DOCUMENT TYPE:	**TECHNICAL NOTE**
TITLE:	**GRID BOARD FOR THE TEST EQUIPMENT OF THE MINICALORIMETER PROTOTYPE**
DOCUMENT Ref. No.:	AGILE-ITE-TN-011 **N° OF PAGES:** i-viii, 61
ISSUE No.:	02 **DATE:** December 2002 IASF section of Bologna Report 354/02
PREPARED BY:	A. BULGARELLI, M. PREST, E. VALLAZZA, E. CELESTI, F. GIANOTTI, C. LABANTI, M. TRIFOGLIO
CHECKED BY:	M. TRIFOGLIO
SUBSYSTEM MANAGER:	M. TRIFOGLIO

APPROVED BY:

SUBSYSTEM LEADER:	G. DI COCCO	**DATE:**
PROJECT LEADER:	M. TAVANI	**DATE:**
PAYLOAD MANAGER:	A. ZAMBRA	**DATE:**
PAPM:	A. BERNABEO	**DATE:**
CONFIGURATION:	C. MANGILI	**DATE:**

DISTRIBUTION LIST

POS.	NAME	DEPT.	N° of Copies	FULL COPY
1	M. Tavani	IFC MI	1	1
2	G. Cafagna	LABEN	1	1
3	G. Di Cocco	IASF BO	1	1
4	C. Labanti	IASF BO	1	1
5	E. Celesti	IASF BO	1	1
6	M. Trifoglio	IASF BO	1	1
7	F. Gianotti	IASF BO	1	1
8	A. Bulgarelli	IASF BO	1	1
9	E. Vallazza	INFN TS	1	1
10	M. Prest	INFN TS	1	1
11	T. Frøysland	INFN ROMA	1	1
		LABEN		

CHANGE RECORD

Issue	Date	Page	Description of Changes	Release
01	18.09.02	All	First issue of the document	1
02	13.01.03		New version due to the new version of the Altera VHDL code after the test, as reported in the test report	

Printed date: 20 September 2019

SUMMARY

INDEX OF TABLES

INTRODUCTION

The GRID VME board is used to store bars energy information coming from the MCAL-FEE in case of a GRID event, and to retrieve time information from the Burst board, in order to add it to the data collected. This data is then sent to a computer through a VME bus.
The board is also provided with a test input that enables it to be controlled via software and used independently from the MCAL-FEE for debug purposes

REFERENCE DOCUMENTS

1. T. V. Frøysland, 'Test equipment for MCAL FEE in BURST mode', 12-2001.
2. L. Nicolini, 'Agile MCAL_TE – MCAL front end electrical I/F', 02-2002.
3. M. Trifoglio et al., 'Design report of the Proto MCAL Test Equipment for the Agile Minicalorimeter SEM model', AGILE-ITE-RE-001, Tesre report 336/02, 02-2002.
4. E. Celesti et al., "*BURST board for the test equipment of the Minicalorimeter prototype*", AGILE-ITE-TN-007 issue 01, 02-2002
5. A Bulgarelli et al., "*HK/CONF board for the test equipment of the Minicalorimeter prototype*", AGILE-ITE-TN-009 issue 01, 09-2002
6. B. P. Douglass, *Real Time UML Second Edition*, Addison Wesley, 2000
7. DESIGN REPORT OF THE PROTO MCAL TEST EQUIPMENT FOR THE AGILE MINICALORIMETER SEM MODEL, A.Bulgarelli et al., AGILE-ITE-RE-001, February 2002, Te.S.R.E. Report 336/02

LIST OF ACRONYMS

DFE	Digital Front End
FEE	Front End Electronic
HC	Host Computer
MCAL	Minicalorimeter
PLD	Programmable Logical Device
PD	Photodiode
VHDL	Very High Speed Integrated Circuit (VHSIC) Hardware Description Language

1. BOARD ARCHITECTURE OVERVIEW

A scheme of the board is shown in Figure 1. The board is composed of:
One Altera PLD which:

- Manages the grid events, compound of bar energy and timing information;
- Generates simulated burst data to the BURST board;
- Sends the START_CONV signal to the MCAL DFE.

Others component of the board are:

- 4 connectors:
  - 1 VME connector
  - 1 Input connector that receives data from FEE.
  - 1 Output connector that sends signals to FEE and test signals.
  - 1 Test connector
- One memory unit for data storage (60 words of energy information and 2 words of On Board Time information).
- 'LVDS to TTL' and 'TTL to LVDS' converters.
- One dedicated buffer that stores the data to/from the VME bus.
- Two selectors that specify the base address of the board, with the relative address comparator.

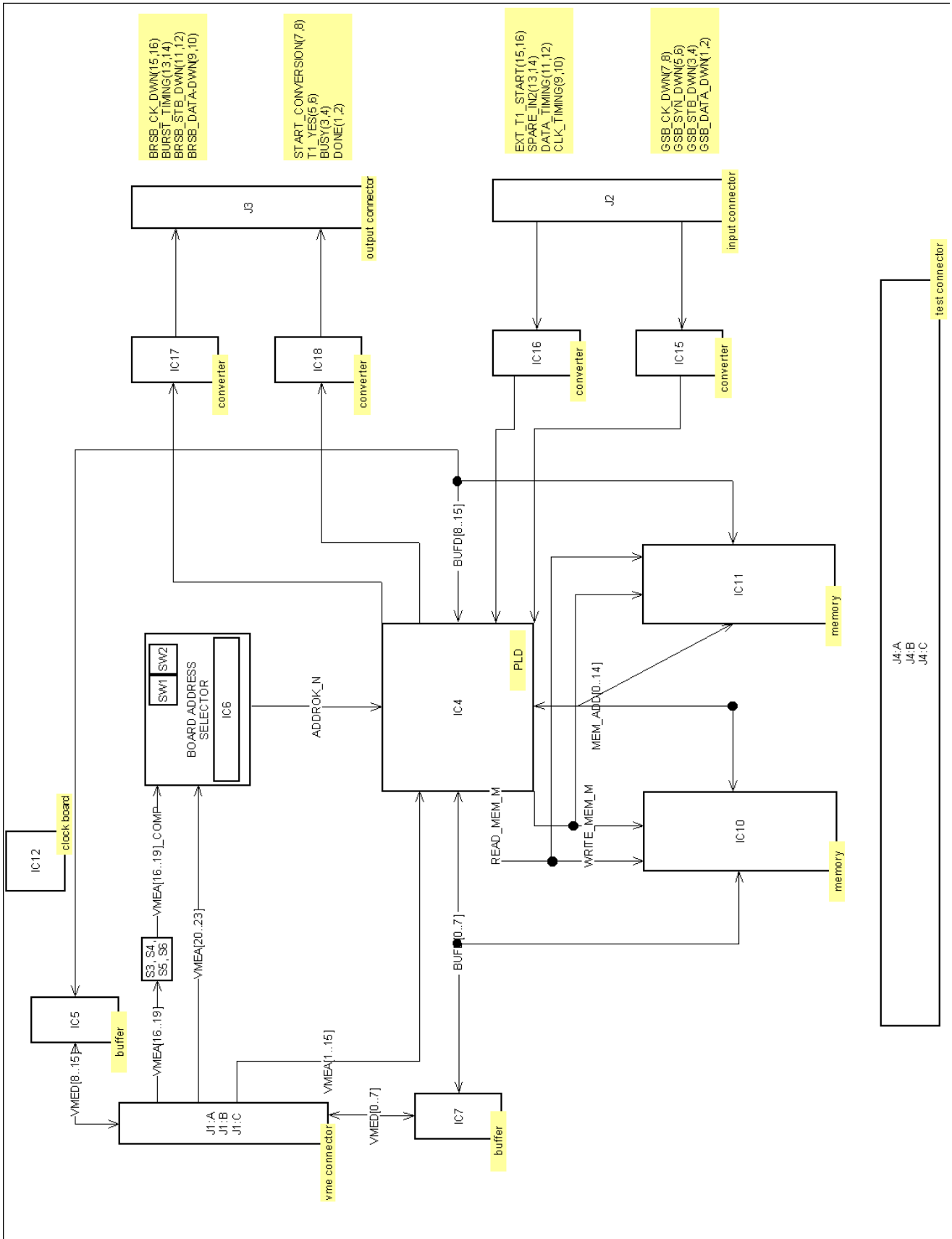

Figure 1: GRID Board Overview

1.1 SYSTEM OVERVIEW

Below is shown an image of the VME crate with the GRID board card installed. PLDs and connectors position is indicated.

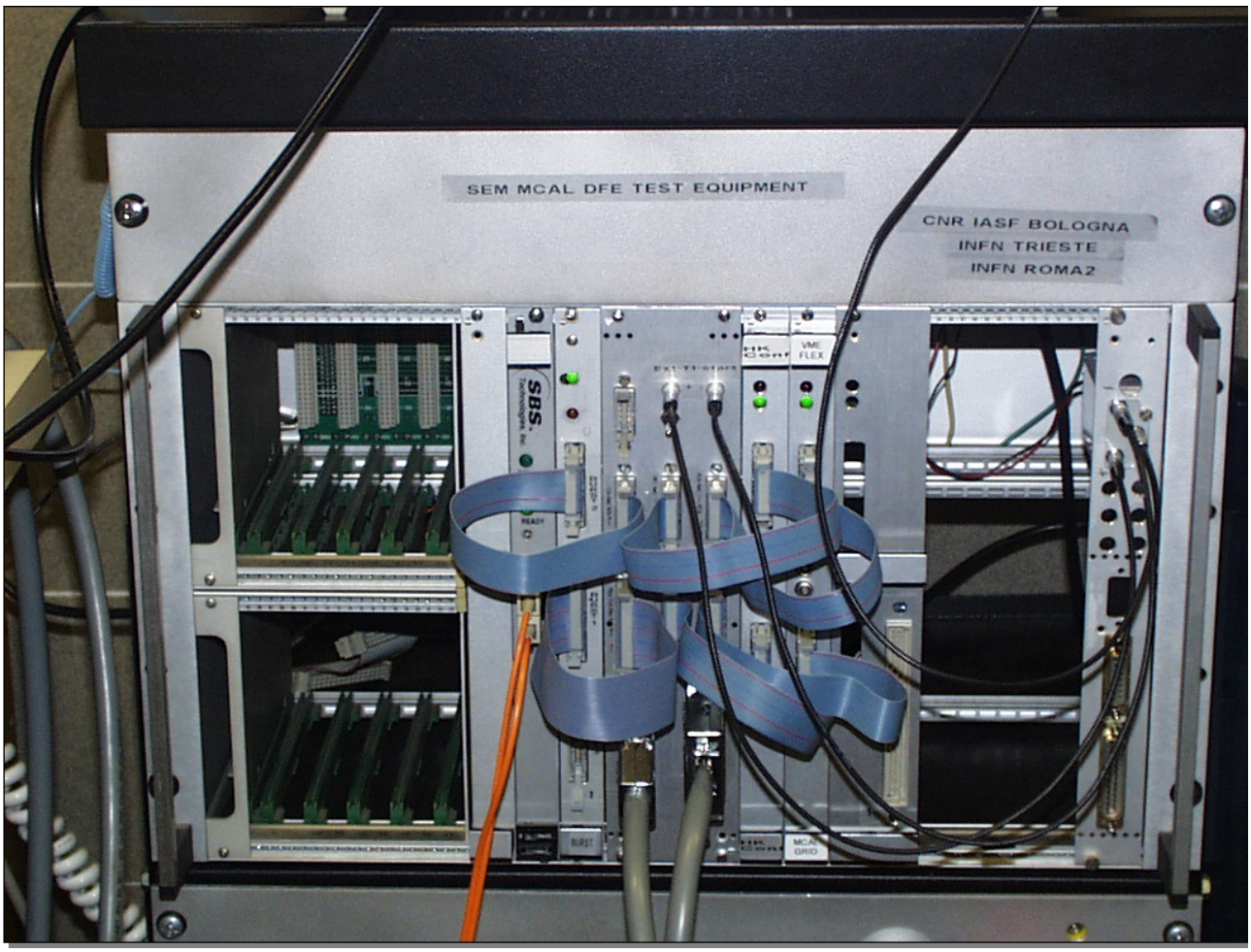

Figure 2: the TE

2. FUNCTIONALITY

2.1 TRIGGERING OF THE GRID EVENT

In order to trigger a new GRID event on the MCAL FEE, the GRID board generates and sends to the FEE a T1_YES signal and, after a predetermined delay, a START_CONVERSION signal.
The rising edge of the T1_YES signal is determinate by the rising edge of the T1_START. This is generated by an internal clock (1 KHz) or by an external signal.

2.2 MANAGEMENT OF THE GRID EVENT

When a GRID event is detected on MCAL, data are sent to the board from the FEE on the dedicated GSB (GRID Scientific Bus). Energy corresponding all the 60 bar sides is sent to the GRID board.
Timing information for every GRID event is collected from the BURST board through a dedicated serial bus (see [4]). The clock on the BURST board used to generate timing information has the frequency of 1.25 MHz.
Energy information of all the 60 bar sides (12 x 60 bit) and time information (32 bit) are then put in memory, and eventually sent to the Host Computer over the VME bus.
The board is be able to:

1) acquire the energy of the 60 photo diode;
2) acquire the timing information;
3) send the data of the event to HC;

For the acquisition of the event, the board must:

4) join the energy and timing information and put it into memory.

2.3 GENERATION OF BURST DATA

The board must be able to generate simulated BURST data and to send these data to the BURST board.

3. DATA FORMAT AND SIGNAL INPUT/OUTPUT

3.1 VME SIGNAL INTERFACE

The GRID board has an I/O interface with the VME bus. The signals of this interface are the following:

- VME ADDR[1..15] = vmea[1..15] indicates the internal address of the board. With the vmea[9] it is possible to selet between memory or configuration register;
- VME ADDR[16..23] for the board selection over the VME bus;
- VME DATA = bufd[0..15]. These are the bidirectional buffered VME data signals.

3.2 GRID SERIAL BUS INPUT SIGNALS

Data I/O between the MCAL-FEE and the GRID board is performed over a dedicated LVDS bus named GSB (GRID Serial Bus) to reduce noise contribution; they are converted in TTL signals by dedicated circuits in order to be used on the GRID board.
In the following list is summarized the GRID board input signals coming from the MCAL-FEE over the GSB:

- GSB_DATA_DWN: is the DATA line, 60 words of 12 bit each, input from FEE.
- GSB_STROBE_DWN: active low signal that defines a single PD energy.
- GSB_SYNCH_DWN: active low signal that defines one set of 60 PD energy.
- GSB_CLK_DWN : 5 MHz clock signal coming from FEE.

When the GSB_SYNCH_DWN signal from FEE becomes low, the data corresponding to the 60 bar sides begins to be transferred, 12 bits for each photo diode.

The following diagram shows the timing between these signals.

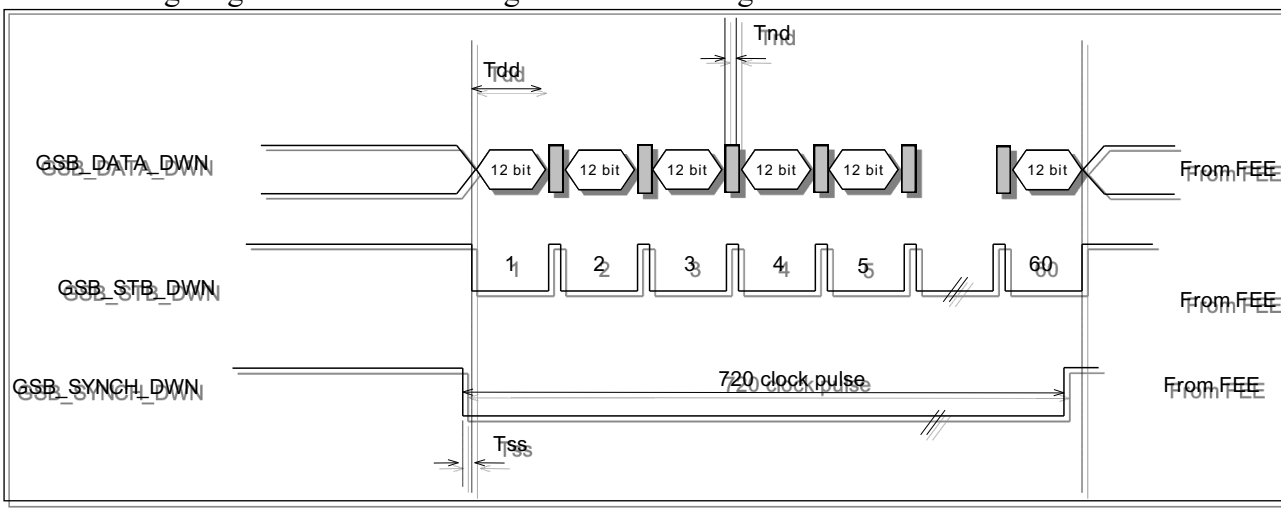

Figure 3: GRID Board signals timing

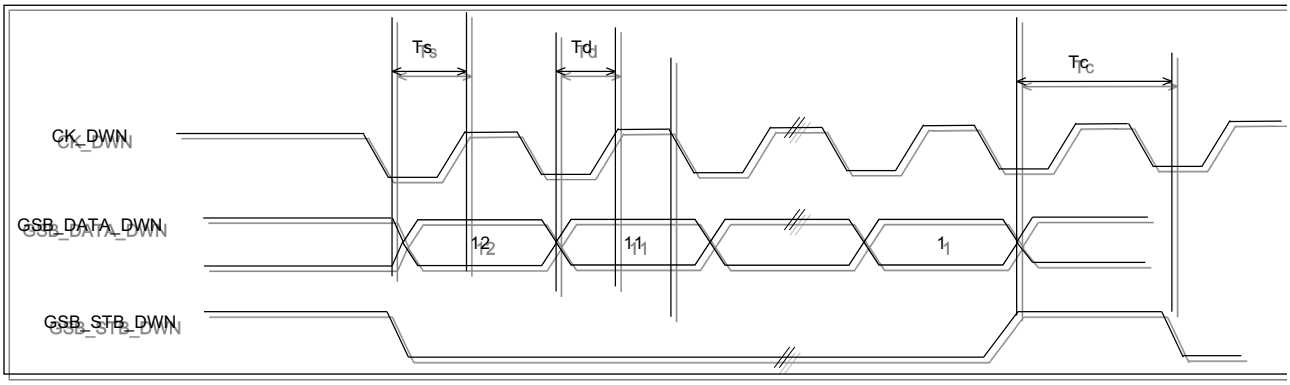

Figure 4: GSB timing diagram

Name	Formula	Min (ns)	Max (ns)	Description
Td	[30,100]	30	100	Data setup time
Ts	[30,100]	30	100	Strobe setup time
Tclk	200	200	200	clock
Tc	200	200	200	
Tnd	200	200	200	1 Tclk
Tdd	12 * Tclk	2400	2400	
Tss	[-30, 30]	-30	30	

Table 1: timing of GSB signals

3.3 OTHER INPUT SIGNALS

3.3.1 CLK_TIMING

This is a serial clock that is driven by the 5 MHz onboard clock and is generated upon arrival of T1_YES signal on BURST board. It is necessary for reading the DATA_TIMING signal.

