Model Based Design of an Efficient CORDIC Algorithm Implementation

Volker Zerbe and Michael Backhaus

***Abstract*—Navigational tasks require the efficient computation of trigonometric functions. For the development of an electronic compass the arc tangent is to be computed for example. The electronic compass is model based design. Hardware solutions are of special interest. The CORDIC algorithm stands in the focus for the computation of trigonometric functions. It is based on *shift* and *add* operations and permits an efficient implementation in FPGA`s. The paper describes the model based design concept of the compass design. It is explained how the CORDIC algorithm works. Thereby first software solutions are compared: Approximation using the Taylor series versus the CORDIC Java implementation. Different hardware solutions of the CORDIC algorithm are analyzed. A Pipeline CORDIC processor is introduced and embedded into the electronic compass. The developed system is validated using an example.**

***Index Terms*—Model based design, compass, CORDIC, FPGA.**

I. Introduction

THE goal of the design task is to develop an optimal overall system under the boundary conditions of limited resources. At the same time the costs are to be minimized and the time to market is to be reduced. The question is: How can, for example, different architectures be analyzed and optimized, in order to reach this goal. First an executable system model has to be provided, before performance assessments can be accomplished. The modelling of the overall system begins on abstract level. A structured approach is essential together with the stepwise refinement of the model. The complex design problem must be divided thereby in a regulatory way into manageable subproblems, so that their complexity permits a treatment. It is important to note that the validity of the solution of a subproblem is given always only in the context of the solution of the entire design problem. This means, the solution of a subproblem does not contribute automatically to the solution of the overall problem. The solution of a subproblem is evaluated always on their contribution for the solution of the superordinate design problem. With this kind of modelling of the system with its subsystems and components over all abstraction levels, internal system details on higher modelling levels are hidden. They emerge only in the leaves of the hierarchical model tree. The execution of the model, the simulation, now permits the analysis of the system for limited, available resources and system requirements.

For the design of complex, heterogeneous, integrated, networked systems different computation models in the architecture have to be integrated. A computation model is a mathematical formalism which defines a set of permissible operations for one computation and of implementation details abstracts. Thus, concurrencies, data flows, reactive and continuous systems, synchronisation and communication aspects can be described adequately. Each subsystem or each component of the whole to be modelled system should able to use every computation model.

Of special importance is the integration of the top down and the bottom up design for the modelling of systems. The top down methodology permits the modelling in an abstract way. The draft can be made clear and manageable. The given design space by the stepwise refined component decomposition is done until the desired degree of detail of the system is reached. With the bottom up methodology, subsystems and components are modelled and combined into an overall system. So experiences from past system developments can be brought into the design. Hence it follows that both techniques for practice must be combined. This combination of top down and bottom up is well known as the meet in the middle strategy [2]. Anyway, the main strategy should be the top down approach. In this case as much as possible alternatives can be examined and optimal decisions be made. Performance parameters can be refined stepwise or can be represented as an annotation in the system modell through the bottom up methodology. These parameters are helpfull to accomplish performance assessments of the system.

II. Model Based Design

At present time complex, embedded, networked systems are developed purely requirement oriented (requirement based). System requirements are gathered to provide paper specifications from different sources. On this basis many distributed developer teams develop a detailed design for the subsystems and components. After reaching an accepted design the validation and the test of the overall system is to be

V. Zerbe is with the University of Applied Sciences Erfurt, Department of Computer Engineering/Embedded Systems, Germany (e-mail: volker.zerbe@fh-erfurt.de).

