A Modular FPGA-based Implementation of the Unscented Kalman Filter

Jeremy Soh, Xiaofeng Wu
School of Aerospace, Mechanical and Mechatronic Engineering
University of Sydney
Sydney, New South Wales 2006
Email: jsoh8701@uni.sydney.edu.au and xiaofeng.wu@sydney.edu.au

***Abstract*—Nanosatellites, while lowering the cost and ease of space access, suffer the issue of reduced performance compared to larger satellites, particularly when it comes to attitude determination and control. Field Programmable Gate Arrays (FPGAs) have been used in the past to make up for the shortfall in capability but tend to have more complicated development processes than general purpose microprocessors. To simplify development as well as promote portability and reusability between satellite missions, a hardware/software co-design of the Unscented Kalman Filter (UKF) implemented on a FPGA device is presented. The design is implemented on a Zynq-7000 XC7Z020 to establish proof-of-concept and verified using simulated data. The design achieved a 1.5× speed-up over a purely software implementation and the resource usage and power consumption are both low enough to be integrated into a full SoC.**

I. Introduction

Nanosatellites have gained popularity in recent years and have seen many successful missions, but many are still for educational or technology demonstration purposes rather than specific scientific objectives, in part due to the limitation in attitude determination and control capabilities [1]. Aggressive miniaturisation of the spacecraft structure and subsystems, not to mention the trend towards commercial-off-the-shelf (COTS) components, has meant cheaper, less precise sensors have to be used and complex algorithms that could potentially compensate for them tend to be infeasible due to limited computing power.

In order for nanosatellites to gain wider use for more specific scientific objectives, such as remote sensing, they also have to operate in a capacity where they can be considered beneficial over a single, larger satellite. One such proposal to this end is to have multiple satellites fly in formation with a homogeneous sensor set on-board; however, again, this approach requires higher accuracy pointing capabilities. Furthermore, each satellite's attitude determination and control system (ADCS) will likely need to be working at a high sampling frequencies in order to maintain the satellites' formation for this dispersed multi-nodal sensing. This of course results in an increased computational load which, as mentioned, is problematic for nanosatellites.

One approach to the computing issue has been to translate the necessary functionality into hardware which is then implemented on a Field Programmable Gate Array (FPGA) device. FPGAs are becoming increasing popular in space applications mainly for two reasons: they allow the implementation of computationally intensive algorithms and their reconfigurability allows them to recover from faults due to radiation or update the algorithms based on changing mission objectives. A FPGA also allows multiple functions to be implemented on a single chip, in a so called System-on-Chip (SoC), which frees up valuable real estate within the satellite normally required by multiple processors. An extreme case of this is where *all* of the satellite's computing is implemented on one or more FPGAs, however, while many of these types of missions are planned (e.g. [2]), few missions have actually flown.

Implementing functionality as hardware offers large performance gains over software implementations in terms of speed and power but tend to greatly increase the length and complexity of development as well as limit reusability. While FPGAs alleviate these issues somewhat compared to the traditional hardware approach of Application Specific Integrated Circuits (ASIC), even FPGAs have long development times when compared to the traditional software approaches. Attitude determination and control algorithms in particular will generally contain a system model to represent the satellite meaning if these algorithms are implemented solely in hardware, the hardware will be mission specific, i.e. developed with a specific satellite in mind. This can restrict the reuse of designs between missions as the algorithm's hardware may then be difficult to adapt if certain parameters change (e.g. the set of sensors or actuators); software implementations, however, can easily deal with such a change.

These issues are usually not a problem with larger satellites since usually only one is built but with the shift towards nanosatellites and the desire for mass production, insofar as a satellite can be mass produced, a more portable design for attitude determination and control that can at least be reused between successive missions is needed. This is especially true in academia, where nanosatellites are an attractive solution to space access, as many researchers desire a generic, stable, reusable platform they can use for their various scientific objectives; an approach that leverages the benefits of the FPGA while maintaining the flexibility and ease of development of the traditional general purpose microcontroller designs would then be desirable.

For state estimation, the traditional approach for aerospace

978-1-4799-5356-1/14/$31.00

applications has been to use the Extended Kalman Filter (EKF), mostly due to its low computational complexity yet remarkable accuracy. Another variant in the Kalman Filter (KF) series of algorithms, the Unscented Kalman Filter (UKF) [3], [4], has been gaining popularity recently in various applications [5], [6], [7], due to its superior performance and stability over the EKF [8], [9]. The improvement is mostly a result of its better handling of nonlinear system models, though comes at the cost of increased computing requirements; the computational *complexity* for the UKF is often higher than the EKF. Though this may be an issue for complex models using the traditional processor approach (e.g. [10]), hardware designed for the FPGA has the potential to reduce execution time back down to feasible levels.

