ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Napoli

INFN/TC–97/36
15 Dicembre 1997

F. Fabozzi, P. Parascandolo:

A MULTICHANNEL ANALOG PULSE GENERATOR

SIS–Pubblicazioni
dei Laboratori Nazionali di Frascati

INFN – Istituto Nazionale di Fisica Nucleare
Sezione di Napoli

INFN/TC–97/36
15 Novembre 1997

A MULTICHANNEL ANALOG PULSE GENERATOR

F. Fabozzi and P. Parascandolo
INFN – Sezione di Napoli
Complesso Universitario di Monte S.Angelo, Edificio "G" – 80126 Napoli, Italy

Abstract

Design and performances of a 96 channels analog pulse generator to be used at BaBar experiment are reported. The output pulses can be programmed either as positive or negative. Amplitude (up to 1.2 V onto a 50 Ω load; 7 bit resolution) and pulse width (8 bit resolution) are also programmable.

1. INTRODUCTION

Resistive Plate Chambers (RPC) [1] have been chosen as active elements in the IFR subdetector [2] of the BaBar experiment [3,4] to identify μ and detect K_L particles. The strips of the RPC detectors (about 50K channels) are hooked to the front end electronic cards (FEC or PFEC) [5,6,7] mounted just at the edges of the chambers. Each front end card serves 16 input channels and acts both as a discriminator and as buffer of Data. From this point the Data will be transmitted to 52 FIFO boards (IFB) [8] located into 8 crates close to the apparatus, which will provide for data buffering (Fig.1).

The standard Readout Module (ROM), common to all subdetectors of the BaBar experiment, is located into the electronic house (EH) and performs crate readout via a high speed (1.2 GHz) optical link (CLINK & DLINK). The fiber optic system serves both to acquire data from the front end electronics to the DAQ (DLINK) and to transmit clock (59.5 MHz), trigger and control signals to the IFR subsystem crates (CLINK).

To check proper operation of the front end electronics both FECs and PFECs incorporate a test input where a pulse of suitable amplitude and width can be applied simulating an hit on all the sixteen channels. Applying pulses of different amplitudes and widths a monitoring of the threshold stability can also be performed.

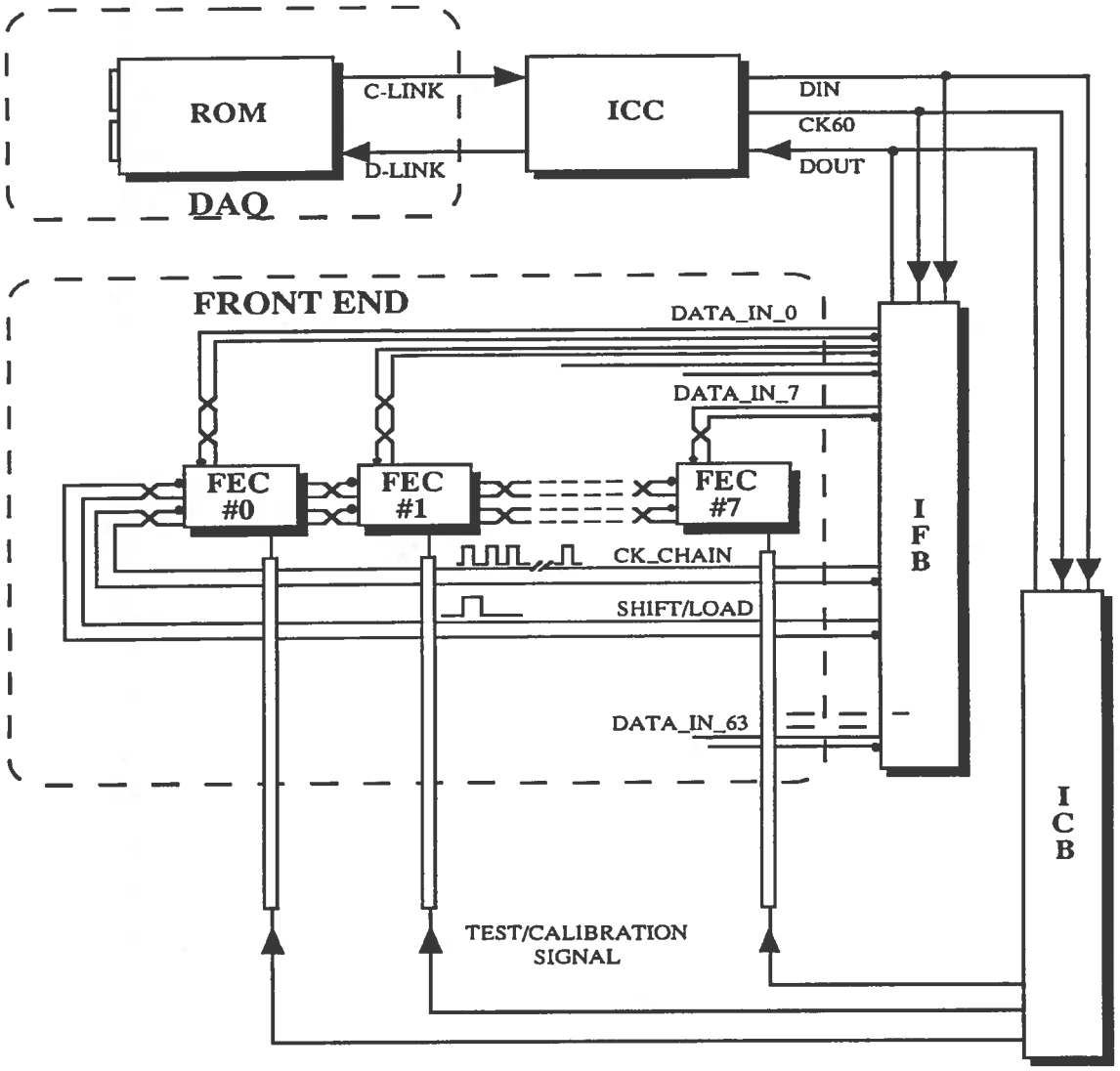

Fig.1: Overview of IFR electronics

The ICB board (IFR Calibration Board) we have developed is aimed to serve these purposes by producing 96 pulses of programmable polarity (to test both FECs and PFECs), programmable amplitude (7 bit resolution) and programmable width (8 bit resolution).

