A CRC Verilog description module for a hard real time communication protocol in a control distributed system.

Authors:

J. Pérez. , S. Felici
Department of Computer Science & Electronics.
University of Valencia.
Adress:C/Hugo de Moncada 4.
46010 Valencia. Spain.
Ph : 34 - 6 -360 44 84
Fax : 34 - 6 - 361 61 98
E-mail: solano@glup.eleinf.uv.es
santi@glup.eleinf.uv.es

S. Pelaez, J.M. Insenser.
Sidsa
Adress: Parque Tecnológico de Madrid.
Edificio centro de Empresas 1.
28760 Tres Cantos. Madrid.
Ph : 34 -1 - 803 50 52

Provided the contribution is accepted, **J. Pérez** will present it at the Conference.

All appropriate organisational approvals for the publication of this paper have been obtained. If accepted, the authors will prepare the final manuscript in time for inclusion in the proceedings.

A CRC Verilog description module for a hard real time communication protocol in a control distributed system.

Topics :
- ***Use of HDLs in industrial projects***
- ***HDL synthesis***

Abstract.

In this article we pretend to describe a system design based on Verilog for industrial control application with emphasis in the specification, synthesis, simulation, place&route, layout and pattern generation steps. The focus of this paper is the implementation of a mechanism in error detection with cyclic redundant check for a communication module in an industrial noise environment with special interest on a custom hard real time communication protocol. We put special attention on hardware description language, Verilog and how from it we designed the CRC circuit. This system has been developed and designed by SIDSA and University of Valencia, supported by GAME[1]*. This article doesn't point to any particular application or installation although it was, but for industrial reason this application should be kept confidential. The system developed could be adapted to any industrial environment with just programming the central unit, based on a 8051 microcontroller for any control application using its hard time protocol mechanism with cyclic redundant check for communication error detection.*

1. Introduction.

This paper is a brief summary of a project developed and designed by SIDSA and University of Valencia, supported by GAME. The objective of this project is the consecution of a robust and reliable industrial control system using HDL in its definition, with better performance, easier maintenance and lower cost than one based on automates.

The fast evolution and progress in the design methodology with hardware description languages have done possible them to be applied easily in a lot of industrial system designs. The point of this paper is just related with a typical feature on industrial environments, such noise and electrical perturbations, and how using HDL that problems are overcome.

[1] This article is part of a project financed by the *Grupo Activador de la Microelectrónica en España* (GAME) and has been developed by the Department of Electronic Engineering of the University of Valencia jointly with the SIDSA company.

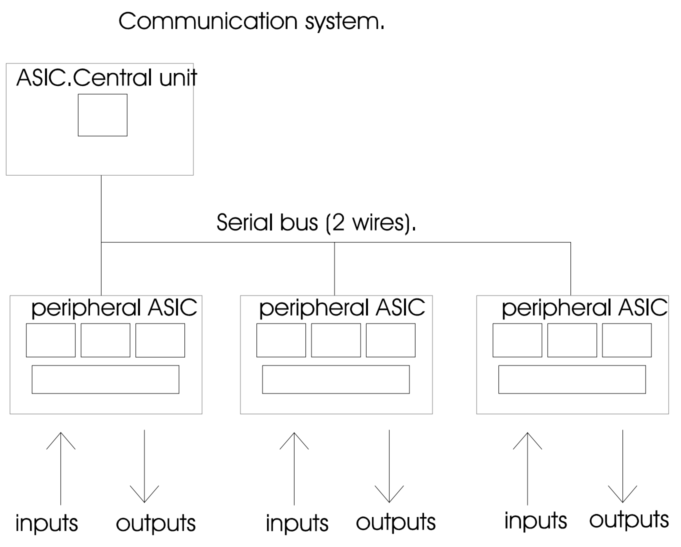

Figure 1.Graphic System overview.

The whole industrial system designed has two parts, on the one hand a central part that controls every task of the system, and on the other hand a part controlling the peripheral components which interface directly with the industrial environment. Both parts have been implemented with specific application integrated circuits (ASICs). Further on we will refer to them as central ASIC and peripheral ASIC. This ASICs have been fully developed with Top-down technology explained later, from a hardware description language like Verilog-XL.

