Reference design for The Digital BBC architecture

G. Comoretto[1], G. Tuccari[2]

[1]INAF - Osservatorio Astrofisico di Arcetri
[2]INAF - Istituto di Radioastronomia

Arcetri Technical Report N° 2/2008

Abstract

The Digital Broadband Converter (DBBC) is a system based on programmable logic, developed at the Noto-IRA radioastronomic station, that is used as a general purpose data acquisition system for radioastronomy. The system includes different board types: a fast Analog to Digital converter board (ADC1), a processing board (CORE1), and a service board (FILA1). Boards can be stacked together to build systems with different complexity. The CORE board contains a Xilinx field programmable gate array, that can be programmed to implement different instruments.

As many functions, like data packaging and unpackaging, computer interface, clock generation, are common to any data acquisition instrument, a library of service modules, has been produced, allowing the developer to write only the application specific code.

In this report these modules are described, together with the general architecture of the board.

1 Introduction

The DBBC instrument is a modular system designed to substitute the VLBI baseband converter, i.e. the last heterodyne stage of the VLBI receiver, with a completely digital system[1]. In the first version of the instrument the DBBC accepts up to 4 analog signals, with a bandwidth of 500 MHz each, and extracts from these a number (up to 32) of signals with a bandwidth of up to 31 MHz at an arbitrary starting frequency.

The DBBC is composed of a series of boards, stacked together. As the boards are completely programmable, and are based on Field Programmable Gate Array (FPGA) components, the DBBC can be used to implement a generic data acquisition instrument or radioastronomy.

To ease the task of developing new applications, a *skeleton* structure has been developed for the FPGA, that includes the chip pinout, the interface to the clock signals and to the external buses and a generic programming interface to the control computer.

1.1 System structure

The DBBC is composed of a stack of small size boards, that connect using an identical set of high speed connectors on both sides of each board. Three types of boards are available:

- ADC1: A data acquisition board, containing a high speed analog to digital converter. The converter has a sampling frequency of 1024 MHz, and thus a maximum instantaneous bandwidth of 500 MHz. The analog bandwidth extends to almost 2 GHz, allowing for Nyquist sampling in this range.
- FILA1. First-Last board in a stack. Provides control, configuration an programming signals for the instrument. The first board interfaces to a control computer using a PCI7200 parallel interface, and the last board contains electric drivers for a standard VLBI recorder, and a monitor DAC with 128 MHz sampling rate.
- CORE1: A general purpose processing board, contains a single XC2V6000 Xilinx Virtex2 FPGA.

Up to 4 ADC boards and 8 CORE1 boards can be stacked together. Two FILA1 boards are always required, to provide timing and control signals to the other boards.

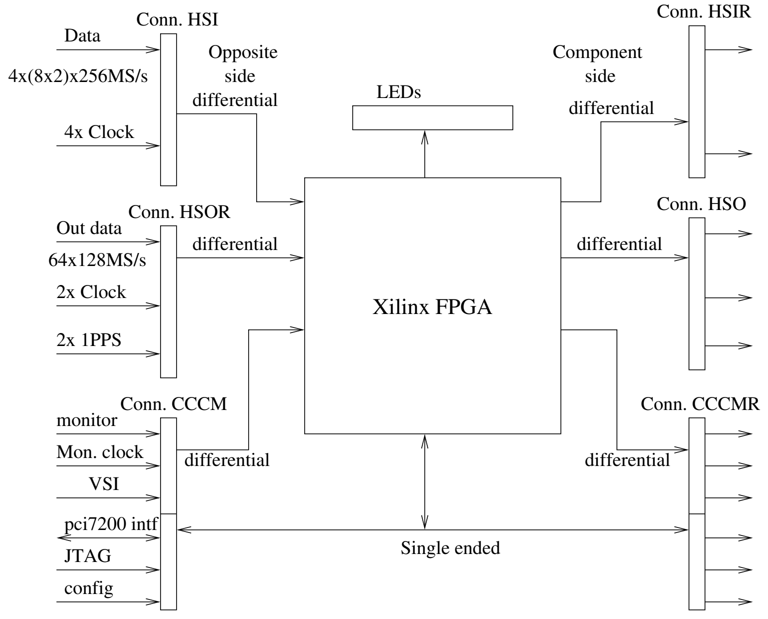

Figure 1: Structure of the CORE1 board

The CORE1 boards contain a high-end Xilinx chip, connected to a standard set of high speed buses (fig. 1):