A number of approaches exist to implementing such an algorithm in hardware though previous work has been mostly restricted to the basic KF or EKF. Quinchia and Ferrer implemented the KF entirely as a hardware IP core and attached it to a soft-core processor as a peripheral device [11]; this improved the performance of their algorithm but as alluded to earlier, restricts the implementation to a single application. Another approach is to modify a processor to use custom instructions that perform parts of the algorithm as in [12], but while this means the implementation can be easily used between applications, it is now restricted to a particular device. Using aspects of both approaches, Cruz et al. implement the majority of the calculations in hardware as multiple IP cores which are then attached to a soft-core processor bus as peripherals [13]; the system models however, are left in software on the processor and the data fed to peripheral cores over the bus. This implementation used the EKF however, and even then only the `update` step was implemented. Previous work by the authors had the UKF implemented directly into hardware as a standalone IP core [14] but as with the other approaches this lacks generality since it is specific to one model/application.

In this paper, a software/hardware co-design of the UKF implemented on a FPGA device is proposed. The overall goal is to create a generic UKF IP core that can be easily attached to a variety of processors, easily implemented on many different devices and easily adapted for different applications. With that in mind, portability and reusability being emphasised, the application specific parts of the algorithm, such as the system models, are implemented in software while the non-application specific parts are implemented as hardware IP cores. In order to establish proof-of-concept and viability, the algorithm is restricted to attitude determination (instead of a full closed-loop ADCS), simulated sensor data is used and since the target application is for use on nanosatellites, potentially in a SoC, area efficiency is favoured over timing performance.

The rest of paper is organised as follows: Section II presents an overview of the Unscented Kalman Filter (UKF), Section III presents an overview of the system and simulation models, Section IV details the proposed design, Section V presents simulation and synthesis results and Section VI concludes the paper.

II. Unscented Kalman Filter

Consider a general nonlinear system described by discrete time equations as:

$$\mathbf{x}_k = f\left(\mathbf{x}_{k-1}, \mathbf{w}_{k-1}\right) \tag{1a}$$

$$\mathbf{z}_k = h\left(\mathbf{x}_k, \mathbf{v}_k\right) \tag{1b}$$

where $\mathbf{x}_k$ is the state vector, $\mathbf{w}_k$ and $\mathbf{v}_k$ are the process and measurement noise respectively which are assumed to be zero-mean Gaussian white noise with covariances $\mathbf{Q}_k$ and $\mathbf{R}_k$, $\mathbf{z}_k$ is the measurement vector and f and h are the process and observation models respectively.

While the EKF handles the nonlinear system models by linearisation via the Jacobian of the models, the UKF instead applies the Unscented Transform (UT) which models the current state as a probability distribution characterised by a mean and covariance. The UT deterministically draws a set of points from the distribution, called sigma points, and propagates them through the system models, after which the mean and covariance of the transformed points are recovered and used to inform the new state.

For a system represented by (1), let the augmented state vector, $\mathbf{x}_k^a$, and the augmented state covariance, $\mathbf{P}_k^a$, be given by:

$$\mathbf{x}_k^a = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{w}_k \\ \mathbf{v}_k \end{bmatrix} \tag{2a}$$

$$\mathbf{P}_k^a = \begin{bmatrix} \mathbf{P}_k & 0 & 0 \\ 0 & \mathbf{Q}_k & 0 \\ 0 & 0 & \mathbf{R}_k \end{bmatrix} \tag{2b}$$

A number of sigma point selection strategies exist, but the method that requires the least number of points ($N + 2$, where N is the length of the augmented state vector), is the spherical simplex set of points [15], generated by:

Choose $0 \leq W_0 \leq 1$, then:

$$W_i = \left(1 - W_0\right) / \left(N + 1\right) \tag{3}$$

The vector sequence is initialised as:

$$\boldsymbol{\sigma}_0^1 = [0]\,, \quad \boldsymbol{\sigma}_1^1 = -\left[\tfrac{1}{\sqrt{2W_1}}\right] \quad \boldsymbol{\sigma}_2^1 = \left[\tfrac{1}{\sqrt{2W_1}}\right] \tag{4}$$

Then the vector sequence is expanded for $j = 2,\ \ldots,\ N$ via:

$$\boldsymbol{\sigma}_i^j = \begin{cases} \begin{bmatrix} \boldsymbol{\sigma}_0^{j-1} \\ 0 \end{bmatrix} & i = 0 \\ \begin{bmatrix} \boldsymbol{\sigma}_i^{j-1} \\ -\frac{1}{\sqrt{j(j+1)W_1}} \end{bmatrix} & i = 1,\ \ldots,\ j \\ \begin{bmatrix} \mathbf{0}_{j-1} \\ \frac{j}{\sqrt{j(j+1)W_1}} \end{bmatrix} & i = j+1 \end{cases} \tag{5}$$

The sigma points themselves are then drawn from:

$$\boldsymbol{\mathcal{X}}_{i,k} = \begin{bmatrix} \boldsymbol{\mathcal{X}}^x_{i,k} \\ \boldsymbol{\mathcal{X}}^w_{i,k} \\ \boldsymbol{\mathcal{X}}^v_{i,k} \end{bmatrix} = \hat{\mathbf{x}}^a_k + \left(\sqrt{\mathbf{P}^a_k}\boldsymbol{\sigma}\right)_i \quad i = 0,\ \ldots,\ N{+}1 \quad (6)$$

where i refers to the i-th column of the matrix product, $\hat{\mathbf{x}}^a_k$ is the current augmented state estimate, $\sqrt{\mathbf{P}^a_k}$ is the matrix 'square root' of $\mathbf{P}^a_k$ performed via Cholesky Decomposition [16] and $\boldsymbol{\mathcal{X}}^x_{i,k}$, $\boldsymbol{\mathcal{X}}^w_{i,k}$, $\boldsymbol{\mathcal{X}}^v_{i,k}$ are the sigma points associated with the state, process noise and measurement noise terms respectively.