The central unit (central ASIC) is the manager and controller of the whole system, capable to execute the control algorithms and to implement efficiently the hard real time communications with the peripheral units (peripheral ASICs) through message passing mechanism using a master-slave model. One of the main factors to take into account, is just the noise environment and how to avoid them using a well designed messages with CRC and modulated with noise immune mechanism such FSK.

2. Design methodology and system description.

The system based on ASICs, have been developed using the top-down methodology. The first step is the hardware modelling with a high level language, VERILOG-XL, to a RTL abstraction level [1]. This language was chosen since it was better for synthesis than VHDL [2] because its advantages are simplicity, direct equivalence among the data types and the quality of the synthesised hardware, among others.

Once the model has been developed, the synthesis process starts [3], passing from a high level description to another of minor abstraction. This process has been carried out with the CADENCE Sinergy synthesiser which, starting from VERILOG modules, generates a schematic at logical gate level using the technology supported by the selected library like those of ACTEL or ES2. The ACTEL library allows the implementation of the design using FPGAs and detects possible design errors before manufacturing the ASICs. The ES2 library is the definitive one to implement the

ASICs, with CMOS 0.7 microns process for the central ASIC and 1 micron for the peripheral ASIC.

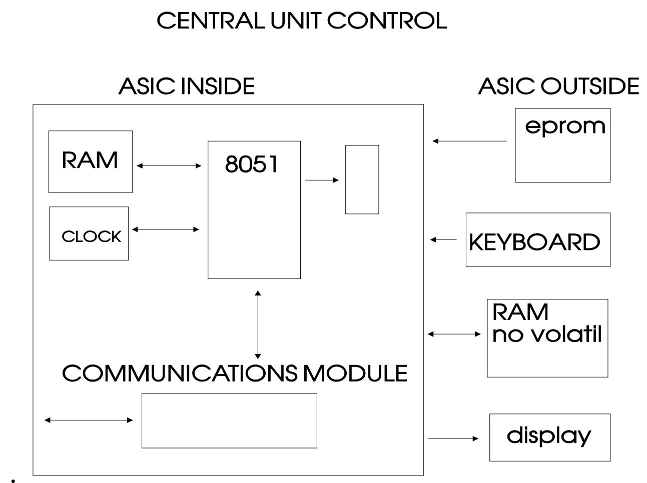

Figure 2. Central Unit Scheme

The system has one central ASIC controlling 8 peripheral ASICs that interfaces with the industrial environment. The central ASIC is placed in the master board (control unit) that contains the program, the memory, the keyboard and the display. The control decisions are taken in the central ASIC. The central ASIC has as main component a 8051 cell that is totally compatible with the Intel uP and executes the control program stored and also it has a communication module and the glue logic for the uP. This ASIC design has been totally synchronous with a maximum operation speed of 12Mhz in the 8051 block and 32.678Hz in the real time clock as we see in the central unit diagram block shown in figure 2.

Each peripheral ASIC has 10 digital outputs, 10 digital inputs, 1 analog output, 1 analog input with 1 A/D and 1 D/A and also a communication module inside. Each peripheral ASIC can be coded as an digital or mixed one (digital + analog) with jumpers, so if digital, it uses only digital messages but it uses mixed messages. The devices to control from this systems such electrovalvules, dossifiers, motors, encoders, variable speed AC drivers, etc are distributed around the system and every peripheral ASIC control some of them.

3. Communication module. CRC block

The physical layer used is a coaxial cable of 50 ohmios and BICMOS drivers for bidirecctional communications with bus topology which enables to enlarge the system.

The communication protocol is a master/slave protocol. The central unit requests information to the units connected to it and these units answer just when they have been requested.

An interruption routine programmed in the central unit every 512 us sends the following types of information frames :

- Out of order, this frame disables the peripheral at the same time.
- Check, it forces the peripheral to answer with the current state of its inputs and it is sent from central unit waiting 138,8 us since the bus is idle
- Digital, it sends the value that the peripherals should have at the digital outputs
- Mixed, the same as the digital but including digitalized analog values

A failure prevention mechanism in the peripheral devices is introduced, at this level of protocol definition, by the polling routine using the check frame. This routine checks all the peripheral devices when the central unit is not carrying out any access to any devices.