- 4 *input buses*, each composed of a clock (256 MHz nominal), and 2 8-bit data paths
- One *high speed output*, composed of two clocks, 64 data lines, and 2 data valid bits
- One *Monitor bus*, composed of one clock, one 12-bit data bus, and two 1pps signals
- One *Control bus*, composed of one input and one output bus, 32 bit wide, and two strobe (trigger) lines.

All buses but the Control bus are differential daisy chained: each differential signal is originated in a board and fed only to the next one. It is then regenerated and propagated out on different physical lines. The physical signal can be modified or substituted inside the chip.

The input buses originate in up to 4 high-speed samplers. The DBBC version 1 standard supports samplers up to 1 GS/s, with consecutive pairs of samples stored on the two data lines, and using double-data-rate (DDR) signal transfer. Up to 4 samplers can be connected on the 4 input buses.

The output bus is fed to a VSI differential bus, typically connected to a high speed recorder. In the Version 2 DBBC standard, it can be serially transmitted on a 10 Gbit optical link.

The monitor bus is used, apart to propagate a station 1PPS signal, to send a user defined digital signal to a high speed DAC (128 MS/s, 64 MHz bandwidth), that can be used as an analog out or scope monitor.

The control bus is used to program and read back the logic inside the FPGAs. Each FPGA is assigned a unique 5 bit address, and up to 64 registers can be addressed within the chip. The bus is physically controlled by a ADLINK 7220 PCI parallel interface. The output bus is interpreted as an address part (upper 16 bits, with bit 31:26 to identify the chip) and a data part (lower 16 bits). The input bus is fully available to read a whole 32 bit quantity in a single read. The bus is paralleled, not daisy chained, and therefore the input bus (output on the FPGA) must be three-stated when not used.

These constrains are general, and independent on the specific application implemented in the chip. A general skeleton structure is therefore quite useful to provide basic input-output and control functionalities. The specific application can be implemented as a HDL module inside this skeleton, with minimal modifications.

The general structure of this skeleton is shown in chapter 2, followed by the structure of the sub-components.

2 Top level structure

The top level module is shown in figure 2. The module ports describe all the connected pins in the chip, with the buses described in the previous chapter. The module is mainly a container, with a limited number of signals and processes. Most functions are performed by hierarchic modules, including most input-output buffering.

The module contains:

- 4 `ddr_io` modules, each one dedicated to one of the four input buses. Each module copies the input bus to the corresponding replicated bus, regenerating clock and data, and decodes the dual DDR data streams at 256 MHz to a single 8 times multiplexed bus at 128 MHz. The module contains all differential buffers for data input/output and for clock output.
- 2 DCM modules. The first is used to regenerate the 256 MHz clock, producing 2 clock phases at 256 MHz and one clock phase at 128 MHz, and the second produces two phases at 128 MHz and the main system clock, also a 128 MHz. A divided clock, at 32 MHz, is also available. The modules contain all global clock buffers, input buffers, feedback logic and optionally logic necessary to re-phase the clock, if required.

