Implementation of Neural Networks to Aid Switched Reluctance Motor Control On the TMS320C6701

Shyam S. Ramamurthy Juan Carlos Balda
Department of Electrical Engineering
University of Arkansas
3217 Bell Engineering Center
Fayetteville, AR 72701
501-575-6578, jcb@engr.uark.edu

Abstract - **Neural Networks (NNs) can be employed to aid in the real-time execution of the torque-prediction and position-estimation tasks in Switched Reluctance Motor (SRM) drives. This paper describes the implementation of such NNs on the TMS320C6701 EVM and highlights the Code Composer Studio Version 1.2 (CCS) features that aid the rapid development of the application. Finally, the paper describes the scheme for interfacing the C6701 EVM to a C240 EVM used for generating the PWM signals for a SRM drive.**

I. INTRODUCTION

SRM drives are now increasing in popularity due to their simple construction features, feasibility for high-speed operation and ability to design them to match an application [1]. Several advantages have been identified for operating the SRM in the multi-phase mode [2-4]. For control under these conditions, it is essential to know (or map) the values of the SRM output variables (e.g., torque) and the machine parameters at different values of the terminal variables (i.e., currents and rotor position).

SRMs typically operate at high levels of saturation of the magnetic circuit. As a result, the mapping between the SRM input variables, output variables and parameters is highly nonlinear. Many techniques have been reported in the past for on-line calculation of these variables under a single-phase operating mode [5]. However, it is difficult to develop equations to calculate easily and accurately the performance variables and parameters under the multi-phase excitation mode. The superposition principle cannot be applied to the multi-phase operation mode by extending the single-phase operation results due to the nonlinear electromagnetic circuit behavior [6]. Hence, a multi-input, single-output mapping is required between the SRM terminal variables and the output or parameters to achieve the desired accuracy in control tasks. Also, it is desirable that this mapping adapts to changes in the parameters of the drive with frequency, temperature and aging effects. These requirements could be met successfully if NNs are applied to provide the desired mapping since they could be also trained on-line to adapt to parameter variations. However, the main obstacle of NNs in the past has been their execution speeds that limit their applicability in real-time control tasks.

This paper first describes the intended application of NNs for SRM control. Then, it reports on the NN implementation entirely in ANSI C using the features of Code Composer Studio Version 1.2. The profiling techniques available in the CCS to aid the rapid code development are then applied to measure the NN execution time of the different stages. The SRM input variables obtained through simulations at different operating conditions are then used to test the NN operation and response. Finally, the paper gives an outline of the schematic for interfacing the C6701 EVM with a C240 EVM that takes care of the low-level tasks of generating the PWM signals for SRM drive control.

II. NN APPLICATIONS FOR SRM CONTROL

This Section addresses the use of NNs in the control of a four-phase 8/6 SRM prototype for an electric vehicle application. Specifically, NNs can be applied for identifying the nonlinear relationship between the terminal variables with the internal parameters and output variables.

Accurate electromagnetic torque feedback is essential in a torque controller. In the particular case of the SRM, the electromagnetic torque is a function of the winding currents and rotor position. The NN should be trained using static torque measurements at different combinations of current levels in the excited windings and at different rotor positions. We used an implementation that predicts the torque as a function of two excited winding currents and rotor position. The two chosen winding currents are those of a phase producing motoring torque (called the working phase) and the one ahead of it (called the leading phase); see Figure 1. The rotor position is measured with respect to the working phase. With this idea in mind, the NN for torque prediction of this SRM prototype is trained for the angle range from –30° to 0° (with respect to the working phase) and for currents from 0 to 15 A in the working and leading phases at various combinations. For our designed SRM and the results presented in this paper, we obtained the above static torque mapping using ANSYS™-based Finite-Element-Analysis (FEA) simulations [7] since the SRM was being constructed.

The NN was then trained and its structure was optimized using MATLAB™. The desired performance accuracy was obtained using a NN consisting of a input layer having 3 neurons, a hidden layer having 7 neurons and an output layer having 1 neuron (see Figure 2). The hidden layer has log-sigmoid activation functions whereas the output layer has a linear activation function.

NNs can be also used to aid the position-estimation task by predicting the mutual interaction factor between the working and leading phases as a function of the winding currents and rotor position. This paper only describes the implementation of the NN for torque prediction on the TMS320C6701 EVM.