The steps needed to generate the information exchanged are the following: the sender builds the frame with a defined format, generates the CRC , explained later, of the frame and after codes it in Manchester and modulate in FSK

Sender: generate the frame=> calculates CRC=> codes Manchester=> modulates FSK=> sends

On the other side, the receiver demodulates from FSK, decodes Manchester and verifies with CRC and if no error is found it gets the information. In other cases the receiver waits to the retransmission by the sender time out.

Receiver: demodulates FSK=> decodes Manchester=> calculates CRC=> receives

The synchronisation employed in this protocol is implicitly in the Manchester code, thus the system could be considered asynchronous. The clock information is included in the binary information, synchronising with the rising and falling edges of the signal.

In the digital FSK modulation (Frequency Shift Keying), three frequencies are utilised:

Symbol.	***Frequency used.***
low Level (f0)	*230.4 KHz*
high Level (f1)	*115.2 KHz*
frame start character (fst)	*-from Central to the peripheral unit 460.8 Khz* *-from peripheral to Central unit 57.6 Khz*

with a transmission speed of 9.600 bits per second.

Next table shows the time needed to complete a whole transaction (send and receive) without any error and any delay for every kind of frame.

Transactions	***Average time (us)***
Out of order	2500 us
Check	6103 us
Digital	6874 us
Mixed	8541 us

The last two frames, digital and mixed, are shown in figure 3. How we can see, the duration frames depends on the destination and on the digital or analog information. Next table contents the time information for each one.

Types of frames	*Sending time in micro seconds*	*Number of bits*
Sending from C.U. *(mixed information)*	4375 us	42 bits
Answering from peripheral. *(mixed information)*	4166 us	40 bits
Sending from C.U. *(digital information)*	3541 us	34 bits
Answering from peripheral. *(digital information)*	3333 us	32 bits

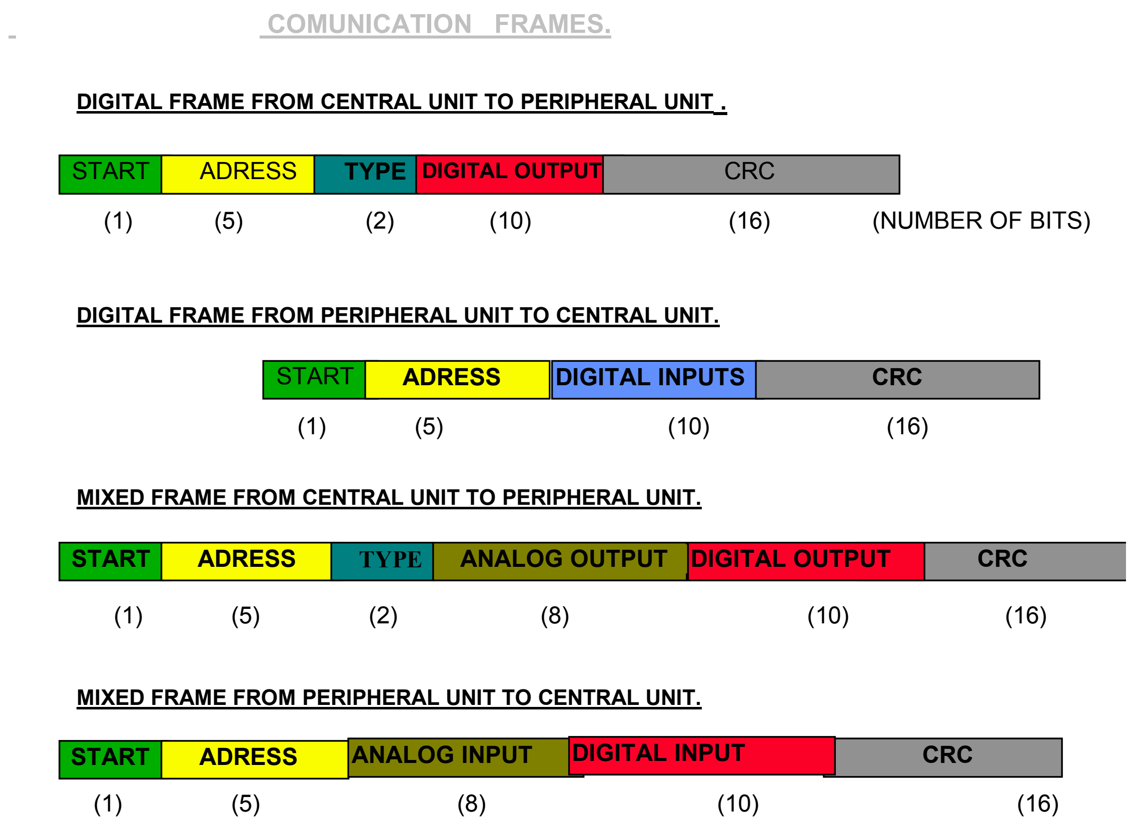

Figure 3. Communication frames.

