CNC FOR EDM MACHINE TOOL – HARDWARE STRUCTURE

Ioan Lemeni

Computer and Communication Engineering Department
Faculty of Automation, Computers and Electronics
University of Craiova
13, A.I. Cuza, Craiova, Romania
Email: Ioan.Lemeni@comp-craiova.ro

Abstract: The main purpose of this paper is to present a cheap solution for retrofitting an EDM machine tool. Because the electronic blocks for such machines are manufacturer specific, it is not possible to mix blocks from different manufacturers. Therefor it is not possible to buy a single block, but whole kits are available on the market. Unfortunately this solution causes high costs. If, in such a structure, a single block must be replaced, is cheaper to design it again. In this paper it is present the hardware solution for the CNC block of the EDM machine tool ELEROFIL-10.

Key words: EDM, CNC, retrofit, linear encoder, interface, PALs, VHDL

1. INTRODUCTION

Wire EDM (Electrical Discharge Machining) is a method to cut conductive materials with a thin electrode that follows a programmed path. The electrode is a thin wire. Typical diameters range from 0.1-3.0 mm although smaller and larger diameters are available. The hardness of the work piece material has no detrimental effect on the cutting speed. There is no physical contact between the wire and the part being machined. Rather, the wire is charged to a voltage very rapidly. This wire is surrounded by de-ionized water. When the voltage reaches the correct level, a spark jumps the gap and melts a small portion of the work piece. The de-ionized water cools and flushes away the small particles from the gap.

In Romania, before 1989, a family of EDM machine tools has been produced. One member of this family was ELEROFIL-10, produced by "Electotimiş-Timişoara" company in 1989. A significant number of units have been installed in many Romanian factories and the results were satisfactory enough. Unfortunately, after 10 years of running, the reliability of these machines diminished drastically.

Instead of buying a new machine tool is worth to consider the retrofit solution. It has become common place in manufacturing industry to update the control technology on machine tools. The benefit of retrofitting depends of the size and type of the machine being retrofitted. On average, retrofitting a good machine is less than 50% of the cost of a new machine of the same size and type.

ELEROFIL 10 machine tool is composed of the following parts:

- Mechanical frame with two stepping motors and two linear encoders.
- Spark generator, ROGIF.
- Conventional and stepping motors power controller.
- CNC (**C**omputer **N**umerical **C**ontrol) NUMEROM 450. This CNC is in fact a CORAL minicomputer extended with several I/O interface cards.

In the last years 99% of the machine breakdowns was due to the CORAL minicomputer; this is the main motif for retrofitting. So, the cheapest way to retrofit ELEROFIL-10 is to replace the NUMEROM 450 with a new CNC. Unfortunately, there is no available CNC on the market to interface with the old spark generator, ROGIF, and with the stepping motors. A solution still exists: to buy a retrofit kit. Such a kit includes CNC, spark generator, linear encoders, motors and their drivers. This is the best solution from technical point of view, but its cost is too high: about $30K. We mention that a new machine tool of the same performance costs about $50K. Due to the relative high cost of the kit, the best solution, from economic point of view, is to design a CNC which interfaces with all existing equipment: ROGIF, linear encoders, stepping motors power controller and conventional. The cost of such a solution is only $5K.

This cheap solution is NUMERIC-01: a PC-based CNC designed as a direct substitute for the old CNC, NUMEROM-450. Is worth to mention that all necessary hardware was embedded in only two additional boards, the software has been rewritten so it emulates perfectly the original NUMEROM software and new graphic features have been added.

2. HARDWARE OVERVIEW

Nowadays a PC has more than enough processing power to carry out all the calculations involved by the interpolation algorithm and can be transformed into a CNC very easy if the appropriate interfaces are added. In ELEROFIL-10 case this is a simple task because most of the external signals are digital, so a general-purpose acquisition board, with an appropriate number of digital inputs and outputs, should be enough. But the acquisition boards available on the market don't deal well with many interrupt sources.

It would be very difficult, if not impossible, to write the software without interrupt support. That is way a specialized hardware interface was designed.