This note first introduces to the established protocol between the standard ROM and the IFR front end electronics; then it focuses on how the serial bit-stream recognizing process is implemented in the ICB and how the DAQ controls proper operation of the board. Last it focuses on the analog section of the module, where pulse generation takes place.

2. COMMAND FORMAT FROM THE ROM TO THE IFR FRONT END ELECTRONICS

In the idle state the Data bit of the GLINK is stuck at zero while the 59.5 MHz clock continues to run. A detailed description of the requirements imposed on the receiving and on the transmitting section of the IFR modules can be found elsewhere [9].

The serial bit pattern received by the IFR subsystem can be identifiable either as Run-time commands or as Non-run time commands. The former are in general time critical while the latter are those commands not allowed during data taking: set pattern mask, pulse width, etc.

A "command" sent by the DAQ is a serial string of twelve bit of Data (clock at 59.5 MHz), the first one being always zero and the second one "start bit" always at 1. It follows a pattern of 5 command bit so a maximum of 32 Operation codes are allowed. On each module (ICB, IFB, ITB) a fixed pattern detector compares the serial bit stream against a predetermined pattern producing different flag signals for each allowed pattern.

The format of the Run-time commands is the following:

z s c c c c c d d d d d

z ==> 0
s ==> 1
c ==> command bits, LSB first
d ==> data bits, LSB first

The ICB pattern detector decodes the Run-time commands NOP and CAS; it also issues a false command flag FC.

NOP -code 0- is a no-operation code. No operation is required onto the module but the command has to be decoded and a flag to be written and transmitted back to the DAQ later.

CAS calibration strobe -code 5- is the command used to start output pulse generation after a proper test pattern has been set.

False Command -code xx-. Each time there is no match between the serial input and the internal fixed pattern, the ICB sets a flag of False Command indicating a reception of a wrong pattern or a failure in the command decoding process. This flag is stored and transmitted later to DAQ.

Non-run time commands have the format indicated below:

z s c c c c c a a a a a d d d d ...

z ==> 0
s ==> 1
c ==> command bits, LSB first
a ==> subaddress bits, LSB first
d ==> data bits, LSB first

Bits "z", "s", and "c" have the same meaning outlined above. The five "a" bits are subaddress bits (LSB first) to perform subcommand functions inside a module. The data pattern, always with the LSB bit first, consists of n "d" bits and can be of variable length depending on the specific operation to be performed. In the IFR subdetector for example n=64 to write a test pattern into the IFB; but for operations onto the ICB module n=96 or 16.

Non-run time commands are issued to the ICB module either to write a pattern of 96 bit to select which output channels must produce a test pulse (WRP command: code 10, subaddress 2) or to write a pattern of 16 bit to set amplitude, width and polarity of the output pulse (WRC command: code 10, subaddress 1). To verify proper operation of the module the DAQ can send an RDP or an RDC command - code 19, subaddress 2 or 1 respectively - to read back the pattern written onto the board with a WRP or WRC command. A header pattern, with the start bit at "one" and consisting of 32 bit with a format similar to that of the IFB, is added and transmitted to DAQ each time an RDP or RDC command is issued.

To guarantee fully compliance with BaBar standard, we have assumed as a requirement that the pattern detector should work up to 14 nsec minimum, well beyond the 16 nsec operating BaBar clock period, with a design goal of 12 nsec.

3. IMPLEMENTATION OF THE DYNAMIC PATTERN DETECTOR (Icbdec)

The design of a dynamic pattern detector operating continuously at a clock frequency of 59.5 MHz is rather tricky and requires a careful choice of the PLD [10]. We have chosen to implement this design into a MACH210A-7 [11]. Employing a shift and decode approach the pattern detector has been built around a 12 bit shift register Y[11:0] whose clock is CK60, the 59.5 MHz master clock (appendix A). The outputs Y[6:0] of the shift register are used to identify commands.

Reset is produced internally after 12 clock cycles for NOP, CAS and FC. For WRC, WRP, RDP, RDC it arrives via the RSPC input (produced into an external PLD) after 12+16 cycles, 12+96 cycles, after 12+32+16 cycles or 12+32+96 cycles. Reset operation of the shift register must be performed in just one cycle to adhere to the BaBar protocol. Therefore, the asynchronous reset of the macrocells is not viable since the t_{AR} is 12 nsec for the fastest device available in the family (MACH210A-7). Consequently, to perform a reset of the Y[11:0] shift register we have opted for a "write zero" operation which is much faster. The asynchronous reset is the RS_OK signal (produced by the PLD and externally connected to RST input) and is used just to reset not time critical registers such as the command flags registers, delay registers, and strobe registers.

The two flag signals VC and FC (standing for valid command and false command) are produced after 8 cycles and when active they disable the decoders so that two commands cannot be active at the same time. GCD, GCDD and GCDDD build up a three stage delay to produce the RS_OK signal at the end of the twelfth command bit.

For all time critical signals used inside or outside of this PLD, the equations cannot be combinatorial. In fact, to assure reliable operation, the time t_{CO} (delay clock to output) plus t_{SU} (set up time from input clock or feedback) plus PCB delay (in case of signals going to or coming from outside this PLD) must be well under our design goal (14 nsec).