The CRC (*Cyclic redundancy check*) module implements the next equation [4] for each message m(x) to send :

$$c(x) = x^{n-k} \cdot m(x) + resto\left(\frac{x^{n-k} \cdot m(x)}{g(x)}\right)$$

where product operator is XOR, c(x) is the code generated, that will be send, with m(x) and the remainder done with the polynomial generator CRC 16:

$$g(x) = x^{16} + x^{15} + x^{2} + 1 = (x+1)\cdot(x^{15} + x + 1)$$

This CRC module is based on systematic lineal block codes that means that each c(x) has first m(x) of k bits followed by n-k redundant bits, the remainder with g(x). The amount of bits needed to transmit a m(x) of k bits are n, then with this methods 2^n-2^k code errors will possible.

This CRC operation has the property that the code generated has g(x) as a polynomial factor, then c(x) could be divided by g(x). But if c(x) is damaged by a error code (similar to add an polynomial error e(x)) that could be detected by the next expression

$$\frac{r(x)}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

where there are more o less error detection possibilities depending on the g(x) selected.

At the implementation step, the g(x) selected is 0xA001 (1010 0000 0000 0001) with n-k=15 represented with 16 bit, corresponding with

$$x^{15} + x^{13} + 1$$

that comes though algebraic transformation from the CRC-16 shown above [5].

4. Verilog description for CRC module.

The source for CRC modules explained above, in Verilog languange is shown below. Two different modules are described one to generate the CRC (module cod_crcuc) and other to check the CRC (module dec-crcuc). The Verilog implementation is listed in the next paragraph :

```
// Verilog HDL for perif, cod_crcuc _functional
// Code CRC.
module cod_crcuc(rb,ck,en,cheq,mix,dig,apag,en_bit,desp,in,si,test,dcrc,so) ;
input rb,ck,en,cheq,mix,dig,apag,en_bit,desp,in,si,test;
        // inputs:
        //(rb) reset,(ck)reloj,(en)enable,message type,
        //(en_bit)enable,
        //(desp) enable the crc output
        //(in) bits input
        //(si,test) scan inputs.
output dcrc,so;
        //(drcr) crc code , (soscan outputs.
reg[16:1] accum;
reg dcrc;
reg[2:1] estado;
`define INIC     2'd0
`define CODIF    2'd1
`define DESPLINT 2'd2
`define DESPL    2'd3

always @(posedge ck)   //state machine
    if(en)
    begin
      case(estado)
        `INIC:
          begin
            accum=0;
            if(cheq || mix || dig || apag) estado<=`CODIF;
            else estado<=estado;
          end
        `CODIF:       //divide bit a bit
          begin
```

```
            if(en_bit)//if the enable bit is active
              begin
                if(in^accum[1])
                  accum<=(accum>>1)^16'ha001;
                else
                  accum<=accum>>1;
              end
            if(desp) estado<=`DESPLINT;
            else estado<=estado;
          end
        `DESPLINT:    //CRC generation
            begin
              dcrc<=accum[1];
              accum<=accum>>1;
              estado<=`DESPL;
            end
        `DESPL:
          if(en_bit)
            begin
              dcrc<=accum[1];
              accum<=accum>>1;
              if(!desp) estado<=`INIC;
              else estado<=estado;
            end
        default estado<=`INIC;
      endcase
    end

always @(rb)   //reset
  if(!rb)
    begin
      assign estado=0;
      assign dcrc=0;
      assign accum=0;
    end
  else
    begin
      deassign estado;
      deassign dcrc;
      deassign accum;
    end
endmodule

// Verilog HDL for perif, dec_crcuc _functional
// Decode CRC. if crc is correct put crc_ok
//(the polinonial is a001).
module dec_crcuc(rb,ck,en,ft,en_trama,in,si,test,crc_ok,so) ;
input test,si,in,en_trama,ck,en,rb,ft;
        // inputs:
        //(rb) reset,(ck)reloj,(en)enable,message type,
        //(ft)final bits input,(en_trama) enable bits input,
        //(in) bits input
        //(si,test) scan inputs.
output so,crc_ok;
        //outputs, (crc_ok) if crc is correct,
        //(so) scan output.
reg[15:0] accum;
reg crc_ok;

always @(posedge ck)   //state machine.
  if(en)
  begin
    if(en_trama)      // decode crc.
      begin
        if(in^accum[0])
          accum<=(accum>>1)^16'ha001;
        else
          accum<=accum>>1;
      end
    if(ft)     // final.
      begin
        if(accum==0)
          crc_ok=1;
        else
          accum=0;
      end
    else
```

```
      crc_ok=0;
  end

always @(rb)             // Reset
  if(!rb)
    begin
      assign accum=0;
      assign crc_ok=0;
    end
  else
    begin
      deassign accum;
      deassign crc_ok;
    end
 endmodule
```

