EE6302 MICROPROCESSOR SYSTEMS

FALL 2014

FINAL PROJECT REPORT

SOLAR SYSTEM SIMULATION

Computation and Visualization

By,

Sudarshan Rajagopalan
(sudarshan.rajagopalan@utdallas.edu)

Contents

1 Objective:

- To implement the communication between the microcontroller and the PCs.
- To use the cortex M4 launch pad to make mathematical calculations to simulate the motion of a set of planetary objects as governed by the effects of the forces of gravity on each other in computation and visualization mode.
- To display the motion of the planetary objects on the GUI in the visualization mode.
- To optimize the precision and the efficiency of the solution and understand the trade-offs between the two.

2 Tools Required:

2.1 Hardware:

ARM TIVA Launchpad, PC (for display purpose) and Micro USB to USB cable

2.2 Software:

Code composer studio, Microsoft Visual Studio

3 Project Description:

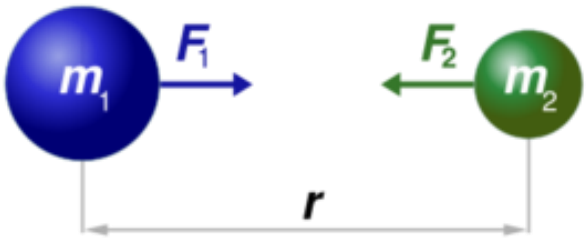
This project demonstrates the simulation of Newtonian n-body problem with the given set of information of the properties of each body in a text file that is to be communicated from the PC to the cortex M4 launch pad. Here, the accurate position and the velocity of each body has to be calculated with a given timestep. The assumptions made are that there are no collisions and the bodies are in vacuum. Apparently, the parameters that influence the position of the objects are the mutual forces of gravity on each object due to all other objects and the initial velocities of the objects.

The project demands us to perform the simulation in two different modes, namely:

- **Computation mode**
- **Visualization mode**

3.1 Computation mode:

In this mode, the calculation of the final positions and velocities of all the bodies are calculated for the given time period. The mutual force on each body is calculated by the using the Newton's law of universal gravitation:



$$F_1 = F_2 = G\frac{m_1 \times m_2}{r^2}$$

Where,

G	– Gravitational constant
F1 = **F2**	– Force between two objects
m1, **m2**	– Masses of the first and the second object
r	– Distance between the objects

- The velocity is calculated by the general formula Velocity = Distance * Timestep
- So, here the change in velocity is calculated as dV = F_{Net}* timestep, where F_{NET} is net force on each body
- The new velocity is updated as, V = dV + V (along both x and y axis)
- The new position is calculated as, New position = Initial Position + (new velocity * timestep)
- These new values are written onto a text file and this loop continues until the required number of iterations are reached.