III. Description Of The NN Implementation

We implemented the NN on the TMS320C6701 EVM entirely in software using the C language and the features available in the CCS Ver 1.2 [8-11]. From the NN structure in Figure 2, one can anticipate that an important portion of the computational time for the torque output is due to the calculation of the weighted sums at each neuron of the hidden and output layers. The C language code implementing these calculations has a nested loop structure. At each neuron, the first step is to compute the weighted sum at each neuron of the hidden layer for the activation functions. The bias at each neuron is then added to this sum to obtain the net activation input. This is then propagated through the activation function at the neuron. The code in our first implementation had the following form:

```
void comptout()
{
        int i,j;
        for (j=0;j<n1;j++) netin1[j]=bias1[j];
        for (j=0;j<n1;j++)
        {
                for (i=0;i<n0;i++) netin1[j]+=weight1[j][i]*netin0[i];
                actv1[j]=squash(netin1[j]);
        }
        netin2[0]=bias2[0];
        for (i=0;i<n1;i++) netin2[0]+=weight2[0][i]*actv1[i];
        netout=netin2[0];
}
```

where float squash(float) computes the log-sigmoid function using the functions from the math library.

Another time-consuming activity is the computation of the activation using the following log-sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}}$$

One can use the function "exp()" or "expf()" to implement this equation in the C program. The CCS profiler was used to measure the number of cycles required by each function. Also, the precision of the value returned by the exp() and expf() functions was noted. Based on this, it was determined that the expf() function requires much lesser number of cycles (390) than the exp() function (1030) and also offers adequate precision for the application. The log-sigmoid function value, thus computed, has a value of 0 for x<=-15 and a value of 1.0 for x>=15. As a result, it is necessary to compute this function only in the range [–15 < x < 15].

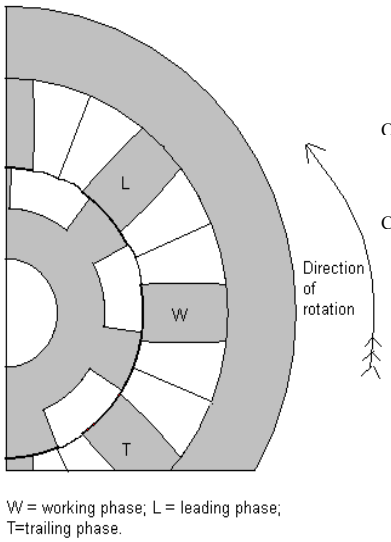

Figure 1. Illustration of the working, leading and trailing phases in the four-phase 8/6 SRM prototype.

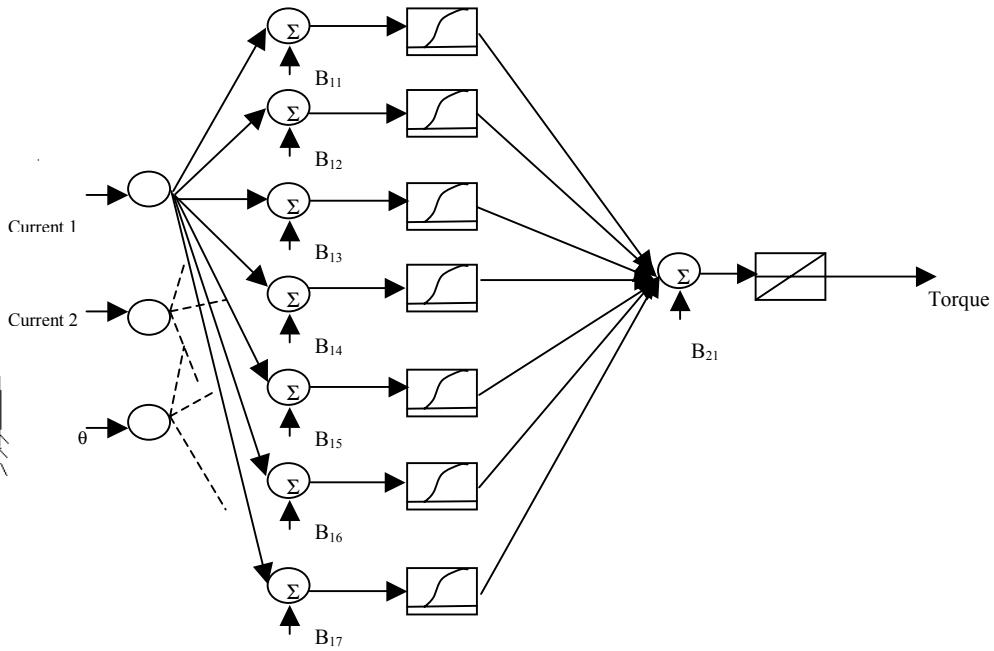

Figure 2. The feed-forward neural network used for torque estimation.