3.3.2 DATA_TIMING

The DATA_TIMING is a signal of 32 bits of on board time data generated by the BURST board.
By means of CLK_TIMING and DATA_TIMING signals (input for this board) the timing information generated from the BURST board (by On Board Timing Altera PLD) are collected.
When a T1_YES is detected on the BURST board, time information is generated and transferred to the GRID board over this dedicated serial data line. The time is tagged on the rising edge of the T1_YES signal.
After T1_YES arrival, a serial clock driven by the onboard 5 MHz clock is activated and 32 bits of timing data start to be transferred to the GRID board over the serial bus.

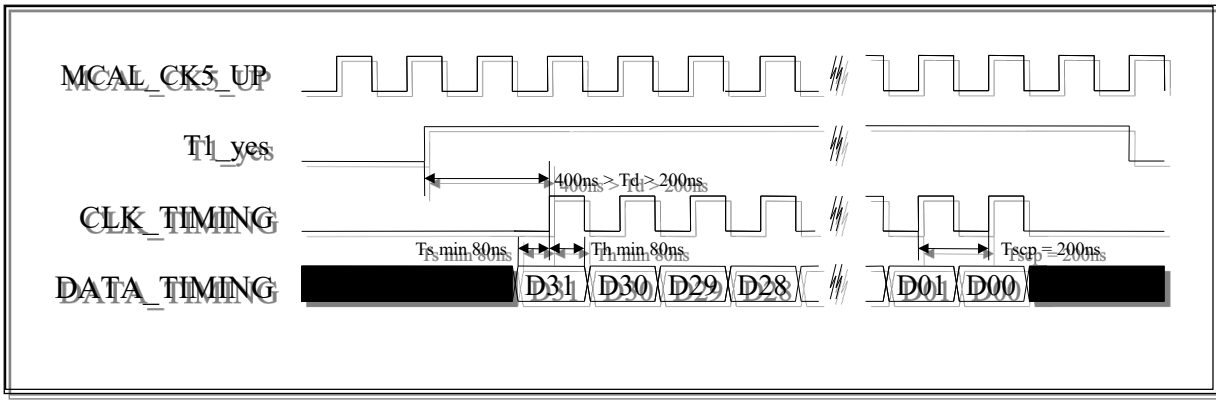

Figure 5: Timing of the serial bus to the GRID board

3.3.3 EXT_T1_START

As mentioned above, this signal is provided with an external equipment in order to generate a T1_START signal (see chapter 5).

3.4 GRID DATA ACQUISITION

3.4.1 THE OVERVIEW

The signal that starts the data acquisition is the **T1_START**, that can be generated by GRID board or by HC. The GRID board shall generate T1_START pulses with a frequency of 1KHz. It is possible to generate a T1_YES signal with the EXT_T1_START signal (coming from external equipment).
After the generation of T1_START the **BUSY** signal goes high. It means, for the HC, the start of the acquisition.
After the generation of T1_START the **GSB_BUSY_UP** signal must be generated.
After the generation of the BUSY, the **T1_YES** goes high. It means that:

- for MCAL FEE it is necessary to sample the 60 PD
- for BURST board is is necessary to send the timing information (with DATA_TIMING and CLOCK_TIMING information) to GRID board.

The duration of the T1_YES (time T1) is defined into the configuration register of GRID board.
Every time a T1_YES is produced, a **START_CONVERSION**[1] pulse is generated after a programmable delay T2 (from 1us to 255us, typical 25us, in the configuration register bit 4-7, see 4.3). After this delay, the data transfer from the MCAL DFE to the GRID board is started by means of the **GSB_DATA_DWN** + **GSB_STROBE_DWN** + **GSB_SYNCH_DWN** + **GSB_CLK_DWN** signals.
Typically the serial data transfer related to the T1_YES will start 1 to 4us after the START_CONVERSION pulse (T3).
It is important to notice that T4 can be positive or negative.

[1] This signal is generated by GRID board, but it may be generated by Silicon Tracker

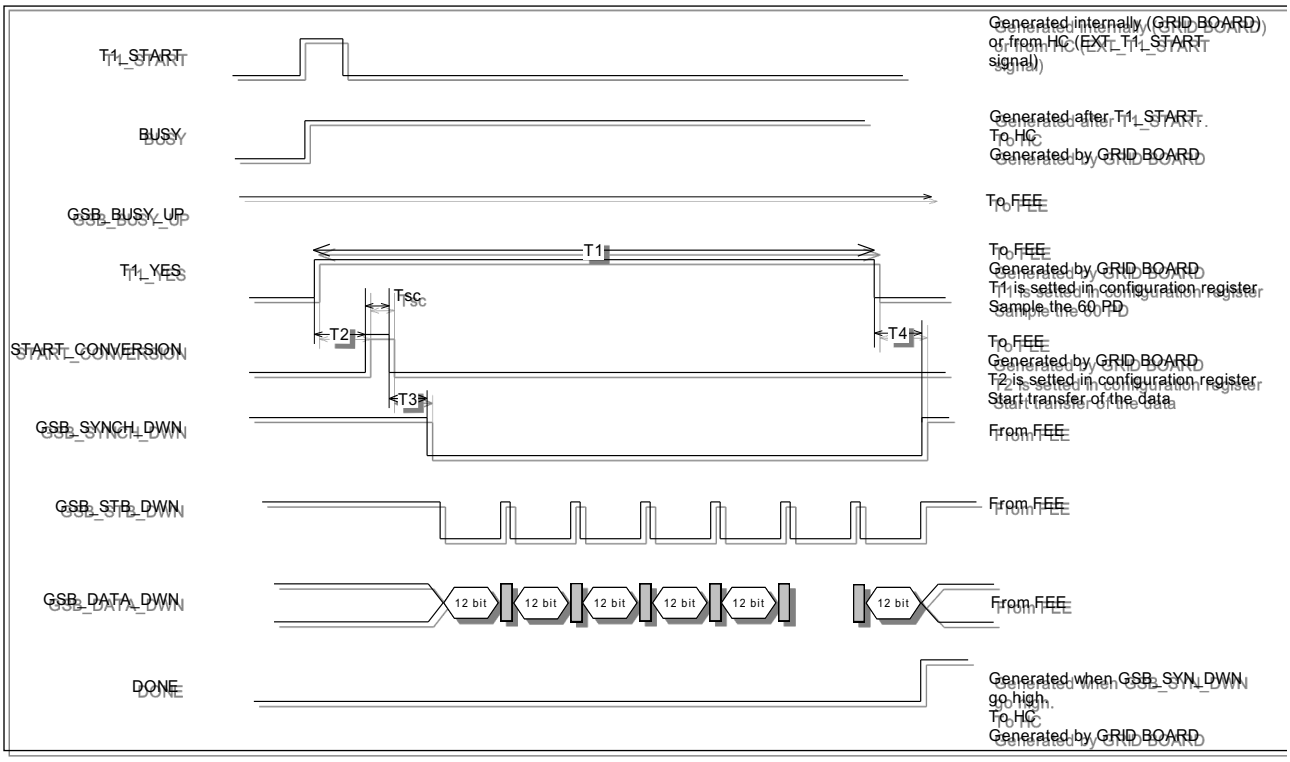

Figure 6: GRID timing diagram

Name	Formula	Min	Max	Description
T1	[0, 255]	0 us	255 us	T1_yes length (programmable)
T2	[0, 255]	0 us	255 us	Start conversion delay (programmable)
T3	[1,4]	1 us	4 us	Data transfer delay
T4				
Tsc	200	200 ns	200 ns	Synchronous with ck5 rising edge

Table 2: timing of output signals

At the end of the acquisition a **DONE** signal is generated by the GRID board.
When both BUSY and DONE are high the HC begins to read data from GRID board memory. When the data transfer is completed the BUSY and DONE signals are reset by the HC. Depending on the setting of the bit 3 on the control register, the board ignores or not the T1_YES signals while BUSY is high.

3.4.2 BUSY

This is an active low signal. For this TE this signal is usually low, but it goes high when a T1_START signal is detected.
The board ignores any other T1_YES signal (coming from the external) when BUSY signal is high (unless bit 3 of the configuration register is set to 0).
The BUSY signal is an indication for the HC of the start of all the process of the data transfer from the MCAL FEE.
This signal is not the BUSY signal of the PDHU.

[This behavior is different of the behavior of the PDHU. This signal is used to stop further downloading of PD energy (i.e. PDHU is busy). When it becomes low, the current 60 energy downloading will be finished and no other energy will be transferred as long as this signal remains low.
On the basis of this, the BUSY signal is set high for representing the PDHU not busy. With this signal high the FEE can read all the 60 photodiode coming for MCAL.]

3.4.3 T1_YES

Can be generated by the GRID board with a rate of 1 KHz (depending on the generation of the T1_START signal) and programmable duration (0-255 μs) or can be generated from the external equipment whit the EXT_T1_START signal.
When this signal his sent to the FEE, the FEE holds the signal of all the electronics chains connected to the PDs (signals are stretched).

3.4.4 START_CONVERSION

Produced by the GRID board every time a T1_YES signal is generated after a programmable delay (0-15 μs with the first version of Altera PLD, 0-255 μs step 16 with the second version of Altera PLD) from T1_YES rising edge.
Duration 200 ns.
When this signal gets to FEE it causes the stretched signal from the PDs to be converted.

3.4.5 DONE

This is an active low signal, goes high when GSB_SYNC_DWN signal goes high. It indicates that all the 60 photodiode are read.

3.4.6 BURST DATA

This signal are used for the data transfer from GRID board to BURST board of the simulated data:

1) **BSB_CK_DWN** (it should be BRSB)
2) **BSB_STB_DWN** (it should be BRSB)
3) **BSB_DATA_DWN** (it should be BRSB)
4) **BURST_TIMING**: with this signal it is possible the communication to FEE that a BURST event is detected.

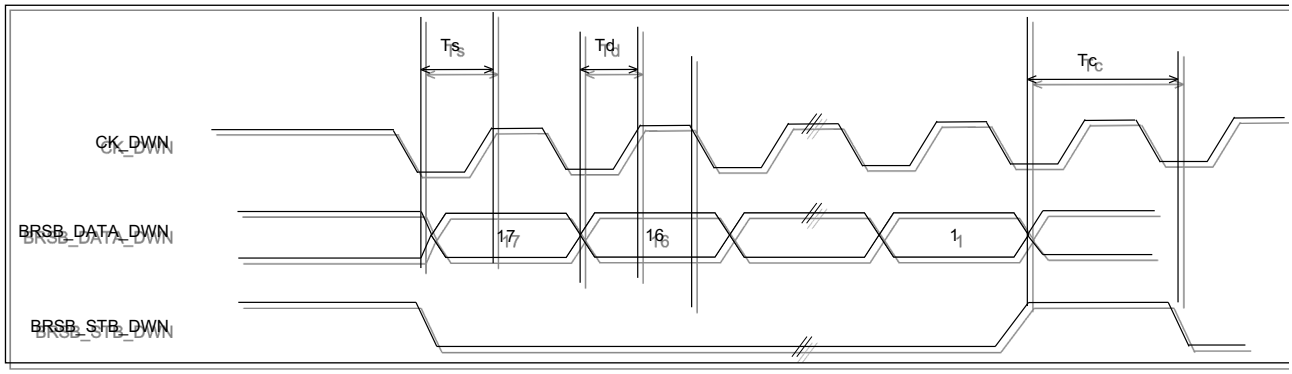

Figura 7: BRSB timing diagram

Name	Formula	Min (ns)	Max (ns)	Description
Td	[30,100]	30	100	Data setup time
Ts	[30,100]	30	100	Strobe setup time
Tclk	200	200	200	clock
Tc	200	200	200	

3.5 FORMAT OF DATA STORED IN THE BOARD MEMORY

MSB															LSB
b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
1	1	1	1	PD energy											
1	1	1	1	PD energy											
1	1	1	1	…											

Table 3: Format of data stored in the board memory

3.6 SERIAL BUS SIGNAL FORMAT

3.6.1 ADDRESS FORMAT FROM HC TO VME

For the communication between the HC and the board, 15 bits are used:

- The bits A1-A8 (8 bits) must contain an address of the memory (when applicable);
- A9 bit is used for working with registers or memory:
  - 0 → working with memory
  - 1 → working with registers
- A10: read the correct configuration register
  - 0 → configuration register 1
  - 1 → configuration register 2 (START_CONV delay)
- A11 bit is used for the reset of the board:
  - 0 → reset
  - 1 → not reset
- A12-A13 (2 bits) are used for the selection of a register (configuration or timing registers):
  - 00 → configuration register

- 01 → timing register 0
- 10 → timing register 1
- 11 → the value read must be 1100101011111110 (0xCAFE)

- A14: with this bit it is possible to read the BUSY and DONE signals:
  - 0 → read memory
  - 1 → read BUSY/DONE

MSB											LSB
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4
NOT USED	BUSY/DONE	REGISTER SELECT	REGISTER SELECT	BOARD RESET	ACCESS TO CONFIGURATION REGISTER 1 OR 2	ACCESS TO CONFIGURATION REGISTER	ADD8	ADD7	ADD6	ADD5	ADD4

Table 4: Address format from HC to board

3.6.2 DATA FORMAT FROM BOARD TO HC

This is the format of the data exchanged between HC and the board.

MSB													LSB
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2
DATA15	DATA14	DATA13	DATA12	DATA11	DATA10	DATA9	DATA8	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2

Table 5: Data format from board to HC

3.6.3 DATA FORMAT FROM BOARD TO FEE

This is the format of the data exchanged between FEE and the board.

MSB											LSB
D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DATA11	DATA10	DATA9	DATA8	DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0

Table 6: Data format from board to FEE

When the GSB_STROBE_DWN signal from FEE becomes low, 12 bits of data are transferred from the FEE to the board, corresponding to energy information of a single bar side.
60 strobe cycles are necessary to transfer a complete set of data.

MSB											LSB
b11	b10	b9	B8	b7	b6	b5	b4	b3	b2	b1	b0
Bar 0 Side A energy (12 bits)											
Bar 0 Side B energy (12 bits)											
…											
Bar 29 Side A energy (12 bits)											
Bar 29 Side B energy (12 bits)											

Table 7: sequence of data between FEE to board

3.7 CONNECTORS PIN FUNCTION

MCAL GRID Board exchanges signals with the MCAL-FEE through the 2 connectors listed below:

Connector J2 (16 PIN) IDC2X8 – INPUT	**Pol.**	**PIN**	**DIR**	**Description**
GSB_DATA_DWN	+	1	IN	Grid Serial Bus: Data coming from FEE
	-	2		
GSB_STB_DWN	+	3	IN	Strobe signal from FEE
	-	4		
GSB_SYN_DWN	+	5	IN	Synchronization signal from FEE
	-	6		
GSB_CK_DWN	+	7	IN	5 MHz clock signal from FEE
	-	8		
CLK_TIMING	+	9	IN	5 MHz clock of the timing line, used for time transfer from the BURST board
	-	10		
DATA_TIMING	+	11	IN	Contains time information of the current GRID event.
	-	12		
SPARE_IN2	+	13	IN	Spare
	-	14		
EXT_T1_START	+	15	IN	Used for the generation of T1_YES signal from external equipment.
	-	16		

Connector J3 (16 PIN) IDC2X8 – OUTPUT	**Pol.**	**PIN**	**DIR**	**Description**
START_CONVERSION	+	1	OUT	Start Conversion signal to FEE
	-	2		
T1_YES	+	3	OUT	T1_YES signal to FEE and to BURST board
	-	4		
T1_YES	+	5	OUT	T1_YES signal to FEE and to BURST board
	-	6		
PDHU_BUSY	+	7	OUT	GSB_BUSY_UP signal
	-	8		
BSB_DATA_DWN	+	9	OUT	Burst Serial Bus: Data toward the BURST board
	-	10		
BSB_STB_DWN	+	11	OUT	Strobe signal to BURST board
	-	12		
BURST_TIMING	+	13	OUT	Burst Timing signal to BURST board
	-	14		
BSB_CK_DWN	+	15	OUT	5 MHz clock signal to BURST board
	-	16		

Table 8: Pin function

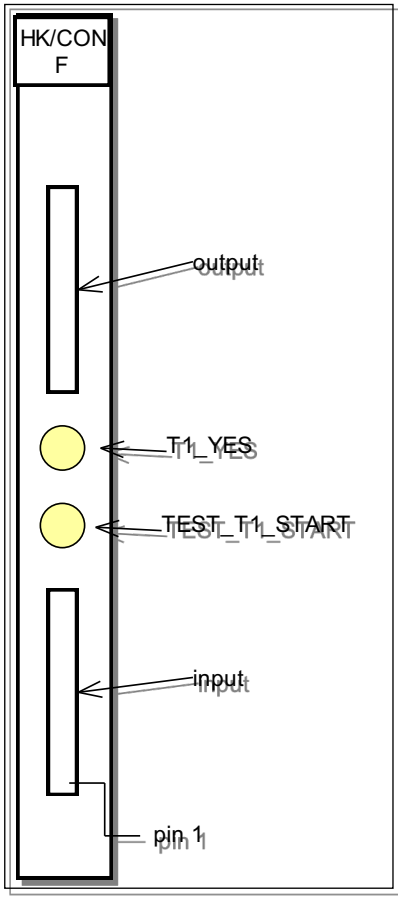

Figure 8: hardware layout of the board

In the next figure is shown the layout of the board.

