

University Bremerhaven

---

Course Documentation

## **Embedded System Design Example (Class/Lab) [ ES–SDEX ]**

- Part 1: Modeling
- Part 2: Programmable Logic
- Part 3: Processing System
- Part 4: Simple Motion Control
- Part 5: Continuous Motion
- Part 6: Experimental System

**Revision:** V1.1c (minor release)

**Release:** February 2020, © Univ. Bremerhaven

---

**Prof. Dr.-Ing. Kai Mueller**

University of Applied Sciences Bremerhaven  
Institute for Automation and Electrical Engineering  
An der Karlstadt 8



D–27568 Bremerhaven / Germany

Phone: +49 471 48 23 – 415

FAX: +49 471 48 23 – 555

Email: [kmuller@hs-bremerhaven.de](mailto:kmuller@hs-bremerhaven.de)

# I Contents

1	Preface .....	1
2	Modeling an Electrical Drive .....	5
2.1	Mechanical Subsystem .....	7
2.2	Electrical Subsystem .....	7
2.3	The KIS Principle (Keep It Simple) .....	8
2.4	Stepper Motor Sequence .....	8
2.5	Power Section .....	9
3	Programmable Logic .....	11
4	Processing System .....	15
5	Programmable Logic for Continuous Motion .....	16
5.1	Sine/Cosine Computation .....	17
6	Experimental System .....	20
7	Bibliography .....	22

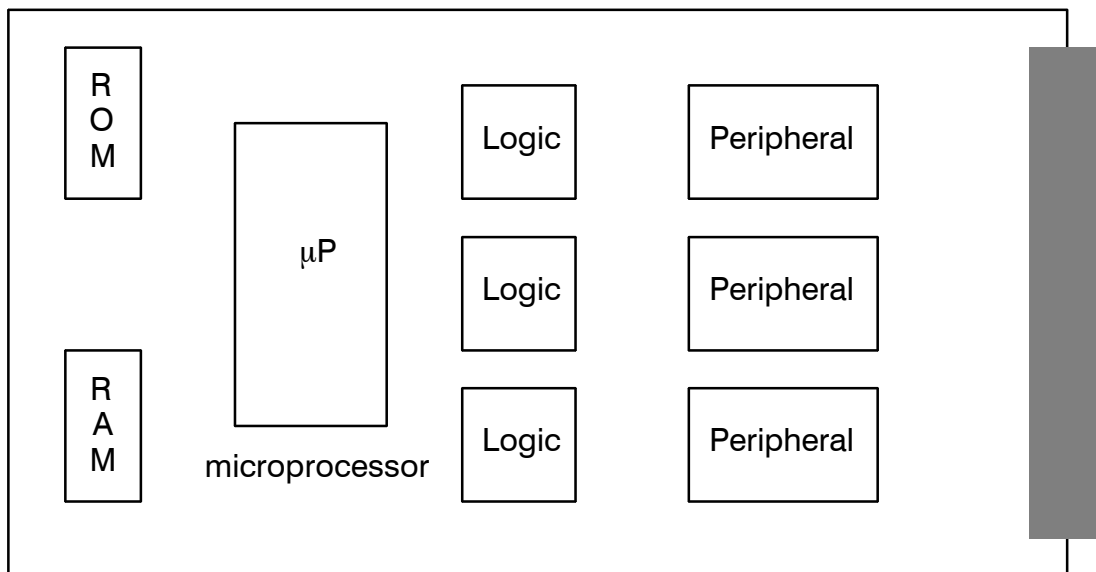
# Embedded System Design Example

## 1 Preface

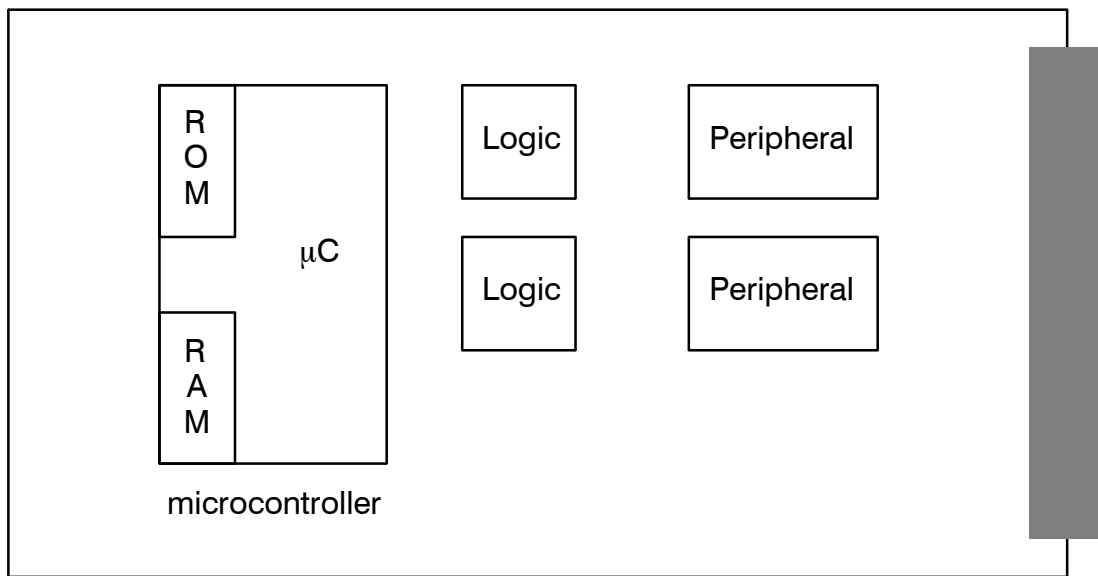
This design example demonstrates how to create an embedded system with modern DSP solutions in programmable logic and a processing system.

Electrical drives are often essential parts of embedded ISM devices. In this example we will design a digital subsystem to control a small drive which can be used in a medical infusion pump.

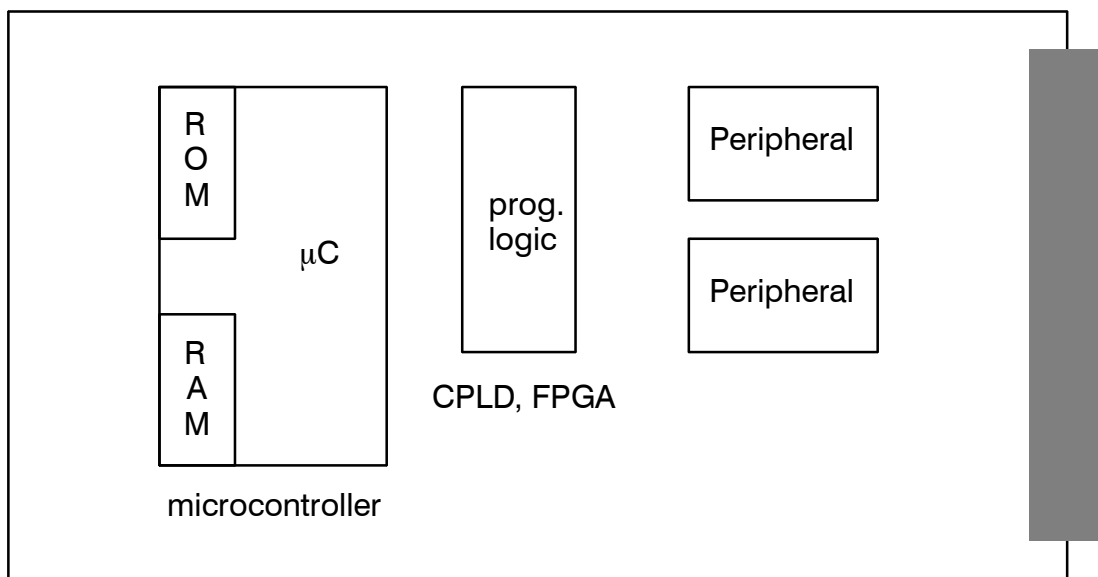
Fig. 1.1-1.4 show the historic development of embedded systems in the industrial/scientific/medical area (ISM market).