In the following graphs we show how, with a 7 nsec set up time of DIN vs. CK60, the PLD works with a 12 nsec clock period for some consecutive commands (NOP–NOP, FC–FC).

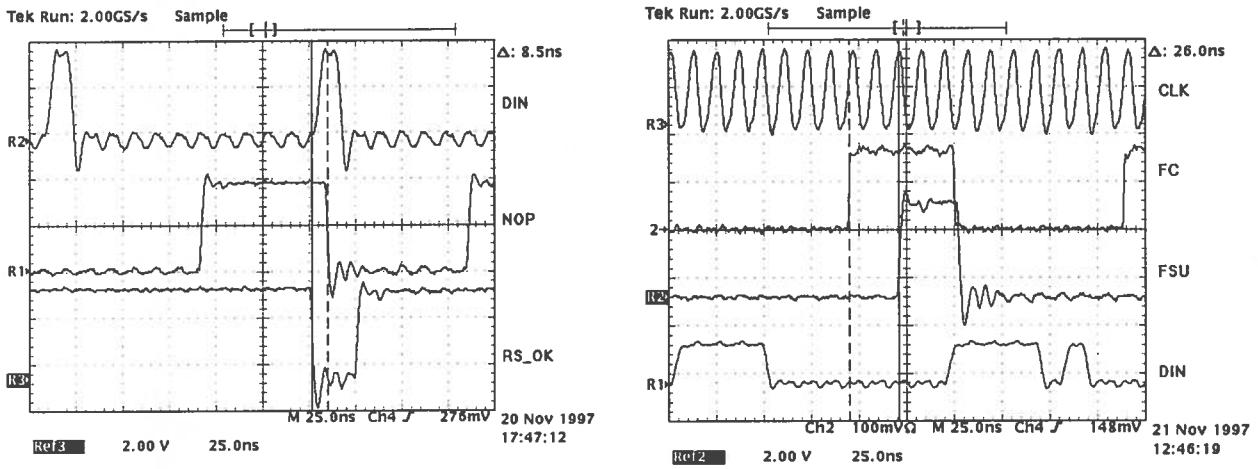

Fig. 2: Decoding action of Icbdec PLD

4. SIXTEEN BIT SHIFT REGISTER AND COUNTER PLD (Sh_ctr)

This PLD (Abel file into appendix B) acts as a controller unit for the loading of the data arriving from the DAQ with a WRC or WRP command. Moreover the 16 bit pattern to be used as width, amplitude and polarity code is stored into an internal register.

To accomplish the first function, this PLD produces a properly timed RSPC signal to reset the pattern decoder PLD (Icbdec). The RSPC signal is produced as follows:

WRP command ====> after 95 cycles;
WRC command ====> after 15 cycles.
RDP command ====> after 127 cycles
RDC command ====> after 47 cycles

Read commands last longer as for them the transmission of a 32 bit header must precede the data sent to DAQ.

Next to a write command an enable-to-shift signal is produced: EN_WC (in case of a WRC, as shown in fig.3) or EN_WP (in case of a WRP).

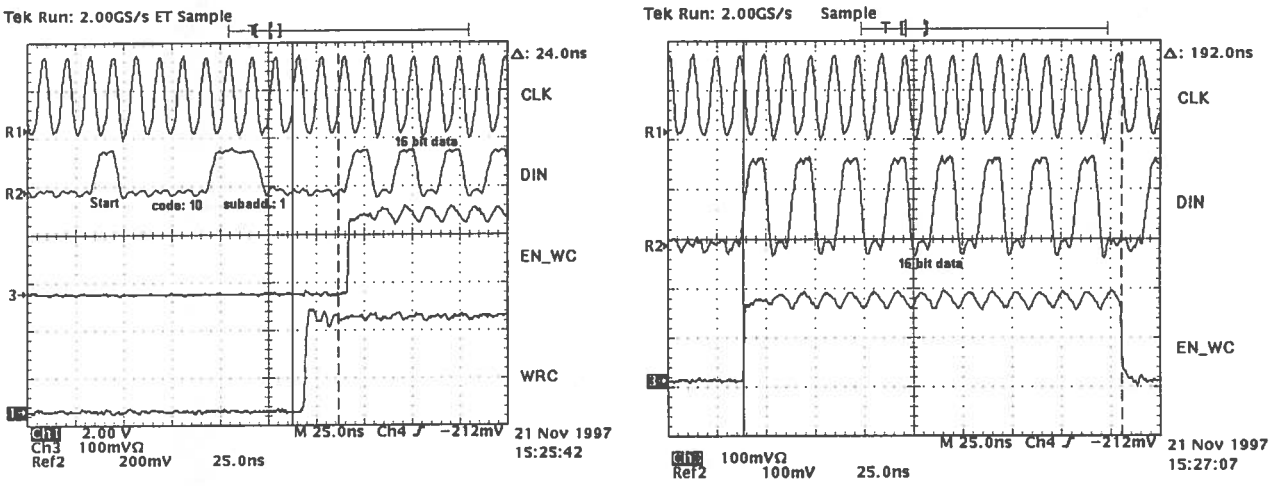

Fig.3: Loading of the 16 bit data after a WRC command (12 nsec clock period)

Next to a read command other enable-to-shift signals are produced: EN_RC (in case of an RDC, see fig.4) or EN_RP (in case of an RDP). To read the pattern without destroying the previous content of the shift register, in case of an RDP another enable signal is activated: EN_REC. EN_REC lasts 96 cycles exactly while EN_RP lasts 32 +96 cycles.