3.8 ELECTRICAL SCHEMES OF THE BOARD

See Annex.

3.9 BOARD REGISTERS

The board has some registers:

- **Configuration register**: useful for board programming (1→**0x0a00**, 2→(**0x3e00**));
- **Timing registers**: useful for storing timing information of the events.

3.9.1 CONFIGURATION REGISTER 1

The GRID board is driven via software, setting appropriate values on the bits of the configuration register.
The bits of the configuration register used are the following:

- Bit 0: Memory
- Bit 1: Test

- Bit 2: T1_START source
- Bit 3: Busy/Continuous
- Bit 4 – 7: unused
- Bit 8 – 15: T1_YES length

3.9.1.1 BIT 0: MEMORY

Because it is impossible to to read at the same time from memory to VME and from FEE (in reality from an internal shift register to memory) it is necessary to set this bit for the selection of the direction of the data bus.
The values are the following:
0 – Read memory and Write memory are performed from VME bus (**internally**).
1 – Data is read from FEE and then written in memory (**externally**).

3.9.1.2 BIT 1: TEST

Enables/Disables the generation of GRID data output.

3.9.1.3 BIT 2: T1_START SOURCE

Defines whether the T1_START signal must be generated internally on the board or supplied externally:
0 → internally
1 → from TE

3.9.1.4 BIT 3: BUSY/CONTINUOUS

Once the bit 2 is set to 0, this bit determines the generation of the internal T1_START signal. There are two modalities: busy (0), continuous (1). In the busy modality only one T1_START pulse is generated in correspondence of the reset of the board (see below). In the continuous modality the board, after the reset, starts generating the T1_START pulse at the rate of 1 KHz.
Once the bit 2 is set to 1, TBD.

3.9.1.5 BIT 8 – 15: T1_YES LENGTH

T1_YES length must be of at least 7 μs, and always longer then START_CONV delay, as it must be still high when a START_CONV signal is generated.

3.9.2 CONFIGURATION REGISTER 2

With this register it is possible to write the START_CONV delay. These bits contain the delay between T1_YES rising edge and START_CONV rising edge. The range is 0 255 μs:
- bit 0-7: start conversion delay
- bit 8-15: unused

The start conversion signal must be generated before the end of the T1_YES.

3.10 **BASIC OPERATIONS**

The basic operations that it is possible to perform with the board are the following:

1. reset the board
2. read registers
3. write registers
4. read memory

3.10.1 RESET THE BOARD

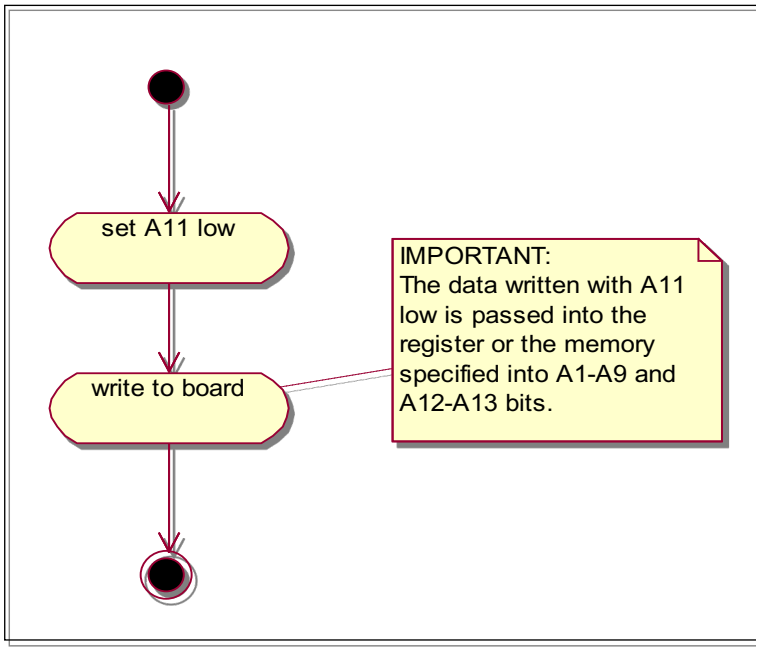

Figure 9: reset the board

A side effect of the board reset is present. The data written with A11 low is passed into the register or the memory specified into A1-A9 and A12-A13 bits.

3.10.2 READ REGISTERS

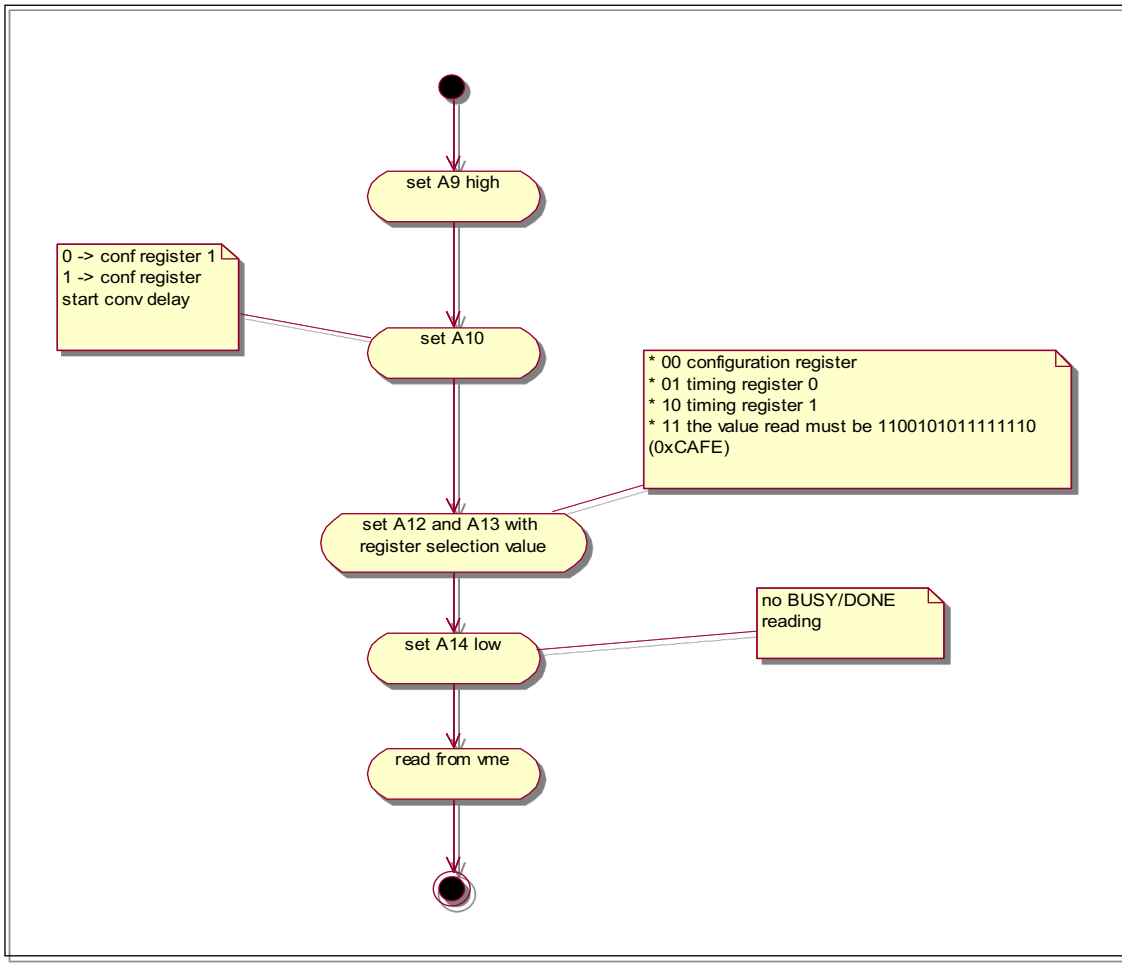

Figure 10: read registers

3.10.3 WRITE REGISTER

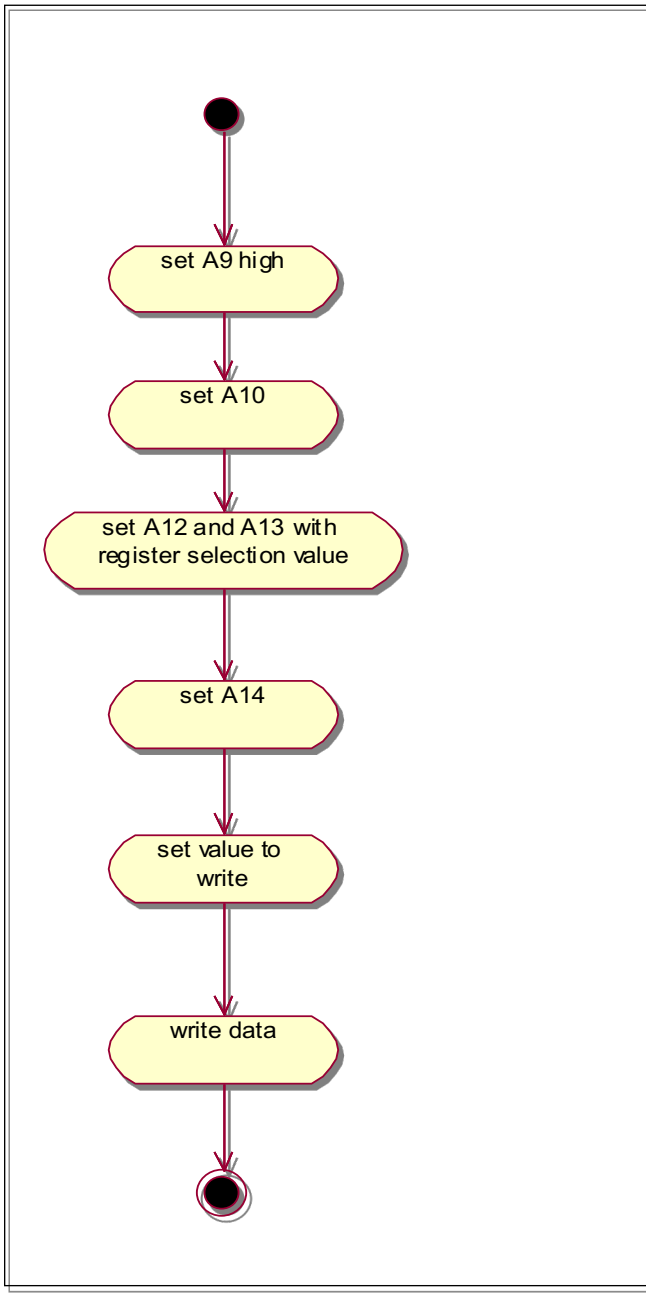

Figure 11: write register

3.10.4 READ MEMORY

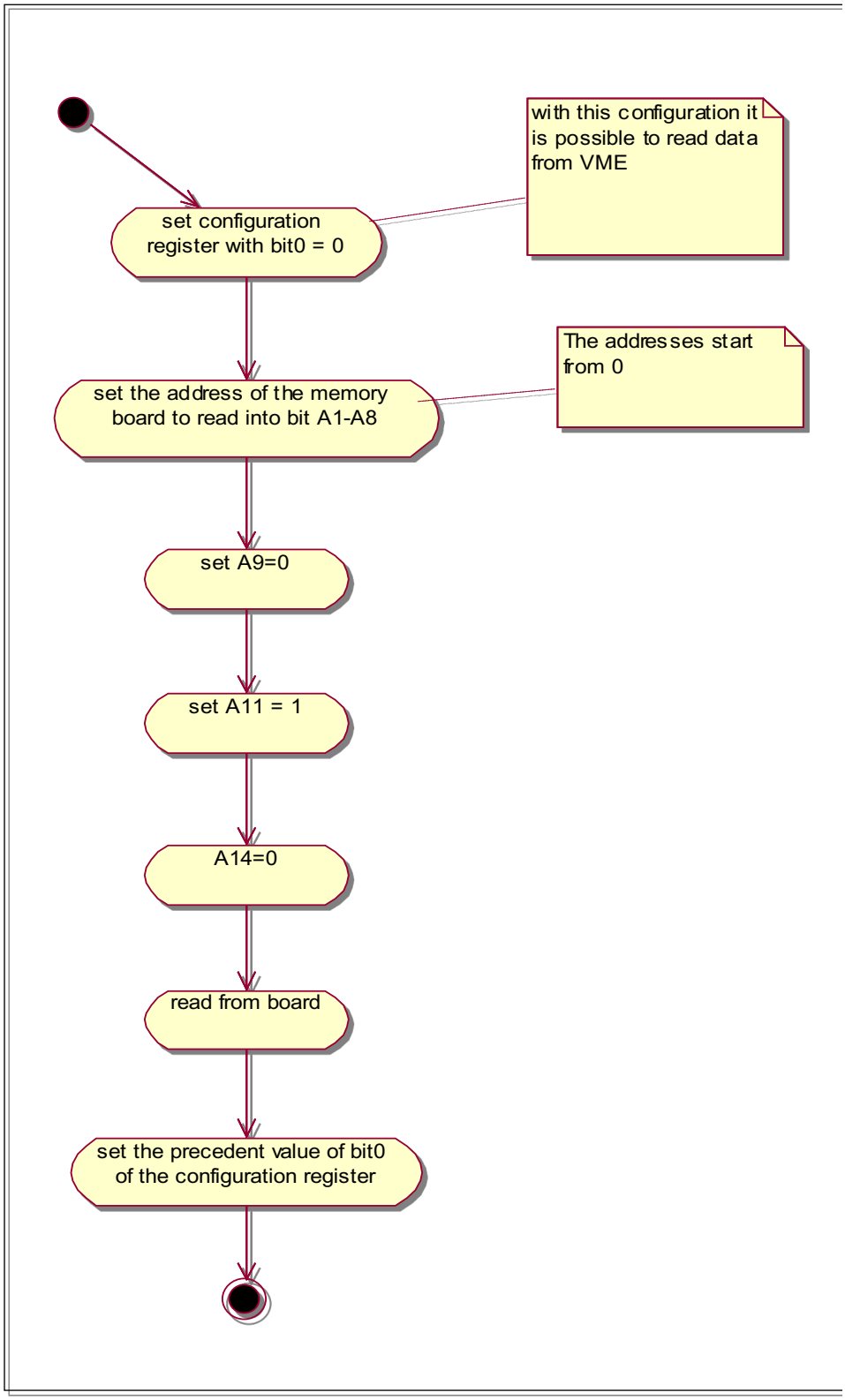

Figure 12: read the board memory

3.11 **BOARD UTILIZATION**

In this paragraph are presented the UML activity diagram for programming of the board. The main operation that it is possible to do with this board are the following:

- read the configuration register
- write the configuration register
- read the timing registers
- reset the board
- read an event from board memory