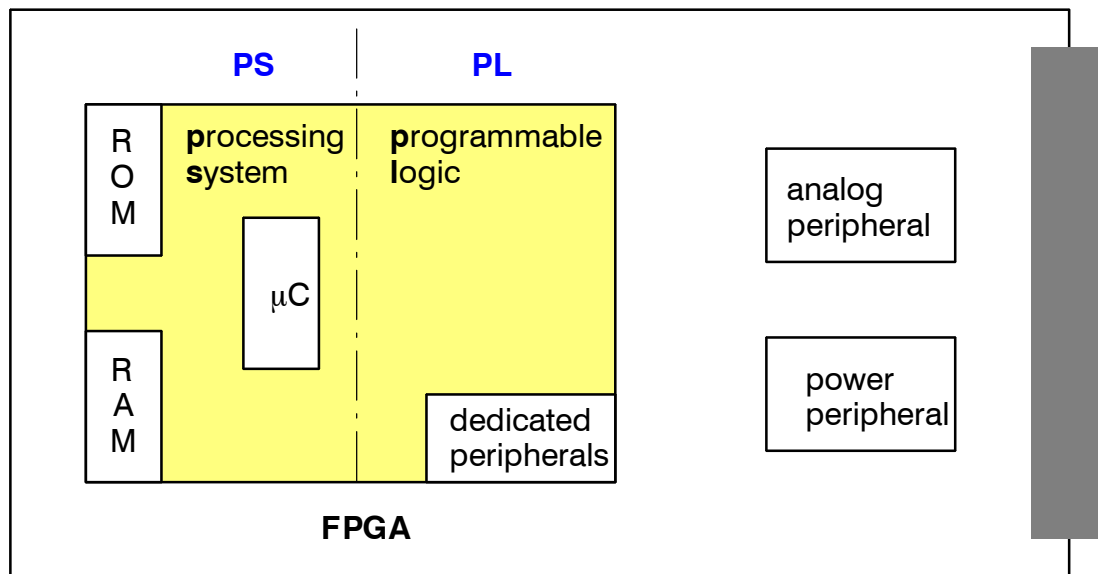
**Figure 1.1:** Single board computer with microprocessor



**Figure 1.2:** Single board computer with microcontroller



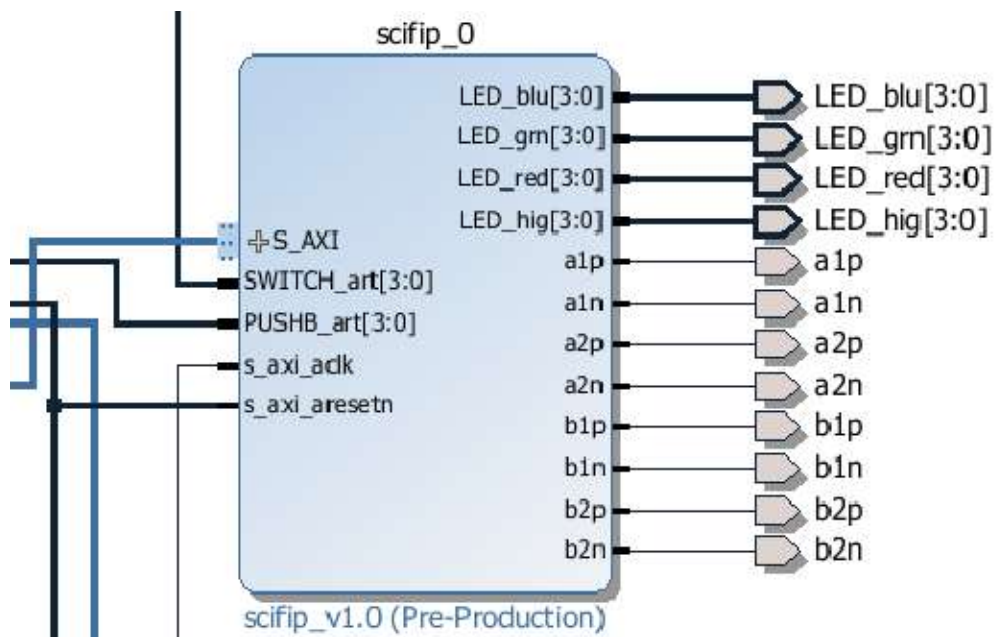
**Figure 1.3:** Single board computer with microcontroller and programmable logic



**Figure 1.4:** SoC structure (System-on-Chip)

Hardware logic and processing system integration for the motor application as a single chip solution is shown below (this was one on Artix-7 FPGA).

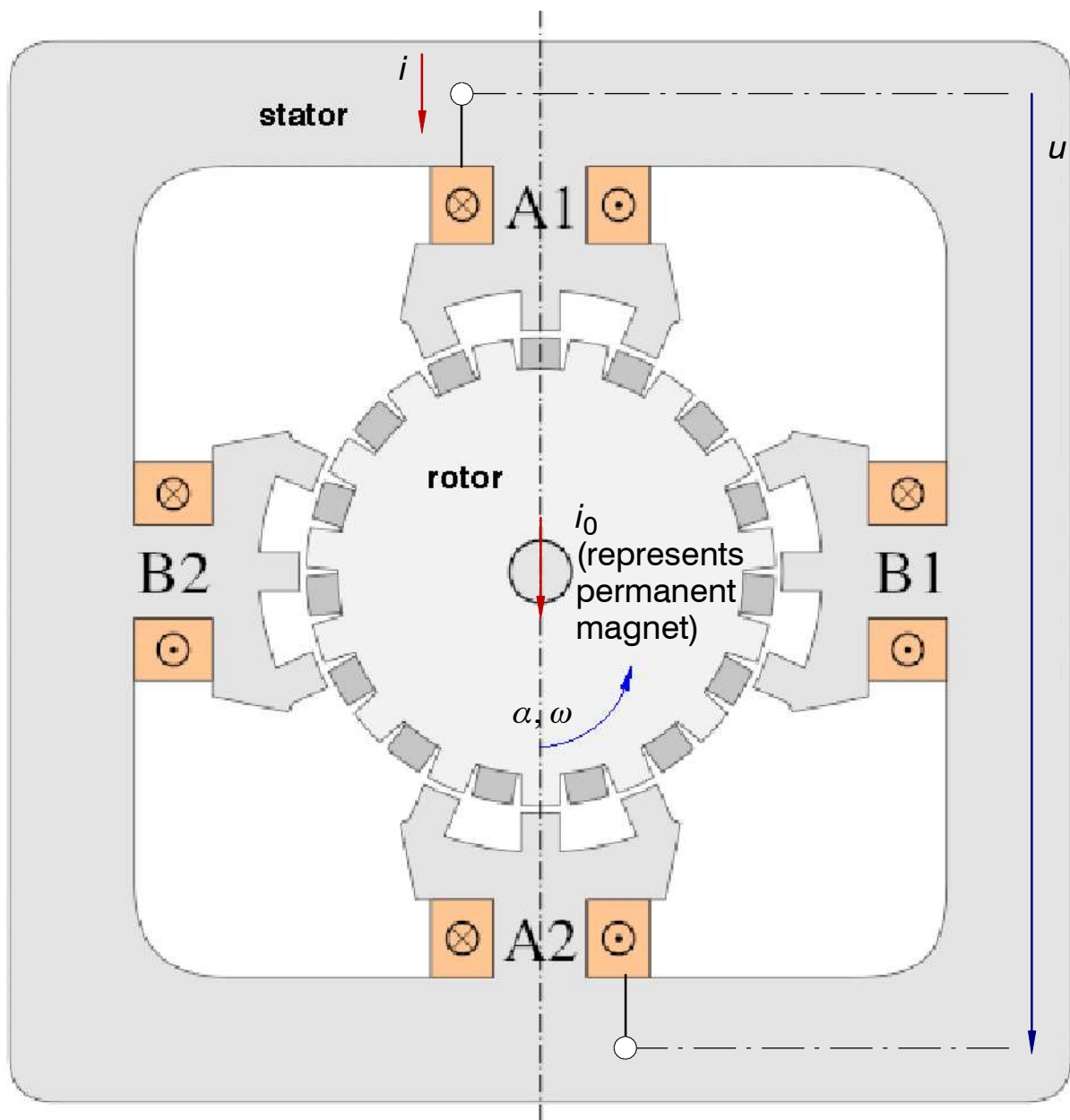




**Figure 1.6:** Detailed hardware IP component for motor control

## 2 Modeling an Electrical Drive

Whenever high torque and maintenance-free operation is required a stepper type motor is a possible choice. Fig. 1.7 shows the principle of operation.



