Implementation of High Precision Orbit Propagator on TI Digital Signal Processor

Naveed Ahmed
Satellite Research & development Centre Karachi, SUPARCO Pakistan
naveedahmed87@live.com

Abstract—Once the satellite is in space, it is very important to make sure that it is in its correct and prescribe orbit, along with the appropriate attitude, at any given point of time. Orbit propagators are used to determine propagated position and velocity of the satellite. Various perturbations, faced by the satellite in the space, are also incorporated to portray the actual picture. An on-board implementation of the orbit propagator is desired for real time propagation of the orbit. All the algorithms are developed for the low earth orbit (LEO) satellites. The propagator is implemented on performance efficient and high speed digital signal processor (DSP) TMS320C6713, a product of Texas Instrument. The key feature of this processor, real time data exchange (RTDX), is exploited to show our results on the host on-board computer.

Keywords— Propagator, Perturbations, LEO, Embedded, DSP, C6713, RTDX

1 INTRODUCTION

Orbit propagation is the process of modelling the orbit for an extended time period using the dynamic equations of motion, models of environmental forces, torques and other physical parameters. In any space mission analysis, prediction of the orbits of satellites is an essential part and it, directly or indirectly, has the impacts on the satellite's power system, attitude control, thermal design and other systems. The problem of reckoning the orbits of satellites, however, is not straightforward. The main factors affecting the orbit of a satellite are: the non-spherical geometry of Earth, atmospheric drag, perturbed effects from the gravitational pull of the Sun and other planets, electromagnetic forces, radiation pressures and so forth. There are a number of propagators available in the space trade e.g. Two Body, SGP4, HPOP. Though for embedded applications, it is necessitated that a customized algorithm will be used with on-board computer to propagate the orbit in real time.

2 ORBIT PROPAGATOR ALGORITHM

The Simulink model of the propagator is shown in figure 1. It takes the current values of the Keplerian elements and computes the current position and velocity from those values. These values are then propagated in the simulated environment incorporating different order perturbations mentioned above.

The higher order calculations are refrained due to the fact that higher order harmonics of these perturbations are extremely small to bring any substantial change in results except for special cases e.g. in geo synchronous orbit, a satellite remains largely over one location on Earth; that's why the sectoral harmonics, which are of order 12, are present all the time [1].

Figure 1 - Physical interpretation of sectoral harmonic

The result position and velocity are converted them from one reference model i.e. ECI (Earth Centred Inertial) to ECF (Earth Centred Fixed). Finally, the orbital elements will be extracted back from the values of propagated position and velocity. These values of the orbital elements are the final outputs and, henceforth, give the deviation of the satellite's orientation in presence of the aforesaid perturbations.

3 SYSTEM'S DESIGN FLOW

The typical design flow, which has been followed during the entire design and implementation phase, is shown in the following titled matrix.

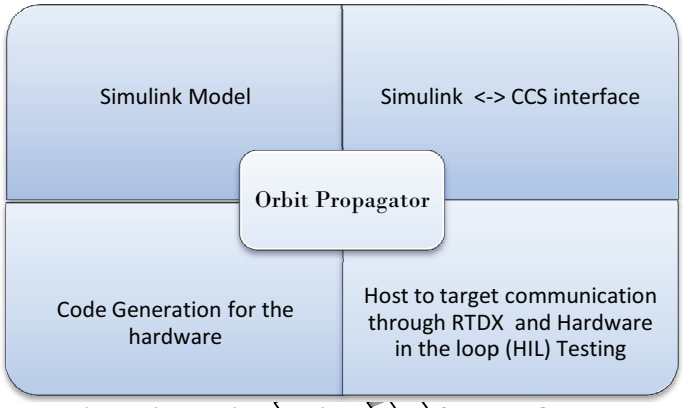

Figure 2 - Typical Design Flow of the HPOP model implemented on DSP

4 SIMULINK MODEL AND ITS OPTIMIZATION

The design is initiated by building the Simulink model. This model uses the embedded MATLAB function blocks. These blocks contain complete algorithm, incorporating all the major perturbations, based on MATLAB programming language. Up till now, the model has no link with the DSP or Code Composer Studio (software required to develop and program the systems on the DSP target devices). Once the model is completed in MATLAB, its working is tested and the results are matched using the commercially available simulation tool.

The use of optimization of the model has a mixed tincture of goodness and otherwise at the same time. More optimizations cause the compiler to take more time to compile a code as it has to take care of compilation constraints. But it makes code faster to execute and performs efficiently than a pessimistic one. Various optimization strategies have been followed during the design of the entire Simulink model. Some of them are listed below:

I. The signals are once defined and stored in the data memory. This prevents us from re-initialization and saves plenty of hardware resources such as data memory.