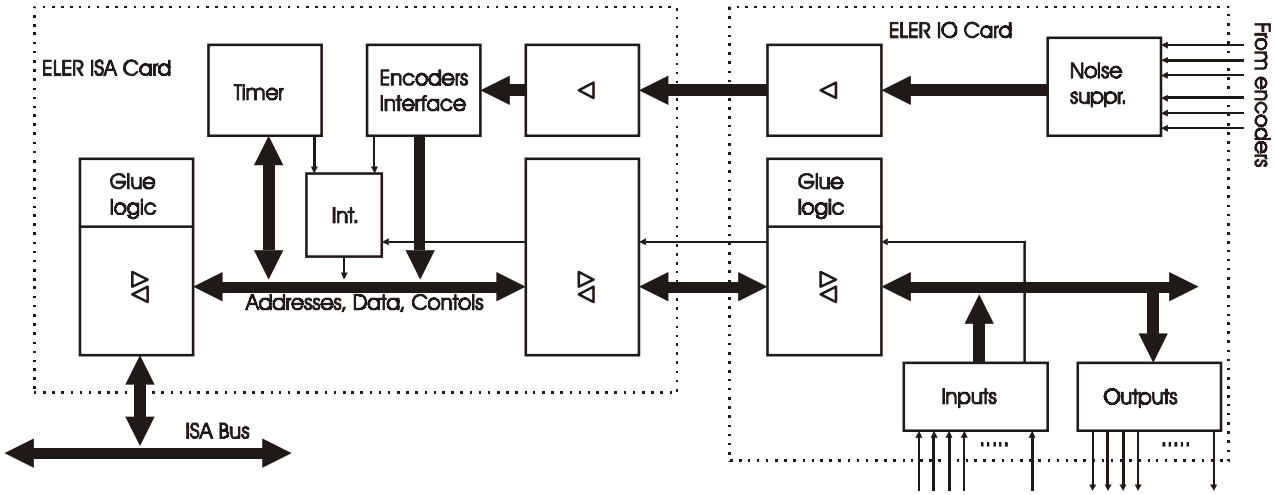

Figure 1

The hardware structure is one of an acquisition board to witch two extra blocks was added. The hardware block diagram is shown in figure 1. Because some external signals are in the 0-24V range, it is necessary to convert them to TTL, because galvanic isolation was mandatory, it wasn't enough room to embed all the hardware on a single ISA card. That is way the hardware implementation has been carried out on two cards. The ISA card insures the bus interface and defines a local bus used to connect any other block. This bus goes to the IO card via a flat ribbon. Due to the distance between the two cards transceivers and drivers are endowed on both sides.

The input signals are converted, through photocouplers, to TTL levels and then wired to input ports. The signal from the spark generator is forwarded directly to the interrupt block on the ISA card. The frequency of this signal encodes the mean voltage value at witch sparks occur. The output signals are latched in output ports, and then passed through photocouplers. After the photocoupler stage these signals are used to command the power drivers. The ISA bus interface, the local bus, the inputs block and the outputs block have nothing special and won't be further detailed. The original part of this paper concerns the processing of the linear encoders signals

3. Encoders Interface

ELEROFIL-10 is a machine tool with two axes. Each axis is endowed with a linear encoder. The encoder is of incremental type, with reference marks from centimeter to centimeter. The glass scale is mounted on the machine frame while the encoder head moves as the axis moves. The position is Gray coded on two bits (figure 2). When the head moves, for instance, from left to right, the sequence of B and A signals is 00, 01, 11, 10, and so on. When the head moves in opposite sense (right to left) the sequence is 00, 10, 11, 01. Any transition means a movement of exactly 1μm. The encoder is 1m long, so the position is in the 0-1,000,000 range.