**Figure 1.7:** Stepper motor schematic ( $n = 15$ ), 2-phase motor

Based on the physical principle that the integral over kinetic energy and the work of external forces always is an extremum we can derive the equations of motion.

An electrical drive converts magnetic energy to mechanical energy. This happens whenever the magnetic energy depends on the rotation angle  $\alpha$ . The magnetic energy (more exact the so-called magnetic coenergy) is given by

$$W_m^i = (k_0 + k_1 \cos(n\alpha)) \frac{(i + i_0)^2}{2}. \quad (1.1)$$



## 2.1 Mechanical Subsystem

The kinetic energy is

$$V = \frac{1}{2}J\omega^2 . \quad (1.2)$$

The sum of these energies is the so-called *extended Lagrange energy* function

$$L^{ex} = W_m^i + V = [k_0 + k_1 \cos(n\alpha)] \frac{(i + i_0)^2}{2} + \frac{1}{2}J\omega^2 . \quad (1.3)$$

The equation of motion follows from the well-known Euler-Lagrange differential equation

$$\frac{d}{dt} \left( \frac{\partial L^{ex}}{\partial \omega} \right) - \frac{\partial L^{ex}}{\partial \alpha} = 0 . \quad (1.4)$$

This results in the *mechanical equation of motion*

$$J \frac{d\omega}{dt} = - \underbrace{k_1 n \frac{(i + i_0)^2}{2} \sin(n\alpha)}_{\text{torque}} . \quad (1.5)$$

## 2.2 Electrical Subsystem

The equations for the electrical subsystem with respect to current  $i$  can be obtained in a very similar way as for mechanical systems

$$\frac{d}{dt} \left( \frac{\partial W_m^i}{\partial i} \right) + \frac{\partial}{\partial i} (P^C + P^i) = 0 . \quad (1.6)$$

which is the electrical equivalent to (1.4). The term  $P^C$  describes the current/voltage relation for capacitors which we don't have, so  $P^C = 0$ . The term  $P^i$  is the same for static terminals (resistor, voltage and current source) satisfying

$$\frac{\partial P^u}{\partial u} = i , \quad \frac{\partial P^i}{\partial i} = u , \quad P^u + P^i = ui . \quad (1.7)$$

Here  $P^u$  is not required but  $P^i$  needs to be computed. The terms are

$$P_R^u = \int_0^u i(u') du' = \int_0^u \frac{u'}{R} du' = \frac{u^2}{2R} , \quad [\text{area under } i(u)] , \quad (1.8)$$

$$P_R^i = \int_0^i u(i') di' = \int_0^i Ri' di' = R \frac{i^2}{2} , \quad [\text{area under } u(i)] . \quad (1.9)$$

For linear resistors ( $u = Ri$ ) both values are identical. The unit is Watt but, of course, it is not the power dissipation in a resistor. According to Telegen's theorem the sum of all powers in an electrical network is always zero. From this it follows that the total  $P^i$  is

$$P^i = R \frac{i^2}{2} - ui \quad (1.10)$$

(second term negative since the voltage source delivers power). Similar to the mechanical subsystem the *electrical system* is given by

$$\frac{d}{dt} \left( \frac{\partial W_m^i}{\partial i} \right) + \frac{\partial P^i}{\partial i} = 0. \quad (1.11)$$

This results in a somewhat complicated ODE for the current  $i$ :

$$\left[ k_0 + k_1 \cos(n\alpha) \right] \frac{di}{dt} - nk_1 \omega (i + i_0) \sin(n\alpha) + Ri = u. \quad (1.12)$$

$\frac{di}{dt}$	$nk_1 \omega (i + i_0) \sin(n\alpha)$
induced voltage by change of current	induced voltage by rotational speed (back-emf)

## 2.3 The KIS Principle (Keep It Simple)

If the maximum performance is required we need to take advantage of the model information given by (1.5) and (1.12). It is common practise for engineers to simplify the complete model if we can restrict the mode of operation. If the drive operates at low speed than  $\omega$  is very small. At the same time the current will be constant for a period of time, so  $di/dt$  becomes zero. In this case (1.12) simplifies to

$$Ri = u, \quad (1.13)$$

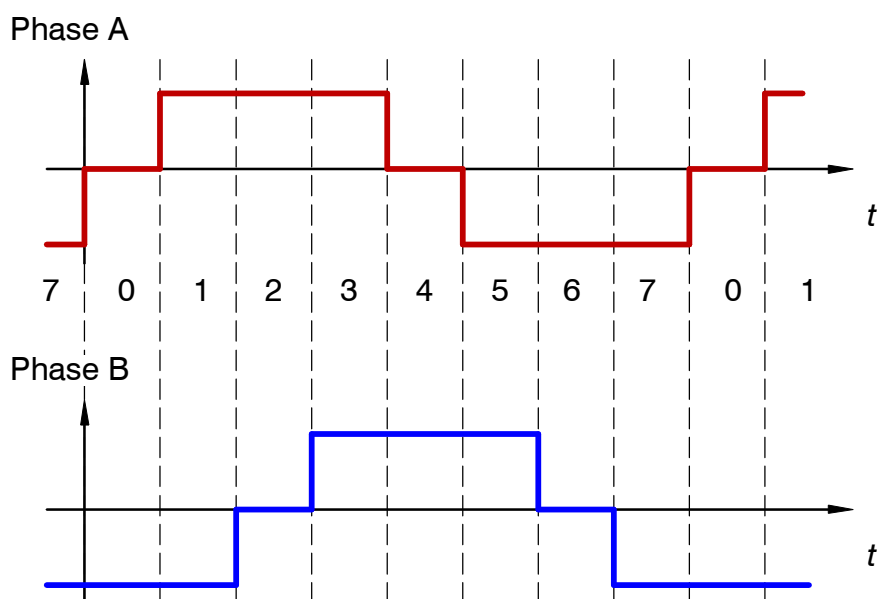
and therefore equation (1.5) results in

$$J \frac{d\omega}{dt} = - k_1 n \left( \frac{u^2}{2R^2} + i_0 \right) \sin(n\alpha). \quad (1.14)$$

The maximum torque is available for  $n\alpha = -90^\circ$ . The drive will perform a movement to  $n\alpha = 0^\circ$ , where the driving torque will be zero. This is called a *step* for the motor. If we switch to the second phase  $B$ , the same situation as for phase  $A$  occurs.

## 2.4 Stepper Motor Sequence

The following sequence of currents in the two phases result in a more or less discontinuous motion of the drive in one direction.

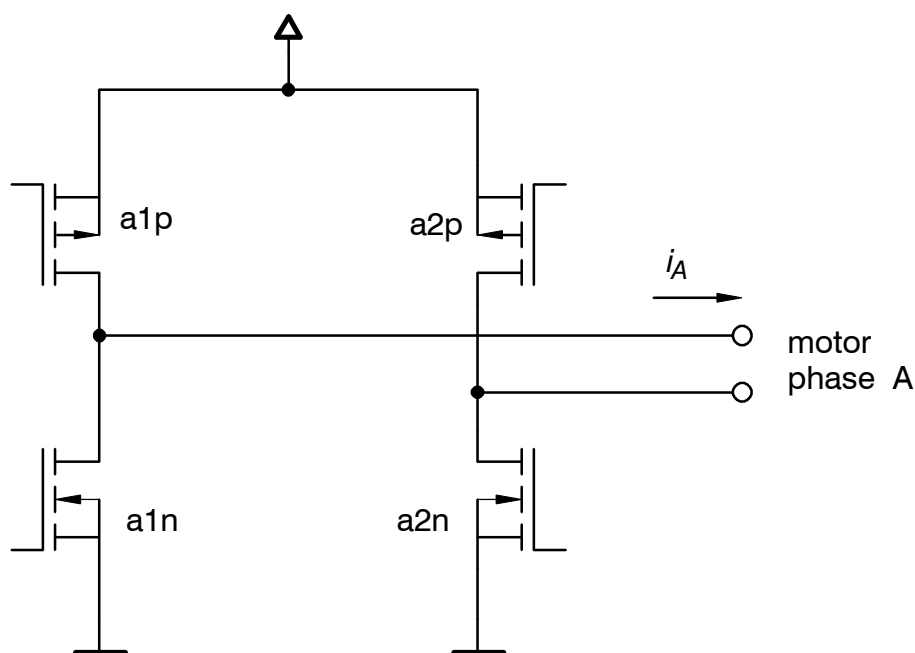


**Figure 1.8:** Sequence of motor currents for positive direction

Please note, that other schemes to control the motor exist. The benefit of this scheme is that the current requirements are low (for a mobile device) and that the so-called full step is divided into 8 sub-steps (0...7) for better position resolution.