With the log-sigmoid computation using the expf() function, the forward propagation (computing the torque output) takes about 3900 cycles in the worst case when [–15 < x < 15].

This is because the usage of the expf() function leads to the disadvantage that the code contains function calls that cannot be software pipelined. To be able to use code that can be pipelined, we used the "lookup-table" technique for determining the log-sigmoid function values for inputs between the range –15 to 15; these computed values were stored in a lookup table. For a given input value for which the function value is required, the index into the table is calculated using:

$$m = round\left\{(x+15)\times\frac{sigPoints}{30}\right\}$$

where *sigPoints* is the number of points in the lookup table for the log-sigmoid function. This equation was implemented in the C program using type-casting to integer value. However, it was found that direct implementation of this equation in the C program leads to the requirement of a large number of cycles (about 2800 cycles in the worst case with [–15 < x < 15]) because the division process which is used in the index computation is very time consuming. This was overcome by replacing the fraction *sigPoints/30* by its value (say *q*) in the program; that is

$$m = round\{(x+15)\times q\}$$

The initial code using this formulation required about 540 cycles for the forward propagation because the code uses simple addition and multiplication that are the "least expensive" for the processing. Also, the compiler was able to setup better pipelining since the number of registers required during each iteration was small. The loop structure and the arrangement of the code was further improved using the CCS Ver 1.2 features [8-11] and the final code obtained required only 217 cycles for the torque computation by forward propagation. These 217 cycles at a clock of 133 MHz mean that the C6701 requires only $\frac{217}{133\times10^6} = 1.63\,\mu\text{sec}$ for the torque computation by forward propagation.

The following features of the CCS were used for improving the code arrangement and pipelining [8]:

- The profiler was used to determine the number of cycles taken by each section of the code and thus identify the "expensive" sections.
- The #pragma MUST_ITERATE directive was used to pass the information about the trip count (number of loop iterations) to the compiler.
- The #pragma UNROLL directive was used to unroll small loops and increase the number of instructions available for execution in the pipeline.
- The –pm and –mt compiler options were respectively used to direct the compiler to use the trip count data for pipelining, and to indicate the absence of memory aliasing.
- The –k option preserved the .asm file for inspection. The compiler includes feedback about the pipelining in this file and also indicates better options that the user might choose. The code was rearranged till smaller iteration intervals and a larger number of parallel iterations were obtained.

The final code for the forward propagation has the following form:

```
void comptout()
{
        int j;
        int m;
        #pragma MUST_ITERATE(n1);
        for (j=0;j<n1;j++)
        {
                netin1[j]=bias1[j];
                netin1[j]+=weight1[j][0]*netin0[0]+weight1[j][1]
                        netin0[1]+weight1[j][2]netin0[2];
        }
        #pragma MUST_ITERATE(n1);
        for (j=0;j<n1;j++)
        {
                if ((netin1[j]>-15)&&(netin1[j]<15))
                {
                        m=(int)((netin1[j]+15.)*80);
                        actv1[j]=sigTable[m];
                }
                if (netin1[j]<-15) {actv1[j]=0.0;}
                if (netin1[j]>15) {actv1[j]=1.0;}
        }
        netin2[0]=bias2[0];
        #pragma UNROLL(n1);
        for (j=0;j<n1;j++) netin2[0]+=weight2[0][j]*actv1[j];
        netout=netin2[0];
}
```

where sigTable[] contains the value for the log-sigmoid function in the range –15 to 15 and has 2400 points.

The process of learning by error back-propagation involves the computation of activations at the neurons. The back-propagation algorithm was also implemented in C and the code was then optimized. The resulting code executed in 344 cycles or 2.58 μs per iteration.

For electric drives applications, torque feedback at a sampling rate of about 25 kHz maybe considered to be adequate under most conditions. Since the C6701 is able to compute the torque output under multi-phase operation at a much higher speed, there is sufficient time available between the samples to schedule training of an adaptive NN based model (which changes with the actual motor parameters) based on on-line measurements.

This can be used to schedule calculations to compute the new actual values of torque. Finally, the error between the NN torque output and the new calculated value can be used to schedule training of the torque NN. Thus, the C6701 can be used to implement several NNs that cannot only estimate the drive output, but also adapt to changing drive parameters.