3.11.1 READ THE CONFIGURATION REGISTER

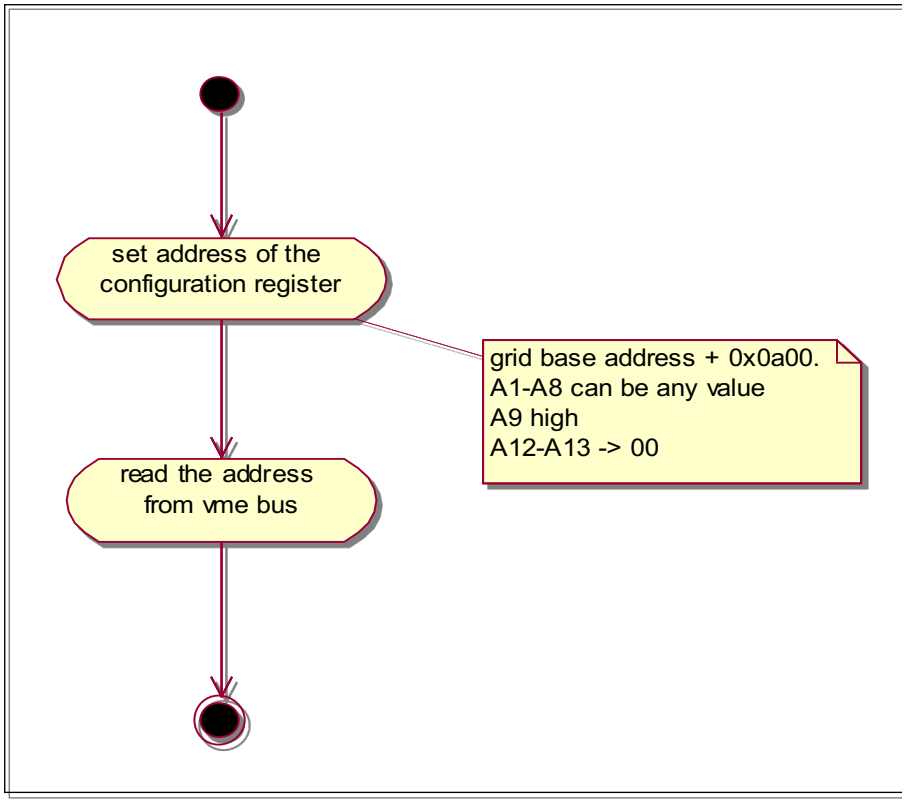

Figure 13: read the configuration register

3.11.2 WRITE THE CONFIGURATION REGISTER 1

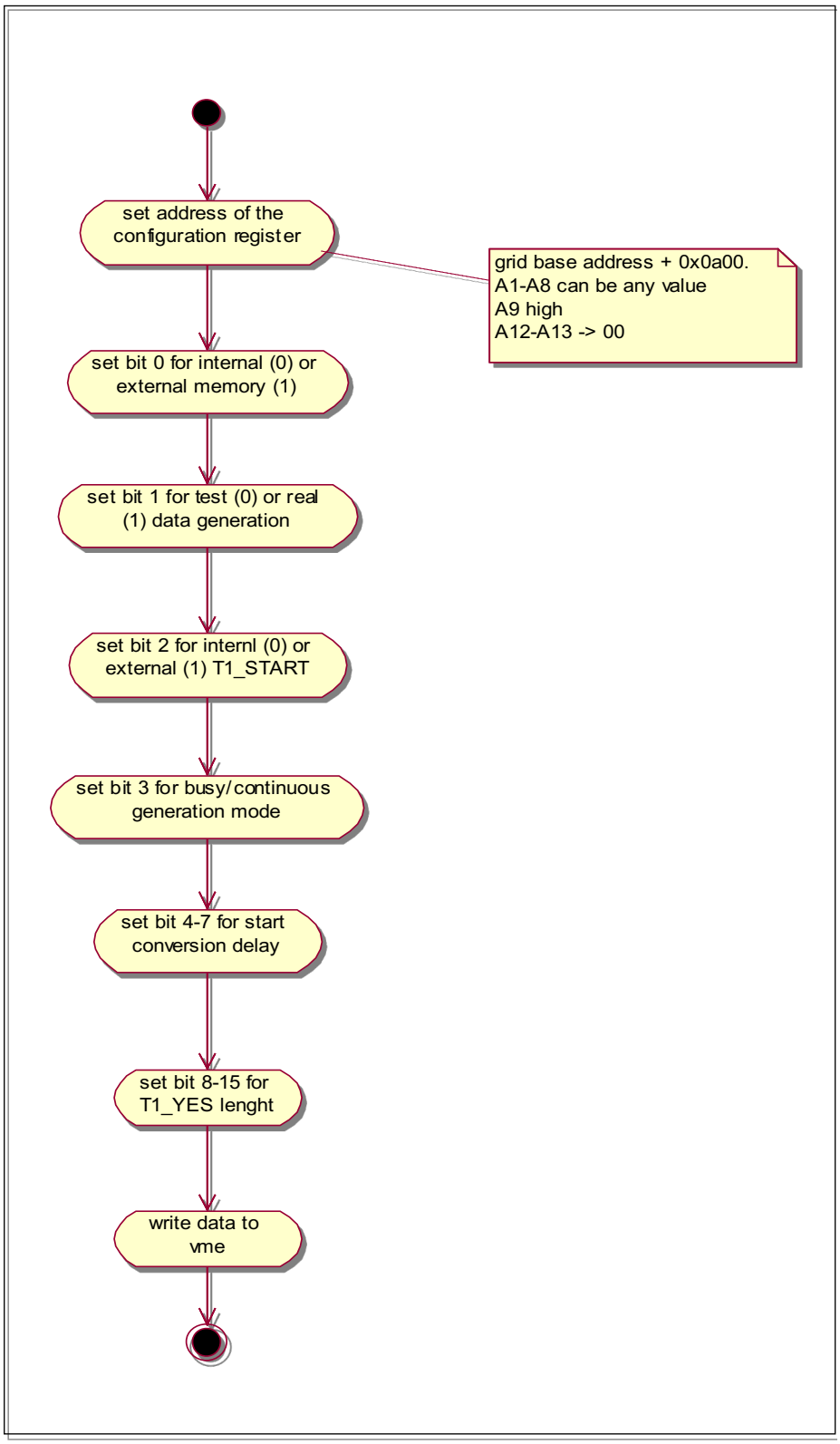

Figure 14: write configuration register

3.11.3 WRITE THE CONFIGURATION REGISTER 2

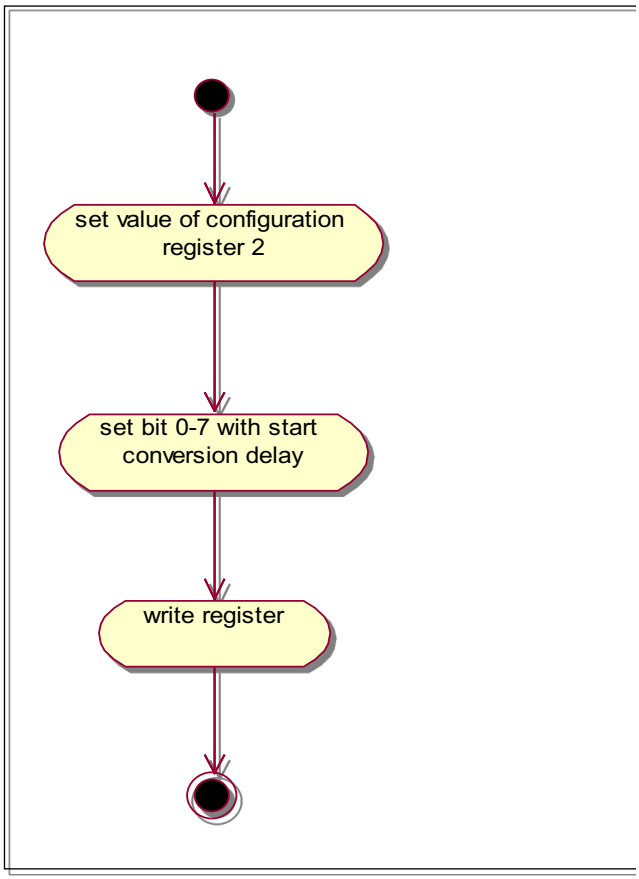

Figura 15:

3.11.4 READ THE TIMING REGISTERS

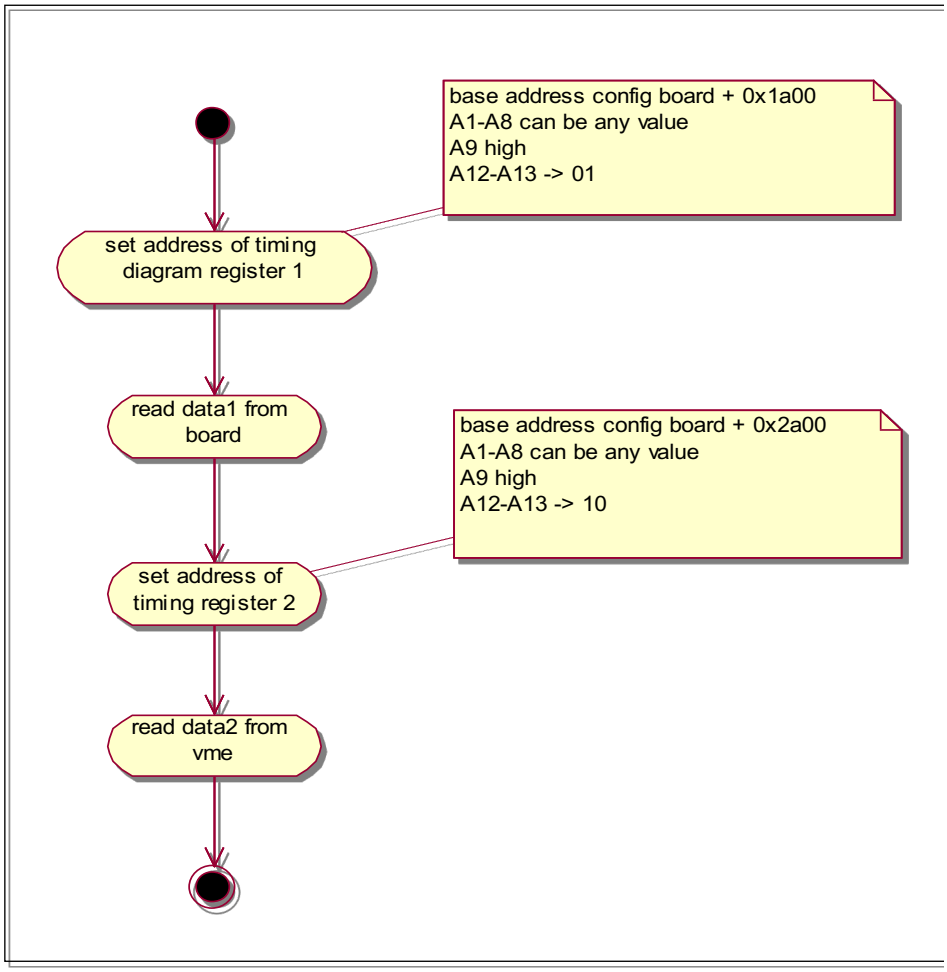

Figure 16: read timing register

3.11.5 RESET THE BOARD

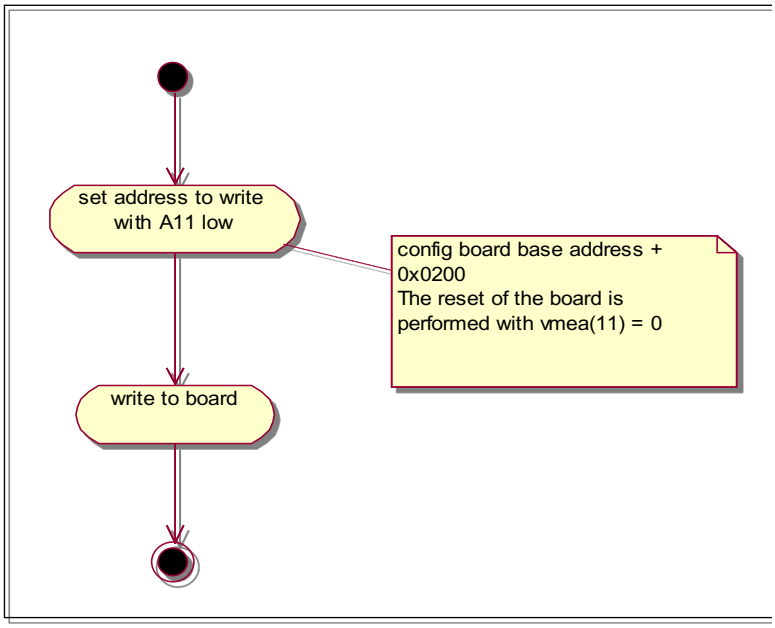

Figure 17: reset the board

3.11.6 READ AN EVENT FROM BOARD MEMORY

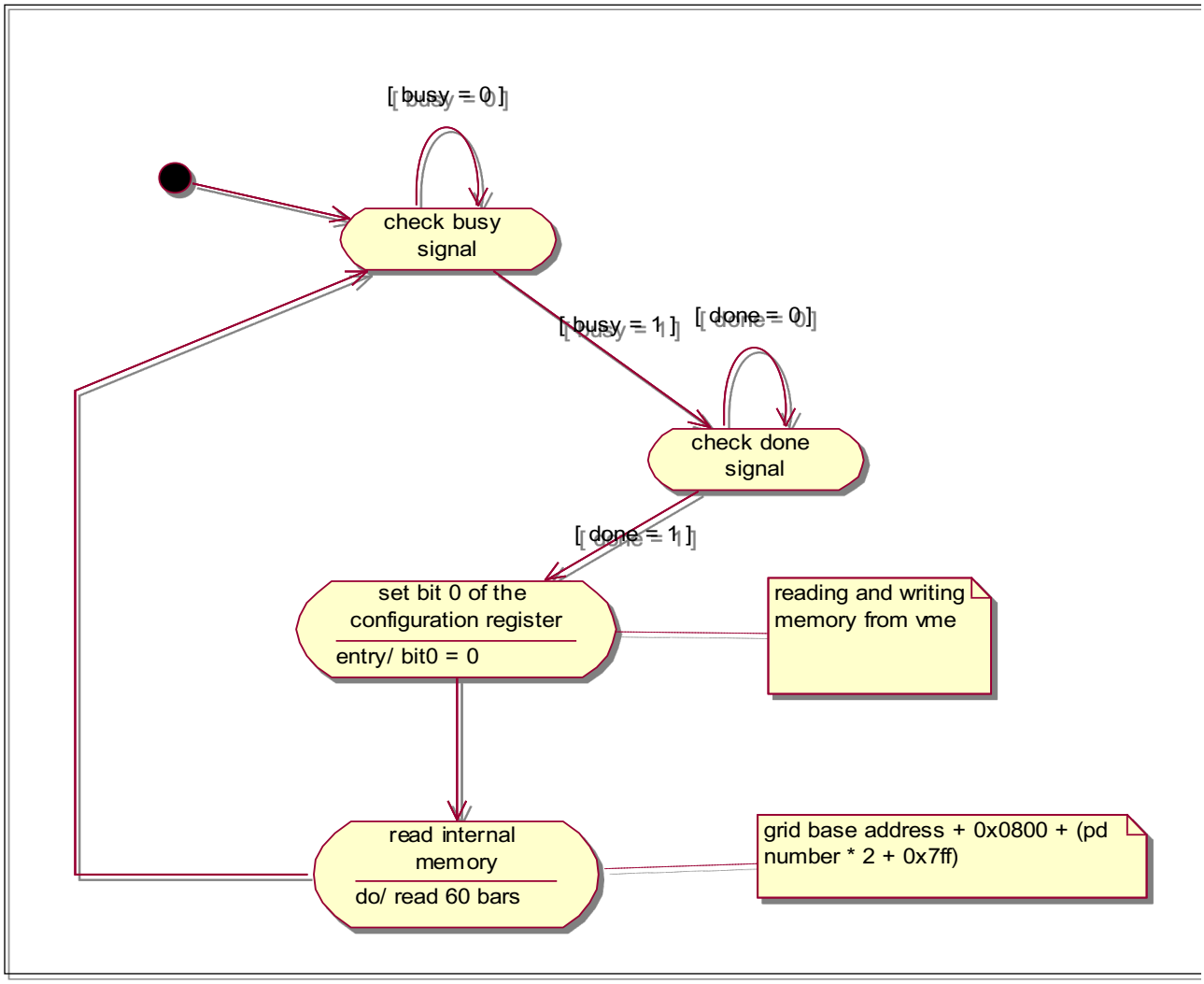

Figure 18: reading event from board

3.12 **STATE CHART DIAGRAM**

3.12.1 CLASS DIAGRAM

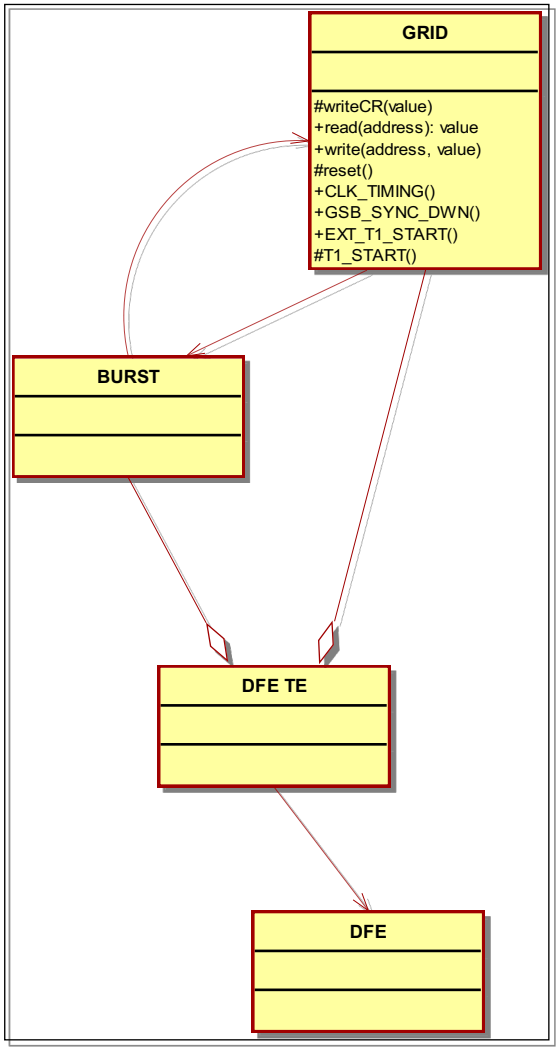

Figure 19: class diagram

3.12.2 STATE CHART

A state chart is a directed graph consisting of state vertices connected by transition. A state is a condition of existence of an object (as the board) that is distinguishable from other conditions of existence (a state is represented by a rounded rectangle). A transition (a line with arrowheads) is the reification of a response of the object to an event when in a particular state.
A transition is described with the following syntax (all the parts are optional):

event-trigger(parameters) [guard condition] / action list

event-trigger is the name of the event triggering the acquisition. The parameters are a comma-separated list containing the names of data parameters passed with the event signal. The guard condition is a Boolean expression that must evaluate true for the transition to be taken. An action list is a comma-separated list of operations executed as a result of the transient being taken.

Into a state can be present some other keywords:

- entry/ action name, an action executed when the object enter in this state
- exit/ action name, an action executed before the object coming out this state
- do/ activity name, an activity executed when the object is in this state

The difference between action and activity is that an action is a short non-interruptible operation. Activity may be performed as long as the state is active and they may be interrupted.
genSignal is a particular action that generates an event for this (if not specified in the parameters list) or other objects.
A state containing other states separated by dashed lines is an *and-state*. This can be useful to represent independent states that may concurrently be active with other states.
A filled circle represents the stat point of a state machine. An empty circle represents a choice-point pseudo-state. A double circle represents the end point of a state machine.
A circle with a number (or *) is a Petri net.