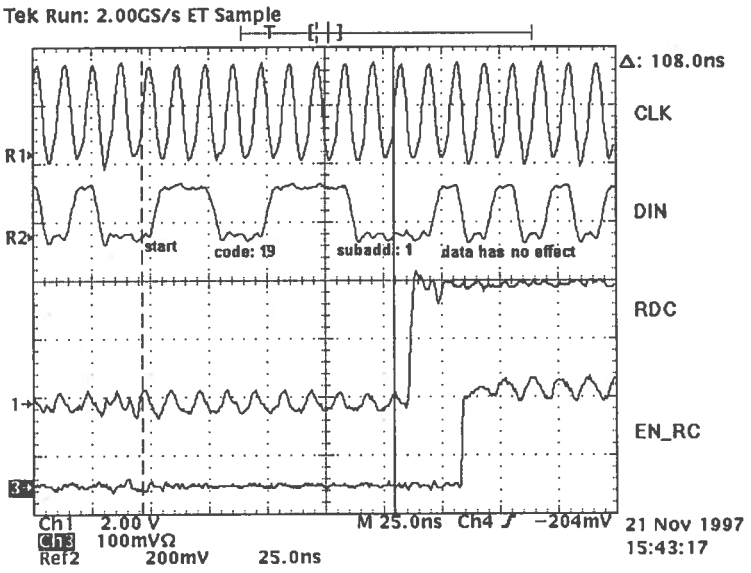

Fig.4: Generation of the enable to read the control register (12 nsec clock period)

To accomplish the second function, into this PLD a 16 bit shift register Y[15:0] works as a control register for amplitude, width and polarity of the output pulse. The lower 8 bit of the shift

register are connected to a time delay generator, while, of the remaining 8 bit, the upper 7 are devoted to control the amplitude of the output pulse via an external DAC and one bit to fix the output polarity of all the 96 channels.

5. 96 BIT SHIFT REGISTER AND HEADER PLDs (Rec_Wpt, Wpt1, Wpt2 and Header)

Three PLDs (Wpt1, Wpt2 and Rec_Wpt1) are necessary to write the 96 bit pattern. These PLD are roughly the same, with the first (Rec_Wpt) being slightly different in that it has two inputs from which to shift in data: the first is DIN fed directly by the ROM, the second is SP to recirculate data next to an RDP command.

The Header PLD is devoted to insert a 32 bit pattern data each time a read command is issued. The format of the header pattern is shown fig.5.

In the successive figures an example of data transmission to DAQ is given (at a clock period of 12 nsec).

In fig.6(a) we show the transmission of the 1st and the 2nd byte of the header (NOP flag is active); in fig.6(b) it is shown the 3rd and the 4th byte of the header; in fig.7(a) and 7(b) the beginning and the end of the transmission of the 96 bit data is given (data: 1010...1011).