A. Predict

Once the sigma points have been generated, the `predict` step begins with the sigma points being propagated through the system model:

$$\boldsymbol{\mathcal{X}}^x_{i,k|k-1} = f\left(\boldsymbol{\mathcal{X}}^x_{i,k-1|k-1}, \boldsymbol{\mathcal{X}}^w_{i,k-1|k-1}\right) \quad (7)$$

The apriori state is then predicted as:

$$\hat{\mathbf{x}}^-_{k|k-1} = \sum_{i=0}^{N+1} W_i \boldsymbol{\mathcal{X}}^x_{i,k|k-1} \quad (8)$$

and the apriori covariance is estimated as:

$$\mathbf{P}^-_{k|k-1} = \sum_{i=0}^{N+1} W_i \left[\boldsymbol{\mathcal{X}}^x_{i,k|k-1} - \hat{\mathbf{x}}^-_{k|k-1}\right]\left[\boldsymbol{\mathcal{X}}^x_{i,k|k-1} - \hat{\mathbf{x}}^-_{k|k-1}\right]^T \quad (9)$$

B. Update

Similarly, the `update` step propagates the sigma points through the observation model:

$$\boldsymbol{\mathcal{Z}}_{i,k|k-1} = h\left(\boldsymbol{\mathcal{X}}^x_{i,k|k-1}, \boldsymbol{\mathcal{X}}^v_{i,k-1|k-1}\right) \quad (10)$$

to predict the observation of the state as:

$$\hat{\mathbf{z}}_{k|k-1} = \sum_{i=0}^{N+1} W_i \boldsymbol{\mathcal{Z}}_{i,k|k-1} \quad (11)$$

The observation covariance is then calculated:

$$\mathbf{S}_{k|k-1} = \sum_{i=0}^{N+1} W_i \left[\boldsymbol{\mathcal{Z}}_{i,k|k-1} - \hat{\mathbf{z}}_{k|k-1}\right]\left[\boldsymbol{\mathcal{Z}}_{i,k|k-1} - \hat{\mathbf{z}}_{k|k-1}\right]^T \quad (12)$$

then the cross-covariance:

$$\mathbf{P}_{xz,k|k-1} = \sum_{i=0}^{N+1} W_i \left[\boldsymbol{\mathcal{X}}^x_{i,k|k-1} - \hat{\mathbf{x}}^-_{k|k-1}\right]\left[\boldsymbol{\mathcal{Z}}_{i,k|k-1} - \hat{\mathbf{z}}_{k|k-1}\right]^T \quad (13)$$

then the observation residual:

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \hat{\mathbf{z}}_{k|k-1} \quad (14)$$

and the Kalman gain:

$$\mathbf{K} = \mathbf{P}_{xz,k|k-1}\mathbf{S}^{-1}_{k|k-1} \quad (15)$$

Finally the standard Kalman equations [17] are used to estimate the current system state and covariance:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}^-_{k|k-1} + \mathbf{K}\tilde{\mathbf{y}}_k \quad (16)$$

$$\mathbf{P}_k = \mathbf{P}^-_{k|k-1} - \mathbf{K}\mathbf{S}_{k|k-1}\mathbf{K}^T \quad (17)$$

III. System Model

The satellite system is modelled in a similar fashion to previous work [14]; the attitude of the satellite is represented by the unit quaternion $q = [\mathbf{q},\ q_0]^T$ where $\mathbf{q} = [q_1,\ q_2,\ q_3]^T$ and which satisfies $q_1^2 + q_2^2 + q_3^2 + q_0^2 = 1$.

The kinematic equations for the satellite in terms of quaternions are then given by:

$$\dot{\mathbf{q}} = \frac{1}{2}\left(q_0 I_{3\times 3} + \mathbf{q}^\times\right)\boldsymbol{\omega} \quad (18)$$

$$\dot{q_0} = -\frac{1}{2}\mathbf{q}^T\boldsymbol{\omega} \quad (19)$$

where $\mathbf{q}^\times$ is the skew-symmetric matrix of $\mathbf{q}$ given by:

$$\mathbf{q}^\times = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \quad (20)$$

A. Sensor Model

For simplicity, only a basic sensor set common on nanosatellites is considered here: a three-axis MEMS IMU (Invensense ITG-3200 gyroscope [18], ADXL345 accelerometer [19], HMC5843 magnetometer [20]).