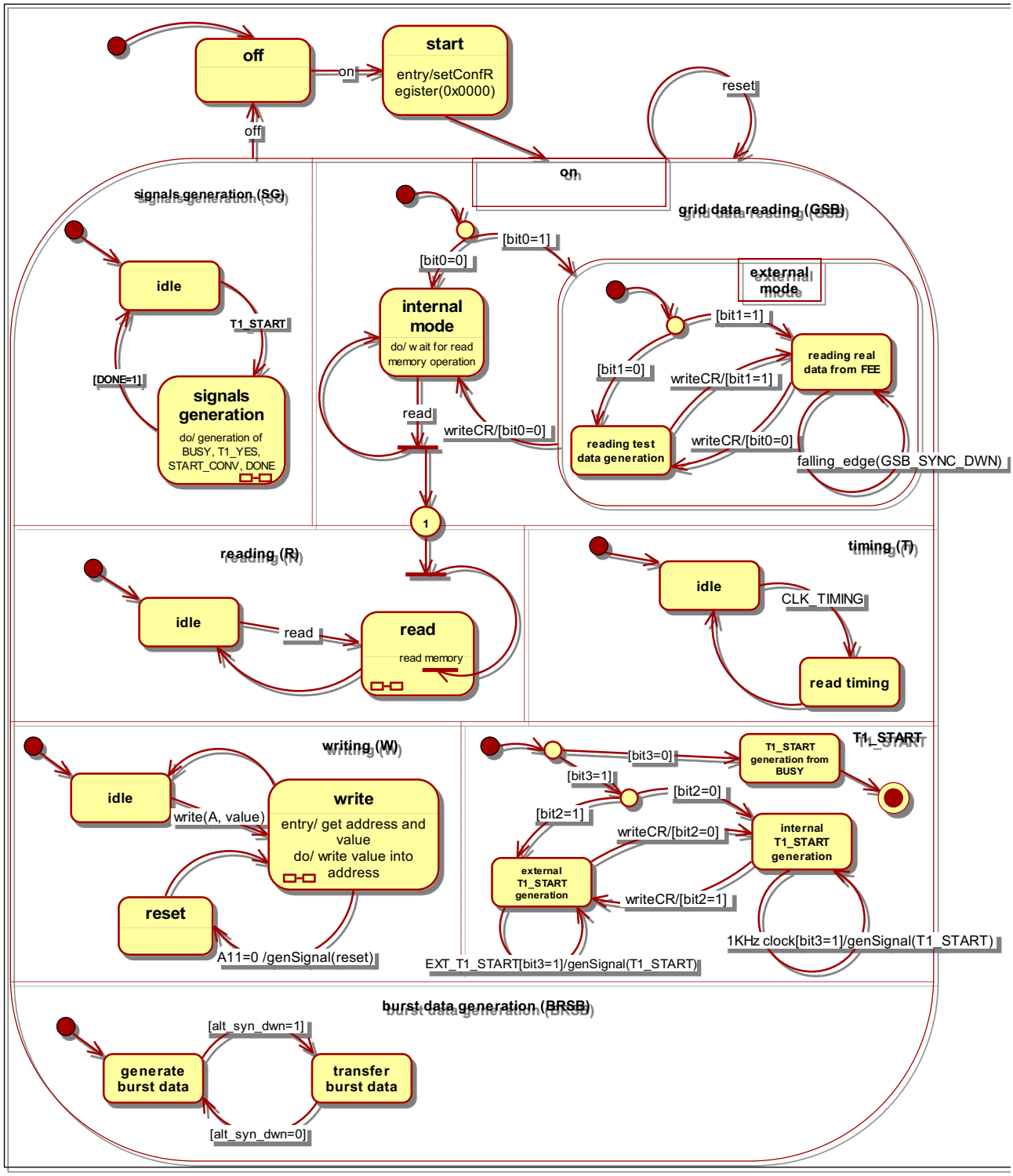

Figura 20: and-state chart

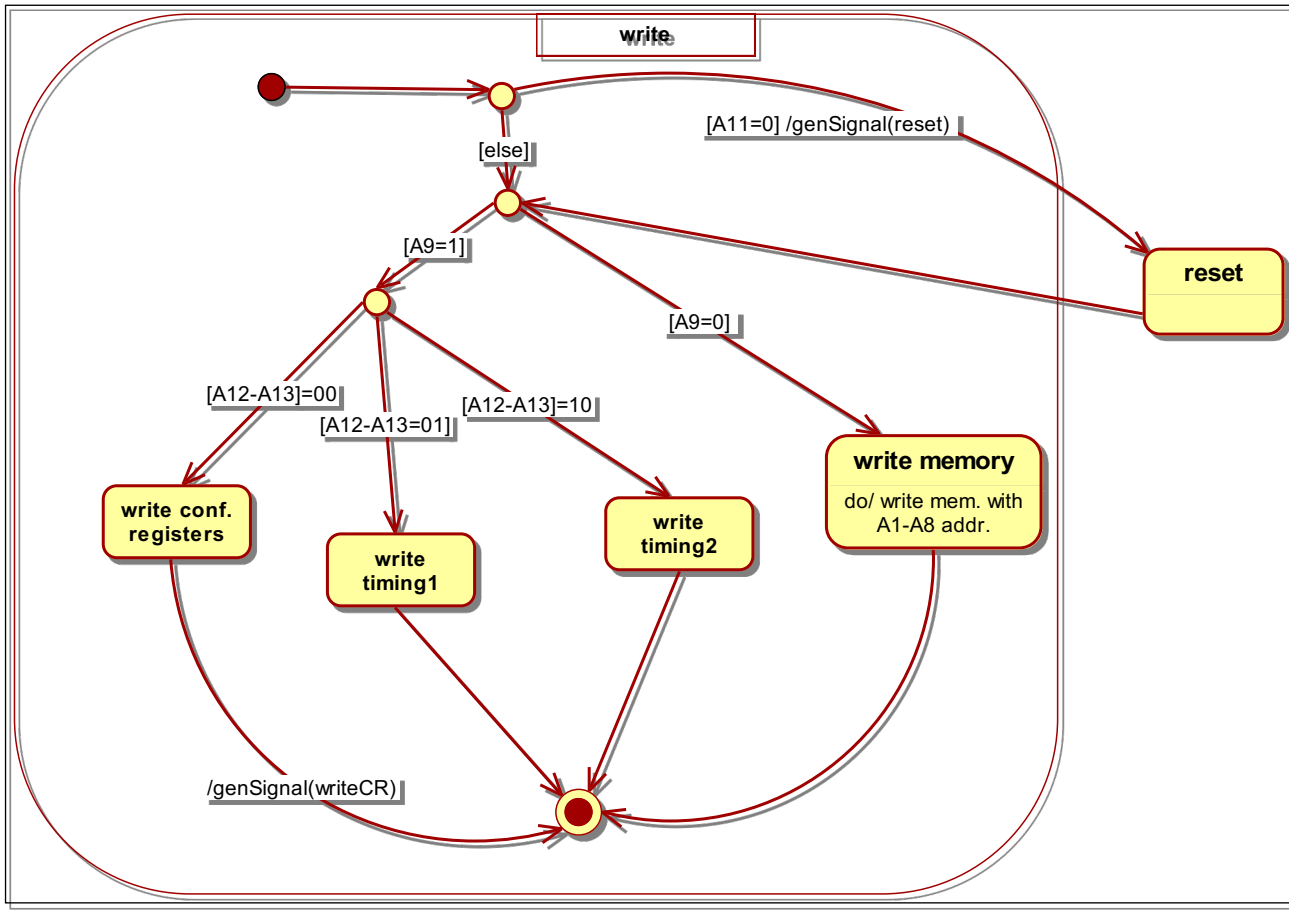

Figure 21: write state chart

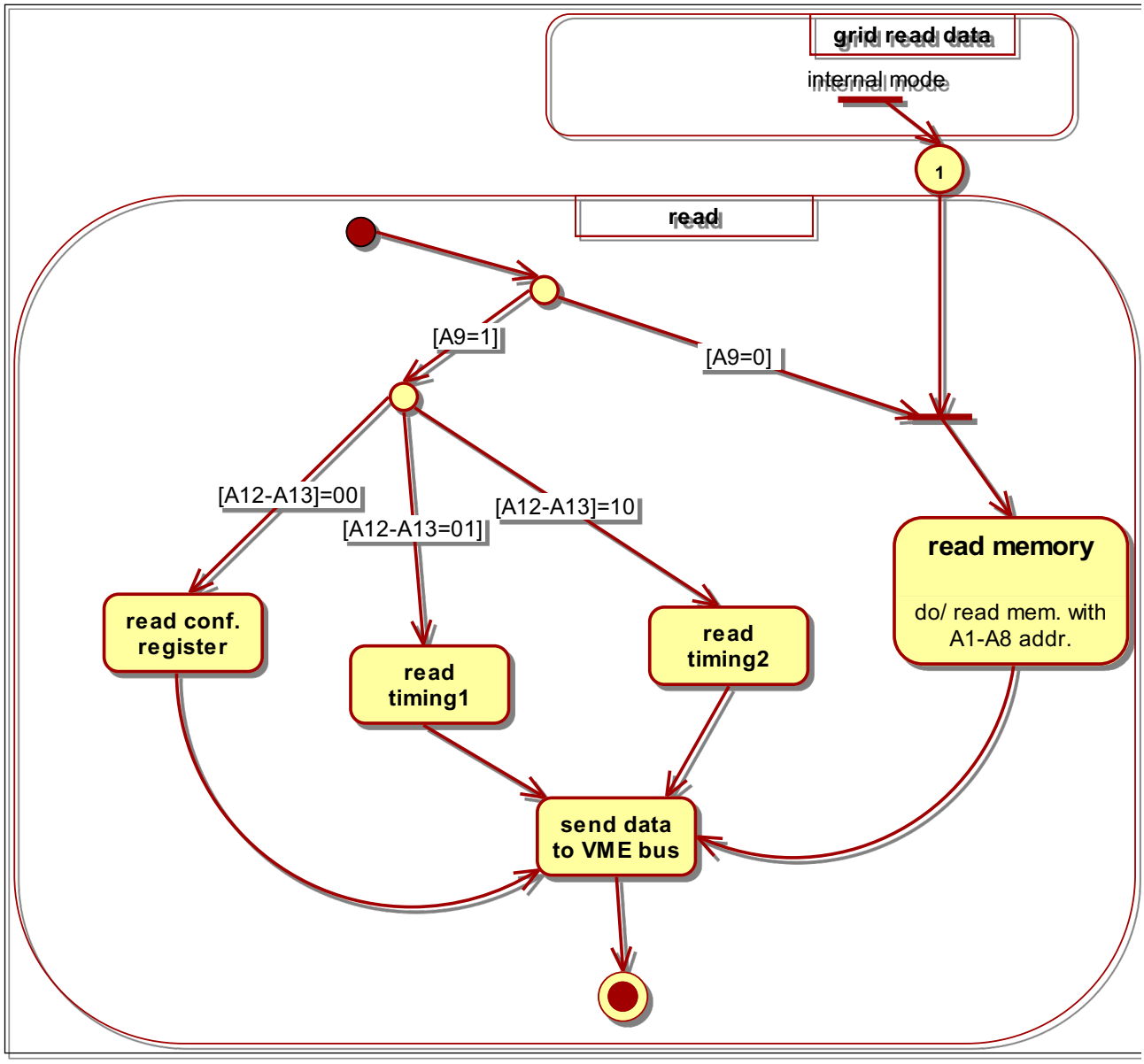

Figure 22: read state chart

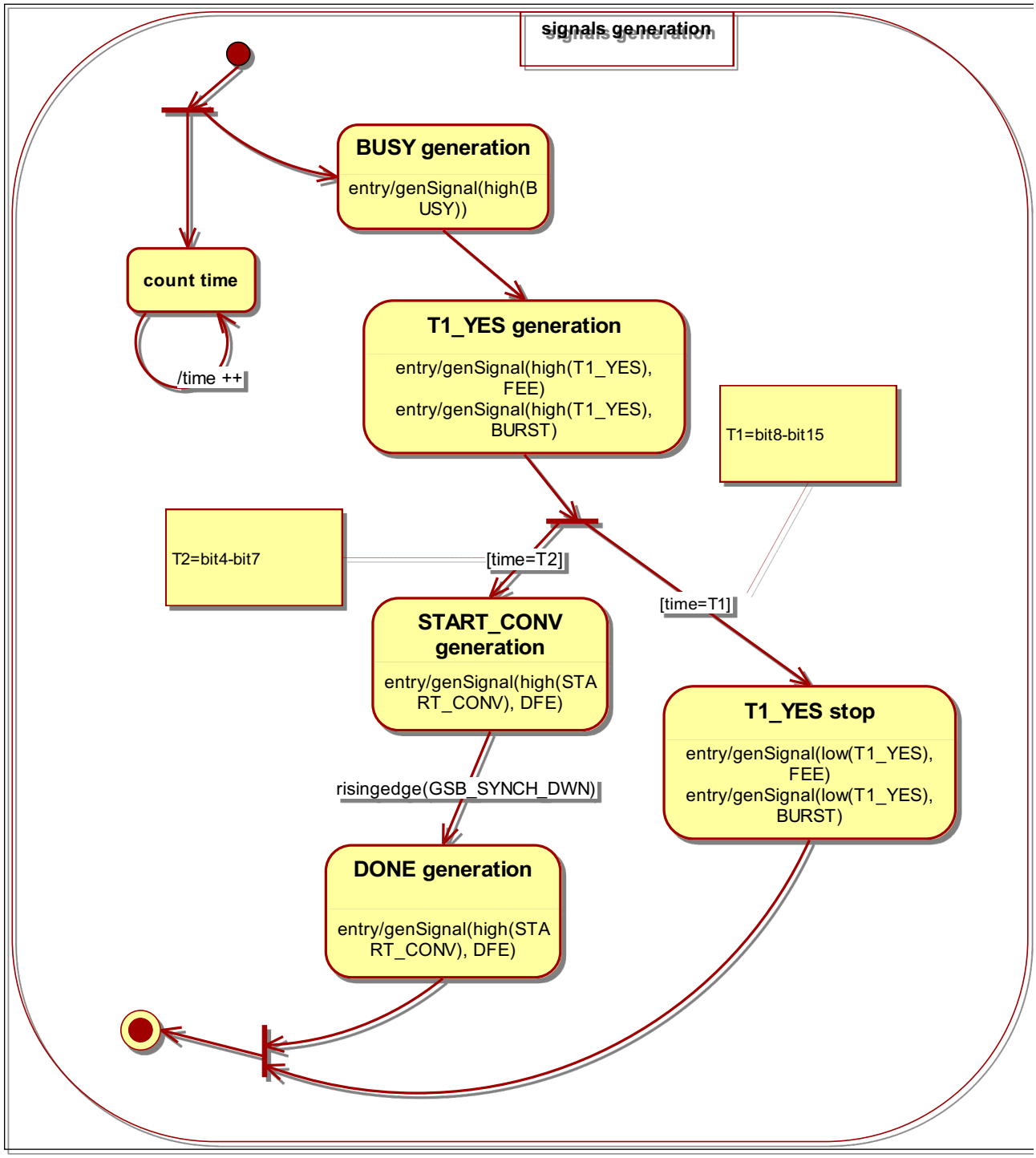

Figura 23: signals generation state chart

•

4. GRID BOARD DETAILED DESIGN

In this chapter is described the detailed design of the board. The VHDL code of the Altera PLD is reported. All the *ports* (input and output of the PLD) and *signals* (internal to PLD) are referred to PLD.

4.1 MAIN VHDL SIGNAL

The main VHDL signals and ports are the following:

```
-- bufd: buffered VME data
bufd : inout std_logic_vector (15 downto 0);

signal bitout    : std_logic_vector(15 downto 0); -- configuration register

-- mux out = multiplexed output of internal registers
signal mux_out                          : std_logic_vector(15 downto 0);
```

The shift register is

```
signal shreg_in                         : std_logic_vector (11 downto 0);
```

4.2 INTERNAL REGISTERS OF THE BOARD

The board, as mentioned above, has three internal registers:

- **bitout**: configuration register
- **timereg0**, **timereg1**: used to store and read timing information.

For the access of this registers it is necessary to set the vmea[9]:

- with vmea[9]=1 for reading and writing registers;
- with vmea[9]=0 for reading and writing memory (see 3.10.2).

4.3 CONFIGURATION REGISTER

The configuration register is defined in VHDL with the following line:

```
signal bitout    : std_logic_vector(15 downto 0); -- configuration register
```

The bits of the configuration register used are the following:

- Bit 0: Memory
- Bit 1: Test
- Bit 2: T1 source
- Bit 3: Busy/Continuous
- Bit 4 – 7: Start conversion delay
- Bit 8 – 15: T1_YES length

4.3.1 BIT 0: MEMORY

See par. 3.9.1.1 for a brief explanation of the meaning of this bit.

```
bitout(0) <= bufd(0)

-- the configuration register holds the values for
-- ext_int_n: memory operation ext(1)=data from the shift register of the board
-- int(0)=data from the VME
ext_int_n    <= bitout(0);
```

See the par. 4.4 for an explanation of the data transfer.