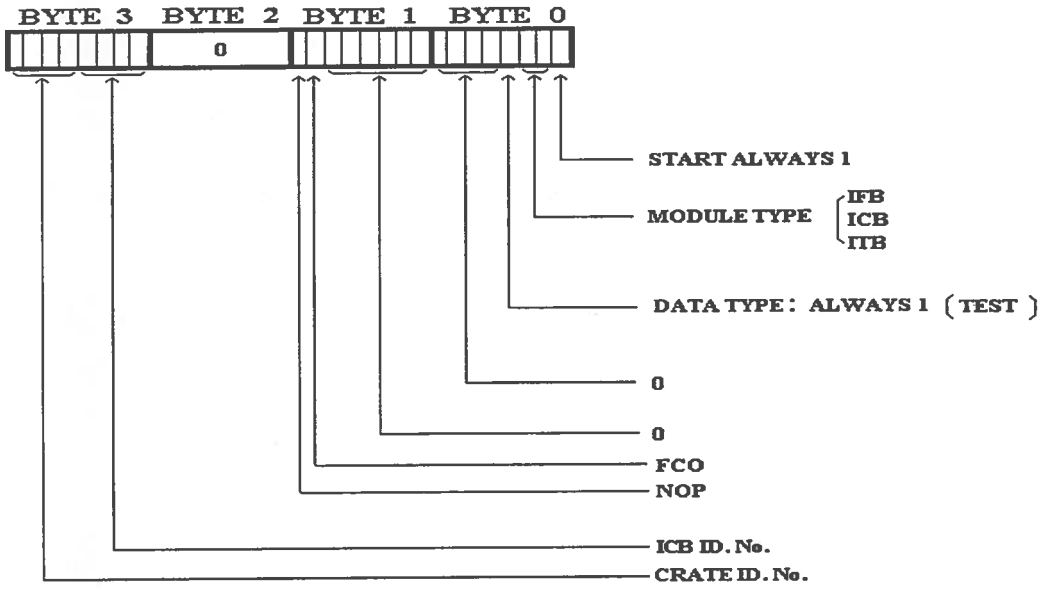

Fig.5: Header format

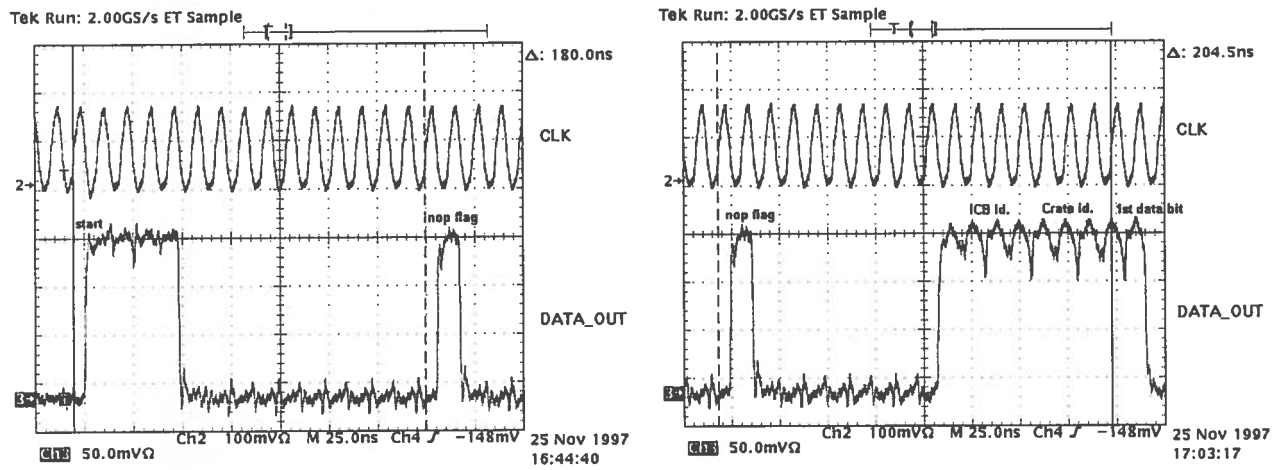

Fig.6(a) and 6(b): Transmission of the header

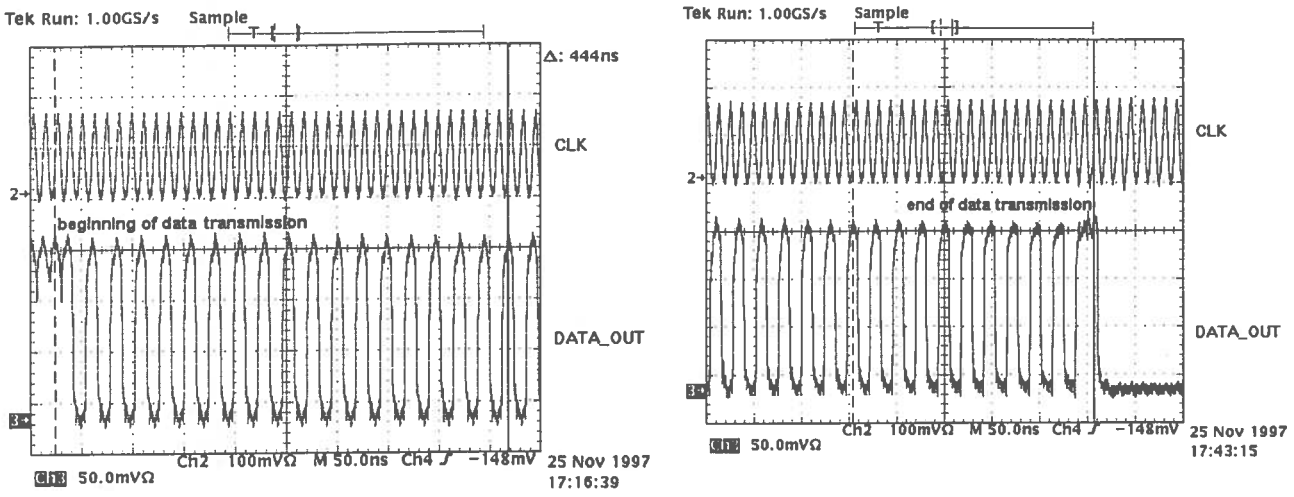

Fig.7(a) and 7(b): Transmission of the data

6. THE ANALOG SECTION

The analog section of the board is based upon a DAC08 to produce a programmable voltage, upon an AD9500 to produce pulses of programmable width, and a CDC391 to produce pulses of programmable polarity.

The DAC08 is a well known old workhorse converter. Its current output, via OP1, is converted into a voltage. OP2 together with a current booster provides a stiff voltage source VR (fig.8) for the following differential stage to control the amplitude of the output pulse.

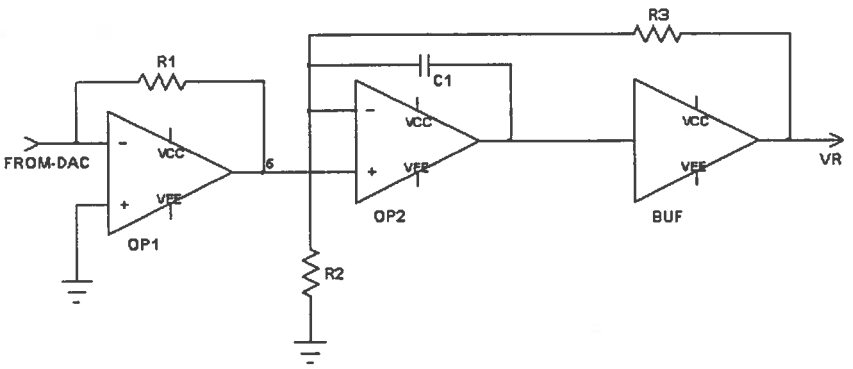

Fig.8: Generation of the programmable VR voltage level setting pulse amplitude

The AD9500 is to date the best digital delay generator with stability in the order of tenth of picoseconds. Each time a calibration strobe is issued, on the twelfth clock cycle a positive pulse (CAS) is produced by Icbdec. This pulse is first converted to an ECL level (fig.9) and then it sets both an MC10131 flip flop and the AD9500. When the programmed time (8 bit) of the AD9500 is elapsed, a fast pulse is produced to give the reset to the MC10131 flip flop.

The pulse produced at the inverted output of the MC10131 (whose width is function of the applied code and of the programming capacitor at the AD9500) is converted to a TTL level (YW), and then is applied to the XOR gate made up with a CDC391. If polarity must be negative, the bit YC8 in the 16 bit control register must be set low otherwise the output of the XOR will be positive.