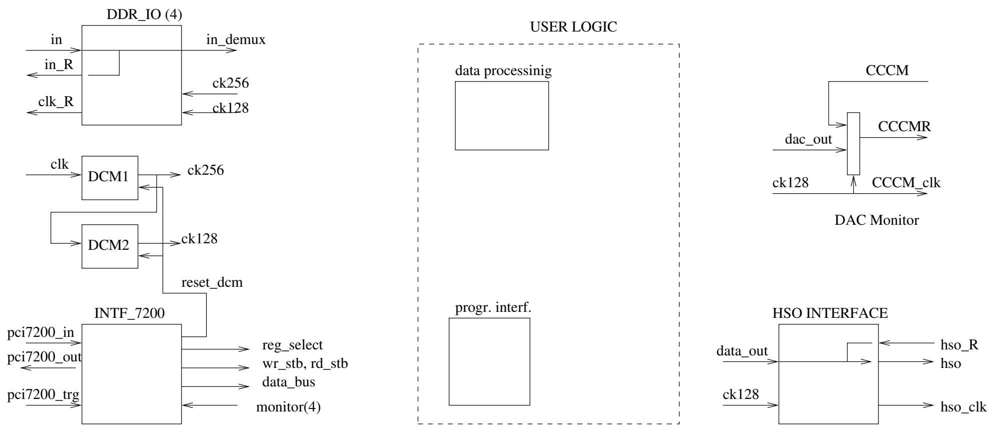

Figure 2: Block diagram of the skeleton design

- A PCI 7200 interface, that connects to the control bus. The block contains logic to address the chip, to separate portions of the address, and to separately address up to 4 sub-chips. Each sub-chip can receive a separate read and write strobe, a 6 bit register address, and can generate 32 bits of data. A reset line is activated when the last register in the chip address space is addressed, in a purely combinatorial way. All other operations are synchronized with the main chip clock.
- An interface to the HSO block. The interface copies the HSO bus from input to output pins, optionally substituting a range of pins with an user defined signal

2.1 User modifiable parts

Some functions are performed modifying this skeleton.

The chip address is hard-coded as a literal constant in the `addr` port of the `interface_7200` module.

The monitor DAC bus is not delegated to a subunit, but is processed directly in the `ccm_reg` process. The user may choose to propagate the CCM bus, or to substitute it with a user defined signal, `dac_out`.

The input clock signal is selected by connecting the two differential signals at the input of the first DCM module. This clock controls all the chip, but different structures using different input clocks are possible.

The HSO interface module is parametrized, and both the input bus width and the range to be substitute in the output bus can be selected. The latter can be chosen to be null, but some input must be present (see chapter 4.3).

2.2 User defined data types

The `dbbc_types` package defines custom data types used in this architecture. In particular, the type datain_t is an array of 8 signed quantities, each one 8 bit wide, used to describe the input path from the sampler.

The package defines also the basic components used in the skeleton and described in chapters 4, i.e. the `interface_dbbc`, the `ddr_io`, the `hso_interface`, the `cccm_interface`, and the two DCM clocks. Component description needs not to be included in the top level code, only component instantiation is required.

2.3 User defined logic

The user can specify up to 4 user defined blocks, in any combinations or architectures. The simplest way to implement a block is using a HDL module, with appropriate input and output signals, and with a control port interfacing to the PCI7200 interface module.

For example, a digital receiver could be implemented as a block, ad 4 such blocks could be packed in the chip. Or two digital receivers could be implemented together with a third block containing a spectropolarimeter, for a complete polarimetric receiver/backend.

The general structure of a block is as follows

```
entity user_block is
port (
  -- timing
    clk     : in std_logic;                      -- System clock
    stb     : in std_logic;                      -- 1 PPS pulse
  -- PCI7200 interface
    wd      : in std_logic_vector(15 downto 0); -- Write data
    wr_stb  : in std_logic;                  -- Write strobe pulse
    rd_stb  : in std_logic;                      -- Read strobe pulse
    monitor : out std_logic_vector(31 downto 0);-- Readback data
    reg_sel : in std_logic_vector(5 downto 0);  -- register addresss
  -- Actual input and output signals
    data    : in datain_t;                       -- data from the sampler
    out_data: out signed(7 downto 0);            -- processed data
    out_dac : out signed(7 downto 0));           -- To monitor DAC
end user_block;
```

All the complexity of data processing, user registers, etc are hidden inside this block. The PCI7200 interface protocol is described in detail in chapter 3, and all other signals have an intuitive behavior.

3 Programming interface

The programming interface, based on the pci7200 parallel interface, allows the user to specify parameters and commands for the instrument implemented in the chip, and to read system status data and observation results.

The control program must provides address and data values on the 32 bit parallel output port, followed by an active-low strobe pulse. The peripheral implemented in the DBBC chip must then decode the address, program the relevant register with the specified value, and/or provide a value to be read back on the trailing edge of the strobe pulse. It is assumed that both operations occur in a much shorter time than the pulse width, thus avoiding the need for a synchronization protocol or wait states. For longer operations, a status bit can be read back, signaling data availability.

