ECE 4999: INDEPENDENT STUDY

MANISH PATEL (mkp53)

SYED TAHMID MAHBUB (sm893)

CORNELL UNIVERSITY

Under the supervision of: Professor Bruce R. Land

Fall 2015

Table of Contents

1 Introduction

The DE1-SoC development board from Terasic uses the Altera Cyclone V SoC (System on Chip) contains both an FPGA and a dual-core ARM Cortex A9 Hard Processor System (HPS). The initial goal of the independent study was to use the SoC to develop a math co-processor, specifically a differential equation solver, using the FPGA as the compute module and the ARM as the control processor, while employing the ARM-to-FPGA communication bus to transfer data within the SoC. However, given the struggles with development tools, the project goals were scaled down to getting a more general familiarity with the system and development tools, as well as developing a parallelized ordinary differential equations solver. This was seen as a reasonable milestone since it would allow development towards the original goal with continued effort.

2 Running Linux on the DE1-SoC

The DE1-SoC supports running Linux, which presents several advantages in software development. A first major advantage is that of running standard Linux applications on the SoC that allows use as a general computer, while running other specific embedded tasks. Additionally, software can be developed for Linux in a variety of languages. One overhead in development, as compared to bare-metal development on a microcontroller, is that programs need to map physical addresses of peripherals into virtual address space to access the peripherals.

The DE1-SoC Resources page presents the MicroSD Card Images for 4 Linux installations:

1. Linux Console – this is the simplest of the options; it runs the Yocto distribution and the console is available over a serial port with USB interface (FTDI USB-to-serial translator on the DE1-SoC board). With an Ethernet connection, it is also possible to SSH into the DE1.
2. Linux Console with framebuffer – this also runs Yocto but forwards the console onto the VGA output of the board so that it is available on an external monitor, making it possible to connect an external keyboard to the board and use it as a standalone computer. The framebuffer is implemented with the FPGA.[1]
3. Linux LXDE Desktop – this runs with the LXDE desktop environment, presenting a full GUI to use. It is possible to connect a keyboard and mouse to the DE1-SoC and use it as a standard computer.
4. Linux Ubuntu Desktop – this runs the Ubuntu desktop environment instead of LXDE.

The desired Linux version was downloaded and an SD card image was formed by using the software Win32 Disk Imager[2]. We used an 8GB microSDHC card. Win32 Disk Imager worked fine on both Windows 7 and Windows 10. Before using Win32 Disk Imager, using the dd command on Ubuntu was considered, but given the risk of deleting the primary Linux partition upon incorrect device specification, this was not used.

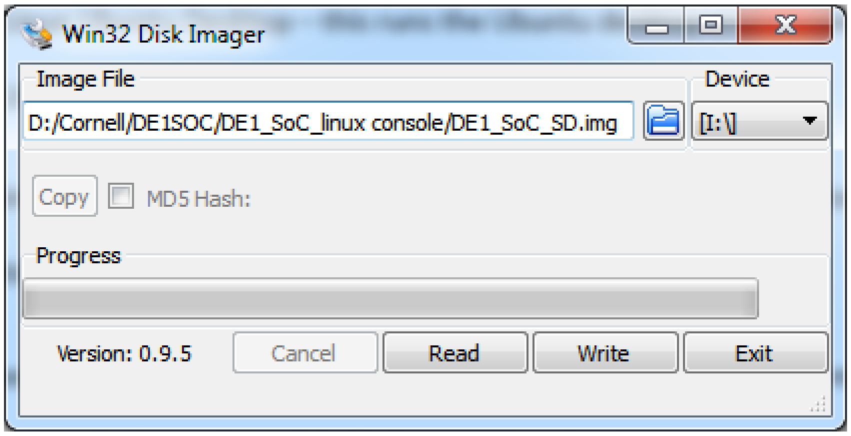


Image File

D:/Cornell/DE1SOC/DE1_SoC_linux console/DE1_SoC_SD.img

Device

[I:\]

Figure 1-Win32 Disk Imager example

It is necessary to set the value of MSEL (JTAG chain) using a DIP switch mounted on the underside of the DE1-SoC board. It is labelled as SW10 and as MSEL, as shown in the image below[3].

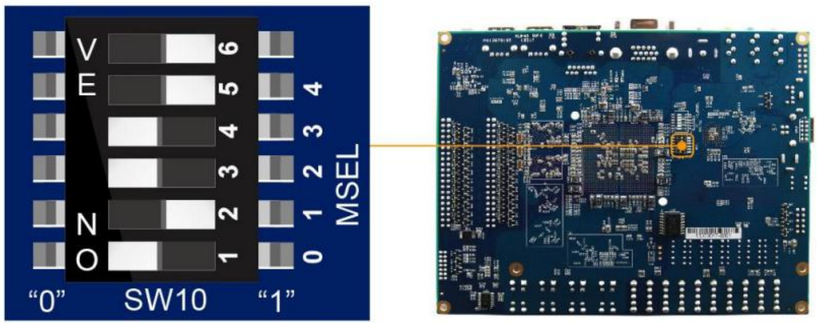


Figure 2-MSEL DIP Switch location

It must be set according to the following table[4]:

MSEL[4:0]	*Configure Scheme*	*Description*
10010	AS	FPGA configured from EPCQ (default)
01010	FPPx32	FPGA configured from HPS software: Linux
00000	FPPx16	FPGA configured from HPS software: U-Boot, with image stored on the SD card, like LXDE Desktop or console Linux with framebuffer edition.

When running Linux Console and writing our own code that accessed FPGA peripherals, we selected MSEL[4:0] = 01010. When running the Linux Console with framebuffer and the LXDE desktop environment, we set it to 00000.

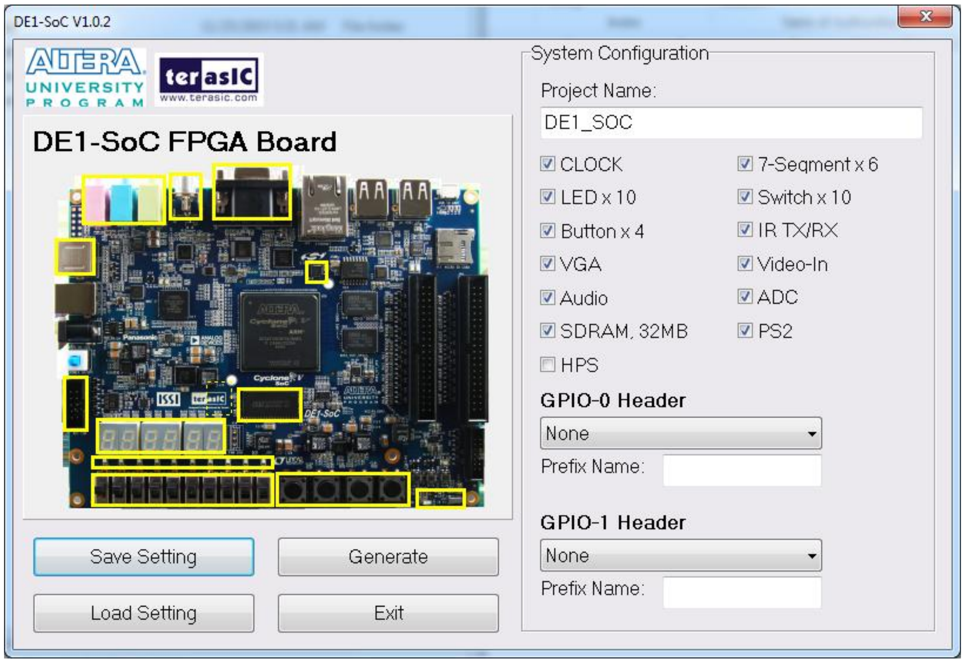
We ran the Linux Console, Linux Console with Framebuffer, and the Linux LXDE Desktop versions. While running LXDE, we noticed occasional issues in operation, most notably that, Firefox kept crashing repeatedly when more than one tab was open. Based on this and that LXDE is lighter than Ubuntu, we decided not to attempt to run Ubuntu. On LXDE, we were able to play a pre-loaded video using the default video player. We connected the board to the internet using an Ethernet connection, and were able to browse the web when Firefox wasn't crashing. It must be noted that on LXDE, it is necessary to login with a password: "terasic" for user: "root".

Despite having running LXDE, we determined that we did not need the GUI. Additionally, it was using the FPGA and we wanted to use the FPGA for our application. Hence, we ran the Linux Console version.

3 Tools for development

- Altera Quartus II v 15.0 or higher
- Altera SoC Embedded Design Suite (EDS)
- DE1-SoC System Builder
- GHRD (Golden Hardware Reference Design), see below
- System CD[5]

To develop using only the FPGA (ie HPS is not required), a Quartus project was created using the DE1-SoC System Builder (whose executable can be found in the CD under **Tools->SystemBuilder** or online[6]):



System Configuration

Project Name: DE1_SOC

[✓] CLOCK [✓] 7-Segment x 6
[✓] LED x 10 [✓] Switch x 10
[✓] Button x 4 [✓] IR TX/RX
[✓] VGA [✓] Video-In
[✓] Audio [✓] ADC
[✓] SDRAM, 32MB [✓] PS2
[ ] HPS

GPIO-0 Header: None
Prefix Name:

GPIO-1 Header: None
Prefix Name:

Figure 3-System Builder

To develop using the FPGA and the HPS, a Quartus project must obviously be created for the FPGA, where the HPS is instantiated. The steps followed to do so are clearly outlined on a UC Davis Lab document[7].

