International Journal of Innovative Computing, Information and Control
Volume **8**, Number **3(A)**, March **2012**

ICIC International ©2012 ISSN 1349-4198
pp. **1987–2000**

A NOVEL APPROACH OF AN FPGA DESIGN TO IMPROVE MONOCULAR SLAM FEATURE STATE COVARIANCE MATRIX COMPUTATION

Mohd. Yamani Idna Idris[1], Hamzah Arof[2], Noorzaily Mohamed Noor[1]
Emran Mohd. Tamil[1], Zaidi Razak[1] and Ainuddin Wahid[1]

[1]Faculty of Computer Science and Information Technology
[2]Faculty of Engineering (Electrical)
University of Malaya
Kuala Lumpur 50603, Malaysia
{ yamani; ahamzah; zaily; emran; zaidi; ainuddin }@um.edu.my

Received December 2010; revised April 2011

Abstract. *Monocular SLAM is a study which concentrates on deriving the position and motion estimation information from tracked features using a single camera. In this paper, a novel approach to improve the computation speed of a Monocular SLAM is proposed. The research concentrates on the feature initialization process which takes place before the standard Extended Kalman Filter (EKF). In order to find the most time consuming process at the initialization stage, a software profiling tool is used. From the result, the section of a program which demands high processing computation is identified. Following that, a specialized design is proposed to improve the computation speed. An FPGA approach is chosen with the intention to offload software processing to a dedicated hardware for overall performance acceleration. In order to accomplish this goal, the section demanding high processing computation is carefully studied. From the studies, it is found that the original approach can be improved by reducing the multiplication process and incorporating parallel processing capability of an FPGA. At the end of the paper, the comparison results of the software and hardware processing are presented.*

Keywords: Simultaneous localization and mapping (SLAM), Parallel design, Matrix multiplication landmark initialization, Inverse depth parameterization

1. **Introduction.** Simultaneous Localization and Mapping (SLAM) is a process where a mobile robot can build a map of the surrounding environment and concurrently use this map to compute its own location. Due to the promising advantages of vision sensor such as being compact, accurate, noninvasive, cheap, well understood and ubiquitous [1,2], various vision SLAM algorithms [3] have been studied. Monocular SLAM is a Bearing-Only SLAM which utilizes a single vision sensor to measure the bearing of image features. From the measured bearings, the depth information is estimated using feature parallax. In this context, parallax is a measured angle of an object or captured rays viewed from two different lines of sight. One of the frequent problems occurring in the Bearing Only SLAM is the robustness of the new feature initialization process in determining the depth information [4]. This initialization problem is commonly addressed by using Delayed and Undelayed approaches. Both approaches have their own advantages and disadvantages. The Delayed approach has an advantage whereby it is able to reject weak features. However, the Delayed approach has to wait until the sensor movement generates sufficient degrees of parallax. On the other hand, the Undelayed approach benefits from the information about the sensor orientation from the beginning. The downside of the Undelayed approach is that the depth estimation is modeled with huge

uncertainty. Moreover, its computational load grows exponentially with the number of landmarks [4].

2. Feature Initialization.

Feature Initialization is a process in SLAM (Figure 1) which reduces the landmark range uncertainty and helps to ensure that the location of landmarks can be inferred from a single bearing measurement. In 2000, Deans et al. [5] proposed a Delayed initialization approach, which makes use of bundle adjustment techniques to compute the optimal least square estimates. The bundle adjustment technique ensures that the Kalman filter is initialized with a good estimate of the robot state and landmark locations. Another delayed initialization approach is suggested by Bailey et al. [6] which modifies the constraint initialization procedure introduced by Williams et al. [7]. Bailey initialization is delayed in a sense that the system will wait until a sufficient base-line is reached before permitting Gaussian initialization, and becomes well-conditioned by using the Kullback distance. Other work related to the Delayed approach is presented in the paper Real-Time 3D SLAM with Wide-Angle Vision by Davison et al. [8]. This paper shows that real time SLAM with a single camera is feasible using an EKF framework. They also employed a Particle Filter to estimate the features depth, which is uncorrelated with the rest of the map. The distribution of possible depths is updated based on each new observation until the variance range is small enough to consider a Gaussian estimation. The problem with such an approach is that the initial particles distribution has to cover the entire possible depth of values for a landmark. This would be complicated when there are many detected features or when the features are far. A particle filter is also utilized by Minoru et al. [9] and Masahiro et al. [10].

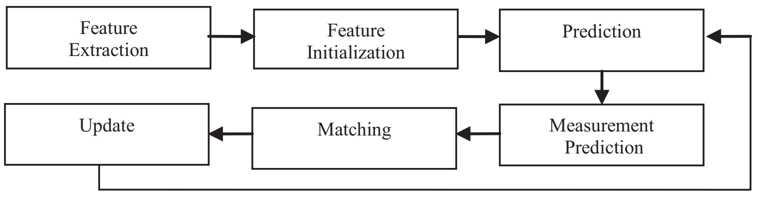


Figure 1. SLAM process

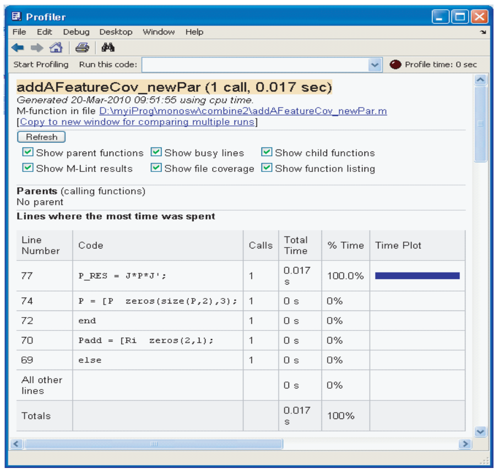
Another initialization method which is commonly used by researchers is called the Undelayed approach. One of the earliest Undelayed approaches was presented by Kwok et al. [11]. They propose that the initial representation is in the form of multiple hypotheses distributed along the direction of the bearing measurement. For the subsequent measurements, Sequential Probability Ratio Test (SPRT) based on likelihood has been utilized to validate the hypotheses. In Sola et al. [12], two drawbacks of the Delayed approach are addressed. The first drawback is the need of criteria to decide whether or not the baseline is sufficient to permit Gaussian initialization. Another drawback is that the initialization has to wait until the criteria are validated. To ensure the delay is below reasonable limits, the camera motion cannot be close to the direction of the landmark. This would be unpractical for outdoor navigation since straight trajectories are common. In overcoming the problem, Sola et al. [12] utilizes Gaussian Sum approximation which permits Undelayed initialization. Since approximation representation has the tendency to be inconsistent and diverge, their work proposes the use of the Federated Information Sharing (FIS) method to minimize the risk. In a more recent study, Monteil [13] and

