Hardware Implementation of a Space Vector PWM Technique Using SystemC and VHDL Autocoding

Zivorad R. Mihajlovic[1], Milan S. Adzic[2], Evgenije M. Adzic[3]

Abstract – **This paper deals with application of SystemC language for modeling and implementation of pulse-width modulation technique, wide used in vector control of standard AC drives. Software implementation of SVPWM require much of processor's time which is necessary for real time execution of complete control structure. This paper considers possibility of SVPWM hardware acceleration. Hardware equivalent of SVPWM modulator was developed based on SystemC model and using SystemCrafter SC tool for automatic VHDL code generation.**

Keywords – **SystemC, VHDL Autocoding, Space Vector PWM, hardware implementation.**

I. INTRODUCTION

Nowadays, the main quailty of general motor drives for house appliance is observed through economic aspects. Market competition is now very strong, with frequently changes in standards and requirements, which lead to time-to-market reduction. The main part of the production price represents development costs, particularly for software. The size and requirements of algorithms that have been implemented in embedded control units has increased dramatically. Programming control units in standard languages, such are assembler and C, is not acceptable anymore. More flexible approach is required, as modeling on high level abstraction. It is recommended to separate hardware and software components on this level. Due to high price of software development, the aim is to implement part of the algorithm into the cheaper hardware. Already in start, software development is delayed expecting newer, cheaper and more reliable hardware, as with slow communication between software and hardware engineers. Hardware-software codesign represents useful technique for product acceleration [1, 3]. In this case a software and hardware engineers works on the same development tool.

SystemC language represents modeling tool for hardware-software codesign. It can simulate the hardware and software partitions in the same framework. It consists of a set of class libraries for C++ that describes hardware constructs and concepts.

[1]Zivorad R. Mihajlovic is with the Faculty of Technical Sciences, Trg D. Obradovica 6a, 21000 Novi Sad, Serbia, E-mail: zivorad@uns.ns.ac.yu

[2]Milan S. Adzic is with the Technical faculty in Subotica, Marka Oreskovica 16, 24000 Subotica, Serbia, E-mail: adzicm@vts.su.ac.yu

[3]Evgenije M. Adzic is with the Faculty of Technical Sciences, Trg D. Obradovica 6a, 21000 Novi Sad, Serbia, E-mail: evgenije@uns.ns.ac.yu

This means that it could be used to develop cycle-accurate models of hardware, software and interfaces, which could be simulated and debugged within existing C++ development environment [2]. For that reason it allows fast and easy verification of complex algorithms.

SystemC was originally developed as a system modelling and verification tool, but still require manual translation to a hardware description language to produce hardware. SystemCrafter SC automates this process, by quickly synthesizing SystemC to RTL VHDL or Verilog [3]. It will also generate a SystemC description of the synthesized circuit, which can be used to verify the generated code using existing test environment.