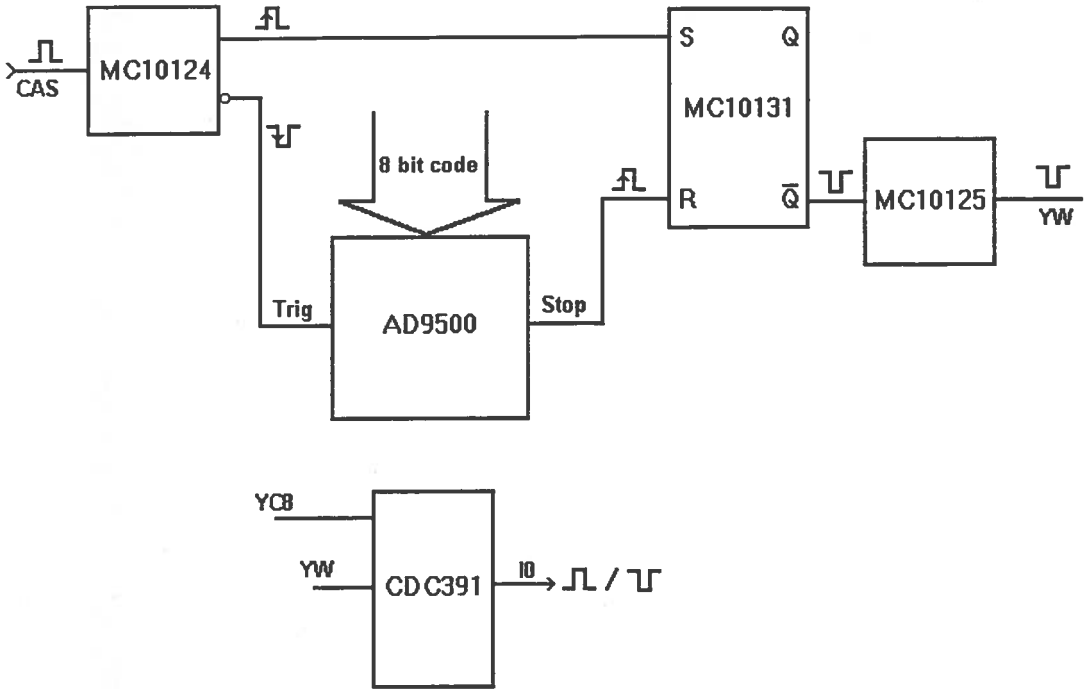

Fig.9: Production of the programmable pulse width

The pulse generator (fig.10) is based upon a differential amplifier with a current source at the emitter. The current source establish the amplitude of the output pulse because the VR voltage sets the output current. The RF BF579 transistor commutates with rise time in the order of 1 nsec and so very quickly once the appropriate positive or negative pulse is applied to the base of Q1. Three such pulse generators are used.

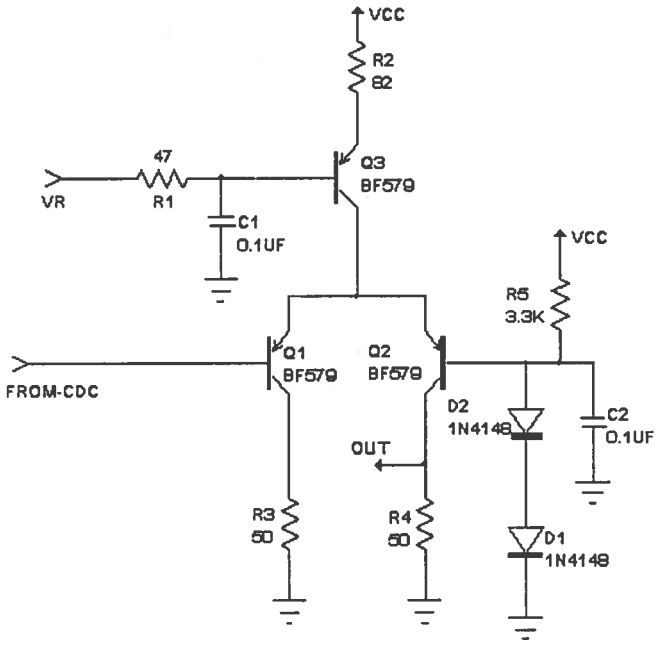

Fig.10: The pulse generator

The output pulses of the generators feed 3 MAX 4135 video amplifier acting as splitters for a bank of 16 MAX 4135 which then drive the input of the FECs.

In the following graphs (fig.11) we show one of 96 output channels producing pulses positive and negative at the maximum amplitude after 9 m of RG174 cable.

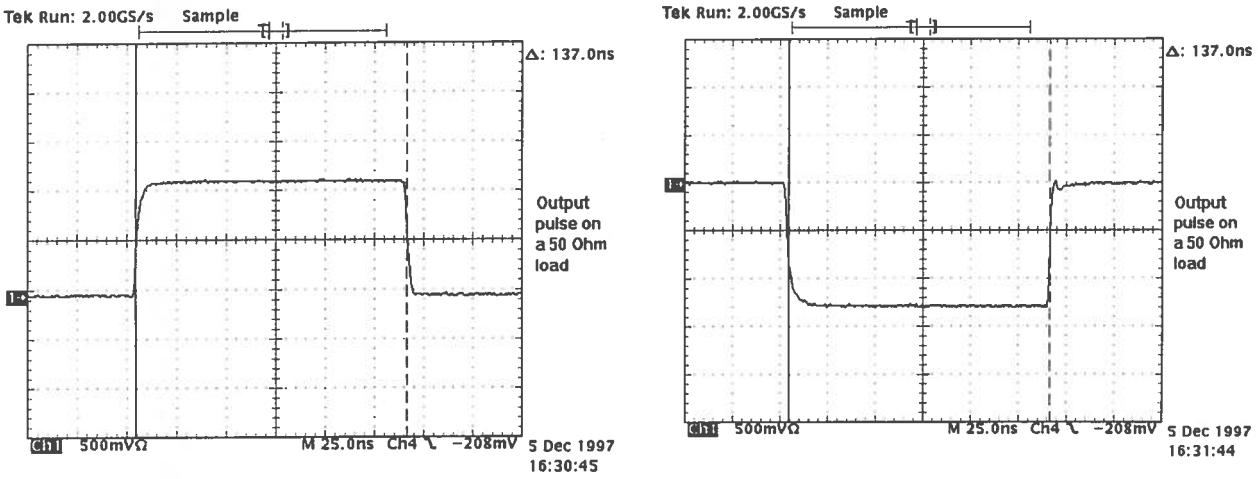

Fig.11: Maximum amplitude output pulses (on a 50 Ohm load)