## 2.5 Power Section

The digital current information requires amplification to drive the motor. For each phase the MOSFET bridge in fig. 1.9 drives the necessary currents.



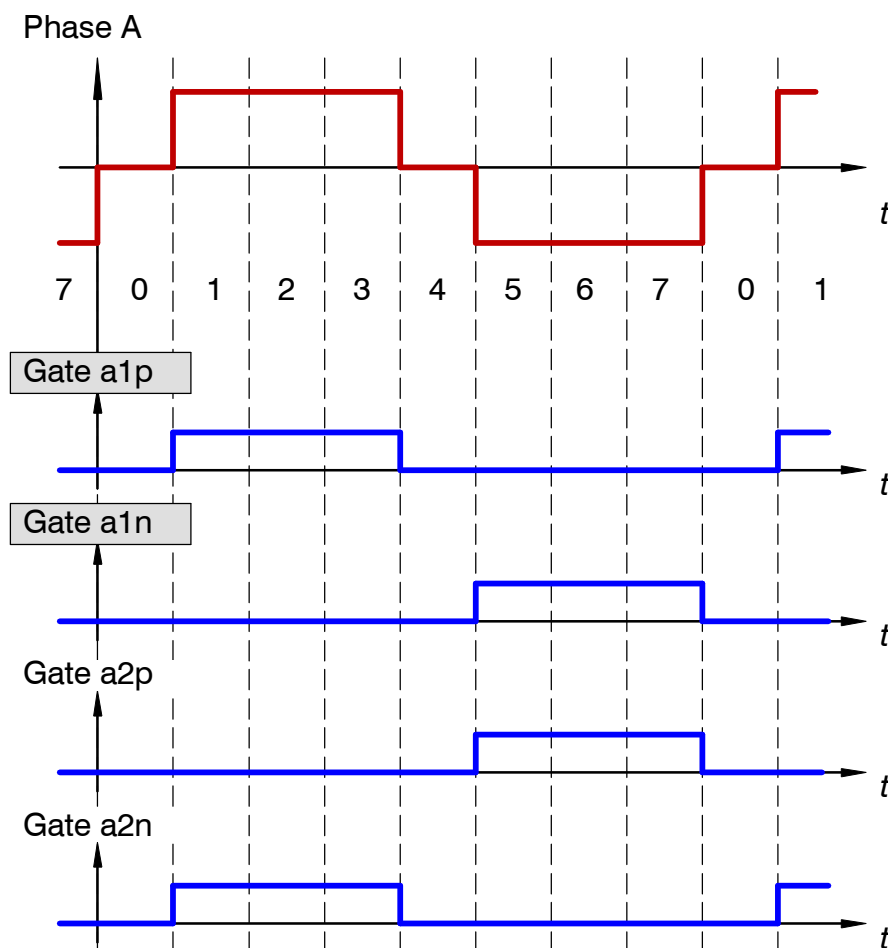
**Figure 1.9:** MOSFET power section

The same power section is used for phase B of the motor.

The gate signals of the power MOSFETs are the outputs of a digital system suitable for programmable logic. The benefits of a programmable logic solution are:

- fast and nanosecond precision
- reliable, safe with respect to software errors
- will result in much simpler software
- easy to synchronize many drives

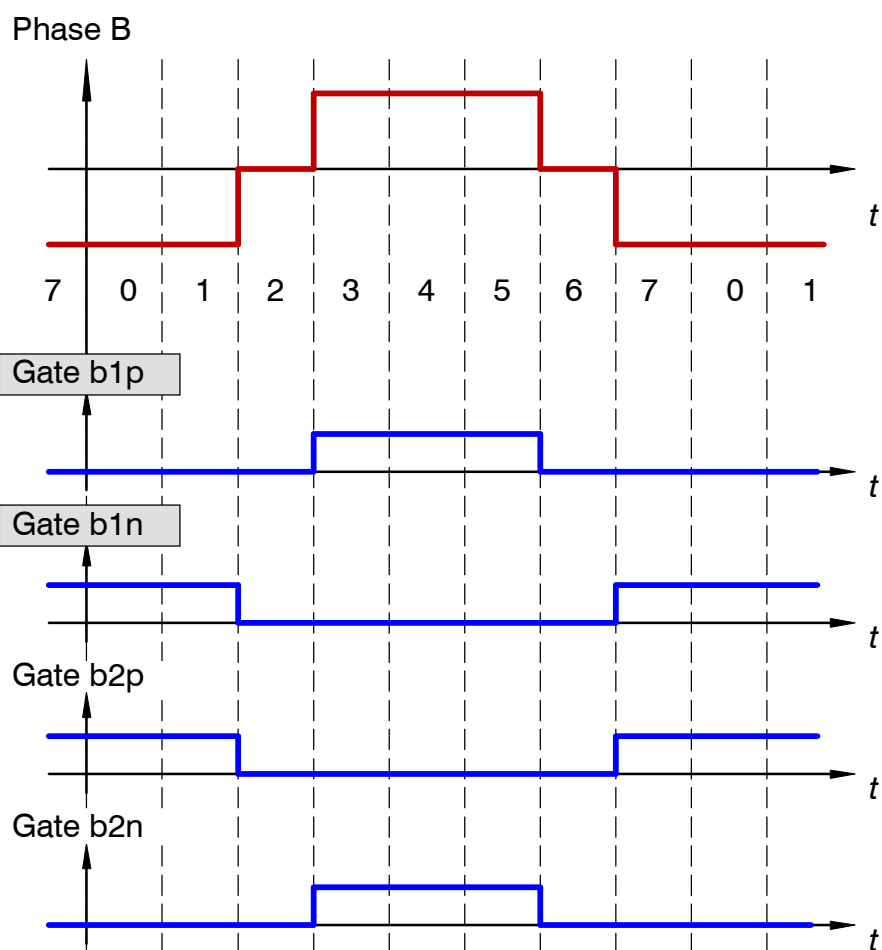
For one phase the transistor gate signals need to be selected according to fig. 1.10.



**Figure 1.10:** Gate signals for phase A

It is obvious that  $a2n = a1p$  and  $a2p = a1n$ , so only two signals need to be generated per phase.

For Phase B the situation is outlined in fig. 1.11.

