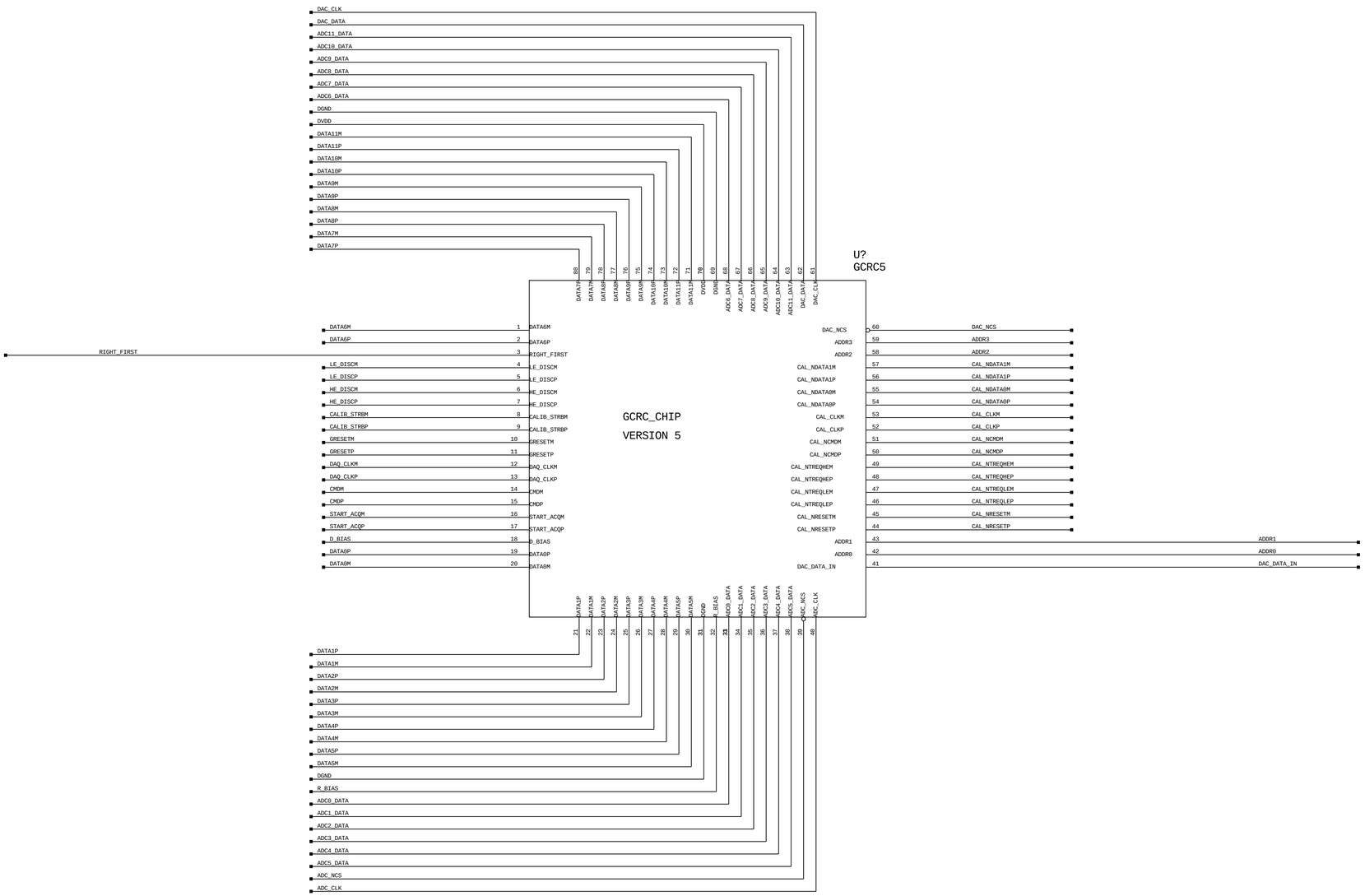
REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



Note: See Appendix, Page 61, for VHDL code

GCRC_TOP

3-6-2003_10:06

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY — STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

