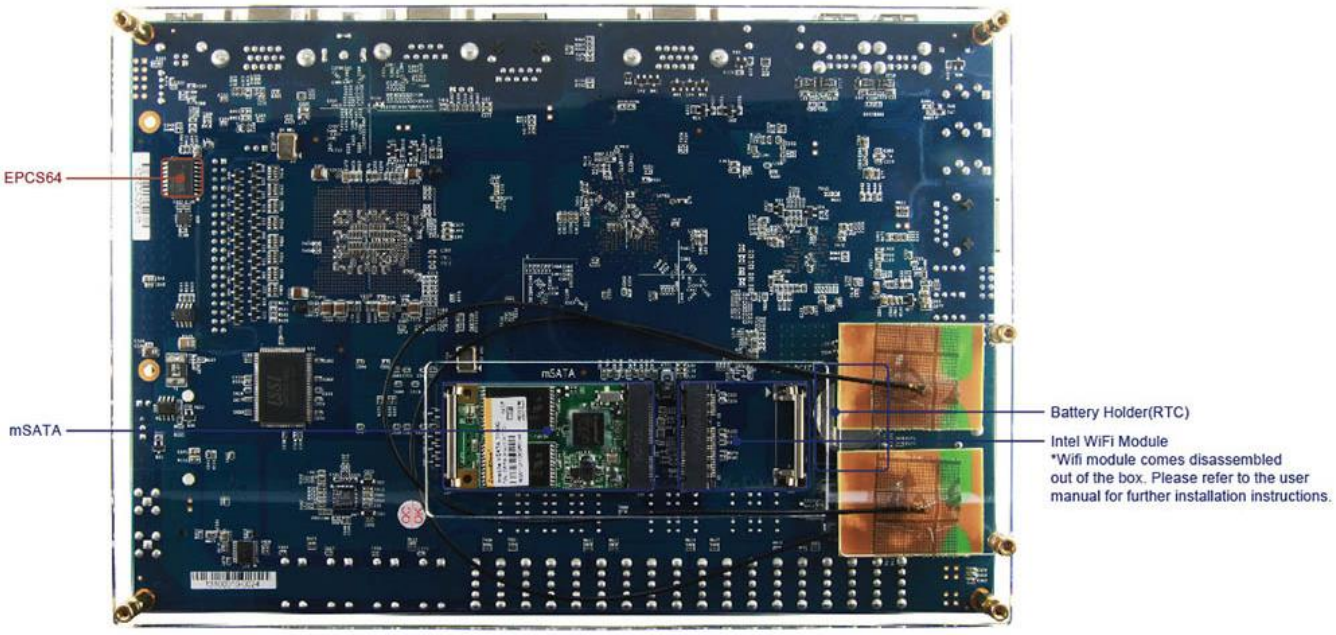

Figure 2. Terasic DE2i-150 FPGA Development Kit: Back Picture with components indicated. Source: *Terasic website*

BOARD SETUP

- Connect the monitor (VGA or HDMI) as well as the keyboard and mouse.
  - ✓ Refer to the *DE2i-150 Quick Start Guide* (page 2) for a useful illustration.
- Install the Wi-Fi module and Antenna (see *DE2i-150 FPGA System User Manual*, page 108). * In the board you receive, this step will be already completed for you.

Powering up the DE2i-150 Board

- Connect the provided power cord to the power supply and plug the cord into a power outlet.
- Connect the supplied 12V DE2i-150 power adapter to the power connect (J1) on the DE2i-150 board. At this point, you should see the 12 V LED (D33) turn on.
  - ✓ Be careful not to plug the power adapter into the SATA power connector (see *DE2i-150 Getting Started Guide*, page 7).
- Click the **Power ON/OFF Button** (lower right corner) to boot the OS.
- The board should power on, emitting some beeps to indicate a successful load of the BIOS.
- Once Ubuntu screen loads, enter the login information: User: student, password: (sent to you via email, you can change it).