M. Backhaus is with the Ilmenau University of Technology, Department of System and Software Engineering, Germany (e-mail: michael.backhaus@tu-ilmenau.de).

done. Therefore errors during the design process will be discovered often very late. The design based on executable models (model based design) has the potential to increase the productivity of the system design process substantially. This design strategy is model driven and begins already in the early design phases with the development of an executable specification. This executable specification is directly linked with the system requirements. In the center of the overall model based design process stands the to be modeled complex system, the executable model. It is refined sequentially, stepwise. By the linkage of the executable model with the system requirements inconsistencies in the system requirements can be found very timely found by simulation. During the overall design process it can be examined whether the requirements correspond to the design and which effects will be caused by the suggested change in the system requirements [4]. The model based design has the following conspicuous characteristics in relation to the requirement based design, (in extracts):

- The investigation of the dynamic behavior of the system becomes already possible in the early phases of the design process.
- Based on simulatable alternatives and associated trade off analyses design decisions can be made.
- Only one model is used on different design levels.
- Nonfunctional requirements can also be modelled and validated.

III. MLDesigner

MLDesigner [10] is a tool for the design of complex systems on mission and system level. This approach integrates architecture, function and application scenarios in only one development environment on a very abstract level. MLDesigner is a multi domain simulator and supports the modelling in discrete event (DE), synchronous data flow (SDF), continuous time (CT), finite state machine (FSM) and other computation models. Different computation models can be combined in order to model a system [12]. Furthermore a system model can be represented with its components in an arbitrary depth of detail. On this basis system performance evaluations can be accomplished. A comparison and an evaluation of different tools for the design on system level was made in [11].

IV. The Electronic Compass Model

An electronic compass is able to continuously indicate [6] the azimuth angle. Two magnetic field sensors, KMZ51 by Philips, generate voltages V_x and V_y from which the azimuth can be determined then.

$$\alpha = \arctan\frac{H_{ey}}{H_{ex}} = \arctan\frac{V_y}{V_x} \quad (1)$$

H_{ex} and H_{ey} are vectors in the earth field, see Fig. 1.

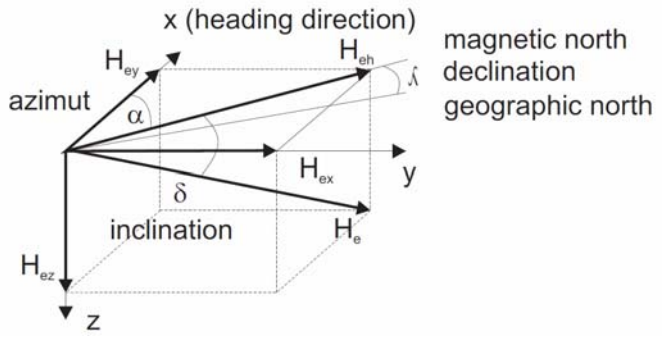

Fig. 1. Earth field vectors.

This picture shows three dimensional the earth field vectors. The $x - y$ plane lies parallel to the earth surface. The azimuth has to be computed. The angle δ, the inclination, is the tilting angle of the magnetic field lines. This angle is different for different positions on the earth (at the north pole differently than in Ilmenau or at the equator). The declination, λ, between magnetic and the true geographic north amounts to approximately 11,5 degrees.

An executable model for an electronic compass was developed with the help of the system design tool MLDesigner. The to be modeled system is a data flow oriented system [1]. One source (magnetic field sensor) produces continuously tokens ($x - y$ voltage data) which are consumed by the following nodes (amplifier, converter, processor, display). Basis for the modeling is the SDF computation model. The goal is the development of an optimal overall system. Fig. 2 shows the overall system of an electronic compass. The model can be formally verified. For example for SDF graphs the following applies:

Definition 1 (Deadlock): Is a computed Schedule not more continuable, since no more node can fire, then a deadlock is present. Cycle free graphs are deadlock free.

The top level compass model is therefore deadlock free in the sense of a SDF graph.