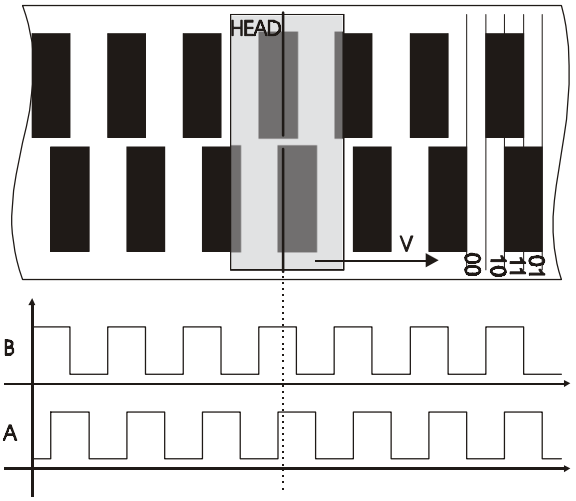

Figure 2

The CNC must know the current position in any moment. In order to keep track of the position, a 20 bits counter is necessary for each axis. This counter will increment or decrement on the transitions of B and A signals. On any axis the maximum speed is 1mm/s, so the maximum transitions frequency of either A or B is 1kHz.

These counters can be implemented hardware or software. The hardware solution implies too much circuitry: two counters on 20 bits each, six 3-state latches and an automaton to solve the exclusive access to the counters. A counter is updated on either B or A transition and is read when the software needs the position. A misread can occur if the counter is incremented or decremented while is being read.

Because the hardware solution for the counters means too many ICs, it was decided to implement them in software. The hardware structure used is shown in figure 3.

The edge detector (FSM1) has to detect any edge (rising or falling) of both B and A. If an edge has been detected, the ACTIVE signal becomes logic '1'. This signal is used to determine which axis has an active pulse. This block also computes the sense. The sense is logic '0' if the sequence of B and A is ..-> 00 -> 01 -> 11 -> 10->.. and logic '1' for the sequence ..-> 00 -> 10 -> 11 -> 01->. The SENSE signal is useful for the

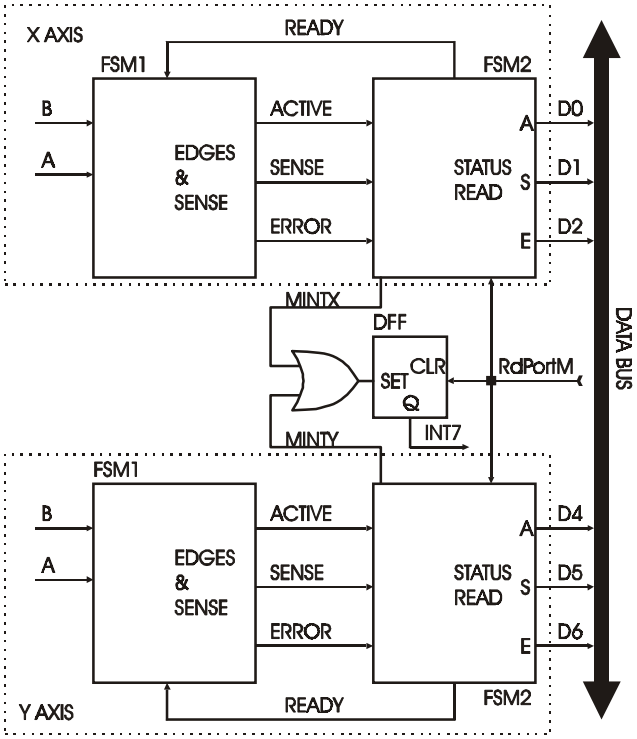

Figure 3

software routine: it tells to the software counter to increment or decrement.

Basically, the detector is implemented as a four states FSM. For each possible value of the B and A inputs there is a corresponding state: if B&A are 00, the machine is in state S00, if they are 01 the state will be S01 and so on. If the B&A inputs don't change, the state won't change, the ACTIVE output will be '0' and the SENSE won't care. When the inputs change, the state will change and ACTIVE will be '1', as in figure 4. The SENSE value was discussed before.