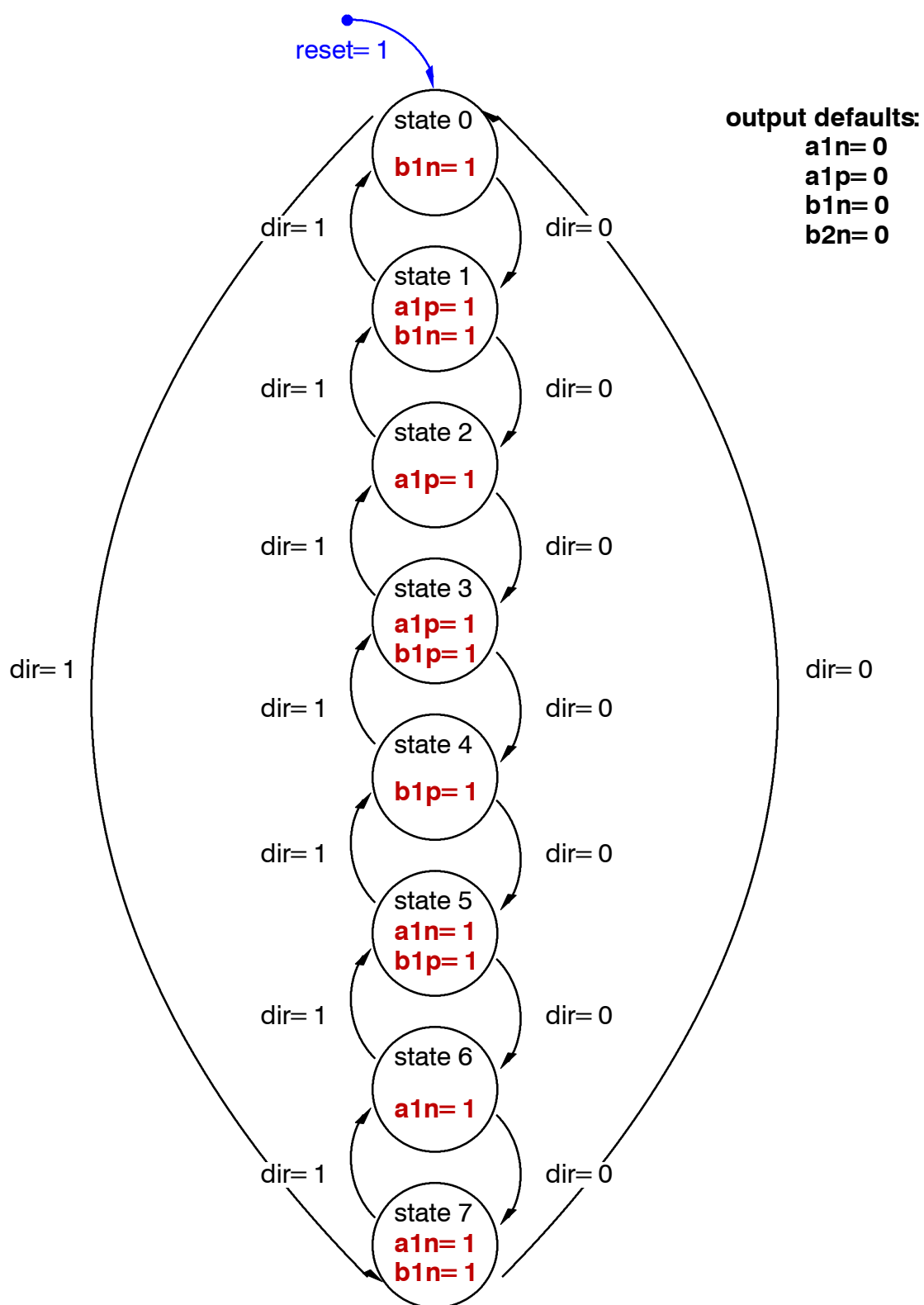


**Figure 1.11:** Gate signals for phase B

Since  $b2n = b1p$  and  $b2p = b1n$  we only need to generate  $b1p$  and  $b1n$ .

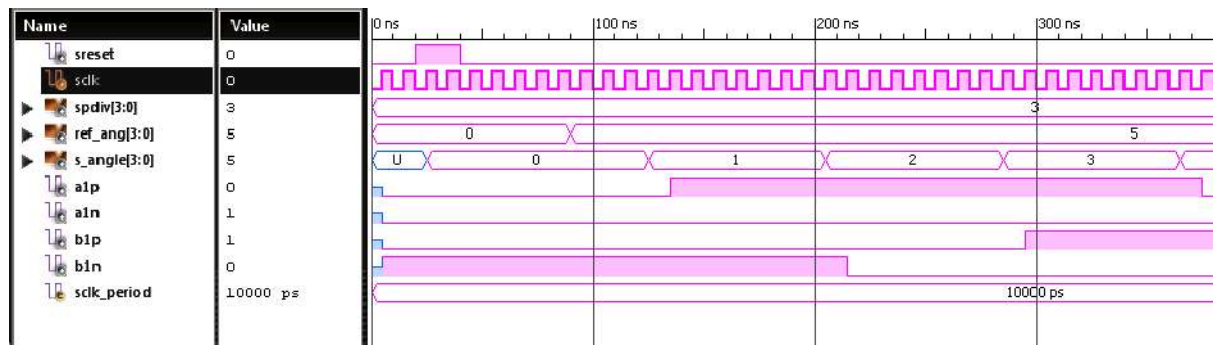
### 3 Programmable Logic

The programmable logic consists of a (timed) state machine where the time between the steps determine the motor speed. The state diagram fig. 1.12 follows from the gate signal plots.



**Figure 1.12:** Finite state-machine (FSM) for MOSFET signals (two phases)

The short sequence of the state machine simulation is shown in fig. 1.13.



**Figure 1.13:** State machine (Moore machine) simulation

The VHDL (Very High Speed Hardware Description Language) realization of the state machine is shown below. For the reason of simplicity only the combinational block is shown.

```

comb_fsm: PROCESS(state, ref_ang, sangle_cnt, a_diff)
BEGIN

    next_state <= state;
    a_diff <= ref_ang - sangle_cnt;

    alpi <= '0';
    alni <= '0';
    blpi <= '0';
    blni <= '0';

    CASE state IS
        WHEN st0 =>
            blni <= '1';
            IF a_diff(STC_BITS-1)='0' THEN
                next_state <= st1;
            ELSE
                next_state <= st7;
            END IF;
        WHEN st1 =>
            alpi <= '1';
            blni <= '1';
            IF a_diff(STC_BITS-1)='0' THEN
                next_state <= st2;
            ELSE
                next_state <= st0;
            END IF;
        WHEN st2 =>
            alpi <= '1';
            IF a_diff(STC_BITS-1)='0' THEN
                next_state <= st3;
            ELSE

```

```
        next_state <= st1;
    END IF;
WHEN st3 =>
    alpi <= '1';
    blpi <= '1';
    IF a_diff(STC_BITS-1)='0' THEN
        next_state <= st4;
    ELSE
        next_state <= st2;
    END IF;
WHEN st4 =>
    blpi <= '1';
    IF a_diff(STC_BITS-1)='0' THEN
        next_state <= st5;
    ELSE
        next_state <= st3;
    END IF;
WHEN st5 =>
    alni <= '1';
    blpi <= '1';
    IF a_diff(STC_BITS-1)='0' THEN
        next_state <= st6;
    ELSE
        next_state <= st4;
    END IF;
WHEN st6 =>
    alni <= '1';
    IF a_diff(STC_BITS-1)='0' THEN
        next_state <= st7;
    ELSE
        next_state <= st5;
    END IF;
WHEN st7 =>
    alni <= '1';
    blni <= '1';
    IF a_diff(STC_BITS-1)='0' THEN
        next_state <= st0;
    ELSE
        next_state <= st6;
    END IF;
END CASE;

END PROCESS comb_fsm;
```

The hardware is attached to the microprocessor (PS) by a 32 bit AXI-lite bus component.



## 4 Processing System

The software executes on the 32 bit microcontroller MicroBlaze™ which is part of the Xilinx EDK (Embedded Design Kit). Since the hardware does all logic the software becomes very simple. The AXI-bus component provides two registers (speed and position). Therefore, the software requires only two instructions to specify **speed**

```
CI_REG(SPDIV_REG_WR) = x_data;
```

and **position**

```
CI_REG(PREF_REG_WR) = x_data;
```

Here, CI\_REG is the C macro

```
#define CI_REG(regc) (*(volatile u32 *)  
                    (XPAR_SCIF_0_BASEADDR+4*regc))
```

to access memory mapped IO registers.

The rest of the software is required to perform user IO for monitoring the drive.

The communication is done by USB connection.