There was one particular thing that we had to do to make it successfully work. After running an "Analysis and Synthesis" (Ctrl+K), a folder named incremental_db is created in the project folder. This caused a full compile (Ctrl+L) to fail. Thus, it was necessary to remove the incremental_db folder between running “Analysis and Synthesis” and performing a full compile.

3.1 GHRD

In the CD accompanying the DE1-SoC (that can be downloaded from the DE1-SoC resources web page), there are several demonstration projects, one of which is the GHRD. This can be located in the directory **Demonstrations->SOC_FPGA->de1_soc_GHRD**. This is a Quartus project file that has the HPS instantiated and the required configurations done for using the HPS with the FPGA for certain tasks. The project also has the ADC, GPIO, seven segment displays, buttons, switches, LEDs, PS2 peripheral and VGA connections instantiated and ready to be used, as would be the case if the project was created with the System Builder tool. More information on GHRD is revealed on the Rocketboards website[8] documenting the GHRD.

The GHRD can be used with the HPS running Linux and can allow the HPS to access the peripherals on the FPGA side. It allows the HPS to access the pushbuttons, DIP switches, LEDs on the DE1-SoC board.

One change was made to the default top level file ghrd_top.v. By default, it presents LEDR1 through LEDR9 (on the board) to be accessed, but not LEDR0 since it was set to blink from the FPGA end. This was changed by removing the always block and assign statement at the end of the program that toggled LEDR0. Additionally, line 215 was changed from

```
assign LEDR[9:1] = fpga_led_internal;
```

to

```
assign LEDR[9:0] = fpga_led_internal;
```

The GHRD contains a file of type sopcinfo: soc_system.sopcinfo, which can be used to generate a header file with the peripheral offsets required to access them from the HPS (see example below). A script required to create the header file was found in a sub-directory in another demo project: Demonstrations -> SOC_FPGA -> HPS_LED_HEX -> LED_HEX_hardware -> generate_hps_qsys_header.sh

The contents of the script are:

```
#!/bin/sh
sopc-create-header-files \
"${SOCEDS_DEST_ROOT}/HPS_LED_HEX/LED_HEX_hardware/soc_system.sopcinfo
" \
--single hps_0.h \
--module hps_0
```

This was changed to:

```
#!/bin/sh
sopc-create-header-files \
"./soc_system.sopcinfo" \
--single hps_0.h \
--module hps_0
```

It was then saved as generate.sh and moved to the required project directory.

This script can be executed using the SoC EDS Command Shell in the SoC EDS install directory. Launching the command shell and using cd to navigate to the project directory, the script can be run:

```
./generate.sh
```

In the generate.sh file, the 3rd line (with the sopcinfo file) tells the script which sopcinfo file to use. Since there was one in the project directory from GHRD, it used that. The 4th line says what the output header file should be. The resultant header file then allows access to FPGA peripherals when writing code for the HPS. A small segment of the header file (for the LEDs) is shown below:

```
      #define LED_PIO_COMPONENT_TYPE altera_avalon_pio
      #define LED_PIO_COMPONENT_NAME led_pio
      #define LED_PIO_BASE 0x10040
      #define LED_PIO_SPAN 64
      #define LED_PIO_END 0x1007f
      #define LED_PIO_BIT_CLEARING_EDGE_REGISTER 0
      #define LED_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
      #define LED_PIO_CAPTURE 0
      #define LED_PIO_DATA_WIDTH 10
      #define LED_PIO_DO_TEST_BENCH_WIRING 0
      #define LED_PIO_DRIVEN_SIM_VALUE 0
      #define LED_PIO_EDGE_TYPE NONE
      #define LED_PIO_FREQ 50000000
      #define LED_PIO_HAS_IN 0
      #define LED_PIO_HAS_OUT 1
      #define LED_PIO_HAS_TRI 0
      #define LED_PIO_IRQ_TYPE NONE
      #define LED_PIO_RESET_VALUE 15
```

3.2 GHRD Program Example

A demo program to increment the value output onto the LEDs, and print the value onto the terminal was developed as an introductory example to allow building on top of the project as more peripherals were used. GHRD was used for the project to simplify design.

The communication between the HPS and the FPGA required for the HPS to access FPGA peripherals is done over the 32-bit wide lightweight HPS to FPGA (lwh2f) bridge.

From a tutorial video online[9], the following setup code was obtained:

```
#define REG_BASE 0xFF200000
// base address of h2f_lw bridge from datasheet

#define REG_SPAN 0x00200000
// 2MB long

void* virtual_base;
void* led;
int fd;
//-------------------------------------------------------------
fd = open("/dev/mem", (O_RDWR|O_SYNC));
virtual_base = mmap(NULL, REG_SPAN, (PROT_READ|PROT_WRITE),
MAP_SHARED, fd, REG_BASE);
led = virtual_base + LED_PIO_BASE;
```

As mentioned previously, it is necessary to map the FPGA peripherals to the running program's virtual memory. This is achieved by using the mmap function as shown above. This assigns the base address for the lwh2f bridge to virtual_base. The offset to the peripherals is included in the header file generated using the generate.sh script as mentioned above. Thus led is assigned the base address to the PIO port driving the LEDs. It is easy to assign values to given addresses using functions from Altera's HW libraries (HWLIB):

```
alt_write_word(led, ledval);
```

The last significant part of the demo application is the use of the POSIX threads library: pthread. While pthread has numerous options (eg setting scheduling parameter, priority, etc), this application only made use of pthread in order to only create three threads to run concurrently: one to write to the LED, one to increment an LED counter, and one to print the value of the counter onto the terminal. While all these are very simple tasks and can be accomplished through a single thread, three threads were created only to demonstrate the use of pthread. A small section of the pthread use is shown below:

```
void* inc_count(void* args){
     while (1){
          if (++ledval >= 1024)
               ledval = 0;
          sleep(1);
     }
     pthread_exit(NULL);
}
//-------------------------------------------------------------
pthread_t count;
if (pthread_create(&count, NULL, inc_count, NULL)){
	fprintf(stdout, "Error creating pthread! Aborting!\n");
	return 2;
}
```

In order to build the demo application, the SoC EDS Command Shell was used again; a Makefile is required to build the program, and a simple Makefile for this was created by altering the one shown in the tutorial video mentioned above and saved in the project directory as “Makefile”:

```
#
TARGET = ghrd

#
CROSS_COMPILE = arm-linux-gnueabihf-
CFLAGS      =      -static      -g      -Wall      -Dsoc_cv_av      -
I${SOCEDS_DEST_ROOT}/ip/altera/hps/altera_hps/hwlib/include       -
I${SOCEDS_DEST_ROOT}/ip/altera/hps/altera_hps/hwlib/include/soc_cv_av
LDFLAGS =  -g -Wall -lpthread
CC = $(CROSS_COMPILE)gcc
ARCH= arm


build: $(TARGET)
$(TARGET): main.o
	$(CC) $(LDFLAGS)   $^ -o $@
%.o : %.c
	$(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
	rm -f $(TARGET) *.a *.o *~
```

A key thing is that in the linker is provided the switch –lpthread since pthread was used in the program. For compilation to be successful, it is also necessary to have soc_cv_av defined (as done in CFLAGS) as well as defining the HWLIBs include paths as include paths for compilation. The target name at the top of the Makefile is the name of the output file. By using the make command in the SoC EDS Command Shell, the output file is generated if there are not build errors. If there are errors or warnings, those are listed.

Once the output file was generated, it is required to transfer it to the DE1-SoC to run in Linux. A simple way to do so is to SSH in with the board connected to an Ethernet cable. However, due to logistic issues, it was not possible to use an Ethernet connection. Thus, the method used was to use a flash drive to transfer the file. The generated file is copied to a USB flash drive, which is then plugged into a USB port on the DE1-SoC. The terminal shows that the flash drive is recognized. For us, the device was /dev/sda. This was mounted, and the code run off of it:

```
	mkdir /media/usb
	mount /dev/sda /media/usb
./ghrd
```

3.3 HWLIBs

We attempted to continue exploring the hardware libraries. These can be located at {SOCEDS_DEST_ROOT}/ip/altera/hps/altera_hps/hwlib, which on Windows 7 for us was D:\Altera\15.0\embedded\ip\altera\hps\altera_hps\hwlib. Based on the include and source files, it seemed like coding for the peripherals would not be too difficult. For example, the following was conceptualized for testing a timer:

```
alt_gpt_all_tmr_init();
// 32-bit timer 0 connected to L4_SP bus clocked by l4_sp_clk
alt_gpt_mode_set(ALT_GPT_SP_TMR0, ALT_GPT_RESTART_MODE_PERIODIC);
alt_gpt_tmr_start(ALT_GPT_SP_TMR0);

fprintf(stdout, "counter = %u\n",
alt_gpt_counter_get(ALT_GPT_SP_TMR0));
```

However, we had troubles creating a Makefile to build the HWLIBs as well the main source file. Thus, progress on the HWLIBs was halted. This is something we plan on addressing immediately as we proceed from here.