4.3.2 BIT 1: TEST

See par. 3.9.1.2 for a brief explanation of the meaning of this bit.

```
signal real_test_n    : std_logic;
real_test_n <= bitout(1) <= bufd(1)
```

For the test mode are used the following signals and ports:

```
with real_test_n select
    alt_clk_dwn <= test_ck_dwn when '0',
    gsb_ck_dwn                 when others;

-- Choose internal/external 1=ext 0=int(test_xxx)
with real_test_n select
    alt_data_dwn <= test_data_dwn when '0',
    gsb_data_dwn                  when others;

with real_test_n select
    alt_stb_dwn_n <= test_stb_dwn_n when '0',
    gsb_stb_dwn_n                   when others;

-- invert synchro signal in order to have it active high
with real_test_n select
    alt_syn_dwn_n <= test_syn_dwn when '0',
    gsb_syn_dwn                   when others;

alt_syn_dwn <= not alt_syn_dwn_n;
```

The above code means that the PLD select all the signals that start with test prefix if real_test_n = 0, or with gsb prefix (coming from FEE) elsewhere. Both gsb and test signals are copied into alt signals.
For the generation of test signals is used the shift register that contains a single bar value:

```
signal shreg : std_logic_vector (11 downto 0);
```

The data contained in the shift register are written out one bit for each clock period. It is copied out the 11 bit of the shift register and, after this, all the bit are shifted left:

```
test_data_dwn <= shreg(11);
```

For the generation of the data into the shift register the following signals are used:

```
signal cksyn1         : std_logic;
signal qtmp_stb       : integer range 0 to 127;

test_ck_dwn <= ck5;
ckshift <= cksyn1 and test_ck_dwn;
```

cksyn1 is read from the clock of the system.

```
process (test_stb_dwn_n, gblres_n)
    variable cnt_stb : integer range 0 to 127;
begin
    if gblres_n = '0' then
      cnt_stb := 0;
    elsif rising_edge(test_stb_dwn_n) then
      cnt_stb := cnt_stb+1;
    end if;
    qtmp_stb <= cnt_stb;
end process;
```

If gblres_n = 0 the starting value of bar energy is setted to 0, otherwise for each rising edge of the test strobe the value of the energy of the bars is incremented of 1 and this value is copied into qtmp_stb. Hence, the value of the bars starts from 0 to 127.
When the strobe go high the current energy value of a bar is transferred into the shreg. During the clock falling edge the shreg is left-shifted and its MSB is transferred to test_data_dwn.

```
process (test_stb_dwn_n, ckshift)
begin
    if test_stb_dwn_n = '1' then
      shreg (6 downto 0)  <= conv_std_logic_vector(qtmp_stb, 7);
      shreg (11 downto 7) <= not conv_std_logic_vector(qtmp_stb, 5);
    elsif falling_edge(ckshift) then
      shreg(11 downto 1)  <= shreg (10 downto 0);
      shreg(0)            <= '0';
    end if;
end process;
```

4.3.3 BIT 2: T1_START SOURCE

See par. 0 for a brief explanation of the meaning of this bit.

```
-- t1_ext_int_n: choose the source of t1, external (1),
-- internal 1 KHz (0) (in test mode, the t1 can be also be external)
signal t1_ext_int_n   : std_logic;

signal test_t1_start  : std_logic;

-- alt_t1_start: t1 start signal NOT T1_yes
alt_t1_start  : buffer std_logic;

t1_ext_int_n <= bitout(2);
```

The t1_start signal is generated from a 1KHz internal clock of the PLD.

```
-- the t1 start signal in test mode is the 1kHz pulse
test_t1_start <= pulse1k;
```

The selection of the correct t1_start depends by the configuration register:

```
-- choose external/internal t1 with bit t1_ext_int_n
  with t1_ext_int_n select
    alt_t1_start <= test_t1_start when '0',
    ext_t1_start
    when others;
```

4.3.4 BIT 3: BUSY/CONTINUOUS

See par. 3.9.1.4 for a brief explanation of the meaning of this bit.

```
signal t1_start, cont_syn_n           : std_logic;
signal busyres                        : std_logic;
alt_t1_start  : buffer std_logic;

-- cont_syn_n: continuous(=0) or synchronous (1=wait for busy to be reset)
cont_syn_n   <= bitout(3);
```

When the board must be resetted or when we are in sync mode the busy signal is resetted.

```
-- reset busy signal in continuous mode
busyres <= not gblres_n or (cont_syn_n and (not stopclk_t1_n));

with cont_syn_n select
    t1_start <= busy when '0',
    alt_t1_start     when others;
```

In the above code is showed that with the bit 3 of the configuration register is established that if the value is high it is necessary to wait the reset of the busy signal. For busyres signal see 4.5.1.

4.4 BOARD ADDRESSING

For addressing of the board are used the following ports:

- VMEA[1..15](I) + AM[0..5](I)

For data transfer are used the following *ports* and *signals*:

- BUFD[0..15](I/O), BUFFDIR(I) (1 vme→board, 0 board→vme);
- ENABUFF_N(I);
- DS0_N(I), DS1_N(I) (data strobes): from this ports is build VME_CYCLE(O);
- LWORD_N(I), IACK_N(I): longword and interrupts not active;
- ENAREAD_N(I): enable reading of the data from board;
- ADRROK_N(I): output of address comparator.

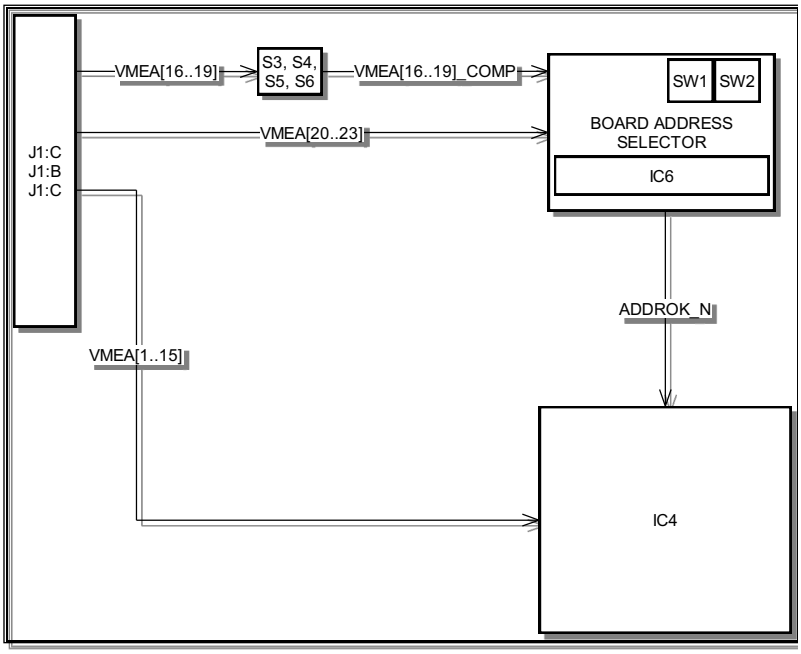

Figura 24: hardware for addressing

The addressing of the board is compound by two parts:

- 15 bits for internal addressing of the board
- 8 bits fot the addressing of the board over the VME bus: the selection of the board address is possible by means of SW1 and SW2 switchs.

For VME protocol it is generated the vme_cycle:

```
vme_cycle <= not (ds0_n and ds1_n);
```

Si verificano alcuni segnali legati probabilmente al bus VME (addressing):

```
-- Check the address modified bits (am2 is not present because is
-- a "don't care" bit)
am3e_3a_n <= not (am5 and am4 and am3 and am1 and (not am0));
am3d_39_n <= not (am5 and am4 and am3 and (not am1) and am0);
```

It is important to verify if addressing is ok.

```
-- Finally check that the standard address modifier is correct
-- ie that it is not a 32 bit or interrupt cycle
stam_n    <= ( ( (not iack_n) or (not lword_n)) or
               (am3e_3a_n and am3d_39_n));
```

If ones of the two conditions (address or addressing) is ok, this implies that it is all ok.

```
-- board ok strobe: addressing ok & address ok
bdok_n    <= stam_n or addrok_n;
```

- DELVME_CYCLE(I): cycle delayed of 100 ns

If the address is correct, when rising edge of the VME cycle occurs it is possible to read and translate the address presents in the IC4 input ports.

```
-- flip-flop to latch the address lines
-- only when the board is addressed with the correct cycle
process (delvme_cycle, bdok_n)
begin
    if rising_edge(delvme_cycle) then
      if bdok_n = '0' then
-- register select
           regsel (1 downto 0) <= vmea(13 downto 12);
-- enable global reset (write mode)
           gblres_ena_n        <= vmea(11);
-- latched vme addresses used to access the memory
      lvme_add            <= vmea(9 downto 1);
      end if;
    end if;
end process;
```

4.4.1 REGISTERS READING

For reading the internal registers is used a logical mux:

```
-- mux out = multiplexed output of internal registers
  signal mux_out                          : std_logic_vector(15 downto 0);

-- select configuration or timing registers (read only) for readout
  with regsel select
    mux_out <=
    bitout  when "00",
    timreg0 when "01",
    timreg1 when "10",
    bitout1 when others;
```

The mux is setted with the value of the registers based on the value of regsel. For the data transfer of this data to output see 4.4.5.

4.4.2 REGISTERS WRITING

For writing the registers it is necessary to set vmea(9) = 1.

```
write_reg_n     <= not ((not clkwrite0_n) and lvme_add(9));

-- write to the configuration register (r/w)
process(write_reg_n, bufd)
begin
    if rising_edge(write_reg_n) then
      bitout <= bufd;
    end if;
```

```
end process;
```

4.4.3 BOARD RESET

Enable the global reset:

```
-- enable global reset (write mode)
        gblres_ena_n        <= vmea(11);
```

This means that if vmea(11) = 0 the reset is enabled. The board address to enable the reset is 0x0200.

After this, the gblres_n is setted:

```
-- Global reset handling-the signal is stopped at the end
-- of VME cycle
process (clkwrite0_n, vme_cycle)
begin
    if vme_cycle = '0' then
      gblres_n <= '1';
    elsif rising_edge(clkwrite0_n) then
      gblres_n <= gblres_ena_n;
    end if;
end process;
```

The gblres_n resets all the signal and port of the PLD.
In particular,

4.4.4 ADDRESS MEMORY SELECTION

An internal address of the memory board is addressed with 8 bits.

```
-- ext_int_n: memory operation ext(1)=data from the shift register of the board
-- int(0)=data from the VME
```

Into mem_add output port is copied the address of the memory to write. If the board is in the internal mode (data from VME) the address is get from the VME bus, otherwise it is obtained from ext_add signal.

```
-- use vme address in internal mode
with ext_int_n select
    mem_add     <= lvme_add(8 downto 1) when '0',
    ext_add                             when others;
```

In external mode the address of the memory is generated in this way::

```
ext_add     <= conv_std_logic_vector(qtmp_add, 8);

process (alt_stb_dwn_n, res_addcnt)
    variable cnt_add : integer range 0 to 127;
begin
    if res_addcnt = '1' then
      cnt_add := 127;
```

```
    elsif rising_edge(alt_stb_dwn_n) then
      cnt_add := cnt_add+1;
    end if;
    qtmp_add <= cnt_add;
end process;
```

This means that the address range is from 0 to 127 (one address for each PD).

4.4.5 DATA TRANSFER TO OUTPUT BUS

The data transfer to output bus (bufd) is performed with the following process:

```
process(enaread_reg_n, transfer_mem, mux_out, shreg_in)
begin
    if enaread_reg_n = '0' then
-- enable readback of internal registers
      bufd     <= mux_out;
-- enable transfer of data from the shift register to memory
    elsif transfer_mem = '1' then
      bufd     <= "1111" & shreg_in;
    else
-- tristate the bus
      bufd     <= (others => 'Z');
    end if;
end process;
```

The data transfer is based on the following signals:

- transfer_mem_n:

```
-- handling of data bus
-- transfer_mem is enabled ONLY when the board is not addressed
-- and the board is in external mode==> NO ACCESS to the board
-- while waiting after a reset
transfer_mem <= ext_int_n and bdok_n;
```

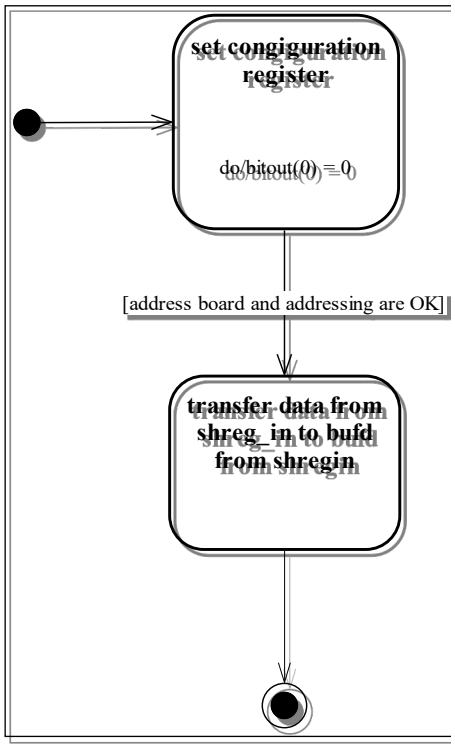

Figure 25: read memory

- **enaread_reg_n**: for the transfer of the registers (enaread_n is an input port)

```
enaread_reg_n   <= not ((not enaread_n) and lvme_add(9));
```

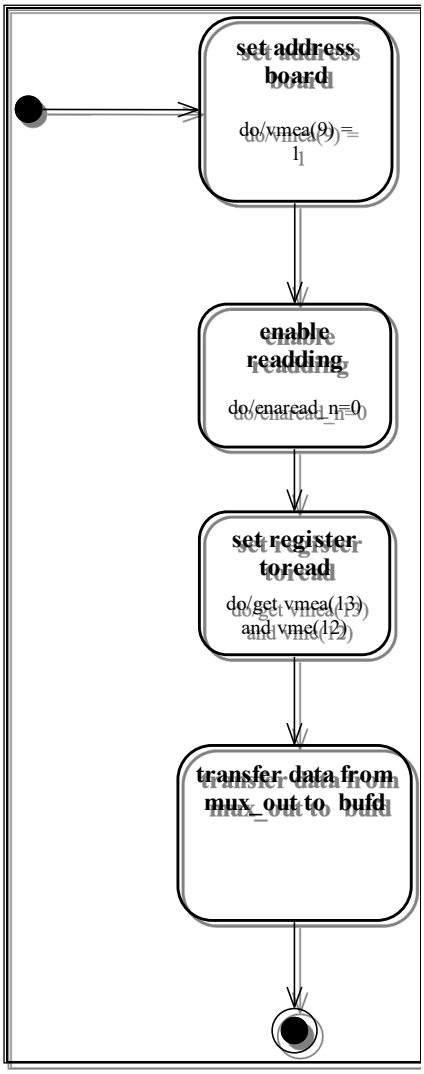

Figura 26: read registers

4.4.6 DATA MEMORY READING

For the selection of the address to read see 4.4.4.

```
-- mem_add: address lines to memory

-- in read mode memory is accessed only by vme
read_mem_n     <= enaread_mem_n;

enaread_mem_n   <= not ((not enaread_n) and (not lvme_add(9)));
```

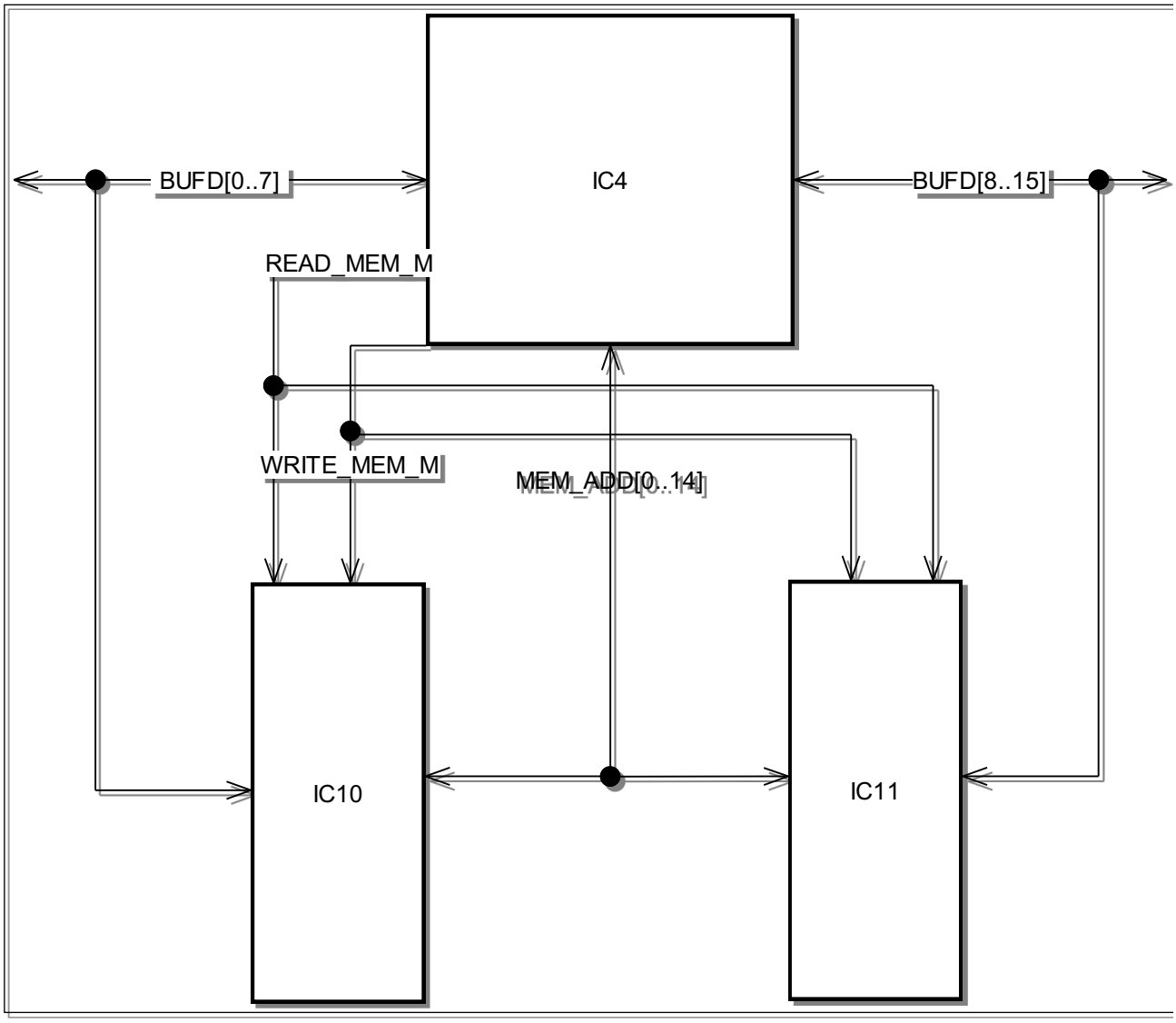

Figure 27: components for reads and writes memory

In Figure 27 is showed that the 16 data bits are obtained from the two memory chips (IC10 and IC11).