5. Simulations.

The simulation process is a fundamental step for avoiding any functional and timing mismatch in the design. The critical parts to check were the 8051 module and the communication module. For the first one, some critical software regions were simulated in conjunction with the whole design, the code generated to execute the 8051 was written with a commercial 8051 C compiler and after that, the generated code was loaded inside a Verilog ROM model. For the second one, the communication protocol, the simulation was done with a sender and receiver modules, one as a master an other as a slave. We can show one of this simulations with the CRC module in the next figure 4.

6. Layout and implementation fase.

After the synthesis showed in the part 2, we now just include a SCAN PATH. This process is performed automatically by the synthesiser linking all the ASIC modules with a register chain that contains and transmits all the test patterns. Although SCAN PATH is normally useful, the internal RAM has been generated like an ES2 Megacell introduced into the BIST [6] since SCAN PATH was less efficient.

To complete the design it is necessary to get some good patterns to test and detect the chip errors once the chip has been fabricated. These patterns include a little executable program for the microcontroller that test all their internal nodes. This program reads and writes the internal registers and the serial port, and executes a wide range of the 8051 instruction set. Once the microcontroller is tested, the remain circuit is checked introducing randomised patterns through the Scan Path. With this test mode we got a fault coverage around the 93% of the internal nodes of the ASIC with the Stuck-0/1 method [7], introducing 73.090 test patterns.

Once the circuit has been implemented, targeting the ES2 library, the PLACE & ROUTE of the ASICs was done, getting the layouts shown the figure 5. This process starts with the schemes got previously where we had the whole system implemented using logical gates. The placement in the central ASIC has been done in zones; all the 8051 module used one, the communications module other, the RAM another and the last for the real time clock.