4 Power Monitoring

One of the subgoals of the project initially considered was to monitor the power consumption of the SoC under different computation conditions when we were told by our advisor that he had heard that the DE1-SoC had on-board power monitoring for the SoC.

However, upon observing the Power Tree[10] for the DE1-SoC shown below (and visually scanning the board), such a monitoring circuitry could not be identified. While it is possible that this is still available on the board, we could not identify it and could not find any relevant documentation detailing it.

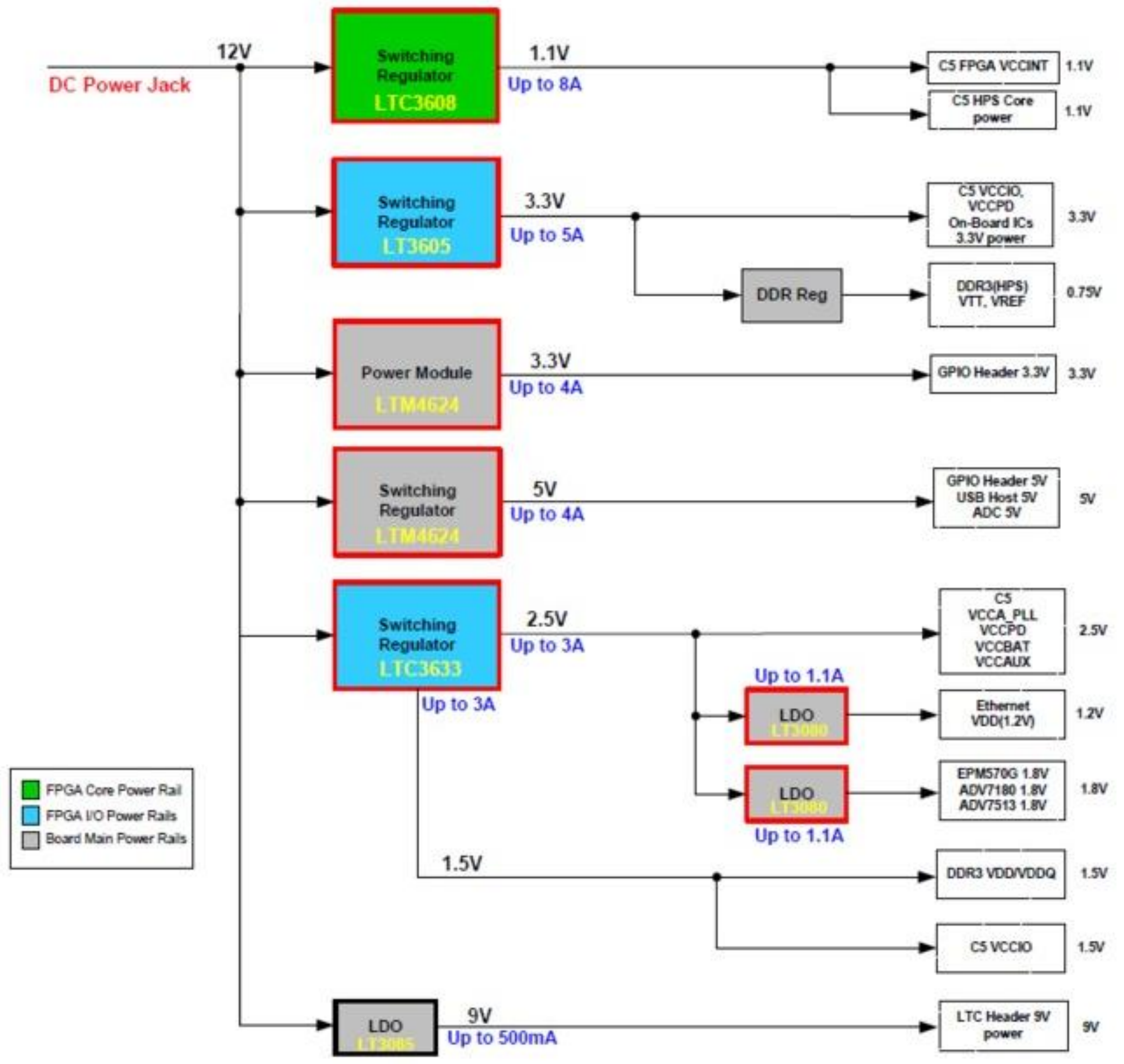


Figure 4 - DE1-Soc Power Tree

5 Parareal Algorithm

One of the goals of this project was to create a way to solve differential equations in parallel by utilizing the capabilities of an FPGA. However, traditional, well-known methods for solving differential equations are inherently serial – just consider the form of Euler's Method, which is one of the simplest ways to solve a first order ODE. Euler's formula solves a differential equation of the following form using the following iterative scheme:[11]

$$\frac{dy}{dt} = f(t, y)$$

$$y_{n+1} = y_n + h * f_n$$

Obviously, Euler's method would be slower to implement on an FPGA due to its serial nature – each successive value in the approximation directly depends on the previous value. Only one computation would be performed per clock cycle, which inherently defeats the purpose of using an FPGA for such a task. Similar techniques, such as the fourth order Runge-Kutta, would also be ineffective on an FPGA because they are also serial in nature.[12]

Despite this, algorithms do indeed exist for solving differential equations in a truly parallel manner. The one that we chose to implement is known as Parareal. Parareal is an interesting algorithm because it allows computations to be performed in parallel by splitting up the time axis into fragments which, for the most part, can be treated as individual entities. This method is known as "parallel-in-time integration" and will converge to the solution that is provided by more traditional serial methods for solving differential equations.[13] This will be shown experimentally in the next section.

The math underlying the Parareal algorithm is quite intense and we are not going to pretend as though we understand everything that is involved with the theory. Instead, we will address this subject purely from the standpoint of implementation details. Simplifying the algorithm to its implementation details actually makes it quite intuitive to understand, even if the proofs are substantially more difficult to comprehend. This document by Scott Field (see reference) from Brown University is perfect for understanding the implementation of Parareal, and is what the following discussion on Parareal is based upon, in addition to the helpful clarifications made by the Wikipedia page.[14]

The first step in the Parareal algorithm is to choose two different "operators," which are simply serial algorithms for solving the first order ODE's that we are interested in. In this regard, there are many viable options, such as Euler's method or the Runge-Kutta. Any simple numerical ODE solver will do. The first of the two operators is known as the coarse operator, because it provides only a coarse solution to the differential equation. Remember that for numerical ODE solvers, the precision of the solution is determined by two major factors – the algorithm itself and the chosen step size. With this in mind, the coarse operator can be achieved by using a relatively low precision technique to approximate the differential equation in question, namely by either changing the step size or the order of the operator. The second operator is known as the fine operator, which has a much higher precision, again by either decreasing the step size or increasing the operator order.[15]

Notice that both the coarse and fine operator are serial, and therefore would not see much benefit to being implemented on an FPGA, as previously discussed. Individually, it is true that neither of these solutions is viable for parallelization. However, the brilliance of Parareal comes from its ability to combine these two inherently serial operators into an overall scheme that is, for the most part, parallelized. The overarching idea is this – one can first use the coarse operator to create a rough approximation to the differential equation, and then use the fine operator over individual slices in between the coarse operator solutions to refine the approximation. For example, if the step size of the coarse operator was 0.1, and we are approximating a solution over the time scale from 0 to 10, then solutions to the differential equation can be obtained for the values at the times 0, 0.1, 0.2, etc. At this point, the fine operator is used. Assuming that the fine operator has a step size of 0.01, solutions can now be found "in between" time slices by using the coarse operator solutions as initial conditions for the fine operator. For example, to solve the differential equation for the values at time 0.11, 0.12, 0.13, … , 0.2, we would simply use the fine operator with the initial condition f(0.1), which was already determined by the coarse operator! The final approximation on this time slice will yield f(0.2), which should be more precise than before, since it was computed on a finer time scale. Note that this isn't entirely correct since the initial condition itself was an approximation, so some mathematical alterations are necessary for the algorithm to converge properly, and the explanation above is simply an intuitive attempt at a justification for why a fine operator is necessary. (More details on the exact equation for computing the Parareal solution will follow shortly.) Notice that when computed in this manner, each time slice for the fine operator can be computed independently, since the initial conditions are just the coarse operator solution values. Thus, while the fine operator itself is inherently serial, multiple fine operators can be running simultaneously to speed up the computation. Notice that Parareal will converge to a solution with a precision equivalent to the fine operator step size, since the fine operator is computed over the region determined by the coarse operator. Using this technique allows one to compute solutions to a higher degree of accuracy in a shorter amount of time by exploiting parallelism.[16]

The specific equation for computing Parareal approximations is as follows. Although the Field document states the equation, the Wikipedia article shows it in a slightly clearer (but equivalent) manner, so the Wikipedia form is what will be displayed here:[17]

$$y_{j+1}^{k+1} = G\left(y_j^{k+1}, t_j, t_{j+1}\right) + F\left(y_j^k, t_j, t_{j+1}\right) - G(y_j^k, t_j, t_{j+1})$$

Where G represents the coarse operator, and F represents the fine operator. The return value of these operators is just the approximation yielded by the operator at time t_{j+1}. The first parameter is the initial condition, the second parameter is the beginning of the time slice, and the third parameter is the end of the time slice. Note that the values t_j simply represent the values of time for which the coarse operator solves the differential equation, and the values y_j are these solutions. The index k is used to represent repeated iterations of the algorithm; more details on this will be provided shortly.