In the original interface protocol, bits 31-27 of the parallel port were used to specify the chip in the system, bits 26-21 to specify one of 64 possible registers, and the remaining bits were available to specify the register value. To address up to 4 sub-chips (blocks), the upper 2 bits of the register address specify the block, and the remaining 4 one of 16 registers in each block. Only 16 bit values are usually written,and this has been assumed to be always the case. Therefore, bits 16-20 of the parallel interface port are not used.

A slightly different protocol has been implemented, with bits 19-16 used to specify one of 4 independent blocks in the chip. Each bit addresses a specific block, and any combination of blocs can be addressed in parallel. This, apart from extending the total number of registers in the chip, allows to program identical blocks in parallel. For example, in a polarimetric receiver, it is useful to program the two polarization channels simultaneously, to guarantee phase coherence of the local oscillators.

Therefore two different modules have been provided, implementing the two different addressing scheme. These modules are identified by the architecture, with architecture `behavioral6bit` for the original (6 bit address) interface, and architecture `behavioral` for the new interface.

The port interface of the tho modules is identical. Each block receives one read and one write strobe, and a 6 bit register address. In the 6 bit interface, the upper 4 bits of the register address are always 0, and only one write strobe is active at each time.

The write strobe is active for one clock cycle, corresponding to the leading edge of the strobe pulse. The specified register must be written when the strobe is high. Register size can be any number of bits.

Zero bit registers are possible, and are used to trigger operations (e.g. to synchronize different chips, or to start a long operation). Up to 16 bits can be specified with a single write. Using multiple writes, longer values can be specified. For convention, multi-word values are specified MSB first.

The read strobe is active for one clock cycle at the trailing edge of the strobe pulse. It can be used to signal that the specified register has been read. This is especially useful for status bits, that can be reset after read, and to read long memories (e.g. spectral data), using multiple read operations on the same address. All 32 bits of the parallel port are available for read, so up to 32 bits can be read at once The value to be read must be presented on the output port either at the write strobe, or asynchronously using the address values. The interface takes care to select the output from the correct block, ad to three-state the output lines when the chip is not addressed. For most read operations the read strobe can thus be ignored.

3.1 Module port description

The VHDL definition of the interface module is the following:

```
component interface_dbbc is
  port (
    wd          : in std_logic_vector(31 downto 0); -- Data from 7200
    trig        : in std_logic;                     -- Trigger input from 7200
    rd          : out std_logic_vector(31 downto 0);-- Data to 7200
    board_addr  : in std_logic_vector(4 downto 0);  -- Chip address
    clock       : in std_logic;                     -- Main system clock
    wr_enable   :  out std_logic_vector (3 downto 0);   -- write strobe
    rd_enable   :  out std_logic_vector (3 downto 0);   -- read strobe
    data        :  out std_logic_vector (15 downto 0);  -- data to sub-chips
    reg_addr    :  out std_logic_vector (5 downto 0);   -- register select
    reset_dcm   :  out std_logic;                       -- Reset to DCM
    -- Monitor (read) input from 4 blocks
    monitor0    : in std_logic_vector(31 downto 0) := (others => '0');
    monitor1    : in std_logic_vector(31 downto 0) := (others => '0');
    monitor2    : in std_logic_vector(31 downto 0) := (others => '0');
    monitor3    : in std_logic_vector(31 downto 0) := (others => '0'));
end component;
```

Ports wd, trig and rd connect to the corresponding pins of the pci7200 bus. The port `board_addr` can be connected either to an hardcoded value, or to a signal derived from an externally generated geographic address. The current version of the CORE1 board has no such pins, but in principle it could be possible to use a subset of the daisy chained buses to automatically generate a daisy chained address.

Write and read strobes are generated using the addressing scheme described in the previous chapter. Timing is shown in fig. 3. Data and register select are sent to each block in parallel, while monitor words are separate for each block. All signals are synchronized with the main system clock.

The reset DCM signal is completely asynchronous, and can be turned on and off even when the main clock is not operating, as it is the case when the reset line itself is asserted.

4 Detailed component description

The library is organized as a *skeleton* Xilinx project. The application performs all the functions necessary for clock generation, signal propagation on the various buses, etc, but has no programmable registers and perform no operations on the sampled data. All output data are either identically zero, or the delayed copy of the input data.