Civera et al. [14] put forward a concept called Inverse Depth Parameterization (IDP). The key idea of the concept is to produce a measurement equation with a high degree of linearity. Their paper discusses the drawback implied by the work done by Davison et al. [1,15] whereby it is only applicable for features that were close to the camera. The drawback is caused by the Euclidean XYZ feature parameterization for low parallax feature. This parameterization is not well represented by the Gaussian distribution implied in the EKF. On the other hand, inverse depth claimed to allow Gaussian distribution to cover uncertainty for both low and high parallax features. Despite this advantage, IDP suffers from computational issues since it requires six state vector parameters (Equation (2)) instead of three as in Euclidean XYZ coding (Equation (1)).

$$x_i = (X_i \; Y_i \; Z_i)^T \quad (1)$$
$$y_i = (x_i \; y_i \; z_i \; \theta_i \; \varnothing_i \; \rho_i)^T \quad (2)$$

where:
$x_i \; y_i \; z_i$ = *camera optical center*
θ_i = *azimuth*
$\varnothing_i$ = *elevation*
ρ_i = *inverse depth*

2.1. **Feature initialization software profiling result and discussion.** In order to improve the performance of the initialization process, a software profiling tool is utilized. Software profiling is a form of dynamic programming analysis tool to determine which section of a program demands a high processing computation. Since an inverse depth Parameterization technique by Civera et al. [14] is considered to be one of the highly rated methods in a SLAM system, it is chosen for analysis. The software profiling result is shown in Figure 2.



addAFeatureCov_newPar (1 call, 0.017 sec)
Generated 20-Mar-2010 09:51:55 using cpu time.
M-function in file D:\myiProg\monosw\combine2\addAFeatureCov_newPar.m
[Copy to new window for comparing multiple runs]

Parents (calling functions)
No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
77	`P_RES = J*P*J';`	1	0.017 s	100.0%	▬
74	`P = [P  zeros(size(P,2),3);`	1	0 s	0%	
72	`end`	1	0 s	0%	
70	`Padd = [Ri  zeros(2,1);`	1	0 s	0%	
69	`else`	1	0 s	0%	
All other lines			0 s	0%	
Totals			0.017 s	100%	

FIGURE 2. Inverse depth parameterization software profiling result

Based on the result acquired, it can be seen that the calculation to determine state covariance matrix ($PRES$) consumes the most processing time after several iterations. The reason for the slow processing time is caused by the increasing matrix size when $PRES = J * P * J'$ calculation is performed. Covariance matrix ($P_{k|k}$) computation starts with 13×13 diagonal matrixes. The thirteen values consist of three (x, y, z) camera optical center positions (r^{wc}), four (qR, qx, qy, qz) values of quaternion defining orientation (q^{wc}),

$$f_v = \begin{pmatrix} r_{k+1}^{WC} \\ q_{k+1}^{WC} \\ v_{k+1}^{W} \\ \omega_{k+1}^{C} \end{pmatrix} = \begin{pmatrix} r_k^{WC} + (v_k^W + V_k^W)\Delta t \\ q_k^{WC} \times q((\omega_k^C + \Omega^C)\Delta t) \\ v_k^W + V^W \\ \omega_k^C + \Omega^C \end{pmatrix} = x_v \tag{3}$$

three values of linear velocity relative to the world frame (v^w) and another three values of angular velocity relative to the camera frame (w^c) (Equation (3)).

When features are inserted, another six values (as in y_i in Equation (2)) will be added into the full state vector, $x = (x_v^T \; y_1^T \; y_2^T \; \ldots \; y_n^T)$. The insertion of more features will increase the size of the full state vector which leads to the increment of the matrix size. For instance, if one hundred features are tested, a covariance matrix with a size of 613×613 will be entailed in the computation, (i.e., 100 (from number of features) $\times$ 6 (from y_i; 6-D vector) + 13 (i.e., from fv) = 613). The increase in matrix size means a larger matrix multiplication has to be performed on the $PRES = J * P * J'$ which eventually increases the execution time.

3. Inverse Depth Parameterization and Matrix Multiplication.

Matrix multiplication has long been studied by many researchers to increase the computational speed of their proposed application. Based on several studies, there are two main approaches employed by researchers to improve matrix multiplication performance. The first approach is by improving the mathematical formula of the classical matrix multiplication, which leads to the introduction of several algorithms such as Winograd's and Strassen's algorithm. The second approach is by manipulating the hardware design architecture using technology such as multi-processor and FPGA. Among researchers utilizing the FPGA to improve matrix multiplication are Prasanna et al. [16], Mencer et al. [17], Amira et al. [18], Jang et al. [19], Bravo et al. [20], Zhuo et al. [21,22] and Idris et al. [23,24].

Both approaches have shown significant improvements of the overall matrix multiplication process. However, the approaches are not specialized to cater for Inverse Depth Parameterization matrix multiplication. For that reason, a study to identify how the original state covariance matrix formula effects the overall computation is carried out.

3.1. Conventional state covariance calculation using classical matrix multiplication approach.

From the pseudocode of state covariance (P) calculation as presented in Figure 3, the matrix size can be seen to increase every time a new feature is inserted. For each feature, dydxv (i.e., $dy/drwc$, $dy/dqwc$, 0, …, 0) and dydhd (i.e., dy/dh, $dy/d\rho$) value of the Jacobian (J) are calculated. Each dydxv has a matrix size of 6×13 and dydhd has a matrix size of 6×3. The matrix size increment as shown in Table 1 and Figure 4 illustrates how many multiplications are needed when classical matrix multiplication approach is used. For three iterations or three features, it can be seen that $4864 + 5776 + 12100 + 13750 + 24304 + 26908 = 87702$ multiplication operations are needed.