The module sensor supplies corrected (offset, sensitivity, orthogonality, temperature compensation) sensor outputs (voltage data). An operational amplifier circuit, modeled in the module *amplify*, is amplifying the weak sensor output signal. These voltage data are analogue digital converted in the module *A/D Conv*. Here for example different parameters for converters are adjustable. The computation of the arc tangent is modeled as a main function in the module *CORDIC processor*. Pure software solutions are examined based on the Taylor series and the CORDIC implementation. Further the

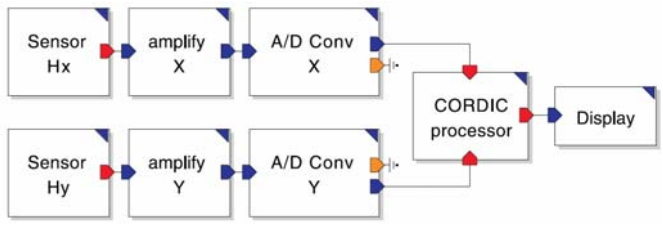

Fig. 2. Top-Level MLDesigner Compass Model.

focus is directed on a FPGA (field programming gate array) implementation. The module *display* is a data sink. Only tokens are consumed. The azimuth angle α is written as output data.

A. The CORDIC Algorithm

The CORDIC algorithm was introduced 1959 for the first time by Jack Volder [5]. CORDIC is the abbreviation for coordinate rotating digital computer. Starting point for the development of the CORDIC was the desire to handover the continuous computation of navigation algorithms to digital systems. The world of the digital signal processing is dominated by microprocessors. On one side they are low priced and extremely flexible, on the other side they are often not really fast enough for heavy DSP (digital signal processing) tasks. Available reconfigurable hardware makes it possible to achieve a higher speed in computation compared to the traditional software approach. Unfortunately for microprocessor based systems, optimized algorithms are not well implementable in hardware. Nevertheless, there exists a multiplicity of hardware efficient solutions. Among them there is also a class of iterative solutions for trigonometric functions. John Walther [7] extended the CORDIC theory. Thus using the CORDIC computation of hyperbolic, exponential and logarithmic functions are also possible. Kota [8] has accomplished error and load analyses.

All computations of trigonometric functions are based on vector rotations. The vector E_0 is rotated around the angle θ, see Fig. 3. The vector E_n results.

$$x_0 = \cos\theta_0 r_i \qquad y_0 = \sin\theta_0 r_i$$

$$x_{i+1} = \cos(\theta_0 + \theta) r_i \qquad y_{i+1} = \sin(\theta_0 + \theta) r_i$$

The general rotation transformation in matrix form results to

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \cos\theta & \sin\theta \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{2}$$

or in a different way of writing:

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2\theta}} \begin{bmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}.$$

The rotation around the angle θ is realized by a sequence of rotations around the angular α_i.

$$\theta = \sum_{i=0}^{n-1} d_i \cdot \alpha_i \quad \text{mit} \quad d_i \in \{-1, 1\} \tag{3}$$

The angle θ is approximated by an alternating approach. A too far rotation is compensated by a change of sign. To control the direction of the rotation an auxiliary variable z_i is introduced.

$$z_0 = \theta \quad z_{i+1} = z_i - d_i \cdot \alpha_i \quad d_i = \begin{cases} 1 & z \geq 0 \\ -1 & z_z < 0 \end{cases}$$

In order to simplify the rotation equation [9], the rotations are replaced by pseudo rotations, see Fig. 4. Thus the length of the rotating vector changes by a well known angle with a constant factor.

$$r_{i+1} = r_i \cdot \sqrt{1 + \tan^2 \alpha_i} \tag{4}$$

Let $\tan\alpha_i = 2^{-i}, \quad i = 0 \ldots n-1$.

Thus the following new, simplified rotation equations result:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & -d_i \cdot 2^{-i} \\ d_i \cdot 2^{-i} & 1 \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{5}$$

$$z_{i+1} = z_i - d_i \cdot \arctan\left(2^{-i}\right) \tag{6}$$

For the computation of the different functions the CORDIC can operate in two modes. These are the rotation and the vector mode. For the computation of the arc tangent the vector mode is used. The given vector is always rotated so that the absolute value of its *y* component is reduced. The rotation angle is signed accumulated thereby. After processing all iteration steps, the following equations results:

$$x_n = \left(\prod \sqrt{1 + 2^{-2i}}\right) \sqrt{x_0^2 + y_0^2}$$

$$y_n = 0 \tag{7}$$

$$z_n = z_0 + \arctan\frac{y_0}{x_0}$$

Choose $z_0 = 0$, so z_n is equal to the desired azimuth.