3.2 Visualization mode:

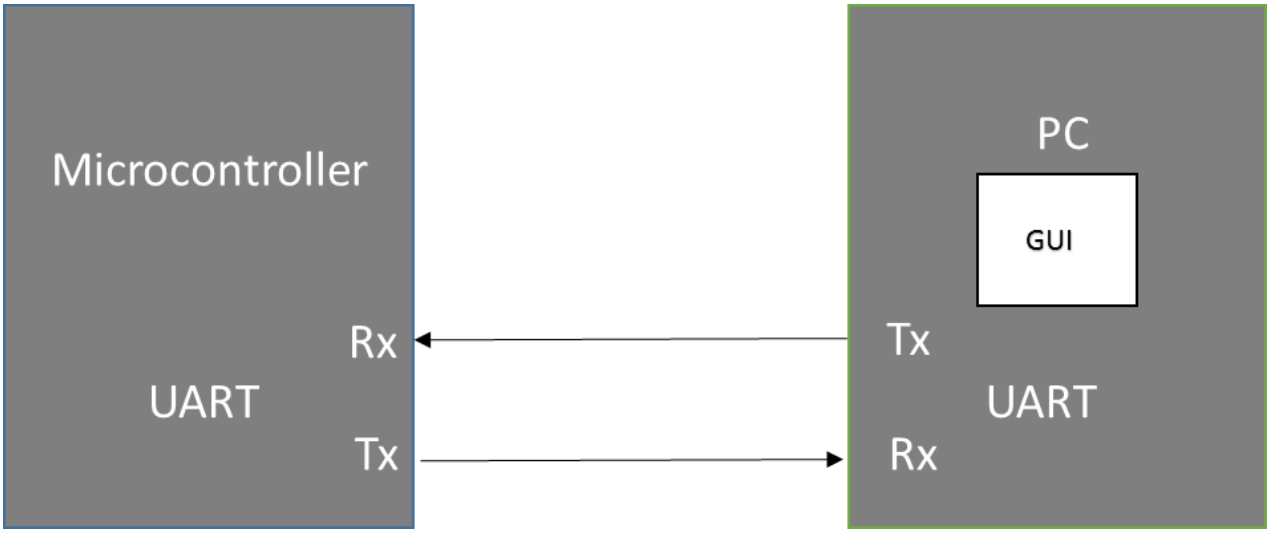
In this mode, the new position of the object has to be communicated to the GUI that will enable it to display the motion of the planetary objects. In order to implement the GUI, any programming language could be used. The language used in our project is C# and Microsoft Visual Studio software is used to implement the GUI.

So, in this mode, basically, the initial position values are converted into pixel values using a function and then the pixel values are communicated to the GUI through UART. Then, the calculation of force, positions, velocities are calculated by the same algorithm as explained in the computation mode and the information is sent to the PC, the PC reads the values and updates it on the GUI that would finally update the visualization of the bodies. One important factor to be considered in this is, the visualization has to be updated at least 24 times per second in order to have an undistorted smooth visualization.

UART is used to establish communication between the microcontroller and the PC and a micro USB to USB cable is used to connect the microcontroller to the PC.