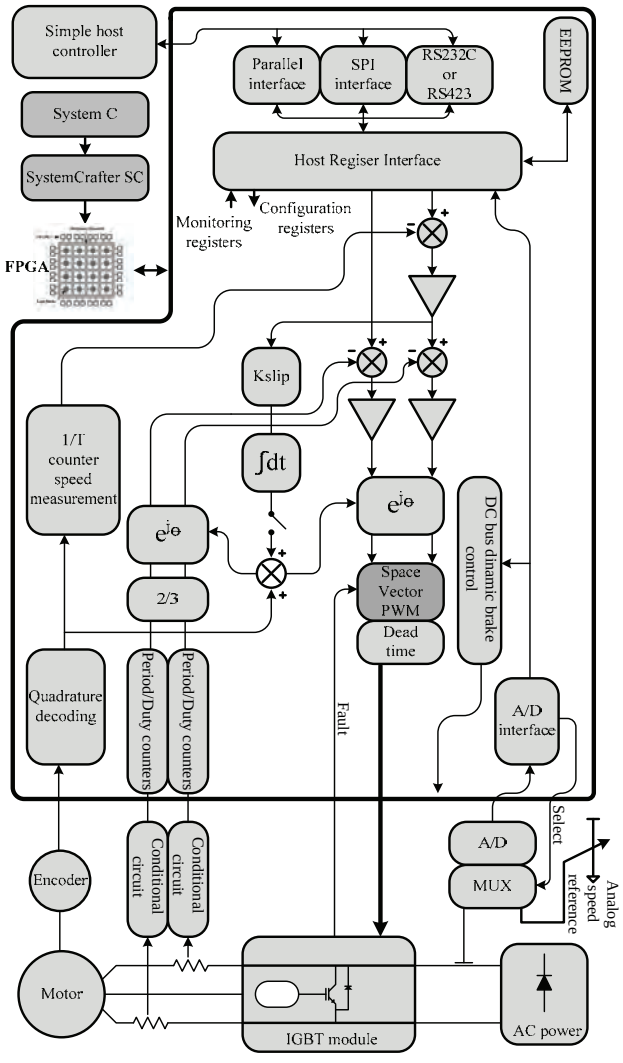

Fig. 1. Vector control system encapsulated on FPGA circuit.

In this paper, attention is to indicate the advantages of this approach on the example in designing a platform for complete drive system. The solution anticipates that all numerical intensive parts of the code (modulator, coordinate transformations, controllers, state and parameters estimators) are transferred to hardware by usage of SystemC language and SystemCrafter SC autocoding tool (Fig. 1). These steps are given on example of space vector pulse-width modulator (SVPWM) which is used for control a three-phase inverters in order to produce variable sinusoidal output voltage.

II. SPACE VECTOR PULSE-WIDTH MODULATOR

One of the most numericaly extensive part of the code for vector control of AC drives is SVPWM. Switching frequency is required to be relatively high, above 16 kHz, in order to achieve high bandwidth torque control and to minimize noise. For that reason there is not enough computing time for calculation of complete control task. Usage of hardware implemented SVPWM and providing the user proper interface, could be solution for this problem. Moreover, complete vector control drive system could be translated to a hardware component, which would dramatically shortened design time. In that case, user could readily evaluate performance of vector control without spending development effort usually required in the traditional DSP based system.

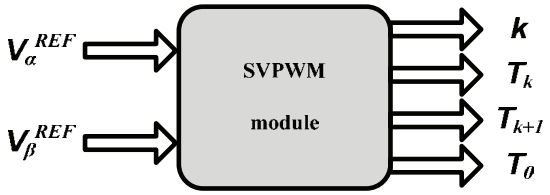

Fig. 2. SVPWM module with inputs V_α^{REF} i V_β^{REF}.

It is known that a balanced three-phase set of voltages is represented in the stationary reference frame by a space vector of constant magnitude, equal to the amplitude of the voltages, and rotating with angular speed $\omega = 2\pi \cdot f_{REF}$ [5]. So, space vector is represented by two components, named alpha and beta. The space vector module can create the PWM switching pattern for three-phase inverter using directly the referent alpha and beta voltage components, V_α^{REF} and V_β^{REF} (Fig. 2). Current control is done in synchronous rotating reference frame where sinusoidal AC signals, like V_α and V_β, became DC quantities, known as *d* and *q* components [5]. SVPWM technique is mostly used for digital current control, giving to the linear current regulators full control over the output voltage *d* and *q* components. These *d* and *q* components, defined on the current regulators outputs, are transfer back to the stationary alpha beta coordinate system and passed thru to the space vector modulator. As it could be seen in Fig. 3, the eight possible states of an inverter are represented as two null-vectors (V_0, V_7) and six active-state vectors forming a hexagon (V_1-V_6). SVPWM approximates the rotating reference vector in each switching cycle by switching between the two nearest active-state vectors and the null-vectors. The main task of space vector modulator is to calculate the needed vector times directly from V_α^{REF} and V_β^{REF}.

The connection between the V_α^{REF} and V_β^{REF} and needed active and zero vector times is easy to be understood looking in Fig. 3. Assuming that reference vector V^{REF} is sitting in sector *k*, and that two nearest vectors are V_k and V_{k+1}, following equation can be written:

$$\vec{V}_{REF}\frac{T_s}{2} = \vec{V}_k T_k + \vec{V}_{k+1} T_{k+1} \tag{1}$$

where T_k represents half of the vector V_k on-times in switching period T_s.