In simpler terms, all this equation says is this – first, compute an initial coarse order approximation in serial using the given initial condition. Next, compute all of the correction terms in parallel using the values obtained by the coarse operator – this is the second two terms. Note that the first term in the correction term is a fine operator over the time slice, and the second term in the correction term is a coarse operator (single step) over the time slice. Finally, compute, in serial, the coarse order approximation once more (term 1 in the equation).[18] Repeating this process for K iterations, where K is chosen by the user, will eventually lead to convergence of the Parareal algorithm. The index k is incremented until it reaches K. As Wikipedia notes, the coarse computation (term 1 in the equation) must be serial because the first coarse operator relies directly on the previous time value in the same iteration stage.[19]

Each successive layer of approximations will yield better and better approximations. Clearly, the value of K needed to reach convergence is going to be dependent on the operators chosen, and the relative ratios of the step sizes of the fine and coarse operators.

6 Implementing and Testing Parareal in Java

For our own sanity, Parareal was not immediately implemented in Verilog for the FPGA. Testing such a complex piece of code and proving that it works is near impossible, and realizing this we decided that it would be best to first create an implementation in Java. This made the initial debugging process easier, at least from the perspective of available debugging tools for each language (Modelsim versus the Eclipse IDE debugger).

For both the coarse and fine operator, a fourth order Runge-Kutta method was chosen. The step sizes can be altered in the program as desired. For the purposes all tests performed here, assume that the coarse operator step size is 0.001, and the fine operator step size is 0.00001.

Code for the Java implementation of Parareal can be found in the Appendix section of this document. The following table documents some test results found when using the Parareal method. It compares the Parareal solution to the standard fourth order Runge-Kutta solution, and compares the execution times of each. For each of the test, assume the initial condition is y(0) = 1, and the value being approximated is y(5). Note that both methods will actually return a large range of approximations between the times 0 and 5, but for simplicity only the final value, y(5), is shown in the table. The "Actual" column is the value of y(5) computed in closed form using Wolframalpha[20] (e.g., for $y' = y$, $y(5) = e^5$). The number of Parareal iterations is set to 7.

Differential Eqn.	Standard RK4	Parareal	Actual	RK4 Time (ns)	Parareal Time (ns)
$y' = y$	148.413159	148.413159	148.413159	5948408	3550096520
$y' = \frac{y * t}{100}$	1.13314845	1.13314845	1.13314845	20712479	3567484803
$y' = y * t^2$	1.2462451E18	1.2462451E18	1.2462449E18	11407033	3546461952

(Note that in the third test, the values between Standard RK4 and Parareal differed at the fourteenth decimal place, but that was far too many decimals to write out. They were equivalent to a reasonable degree of accuracy.)

From the above table, it can be seen that the approximations are quite good, and are consistent with the expected result to a large number of decimal places. The Parareal solution generated by our code also matches what one would expect from the Runge-Kutta solution using a step size equivalent to the step size of the fine operator, which is what Parareal should converge to. Based on this, it would seem that our implementation of the Parareal algorithm is correct.

However, there is one rather large anomaly in the above table – the Parareal solution takes longer to execute by many orders of magnitude! This is an interesting result, since we expect Parareal to be significantly faster. As it turns out, using threads in Java doesn't always mean that the threads will execute on the different cores of the computer. Furthermore, there is overhead in initializing the threads in the first place. Essentially, this code adds additional overhead while still executing on the CPU in a serial fashion, defeating the purpose of the Parareal algorithm. In order to rectify this, a truly parallelized solution must be created using Verilog and an FPGA. The next section is devoted to the Verilog implementation of the Parareal algorithm.

7 Implementation and Testing of Parareal in Verilog

As noted in the previous section, implementing Parareal in Java on a laptop is somewhat tricky because laptops are not always able to fully exploit parallelism. Thus, a Verilog implementation was created. It uses 64-bit fixed point mathematics (a choice that eventually became a problem) rather than floating point due to the ease of implementation. The project file was generated by the Terasic System Builder.[21] What follows is a description of the states of the finite state machine used to implement Parareal in Verilog.

State_RESET – In this state, all variables are reset. TN is set to T0, and YN is set to Y0, the initial condition. HCOARSE, the step size of the coarse operator, is set to the difference between the start and end time (T1-T0) divided by N, the number of steps for the coarse operator. HFINE is set to HCOARSE divided by NFINE, the number of fine operator steps.

STATE_INIT_COND – In this state, the first values in the arrays lastApprox and approx. are set to the initial condition, Y0. START, the signal controlling the start of the parallel execution stage, is set to 0.

STATE_INIT_COARSE – In this state, the initial coarse approximation is computed and stored in the lastApprox array.

STATE_PARALLEL – In this state, all of the correction terms are computed in parallel by separate modules that were created using generate blocks.

STATE_CORRECT – In this state, the correction terms are added to the approx array.

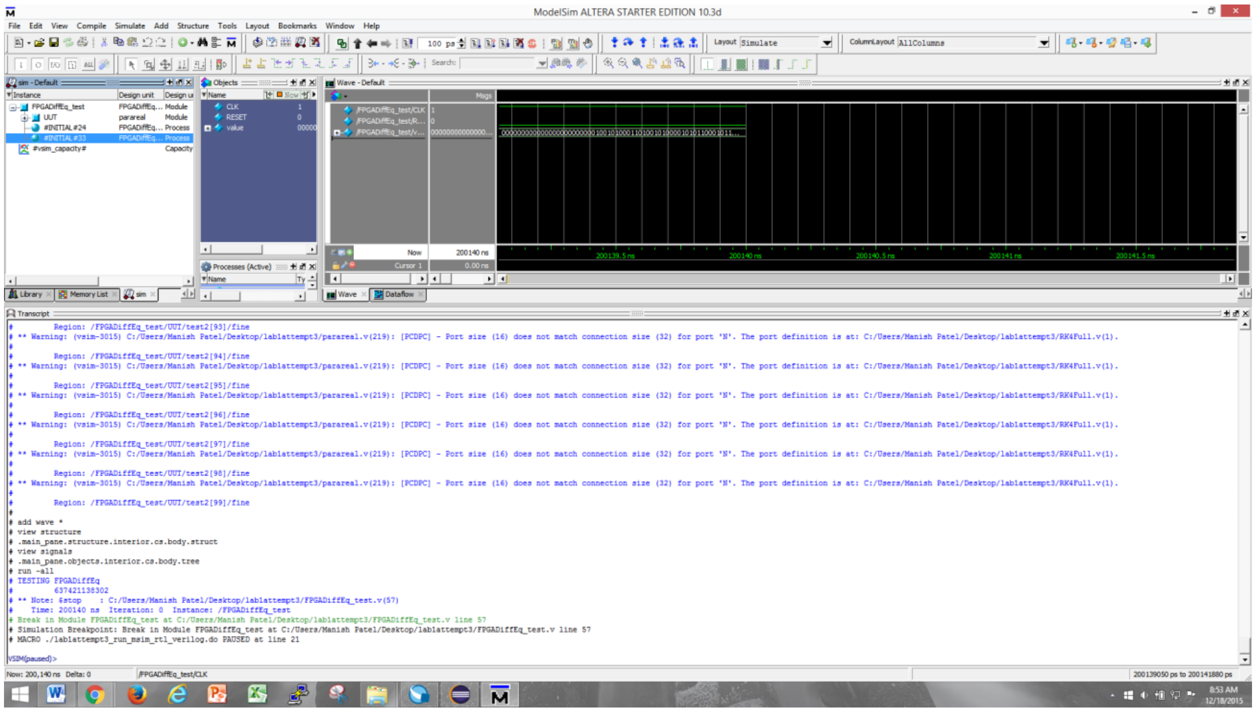
STATE_SERIAL – In this state, the serial coarse operator is run again, in order to update the approx array.

STATE_SETTLE – In this one cycle state, all of the values in approx. are copied into lastApprox for the next iteration of the algorithm. The value of k is incremented since another iteration has been completed.

STATE_END – This state, once entered, will stay here forever. It indicates that all iterations of the Parareal algorithm have finished.