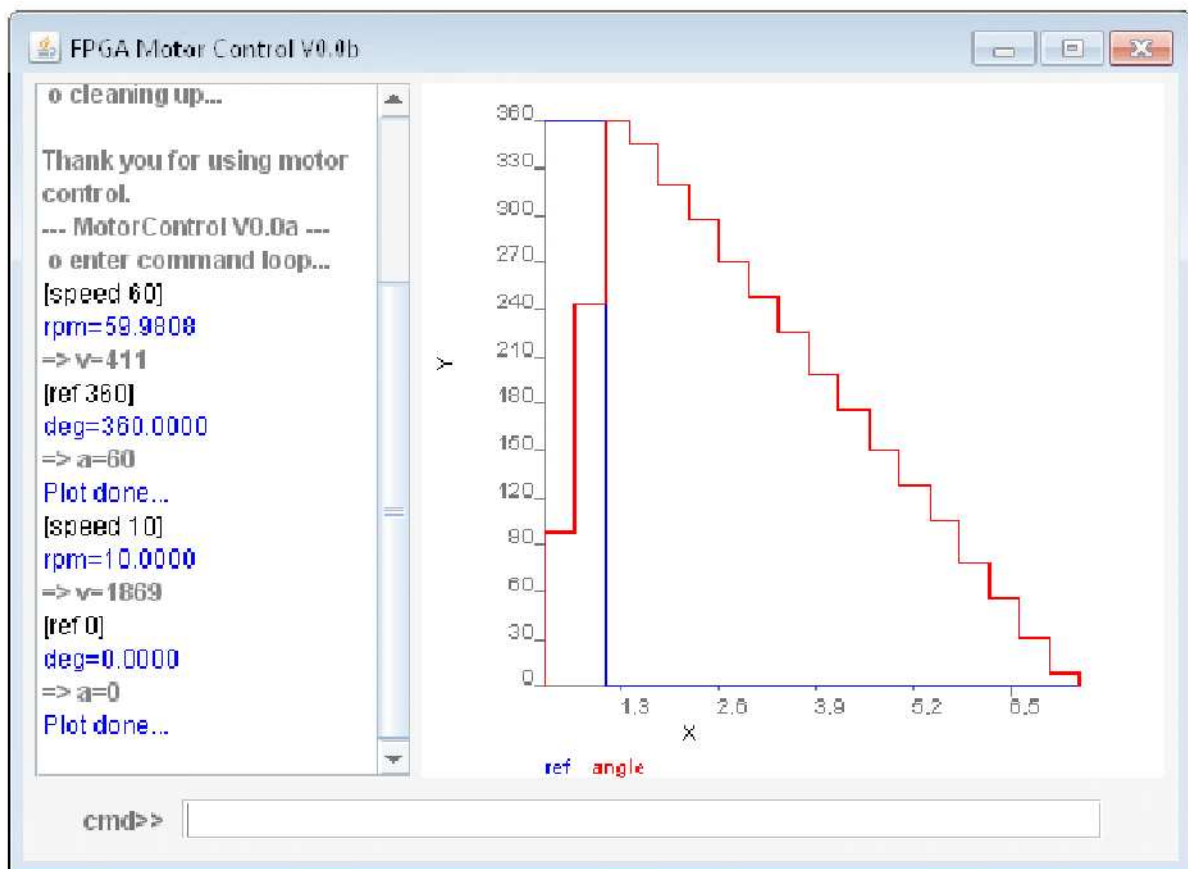
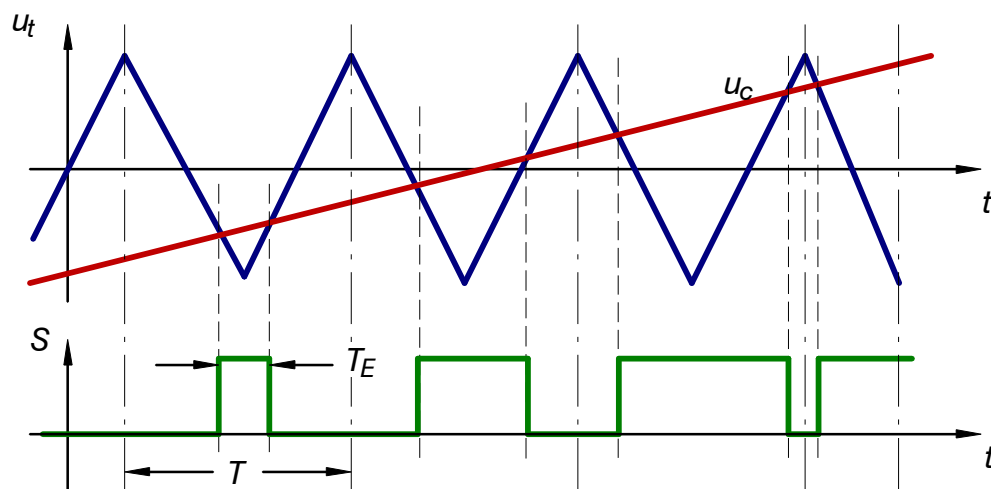


Figure 1.14: Motor control user interface

## 5 Programmable Logic for Continuous Motion

For some application the step type movement of the motor is not appropriate. Continuous motion requires sine/cosine currents in the two phases of the motor. This mode is closely related to synchronous servo drives – the highest performance drives in industry.

The power transistors are used in switch mode, i.e. the transistors are switched completely on and completely off. Only the mean value of the output voltage can take any value. This is accomplished by PWM (pulse width modulation).



**Figure 1.15:** Pulse width modulation (PWM)

The PWM compares a continuous function  $u_c$  with a triangular signal  $u_t$ . Since  $u_t$  is piecewise linear over time the output signal  $S$  (a binary signal!) has a mean value which corresponds to  $u_c$  by

$$\bar{S} = \frac{T_E}{T} = \frac{u_c + 1}{2}. \quad (1.15)$$

The input signal for phase  $A$  is cosine and for phase  $B$  the input signal is sine. Taking into account left and right half bridges we need four modulators. All modulators are synchronized to the same triangular signal. This signal is created by an up-down 12 bit counter. The hardware solution for the up-down counter is shown below.

```

udctr: PROCESS(sclk, clkdiv_p, reset, pwmcnt, din_0, din_1)
BEGIN
  if sclk'event AND sclk='1' THEN
    IF reset='1' THEN
      pwmcnt <= (OTHERS => '0');
      up_dwn <= '0';
      hithr_a <= (OTHERS => '0');
      lothr_a <= (OTHERS => '0');
      hithr_b <= (OTHERS => '0');
    
```

```

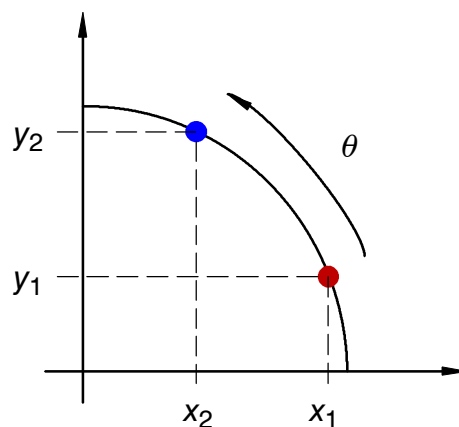
        lothr_b <= (OTHERS => '0');
ELSIF clkdiv_p='1' THEN
    IF up_dwn='0' THEN
        pwmcnt <= pwmcnt + 1;
        IF pwmcnt=UD_COUNT_MAX THEN
            up_dwn <= '1';
        END IF;
    ELSE
        pwmcnt <= pwmcnt - 1;
        IF pwmcnt=UD_COUNT_MIN THEN
            up_dwn <= '0';
            hithr_a <= din_0;
            lothr_a <= -din_0;
            hithr_b <= din_1;
            lothr_b <= -din_1;
        END IF;
    END IF;
END IF;
END IF;
END PROCESS udctr;

```

## 5.1 Sine/Cosine Computation

If mathematical functions need to be computed at high speed (nanoseconds) CORDIC (**C**oordinate **R**otation **D**igital Computer) provides a powerful algorithm for this purpose. Calculating sine and cosine is equivalent to the rotation of a unit vector in the cartesian plane.