4 Block Diagram:

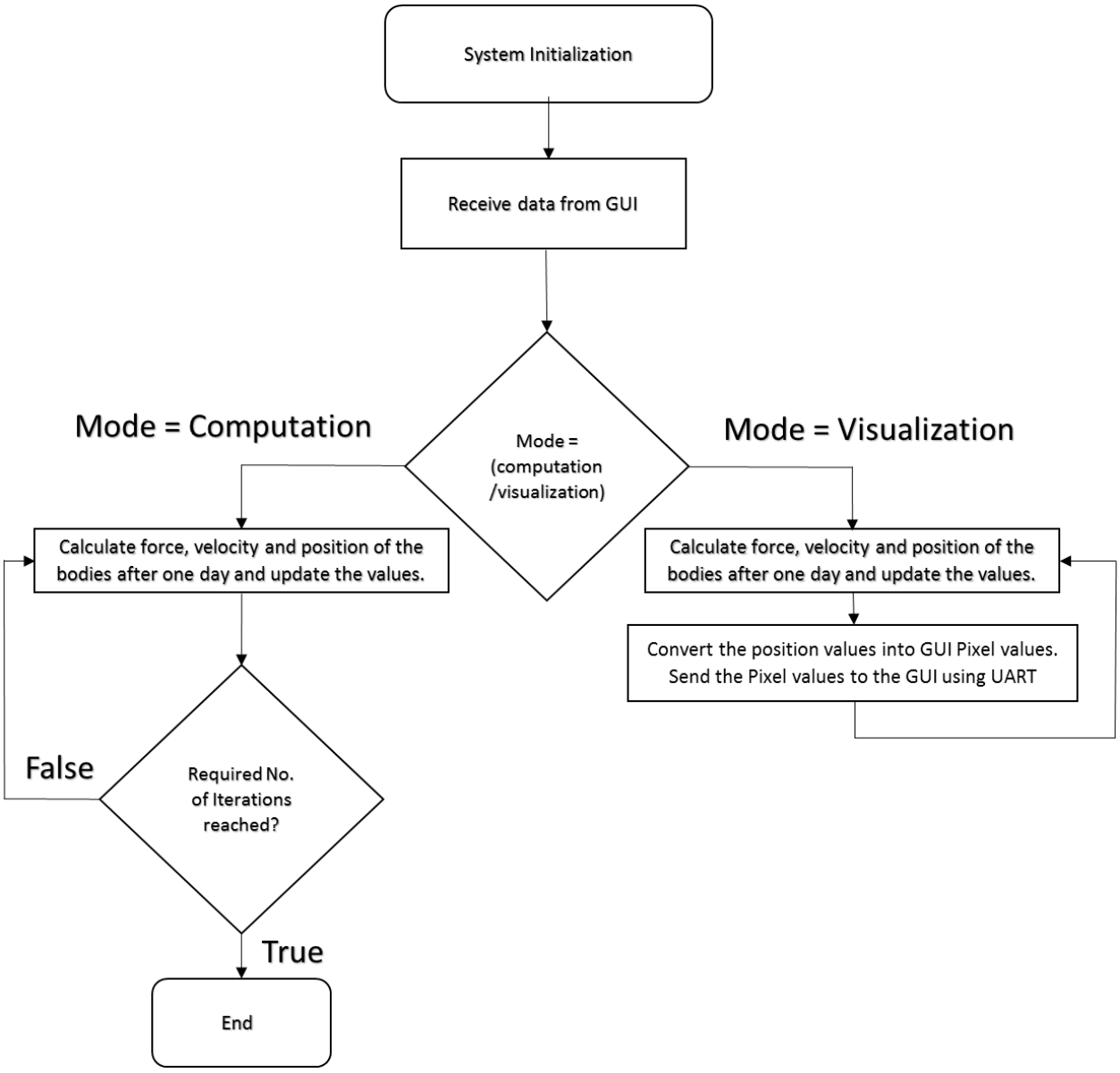
The following block diagram depicts the working of the project in a simplified manner:



5 Implementation:

5.1 Microcontroller Implementation:

The implementation of microcontroller in order to achieve the desired results are explained in the following flowchart:



5.1.1 System Initialization:

- Floating Point Unit and FPU Lazy stacking is enabled in order to avoid the latency caused due to interrupts.
- The clock frequency is set to 50 MHz.
- Clock, GPIO pins, Global Interrupts, UART Rx, Tx, UART interrupts are all enabled.

5.1.2 Receive Data from GUI:

The required input data like the total number of bodies, number of iterations, mass (M), initial position along X and Y directions (Px, Py), initial velocity along X and Y directions (Vx, Vy) of all the bodies are received from the GUI.

5.1.3 Select mode:

1) Computation mode:

- If the mode is selected to be computation, the calculations of the mutual force on all the bodies are executed by the formula of Newton's law of gravitation F = Gm1m2/r2
- The change in velocity is calculated by, dv = netforce*timestep/mass (both x and y)
- The new velocity is calculated by, dV + V (both x and y)
- The new position is calculated by, Position = position + (Velocity * timestep) (both x and y)
- These steps are continued until the number of iterations are completed and the values are written after each iteration.