II. The block outputs are declared in the local scope, wherever possible, to reduce the target device's global RAM usage.

III. The existing global buffers are used to store temporary results. This reduces the number of local variables in the code and hence the memory requirement.

Floating point calculations are not always recommended/ required. For floating point to integer conversions, the simulation handles out-of-range values by saturation or by modulo two wrapping strategy. This enhances the efficiency of the code as the floating point calculations consume more software and hardware resources.

5 REAL TIME DATA EXCHANGE (RTDX)

Real-Time Data Exchange provides real-time, continuous visibility into the way target applications operate in the real world. RTDX allows system developers to transfer data between a host (which is normally a PC or the other co-processor) and target devices without interfering with the target application. The data can be analyzed and visualized on the host using any host client. This shortens the development time by giving you a realistic representation of the way your system actually operates.

RTDX consists of both target and host components [2]. A small RTDX software library runs on the target application (RTDX.lib). The target application makes the function calls to this library's Application Programming Interface (API) in order to pass data to or from it. This library makes use of a scan-based emulator (can be either USB or the onboard Joint Test Action Group [commonly known as JTAG] interface) to move data to or from the host platform. Data transfer to the host occurs in real-time while the target application is running.

On the host platform, an RTDX Host library operates in conjunction with Code Composer Studio. Displays and analysis tools communicate with RTDX via an easy-to-use COM API to obtain the target data and/or to send data to the target application. It allows for data exchange between the host PC (or any other processor) and the target DSP Starter Kit (DSK), as well as analysis in real time without stopping the target.

A. RTDX Architecture

The architecture of RTDX, up to a great extent, matches with that of HSRTDX (High Speed Real Time Data eXchange). It relies on a silicon module (referred to as the RTDX unit) that provides an interrupt-driven DMA (Direct Memory Access) interface between the EMU pin selected for RTDX data transport (EMU0 is the default) and the memory system of the target device. The EMU pin used for RTDX data transport may be modified through the RTDX configuration tool. All memory spaces addressable by the processor are accessible to the RTDX (or HSRTDX) unit. It uses three interrupts (INT3, INT11 and INT12 in the C6x1x™ DSP and C64xx™ DSP devices) to facilitate communications with the emulator, and to signal the RTDX library when a transfer is complete. The following diagram (figure 4) shows the basic RTDX target architecture.