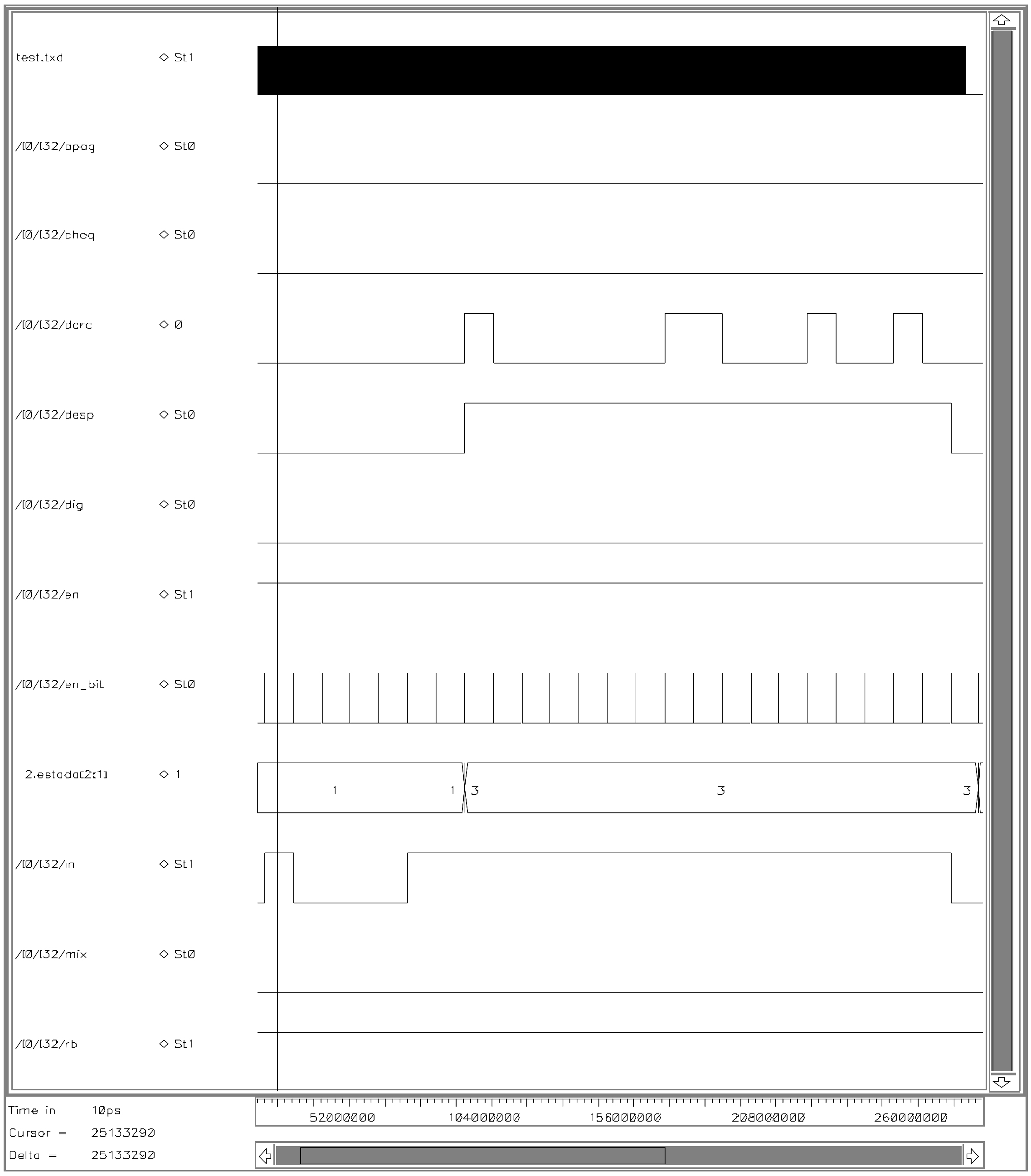

Figure 4. CRC Module Simulation.

The routing process was automatically done with the CADENCE software. The size of the central ASIC was $24.3mm^2$, and the package was the BCQ84VS ceramic of 84 pins.

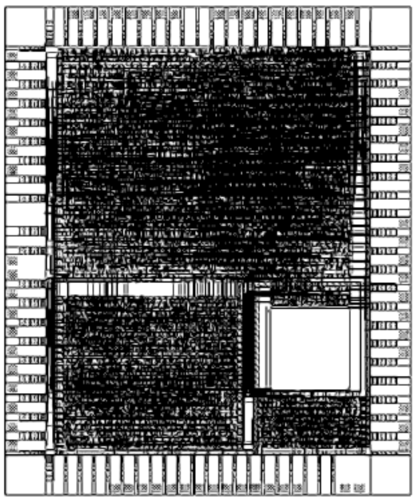

Figure 5. Layout.

7. Experimental results.

In our experimental results we have tried to justify every equation and time estimation done. Since such point of view the next results come from checking the connection with a digital scope PM3394.

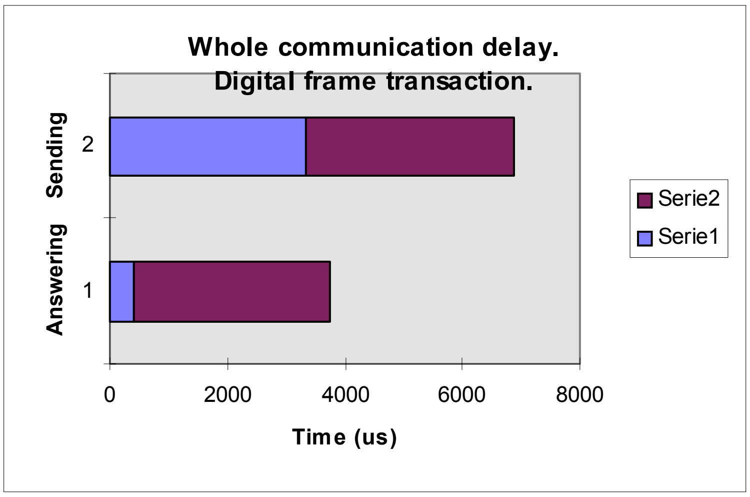

Figure 6.