The following two pictures from ModelSim show two different tests that were run using the Parareal Verilog code. Both tests output the value (in 64-bit fixed point, 32 bits after the decimal point) of the solution to the differential equation $y' = y$ with initial condition $y(0) = 1$. It returns the value of $y(5)$.

In the first test, the number of steps for the coarse operator is set to 100, and the number of steps for the fine operator is also set to 100. Note that this is slightly different than the notation used above for the Java code, where a step size was specified. In the Verilog implementation it is easier to specify a fixed number of steps instead. Step size and number of steps are inversely proportional, so a higher number of steps indicates a higher degree of precision. The number of Parareal iterations was set to 5. The results of this test are shown in the below image:



```
#         Region: /FPGADiffEq_test/UUT/test2[93]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[94]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[95]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[96]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[97]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[98]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[99]/fine
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# TESTING FPGADiffEq
#         637421138302
# ** Note: $stop    : C:/Users/Manish Patel/Desktop/lablattempt3/FPGADiffEq_test.v(57)
#    Time: 200140 ns  Iteration: 0  Instance: /FPGADiffEq_test
# Break in Module FPGADiffEq_test at C:/Users/Manish Patel/Desktop/lablattempt3/FPGADiffEq_test.v line 57
# Simulation Breakpoint: Break in Module FPGADiffEq_test at C:/Users/Manish Patel/Desktop/lablattempt3/FPGADiffEq_test.v line 57
# MACRO ./lablattempt3_run_msim_rtl_verilog.do PAUSED at line 21

VSIM(paused)>
```

Figure 5-Simulation Result 1

The fixed point value returned was 637421138302. In standard notation, this would correspond to a value of:

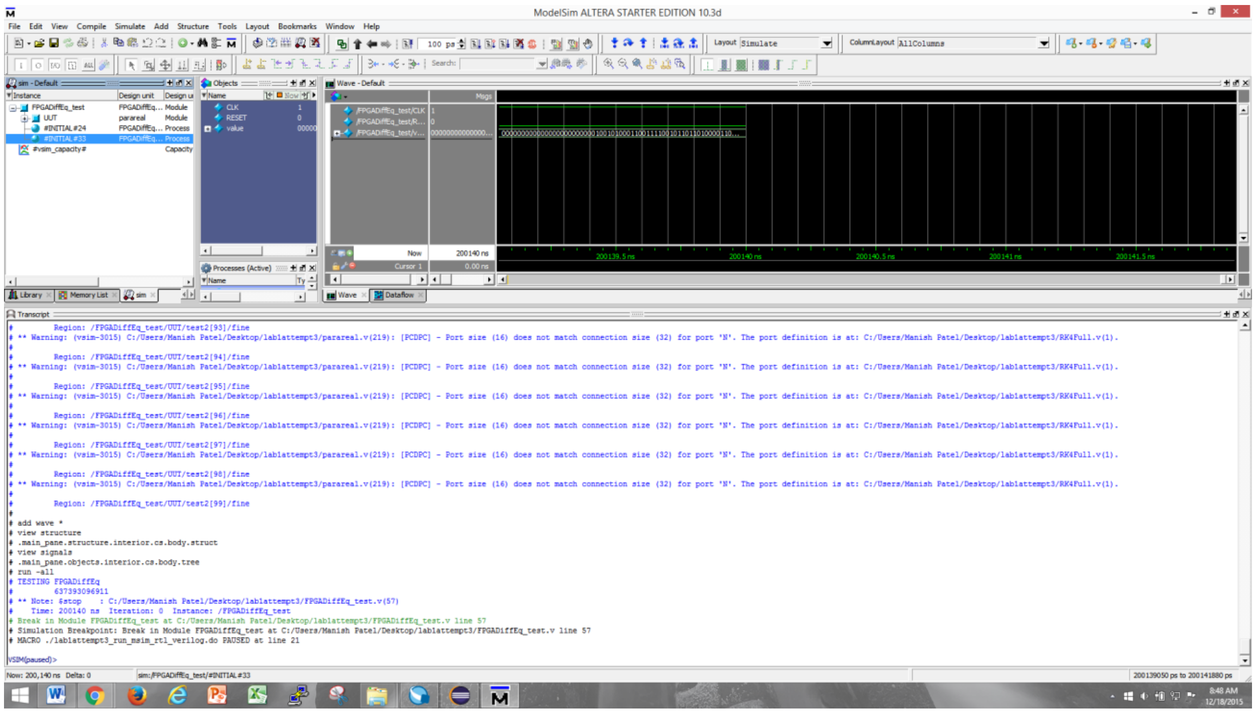
$$y(5) \approx \frac{637421138302}{2^{32}} = 148.4111739$$

The actual value that is expected is:

$$y(5) = e^5 = 148.4131591$$

Given that a relatively small number of steps was chosen for both the coarse and fine operator, this approximation is fairly decent, though still not as good as we would expect.

Test 2 may help shed some light on why the approximation found in Test 1 was less accurate than expected. In Test 2, everything is the same except the number of steps for the fine operator is not 1000 instead of 100. Given this, it is expected that accuracy should increase. The results are shown in the image below:



```
#         Region: /FPGADiffEq_test/UUT/test2[93]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[94]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[95]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[96]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[97]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[98]/fine
# ** Warning: (vsim-3015) C:/Users/Manish Patel/Desktop/lablattempt3/parareal.v(219): [PCDPC] - Port size (16) does not match connection size (32) for port 'N'. The port definition is at: C:/Users/Manish Patel/Desktop/lablattempt3/RK4Full.v(1).
#
#         Region: /FPGADiffEq_test/UUT/test2[99]/fine
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# TESTING FPGADiffEq
#         637393096911
# ** Note: $stop    : C:/Users/Manish Patel/Desktop/lablattempt3/FPGADiffEq_test.v(57)
#    Time: 200140 ns  Iteration: 0  Instance: /FPGADiffEq_test
# Break in Module FPGADiffEq_test at C:/Users/Manish Patel/Desktop/lablattempt3/FPGADiffEq_test.v line 57
# Simulation Breakpoint: Break in Module FPGADiffEq_test at C:/Users/Manish Patel/Desktop/lablattempt3/FPGADiffEq_test.v line 57
# MACRO ./lablattempt3_run_msim_rtl_verilog.do PAUSED at line 21

VSIM(paused)>
```

Figure 6 - Simulation Result 2

The fixed point value returned was 637393096911. In standard notation, this would correspond to a value of:

$$y(5) \approx \frac{637393096911}{2^{32}} = 148.404645$$

Again, the actual value that is expected is:

$$y(5) = e^5 = 148.4131591$$

Here something peculiar has occurred. The approximation yielded by increasing the number of steps has actually decreased in accuracy, when it should have increased. Given that the first approximation was also off by more than expected, it seems that a possible reason for this is that fixed point mathematics is not nearly as precise as double mathematics, which is what was used in the Java version. As the number of computations increases, as does when the number of steps is increased, the more opportunity there is for errors to accumulate. Thus, given that we are using fixed point mathematics, it seems reasonable that error could increase when the number of steps is increased, even though we would prefer that it did not. The other possible explanation is that there is something wrong in the implementation itself, but given that the approximations are very close to expected, this seems unlikely. Numerical truncation errors seem more likely. Another error that should be mentioned is that occasionally there are over and underflow errors when performing the floating point math, and these need to be handled in a more robust manner. More work needs to be done in the future to make sure that the implementation is as robust as possible, but for now it seems to be working moderately well. Future work (over winter break of 2015-2016) will include thoroughly looking over the Verilog implementation for any bugs and attempt to fix them.

8 Major Obstacles

Getting all the required resources took us a while since we could not find an organized set of documentation describing what was necessary. We eventually found that there was a lot of documentation and references available from different sources. However, they were a bit all over the place and finding them was not straightforward.

Initially, a lot of time was spent attempting to work with the DS-5 IDE for cross-compiling source code. The Altera Edition DS-5 is installed as part of the Altera SoC EDS and can be installed with a free community license[22]. After having done this, we attempted to create a test project in DS-5 to enable us to develop within it. We had trouble getting it to correctly identify included header files for programming, even standard C header files for development. Following the steps listed out on the SoC EDS Wiki (which we had found later) did not result in successful build of the project. Including a default demo application did not work either. Upon failing to solve the issue, we decided to look for an alternate way of developing.

We used the SoC EDS Command Shell, along with a Makefile to build our code. This required us to learn how to create and work with Makefiles. The one major issue we ran into with this is that we had a folder named DE1-SOC in the project path and make would keep failing with a permission denied error because of the hyphen in the path name. Removing the hyphen so that the folder now read DE1SOC fixed the issue.

The Makefile required to interface with the HWLIBs that had us stuck. We haven't yet been able to complete that successfully and have spent a lot of time dealing with that.

A major hurdle was compiling the Verilog code for the Parareal implementation. The Quartus project would get stuck at 46% when running "Analysis and Synthesis" on Tahmid's computer, two attempted desktops in Philips 238 Lab, as well as one in Carpenter Hall. However, when testing was moved to Manish's computer, the compilation completed successfully with no issues. We were not able to identify the reason for this issue, except noticing that all attempted computers that failed were running Windows 7, while Manish's was running Windows 10 – although that is not a conclusive cause, just an observation. The build time was still long. This was later fixed by creating a new project and removing the HPS component from it since it was not being used.