The components of this project include a top level entity, to be used as a template for a real application, and several modules, described here.

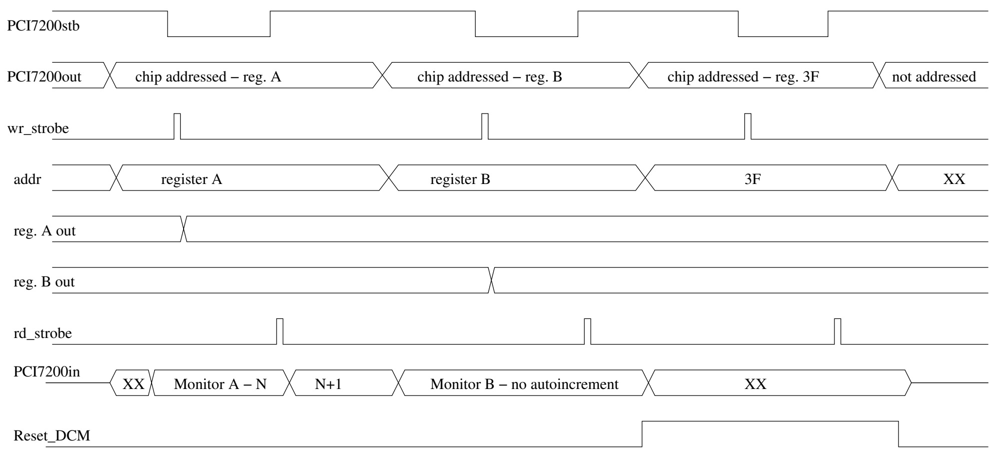

Figure 3: Timing of the programming interface

4.1 Clock generation

Two modules implement the two clock generators used in the system, resp. for the 256 MHz and 128 MHz clocks. Both modules have been derived using the Xilinx core generation routine, and are basically an envelope around the DCM primitive component.

The two modules are basically an envelope for the DCM primitive. The relevant global clock buffers are also included.

4.2 DDR Input/output

Sampler data are transmitted on the high speed input bus as two streams of double-data-rate (DDR) signals, and a clock at 256 MHz, for a total of 1024 Msamples/s.

There are four independent data streams, and the chip must at least propagate each stream from the input to the output port. Each sample is 8 bit wide, in offset binary representation. Each stream is dealt by a `ddr_io` module. Each input is physically represented as a differential signal, in LVPECL standard.

In principle, each stream has an independent clock, but for simplicity it has been assumed here that all clocks are identical. One clock is thus used for all functions in the system (see previous chapter).

Each stream contains a *primary* and a *auxiliary* data path. For 1024 MS/s, even samples (0, 2, . . .) use the primary path, and odd ones (1, 3, . . .) the auxiliary path. The module de-multiplexes 4 consecutive samples for each stream in a single 8x output stream. In the output stream, sample 0 is the earlier one, and 7 the later one. The output format of each sample is 8 bit signed, i.e. with the MSB negated with respect to input. Module ports are:

```
component ddr_io is
port (
  -- Input data, primary and secondary bus
   in_pri_p,in_pri_n    : in std_logic_vector(7 downto 0);
   in_aux_p,in_aux_n    : in std_logic_vector(7 downto 0);
  -- clock at 256 MHz and 128 MHz, 2 phases each
   clk0, clk180         : in std_logic;
   clk128M, clk128M_inv : in std_logic;
  -- Output data, primary and secondary bus
   out_aux_p,out_aux_n  : out std_logic_vector(7 downto 0);
   out_pri_p,out_pri_n  : out std_logic_vector(7 downto 0);
  -- Output clock, 256MHz
   out_clk_p,out_clk_n  : out std_logic;
  -- Data to system
```

```
   data);      : out datain_t  -- 8x multiplexed 8 bit signed
end component;
```

4.3 HSO Interface

The High Speed output port is composed of 64 data lines, 2 clock lines, 2 1PPS strobe pulses, and 2 data valid signals. Each signal has a sample rate of 128 MS/s, and is electrically coded as differential LVDS.