BASIC UTILITIES

- Even if you do not have experience with Linux, we will only use a few applications in this course:
  - ✓ *Files* application (the counterpart of the File Explorer in Windows).
    - You should know how to navigate the file system graphically, as well as create/delete/open folders and files.
  - ✓ *Terminal*: This will be heavily used to compile and execute code.
    - Terminal commands that you should explore: `sudo, mkdir, cd, rm, ls, pwd`.
    - You should also know how to navigate the file system via the Terminal.
  - ✓ Text Editor (*Gedit, Kate*). Kate is recommended (it includes line number).
    - You might need to install Kate (`sudo apt-get install -y kate`)
    - Feel free to install your favorite Text Editor or Integrated Development Environment.

- Once your system is up and running, you can transfer information via the Internet and a USB stick.

File System

- The contents of the SSD is known as its file system, and it is split into multiple sections each with a particular purpose. In Linux, all drives, files, folder, and devices appear as a branch beneath what is known as the *root file system* (represented as '/'). Some of these folders are areas of the SSD for storing files, while others are virtual directories for accessing different portions of the OS or hardware.
  - ✓ Open the *Files* utility (available on the left pane) and navigate to Computer. This will show you the Root folder. Table II describes the most common directories.
  - ✓ Your work files should be stored in the directory `/home/<user name>`.
  - ✓ At this point, you should be ready to create new folder and files.

TABLE II. KEY VISIBLE DIRECTORIES IN THE UBUNTU LINUX DISTRIBUTION

/		
	boot	Contains the *Linux kernel* and other packages needed to start the system
	bin	Operating system-related binary files, like those required to run the GUI (Graphical User Interface) are stored here.
	dev	Virtual directory (doesn't actually exist on the SSD). All devices connected to the system (e.g., storage devices, sound card, HDMI port) can be accessed from this directory.
	etc	Stores miscellaneous configuration files, including the list of users and their encrypted passwords.
	home	Each user gets a subdirectory beneath this directory to store all their personal files.
	lib	Storage space for libraries, which are shared pieces of code required by numerous different applications
	media	Special directory for removable storage devices, like USB memory sticks or external drives.
	mnt	This directory is used to manually mount (load) storage devices, such as external hard drives.
	opt	Stores optional software not part of the operating system itself. It you install new software, it will usually go here.
	proc	Virtual directory, containing information about running programs which are know in Linux as *processes*.
	sbin	Stores special binary files, primarily used by the root (superuse) account for system maintenance.
	sys	Directory where special operating system files are stored
	tmp	Temporary files are stored here automatically.
	usr	This folder provides storage for user-accessible programs.
	var	Virtual directory. Programs use it to store changing values or *variables*.

SECOND ACTIVITY: FIBONACCI SERIES

- Compute the elements of the following series for $n > 1$:

$$F_n = F_{n-1} + F_{n-2},\ F_0 = 0, F_1 = 1$$

- A straightforward implementation in C is shown below:

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

int main (int argc, char **argv) {
  int i, n;
  int *f;

  // Reading Input arguments
  //   argc: # of arguments.
  //   argv: array of strings holding the arguments. argv[0]: ./fibo, argv[1]: n
  if (argc != 2) {
    printf ("Warning: Usage: %s n\n", argv[0]);
    printf ("Using n = 10 as default\n");
    n = 10;
  }
  else n = atoi(argv[1]);

  if (n < 1) { printf ("Incorrect number of elements\n"); return -1; }

  f = (int *) calloc (n, sizeof(int)); // Dynamic memory allocation

  f[0] = 0; f[1] = 1;

  for (i = 2; i < n; i++)  f[i] = f[i-1] + f[i-2];

  for (i = 0; i < n; i++) printf ("F[%d] = %d\n", i, f[i]);