3.2. Proposed state covariance matrix computation.

Based on the pseudocode in Figure 3, it can be seen that several problems could affect the processing time. The insertion of zero and identity matrix into the matrix J and P to suit the covariance matrix formula increases the number of operations in the program. The problem was addressed by Zienkiewicz et al. [25] by proposing a typical approach which avoids multiplication with zero. The abstaining from immaterial multiplication has resulted in reduced execution time. Another problem perceived is the self matrix multiplication in $P = JPJ'$ which will cause matrix size to expand (as illustrated in Figure 4) every time the program iterates.

```
dydxv=[dydxv1 dydxv2 dydxv3 ……  dydxv_k]; % each dyddxv_k has constant 6x13 matrix size
dydhd=[dydhd1 dydhd2 dydhd3 ……  dydhd_k]; % each dyddxv_k has constant 6x3 matrix size

for k=1:numfeat % number of features/iterations
    J = [eye(size(P,2))  zeros(size(P,1),3); % Initial P is 13x13 & will increase following iterations
        dydxv(1:6,k*13-12:k*13) zeros(6,(size(P,2)-13))dydhd(1:6,k*3-2:k*3)];

    P = [P  zeros(size(P,2),3);
        zeros(3,size(P,2)) Padd]; % Padd has constant 3x3 matrix size

    P = J*P*J';
end
```

FIGURE 3. Pseudocode for calculating state covariance (P)

TABLE 1. Matrix size increment

Matrix	1st Feature/Iteration Matrix Size	2nd Feature/Iteration Matrix Size	3rd Feature/Iteration Matrix Size	nth Iteration Matrix Size
J	19x16	25x22	31x28	19+(6)(n-1)x16+(6)(n-1)
P	16x16	22x22	28x28	16+(6)(n-1)x16+(6)(n-1)
J’	16x19	22x25	28x31	16+(6)(n-1)x19+(6)(n-1)

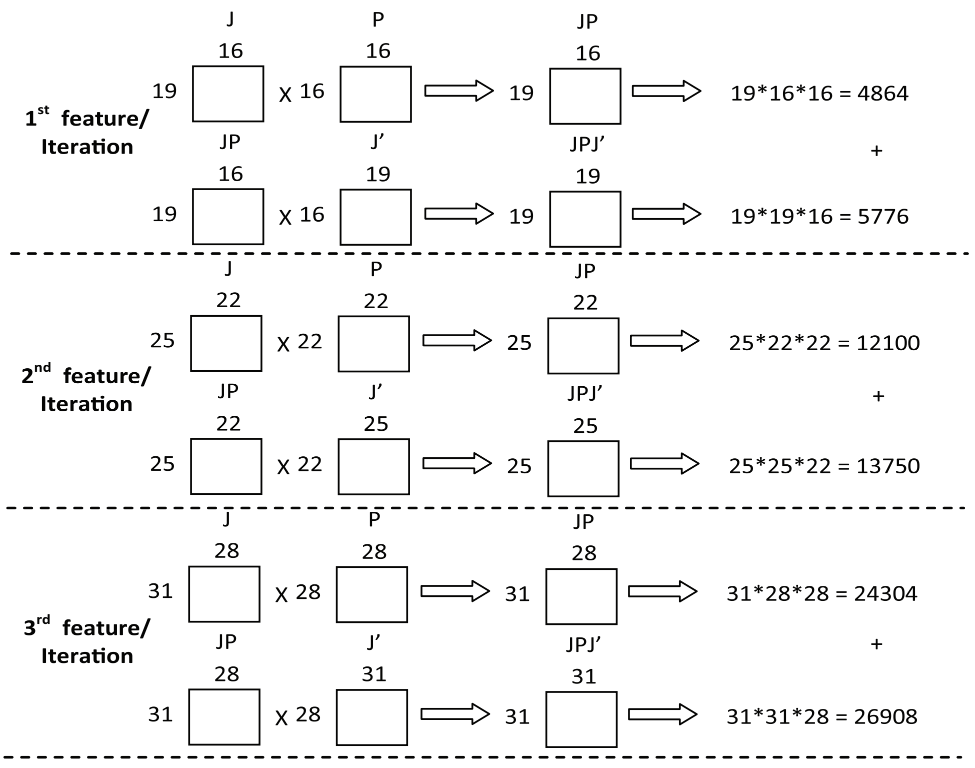

1st feature/ Iteration
J 16 19 X 16 P 16 ⟹ 19 JP 16 ⟹ 19*16*16 = 4864
+
JP 16 19 X 16 J’ 19 ⟹ 19 JPJ’ 19 ⟹ 19*19*16 = 5776
2nd feature/ Iteration
J 22 25 X 22 P 22 ⟹ 25 JP 22 ⟹ 25*22*22 = 12100
+
JP 22 25 X 22 J’ 25 ⟹ 25 JPJ’ 25 ⟹ 25*25*22 = 13750
3rd feature/ Iteration
J 28 31 X 28 P 28 ⟹ 31 JP 28 ⟹ 31*28*28 = 24304
+
JP 28 31 X 28 J’ 31 ⟹ 31 JPJ’ 31 ⟹ 31*31*28 = 26908

FIGURE 4. Number of multiplication operations needed to solve state covariance matrix using classical approach

This will lead to a huge exponential growth problem involving the number of multiplication operations. Furthermore, the function $P = JPJ'$ is a recursive function, which depends on solutions with smaller instances of the same problem. This will inevitably cause the process to come to a halt until the computation process for value P is completed.