7. SUMMARY

In this note we have described the requirements and the design of the multichannel analog pulse generator board (ICB) developed for test and calibration of the Front End Electronics of the IFR subdetector of the BaBar experiment. In particular we have shown details both of the digital (interfacing to the DAQ) and the analog section of the module.

The tests on the bench of the board have demonstrated successful operation up to a clock period of 12 nsec.

REFERENCES

[1] R. Santonico and R. Cardarelli, Nucl. Instr. and Meth. **187**, 337 (1987).

[2] F. Anulli et al., "The muon and K_L detector for the BaBar experiment: physics requirements, final design and start of construction", INFN/AE - 97/30.

[3] Letter of Intent for the Study of CP Violation and Heavy Flavour Physics at PEPII, BaBar collaboration, SLAC Report SLAC-433, June 1994.

[4] BaBar Technical Design Report, BaBar Collaboration, SLAC Report SLAC-R-95 457, March 1995.

[5] N. Cavallo et al., "A possible front-end readout scheme for the resistive plate chamber detector at BaBar", INFN/TC - 95/07.

[6] N. Cavallo et al., "Front-end card design for the RPC detector at BaBar", INFN/TC - 96/22.

[7] A. Anastasio et al., "A positive strip readout card for RPC detectors", INFN/TC - 97/20.

[8] F. Fabozzi and P. Parascandolo, "Design of a high throughput FIFO board", INFN/TC - 97/09.

[9] BaBar Note 281, BaBar DAQ Group, Draft 4/15/96.

[10] F. Fabozzi and P. Parascandolo, "Operation and performances of a high speed dynamic pattern detector", INFN/TC - 97/04.

[11] "MACH 1, 2, 3, and 4 Family Data Book", AMD, 1995.

APPENDIX A

Abel file of Icbdec PLD

```
module icbdec17
title 'Command recognizer for ICB';
icbdec17 device 'mach210a';

"INPUTS
CLK, DIN, RST, PWRUP                    pin 13,9,33,11;
RSPC                                    pin 32;
" ------------------------------------------------------------------------
"OUTPUT
Y0, Y1, Y2, Y3, Y4, Y5,Y6               node  istype 'reg_d, buffer';
Y7,Y8,Y9,Y10, Y11                       node istype 'reg_d, buffer';
NOP                                     pin 6 istype 'reg_d, buffer';
NWCAS                                   pin 8 istype 'reg_d, buffer';
CAS                                     pin 2 istype 'reg_d, buffer';
RDC,RDP                                 pin  26,24 istype 'reg_d, buffer';
VC                                      pin 3 istype 'reg_d, buffer';
FC                                      pin 5 istype 'reg_d, buffer';
FSU                                     pin 25 istype 'reg_d, buffer';
GCD, GCDD, GCDDD                        node  istype 'reg_d,  buffer';
RS_OK                                   pin  39 istype 'reg_d, invert';
WR, RD, WRRD                            node  istype 'reg_d,  buffer';
WRP,WRC                                 pin  28,30 istype  'reg_d, buffer';
NWRC                                    pin 27 istype 'reg_d, invert';
RS_F                                    pin 4 istype 'reg_d, buffer';

"-------------------------------------------------------------------------

X = .x.;
C = .C.;
k = .k.;
Z = .Z.;
YY    =     [Y10,Y9,Y8,Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0,DIN];
KF    =     [RDP,RDC,CAS,WRP,WRC,NOP];
YT    =     [Y11..Y0];

equations
 YT.clk  =    CLK;
 KF.clk  =    CLK;
[VC.clk,FC.clk,GCD.clk,GCDD.clk,GCDDD.clk,RS_OK.clk,RS_F.clk]  = CLK;
[WR.clk,RD.clk,WRRD.clk,NWCAS.clk,NWRC.clk,FSU.clk] = CLK;
 KF.ar      = !RST;
[VC.ar,FC.ar,GCD.ar,GCDD.ar,GCDDD.ar,FSU.ar]       = !RST;
[WR.ar,RD.ar,WRRD.ar,NWCAS.ar,NWRC.ar,RS_F.ar]     = !RST;

YT    :=     (YY & !(Y10&!(WR # RD))) & RSPC & !RS_F.fb;
```

```
NOP   :=    (!Y6 & Y5 & !Y4 & !Y3 & !Y2 & !Y1 & !Y0
            & !VC & !FC.fb)     #       NOP.fb;

CAS   :=    (!Y6 & Y5 &  Y4 & !Y3 & Y2 &  !Y1 & !Y0
            & !VC & !FC.fb)     #       CAS.fb;

WR    :=    (!Y6 & Y5 & !Y4 &  !Y3 & !Y2 & !Y1 & Y0
            & !VC & !FC.fb)     #       WR.fb;

RD    :=    (!Y6 & Y5 & Y4 & !Y3 & !Y2  & Y1 & Y0
            & !VC  & !FC.fb)    #       RD.fb;

WRRD :=     WR # RD;


VC    :=    ( (NOP.fb # CAS.fb # WR.fb # RD.fb) & Y6) # VC.fb ;
FC    :=    (!(NOP.fb # CAS.fb # WR.fb # RD.fb) & Y6) # FSU.fb # FC.fb ;

FSU   :=    (!(WRP.fb # WRC.fb # RDP.fb # RDC.fb) & Y8 & !NOP.fb & !CAS.fb) # FSU.fb;

WRP   :=    (WR & VC & Y0 & !Y1 & Y2 & !Y3 & !Y4
            & !WRC & !FSU.fb) # WRP.fb;
WRC   :=    (WR & VC & !Y0 & Y1 & Y2 & !Y3 & !Y4
            & !WRP & !FSU.fb) # WRC.fb;
RDP   :=    (RD & VC & Y0 & !Y1 & Y2 & Y3 & !Y4
            & !RDC & !FSU.fb) # RDP.fb;
RDC   :=    (RD & VC & !Y0 & Y1 & Y2 & Y3 & !Y4
            & !RDP & !FSU.fb) # RDC.fb;

NWRC  :=  !((WR & VC & !Y0 & Y1 & ! WRP) # WRC.fb);

GCD   :=    (VC # FC.fb) & !WRRD;
GCDD  :=    GCD;
GCDDD :=    GCDD # FSU;


RS_OK  :=   !GCDDD & RSPC & !PWRUP;

NWCAS  := CAS.fb & !Y11 &!Y10 & GCDD;

RS_F  := FSU.fb & !Y11 & !Y10;

end
```