To model the sensors, first the standard gyroscopic model is used:

$$\mathbf{z}_g = \boldsymbol{\omega} + \boldsymbol{\beta} + \boldsymbol{\eta}_g \quad (21)$$

$$\dot{\boldsymbol{\beta}} = \boldsymbol{\eta}_d \quad (22)$$

where $\boldsymbol{\omega}$ is the true angular velocity, $\boldsymbol{\beta}$ is the gyroscopic bias, $\dot{\boldsymbol{\beta}}$ is the bias drift and $\boldsymbol{\eta}_g, \boldsymbol{\eta}_d$ are zero-mean Gaussian noise terms. Similarly, the accelerometer and magnetometer are modelled as:

$$\mathbf{z}_a = \mathbf{a} + \boldsymbol{\eta}_a \quad (23)$$

$$\mathbf{z}_m = \mathbf{m} + \boldsymbol{\eta}_m \quad (24)$$

where $\mathbf{a}$ is the true, total acceleration vector, $\mathbf{m}$ is the true magnetometer vector and $\boldsymbol{\eta}_a, \boldsymbol{\eta}_m$ are zero-mean Gaussian measurement noise terms.

B. Predict Model

The gyroscopic data is integrated to estimate the satellite's attitude at each time step, so the predict model, f, is then:

$$\dot{\boldsymbol{\mathcal{X}}} = f(\boldsymbol{\mathcal{X}}^a) = \begin{bmatrix} \frac{1}{2}\left(q_0 I_{3\times 3} + \mathbf{q}^{\times}\right)\boldsymbol{\omega} \\ -\frac{1}{2}\mathbf{q}^T\boldsymbol{\omega} \\ \mathbf{0}_{3\times 1} \end{bmatrix} + \mathbf{w}_k \qquad (25)$$

where $\mathbf{w}_k = [\boldsymbol{\eta}_q,\ \dot{\boldsymbol{\beta}}]^T$ is the process noise and $\boldsymbol{\eta}_q$ is assumed to be a zero-mean Gaussian.

C. Update Model

The accelerometer and magnetometer data is used to correct for the gyroscopic bias and noise, so the observation model, h, is:

$$h(\boldsymbol{\mathcal{X}}^a) = \begin{bmatrix} A_a\left(q\right)\mathbf{b}_a \\ A_m\left(q\right)\mathbf{b}_m \end{bmatrix} + \mathbf{v}_k \qquad (26)$$

where A_a, A_m are the rotation matrices between the body frame and local frame as a function of the state quarternions, $\mathbf{b}_a, \mathbf{b}_m$ are the respective body vectors and $\mathbf{v}_k = [\boldsymbol{\eta}_a,\ \boldsymbol{\eta}_m]$ is the measurement noise.

D. Simulation Model

Finally, the augmented state vector is given by:

$$\mathbf{x}^a = [\mathbf{q},\ q_0,\ \boldsymbol{\beta},\ \boldsymbol{\eta}_q,\ \dot{\boldsymbol{\beta}},\ \boldsymbol{\eta}_a,\ \boldsymbol{\eta}_m]^T \qquad (27)$$

The quaternion process noise term was modelled with covariance $\boldsymbol{\eta}_q = 10^{-6}$, while the gyroscopic bias drift was modelled with an uncharacteristically large covariance of $\boldsymbol{\eta}_d = 1^{\circ}/s^2$ to emphasise the algorithms ability to correct for the bias. The measurement noise terms were modelled with covariances: $\boldsymbol{\eta}_g = 10^{-1}\ ^{\circ}/s,\ \boldsymbol{\eta}_a = 10^{-2}g,\ \boldsymbol{\eta}_m = 10^{-3}\ gauss$ using values from their respective datasheets.

The satellite is modelled as an ideal 1U CubeSat and its motion as oscillating about all three axes with amplitude $\pm 5^{\circ}$ and frequency $\omega_n = 0.1$ Hz; as with the gyroscopic drift bias, the satellite parameters are deliberately set to be harsh to better demonstrate the effectiveness of the filter. The sampling frequency of the sensors was set at 1 kHz; an example of the simulated sensor data can be seen in Fig. 1.

IV. Design

A. Overall Architecture

Reviewing the UKF algorithm, only the `predict` and `update` models ((7) and (10)) are application specific; the remainder of the algorithm is essentially a series of matrix manipulations. The two models are then implemented in software on a processor (either soft-core or hard-core) while the rest is implemented as parameterisable hardware and a series of automatically generated headers, for both the software and HDL, are used to pass the correct algorithm parameters (e.g. number of state variables, number of augmented state variables, sigma point weights etc.) to both parts.

For the greatest portability, the interface between the two parts is made simple: a memory buffer for data and a set of 16 control lines are used (see Fig. 2). The memory buffer has a bus interface on one end to communicate with the processor and a simple generic RAM interface on the other to communicate with the IP core. This means the IP core can easily be attached to different processors with only minor modifications (to the bus interface) required. The 16 control lines are simply digital signals that can be attached to either a GPIO module or an interrupt controller.

These control lines allow the processor to initiate the IP core calculations, to know where the IP core calculations are up to and when the processor needs to deliver model data. The IP core remains in an idle state until given a signal by the processor to perform an initialisation, the `predict` step or the `update` step and enters a waiting state when it requires the model data (see Fig. 3); these steps can be performed independently as required, or as the available sensor data will allow.