GCRC_CHIP

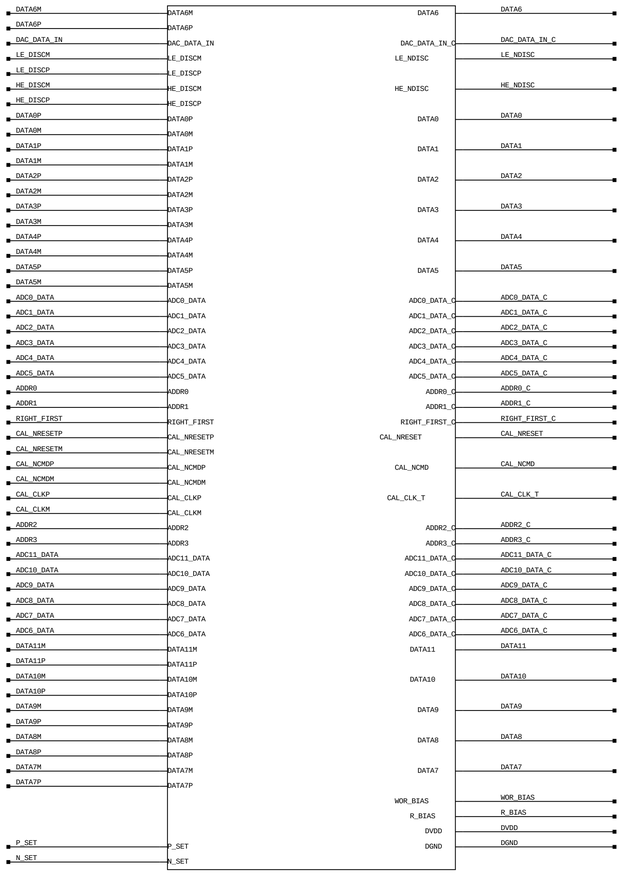
VERSION 5

LAT-DS-03519-50

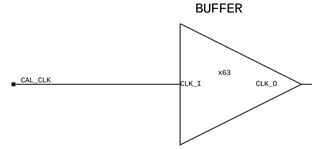
D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE

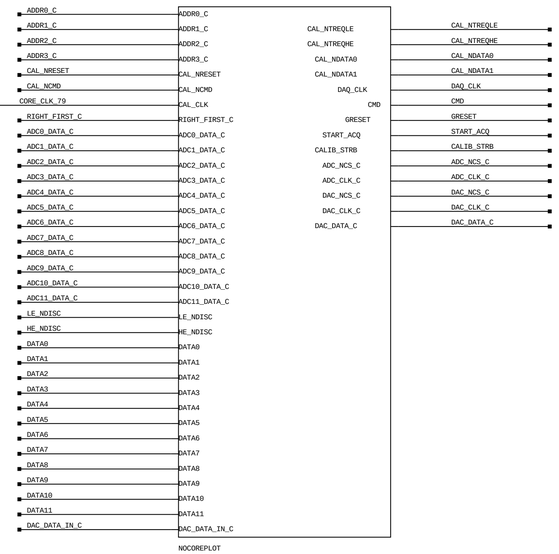
GCRC_INPUTS



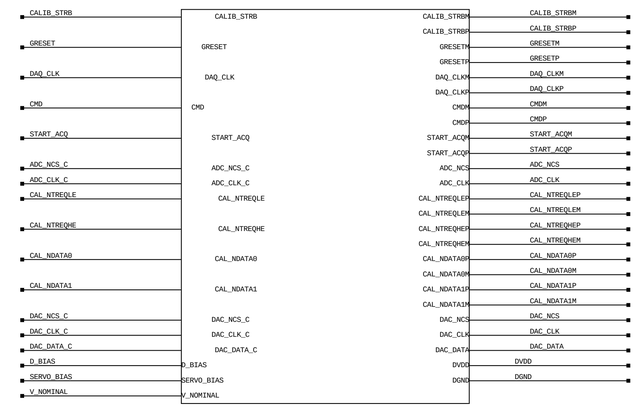
CORE CLOCK BUFFER



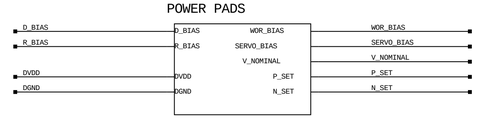
GCRC_CORE

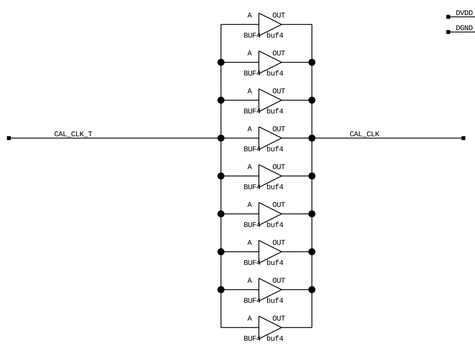


GCRC_OUTPUTS



BIASES and POWER PADS





GCRC_VER5

3-6-2003_10:05

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY — STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

GCRC_CHIP

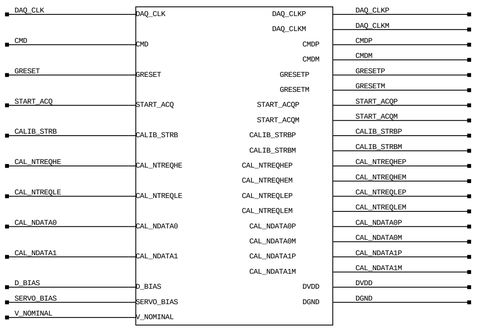
VERSION 5

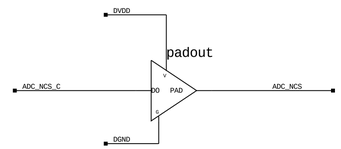
LAT-DS-03519-50

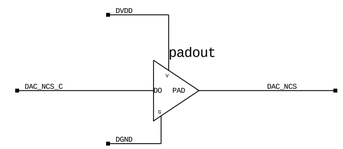
D

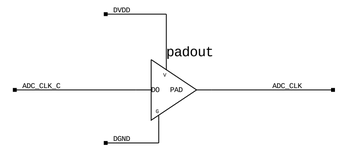
REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE

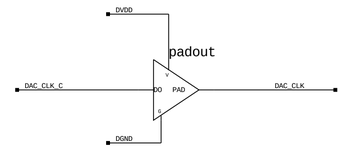
LVDS TRANSMITTERS

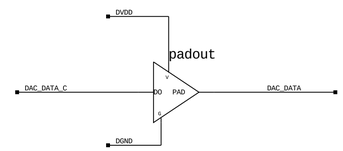












GCRC_OUTPUTS

2-24-2003_11:36

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

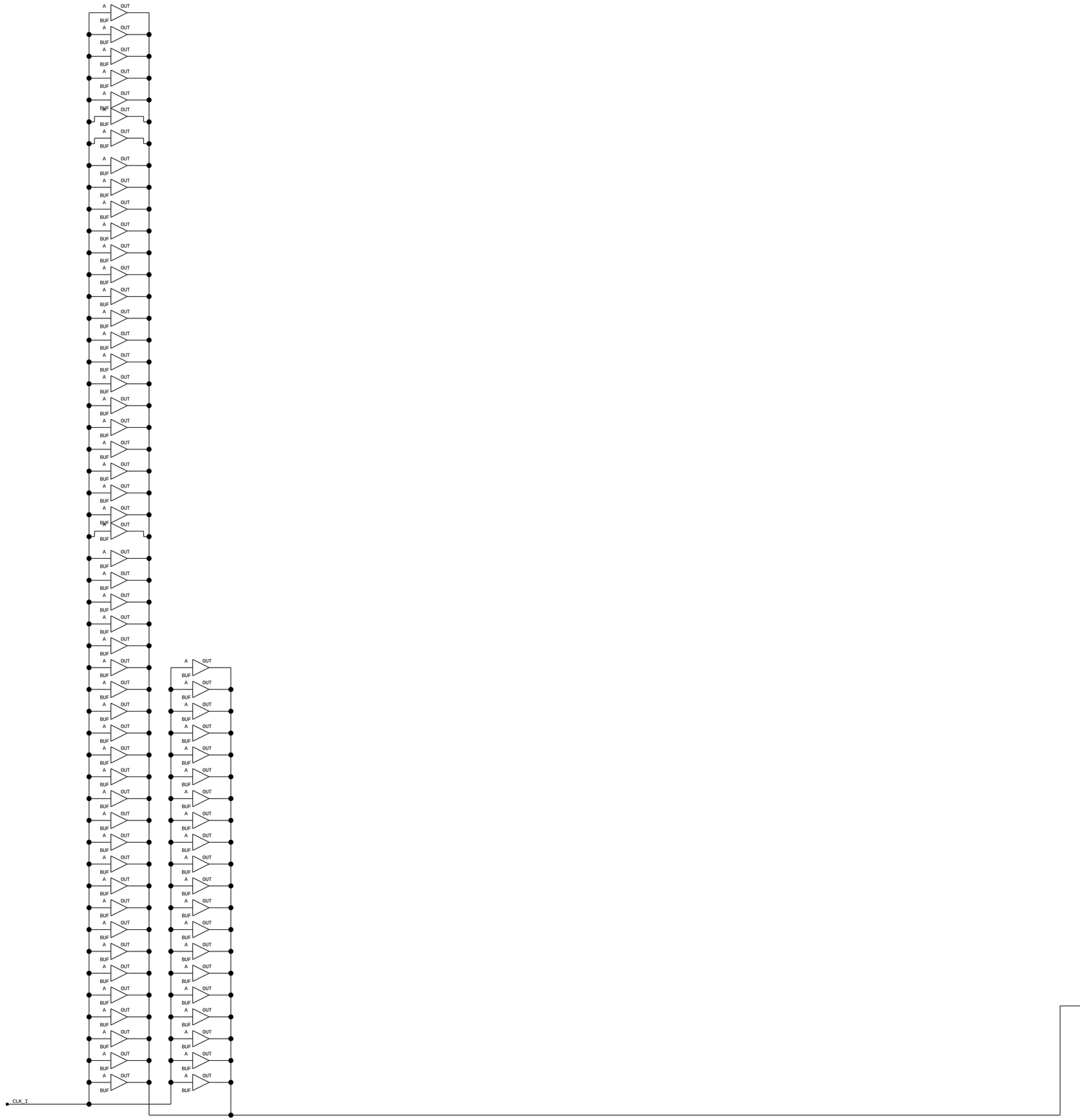
GCRC_CHIP

OUTPUT PADS

VERSION 5

LAT-DS-03519-50 D

50 per column



REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE

CORE_CLK_BUF

2-24-2003_11:34

STANFORD LINEAR ACCELERATOR CENTER
U.S. DEPARTMENT OF ENERGY
STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

SHEET 6 OF 60

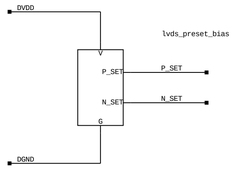
GCRC_CHIP

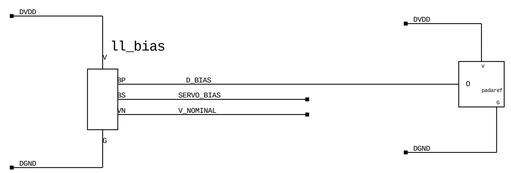
VERSION 5

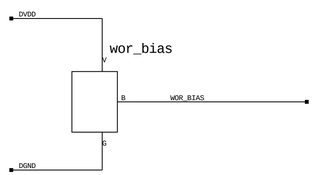
LAT-DS-03519-50

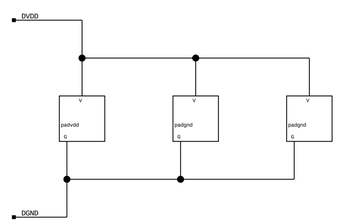
D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE









BIASES

3-6-2003_9:53

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

GCRC_CHIP

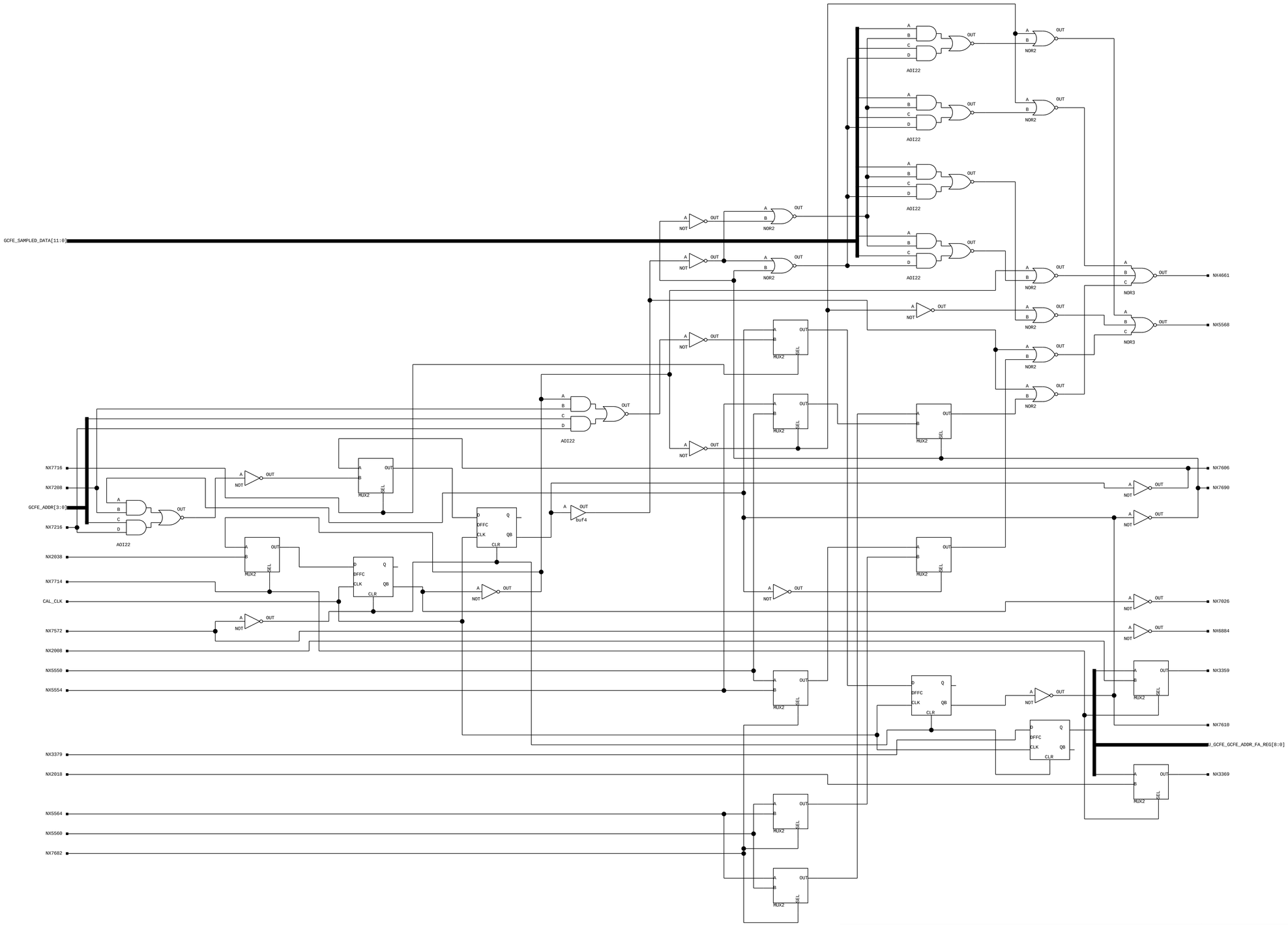
BIASES

VERSION 5

LAT-DS-03519-50

D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

SHEET OF 60