B. Algorithm Analysis

In the module CORDIC processor, see Fig. 2, the arc tangent is modeled. First an approximation, using the Taylor series, is compared with the CORDIC algorithm. Both algorithms were implemented directly in the MLDesigner model using the programming language C.

1) **Approximation using the Taylor series**

Taylor series expansions sometimes show a slow convergence and a numeric instability and require many multiplications additionally. The description complexity is nevertheless low, see the following Java code.

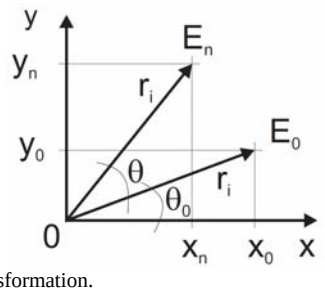

Fig. 3. Vector transformation.

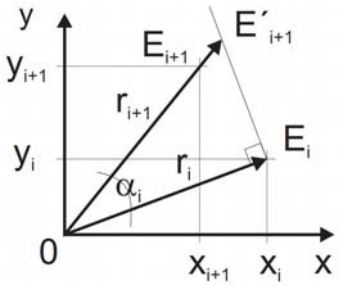

Fig. 4. Pseudo rotation.

```
  public static double taylor_atan2(double y,
double x, double math, int n){
  double z=y/x;
  double result=0;
  double numerator=z;
  double denominator=1;
  z*=z;
  for(int i=0; i<n; i++){
  result+=enumerator/denominator;
  numerator*=-z;
  denominator+=2;}
return Math.toDegrees(ergebnis);
}
```

2) **CORDIC Java implementation**

In comparison with the approximation using the Taylor series the CORDIC needs only simple shift and add operations as well as a LookUpTable operation. The description complexity is comparable with that of the Taylor implementation.

```
  public static double cordic_atan2(double y,
double x, int n){
  double newX;
  double z = 0;
  double half=1;
  int d=1;
  for(int i=0; i<=n; i++){
  if(y>=0) d=-1; //direction of rotation
  else d=1;
  newX=x-d*half*y;
  y=y+d*half*x;
  x=newX;
  z=z-d*angles[i]; //precalculated angles
  half/=2;}
  return z;}
}
```

The simulation of both modeled algorithms produces the following results, which are exemplary summarized in a table. First the number of iterations was investigated, which leads to an accuracy of ≤ 0.001° concerning the reference angle.

x	y	ref. angle	CORDIC (iterations)	Taylor (iterations)
0.92	0.39	22.97272	12	5
0.72	0.69	43.78112	14	64

If the number of iterations is set to 16 for both algorithms, then the accuracy of the results varies very strongly. In particular the Taylor algorithm shows clear weaknesses regarding the accuracy, see table below.

x	y	ref. angle	CORDIC (error)	Taylor (error)
0.92	0.39	22.97272	0.0134	0.0000
0.72	0.69	43.78112	0.0115	0.2286

The CORDIC obtains a high accuracy of the computed result for a constant number of iterations in the entire coordinate system. Introducing the pseudo rotations and the represented simplifications the CORDIC is limited to only two *shift*, three *add/sub* operations and one *LookUpTable* operation. That way the system is also efficient implementable in hardware.

In the sense of the model based design the CORDIC processor, see Fig. 2, is now modelled in a third variant with MLDesigner using logic elements/finite state machines (FSM). The specified CORDIC is validated in the context of the compass model by simulation. This validated specification is the basis for an implementation on a Cylone II EP2C35 FPGA, or in other words, the validated MLDesigner model is input for the Quartus II Web edition, the FPGA development environment of the company Altera.

V. Implementation

For the implementation in hardware different design variants are possible. These designs differ regarding the execution time, hardware costs (number of logic cells) and the principle suitability for the implementation on a FPGA.

1) **Bit parallel iterative CORDIC processor**

Each of the three to be solved functions x_n, y_n and z_n are directly implemented in hardware, see Fig. 5. The shown processor represents an iterative CORDIC structure.