The IP core is implemented using single precision (IEEE-754) arithmetic but is easily extensible to double precision. As mentioned earlier, for this design, timing performance is sacrificed for area efficiency and so the basic arithmetic is performed in a largely sequential fashion (i.e. *not* unrolling the loops). Although this method is slower and increases the overall latency of the IP core, it does reduce the number of DSPs used, which may aid later integration into a larger SoC design.

B. Matrix Right 'Divide'

There are two operations in the UKF that have the most potential to limit hardware performance: the matrix 'square root' in the sigma points calculation (see (6)) and the inverse

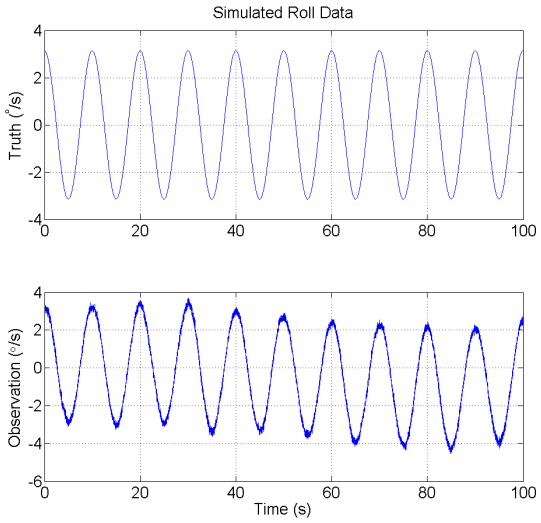

Fig. 1. Sample of the simulated gyroscopic sensor data. The top figure shows the simulated truth motion and the bottom figure shows the simulated measurements from the gyroscope including the measurement noise and gyroscopic bias

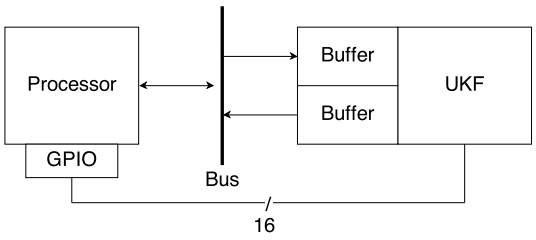

Fig. 2. Block diagram of the top level FPGA implementation

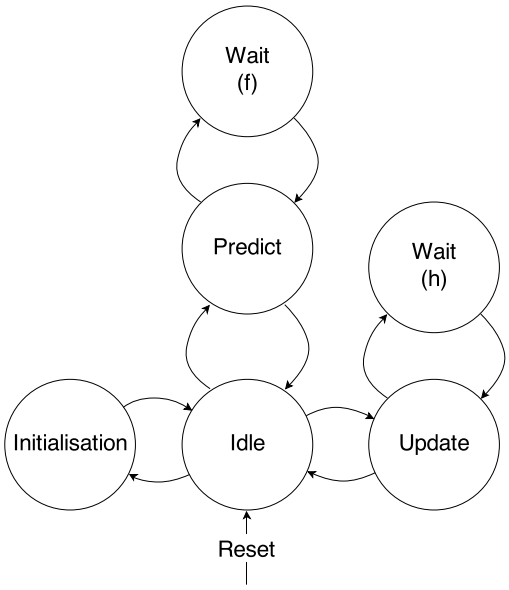

Fig. 3. Top level state diagram. Transition conditions are omitted for clarity, but are all tied to the control signals. The two wait states are required to give the processor time to handle the transformation of the sigma points through the system models. One occurs during the `predict` step for the process model, f (see (25)), and the other during the `update` step for the observation model, h (see (26)).

matrix in the `update` equations (see (15)). The matrix 'square root' is unavoidable in this form of the UKF but directly inverting a matrix is complicated and can be replaced by the matrix right 'divide'. For positive definite matrices, which the covariance matrices are, the matrix 'divide' simply involves a Cholesky decomposition followed by forward elimination and back substitution. This module still has the potential to be a major performance bottleneck unless a completely parallel implementation of the decomposition is used, which would incur a severe area cost. To avoid this, while maintaining reasonable performance, instead of the original decomposition:

$$A = L_1 L_1^T \tag{28}$$

where L_1 is lower triangular, an alternative version of the decomposition is used:

$$A = L_2 D L_2^T \tag{29}$$

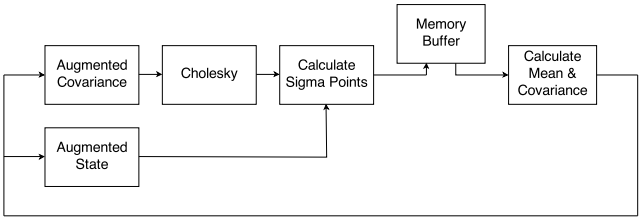

Fig. 4. Block diagram of the `predict` step

where L_2 is lower triangular and its diagonal terms are unit elements, D is diagonal and the two versions are related by $L_1 = L_2\sqrt{D}$. The second form of the decomposition is favoured in part because it avoids needing to calculate a (scalar) square root which features in the Cholesky algorithm (see [16]) as well as moves a division operation out of the critical path. Unfortunately, the calculation to generate the sigma points before the `predict` step still requires the original product, L_1 (see (6)). The recombination process to recover L_1 from L_2 and D – including the (scalar) square root – however, can be fully pipelined after the decomposition, so the performance penalty for having to perform an additional calculation is not too great. More importantly, using this version of the decomposition allows the decomposition as well as the back substitution and forward elimination to be treated as solving a series of triangular linear equations [21] and the hardware reused for each of these calculations.