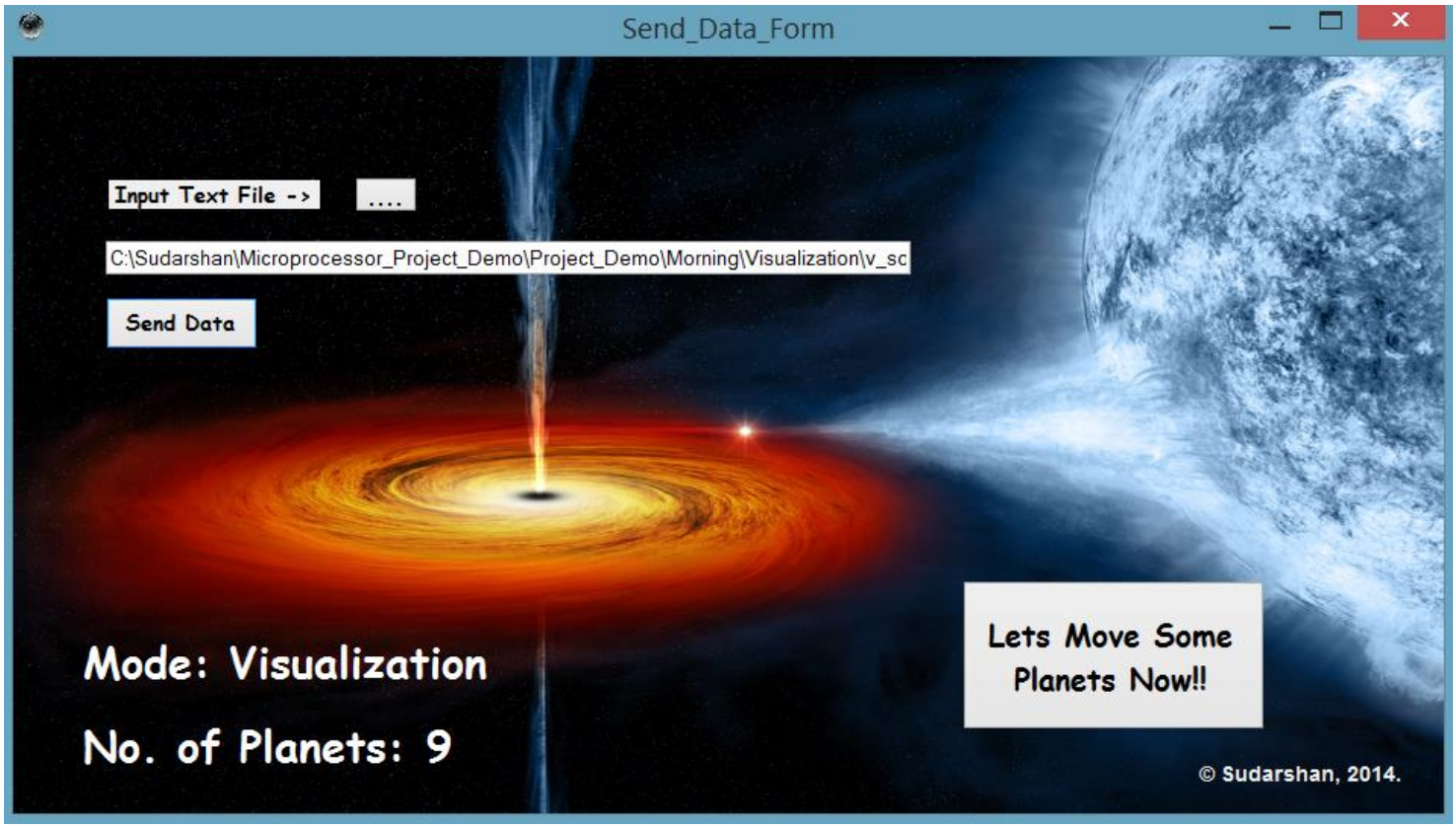
2) Visualization mode:

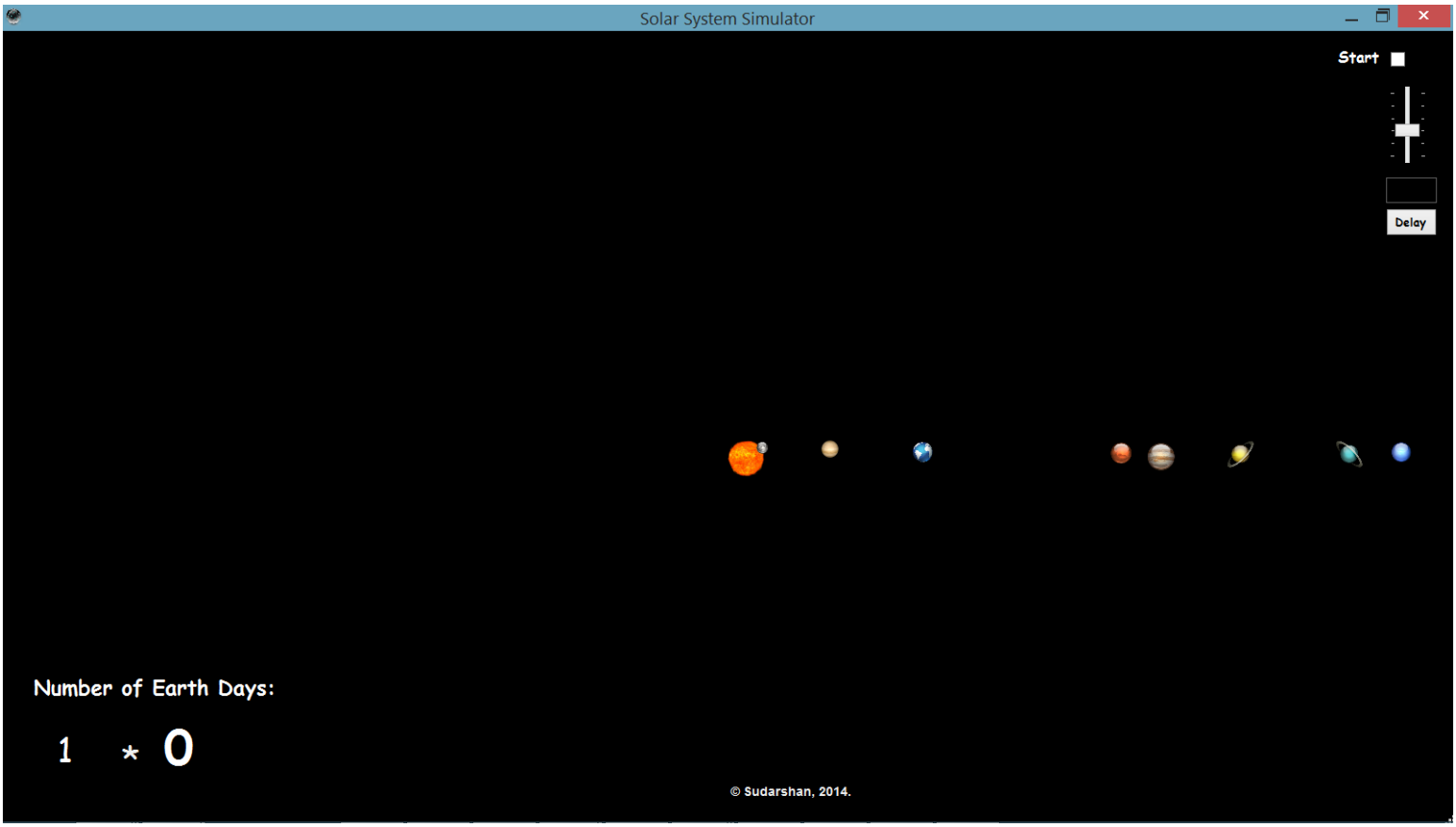
- If the mode is selected to be visualization, the calculations are performed, the values are converted into pixel values to be sent to the GUI.
- After this step, the loop continues and the values are sent to the GUI after every 24 hours of time and this updates the position of the planetary bodies every day.
- In this mode, all the planetary objects and their positions are displayed on the PC continuously which looks like an actual solar system with planets revolving in their own orbit.