In the proposed design, the aforementioned problems will be reduced by modifying the conventional state covariance matrix calculation. The pseudocode and diagram to illustrate the proposed design are depicted in Figures 5 and 6. The original state covariance matrix formula is altered to avoid multiplication with zero. The proposed design also avoids matrix size expansion caused by self matrix multiplication (i.e., $P = JPJ'$). Instead, a memory look up approach is utilized to reduce the multiplication operation. The number of multiplication involved in the proposed approach is illustrated in Figure 7. An example with reference to the number of multiplication operations needed for calculating three features is shown in Figure 8. As can be seen in the figure, the multiplication of dvP or Pdv' (i.e., number of multiplication $= 1014$) starting from column 2 and row 2 is not counted in the multiplication process. This is because the proposed design has stored the result of dvP and Pdv', which has been multiplied before into a memory. Therefore, there is no need to redo the same multiplication all over again and thus reducing the number of multiplication processes. By using the proposed design, only $(1014 + 1014 + 54 + 108) \times 3 + 468 \times 3 \times 3 = 10,782$ multiplication operations are needed to process three features.

```
dydxv=[dydxv1 dydxv2 dydxv3 ……  dydxv_k]; % each dyddxv_k has constant 6x13 matrix size
dydhd=[dydhd1 dydhd2 dydhd3 ……  dydhd_k]; % each dyddxv_k has constant 6x3 matrix size

for n=1:numfeat % number of features
    left=dydxv(1:6,n*13-12:n*13)*P(1:13,1:13); % dv_n X P
    right=P(1:13,1:13)*dydxv(1:6,n*13-12:n*13)'; % P X dv_n'
    rightdd=dydhd(1:6,n*3-2:n*3)*Padd*dydhd(1:6,n*3-2:n*3)'; %dd_n X Pad X dd_n'
                                                              % Pad has constant 3x3 matrix size

    for k=1:numfeat
        dvPdvp=left*dydxv(1:6,k*13-12:k*13)'; %dv_n X P X dv_k'
        % store values in memory
        dvPdvp1=[dvPdvp1 dvPdvp];
    end

    % select diagonal from memory
    leftdv=dvPdvp1(1:6,6*numfeat*n-(6*numfeat-1)+m1:(6*numfeat*n-(6*numfeat-1))+5+m1);
    m1=m1+6;
    bottomright=leftdv+rightdd; % (dv X P X dv') + (dd X Pad X dd')

    % store values in memory
     ………
     ………

end

%Rearange P2 addresses
………
………
```

Figure 5. Pseudocode of the altered state covariance matrix

To accelerate the execution time even further, an independent processing approach is proposed. Unlike the function $P = JPJ'$ in the conventional approach, the proposed design is not a recursive function. The process in each column and row of the proposed design in Figure 6 can be executed independently. This will allow for the features to be processed simultaneously if multiprocessing or parallel architecture is applied.

3.3. **Number of multiplication comparison.** In general, the number of multiplication operations for n features can be compared using the pseudocode in Figure 9. The graph which compares the number of multiplications is shown in Figure 10. From the figure, it can be seen that the number of multiplications of the conventional state covariance matrix using classic matrix multiplication increases exponentially. The proposed approach on the

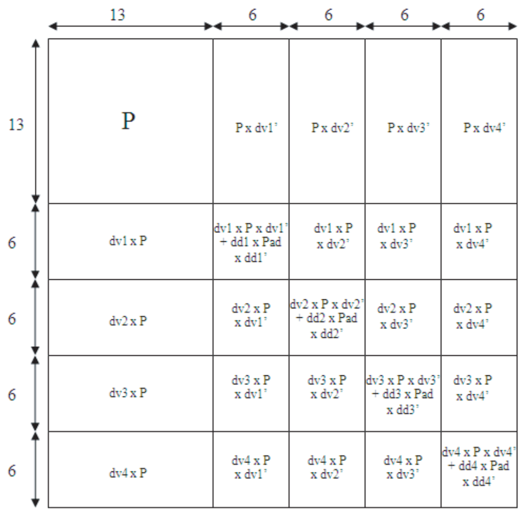


Figure 6. Altered state covariance matrix

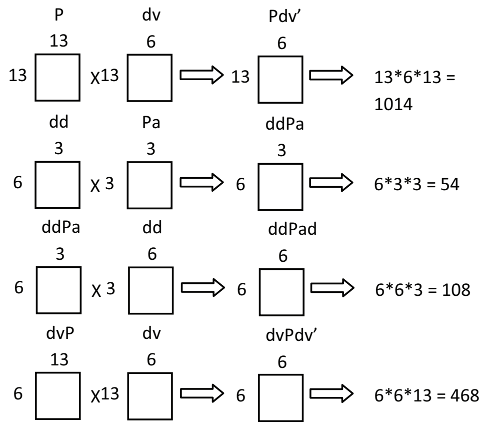


Figure 7. Number of multiplication operation needed to solve state covariance matrix using proposed approach

P	1014	1014	1014
1014	468+54+108	468	468
1014	468	468+54+108	468
1014	468	468	468+54+108

Figure 8. Number of multiplication operation for three features

```
% number of multiplication using normal classical matrix multiplication approach
for n=1:numfeat % number of features
    normal1=((19+6*(n-1))*(16+6*(n-1)) *(16+6*(n-1))) + ((19+6*(n-1))*(19+6*(n-1))*(16+6*(n-1)));
    normal2=normal2+normal1;
    featnum=[featnum n];
    normalmult=[normalmult normal2];
end

% number of multiplication using proposed approach
for k=1:numfeat % number of features
   proposedmult1= (6*13*13+13*6*13+6*3*3+6*6*3+(6*6*13)*k)*k;
   proposedmult=[proposedmult proposedmult1];
end
```

Figure 9. Number of multiplication comparison pseudocode

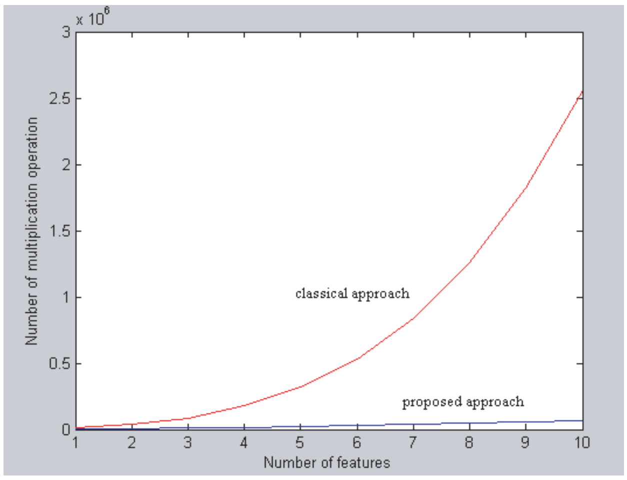


Figure 10. Number of multiplication comparison

other hand, does not show a large increment. This clearly shows that the proposed method is able to perform state covariance matrix calculations with much less multiplication operation. The key idea is to reuse the same calculated values stored in the memory. Furthermore, the suggested approach is designed to avoid multiplication with zero, which could be affecting the overall cycle time.