IV. Testing Of The NN Using Probe Points

The CCS has the facility to connect data from files on the PC to the program running on the target DSP. This is done using probe points [9]. The terminal current and rotor position data were obtained from SIMULINK™ based simulations [6] and the data corresponding to two different speeds, 100 rpm and 6000 rpm, were stored in ASCII text files in floating-point format. Probe points were used to couple these data files at runtime to the C6701. This was used to test the NN output. Portions of the resulting graphs for the two winding currents, the rotor position and the output torque are included as Figures 3 and 4, which are screen snapshots taken during the simulations.

With sigPoints equal to 2400, it was found that the resulting torque waveform includes additional high-frequency noise introduced by the NN due to rounding of the index value. Increasing sigPoints to 6000 mitigated this noise.

V. Interfacing The C6701 EVM To The C240 EVM

The intended role of the C6701 in the SRM drive system is to provide computational support to the drive controller. The main reason is that present-day DSPs for motion control do not have the required bandwidth to implement novel control algorithms (e.g., model reference adaptive control) when the electric motor is running at high speeds (e.g., greater than 6000rpm). In our implementation, the SRM will be controlled, at the low-level, through PWM signals generated using the C240 EVM. To transfer the computed torque value to the C240 EVM as feedback, an interface between the two DSPs has to be designed. This section presents an overview of the efforts in this direction. The interface between the C6701 and the C240 EVM can be implemented conveniently through the C6701 Host Port Interface (HPI) [10].

The HPI is a 16-bit-wide parallel port through which the C6701 memory space is visible to the host processor (the C240, in this case). The C240 functions as a master to the interface. The data exchange can take place using internal or external memory. The C240 can also access the memory mapped peripherals. Connectivity to the C6701's memory space is provided through the DMA controller in the CPU.

The $\overline{HRDY}$ pin can be used to insert a delay during a transfer and characterize the data transfer speed through the HPI. Since the C240 has a separate address bus, the $\overline{HAS}$ can be inactive high at all times.

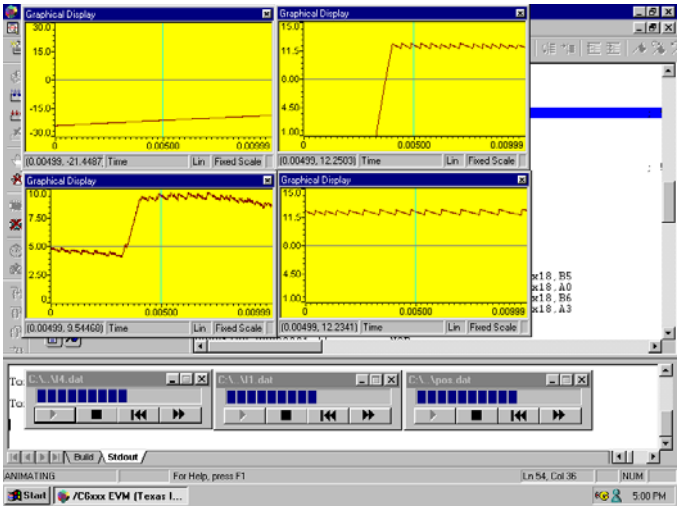

Figure 3. Test results of the NN processing using probe points in CCS Ver 1.2 at 100 rpm. The graphs indicate (CCW from the top left corner) the rotor position (with respect to the working phase), the NN torque output, the current in the leading phase, and the current in the working phase.

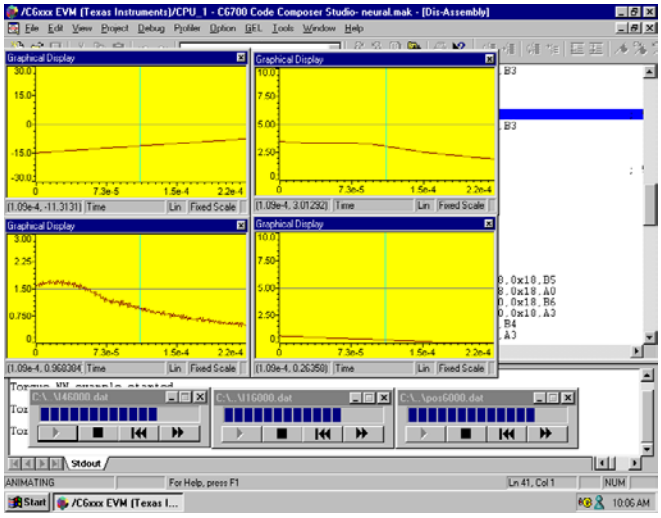

Figure 4. Test results of the NN processing using probe points in CCS Ver 1.2 at 6000 rpm. The graphs indicate (CCW from the top left corner) the rotor position (with respect to the working phase), the NN torque output, the current in the leading phase, and the current in the working phase.