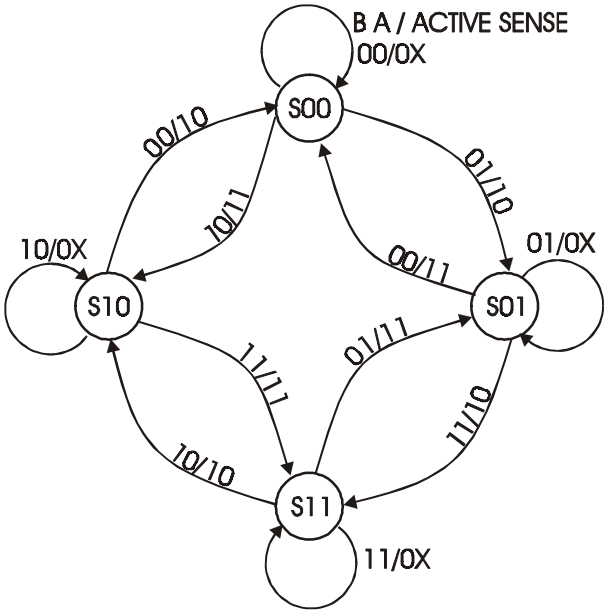

Figure 4

Actually, FSM1 is not implemented exactly as in figure 4. The transitions from one state to the other depend on the READY signal: when READY is inactive, the machine can't change its state. In figure 5 is presented only the S00 state of the modified machine:

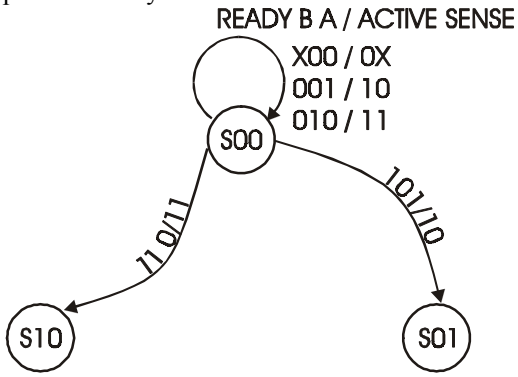

Figure 5

The role of READY signal will be discussed when FSM2 will be detailed.

Supplementary, this block also determines if an error has occurred. For example, the sequence 00->11 of BA is an error. This sequence never appears on the linear encoder. When an error has been detected the ERROR signal becomes logic '1'.

FSM1 was implemented with a PAL22V10 using OrCAD 9 environment. The VHDL code is given in the sequel:

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

ENTITY masura is
 PORT (
   BCLK         : IN STD_LOGIC; --ISA Clock
   BRST         : IN STD_LOGIC; --ISA Reset
   A,B, NUL     : IN STD_LOGIC;
   READY         : IN STD_LOGIC;

   ACTIV, SENS : OUT STD_LOGIC;
   ERR          : OUT STD_LOGIC;
   BNUL         : OUT STD_LOGIC;
   Abuf, Bbuf   : BUFFER STD_LOGIC;
   state        : BUFFER STD_LOGIC_VECTOR
                  (2 downto 0)
 );
END masura;

ARCHITECTURE behavior OF masura IS
 signal nextstate: STD_LOGIC_VECTOR(2 downto
0);
BEGIN

--synchronize A,B,NUL pulse
 process(BCLK)
 begin
   if BCLK='1' and BCLK'event then
     Abuf<=A; Bbuf<=B; BNUL<=NUL;
   end if;
 end process;

 -- sequential state machine, flipflops
 process (BCLK)   -- state machine, flipflops
 begin
   if BCLK='1' and BCLK'event then
     if BRST='0' then state<='0'&B&A;
     else state<=nextstate;
     end if;
  end if;
 end process;

 process (state, Abuf, Bbuf, READY)
 begin
   ACTIV <='0'; SENS <='0'; ERR<='0';
   case (state) is
    when "000" =>    --B=0, A=0
      nextstate <= "000";
      case (Bbuf & Abuf & READY) is
       when "000" | "001"=> null;
       when "010" => ACTIV <= '1';
       when "011" => ACTIV <= '1';
```

```
                    nextstate<="001";
      when "100" => ACTIV <= '1'; SENS <= '1';
      when "101" => ACTIV <= '1'; SENS <= '1';
                    nextstate <= "010";
      when others =>nextstate <= "100";
     end case;

    when "001" =>   --B=0, A=1
     nextstate <= "001";
     case (Bbuf & Abuf & READY) is
      when "010" | b"011"=>  null;
      when "110" => ACTIV <= '1';
      when "111" => ACTIV <= '1';
                    nextstate<="011";
      when "000" => ACTIV <= '1'; SENS <= '1';
      when "001" => ACTIV <= '1'; SENS <= '1';
                    nextstate <= "000";
      when others =>nextstate <= "100";
     end case;