4. **FPGA Block Diagram.** The altered state covariance matrix in Figure 6 is designed as an FPGA block diagram as shown in Figure 11. There are several matrix multiplier modules in the implementation. The modules are designed using a specific matrix size to cater for the needs of each variable. This is indicated by the number besides the Mtrx_mult where the numbers are the multiplicand and multiplier size. For example, to multiply dvk (i.e., 6×13 matrix size) with P (i.e., 13×13 matrix size) the Mtrx_mult (6,13,13,13) is used. The matrix multiplier module consists of a Multiplier Accumulator (MAC), which computes the product of two numbers and adds them together to an accumulator. The MAC operation is controlled by an address controller to suits the matrix size. The address controller is divided into write and read mode to ensure the multiplication is processed correctly. Appendix 1 shows the values of the address which is used to access RAMs and their time steps. These values are generated by *write_addr* and *read_addr* of the *addr_ctrl* in Figure 12. Finally, the results of the altered state covariance matrix are stored in a register to be used by the next process in the feature initialization.

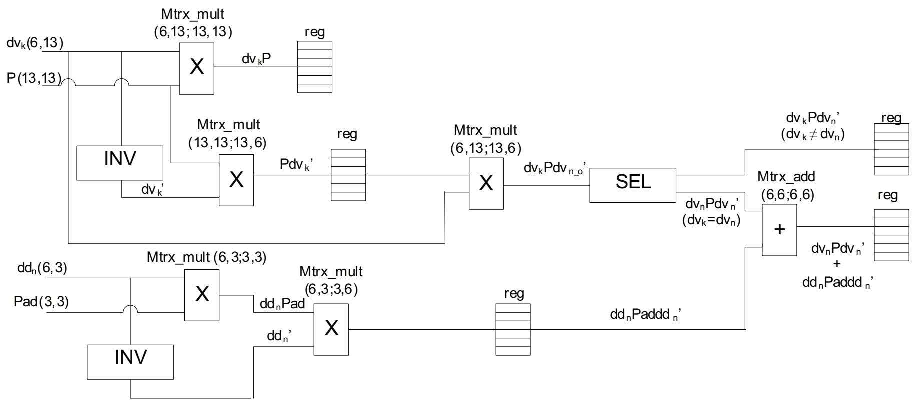


Figure 11. FPGA block diagram

4.1. **RTL schematic result after synthesis.** The design in Figure 11 is converted into a hardware block using Xilinx System Generator tools and hardware description language. Following that, Xilinx ISE tools are used for synthesis, simulation and implementation of the design into an FPGA. Figure 12 shows the result after synthesis.