C. Predict Step

The architecture of the `predict` step can be seen in Fig. 4. After the sigma points are calculated, they are written into the memory buffer to the processor and the system enters a wait (see Fig. 3) state. Once the sigma points have been transformed by the `predict` model on the processor, the apriori state and covariance are calculated and result written back to the memory buffer as well the IP core's internal augmented state memory.

D. Update Step

The architecture for the `update` step can be seen in Fig. 5. The `update` step starts the IP core in a wait state (see Fig. 3) while the processor propagates the sigma points through the `update` model and places the transformed sigma points, as well as the current sensor observation, in the memory buffer. As mentioned earlier, the Cholesky decomposition hardware from the `predict` step is reused for the matrix right 'divide'. After the current state estimates are calculated they are, like the `predict` step, written back to the processor memory buffer as well as the internal IP core memories. In this way, the `predict` and `update` steps can be performed asynchronously depending on the available sensor data and the processor is always up-to-date with the latest state estimate.

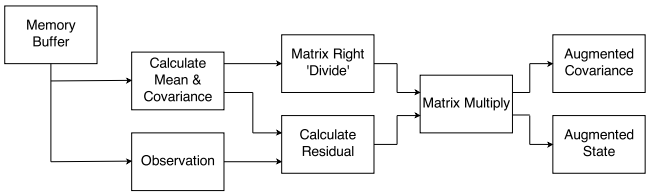

Fig. 5. Block diagram of the `update` step

V. Results

A. Zynq-7000

For establishing proof-of-concept and initial testing purposes only, the Zedboard development board featuring a Xilinx Zynq-7000 series XC7Z020 [22] was used. The relevant features of the target device are:

- Dual ARM Cortex-A9 hard-core processor system (PS) @ 667 MHz
- The equivalent of an Artix-7 device in programmable logic (PL) @ 100 MHz
- AXI4 PS-PL interface

One of the processors was used to implement the software part and the hardware IP core attached through one general purpose AXI4 port and 16 GPIO lines; the memory buffer utilises the on-chip block RAM which is 32 bits wide and 512 words deep in either direction. For this preliminary testing, the software was written as a bare-metal application (i.e. no OS) and the hardware was instantiated with $N = 20$ for the augmented state vector length (cf. (27)). For comparison purposes, the UKF was also implemented purely in software on one processor, utilising the GNU Scientific Library (GSL) for vector and matrix manipulation.

The software parts, both in the co-design and the pure software design, were implemented in C. The hardware part was developed in Verilog and synthesised with Xilinx's Vivado 2013.4; basic arithmetic (i.e. add, multiply etc.) was implemented using floating point IP core from Xilinx's IP catalogue. Matlab scripts were used to generate the simulated sensor data as well as generate headers containing the relevant parameters for both the software and hardware parts, ensuring coherence.

B. Simulation

Both implementations used the same dataset and both produced (within working precision) the simulation results in Fig. 6 and 7; these figures show the absolute attitude error, i.e. the difference between the UKF estimated attitude and the simulated 'truth', over the simulation duration of 100 seconds. Fig. 6 shows the error over the full duration of the simulation and demonstrates that the filter was able to correct for the gyroscopic bias and bias drift (cf. Fig. 1) for the full duration. Fig. 7 shows a small section at the start of the simulation demonstrating that although the UKF was initialised with a noisy measurement, the filter was able to quickly correct for the errors and maintain stable performance.