  free(f);
  return 0;
}
```

  - ✓ Note that by using int, we are restricting our results to -2^{31} to $2^{31} - 1$.

- Create this file (use Gedit or Kate).
- Application file: `fibo.c`

- Compile this code:

  `gcc fibo.c -Wall -o fibo` ↵
  - ✓ The `-Wall` option will check for all kinds of warnings like unused variables (recommended practice).

- Execute this application:

  `./fibo` ↵
  - ✓ By default, it would compute Fibonacci numbers from F_0 to F_9.
  - ✓ This application allows the specification of a parameter (the largest desired index of the Fibonacci series):
    `./fibo n` // This computes F_0 to F_{n-1}.
  - ✓ For example, you can execute:
    `./fibo 20` // This computes F_0 to F_{19}.

THIRD ACTIVITY (SAXPY)

- Implement SAXPY (Scaled Vector Addition):

$$\vec{y} \leftarrow a\vec{x} + \vec{y}$$

- A serial implementation is shown below. We use three different files:
  - ✓ `saxpy_ex.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>  // for gettimeofday()
#include "saxpy_fun.h" // list the functions in .h file, implement the functions on saxpy_fun.c
#define N 20

int main() {
    float y[N], x[N];
    float a = 1.618;
```

```
    struct timeval start, end; // structure with us and s. Resolution: 1 us.
    long t_us;

    // Initializing input vectors
    saxpy_init (x, y, N);
    saxpy_print (y, N); saxpy_print (x, N);

    gettimeofday(&start, NULL);  // Measuring processing time: start

    saxpy (a,x,y, N);

    gettimeofday(&end, NULL);   // Measuring processing time: start
    // gettimeofday: returns current time. So, when the secs increment, the us resets to 0.

    saxpy_print (y, N); // printing resulting vector 'y'

    printf ("start: %ld us\n", start.tv_usec); // start.tv_sec
    printf ("end: %ld us\n", end.tv_usec);    // end.tv_sec;
    t_us = (end.tv_sec - start.tv_sec)*1000000 + end.tv_usec - start.tv_usec;
    printf ("Elapsed time: %ld us\n", t_us);

   return 0;
}
```

- ✓ saxpy_fun.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "saxpy_fun.h"

void saxpy (float a, float *x, float *y, int SIZE) {
    int i;
    for (i = 0; i < SIZE; i++) {
        y[i] = a*x[i] + y[i]; }
}

void saxpy_init (float *x, float *y, int SIZE) {
     int i;
     for (i = 0; i < SIZE; i++) {
        x[i] = sin(i*3.416/SIZE);
        y[i] = cos(i*3.416/SIZE);
     }
}

void saxpy_print (float *y , int SIZE) {
   int i;

   printf ("Vector is: \n");
   for (i = 0; i < SIZE; i++) printf("y[%d] = %4.4f\n", i, y[i]);

}
```

- ✓ saxpy_fun.h

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

void saxpy (float a, float *x, float *y, int SIZE);
void saxpy_init (float *x, float *y, int SIZE);
void saxpy_print (float *y , int SIZE);
```

- Create this file (use Gedit or Kate).
- Application files: `saxpy_ex.c, saxpy_fun.h, saxpy_fun.c`
  - ✓ Note that we measure the processing time (us) using `gettimeofday()`.
  - ✓ This example includes a main file (`saxpy_ex.c`), a header (`.h`) file for function declarations, and another `.c` file for function definitions.

- Compile this code:
  - ✓ First method:

    `gcc -Wall saxpy_ex.c saxpy_fun.c -o saxpy_ex -lm` ↵
    - The `-lm` modifier allows for the `math.h` library to be used.

  - ✓ Second method: Using a makefile

Makefile:

```
# Compiler/linker setup -----------------------------------------------------
# Linux-specific flags.
PLATFORM = linux
CC       = gcc
CFLAGS   = -O3 -Wall
OSLIBS   =
LDFLAGS  =

OBJS = saxpy_ex
all: $(OBJS)

saxpy_ex: saxpy_ex.c saxpy_fun.o
  $(CC) $(CFLAGS) -o saxpy_ex saxpy_ex.c saxpy_fun.o -lm

# library
saxpy_fun.o: saxpy_fun.c saxpy_fun.h
  $(CC) $(CFLAGS) -c saxpy_fun.c -lm

# Maintenance and stuff -----------------------------------------------------
clean:
  rm -f $(OBJS) *.o core
```

- Once the Makefile is created, you can do:
  - `make all`: all OBJS are compiled and executable is available
  - `make clean`: cleans indicated files in the `Makefile`.