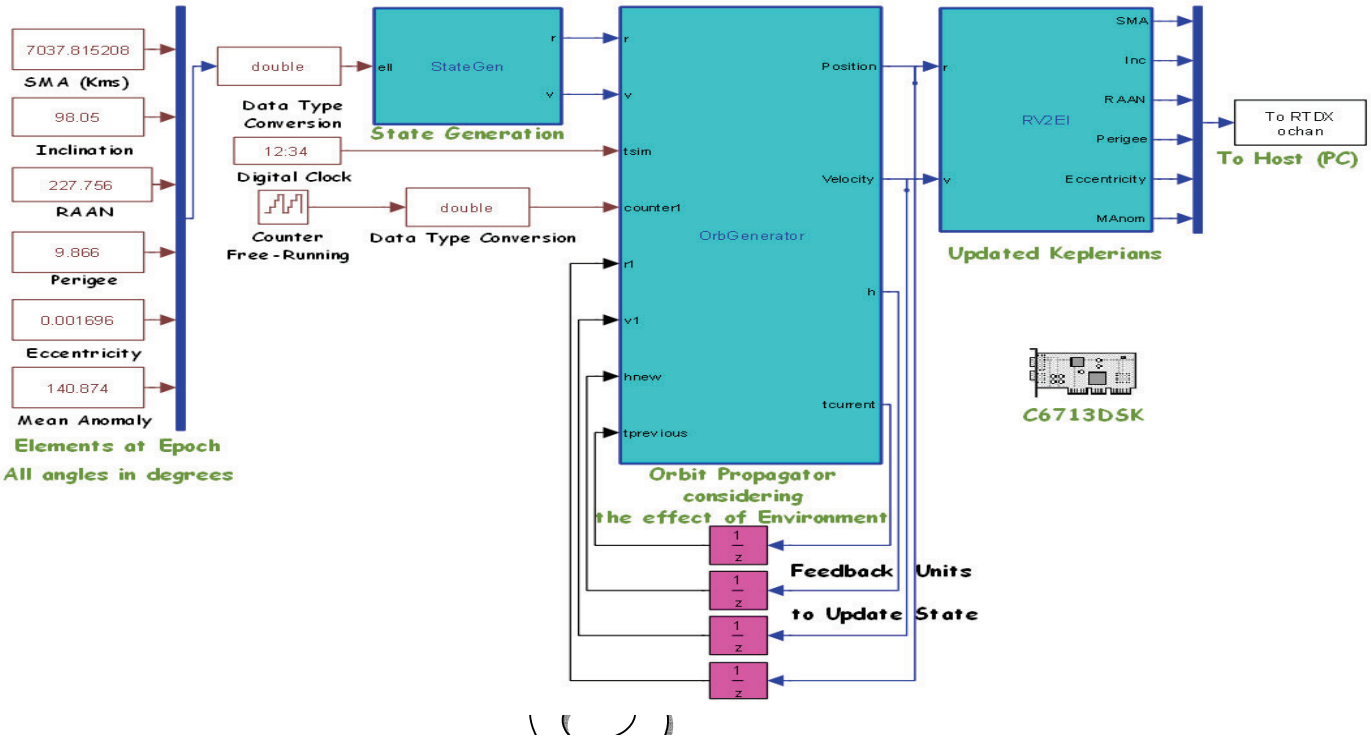

Figure 3 - Orbit Propagator Simulink Model

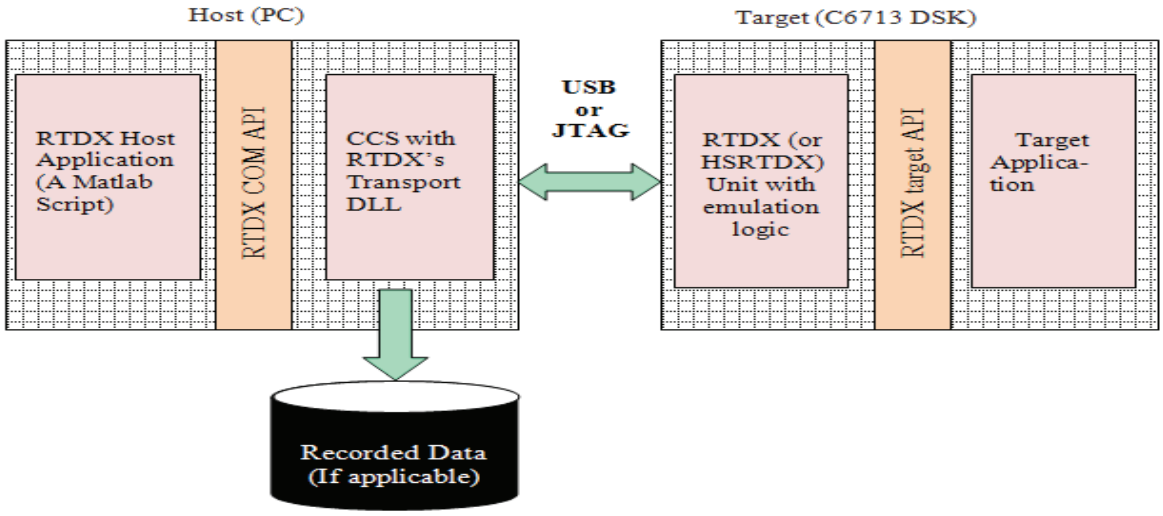

Figure 4 - Basic Architecture of RTDX Communication between host and target

B. Configuring the RTDX

Before actually using the RTDX module, it should be configured properly in CCS and MATLAB alike. In CCS, one can do this by selecting RTDX from Tools pull-down menu. This can be used further to invoke the RTDX configuration window. From the channel viewer control, it can be viewed which channels are currently enabled to communicate with.

C. Reading data by the host

A separate MATLAB script file (also called m-file) is written to read the RTDX-transmitted data on the host PC[3].This script file performs series of tasks that are inevitable before actually setting up communication with the target. It first requests the target board info by `ccsboardinfo` MATLAB command and then set up an object (usually a structure) against this information.It then resets the board to bring the target board in its initial state. RTDX module is enabled by `enable(cc.rtdx)` MATLAB command, where cc is the name of the object which has been created prior to execution of this command; it can be anything.The RTDX channel can be set up permanently or for any specific duration of time.