The function for the determination of the direction of rotation d_i determines itself in the rotation mode through $sign(z_i)$ and in the vector mode through $sign(y_i)$. At the beginning of the processing the initial values x_0, y_0 and z_0 are loaded into the registers. This is done via the upstream multiplexers. In one iteration step the values are loaded from the registers into the adders/subtracters and the shift registers. The results of the computations are then loaded again into the individual registers via the multiplexers. The angles for the computation of z_i are stored in successive addresses in a ROM. So one sufficient incremental access is enough. After processing of the last iterations the result of the respective variables can be read directly at the outputs (the adders/subtracters). In this design the arithmetic and the shift operations are implemented with data word length. This implementation does not lead to an efficient solution. The signals are passing a high number of FPGA cells. This leads to a slow design with a high number of logic cells.

2) **Bit serial iterative CORDIC processor**

A more compact method arises as a result of the use of bit serial arithmetic. A substantially higher clock than in the bit parallel variant is reached. This design consists of three bit serial adders/subtracters, three shift registers as well as one serial ROM. Thus it needs a minimum of hardware costs. The shift registers are in the length wise identical to the word length. For the initialization the data are loaded into the shift registers via the multiplexers. Each iteration

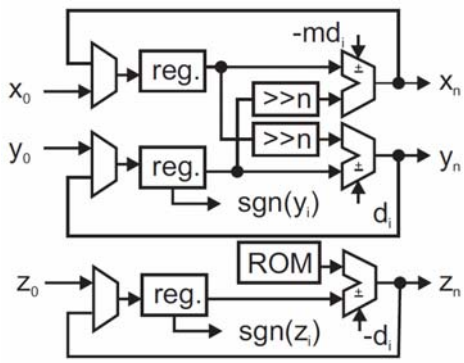

Fig. 5. Bit parallel iterative CORDIC processor.

needs exactly *w* clock cycles (*w* is the word length). The whole word is loaded into the adders/subtracters and afterwards it is pushed again into the shift registers. At the beginning of every step the sign of the variables is read and passed on to the adders/subtracters. After the last iteration the words are pushed to the outputs, at the same time a new word can be initially loaded again already. The advantage of this design is in the simple and minimal hardware. This permits to work with a very high clock frequency, which is necessary for the high number of clock cycles.

3) **Bit parallel combinatorial CORDIC processor**

Beside the iterative design variants, where n iterations for the computation are needed, there are also other possibilities to implement the CORDIC. One of these possibilities, described in [13], is the so called *unrolled CORDIC processor*. The idea thereby is to implement the hardware for every individual iteration step. This design variant is shown in Fig. 6.

This structure has some advantages. Because each iteration uses their own elements, always the same operations are executed. So the shift registers are not required and can be hard wired, because the same number of shift operations always achieved. The same is valid for the angles. They are a constant for each iteration step. That means, there is no need to use memory and the values are likewise hard wired. So there is no need of registers at all. The structure reduces the number of adders/subtracters. But that purely combinatorial structure has also its disadvantages. The processing time is accordingly high due to the multiplicity of the elements. However this structure is faster than those of the iterative variants, because the time for initialization and setup and hold of the registers is completely omitted.

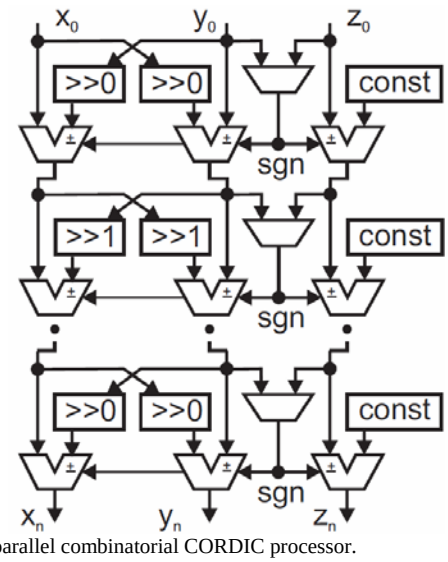

Fig. 6. Bit parallel combinatorial CORDIC processor.