- Execute this application (Fig. 3 depicts execution on the Terasic board):

  `./saxpy_ex` ↵

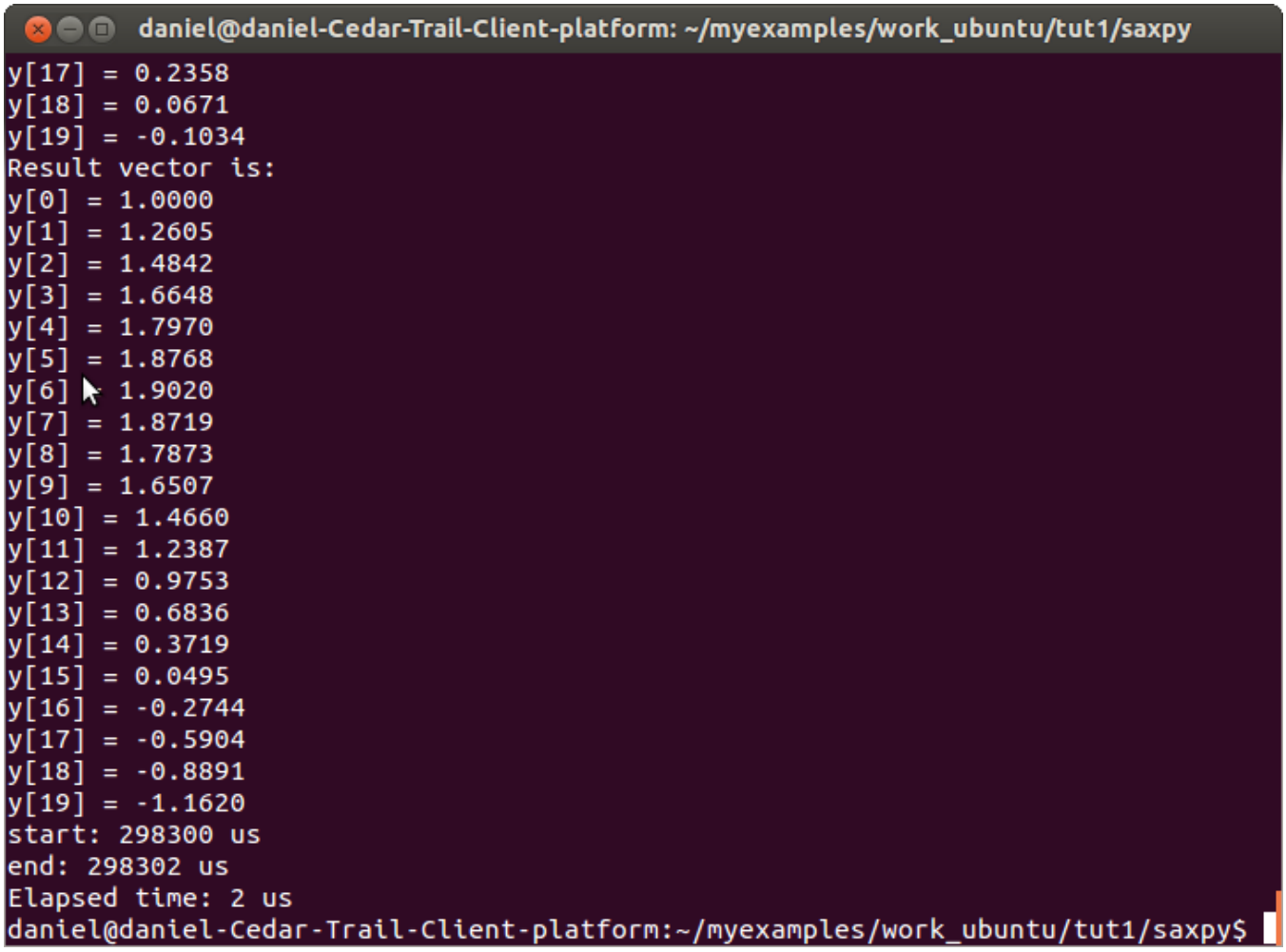

Figure 3. SAXPY Execution on Terasic DE2i-150 FPGA Development Kit.