9 Acknowledgement

We would like to thank Professor Bruce Land for giving us the opportunity to be involved with this project, as well as for his guidance and inspiration.

10 Conclusion

We have made progress in both sections of our independent study – the parallel implementation of the ODE solver and the HPS programming section. Parareal has been tested in Java and in Verilog, although the Verilog version has some computational error. We have familiarized ourselves with the programming environment, although we would like to be able to use DS-5. We have looked at other development tools such as the Mentor Graphics Sourcery Codebench. However, we have stuck to SoC EDS. In both sections we are at an obstacle, which we believe we will be able to solve soon and accelerate progress. We hope to continue this next semester.

11 Code Listing

11.1 Demo Application Code for GHRD using FPGA and HPS

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

#include <hwlib.h>
#include "socal/socal.h"

#include "hps_0.h"
#include "pthread.h"

#define REG_BASE 0xFF200000 // base address of h2f_lw bridge from datasheet
#define REG_SPAN 0x00200000 // 2MB long

void* virtual_base;
void* led;
int fd;

unsigned int ledval = 0x55;

void* led_out(void* args){
	while (1){
		alt_write_word(led, ledval);
	}
	pthread_exit(NULL);
}

void* inc_count(void* args){
	while (1){
		if (++ledval >= 1024)
			ledval = 0;
		sleep(1);
	}
	pthread_exit(NULL);
}

void* print_out(void* args){
	while (1){
		fprintf(stdout, "ledval = %d\n", ledval);
```

```
            //fprintf(stdout,      "counter      value      =      %u\n",
alt_gpt_counter_get(ALT_GPT_SP_TMR0));
            sleep(1);
      }
      pthread_exit(NULL);
}

int main(void){
            fd = open("/dev/mem", (O_RDWR|O_SYNC));
            virtual_base  =  mmap(NULL,  REG_SPAN,  (PROT_READ|PROT_WRITE),
MAP_SHARED, fd, REG_BASE);
            led = virtual_base + LED_PIO_BASE;

            fprintf(stdout, "Starting!\n");

            pthread_t blink;
            if (pthread_create(&blink, NULL, led_out, NULL)){
                  fprintf(stdout, "Error creating pthread! Aborting!\n");
                  return 1;
            }

            pthread_t count;
            if (pthread_create(&count, NULL, inc_count, NULL)){
                  fprintf(stdout, "Error creating pthread! Aborting!\n");
                  return 2;
            }

            pthread_t print;
            if (pthread_create(&print, NULL, print_out, NULL)){
                  fprintf(stdout, "Error creating pthread! Aborting!\n");
                  return 3;
            }

            fprintf(stdout, "All good!\n");
            while (1);
            return 0;
}
```

11.2 Parareal Code

The following is the Java code for the Parareal implementation and testing. It includes three major functions, serialParareal, parallelParareal, and RK4. The first two functions are equivalent, with the only difference being that parallelParareal uses threads for parallel computation (which, as previously stated, ended up being serialized by the laptop.) RK4 simply computes a standard fourth order Runge-Kutta. Another important function is eval, which is where the user can modify the differential equation they are trying to solve.

```
/*
 * Helpful Literature for Implementation
 *          ***          This          one          especially          ***
http://www.cfm.brown.edu/people/jansh/page5/page10/page40/assets/Field_Talk.
pdf
 * *** Wikipedia *** https://en.wikipedia.org/wiki/Parareal
 */
public class Parareal {

      public static void main(String[]args) {
            //http://stackoverflow.com/questions/180158/how-do-i-time-a-
methods-execution-in-java


      //System.out.println(Runtime.getRuntime().availableProcessors());
            /*
            long startTime = System.nanoTime();
            double[]res = parallelParareal(.01, 0, 1, 3, 15);
            long endTime = System.nanoTime();
            long parallelTime = endTime - startTime;
            System.out.println("Parallel  Parareal  Execution  Time  =  "  +
(parallelTime));

            startTime = System.nanoTime();
            double[]res2 = serialParareal(.01, 0, 1, 3, 15);
            endTime = System.nanoTime();
            long serialTime = endTime - startTime;
            System.out.println("Serial  Parareal  Execution  Time  =  "  +
(serialTime));

            */

            long startTime = System.nanoTime();
            double[]res = parallelParareal(.001, 0, 1, 5, 7);
            System.out.println(res[res.length-1]);
            long endTime = System.nanoTime();
            long pararealTime = endTime - startTime;
            System.out.println("Parareal    Execution    Time    =    "    +
(pararealTime));

            startTime = System.nanoTime();
            double res3 = RK4(.00001, 0, 1, 5);
            endTime = System.nanoTime();
            long fullySerialTime = endTime - startTime;
            System.out.println("Fully   Serial   Execution   Time   =   "   +
(fullySerialTime));
            System.out.println(res3);
            /*
            for(double x : res)
                  System.out.println(x);
                  */

            /*
            System.out.printf("%.20f", res[res.length-1]);
            System.out.println("");
            System.out.printf("%.20f", res2[res2.length-1]);
            System.out.println("");
            System.out.printf("%.20f", res3);
            System.out.println("");

            System.out.println("Parallel     Execution     was     "     +
serialTime/parallelTime + " times faster.");
            */
      }

      /*
       * Computes the Parareal approximation to the differential equation
without using threads (so
       * this is effectiely a serialized version of Parareal that was
implemented initially to make
       * sure that the algorithm worked.)
       */
      public static double[] serialParareal(double hCoarse, double t0, double
y0, double t1, int K) {
            int N = (int)(Math.round((t1-t0)/hCoarse))+1;
            double hFine = hCoarse/100.0;

            double[][] approx = new double[K+1][N];

            // init y0's in positions approx[k][0]
            for(int k=0; k<K+1; k++) {
                  approx[k][0] = y0;
            }

            // run coarse operator
            for(int n=0; n<N-1; n++) {
                  approx[0][n+1] = RK4(hCoarse, t0+hCoarse*n, approx[0][n],
t0+hCoarse*(n+1));
            }

            for(int k=0; k<K; k++) {
                  double[] correction = new double[N];
                  for(int n=0; n<N-1; n++) {
                        //approx[k+1][n+1] = approx[0][n+1];
                        correction[n+1]   =      RK4(hFine,    t0+hCoarse*n,
approx[k][n], t0+hCoarse*(n+1));
                        correction[n+1]   -=   RK4(hCoarse,    t0+hCoarse*n,
approx[k][n], t0+hCoarse*(n+1));
                        approx[k+1][n+1] += correction[n+1];
                  }

                  for(int n=0; n<N-1; n++) {
                        approx[k+1][n+1]   +=   RK4(hCoarse,   t0+hCoarse*n,
approx[k+1][n], t0+hCoarse*(n+1));
                  }
            }

            /*
            for(int x=0; x<=K; x++) {
                  System.out.println(approx[x][N-1]);
            }
            */
            return approx[K];
      }

      /*
       * Parallelized version of Parareal that uses threads to compute the
correction terms in parallel.
       */
      public static double[] parallelParareal(double hCoarse, double t0,
double y0, double t1, int K) {
            int N = (int)(Math.round((t1-t0)/hCoarse))+1;
            double hFine = hCoarse/100.0;

            double[][] approx = new double[K+1][N];

            // init y0's in positions approx[0][n]
            for(int k=0; k<K+1; k++) {
                  approx[k][0] = y0;
            }

            // run coarse operator
            for(int n=0; n<N-1; n++) {
                  approx[0][n+1] = RK4(hCoarse, t0+hCoarse*n, approx[0][n],
t0+hCoarse*(n+1));
            }

            for(int k=0; k<K; k++) {
                  double[] correction = new double[N];
                  //CorrectionThread[]threads = new CorrectionThread[N-1];
                  for(int n=0; n<N-1; n++) {
                        Runnable r = new CorrectionThread(correction, hFine,
hCoarse, t0, approx[k][n], n);
                        Thread thread = new Thread(r);
                        thread.start();

                        //threads[n] = (CorrectionThread)r;


      //System.out.println(Runtime.getRuntime().availableProcessors());

                        //http://stackoverflow.com/questions/7939257/wait-
until-all-threads-finish-their-work-in-java
                        try {
                              thread.join();
                        }
                        catch(Exception e) {}
                  }

                  for(int n=0; n<N-1; n++) {
                        approx[k+1][n+1]    =    RK4(hCoarse,    t0+hCoarse*n,
approx[k+1][n], t0+hCoarse*(n+1)) + correction[n+1];
                  }
            }
            return approx[K];
      }

      public static double RK4(double h, double t0, double y0, double t1) {
            double yn = y0;
            int N = (int)(Math.round((t1-t0)/h));
            for(double index=0; index<N; index++) {
                  double tn = t0 + index*h;
                  double k1 = eval(tn, yn);
                  double k2 = eval(tn+h/2, yn+h/2*k1);
                  double k3 = eval(tn+h/2, yn+h/2*k2);
                  double k4 = eval(tn+h, yn+h*k3);

                  yn = yn + h/6.0*(k1 + 2*k3 + 2*k3 + k4);
            }

            return yn;
      }

      public static double eval(double tn, double yn) {
            return yn*tn*tn;  // set this to whatever differential equation
you want to solve
      }
}

public class CorrectionThread implements Runnable {
      double[]correction;
      double hFine;
      double hCoarse;
      double t0;
      double y0;
      int n;