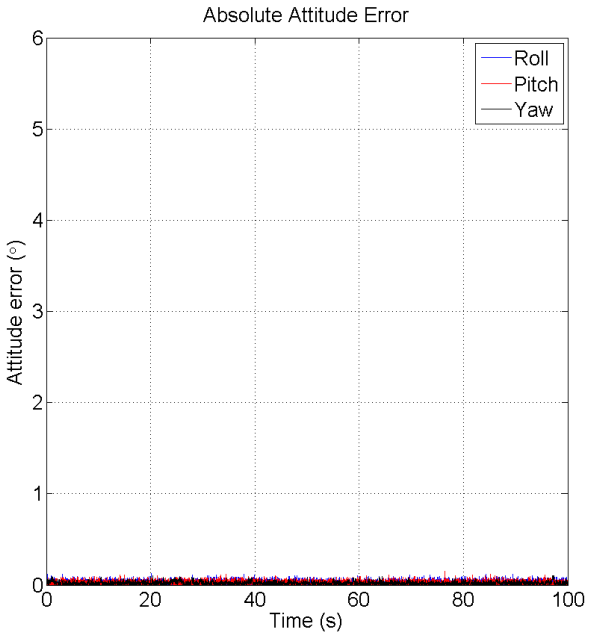

Fig. 6. Absolute attitude error for the full simulation

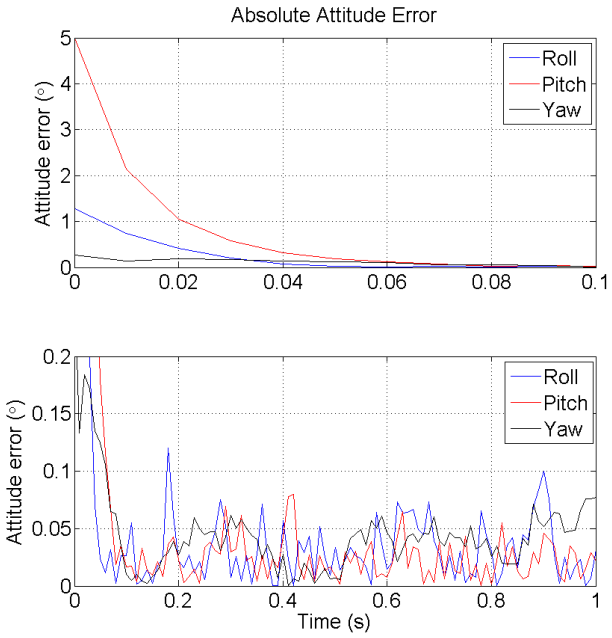

Fig. 7. Absolute attitude error for the very beginning of the simulation. The top figure shows the first tenth of a second of the simulation highlighting early convergence of the filter to the truth from an initial noisy estimate. The bottom figure shows the first second of the simulation highlighting the ability of the filter to maintain its accuracy ($< 0.1°$ error) after convergence

C. Synthesis

Synthesis results for the proposed design can be seen in Table I. These results do not include the processor, as mentioned earlier the processor is a hard-core, but do include

TABLE I
SYNTHESIS RESULTS

Resrouce	Utilisation	Utilisation %
FF	8238	8
LUT	6283	12
BRAM	16	11
DSP48	35	16

TABLE II
TIMING PERFORMANCE

	SW (μs)	SW/HW (μs)
Predict	639	337
Update	121	158
Total	760	495

TABLE III
TIMING BREAKDOWN OF PROPOSED DESIGN

	Time (μs)	% Total
`Predict` step:		
Cholesky	57	12
Calculate sigma points	93	19
`Predict` model	161	33
Apriori estimates	20	4
`Update` step:		
`Update` model	102	21
`Update` covariances	22	4
Matrix right 'divide'	26	5
State estimates	5	1
Total	495	100

TABLE IV
POWER CONSUMPTION BREAKDOWN

Resource	Usage (mW)	% Total
Clocks	39	33
Signals	27	23
Logic	23	19
RAM	21	18
DSP	7	6
Total	119	100

the logic necessary for the AXI4 interface ports.

The resources required by the proposed design are relatively low and it does not require a proportionally large amount of any one resource. This will allow easier integration into a full SoC, particularly if partially reconfigurable regions are used, as requiring too much of any one resource type can lead to placement issues.

D. Timing

The proposed design was able to be synthesised with a target frequency of 100 MHz; the design at this stage is largely unoptimised and higher target frequencies may be possible however. Timing was measured using two approaches: in the software using the ARMv7 Performance Monitoring Unit (PMU) which counts processor clock cycles and in hardware using an instantiated debug core to count the PL clock cycles; results from both approaches were cross-checked with each other as well as simulation timing results to ensure their accuracy. The timing results were calculated for a single full interation, that is a single `predict` step immediately followed by a single `update` step. The overall timing for the UKF can be seen in Table II while a breakdown of the time spent in different modules for the proposed design can be seen in Table III.

The proposed design represents a $1.5\times$ speed up over the pure software implementation and can be run at a frequency of $\approx$ 2 kHz which is more than adequate for the sampling frequency assumed in the simulation.

The design spends the majority of the time (total of 54%) propagating the sigma points through the two system models. Much of this time is actually spent transferring the data from the IP core memory buffer into memory attached to the processor that the software can access (i.e. from the PL side to the PS side) – this is the reason why the `update` step in the proposed (SW/HW) design actually takes longer than the purely software (SW) implementation (see Table II). The Zynq-7000 PS features memory directly connected to the processor (see [22]) but access to the PL (and thus the IP core memory buffer) happens over the AXI4 interconnect; this means the SW design is able to access data much faster than the SW/HW design during the two sigma point propagation calculations. For processors that have *all* memory and peripherals attached over an interconnect, this advantage will be reduced; the communication latency between hardware and software could still have a major impact on overall performance however.

Nearly three quarters of the time used by the hardware is spent in the Cholesky decomposition and calculating the sigma points. These two modules involve large matrix operations which scale poorly as the number of state variables increase and are first target for parallelisation; the matrix 'right' divide similarly scales as the number of observation variables making the triangular linear equation solver the higher priority for any optimisations. The processor will likely still be the performance bottleneck for the design however, especially if less powerful soft-core processors are used instead.

E. Power

The Xilinx Power Analyzer was used to generate a power consumption breakdown for the hardware IP core (i.e. excluding the processor) of the proposed design; the breakdown can be seen in Table IV. The power consumption is reasonably low likely due to the area efficiency design goals and heavy utilisation of the FPGA clock enable resources to disable modules that are not currently in use. The relatively low power consumption by the DSPs suggest they could be under-utilised.