FOURTH ACTIVITY (BUBBLE SORT)

- Sort an array of numbers via the bubble sort algorithm:

```
procedure bubbleSort (A: list)
   n = length(A)
   p = n-1
   for i = 0 to n-2 do
      for j = 0 to p-1 do
         if A[j] > A[j+1] then
            swap (A[j], A[j+1])
         end if
      end for
      p = p-1
   end for
end procedure
```

For every i, compare (and swap) pairs. After each iteration one less element (the last one) is needed to be compared.

i = 0: pairs (0,1), (1,2), (2,3), (3,4), (4,5)
i = 1: pairs (0,1), (1,2), (2,3), (3,4)
i = 2: pairs (0,1), (1,2), (2,3)
i = 3: pairs (0,1), (1,2)
i = 4: pairs (0,1)

n=6	64	35	12	22	11	90
i=0,p=5:	35	12	22	11	64	90
i=1,p=4:	12	22	11	35	64	90
i=2,p=3:	12	11	22	35	64	90
i=3,p=2:	11	12	22	35	64	90
i=4,p=1:	11	12	22	35	64	90

- A serial implementation is shown below. We use three different files:
  - ✓ bsort.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h> // for gettimeofday()
#include "bsort_fun.h"

int main() {
   struct timeval start, end; // structure with us and s. Resolution: 1 us.
   long t_us;

   int arr[] = {17, 65, 12, 41, 9, 31, 87, 28}; // arbitrary array defined

   // Getting the size of the array:
   int n = sizeof(arr)/sizeof(arr[0]);
   printf ("Total # of bytes in array: %ld\n", sizeof(arr));
   printf ("# of bytes per array element: %ld\n", sizeof(arr[0]));
   printf ("n = %d\n", n);

   printf("Original array: \n"); printArray(arr, n);

   // Measuring processing time:
   gettimeofday(&start, NULL);
   bubbleSort(arr, n);
   gettimeofday(&end, NULL);

   printf("Sorted array: \n");  printArray(arr, n);

   printf ("start: %ld us\n", start.tv_usec); // start.tv_sec
   printf ("end: %ld us\n", end.tv_usec);    // end.tv_sec;
   t_us = (end.tv_sec - start.tv_sec)*1000000 + end.tv_usec - start.tv_usec;
   printf ("Elapsed time: %ld us\n", t_us);

   return 0;
}
```

  - ✓ bsort_fun.c

```c
#include <stdlib.h>
#include <stdio.h>
#include "bsort_fun.h"

void swap(int *xp, int *yp) {
    int tmp = *xp; // value pointed by pointer xp
    *xp = *yp;
    *yp = tmp;
}

void bubbleSort(int *arr, int n) {
   int i, j;
   int p = n-1;

   for (i = 0; i < n-1; i++) // counter for all the passes: 0 to (n-2)th pass
       for (j = 0; j < p; j++)
           if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
       p = p -1;
}
```

```
void printArray(int *arr, int n) {
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

- ✓ bsort_fun.h

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

void swap(int *xp, int *yp);
void bubbleSort(int *arr, int n);
void printArray(int *arr, int n);
```

- Create this file (use Gedit or Kate).
- Application files: `bsort.c, bsort_fun.h, bsort_fun.c`
  - ✓ Note that we measure the processing time (us) using `gettimeofday()`.
  - ✓ This example includes a main file (`bsort.c`), a header (`.h`) file for function declarations, and another `.c` file for function definitions.

- Compile this code:
  - ✓ First method:

```
gcc -Wall bsort.c bsort_fun.c -o bsort ↵
```

  - ✓ Second method: Using a makefile

Makefile:

```
# Compiler/linker setup -----------------------------------------------------
# Linux-specific flags.
PLATFORM = linux
CC       = gcc
CFLAGS   = -O3 -Wall
OSLIBS   =
LDFLAGS  =

OBJS = bsort
all: $(OBJS)

bsort: bsort.c bsort_fun.o
  $(CC) $(CFLAGS) -o bsort bsort.c bsort_fun.o

# library
bsort_fun.o: bsort_fun.c bsort_fun.h
  $(CC) $(CFLAGS) -c bsort_fun.c

# Maintenance and stuff -----------------------------------------------------
clean:
  rm -f $(OBJS) *.o core
```

    - Once the Makefile is created, you can do:
      - `make all`: all OBJS are compiled and executable is available
      - `make clean`: cleans indicated files in the `Makefile`.

- Execute this application:

```
./bsort ↵
```