      //http://stackoverflow.com/questions/877096/how-can-i-pass-a-
parameter-to-a-java-thread
      public  CorrectionThread(double[]correction,  double  hFine,  double
hCoarse, double t0, double y0, int n) {
            this.correction = correction;
            this.hFine = hFine;
            this.hCoarse = hCoarse;
            this.t0 = t0;
            this.y0 = y0;
            this.n = n;
      }

      public void run() {
            correction[n+1] = RK4(hFine, t0+hCoarse*n, y0, t0+hCoarse*(n+1));
            correction[n+1]    -=    RK4(hCoarse,    t0+hCoarse*n,    y0,
t0+hCoarse*(n+1));
      }

      public static double RK4(double h, double t0, double y0, double t1) {
            double yn = y0;
            int N = (int)(Math.round((t1-t0)/h));
            for(double index=0; index<N; index++) {
                  double tn = t0 + index*h;
                  double k1 = eval(tn, yn);
                  double k2 = eval(tn+h/2, yn+h/2*k1);
                  double k3 = eval(tn+h/2, yn+h/2*k2);
                  double k4 = eval(tn+h, yn+h*k3);
```

```
                yn = yn + h/6.0*(k1 + 2*k3 + 2*k3 + k4);
            }

            return yn;
        }

        public static double eval(double tn, double yn) {
            return yn*tn*tn;  // insert differential equation to be solved
here
        }
}
```

The following is the Verilog Code for the Parareal implementation.

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module lab1attempt3(

        //////////// ADC //////////
        inout                   ADC_CS_N,
        output                  ADC_DIN,
        input                   ADC_DOUT,
        output                  ADC_SCLK,

        //////////// Audio //////////
        input                   AUD_ADCDAT,
        inout                   AUD_ADCLRCK,
        inout                   AUD_BCLK,
        output                  AUD_DACDAT,
        inout                   AUD_DACLRCK,
        output                  AUD_XCK,

        //////////// CLOCK //////////
        input                   CLOCK_50,
        input                   CLOCK2_50,
        input                   CLOCK3_50,
        input                   CLOCK4_50,

        //////////// SDRAM //////////
        output          [12:0]          DRAM_ADDR,
        output           [1:0]          DRAM_BA,
        output                  DRAM_CAS_N,
        output                  DRAM_CKE,
	output			DRAM_CLK,
	output			DRAM_CS_N,
	inout		[15:0]		DRAM_DQ,
	output			DRAM_LDQM,
	output			DRAM_RAS_N,
	output			DRAM_UDQM,
	output			DRAM_WE_N,

	/////////// I2C for Audio and Video-In //////////
	output			FPGA_I2C_SCLK,
	inout			FPGA_I2C_SDAT,

	/////////// SEG7 //////////
	output		[6:0]		HEX0,
	output		[6:0]		HEX1,
	output		[6:0]		HEX2,
	output		[6:0]		HEX3,
	output		[6:0]		HEX4,
	output		[6:0]		HEX5,

	/////////// IR //////////
	input			IRDA_RXD,
	output			IRDA_TXD,

	/////////// KEY //////////
	input		[3:0]		KEY,

	/////////// LED //////////
	output		[9:0]		LEDR,

	/////////// PS2 //////////
	inout			PS2_CLK,
	inout			PS2_CLK2,
	inout			PS2_DAT,
	inout			PS2_DAT2,

	/////////// SW //////////
	input		[9:0]		SW,

	/////////// Video-In //////////
	input			TD_CLK27,
	input		[7:0]		TD_DATA,
	input			TD_HS,
	output			TD_RESET_N,
	input			TD_VS,
	/////////// VGA /////////
	output		[7:0]		VGA_B,
	output				VGA_BLANK_N,
	output				VGA_CLK,
	output		[7:0]		VGA_G,
	output				VGA_HS,
	output		[7:0]		VGA_R,
	output				VGA_SYNC_N,
	output				VGA_VS
);



//=======================================================
//  REG/WIRE declarations
//=======================================================




//=======================================================
//  Structural coding
//=======================================================

parareal para (
			.CLK(CLOCK_50),
			.RESET(0),
			.Y0(0),
			.T0(0),
			.T1(5),
			.value(approx)
	);

endmodule
```

```
module parareal(CLK, RESET, Y0, T0, T1, value);

	//NEED TO DEAL WITH SIGNED NUMBERS - DEFAULT IS UNSIGNED

	`define STATE_RESET 0
	`define STATE_INIT_COND 1
	`define STATE_INIT_COARSE 2
	`define STATE_PARALLEL 3
	`define STATE_SERIAL 4
	`define STATE_CORRECT 5
	`define STATE_SETTLE 6
```

```
`define STATE_END 7

`define K 10
`define N 100
`define NFINE 1000

input           CLK;
input                   RESET;
input [63:0] Y0;
input [63:0] T0;
input [63:0] T1;
reg   [3:0]   STATE = 0;
reg   [3:0]   NEXT_STATE = 0;



// approx arrays
reg   [63:0] lastApprox [0:(`N)];
reg   [63:0] approx             [0:(`N)];
wire   [63:0] correction [0:(`N)];
wire   [63:0] correction2[0:(`N)];

output [63:0] value;
assign value = approx[`N];

// RK4 variables
reg       [63:0] TN;
reg   [63:0] YN;
reg       [63:0] HCOARSE = 1;
reg       [63:0] HFINE = 1;

reg                         START = 0;
wire   [(`N):0] DONE;
wire  [63:0] RK4COARSE;

// loop variables
integer x;
integer y;
integer k;
integer n;

always @(*) begin
        if(RESET) begin
                STATE <= `STATE_RESET;
        end
        else begin
```

```
                STATE <= NEXT_STATE;
        end
end

always @(posedge CLK) begin
        if(STATE == `STATE_RESET) begin
                k <= 0;
                n <= 0;
                TN <= T0;
                YN <= Y0;
                HCOARSE <= ((T1-T0)/`N);
                HFINE <= HCOARSE/`NFINE;
                START <= 0;


                NEXT_STATE <= `STATE_INIT_COND;
        end
        else if(STATE == `STATE_INIT_COND) begin
                lastApprox[0] <= Y0; // index = time
                approx[0] <= Y0;

                k <= 0;
                n <= 0;
                TN <= T0;
                YN <= Y0;
                HCOARSE <= ((T1-T0)/`N);
                HFINE <= HCOARSE/`NFINE;
                START <= 0;

                NEXT_STATE <= `STATE_INIT_COARSE;
        end
        else if(STATE == `STATE_INIT_COARSE) begin
                lastApprox[n+1] <= RK4COARSE;
                k <= 0;

                TN <= TN + HCOARSE;
                YN <= RK4COARSE;
                HCOARSE <= ((T1-T0)/`N);
                HFINE <= HCOARSE/`NFINE;
                START <= 0; // flag to start parallel computation of all modules


                if(n < `N-1) begin // check this is the right ending condition
                        NEXT_STATE <= `STATE_INIT_COARSE;
                        n <= n+1;
                end
```

```
				else begin
					NEXT_STATE <= `STATE_PARALLEL;
					n <= 0;
				end
			end
			else if(STATE == `STATE_PARALLEL) begin
				k <= k;

				TN <= T0;
				YN <= Y0;
				HCOARSE <= ((T1-T0)/`N);
				HFINE <= HCOARSE/`NFINE;
				START <= 1;

				// until parallel computations are done spin in this state
				if(DONE[0] == 1) begin
					NEXT_STATE <= `STATE_CORRECT;
					n <= 0;
				end
				else begin
					NEXT_STATE <= `STATE_PARALLEL;
					n <= n+1;
				end
			end
			else if(STATE == `STATE_CORRECT) begin
				k <= k;
				n <= 0;
				TN <= T0;
				YN <= approx[0];
				HCOARSE <= ((T1-T0)/`N);
				HFINE <= HCOARSE/`NFINE;
				START <= 1;

				for(y=0; y<`N; y=y+1) begin
					approx[y+1] <= correction[y+1] - correction2[y+1];
				end

				NEXT_STATE <= `STATE_SERIAL;
			end
			else if(STATE == `STATE_SERIAL) begin
				approx[n+1] <= approx[n+1] + RK4COARSE;

				TN <= TN + HCOARSE;
				YN <= approx[n+1] + RK4COARSE;				// originally
RK4COARSE
				HCOARSE <= ((T1-T0)/`N);
```

```
            HFINE <= HCOARSE/`NFINE;
            START <= 0;

            // need to convert N to an integer
            // originally `N-1
            if(n < `N-1) begin
                NEXT_STATE <= `STATE_SERIAL;
                n <= n+1;
            end
            else begin
                NEXT_STATE <= `STATE_SETTLE;
                n <= 0;
            end
        end
        else if(STATE == `STATE_SETTLE) begin
            k <= k+1;
            n <= 0;
            TN <= T0;
            YN <= Y0;
            HCOARSE <= ((T1-T0)/`N);
            HFINE <= HCOARSE/`NFINE;
            START <= 0;

            for(y=0; y<=`N; y=y+1) begin