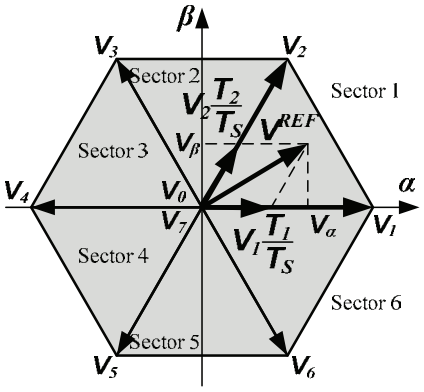

Fig. 3. Generation of V^{REF} vector using V_1, V_2 and zero vector.

Splitting vector equation Eq. (1) into its real and imaginary part, and after rearranging it follows:

$$\begin{bmatrix} T_k \\ T_{k+1} \end{bmatrix} = \frac{(T_s/2)}{V_{DC}} \begin{bmatrix} \sqrt{3}\sin(k\frac{\pi}{3}) & -\sqrt{3}\cos(k\frac{\pi}{3}) \\ -\sqrt{3}\sin((k-1)\frac{\pi}{3}) & \sqrt{3}\cos((k-1)\frac{\pi}{3}) \end{bmatrix} \begin{bmatrix} V_\alpha \\ V_\beta \end{bmatrix}$$

$$T_0 = \frac{T_s}{2} - T_k - T_{k+1} \tag{2}$$

III. SYSTEMC REALIZATION

As a language for high level modeling, SystemC supports not only first realization of user's project specification but the first functional verification and creation of executable description of intended design [4]. It is possible to divide complete model in desired number of functional parts. This means that designer might refine a module from a high level functional specification down to a cycle-accurate RTL model while other modules in the system remain at higher level of abstraction. Modeling of SVPWM represents this methodology.

SystemC model of SVPWM consists of a two processes. These processes execute functions *do_pwm_gen()* and *do_svpwm()*, concurrently. Test bench generate input stimulus for SVPWM module and also record outputs for verification. Interface of SVPWM module was shown in Fig. 4. Input ports are alpha and beta components of reference vector and clock signal for implemented PWM peripheral unit. Output ports are driving signals for each transistor of three-phase inverter. It is possible to use whichever module's signal for functional verification. In this case, test bench observes signal *pwm_int*.

Function *do_pwm_gen()* is a part of process which generates driving signals, and also signal *pwm_int* needed for calling PWM interrupt service routine *do_svpwm()*, which calculate needed duty cycle times. *do_pwm_gen()* function triggers on every clock cycle and increment or decrement timer of realized PWM unit. On every PWM timer underflow, interrupt *pwm_int* is generated. Also, *do_pwm_gen()* compares PWM timer and calculated duty cycle times, in order to generate output driving signals.

```
#ifndef MODULATOR_H
#define MODULATOR_H
#ifndef SC_SYNTHESIS
#include "systemc.h"
#endif
class modulator : public sc_module
{
    public:
    sc_in<sc_int<16> > v_alpha_ref;
    sc_in<sc_int<16> > v_beta_ref;
    sc_in<bool>      clock;
    sc_out<bool> pwm1h;
    sc_out<bool> pwm1l;
    sc_out<bool> pwm2h;
    sc_out<bool> pwm2l;
    sc_out<bool> pwm3h;
    sc_out<bool> pwm3l;
    sc_out<bool>   pwm_int;
    sc_uint<16>       pwm_period_reg;
    sc_signal<sc_int<16> > pwm_val_a;
    sc_signal<sc_int<16> > pwm_val_b;
    sc_signal<sc_int<16> > pwm_val_c;
    void do_pwm_gen();
    void do_svpwm();
    SC_CTOR (modulator)
    {
            SC_THREAD(do_pwm_gen);
                    sensitive_pos << clock;
            SC_METHOD(do_svpwm);
                    sensitive_pos << pwm_int;
    }
};
#endif
```

Fig. 4. SystemC interface of SVPWM module.

Function *do_svpwm()* use fixed-point arithmetic, known as IQ math, to calculate duty cycle times on every rising edge of *pwm_int* signal generated in process described above. First, sector number of reference vector is detected and than the space vector times are calculated using Eqs. (1) and (2) . In contrast to *do_pwm_gen()*, which is a thread, *do_svpwm()* is a method process. Method process executes all instructions on rising edge of *pwm_int* signal and ensures that all calculations are accomplished between two transitions of this signal.