HCNTL [1:0] values would be set by decoding the address of the C240. The base address is determined by the decoding logic and is chosen such that HCNTL1 = 0, HCNTL0 = 0 and HHWIL = 0.

On the C240 EVM, the RAMOE on the GAL device controls the read from external memory operation. On the other hand, RAMWE controls the write to memory operation [11]. These signals are used as separate strobes for the $\overline{HDS\ 1}$ and $\overline{HDS\ 2}$ for read and write. The C240 $R/\overline{W}$ signal is connected to the HPI $HR/\overline{W}$ signal input. HWOB=1 indicates that the first write is the least significant half word. The connection diagram for the interface between the C6701 and the C240 EVM is shown in Figure 5. $\overline{HBE\ [1:0]}$ = 0 since only a half word (16 bits) is written by the C240 during a write operation.

The HPI Control register (HPIC) is normally the first register accessed to set the configuration bits and initialize the interface [10]. The DSPINT bit can be used by the C240 to interrupt the C6701. The FETCH bit of the HPIC is used with the $\overline{HRDY}$ during a read or write to set up a software handshake.

Knowing the base address of the C6701 HPI chosen during the design, the addresses of the control, address and data registers of the HPI are relative displacements to this address and are thus accessible by programs written on the host C240. For example, if
"hpiBaseAddr" denotes the base address of the C6701 HPI chosen to access the control register first half word.
Then:
"hpiBaseAddr+1" accesses the second half word of the control register HPIC to change the control settings and initialize the interface.
"hpiBaseAddr+2" accesses the first half word of the address register HPIA to write the address of the C6701 address map to which access is required.
"hpiBaseAddr+3" accesses the second half word of the address register HPIA to write the address of the C6701 address map to which access is required.