5.2 GUI Implementation:

C# programming language is used in order to implement the GUI. The planetary objects are given shapes with different image files and are used in visualizing the motion of the bodies in the GUI. Here are a few screenshots of how the GUI looks:

SOLAR SYSTEM SIMULATION





6 Optimization:

6.1 CCS Optimization:

In order to optimize the whole program in CCS, few optimization properties are set like:

- Set the optimization level: 4(highest)
- Speed : 5(highest)
- Floating point mode: strict

6.2 Microcontroller Code Optimization

In order to improve the speed of execution of the microcontroller:

- The clock frequency is set to 50 MHZ
- "float" data type can be used instead of "double" during calculation.
- The force calculation algorithm also could be optimized

7 CPU Utilization:

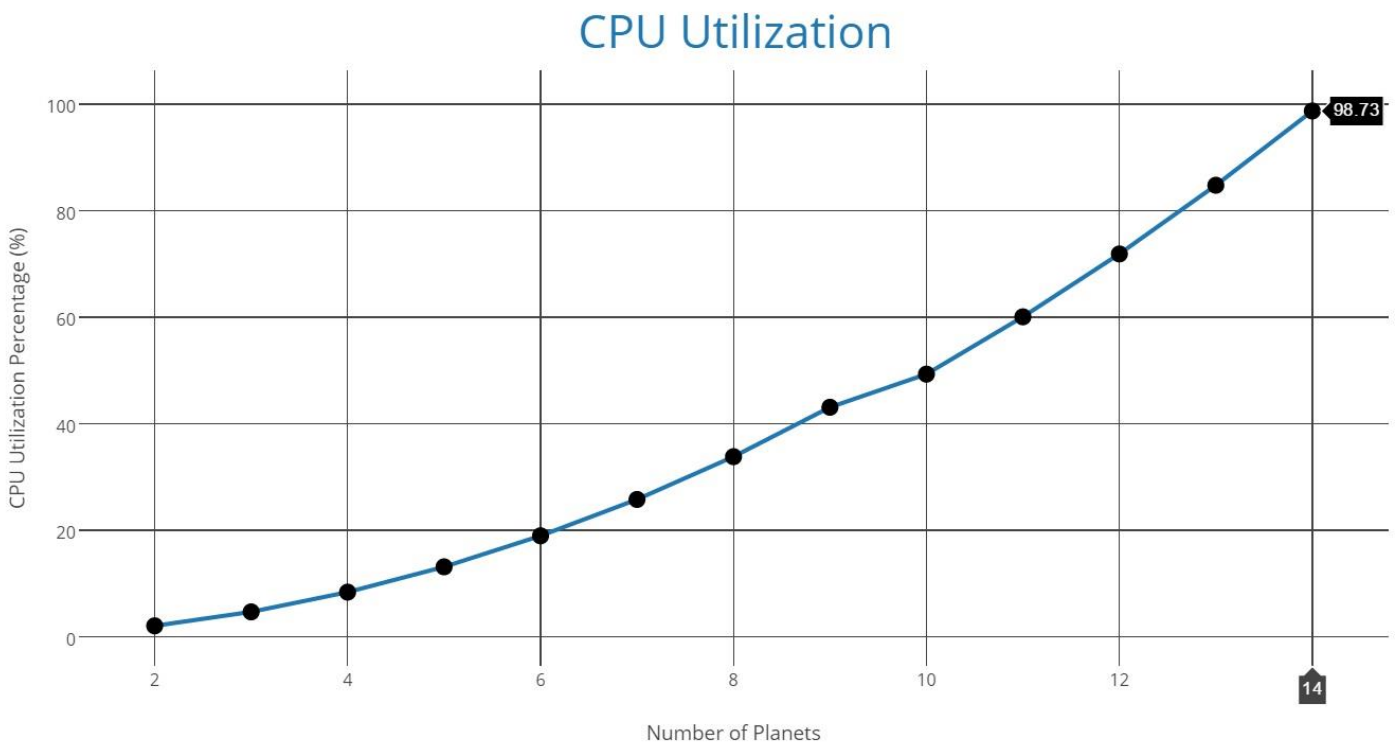
The CPU utilization of the Cortex-M4 is calculated in the following way:

For a refresh rate of 50 frames/second, we implement a timer every 20ms to start the force calculation function. In this period of 20ms, we calculate the time taken by the Cortex-M4 to finish the force calculation of all the bodies for one day. We then divide this time by the 20ms period to get the CPU utilization for force calculation.