    when "011" =>   --B=1, A=1
     nextstate <= "011";
     case (Bbuf & Abuf & READY) is
      when "110" | "111"=>  null;
      when "100" => ACTIV <= '1';
      when "101" => ACTIV <= '1';
                    nextstate<="010";
      when "010" => ACTIV <= '1'; SENS <= '1';
      when "011" => ACTIV <= '1'; SENS <= '1';
                    nextstate <= "001";
      when others =>nextstate <= "100";
     end case;

    when "010" =>   --B=1, A=0
     nextstate <= "010";
     case (Bbuf & Abuf & READY) is
      when "010" | "011"=>  null;
      when "000" => ACTIV <= '1';
      when "001" => ACTIV <= '1';
                   nextstate<="000";
      when "110" => ACTIV <= '1'; SENS <= '1';
      when "111" => ACTIV <= '1'; SENS <= '1';
           nextstate <= "011";
      when others =>  nextstate <= "100";
     end case;

--Error
    when "100" =>
     ERR<='1'; nextstate <= "100";
     if READY='0' then null;
     else
      case (Bbuf & Abuf ) is
       when "00" =>  nextstate <= "000";
       when "01" =>  nextstate <= "001";
       when "10" =>  nextstate <= "010";
       when "11" =>  nextstate <= "011";
       when others => nextstate <="100";
      end case;
     end if;
    when others => nextstate <= "100";