4) **Pipeline structure**

The preceding structure is simply implementable as pipeline structure. Between the *add* and *sub* blocks pipeline registers are connected. So for a pipeline architecture only a few additional hardware costs are needed. The advantage is that with a filled pipeline after each clock a result can be read on the outputs.

Particularly the variants of bit serial iterative CORDIC processor and the pipeline structure were examined. Both MLDesigner models are transformed into the Quartus II model. The chip analyses show the following results:

	Bit serial iterative	Pipeline structure
Logic elements all	1469	2526
Logic elements CORDIC	887	1934
number register all	640	1477
number register CORDIC	121	958

In the compass implementation for the computation of the arc tangent the pipline structure was used. The clock diagram, see Fig. 7, shows for every clock the data at the input of a pipeline step. The steps 1, 2 and 5. are represented.

On the basis of the CORDIC pipeline structure the complete compass was implemented on a FPGA. The top level compass des ign is shown in Fig. 8.

The system is simulated/validated in the tool chain: MLDesigner, Quartus II Web edition and tested as a real system. As a test scenario a rotated house of the St. Nicholas is used, see Fig. 9. A real scenario could be a navigational task in a robot system [3].

When pacing down the house of the St. Nick the compass continuously supplies the expected azimuth angles per clock, see Fig. 10. The implementation runs with 50 MHz and the individual blocks from figure 8 have the following number of

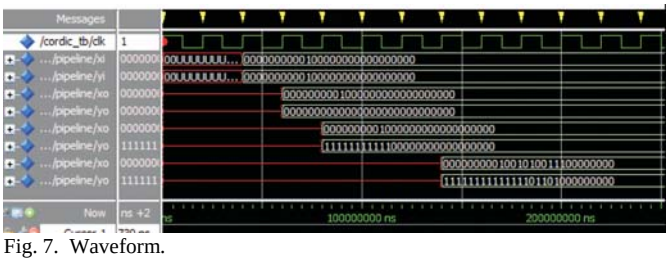

Fig. 7. Waveform.

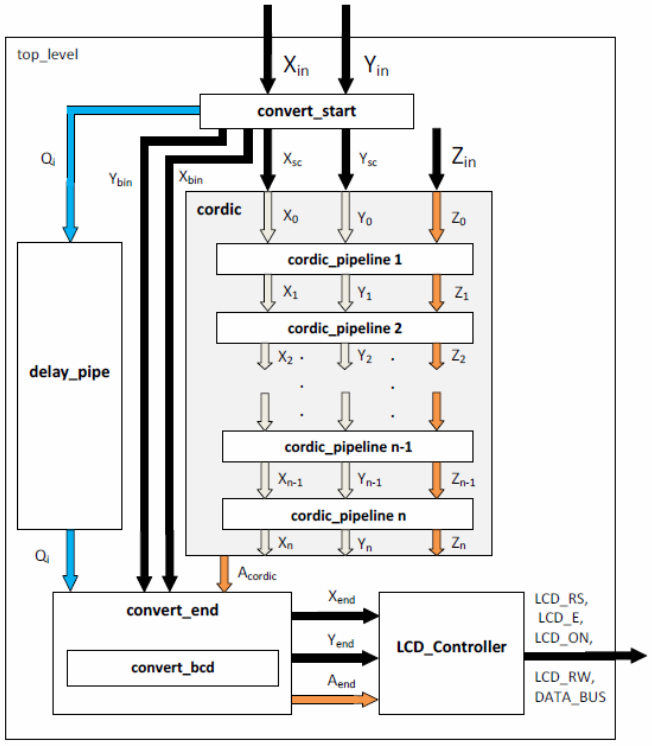

Fig. 8. Top-Level compass design.