Figure 6 shows the average time needed to complete a whole transaction without any error detected in the CRC module described. *Duration serie1* is the time needed to process the information (note that in the sending way we need an average of 3328 us waiting to finish a check processing started) and *duration serie2* shows the time needed transmitting at 9600 bauds. The whole transaction for 10 output and 10 digital input is around 10602 us.

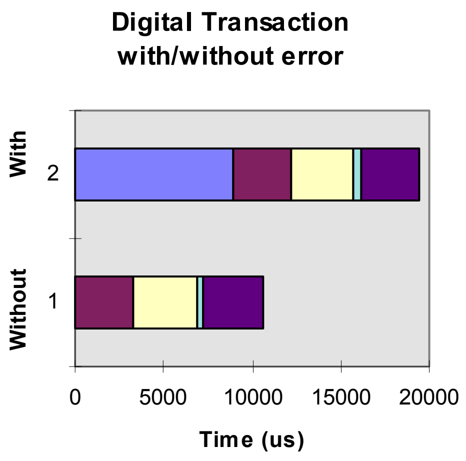

Figure 7.

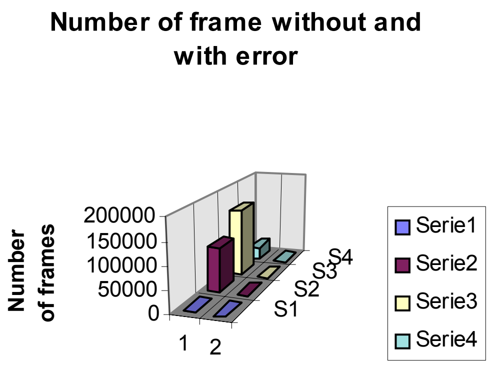

Figure 8.

Figure 7 shows a digital frame transaction with an CRC error and consequently its TimeOut mechanism (time out is just the time needed to complete the largest transaction, 8888 us), so when an error is detected and the TimeOut expired the communication module sends again a new frame just after the TimeOut. Figure 8 shows the number of retransmission needed in the noise environment application to complete each type of frame in one hour working at its 85% capacity. Serie1 corresponds to out of order frames, serie2 corresponds to check frames, serie3 corresponds to digital frames and serie4 corresponds to mixed frames.

8. CONCLUSION.

We have shown all the design methodology and steps for the development of an industrial control system with a control unit and some peripherals, based on ASICs, with advantages in noise tolerance, versatility and easy programming. We must highlight that every feature described here has been checked with a noisy industrial environment, obtaining the expected outputs shown in the experimental results.

Typical applications that could make use of the developed system could be the sewage treatment, automatization processes, dangerous installations, robotics, etc, including mechanisms like those that can be seen in the next table.

Industrial sector	**Applications**
Chemical industry	*Sewage treatment, toxic substances*
Cement industry	*Dusty environment*
Automotive industry	*Assembly chain, automatization processes*
Iron & steel industry	*processes control*
Manufacturing	*robotics, dangerous situation*

With this technology, we could enlarge the system as much as we like and in other way, it is a very protected mechanism to avoid any copy, main factor in our industrial application.

9. REFERENCES.

[1] SYNERGY. Family of design synthesis technology. VERILOG HDL Design Guide. + CADENCE Design Systems 1992.

[2] B. Fuchs “Verilog HDL est bien preferable a VHDL”. Electronique, Nº 35. February 1994.

[3] R. Camposano & W. Rosenstiel "Synthesizing circuits from behavioral descriptions" IEEE Trans. Computer-Aided Design, [vol]. 8, [pp] 171-180, Feb. 1989.

[4] J. Rifa LL. Huguet. Comunication digital. Teoria matemática de la información. Masson.

[5] Joe Campbell.Comunicaciones Serie. Guía de referéncia del programador en C.

[6] P.A.and T. Anderson, Fault Tolerance. Principles Practice [and]. Ed Springer-Verlag Wien. 1981.

[7] Manual of reference, Verifault CADENCE Design Systems February 1991.