For the 20ms timer ticks, we use the basic Timer module of the Cortex-M4. And for calculating the time required for force calculation, we find the number of clock cycles it took from start till end of the calculation. This is done by using Cycle Count Register available in the ARM Cortex-M4. The counter has to be initialized using the following code snippet:

```
HWREG(NVIC_DBG_INT) |= 0x01000000;   /*enable TRCENA bit in NVIC_DBG_INT*/
HWREG(DWT_BASE + DWT_O_CYCCNT) = 0;   /* reset the counter */
HWREG(DWT_BASE) |= 0x01;                    /* start the counter */
```

The CPU utilization graph for 15 bodies, 20ms timer tick, and system running at 40MHz is:



As observed from the graph, for an embedded system (Cortex-M4) running at 40MHz clock frequency and timer period of 20ms, the maximum bodies that it can take is 14 bodies. This can be improved by increasing the clock frequency (but at the expense of power consumption) and increasing the timer period (but at the expense of slower refresh rate).

8 Lab Questions:

1. **What problems have you encountered in this lab?**
   When trying to send data to the GUI using UART, there is a delay seen in the GUI at which the data received event is being generated. This is probably due to Windows Serial Port driver being outdated.

2. **What is the maximum number of bodies you can simulate?**
   Twenty bodies could be simulated for a refresh rate of 25 frames per second. Fourteen bodies for a refresh rate of 50 frames per second.

3. **How do you optimize your algorithm to shorten the calculation time?**
   In order to shorten the calculation time:
   - The clock frequency is set to 50 MHZ
   - "float" can be used instead of "double" during calculations