Test bench consists of a three processes. First, that is a thread process sensitive to rising edge of *pwm_int* which supplies input ports *v_alpha_ref* and *v_beta_ref* with packets of data for verification of the module. For testing, input port *v_alpha_ref* was an array of predefined values of sinusoidal signal. Input port *v_beta_ref* use the same array but delayed in order to represent cosine function (Fig. 5). Supplying inputs *v_alpha_ref* and *v_beta_ref* in this manner can lead to understandable results, by monitoring calculated duty cycle times.

```
while(1)
{
    v_alpha_ref = sineTable[i%401] >> 1;
    v_beta_ref = sineTable[(i+100)%401] >> 1;
    i++;
    if (i==4010) i=0;
    cout      << "Input is "
              << v_alpha_ref.read()
              << v_beta_ref.read()
              << "\n";
    wait();
}
```

Fig. 5. Part of test bench program for supplying inputs.

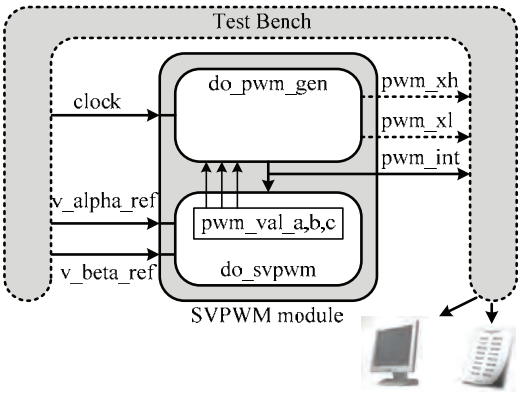

Fig. 6. Conection between SVPWM module and test bench.

Second and third processes are method type. One prints outputs on standard monitor, while second writes them to a file (Fig. 6). Usage of these accessories is allowed including the *fstream* C++ header in the project. Processes can monitor not only output ports, but all signals inside the SVPWM module. For example, in interface for SVPWM module (Fig. 6), there are three duty cycle signals named *pwm_val_a*, *pwm_val_b*, and *pwm_val_c* which are compared with PWM timer and generate output driving signals. These signals are not ports, but their monitoring can help in the module verification.

IV. AUTOCODING

After verification in SystemC, SVPWM module can be translated to hardware description language such are VHDL or Verilog. Whole process of translating SystemC model to hardware requires skilled engineers with experiences in both design methodology, but this request is rarely satisfied at the moment. This is a main reason for appliance of autocoding approach. One of the tools available on the market for automatic translation is SystemCrafter SC.

SystemCrafter SC automatically synthesizes hardware designs written in SystemC to HDL. The HDL can then be used with commonly available tools to target Xilinx FPGAs. This enables engineers and programmers to design, debug and simulate hardware and systems using their existing C++ development environment. The result is improved

productivity and very fast simulations, plus all the benefits of using their existing VHDL or Verilog design flow.

In Fig. 7., a place of the SystemCrafter tool in complete designing process, is shown. Interface in Fig 4. of SVPWM includes macro SC_SYNTHESIS used to distinguish between standard SystemC and SystemCrafter compilation. As it is a new tool on the market, SystemCrafter do not support all aspects of SystemC modeling. For example, it is impossible to initialize array after defining. It is allowed to initialize each member of an array, individually. Dealing with inout ports is not implemented yet. After refining SystemC model of SVPWM, according to these limitations, SystemCrafter model was attained. Gate level model generated by SystemCrafter with the same test bench provided the same results, which was proof of successful translation. SystemCrafter also generates VHDL files, ready for FPGA implementation. Every process from SystemC model was translated into one VHDL file. These files are encapsulated into the main VHDL file also generated by SystemCrafter. Together with library file *craftgatelibrary.h*, included in SystemCrafter installation, these files represents hardware implementation of SVPWM module.

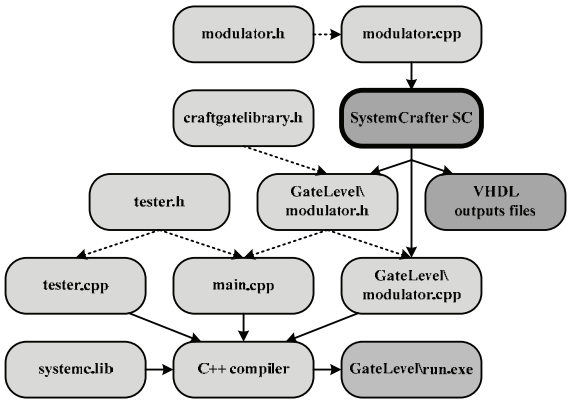

Fig. 7. Using SystemCrafter SC tool for VHDL autocoding.