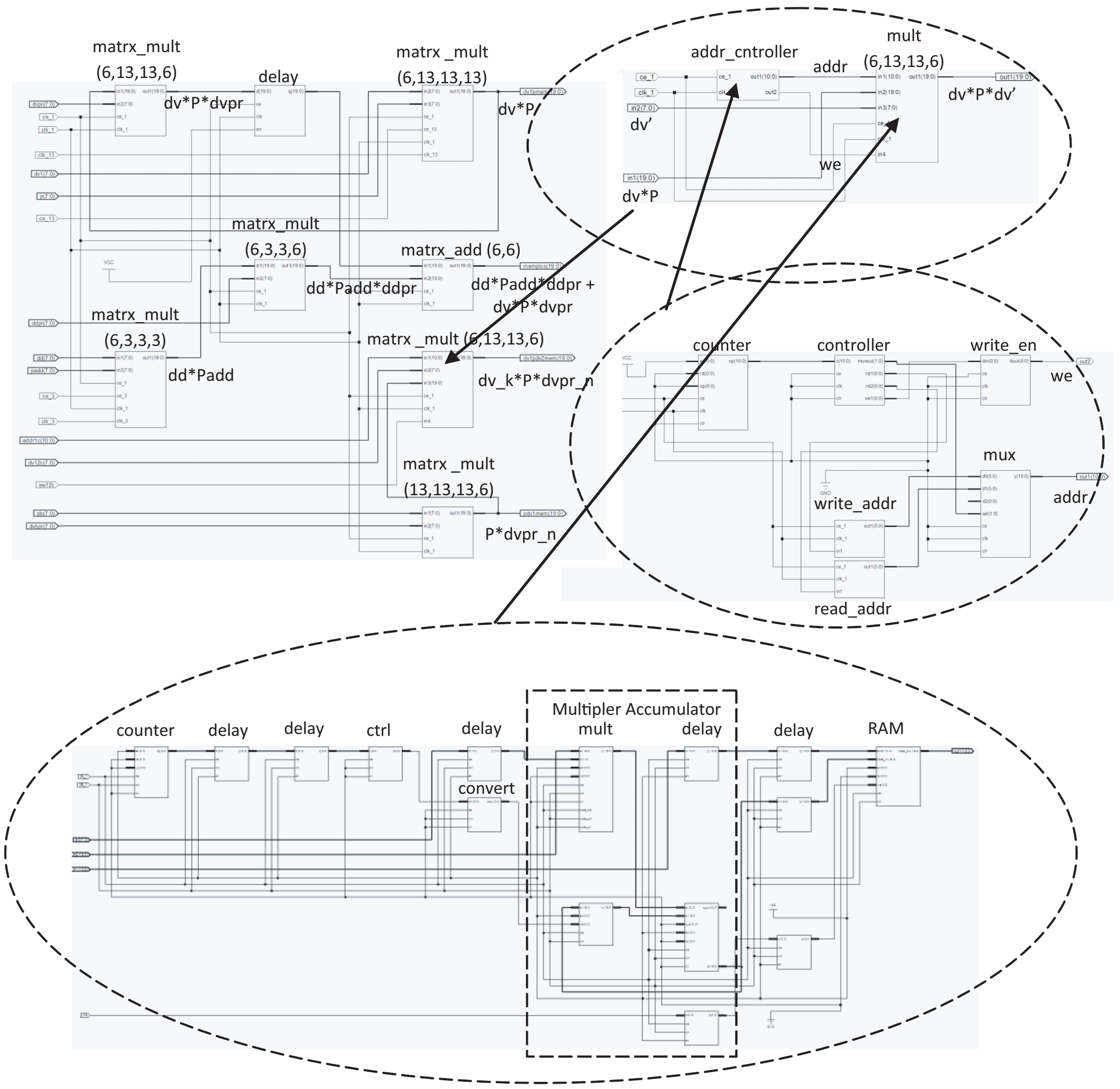


FIGURE 12. RTL schematic after synthesis

4.2. **Device utilization and timing summary.** A Spartan-3A DSP 34000A FPGA chip has been chosen for the synthesis. The device utilization and timing summary results using Xilinx ISE tools are presented in Figure 13. A total of 4496 slices are utilized in the design which consume 18% of the overall resources of the Spartan-3A DSP 3400A FPGA. Though it appears larger than a classical matrix multiplication implementation, the execution speed will be much faster. The maximum frequency that allows the paths to run is at 133.219 MHz. This signifies that the minimum clock period can reach 7.506 ns. Based on this result, a post route simulation is done to compare both software and hardware implementation results.

```
Device utilization summary:
---------------------------

Selected Device : 3sd3400afg676-5

 Number of Slices:                      4496  out of  23872    18%
 Number of Slice Flip Flops:            8292  out of  47744    17%
 Number of 4 input LUTs:                8519  out of  47744    17%
    Number used as logic:                772
    Number used as Shift registers:     7747
 Number of IOs:                          166
 Number of bonded IOBs:                  165  out of    469    35%
 Number of BRAMs:                         21  out of    126    16%
 Number of GCLKs:                          1  out of     24     4%
 Number of DSP48s:                         9  out of    126     7%

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 7.506ns (Maximum Frequency: 133.219MHz)
   Minimum input arrival time before clock: 1.216ns
   Maximum output required time after clock: 7.156ns
```

Figure 13. Device utilization and timing summary

5. **Software and Hardware Implementation Performance Comparison.** Figure 14 shows a post route simulation result using 8 ns or 125 MHz clock input of the proposed method. For four features, the feature state covariance matrix result is obtained at 77652 ns. For performance comparison, a stopwatch timer tool to capture execution time of the software implementation is utilized. The software implementation matrix multiplication uses General Matrix Multiply (GEMM) which is a subroutine in the Basic Linear Algebra (BLAS). GEMM is a building block for so many other routines and has been often tuned by High Performance Computing vendors to run as fast as possible. For the GEMM software implementation test, an Intel (R) Core (TM) 2 Duo CPU E6550 @ 2.23 GHz, 3.32 GB of RAM is used.

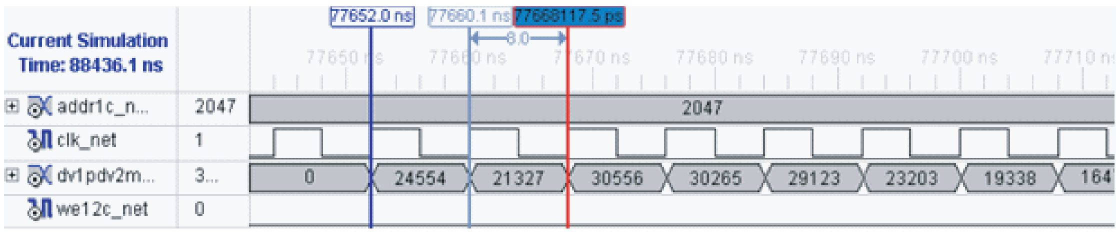


Figure 14. Post route simulation result

The performance comparison also includes four FPGA matrix multiplication implementation methods as listed in Table 2. In MYI classical FPGA implementation, a simple multiplier accumulator is used to calculate the matrix multiplication result. The other three methods utilized a pipelined and parallel architecture of an FPGA to calculate a 4×4 matrix multiplication.

Figure 15 shows proposed implementation results compared with GEMM (SW), Classical FPGA, Idris et al., Prasana et al. and Jang et al. implementation. The execution time to calculate 64 features has been tabulated in Table 3 for clearer comparison. As can be seen, the proposed approach shows significant advantages over other methods which utilize a conventional approach in performing the Monocular SLAM feature state covariance matrix computation.

6. **Discussion and Conclusions.** Feature initialization can be divided into Delayed and Undelayed initialization approach. Based on the advantages which is weighted opposed to

Table 2. Prior FPGA implementation for 4 × 4 matrix multiplication

Design	FPGA	Area (CLBs)	Area (equivalent)	Speed (MHz)
Classical FPGA (24)	XCV1000E	29	29	249/4=62.5
Ju-wook Jang et al (19)	XC2V1500	140	390	166
Prasanna and Tsai (16)	XC2V1500	155	420	166
MYI Idris et al (24)	XCV1000E	242	242	164

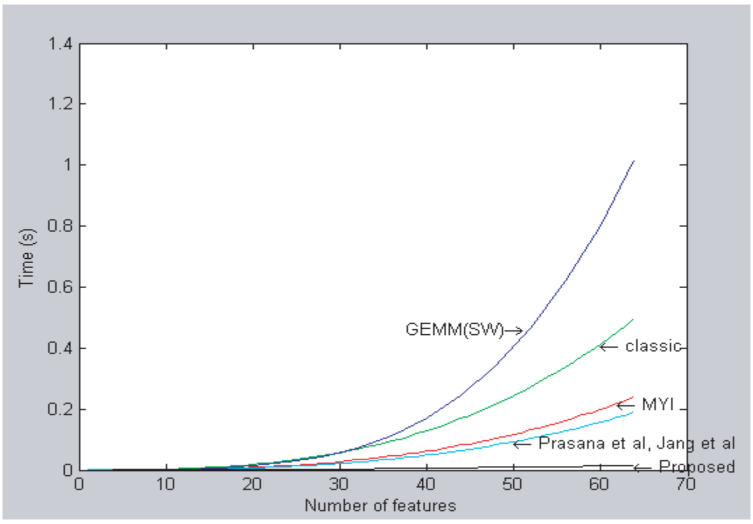


Figure 15. Performance comparison

Table 3. Comparison results when number of features are 64

Design	Time (s)
GEMM (Software)	1.017756
Classical FPGA (24)	0.496894130420163
MYI Idris et al (24)	0.240256180007
Prasana et al (16), Jang et al (19)	0.189889329783728
Reduced Multiplication (proposed)	0.015614976