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}. \quad (1.16)$$



**Figure 1.16:** Rotation of a vector in the cartesian plane by an angle  $\theta$

If we rotate the vector  $(x_1, y_1) = (1, 0)$  by  $\theta$  then  $x_2 = \cos(\theta)$  and  $y_2 = \sin(\theta)$ . Unfortunately this rotation is not suited for fast DSP. In 1959 Jack E. Volder discovered the CORDIC algorithm, the modified vector rotation

$$x_2 = x_1 \cos \theta - y_1 \sin \theta = \cos \theta (x_1 - y_1 \tan \theta), \tag{1.17}$$

$$y_2 = x_1 \sin \theta + y_1 \cos \theta = \cos \theta (y_1 + x_1 \tan \theta). \tag{1.18}$$

Dropping the  $\cos \theta$  term lead to the *pseudo-rotation*

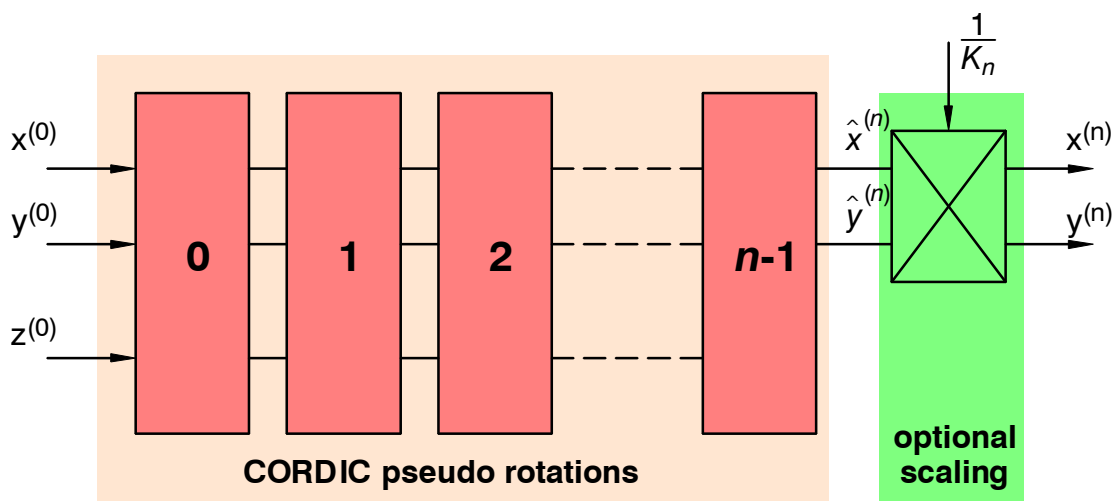
$$\hat{x}_2 = x_1 - y_1 \tan \theta, \tag{1.19}$$

$$\hat{y}_2 = y_1 + x_1 \tan \theta. \tag{1.20}$$

The error by  $\cos \theta$  can be easily corrected later. If we restrict  $\tan \theta$  to be powers of 2 than the algorithm requires only add, subtract and binary shift operations.

$i$	$\tan \theta^i$	$\theta^i$ (degrees)
0	1	45.0
1	0.5	26.565
2	0.25	14.036
3	0.125	7.125
4	0.0625	3.576
5	.	.
.	.	.

Any angle rotation between  $\pm 90^\circ$  can be achieved by a sequence of so-called microrotations. The number of stages  $n$  ( $0 \leq i \leq n-1$ ) determines the precision. The angles  $\theta^i$  are constants, it is not necessary to compute them on-line.



**Figure 1.17:** CORDIC hardware/software structure

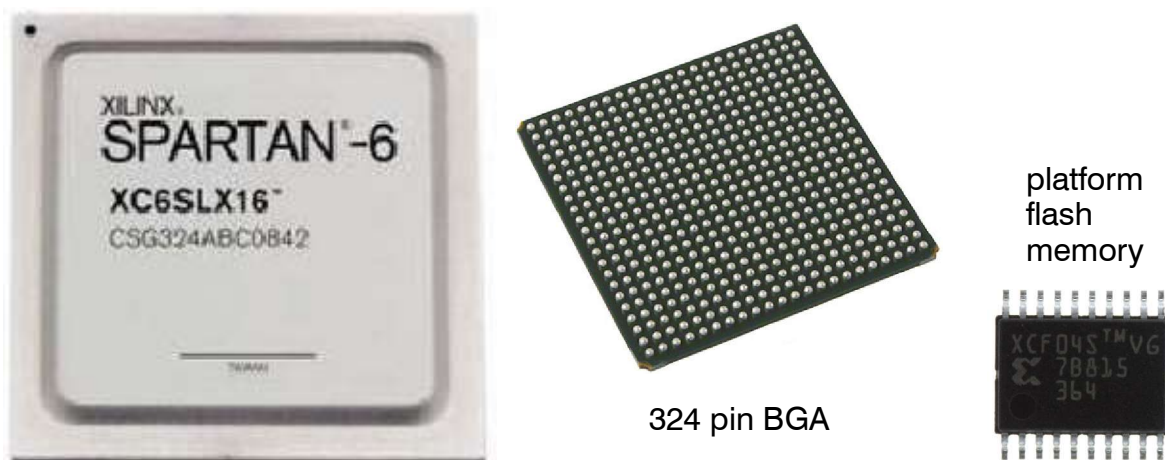
The following function shows the CORDIC algorithm for a 16 bit sine/cosine computation. The function has been optimized for fixed point computation, an angle of  $360^\circ$  corresponds to  $2^{15} = 32,768$ .

```
static void CordicSinCos(s16 z_ang, s16 *x_cos, s16 *y_sin)
{
    s16  x_in, y_in, z_in, x_out, y_out, z_out, reduct;
    int  k;

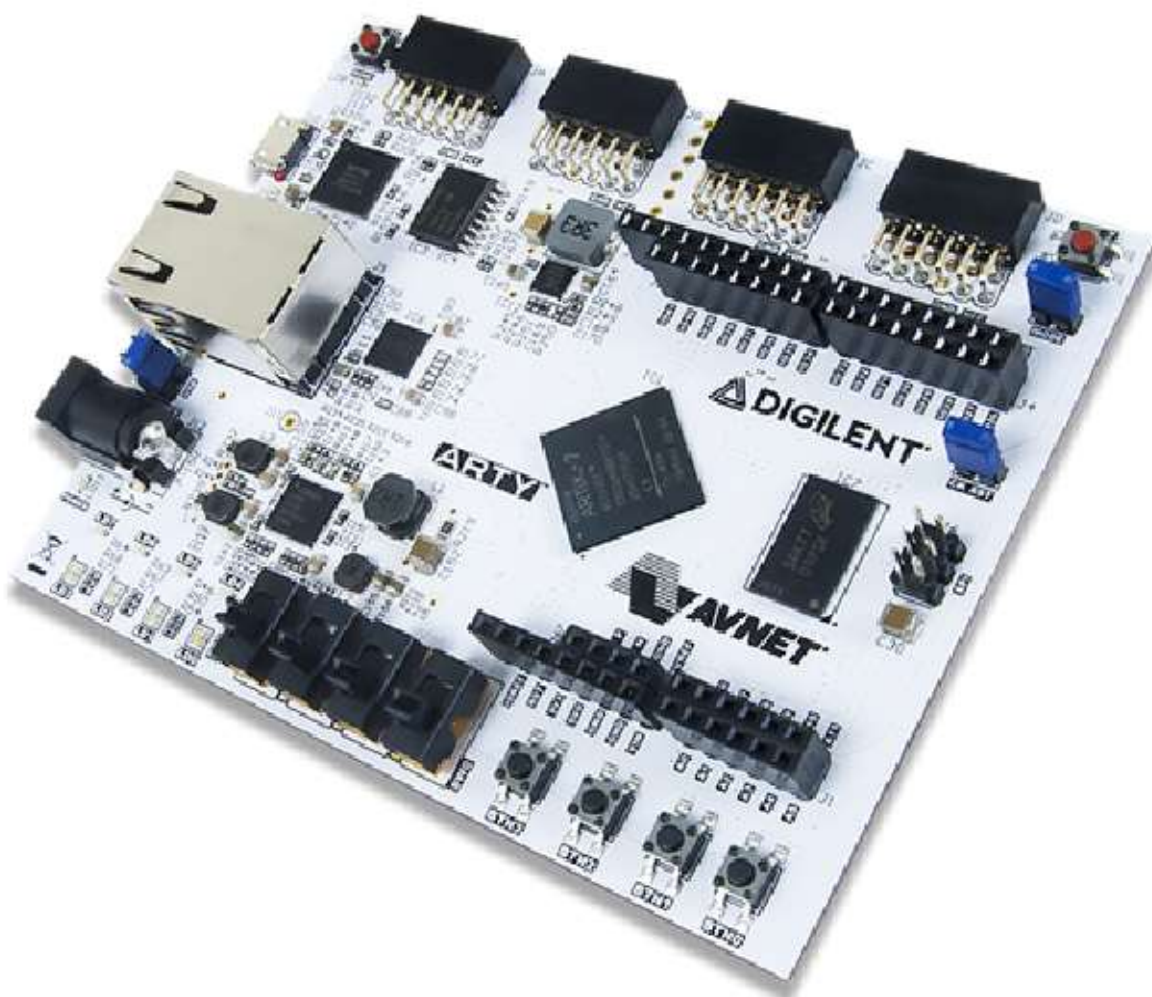
    x_in = 9900;
    y_in = 0;
    reduct = (z_ang >> 13) & 0x03;
    if (reduct == 1) {
        z_ang -= 16384;
    } else if (reduct == 2) {
        z_ang += 16384;
    }
    z_in = z_ang;
    for (k = 0; k < CORDIC_STAGES; k++) {
        if (z_in >= 0) {
            x_out = x_in - (y_in >> k);
            y_out = y_in + (x_in >> k);
            z_out = z_in - ZValues[k];
        } else {
            x_out = x_in + (y_in >> k);
            y_out = y_in - (x_in >> k);
            z_out = z_in + ZValues[k];
        }
        x_in = x_out;
        y_in = y_out;
        z_in = z_out;
    }
    if (reduct == 1) {
        *x_cos = -x_out;
        *y_sin = -y_out;
    } else if (reduct == 2) {
        *x_cos = -x_out;
        *y_sin = -y_out;
    } else {
        *x_cos = x_out;
        *y_sin = y_out;
    }
}
```