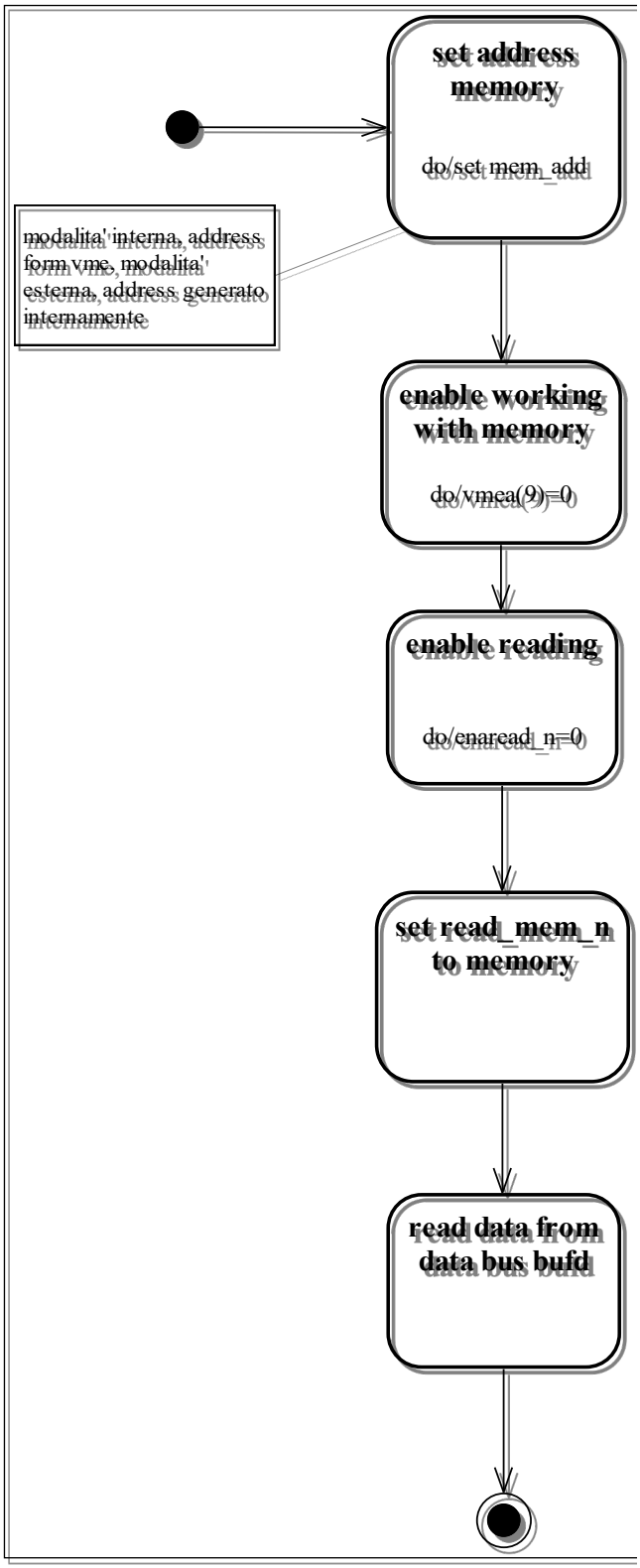

Figura 28: memory reading

4.4.7 DATA MEMORY WRITING

For the data memory writing operation it is necessary to select the write mode depending by internal or external mode (configured into configuration register).

```
-- choose write_mode
with ext_int_n select
    write_mem_n <= pre_write_mem_n      when '0',
    ext_write_n                         when others;
```

1) external mode

```
ext_write_n <= pulse_stb_short_n;

-- external info are dummy for the moment
process (ck5, alt_stb_dwn_n)
begin
    if alt_stb_dwn_n = '0' then
      pulse_stb_short_n <= '1';
    elsif rising_edge(ck5) then
      pulse_stb_short_n <= not alt_stb_dwn_n;
    end if;
end process;
```

When the strobe signal is down the data coming from FEE (ones PD) and this enables the writing of this data into memory.

2) internal mode

```
pre_write_mem_n <= not ((not clkwrite0_n) and (not lvme_add(9)));
```

where clkwrite0_n and clkwrite1_n are two clock for writing the address.

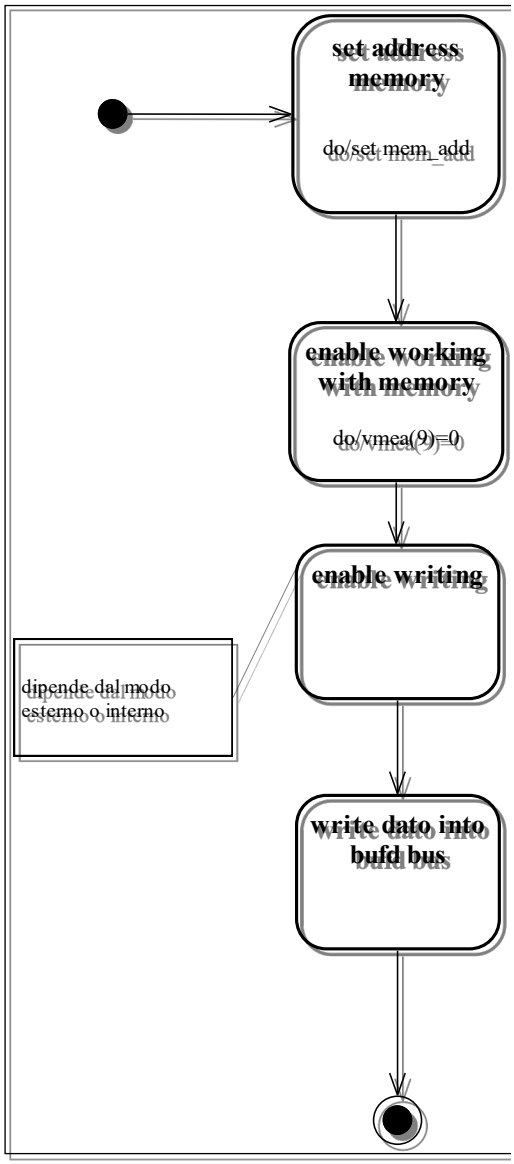

Figure 29: write data to memory

4.5 BUSY, T1_YES AND START_CONV GENERATION

4.5.1 BUSY AND T1_YES GENERATION

The busy signal is an active low signal (0=busy).
For the generation of the T1_YES signal it is necessary to receive a T1_START. After the reception of T1_START, the next step is the generation of the BUSY signal.

```
busy              : buffer std_logic;

-- reset busy signal in continuous mode
busyres <= not gblres_n or (cont_syn_n and (not stopclk_t1_n));
```

```
-- generate busy signal

process(alt_t1_start, busyres)
begin
    if busyres = '1' then
      busy <= '0';
    elsif rising_edge(alt_t1_start) then
      busy <= '1';
    end if;
end process;
```

As consequence of this, the BUSY signal is generated at the same moment of the T1_START.
After T1_START and BUSY generation, the next step is the generation of T1_YES. See the following VHDL code (from clock_s2.vhd file):

```
-- flip-flop to generate q1
process (start, reseta_n)
begin
      if reseta_n = '0' then
            q1 <= '0' ;
            q1_n <='1';
-- clock on the rising edge
      elsif rising_edge(start) then
            q1 <= '1';
            q1_n <= '0';
      end if;
end process;

-- flip-flop to generate q2
process (clk, resetb_n)
begin
      if resetb_n = '0' then
            q2   <= '0';
            q2_n <= '1';
-- clock on the rising edge
      elsif falling_edge(clk) then
            q2   <= q1;
            q2_n <= q1_n;
            end if;
end process;
```

When t1_start is generated, the T1_YES is generated according with a 1 MHz clock.
The mapping between the two *entity* is the following:

```
-- clock synchronizer for t1 signal
  clock_syn_2 : clock_s2 port map (
    start    => t1_start,              -- IN: start signal on the rising edge
    reseta_n => dres_t1_n,             -- IN: reset of the first ff
    resetb_n => dumres_n,              -- IN: reset of the second ff
    clk      => ck1m,                  -- IN: clock to be syncronized
    clksyn   => cksyn1_t1,             -- OUT: syncronized clock
    clksyn_n => cksyn1_t1_n,           -- OUT: syncronized clock negated
    q1       => q1_t1,                 -- OUT: Q of the first FF
    q2       => t1_yes,                -- OUT: Q of the second FF
```

```
    q1_n     => q1_t1_n,              -- OUT: Q of the first FF
    q2_n     => q2_t1_n,              -- OUT: Q of the second FF
    clk_n    => ck1m_n);              -- OUT: input clock reversed

  with cont_syn_n select
    t1_start <= busy when '0',
    alt_t1_start     when others;
```

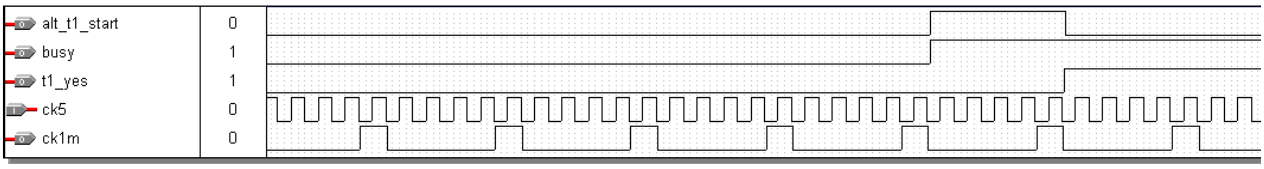

Figure 30: timing diagram of t1_start, busy, t1_yes

4.5.1.1 BUSY/CONTINUOUS MODE

Based on the analysis of par. 4.3.4 and 4.5.1, the conclusions about the BUSY/CONTINUOUS mode are the following:

- in continuous mode (bit3=1), the t1_start signal coming from ext_t1_start or test_t1_start (if bit 2 of the configuration register is low) and the busy signal is setted low for a new cycle;
- in busy mode (bit3=0), the t1_start signal coming from busy signal. This means that the t1_start go high when busy go high and this starts the generation of all the signal. For a reset of the busy signal it is necessary a reset of the board.

4.5.2 START_CONV GENERATION FROM T1_YES

After the T1_YES generation it is necessari to generate the START_CONV. The first step is the generation of delay.

```
-- t1_yes signal
t1_yes           : buffer std_logic;

-- t1_len: length of t1_yes
signal t1_len : std_logic_vector(7 downto 0);

-- t1_del: start_conv_del between t1_yes and start_conversion
signal t1_del : std_logic_vector(7 downto 0);
```

The value of delay is read from configuration register:

```
-- use bits 7->0 for the delay before start_conversion
t1_del <= bitout1(7 downto 0);
```

When t1_yes rising edge occours, a counter starti:

```
-- generate the start conversion delay (used afterwards as a reset signal)
process (t1_yes, stopdel_n)
begin
    if stopdel_n = '0' then
      start_conv_del <= '0';
```

```
    elsif rising_edge(t1_yes) then
      start_conv_del <= '1';
    end if;
end process;
```

When the counter starts, the counting continues until the value of t1_del is reached. After this, the start_conversion signal is generated.

```
-- end of delay
  x_stopdel_n          <= '0' when qtmp_t1 = t1_del else '1';
-- generate start_conversion
  x_start_conversion   <= '1' when qtmp_t1 = t1_del else '0';
```

The overall process is driven by a clock of 1 MHz.

```
process (ck1m)
begin
    if falling_edge(ck1m) then
      stopclk_t1_n     <= x_stopclk_t1_n;
      stopdel_n        <= x_stopdel_n;
      start_conversion <= x_start_conversion;
    end if;
end process;
```

For generation of qtmp_t1 the following code is reported:

```
signal qtmp_t1                          : std_logic_vector(7 downto 0);

qtmp_t1     <= conv_std_logic_vector(x_qtmp_t1, 8);
-- counter for the t1_clocks

process (cksyn1_t1, rescnt_t1_n)
    variable cnt_t1 : integer range 0 to 255;
begin
    if rescnt_t1_n = '0' then
      cnt_t1 := 0;
    elsif rising_edge(cksyn1_t1) then
      cnt_t1 := cnt_t1+1;
    end if;
    x_qtmp_t1 <= cnt_t1;
end process;
```

qtmp_t1 is a counter that contains the numbers from 0 to 255 and it is syncronized with cjsyn1_t1 signal generated by clock.

About the duration of start_conversion, the start_conversion go low when qtmp_t1 counter is resetted or when it is modified. For this reason only 5 fronts of the 5 Mhz clock are necessary. This is showed in the following simulation:

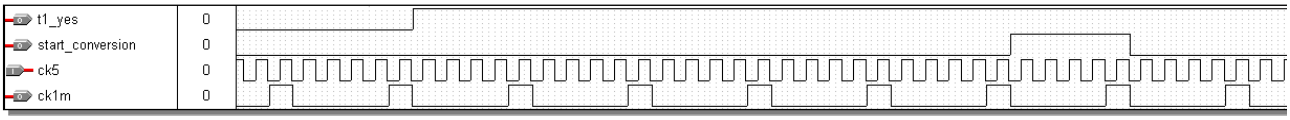

Figure 31: simulation of start_conversion signal duration

In the above picture is even showed the delay between t1_yes and start_conversion generation.

4.5.3 DONE GENERATION

This is the done signal:

```
-- done: set at the end of conversion
done              : buffer std_logic;
```

The done signal goes high with sync_dwn.

```
-- alt_syn_dwn is the syncro pulse active during conversion (high)
process(gblres_n, alt_syn_dwn)
begin
    if gblres_n = '0' then
      done <= '0';
    elsif falling_edge(alt_syn_dwn) then
      done <= '1';
    end if;
end process;
```

4.5.4 BURST_TIMING GENERATION

The generation of burst_timing signal is very simple:

```
burst_timing                <= start_conversion;
```

4.6 MANAGEMENT OF TIMING INFORMATION

The timing information are stored into two board internal registers:

- timereg0
- timereg1

It is possibile to read this registers as showed in the following code:

```
-- copy the shift register to the timing registers
timreg0(15 downto 0) <= shreg_timing(15 downto 0);
timreg1(15 downto 0) <= shreg_timing(31 downto 16);
```

DATA_TIMING(I) are read only when t1_yes is high and at the falling edge of the CLK_TIMING(I). The DATA_TIMING(I) is a 1 bit signal and it is necessary a shift register for store all the timing information.

```
-- timing part: shift data only when t1_yes is high
process (t1_yes, clk_timing, data_timing)
begin
-- transfer timing info only during t1_yes (which has to be
-- longer then 32/5=6.4 us)
```

```
    if falling_edge(clk_timing) then
-- data is valid at the falling edge of clock
      if t1_yes = '1' then
        shreg_timing(31 downto 1) <= shreg_timing (30 downto 0);
        shreg_timing(0)           <= data_timing;
      end if;
    end if;
end process;
```

-

4.7 BURST DATA GENERATION

This board are able to generate test data for burst board as the MCAL FEE.
For this purpose 4 signal are used:

- BSB_CK_DWN
- BSB_STB_DWN
- BSB_DATA_DWN
- BURST_TIMING

```
bsb_data_dwn                <= shreg_out(101);
bsb_ck_dwn  <= ck5;
bsb_stb_dwn <= cksyn_b_n;
burst_timing                <= start_conversion;
```

For generation of the data a shift register is used:

```
signal shreg_out : std_logic_vector(101 downto 0);
```

The data are generated with a fixed pattern. The shift register is filled when the sync signal is low and the data transfer from shift register to output bus is performed when il sync is high (i.e. when the grid data aren't read).

```
process (ckshift_b, alt_syn_dwn)
begin
    if alt_syn_dwn = '0' then
      shreg_out                  <= "11111000010111011" & "00011000000110011" &
"11110000011011101" & "00100000011101110" &
                  "00000000011101101" & "00101000010101100";
    elsif falling_edge(ckshift_b) then
      shreg_out(101 downto 1) <= shreg_out (100 downto 0);
      shreg_out(0)            <= '0';
    end if;
end process;
```

About the clock:

```
ckshift_b   <= cksyn_b and ck5;
```

The cksyn_b and the cksyn_b_n are get from

```
clock_syn_3 : clock_s2 port map (
```

```
start    => alt_syn_dwn,          -- IN: start signal on the rising edge
reseta_n => dres_b_n,             -- IN: reset of the first ff
resetb_n => dumres_n,             -- IN: reset of the second ff
clk      => pulse_b,              -- IN: clock to be syncronized
clksyn   => cksyn_b,              -- OUT: syncronized clock
clksyn_n => cksyn_b_n,            -- OUT: syncronized clock negated
q1       => q1_1_b,               -- OUT: Q of the first FF
q2       => q2_1_b,               -- OUT: Q of the second FF
q1_n     => q1_1_b_n,             -- OUT: Q of the first FF
q2_n     => q2_1_b_n,             -- OUT: Q of the second FF
clk_n    => pulse_b_n);           -- OUT: input clock reversed
```

5. SOFTWARE

5.1 USING THE INFN SOFTWARE

To launch the software go to the program directory and type *./test_mcal.tcl* . The main window appears, here the user has to select the button *GRID TEST* to start the auto-check routine.
A new window appears, that enables the user to perform

- read and write of the configuration register
- read and write memory
- using board test mode