  end case; --status
 end process;
 END behavior;
```

The FSM2 block interfaces the Edge&Sense detector with the local bus. FSM2 interacts with FSM1 as follows:

1. A and B don't change, so FSM1 remains in the state corresponding to the A and B signals. ACTIV and ERR signals are logic '0'. FSM2 waits for either ACTIV or ERR to become logic '1'. READY output from FSM2 to FSM1 is inactive.
2. Either A or B changes. Because READY is inactive, FSM1 won't change its state, but ACTIV will become logic '1'.
3. FSM2 catches this event and latches ACTIV, SENS and ERR. This action is controlled by the WR signal. On the same clock pulse it activates READY, allowing FSM1 to change its state. It also activates MINT signal (measure interrupt) in order to issue an interrupt.
4. FSM1 changes its state according to A and B values. As a consequence ACTIV becomes '0'. FSM1 continues to monitor A and B signals, while FSM2 waits for the software to read the latched values of ACTIV, SENS and ERR. The latched values are called A for ACTIV, S for SENS and E for ERR. These latches are organized as an input port (fig.3). When the processor reads this port (status port) the RDPML signal becomes '0'.
5. After the software has read the port, FSM2 revert in the initial states.

A problem could arise the software is not fast enough to read the measure port. Suppose the following sequence of events:

1. FSM1 has an event on A or B and activates ACTIV.
2. FSM2 activates READY, latches the event and waits for the software to read the status port.
3. FSM1 changes its states according to A and B values.
4. Another event occurs on A or B. FSM1 activates again the ACTIV signal but cannot change its state because FSM2 still waits.
5. If a second event occurs appears on either A or B, before the status port is read, FSM1 will enter into the error state because this automaton was design to report a single event, not two.

As was state before, the maximum transitions frequency on each axis is 1kHz. So, after an interrupt was issued, the status port must be read in 1ms. Concluding on this subject, the longest uninterruptible instruction sequence must execute in maximum 1ms.

The FSM2 sequential machine was implemented with a single PAL22V10 as in figure 6.

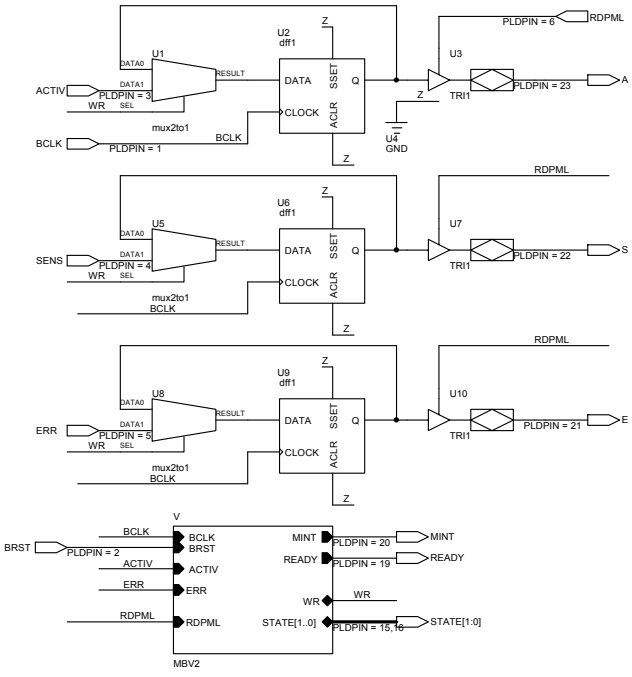

Figure 6

The description has a schematic par and a VHDL part. Although the whole description could be done in VHDL, this approach has the advantage of clarity. The schematic

part depicts how the ACTIV, SENS and ERR signal are processed. As was said before these signals have to be latched on the raising edge of the WR signal. Because the implementation is carried out with a single PAL22V10, a single clock source is available. The signal chosen as clock was the ISA BCLK signal. In order to latch these signals three flip-flops with chip enable were used. Each flip-flop was implemented with a MUX2 and a regular D flip-flop.The control part, with comments, is given in the sequel:

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

ENTITY MBV2 is
 PORT (
  BCLK       : IN STD_LOGIC; --ISA clock
  BRST       : IN STD_LOGIC; --ISA reset
  ACTIV, ERR : IN STD_LOGIC;
  RDPML      : IN STD_LOGIC;

  MINT       : OUT STD_LOGIC;
  READY      : OUT STD_LOGIC;
  WR         : BUFFER STD_LOGIC;
  STATE  : BUFFER STD_LOGIC_VECTOR (1 downto 0)
   );
END MBV2;

ARCHITECTURE behavior OF MBV2 IS
 signal nextstate: STD_LOGIC_VECTOR (1 downto
0);

BEGIN
 process (BCLK)
 begin
  if BCLK'event and BCLK='1' then
   if BRST='0' then state<="00";
   else state <= nextstate;
   end if;
  end if;
 end process;

 process (state,ACTIV,ERR,RDPML)
 begin
  WR<='0'; READY<='0'; MINT<='1';
  case state is
   when "00" => --wait for ACTIV or ERROR
    nextstate<="00";
-- If a status port read is in progress, wait till ends.
--Normally, it is not possible to have a port read before
--the interrupt is issued, but this statement was added
--as a precaution.
     if RDPML='0' then null;
     elsif ACTIV ='1' or ERR = '1' then
      WR<='1'; READY<='1'; MINT<='0';
      nextstate<="01";
     end if;

   when "01" =>  --wait for status port read
                 --to begin
    if RDPML='1' then nextstate<="01";
    else nextstate<="10";
    end if;

   when "10" =>  --wait for port read
         --to end
    if RDPML='0' then nextstate<="10";
    else nextstate<="00";
    end if;

   when others => nextstate<="00";
  end case;
 end process;

END behavior;
```

4. CONCLUSIONS

This paper presents the hardware structure of a CNC, specially design to replace an old CNC, NUMEROM-450 CNC. The new CNC, NUMERIC-01, has an industrial PC as core. Two specialized board were designed: an ISA board and an external board, connected with the first one through a flat ribbon. The hardware structure is one of an acquisition board, with a specialized block that interfaces the linear encoders with the system. In order to reduce the number of ICs, PALs were used. The logic embedded in these devices was described in VHDL. The whole logic design, namely schematic capture, VHDL compilation and PAL implementation, simulation, PCB design was carry out with OrCAD 9.1 package.

REFERENCES

Basker J., VHDL Primer, third edition, Prentice Hall,1989

Cottet, J.,J., Les circuits logiques Programmables (PLD), URL: http://www.ac-orleans-tours.fr/sti-gel/Pld/Orcad_express.htm

M.I.C.M.-C.I.E.T.A I.P.A TcT, Echipament de comandă numerică NUMEROM-450. Scheme electice, 1989

OrCAD Company, OrCAD Express User's Guide