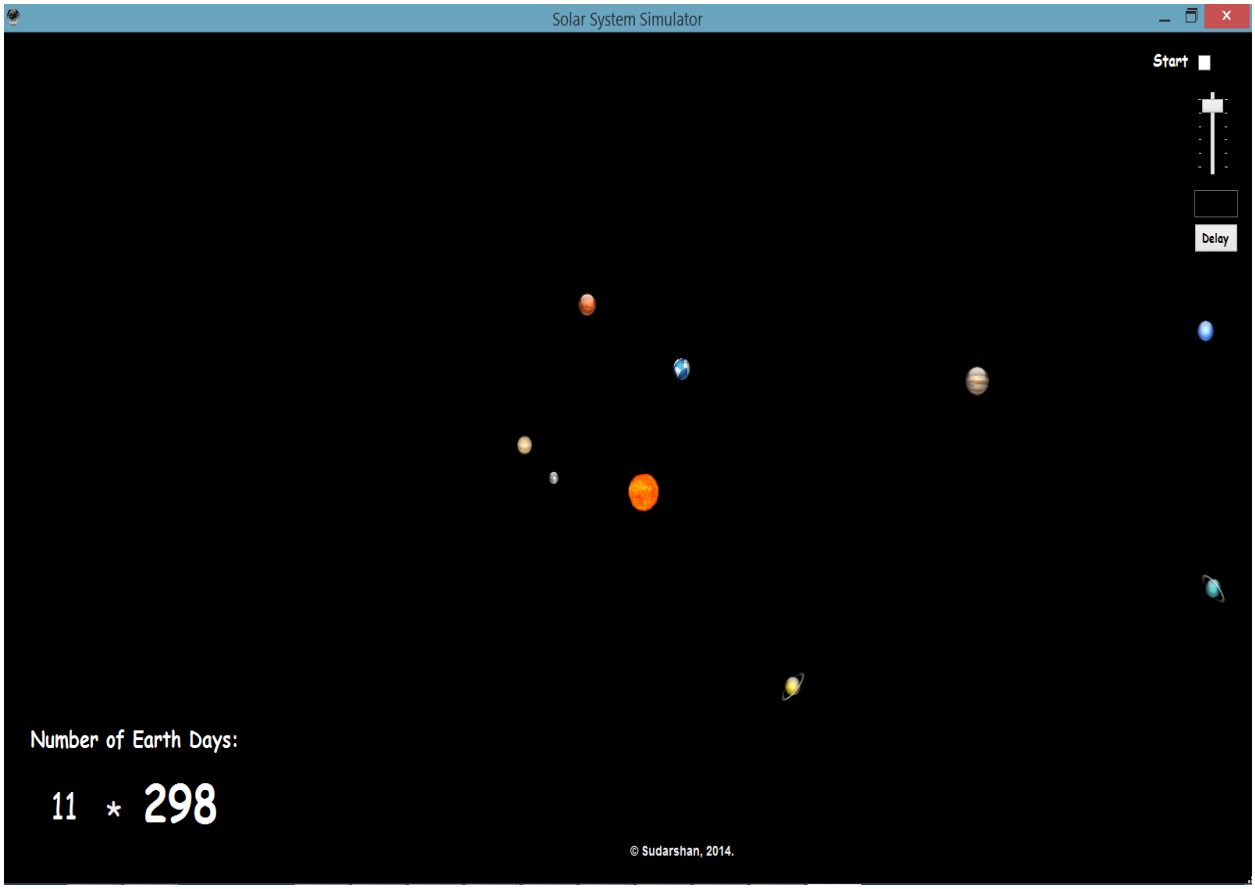
4. **How do you maintain precision of numbers for accuracy?**
   In order to maintain the precision of calculated values for accuracy, long double could be used instead of float in the program code.
   Also, cortex M4 is used in which FPU is present, that fully supports single-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. It also provides conversions between fixed-point and floating-point data formats, and floating-point constant instructions. FPU supports:
   - 32-bit instructions for single-precision (C float) data-processing operations
   - Combined multiply and accumulate instructions for increased precision (Fused MAC)
   - Hardware support for conversion, addition, subtraction, multiplication with optional accumulate, division, and square-root.
   - Decoupled three stage pipeline. This feature helps in maintaining the precision of numbers for accuracy.

5. What is the percentage CPU utilization?

The graph of CPU utilization vs number of bodes is shown above. For a no. of bodies of 10, the CPU utilization is 50% and reaches max of 98.73% for number of bodies of 14.

9 Project Results:

- The simulation of the given number of planetary objects were done in both the computation and the visualization mode.
- We were able to perform the simulation of 9 bodies for a period of 100 years in the computation mode and it took around 5 minutes to calculate the values.
- With the increase in number of bodies, the calculation time increases which in turn makes the visualization slower.
- The GUI was successfully implemented to visualize the motion of 9 planets.



10 Conclusion:

- The simulation of the N-body problem was achieved successfully by implementing communication between the Launchpad and the PC using UART. All the computations are performed on the Launchpad and the PC is used to display the computation and the visualization part.
- C# is used to implement the GUI that is used to display the motion of all the planets in their respective orbits. A few optimizations has been done in order to get accurate output values. On the other hand, we have also used certain techniques to improve the speed of execution.
- UART communication module seems to suffice the required communication between the uC and GUI (PC).
- The CPU utilization is also analyzed. Thus, the solar system with all the nine planets were simulated in computation and visualization mode successfully.

11 References:

[1] *http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf*
N-body problem – wikipedia
[2] *www.ti.com/lit/pdf/spms376*
Tiva™ TM4C123GH6PM Microcontroller DATA SHEET
[3] *http://www.csee.umbc.edu/~motteler/parpro/proj2/*
The N-body Problem with MPI
[4] *http://processors.wiki.ti.com/index.php/Optimizer_Assistant*
Optimizer Assistant