The module substitutes a predefined range of the HSO bus with signals from its output. For example, a digital BBC may produce 8 2-bit data streams, that substitute bits 16-31 of the HSO bus.

Port definition

The module accepts three integer generics, specifying the range of lines in the HSO bus that is substituted with user provided data, and connects directly to the differential HSO and HSOR pins.

All output clocks are generated from system clock (`clk` input). The module copies all `hsoR` input lines to the corresponding `hso` outputs, substituting `output_size` lines, starting at index `index_first`, with the first lines from the input port `output_data`. If `output_size` is zero, no lines are substituted, but `output_data` cannot have zero size.

```
component hso_interface
generic (
   index_first   : integer;  -- First output to substitute
   output_size   : integer;  -- number of bytes to substitute (may be 0)
   input_size    : integer); -- Dimension of data to substitute (> 0 )
port (
  -- HSO input
   hsoR_p, hsoR_n        : in  std_logic_vector(63 downto 0);--input
   hsoRclk1_p,hsoRclk1_n : in std_logic;    --input clock VSI1
   hsoRclk2_p,hsoRclk2_n : in std_logic;    --input clock VSI2
   hsoR1pps1_p,hsoR1pps1_n  : in std_logic; --input 1PPS VSI1
   hsoR1pps2_p,hsoR1pps2_n  : in std_logic; --input 1PPS VSI2
  -- HSO output
   hso_p, hso_n          : out std_logic_vector(63 downto 0);--output
   hsoclk1_p,hsoclk1_n   : out std_logic;   --output clock
   hsoclk2_p,hsoclk2_n   : out std_logic;   --output clock
   hso1pps1_p,hso1pps1_n : out std_logic;   --output 1PPS
   hso1pps2_p,hso1pps2_n : out std_logic;   --output 1PPS
   hso1ppsvsi1_p,hso1ppsvsi1_n : out std_logic;      --output 1PPS VSI1
  -- data valid I/O
   pvalid1_p,pvalid1_n   : in std_logic;   --input validity VSI1
   pvalid2_p,pvalid2_n   : in std_logic;   --input validity VSI2
   pvalidR1_p,pvalidR1_n : out std_logic;  --output validity VSI1
   pvalidR2_p,pvalidR2_n : out std_logic;  --output validity VSI2
  -- Input from system
   clk                   : in std_logic;   -- system clock
   out_data              : in std_logic_vector(input_size-1 downto 0);
   hso1pps               : out std_logic); -- 1pps to system
end component;
```

4.4 CCCM Interface

The interface allows to dynamically connect a signal to the monitor DAC. The module connects directly to the differential pins of the CCCM bus, and if `enable` is set to 0 simply propagates the bus. Clock is derived from system clock `clk`. If `enable` is set to 1, then the signed signal on port `dac_out` is sent on the monitor bus, substituting the signal form the previous CORE1 card.

```
component cccm_interface
```

```
port (
   clk             : in std_logic;   -- system clock
   enable          : in std_logic;   -- = '1' => substitute cccm with dac_out
   dac_out         : in signed (11 downto 0);           -- Data from system
   clkM_p,clkM_n   : in std_logic;                      -- Monitor Clk,
   cccM_p,cccM_n   : in  std_logic_vector(11 downto 0); -- Monitor Data,
   clkMR_p,clkMR_n : out std_logic;                     -- CCMR Monitor clk,
   cccMR_p,cccMR_n : out std_logic_vector(11 downto 0));-- CCMR Monitor Data,
end component;
```

References

[1] Tuccari, G.: DBBC - a Wide Band Digital Base Band Converter, *IVS 2004 General Meeting Proceedings*, 234-237, `ftp://ivscc.gsfc.nasa.gov/pub/general-meeting/2004/psgz/tuccari3.ps.gz`

[2] ADLINK PCI7200 programming manual

Contents

1 Introduction **1**
1.1 System structure 1

2 Top level structure **2**
2.1 User modifiable parts 3
2.2 User defined data types 3
2.3 User defined logic 3

3 Programming interface **4**
3.1 Module port description 5

4 Detailed component description **5**
4.1 Clock generation 6
4.2 DDR Input/output 6
4.3 HSO Interface 7
4.4 CCCM Interface 7