VI. Conclusion

In this paper a hardware/software co-design of the Unscented Kalman Filter implemented on a FPGA device was presented. The design was developed to leverage the benefits of both software (portability) and hardware (performance) to promote reusability. Area usage is low which either allows for easier integration into a SoC or leaves room for further performance optimisations. The proposed design outperforms a purely software implementation on a powerful ARM processor, though the processor itself remains as the performance bottleneck. Power consumption for the hardware also remains low meaning the processor used will still be the main consideration for power constraints.

Future work for this design include:

- Optimise hardware for further increases in timing performance
- Testing with the actual IMU and CubeSat structure
- Incorporate a more realistic set of sensors
- Extend to full closed-loop system

References

[1] J. Bouwmeester and J. Guo, "Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology," *Acta Astronautica*, vol. 67, no. 78, pp. 854 – 862, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576510001955

[2] T. Kuwahara, F. Böhringer, A. Falke, J. Eickhoff, F. Huber, and H. Röser, "FPGA-based operational concept and payload data processing for the Flying Laptop satellite," *Acta Astronautica*, vol. 65, no. 11, pp. 1616–1627, 2009.

[3] S. Julier and J. Uhlmann, "A new extension of the kalman filter to nonlinear systems," in *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, vol. 3, 1997, p. 26.

[4] E. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. IEEE, 2000, pp. 153–158.

[5] K. Xiong, C. W. Chan, and H. Zhang, "Detection of satellite attitude sensor faults using the UKF," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 43, no. 2, pp. 480–491, 2007.

[6] J. Zhou, Y. Yang, J. Zhang, E. Edwan, O. Loffeld, and S. Knedlik, "Tightly-coupled INS/GPS using Quaternion-based Unscented Kalman filter," in *AIAA Guidance, Navigation and Control Conference*, August 2011.

[7] S. Jafarzadeh, C. Lascu, and M. Fadali, "State Estimation of Induction Motor Drives Using the Unscented Kalman Filter," *Industrial Electronics, IEEE Transactions on*, vol. 59, no. 11, pp. 4207–4216, 2012.

[8] J. Crassidis, F. Markley, and Y. Cheng, "Survey of nonlinear attitude estimation methods," *Journal of Guidance Control and Dynamics*, vol. 30, no. 1, p. 12, 2007.

[9] R. Kandepu, B. Foss, and L. Imsland, "Applying the unscented Kalman filter for nonlinear state estimation," *Journal of Process Control*, vol. 18, no. 78, pp. 753 – 768, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0959152407001655

[10] S. Holmes, G. Klein, and D. Murray, "An $O(N^2)$ Square Root Unscented Kalman Filter for Visual Simultaneous Localization and Mapping," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 7, pp. 1251 –1263, july 2009.

[11] A. Quinchia and C. Ferrer, "A low-cost GPS&INS integrated system based on a FPGA platform," in *Localization and GNSS (ICL-GNSS), 2011 International Conference on*, 2011, pp. 152–157.

[12] V. Bonato, R. Peron, D. Wolf, J. de Holanda, E. Marques, and J. Cardoso, "An FPGA Implementation for a Kalman Filter with Application to Mobile Robotics," in *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*, July 2007, pp. 148–155.

[13] S. Cruz, D. Munoz, M. Conde, C. Llanos, and G. Borges, "FPGA implementation of a sequential Extended Kalman Filter algorithm applied to mobile robotics localization problem," in *Circuits and Systems (LASCAS), 2013 IEEE Fourth Latin American Symposium on*, 2013, pp. 1–4.

[14] J. Soh and X. Wu, "A FPGA-based approach to attitude determination for nanosatellites," in *Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on*, July 2012, pp. 1700–1704.

[15] S. Julier, "The spherical simplex unscented transformation," in *American Control Conference, 2003. Proceedings of the*, vol. 3, June 2003, pp. 2430–2434 vol.3.

[16] G. H. Golub and C. F. Van Loan, *Matrix computations*, 3rd ed. Baltimore: Johns Hopkins University Press, 1996.

[17] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," University of North Carolina, Chapel Hill, NC, USA, Tech. Rep., 1995.

[18] *ITG-3200 Product Specification*, Inversense, February 2011, Rev. 1.7. [Online]. Available: http://www.invensense.com/mems/gyro/documents/PS-ITG-3200A.pdf

[19] *Digital Accelerometer (ADXL345)*, Analog Devices, February 2013, Rev. D. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf

[20] *3-Axis Digital Compass IC HMC5843*, Honeywell, June 2010. [Online]. Available: http://www.honeywell.com/sites/servlet/com.merx.npoint.servlets.DocumentServlet?docid=DA9ACFE3C-F7C0-9998-6085-D9D84941499D

[21] D. Yang, G. Peterson, H. Li, and J. Sun, "An FPGA Implementation for Solving Least Square Problem," in *Field Programmable Custom Computing Machines, 2009. FCCM '09. 17th IEEE Symposium on*, April 2009, pp. 303 – 306.

[22] *Zynq-7000 All Programmable SoC Technical Reference Manual*, Xilinx, September 2013, UG585 (v1.6.1).