The algorithm provides 14 bit precision with only add, subtract, and shift operations.

## 6 Experimental System



**Figure 1.18:** Required devices for DSP and control Spartan-6 (45nm) or Artix-7 (28nm)



**Figure 1.19:** Artix-7 experimental board (33,280 logic cells, 41,600 FFs, 90 DSP slices)



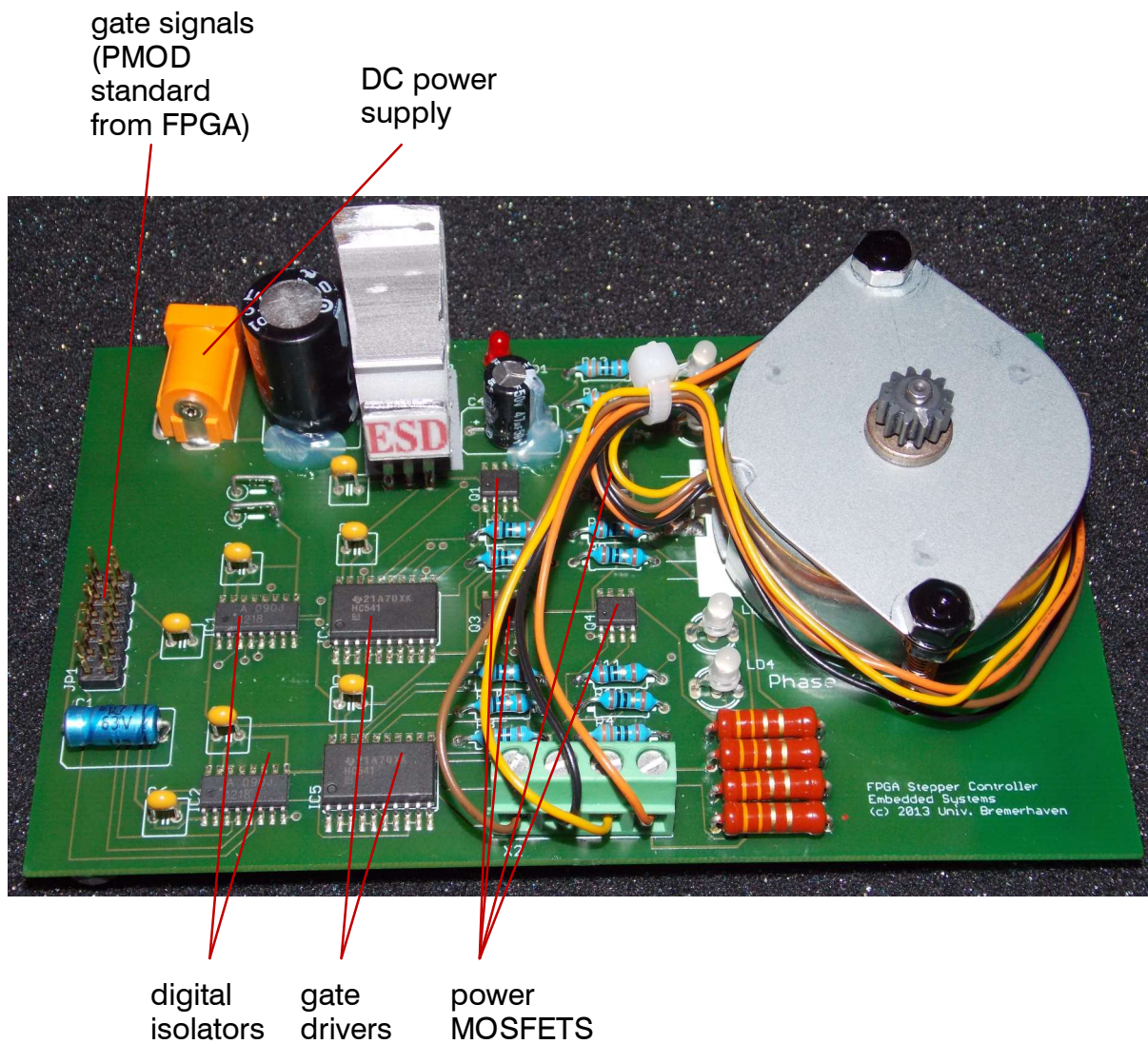


Figure 1.20: MOSFET power module and drive

\*\*\*\*

## 7 Bibliography

- [1] Ashenden, Peter J.: The Designer's Guide to VHDL, 3rd. Ed.  
Morgan Kaufmann, 2008
  
- [2] Al-Hashimi. Bashir M.: System-on-Chip: Next Generation Electronics.  
Institution of Electrical Engineers, 2006
  
- [3] Lipsett, Rger, Schaefer, Carls and Ussery Cary: VHDL Hardware Description and Design.  
Kluwer Academic 1990
  
- [4] Lyons, Richard G.: Understanding Digital Signal Processing.  
Prentice Hall, 2011
  
- [5] Predroni, Volnei A.: Circuit Design and Simulation with VHDL, 2nd. Ed.  
MIT Press, 2010
  
- [6] Prutchi, David and Norris, Michael: Design and Development of Medical Electronic Instrumentation.  
John Wiley & Sons, 2005
  
- [7] Reis, Ricardo, Lubaszewski, Marcelo and Jess, Jochen: Design of Systems on a Chip: Design and Test.  
Springer, 2010
  
- [8] Reichardt, J. und Schwarz, B.: VHDL-Synthese.  
Oldenbourg, 2001
  
- [9] Sass, Ron and Schmidt, Andrew G.: Embedded Systems Design with Platform FPGAs: Principles and Practices.  
Elsevier Inc. 2010
  
- [10] Wakerly, John F. : Digital Design, Principles & Practices.  
Prentice Hall, 2001

\*\*\*