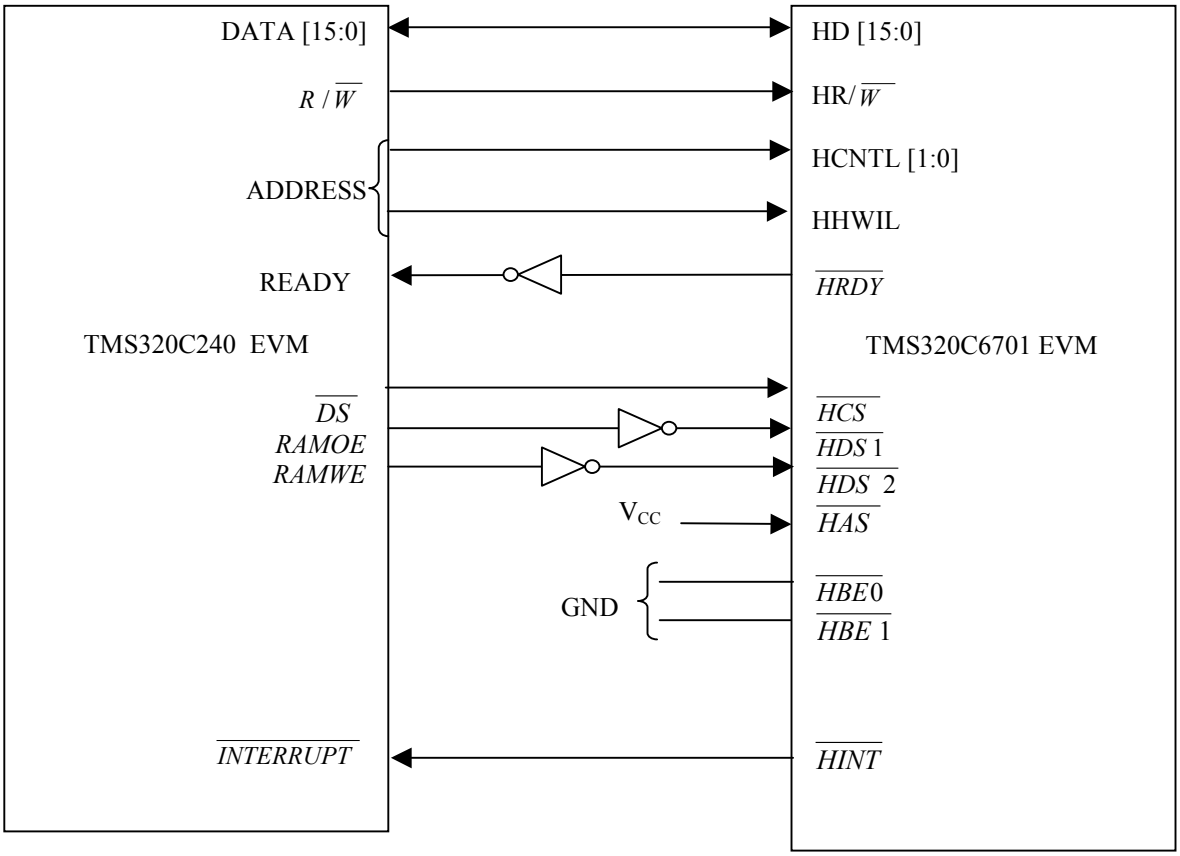

Figure 5. Interface between the C6701 and C240 through the HPI.

"hpiBaseAddr+4" accesses the first half word of the data corresponding to the address placed in the address register through HPID.
"hpiBaseAddr+5" accesses the second half word of the data corresponding to the address placed in the address register through HPID.

V. CONCLUSIONS

The paper demonstrated an implementation of NNs to aid SRM control using the C6701 EVM. The implementation gave an idea of the possible execution speeds. The study indicated the feasibility of implementing several NNs to aid in tackling different aspects of the SRM control problem. A scheme for interfacing the C6701 EVM and the C240 EVM was also presented. At the present time, the cost of the C6701 may make it unfeasible for low-power low-cost drives. However, the benefits in control obtained by applying NN aids makes it attractive for the higher-power drives. The feasibility of incorporating high speeds NNs as control aids when they are implemented on the C6xxx architecture may also indicate the direction for possible enhancements in the architecture of DSPs such as the C240, that are specialized for motion control. It might be useful to incorporate some of the features of express-DSPs such as the C6xxx on the motion control DSPs to allow the implementation of more advanced and adaptive control techniques based on NNs.

ACKNOWLEDGEMENTS

The authors greatly appreciate the financial and technical support from the Office of Naval Research under Grant N00014-98-1-0617 as well as the support extended by the Texas Instruments DSP University Program for this work. The authors also appreciate the FEA data provided by Mr. R. Marcelo Schupbach.

VI. REFERENCES

[1] T. J. E. Miller, Switched Reluctance Motors and their Control, Oxford Science Publications, 1993.
[2] A. Michaelides, C. Pollock, " Short Flux Paths Optimize the Efficiency of a 5-Phase Switched Reluctance Drive", *Conference Proceedings of the 1995 Annual Meeting of the IEEE Industry Applications Society*, pp.286-293.
[3] B. C. Mecrow, "New Winding Configurations for Doubly Salient Reluctance Machines", *IEEE Transactions on Industry Applications*, Vol. 32, No. 6, November-December 1996, pp. 1348-1356.
[4] J. Moon, S. Oh, J. Ahn, Y. Hwang, " Switched Reluctance Motor with 2-Phase Excitation", *Conference Proceedings of the 1998 Annual Meeting of the IEEE Industry Applications Society*, pp. 547-552.
[5] M.Ehsani, B. Fahimi, G. Suresh, J. Mahdavi, "A New Approach to Model Switched Reluctance Motor Drive; Application to Dynamic Performance Prediction, Control and Design", *Power Electronics Specialist Conference*, pp. 2097-2102, 1998.
[6] R.M.Schupbach, Switched-Reluctance Motors: Dynamic Simulation Techniques, M.S. Thesis, Department of Electrical Engineering, University of Arkansas, July 2000.
[7] Swanson analysis systems, ANSYS Revision 5.0 Tutorials, 1992.
[8] TI literature number SPRU198C: TMS32062x/TMS32067x Programmer's Guide, May 1999.
[9] TI literature number SPRU301C: TMS320C6000 Code Composer Studio Tutorial, February 2000.
[10] TI literature number SPRU190C: TMS320C6000 Peripherals, April 1999.
[11] TI literature number SPRU248A: TMS320C24x DSP Controllers Evaluation Module, August 1997.