APPENDIX B

Abel file of Sh_ctr PLD

```
module sh_ctr12;
title 'Shift registers controller for ICB';
sh_ctr12 device 'mach210a';

"INPUTS
CK60                                pin 13;
NWRC                                pin 35;
DIN                                 pin 8;
WRP, WRC, RDP, RDC, PWRUP           pin 32,11,10,4,33;
SC                                  pin 6;
" ------------------------------------------------------------------------
"OUTPUT
Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7      node  istype 'reg_d, buffer';
Y8, Y9, Y10, Y11,Y12,Y13,Y14,Y15    pin 39,43,41,38,40,42,37,36 istype 'reg_d, buffer';
K0, K1, K2, K3, K4, K5, K6, K7      pin 29,17,27,25,30,28,26,24 istype 'reg_d, buffer';
C0, C1, C2, C3                      node istype 'reg_d, buffer';
C4, C5, C6                          node istype 'reg_d, buffer';
EN_WP                               pin 15 istype 'reg_d, buffer';
EN_WC                               pin 21 istype 'reg_d, buffer';
EN_RP                               pin 20 istype 'reg_d, buffer';
EN_RC                               pin 19 istype 'reg_d, buffer';
RSPC                                pin 2 istype 'reg_d, invert';
WRRD_P                              node istype 'reg_d, buffer';
EN_REC                              pin 18 istype 'reg_d, buffer';
ST_REC                              node istype 'reg_d, buffer';
DIS_REC                             node istype 'reg_d, buffer';
"------------------------------------------------------------------------
X = .x.;
C = .C.;
k = .k.;
Z = .Z.;
YY      =       [Y14..Y0];
YT      =       [Y15..Y1];
YK      =       [Y7..Y0];
KK      =       [K7..K0];
COUNTER = [C6..C0];

equations
YT.clk          = CK60;
WRRD_P.clk = CK60;
EN_WP.clk    = CK60;
EN_WC.clk    = CK60;
EN_RP.clk    = CK60;
EN_RC.clk    = CK60;
EN_REC.clk  = CK60;
ST_REC.clk  = CK60;
```

```
DIS_REC.clk  = CK60;
RSPC.clk     = CK60;
Y0.clk       = CK60;
KK.clk       = NWRC;

Y0.ar        = PWRUP;
YT.ar        = PWRUP;
WRRD_P.ar    = PWRUP;
EN_WP.ar     = PWRUP;
EN_WC.ar     = PWRUP;
EN_RP.ar     = PWRUP;
EN_RC.ar     = PWRUP;
EN_REC.ar    = PWRUP;
ST_REC.ar    = PWRUP;
DIS_REC.ar   = PWRUP;
RSPC.ar      = PWRUP;
KK.ar        = PWRUP;

Y0 := DIN & (EN_WC == 1) #
      SC & (EN_RC == 1) #
      Y0.fb & ((EN_WC # EN_RC) == 0);

YT := YY.fb & ((EN_WC # EN_RC) ==1) #
      YT.fb & ((EN_WC # EN_RC) == 0);

COUNTER.clk = CK60;
COUNTER.ar= PWRUP;
COUNTER:= (COUNTER + 1) & (EN_WP.fb # EN_WC.fb # EN_RP.fb # EN_RC.fb) & RSPC.fb;

WRRD_P := (WRP # RDP # WRC # RDC) & RSPC.fb;

EN_WP := WRRD_P & WRP & RSPC.fb;
EN_WC := WRRD_P & WRC & RSPC.fb;
EN_RP := WRRD_P & RDP & RSPC.fb;
EN_RC := WRRD_P & RDC & RSPC.fb;

EN_REC:= WRRD_P & RDP & !DIS_REC;

RSPC := !((C6 & !C5 & C4 & C3 & C2 & C1 & !C0 & WRP)  #
      (!C6 & !C5 & !C4 & C3 & C2 & C1 & !C0 & WRC)  #
      (C6 & C5 & C4 & C3 & C2 & C1 & !C0 & RDP)  #
      (!C6 & C5 & !C4 & C3 & C2 & C1 & !C0 & RDC)) ;

ST_REC := C6 & !C5 & C4 & C3 & C2 & !C1 & C0 & RDP;

DIS_REC := (ST_REC # DIS_REC) & RSPC;

KK := YK;

end
```