                lastApprox[y] <= approx[y];
            end

            if(k < `K) begin
                NEXT_STATE <= `STATE_PARALLEL;
            end
            else begin
                NEXT_STATE <= `STATE_END;
            end
        end
        else if(STATE == `STATE_END) begin
            // just outpu approx somewhere
        end
    end


    RK4 rk4coarse(
        .H( HCOARSE ),
        .TN( TN ),
        .YN( YN ),
        .OUTPUT( RK4COARSE )
    );
```

```
        // change this to a simple RK4 module since it only does one iteration
        genvar a;
        generate

                for(a=0; a<`N; a=a+1) begin: test
                        RK4Full coarse(
                                .CLK(CLK),
                                .START(START),
                                .H(HCOARSE),
                                .T0(T0+(HCOARSE*a)),
                                .Y0(lastApprox[a]),
                                .N(1),
                                .YN(correction2[a+1]),
                                .DONE()
                        );
                end

        endgenerate

        genvar b;
        generate

                for(b=0; b<`N; b=b+1) begin: test2
                        RK4Full fine(
                                .CLK(CLK),
                                .START(START),
                                .H(HFINE),
                                .T0(T0+(HCOARSE*b)),
                                .Y0(lastApprox[b]),
                                .N(`NFINE),                         // originally `NFINE
                                .YN(correction[b+1]),
                                .DONE(DONE[b])
                        );
                end

        endgenerate

endmodule
```

```
module RK4Full(CLK, START, H, T0, Y0, N, YN, DONE);

        input                           CLK;
        input                         START;
        input       [63:0] H;
        input       [63:0] T0;
        input      [63:0] Y0;
        input      [15:0]  N;
        wire        [63:0] YNP1;
        output reg [63:0] YN;
        output reg                        DONE = 0;

        reg                    [63:0] TN;

        integer                                    n;

        reg                                                    processing = 0;

        always @(posedge CLK) begin
                if(!START) begin
                        YN <= Y0;
                        TN <= T0;
                        processing <= 0;
                        n <= 0;
                        DONE <= 0;
                end
                else if(START && !processing && !DONE) begin
                        YN <= Y0;
                        TN <= T0;
                        processing <= 1;
                        n <= 0;
                        DONE <= 0;
                end
                else if(processing) begin
                        YN <= YNP1;
                        TN <= TN + H;
                        n <= n+1;

                        // originally N-1
                        if(n < N-1) begin // might want to change this to a specific number of
iterations for safety
                                DONE <= 0;
                                processing <= 1;
                        end
                        else begin
                                DONE <= 1;
                                processing <= 0;
                        end
                end
                else begin
                        YN <= YN;
                        TN <= TN;
                        processing <= 0;
                        DONE <= 1;
                end
        end

        RK4 rk4(
                .H(H),
                .TN(TN),
                .YN(YN),
                .OUTPUT(YNP1)
        );

endmodule
```

```
module RK4(H, TN, YN, OUTPUT);
    input  [63:0] H;
    input  [63:0] TN;
    input  [63:0] YN;
    output [63:0] OUTPUT;

    //wires
    wire [127:0] K1;
    wire [127:0] K2;
    wire [127:0] K3;
    wire [127:0] K4;

    wire [127:0] tempH;
    wire [127:0] tempTN;
    wire [127:0] tempYN;

    wire [127:0] tempOUT;

    // assign the temporary variables for mathematical calculations
    assign tempH[127:64]  = 64'b0;
    assign tempH[63:0]    = H;

    assign tempTN[127:64] = 64'b0;
    assign tempTN[63:0]   = TN;

    assign tempYN[127:64] = 64'b0;
    assign tempYN[63:0]   = YN;


    //          http://stackoverflow.com/questions/5028986/c-emulated-fixed-point-division-
multiplication?rq=1
```

```
	// REFER to MARTIN STONE's POST ON HOW FIXED POINT MULTIPLICATION
WORKS - IT'S NOT JUST A SIMPLE MULT AND SHIFT!

	fun k1(
		.T( tempTN ),
		.Y( tempYN ),
		.EVAL(K1)
	);

	fun k2(
		.T( tempTN + tempH/2 ),		// don't need to worry about shifting by 64 here
		.Y( tempYN + (((tempH/2)*K1)>>32) ),
		.EVAL(K2)
	);

	fun k3(
		.T( tempTN + tempH/2 ),
		.Y( tempYN + (((tempH/2)*K2)>>32) ),
		.EVAL(K3)
	);

	fun k4(
		.T( tempTN + tempH ),
		.Y( tempYN + ((tempH*K3)>>32) ),
		.EVAL(K4)
	);

	//assign OUTPUT = YN + temp3[63:0];

	assign tempOUT = tempYN + ((tempH/6*(K1+2*K2+2*K3+K4))>>32);
	assign OUTPUT  = tempOUT[63:0];

endmodule
```

```
module fun(T, Y, EVAL);
	input  [127:0] T;
	input  [127:0] Y;
	output [127:0] EVAL;
	//evaluate the function y' = yt

	assign EVAL = Y;

	//alu alu1(
	//	.INPUT1(T),
	//	.INPUT2(Y),
	//	.OPCODE(4'b0010),
	//	.OUTPUT(EVAL)
	//);
endmodule
```

```
module alu(INPUT1, INPUT2, OPCODE, OUTPUT);
	input		[63:0] INPUT1;
	input		[63:0] INPUT2;
	input		[3:0]	OPCODE;
	output reg  [63:0] OUTPUT;

	// OPCODE = 0 <- ADD
	// OPCODE = 1 <- SUB
	// OPCODE = 2 <- MUL
	// OPCODE = 3 <- DIV
	always @(*) begin
		if(OPCODE == 0) begin
			OUTPUT = INPUT1 + INPUT2;
		end
		else if(OPCODE == 1) begin
			OUTPUT = INPUT1 - INPUT2;
		end
		else if(OPCODE == 2) begin
			OUTPUT = (INPUT1*INPUT2) >> 32;
		end
		else begin // OPCODE == 3
			OUTPUT = (INPUT1/INPUT2) << 32;
		end
	end
endmodule
```

```
`timescale 1 ns / 1 ns


module FPGADiffEq_test();


  reg      CLK;
  reg      RESET;


  wire [63:0] value;



  parareal UUT(
   .CLK(CLK),
   .RESET(RESET),
   .Y0(64'd1 << 32),
            .T0(64'd0 << 32),
            .T1(64'd5 << 32),
            .value(value)
  );

  initial begin
   CLK = 1'b0;
   forever begin
    #10
    CLK = ~CLK;
   end
  end


  initial
  begin

   RESET    = 1'b0;

   #100;

   $display("TESTING FPGADiffEq");
   RESET = 1'b1;
   #20;
   RESET = 1'b0;
   #20;

   // delay until Parareal is done running
            // length needs to be adjusted based on number of steps
   #100000;

            $display("%d", value);

            $stop;
  end

endmodule
```

[1] http://www.terasic.com/downloads/training/de1-soc/1_DE1_SOC_Introduction.pdf
[2] http://sourceforge.net/projects/win32diskimager/
[3] ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE1-SoC/DE1_SoC_User_Manual.pdf
[4] ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE1-SoC/DE1_SoC_User_Manual.pdf

[5] http://www.terasic.com/downloads/cd-rom/de1-soc/
[6] http://www.terasic.com/downloads/cd-rom/de1-soc/DE1SoC_SystemBuilder_V102.zip
[7] https://goo.gl/1Qov15
[8] http://rocketboards.org/foswiki/view/Documentation/GSRDGhrd
[9] https://www.youtube.com/watch?v=2WUkEt4-Q7Q
[10] http://www.linear.com/solutions/5176
[11] http://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx
[12] https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods
[13] https://en.wikipedia.org/wiki/Parareal
[14] http://www.cfm.brown.edu/people/jansh/page5/page10/page40/assets/Field_Talk.pdf
[15] http://www.cfm.brown.edu/people/jansh/page5/page10/page40/assets/Field_Talk.pdf
[16] http://www.cfm.brown.edu/people/jansh/page5/page10/page40/assets/Field_Talk.pdf
[17] https://en.wikipedia.org/wiki/Parareal
[18] http://www.cfm.brown.edu/people/jansh/page5/page10/page40/assets/Field_Talk.pdf
[19] https://en.wikipedia.org/wiki/Parareal
[20] http://www.wolframalpha.com/
[21] http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=4
[22] http://ds.arm.com/altera/altera-community-edition/