The RTDX channels are then created; if two way communication is required separated channel should be created for writing (transmit) and reading (receive) the data. The read channel needs to be created. It can be done by `open (cc.rtdx,'ochan1','r')` MATLAB command. '`outdata =readmsg(cc.rtdx,'ochan1','double',1);`' is another MATLAB command used in the said script file.

The read data is stored in buffer named '`outdata`'. It is always recommended to disable and close the channel when either it is timed out or we are not using it anymore. This saves our system resources from being used wastefully.

D. Advantages of using RTDX

RTDX is getting ubiquitous day by day. It is starting to find its usage for variety of applications ranging from automation to communication, from electronic systems to military applications and so forth keeping the imagination a limit. There are couples of advantages of using RTDX in presence of its counter-parts. The first and foremost advantage is its speed. The data rate achieved by the real time data exchange, supported by the C6713 devices, is around 10 KBps (kilobytes per second). The high speed real time data exchange is giving even faster data rate of 130 KBps (kilobytes per second)[3]. Secondly, it is also not voracious of ample hardware requirements and additional cables.

.

6 TESTING AND VERIFICATION OF RESULTS

In order to compare its results with those from commercially available simulation tool, various simulations are performed with the same epoch values of the orbital elements and the force models used in Simulink. The obtained results, both from MATLAB and commercially available simulation tool, are overlapped in figure 5

In future, the project can be refined in multiple ways to make it more practical. For instance, it can be used in the mission analysis of various satellites. The efficacy of the propagator can also be determined using this practice.

7 CONCLUSION

It was aimed to develop a customizable DSP based orbit propagator addressing the mission needs of a LEO satellite that simulates a real scenario considering all the perturbations influencing its propagation. The results were assessed against simulations of commercially available simulation tool and were found to be matched satisfactorily. In a nutshell, it can be asserted that the code provides high accuracy in predicting the orbit and is successfully embedded into the DSP processor for real time orbit propagation so that the state may be passed to other modules as needed.

REFERENCES

[1] James R. Wertz, Mission Geometry; Orbit and Constellation Design and Management, Chap 2, pp65, ISBN: 1-881883-07-8

[2] www.focus.ti.com/lit/an/spra895/spra895.pdf

[3] Rulph Chassaing, Digital Signal Processing and Applications with the C6713 and C6416 DSK, Ch 9, pp313, ISBN 0-471-69007-4

[4] www.processors.wiki.ti.com/index.php/Real_Time_Data_eXchange_(RTDX)

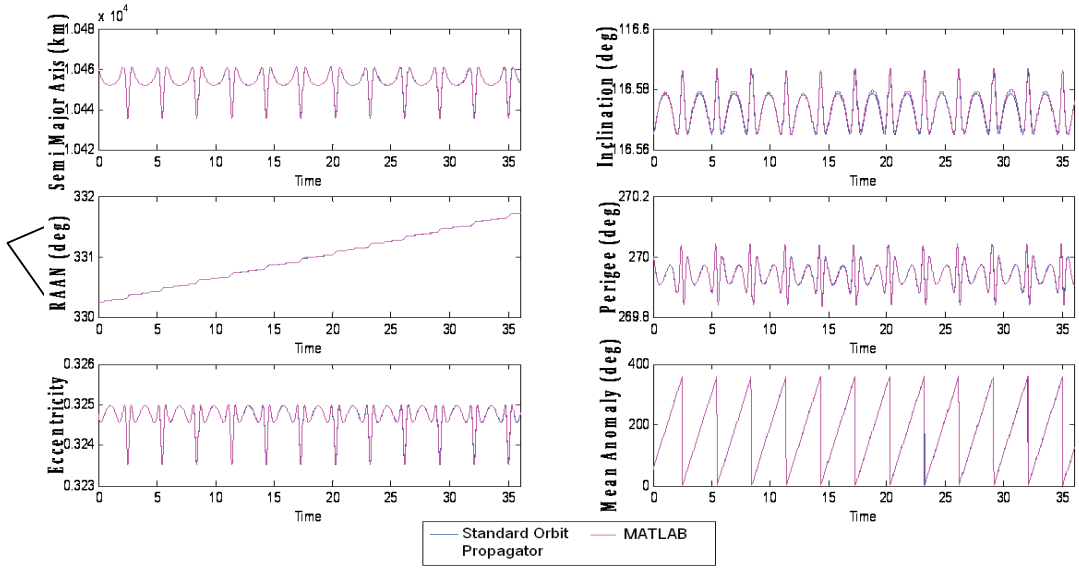

Figure 5 - Plots of Keplerian Elements Set between Commercially available simulation tool and Customized Embedded Orbit Propagator