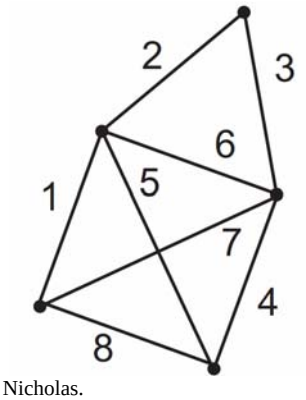

Fig. 9. House of St. Nicholas.

clock cycles:

- convert_start 3 clocks
- cordic_pipeline 16 clocks
- convert_end 2 clocks
- convert_bcd 6 clocks

VI. Conclusion

An electronic compass has been developed consequently model based. First an abstract system model was built using MLDesigner. So already in the early design phases performance analyses for system could be accomplished. In the special focus stands thereby the CORDIC algorithm. The model was stepwise refined and implemented by Quartus II on a FPGA. It was shown that the model based design using a tool chain allows extensive analyses and generates a fast, clear implementation.

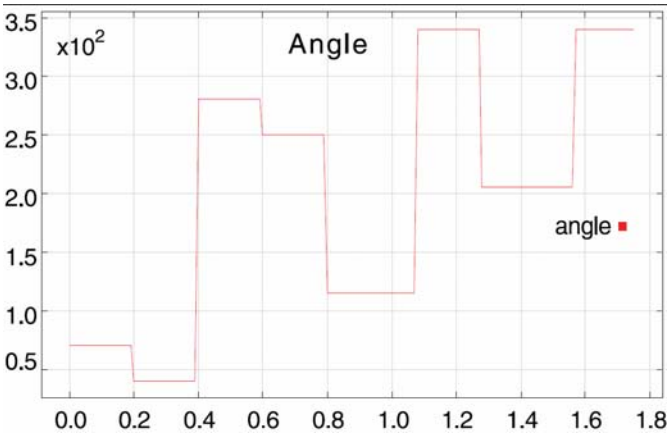

Fig. 10. Angle output for the house of St. Nick.

References

[1] E. A. Lee, D. G. Messerschmidt, Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE Trans. Comput. Vol C-36, no 1 pp. 24-35, Jan. 1987.

[2] K. D. Mueller Glaser, Systems Engineering in Microsystems Design. IFIP Workshop on Modelling of Microsystems. Stirling Scotland, 1997.

[3] M. Milushev, N. Krantov, V. Zerbe, Scalable Modular Control Architecture for Walking Machines. ICEST 2007, Ochrid, Macedonia, 24-27 June 2007, pp. 901-903.

[4] F. Lohse, V. Zerbe, Model based Performance Estimation in ZigBee based Wireless Sensornetworks. 27th MIEL 2010, IEEE International Conference on Microelectronics, Nis, Serbia, 16-19 Mai 2010, accepted.

[5] J. E. Volder, The cordic trigonometric technique. IRE Transactionon Electronic Computers, EC-8, pp 330-334, 1959.

[6] B. Andjelkovic, V. Litovski, V. Zerbe, A Mission Level Design Language Based on AleC++. MIEL 2006 - 25thIEEE International Conference on Microelectronics, Nis, Serbia, May 14-17 2006, pp 659-662.

[7] J. S. Walther, A unified algorithm for elementary functions.Spring Joint Computer Conf., 1971, pp. 379-385.

[8] K. Kota, J. R. Cavallaro, Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processors. IEEE Transactions on Computers, Vol. 42, NO. 7, 1993, pp. 769-779.

[9] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press, USA, 1999.

[10] http://www.mldesigner.

[11] A. de A. Agarwal, C.-D. Iskander, R. Shankar, G. Hamza-Lup. System-Level Modeling Environment: MLDesigner. SysCon 2008, IEEE International Sys-tems Conference, Montreal, Canada, April 7-10 2008.

[12] I. Paunovic, V. Zerbe. Modeling and Simulation of Digital Systems in different Domains. 3rd SSSS- Small Systems Simulation Symposium, Nis, Serbia, February 12-14 2010, p. 17-23.

[13] R. Andraka. A survey of cordic algorithms for fpga based computers. 6th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 1998, New York, USA, p. 191-200.