the Delayed approach, the Undelayed Inverse Depth Parameterization (IDP) or specifically work by Civera et al. [14] is chosen in this paper for further study and improvement. Though IDP is able to reduce some problems faced by the Delayed approach, it suffers from exponential computational load growth which is contributed by the increase of the number of landmarks or features. One of the reasons is the requirement of six state vector parameters compared with three in Euclidean XYZ coding. In this paper, the problem is verified by using a software profiling tool. The profiling tool result shows that the state covariance matrix calculations which include the six state vector parameters contribute to the increase of the execution time. This is true especially when many features are involved. For that reason, the conventional state covariance matrix program as implemented by Civera et al. [14] is studied and tested. Performance to calculate the conventional state covariance matrix is conducted using various matrix multiplication modules. The matrix multiplication module implements GEMM software approach and other four FPGA approaches (i.e., Classical FPGA, Idris et al. [24], Prasana et al. [16] and Jang et al. [19]). Based on the results, Prasana et al. [16] and Jang et al. [19] generate the fastest state covariance matrix computation result. This is because they implement the parallel and pipeline architecture in their 4×4 matrix multiplication module. Although their implementation is fast compared with the software approach, their matrix multiplication module is not specifically designed for Monocular SLAM state covariance matrix calculation. Their implementation still inherited the three conventional state covariance matrix calculation problems. The first problem is related to the insertion of the identity matrix and zero matrix to suits the covariance matrix calculation. This increases the number of multiplication operations. The second problem is due to the matrix size expansion caused by the self matrix multiplication (i.e., in $P = JPJ'$) which contributes to a huge exponential growth problem. The final problem is associated with the recursive function of $P = JPJ'$ that forces the subsequent process to halt until the prior P process completed.

In this paper, the three addressed problems are carefully studied and a proposed FPGA design is put forward. The suggested design proposes a reduced multiplication approach, which discards matrix zero multiplication by modifying the conventional state covariance matrix calculation. The original state covariance matrix formula is also altered to avoid matrix expansion and huge exponential growth problem. By implementing the memory look up approach, the number of multiplications is reduced. Another important component in the proposed design is the independent processing capability. The independent processing capability will be useful for more parallelism implementations in the future. In Figure 15 and Table 3, the proposed approach is compared with other implementations. The result clearly shows the advantage of the proposed design over the other approaches. The proposed FPGA design is 11 times faster than the 4×4 parallel matrix multiplication proposed by Jang et al. and 66 times faster than the GEMM software approach.

Acknowledgment. This project is funded by the Malaysian Government, Ministry of Science Technology and Innovation (MOSTI) and the University of Malaya.

REFERENCES

[1] A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, MonoSLAM: Real-time single camera SLAM, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.29, no.6, pp.1052-1067, 2007.

[2] H. Durrant-Whyte and T. Bailey, Simultaneous localization and mapping: Part I, *IEEE Rob. Autom. Mag.*, vol.13, pp.99-110, 2006.

[3] M. Y. I. Idris, H. Arof, E. M. Tamil, N. M. Noor and Z. Razak, Review of feature detection techniques for simultaneous localization and mapping and system on chip approach, *Information Technology Journal*, vol.8, no.3, pp.250-262, 2009.

[4] R. Munguía and A. Grau, Concurrent initialization for bearing-only SLAM, *Sensors*, vol.10, no.3, pp.1511-1534, 2010.

[5] M. Deans and M. Hebert, Experimental comparison of techniques for localization and mapping using a bearing only sensor, *International Conference on Experimental Robotics*, Honolulu, HI, USA, 2000.

[6] T. Bailey, Constrained initialisation for bearing-only SLAM, *Proc. of IEEE International Conference on Robotics and Automation*, 2003.

[7] S. Williams, G. Dissanayake and H. Durrant-Whyte, Constrained initialisation of the simultaneous localisation and mapping algorithm, *International Conference on Field and Service Robotics*, pp.315-330, 2001.

[8] A. J. Davison, Y. G. Cid and N. Kita, Real-time 3D SLAM with wide-angle vision, *The 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, Lisboa, Portugal, pp.5-7, 2004.

[9] M. Ito and M. Tanaka, Localization of a moving sensor by particle filters, *International Journal of Innovative Computing, Information and Control*, vol.4, no.1, pp.165-174, 2008.

[10] M. Tanaka, Reformation of particle filters in simultaneous localization and mapping problems, *International Journal of Innovative Computing, Information and Control*, vol.5, no.1, pp.119-128, 2009.

[11] M. Kwok and G. Dissanayake, An efficient multiple hypothesis filter for bearing-only SLAM, *Proc. of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendal, Japan, 2004.

[12] J. Sola, A. Monin, M. Devy and T. Lemaire, Undelayed initialization in bearing only SLAM, *Proc. of 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.2499-2504, 2005.

[13] J. M. M. Montiel, J. Civera and A. J. Davison, Unified inverse depth parametrization for monocular SLAM, *Proc. of Robotics: Science and Systems*, 2006.

[14] J. Civera, A. J. Davison and J. M. M. Montiel, Inverse depth parametrization for monocular SLAM, *IEEE Trans. on Robotics*, vol.24, no.5, 2008.

[15] A. J. Davison, Real-time simultaneous localization and mapping with a single camera, *Proc. of International Conference on Computer Vision*, 2003.

[16] K. P. Kumar and Y. Tsai, On synthesizing optimal family of linear systolic arrays for matrix multiplication, *IEEE Trans. on Computers*, vol.40, no.6, 1991.

[17] O. Mencer, M. Morf and M. Flynn, PAM-Blox: High performance FPGA design for adaptive computing, *Proc. of PDCS*, 2001.

[18] A. Amira, A. Bouridane and P. Milligan, Accelerating matrix product on reconfigurable hardware for signal processing, *Field-Programmable Logic and Applications*, pp.101-111, 2001.