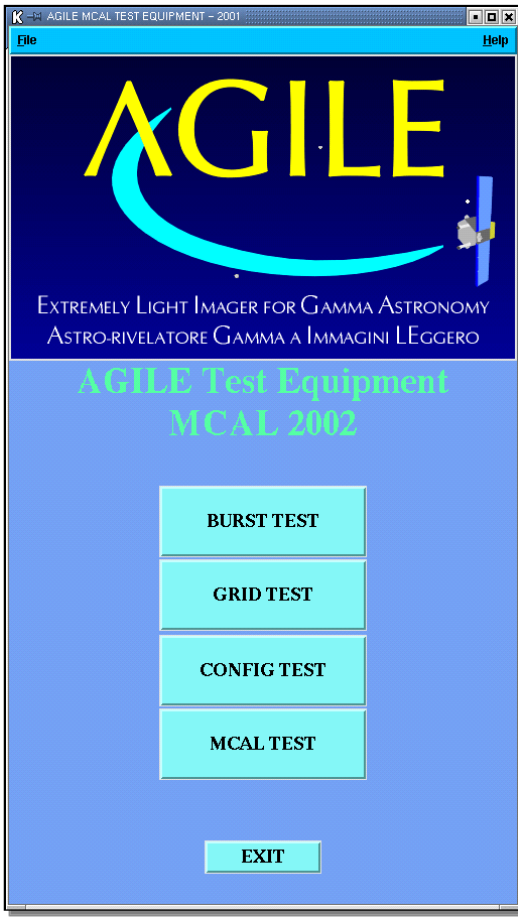

Figure 32: **Software main window.**

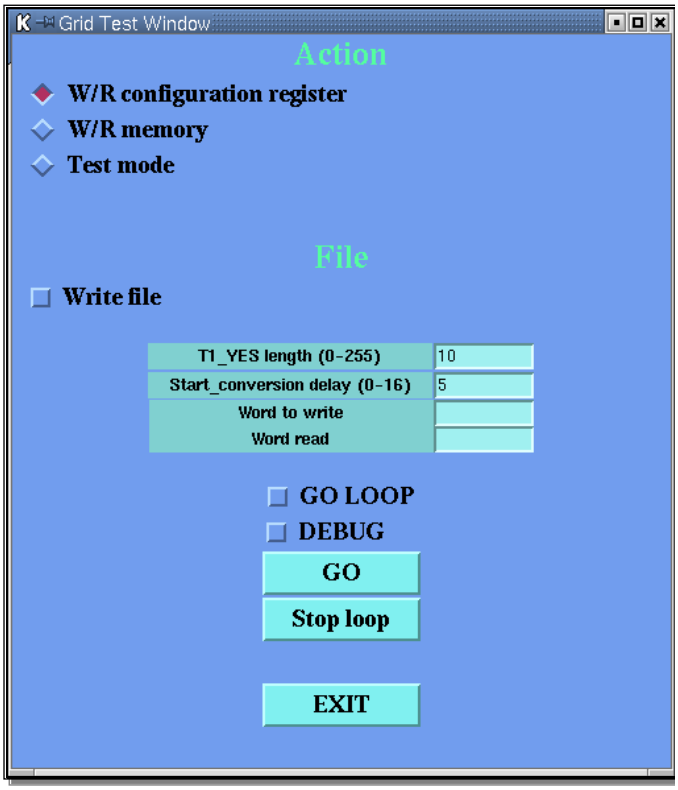

Figure 33: **Grid test window.**

5.1.1 W/R CONFIGURATION REGISTER

With this software it is possible to write and read the configuration register of the board
For the use of this functionality it is necessary to select the first element of the action list (*W/R configuration register*). The look of the window is showed in the next figure.

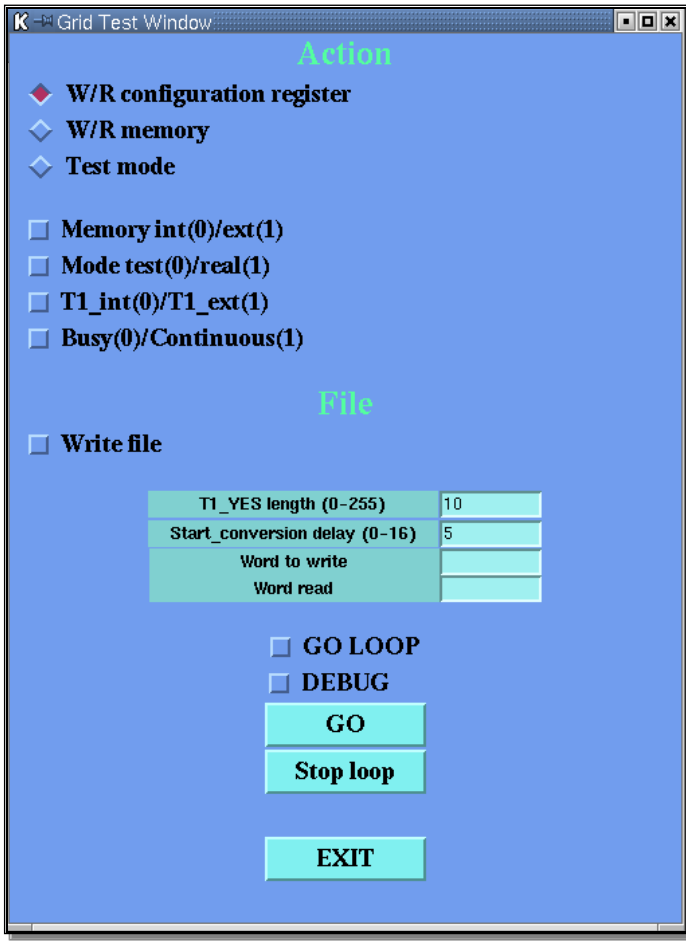

Figure 34: **W/R configuration register window.**

With a check in the checkbox widget it is possible to select the second option (1 value).
The meaning of the command are the following:

- **Memory**: if 0, the memory are read and write from VME bus. If 1, the data is read from FEE bus.
- **Mode test**: if 0, test data are generated internally.
- **T1**: if 0, the T1_START signal is generated internally (with 1 KHz of frequency). If 1, the T1_START signal is generated externally form EXT_T1_START signal.
- **Busy** or **Continuous** mode (see par. 4.3.4)

In addition, with this widget it is possible to set the T1_YES length time and start_conversion delay with appropriated value.
With the *GO* button it is possible to write the configuration register. After write operation, the configuration register is read and this value is displayed in the *Word read* text box.
The *DEBUG* option can be useful for reading debug message on console. With *GO LOOP* option it is possible to write and read the configuration register many times. The end of the loop is determinate by the *Stop loop* button.
The *Write file* option appears not working.

5.1.2 W/R MEMORY

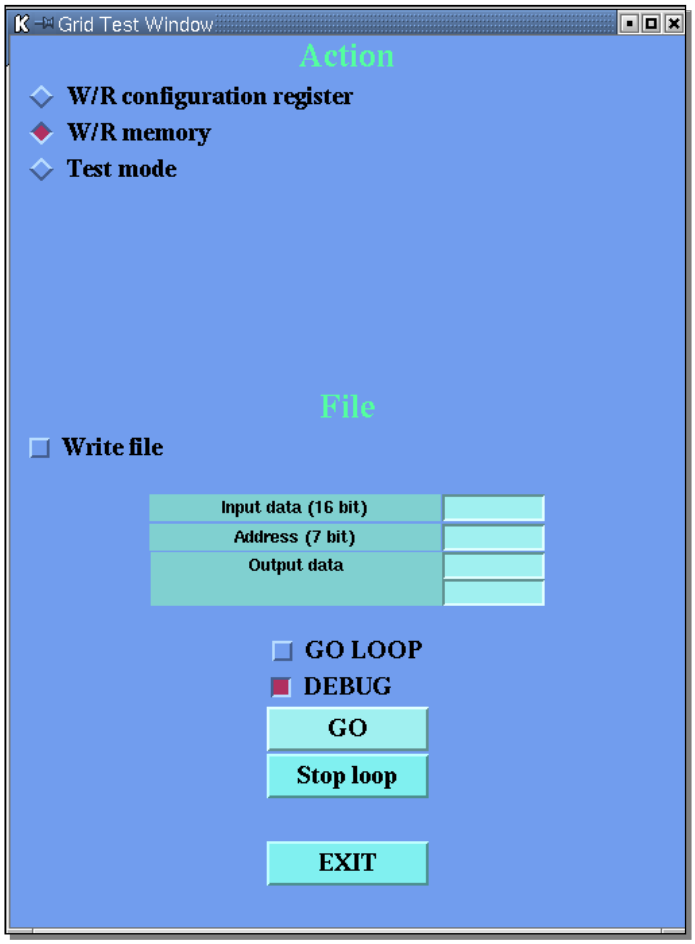

Figure 35: W/R memory widget

With the W/R memory widget it is possible to write and immediately read a value into a specified address of the memory board.
The widget works similarly as the precedent widget. It is necessary to insert the input data (16 bit value in hexadecimal format), the 7 bit address of the memory board (in hexadecimal format) and push the *GO* button for writing and reading the value into memory.

5.1.3 TEST MODE

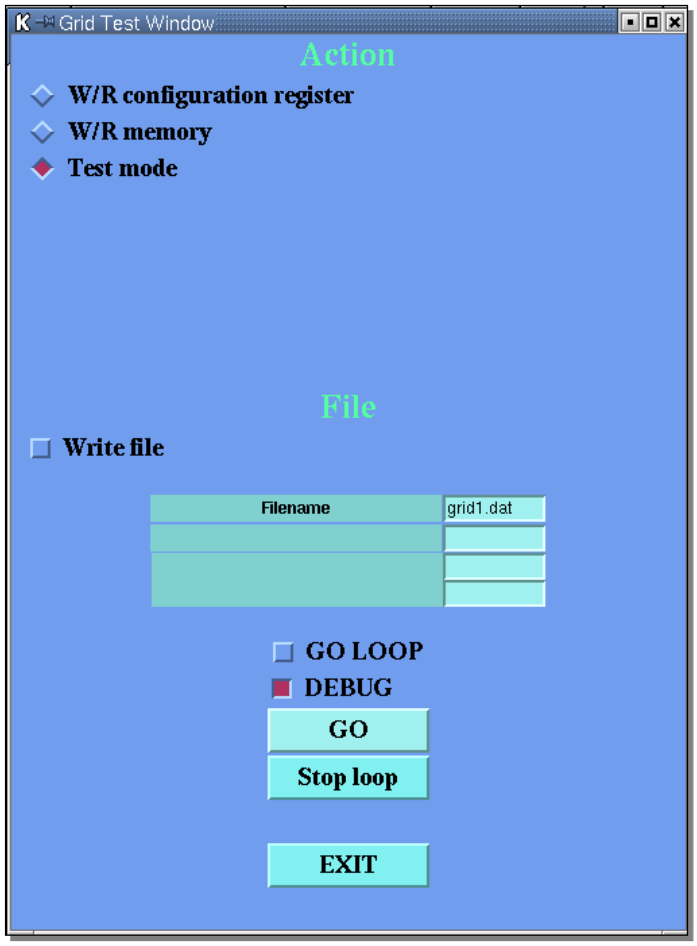

Figure 36: Test mode

With the Test Mode widget it is possible to read the data produced by Test Equipment.

APPENDIX A – SCHEMATIC DIAGRAM

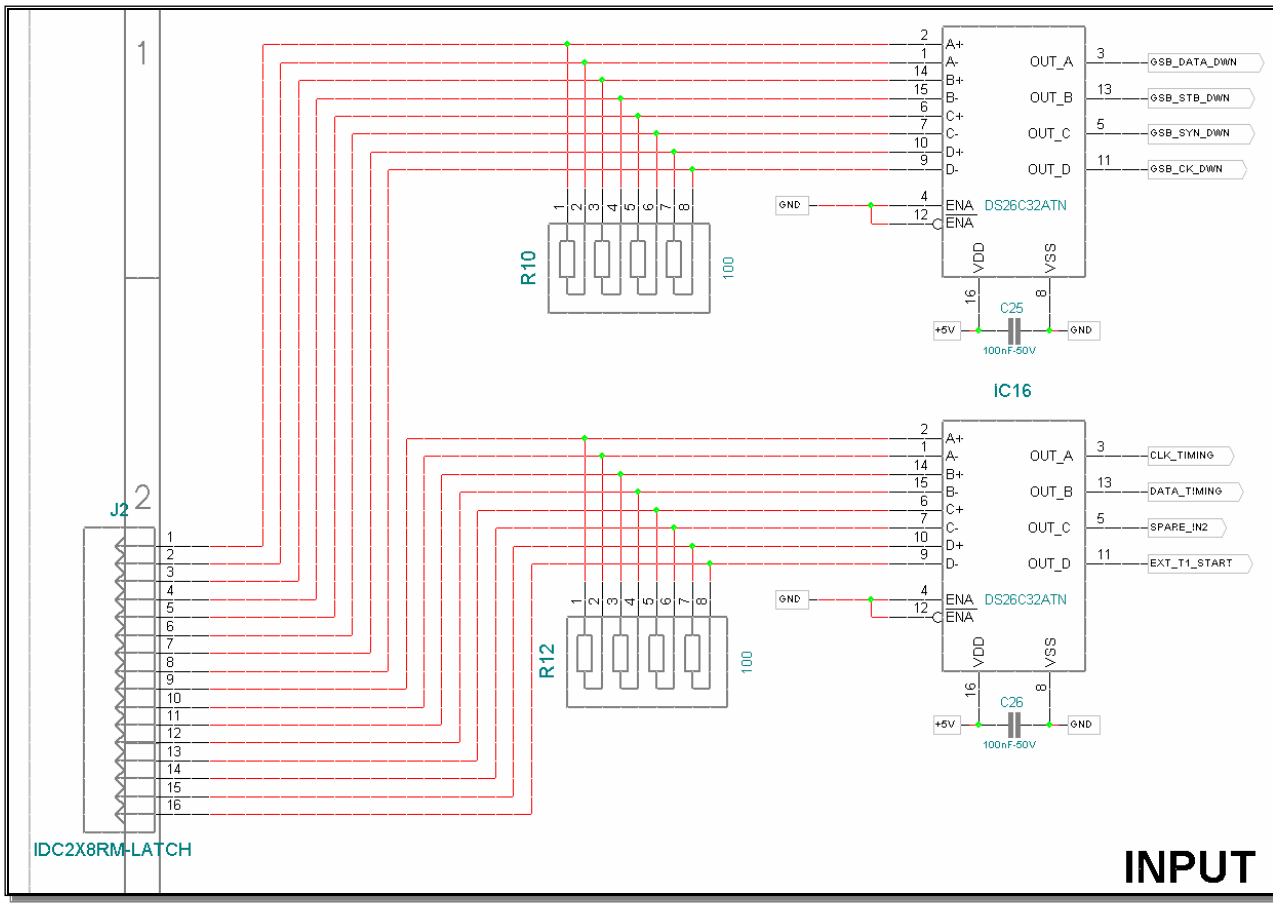

Figura 37: input

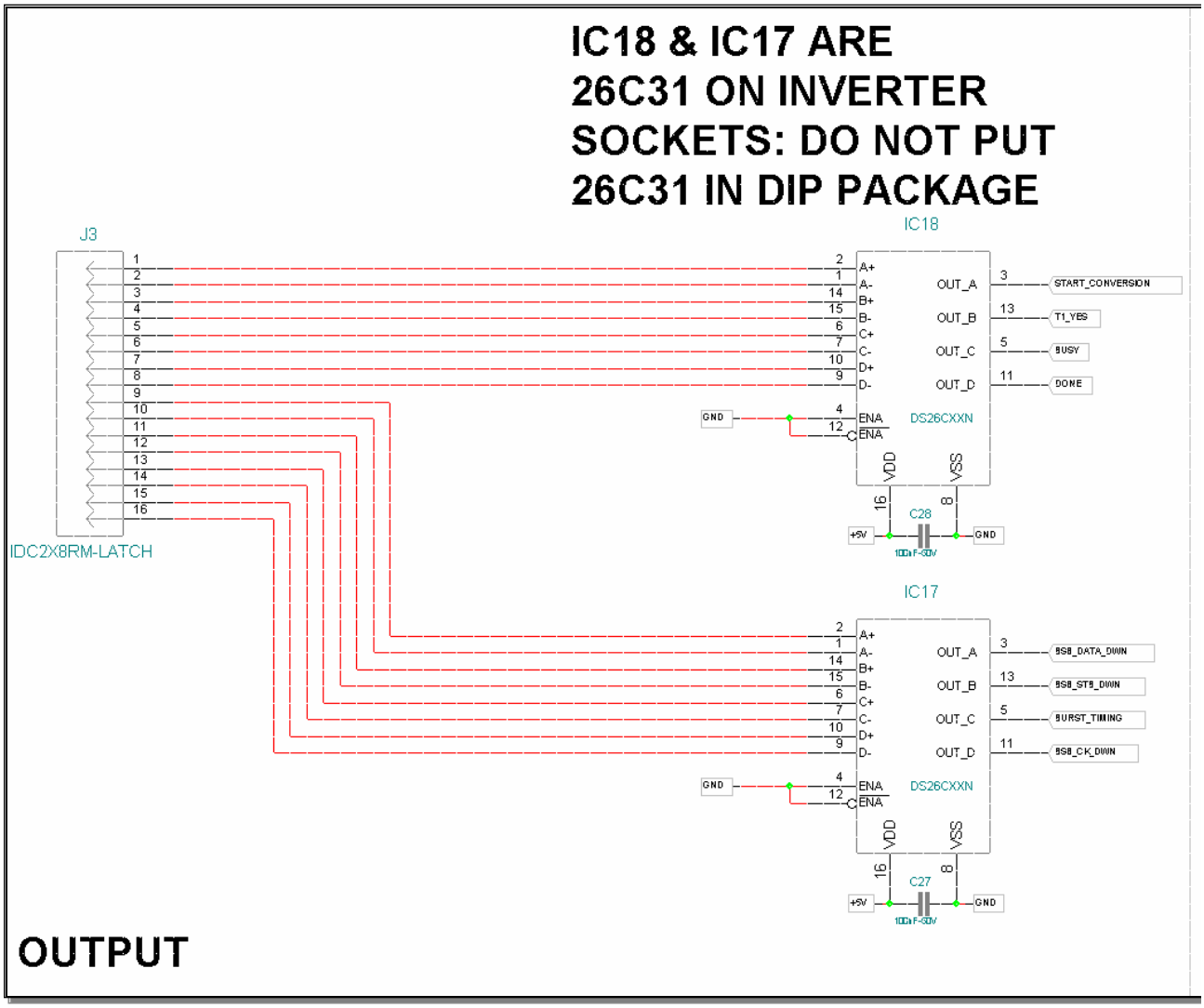

Figura 38: output