V. RESULTS

In Figs. 8 and 9, results of the test are shown. Input ports *v_alpha_ref* and *v_beta_ref* are associated with sine and cosine functions of magnitude $|V^{REF}| = 16384$, which represents normalized value of *0.5 p.u.* Period of clock signal for incrementing/decrementing PWM timer was *20 ns*. For that reason, period register in PWM unit is initialized to a value of *1250*, in order to achieve switching frequency of *20 kHz*. Sine and cosine look-up tables had *400* values, so with PWM frequency of *20 kHz*, desired frequency of output was *50 Hz*.

Fig. 8 shows inputs *v_alpha_ref* and *v_beta_ref*, while Fig. 9 shows changes of calculated duty cycle values for two output phases during test time of *20 ms*. It can be seen that duty cycle values are changed around half of the PWM period register (= *625*), that they have expected waveform with two noticable hunchs in the region of peak values, and that they are phase shifted by *120º*. Results was same in both system and gate level, and in accordance with expected.

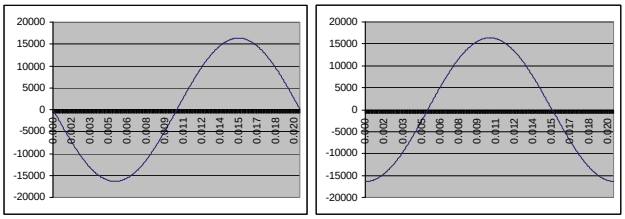
Fig. 8. *v_alpha_ref* and *v_beta_ref* inputs.

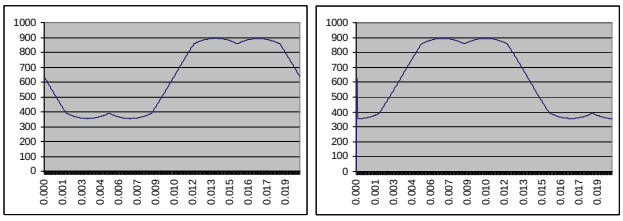
Fig. 9. SVPWM duty cycle times for two output phases.

VI. CONCLUSION

Fast growing of circuit compexity leads to larger usage tools for automatic design. This example shows that future design methodology moves through precise defined steps. These steps are:

- Development of initial SystemC description.
- Writing a test bench.
- Debuging and verification of description.
- Refinement to more efficient hardware.
- Experimentation with trade-offs.
- Verification of the refined description.
- Usage of autocoding tool.
- Verification of the synthesized hardware.

This paper describes and verifies above steps on the example of hardware implemented SVPWM modulator used in control of three-phase AC drives. In this case, SystemC has approved as excellent enviroment for testing complex algorithms and that together with VHDL autocoder SystemCrafter represents powerful tool for high level abstraction. Next step, would be testing of generated code on real FPGA platform, which would show true power of used tools.

REFERENCES

[1] A. Avila, R. Santoyo, S. Martinez, G. Dieck, "Hardware/Software Implementation of a Discrete Cosine Transform Algorithm Using SystemC", IEEE, *ReConFig 2005 Conference Proceedings*, pp. 28-31 IEEE Computer Society Digital Library, Puebla City, Mexico, 2005.

[2] P. R. Panda, "SystemC - A Modeling Platform Supporting Multiple Design Abstractions", IEEE, ISSS 2001 Conference Proceedings, pp. 75-80, Montreal, Canada, 2001.

[3] G. Dujic, Z. Mihajlovic, I. Mezei, "Using SystemC to Model and Synthesize Position Measuring Controller on FPGA Circuit", 14th International Symposium on Power Electronics - Ee 2007, Novi Sad, Serbia, 2007.

[4] T. Grotker, System Design with SystemC, Hingham, MA, USA, Kluwer Academic Publishers, 2002.

[5] P. Krause, Analysis of Electric Machinery and Drive Systems, New York, John Wiley and Sons, 2002.