[19] J. W. Jang, S. Choi and V. K. Prasanna, Area and time efficient implementation of matrix multiplication on FPGAs, *Proc. of the 1st IEEE Int. Conf. Field Programmable Technology*, 2002.

[20] I. Bravo, P. Jimenez, M. Mazo, J. L. Lazaro, J. J. de las Heras and A. Gardel, Different proposals to matrix multiplication based on FPGAs, *IEEE International Symposium on Industrial Electronics*, pp.1709-1714, 2007.

[21] L. Zhuo and V. K. Prasanna, Scalable and modular algorithms for floating-point matrix multiplication on FPGAs, *Proc. of the 18th International Parallel & Distributed Processing Symposium*, vol.1, pp.92, 2004.

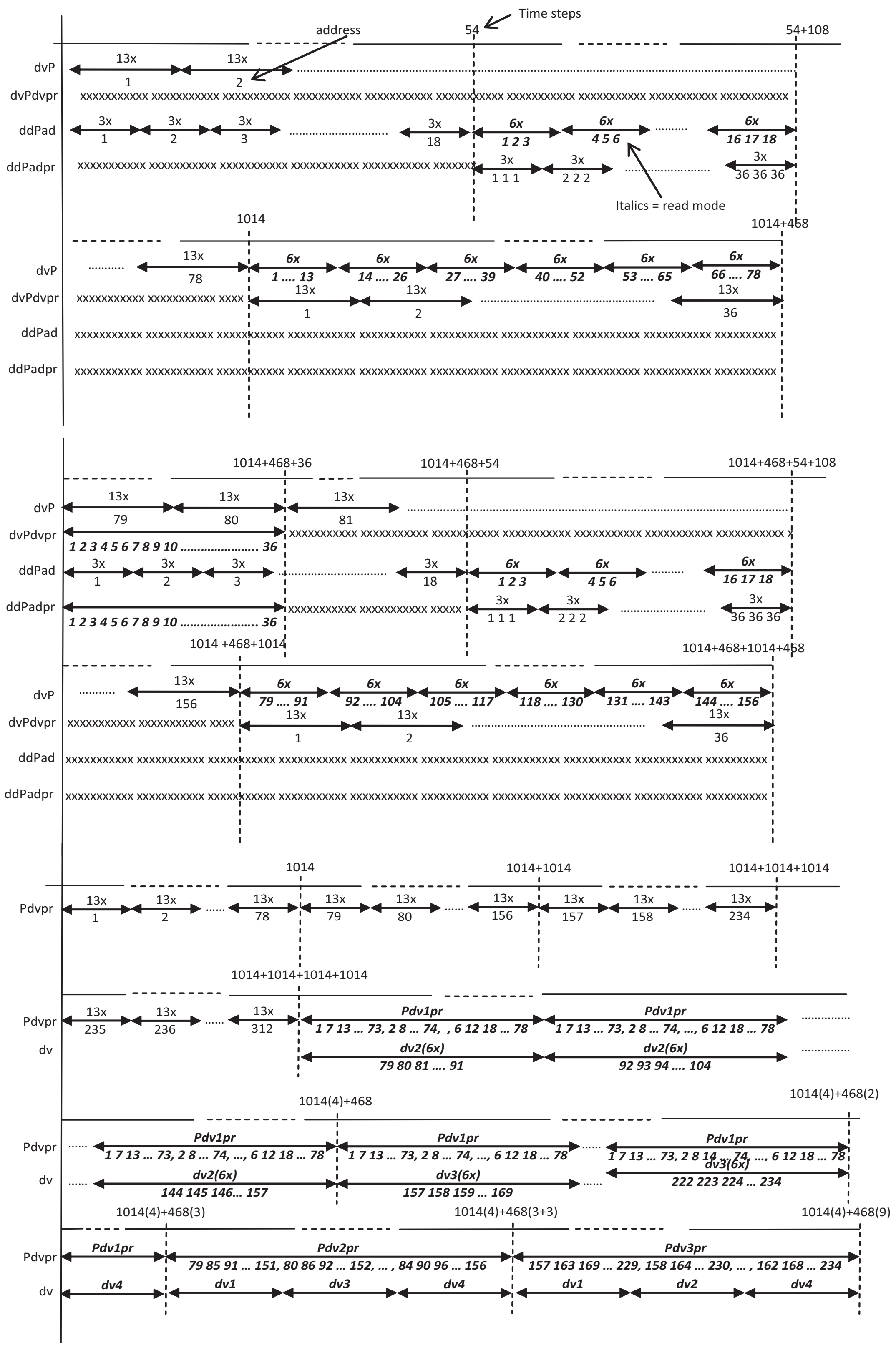
[22] L. Zhuo and V. K. Prasanna, Scalable and modular algorithms for floating-point matrix multiplication on reconfigurable computing systems, *IEEE Transactions on Parallel and Distributed Systems*, vol.18, no.4, pp.433-448, 2007.

[23] M. Y. I. Idris, N. M. Noor, E. M. Tamil, Z. Razak and H. Arof, Parallel matrix multiplication design for monocular SLAM, *Proc. of the 4th Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation*, pp.492-497, 2010.

[24] M. Y. I. Idris, H. Arof, E. M. Tamil, N. M. Noor and Z. Razak, Improving monocular SLAM inverse depth parameterization computation time via software profiling and parallel matrix multiplication, *International Journal of Innovative Computing, Information and Control*, 2011 (in press).

[25] O. C. Zienkiewicz, R. L. Taylor and J. M. Too, Reduced integration technique in general analysis of plates and shells, *International Journal for Numerical Methods in Engineering*, vol.3, pp.275-290, 1971.

Appendix 1



Time steps
address
54
54+108
dvP
dvPdvpr
ddPad
ddPadpr
Italics = read mode
1014
1014+468
1014+468+36
1014+468+54
1014+468+54+108
1014 +468+1014
1014+468+1014+468
1014+1014
1014+1014+1014
1014+1014+1014+1014
Pdvpr
dv
Pdv1pr
Pdv2pr
Pdv3pr
dv1
dv2(6x)
dv3(6x)
dv4
1014(4)+468
1014(4)+468(2)
1014(4)+468(3)
1014(4)+468(3+3)
1014(4)+468(9)