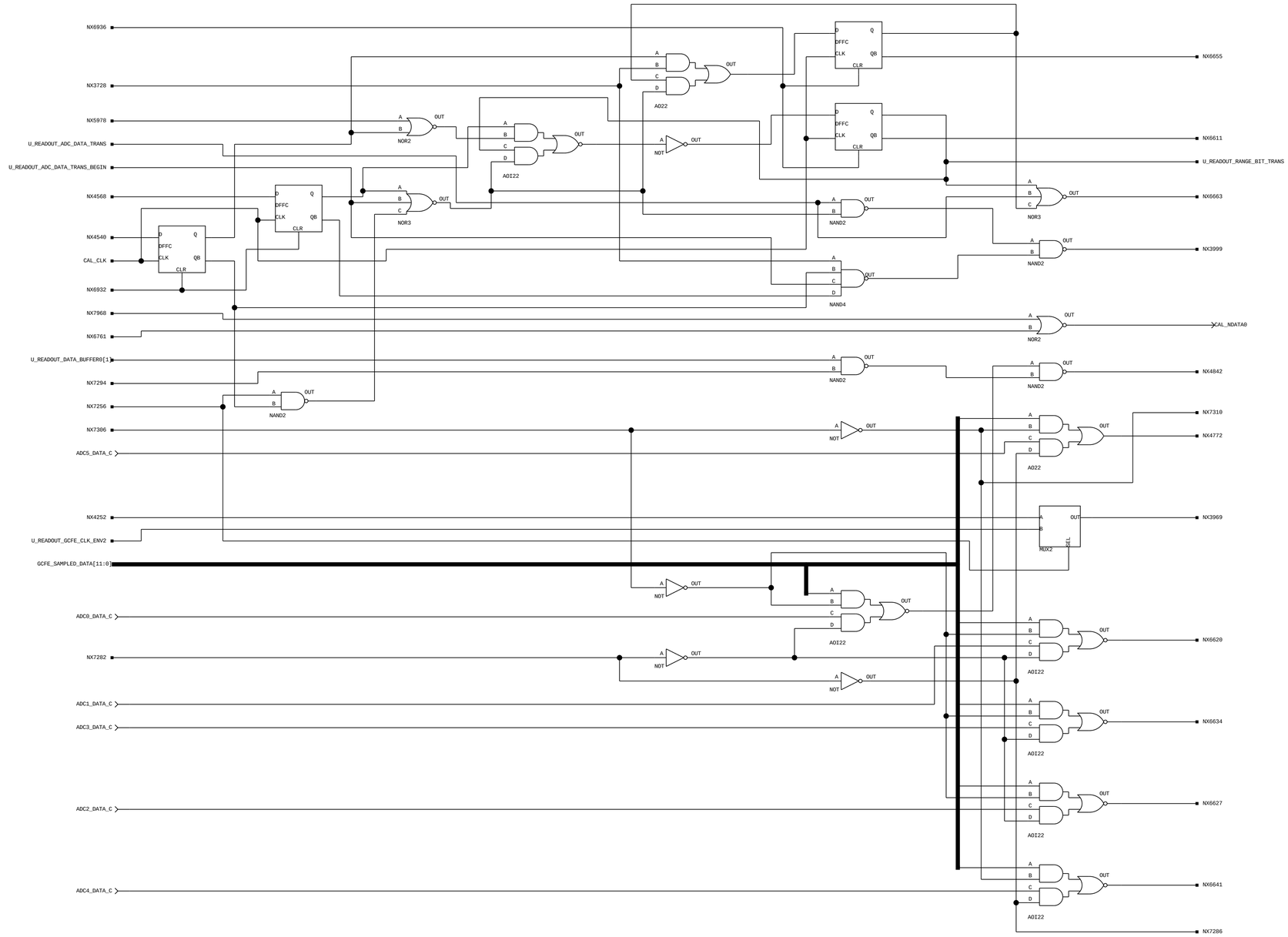
GCRC_CHIP

VERSION 5

LAT-DS-03519-50

D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



STANFORD LINEAR ACCELERATOR CENTER
U.S. DEPARTMENT OF ENERGY
STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

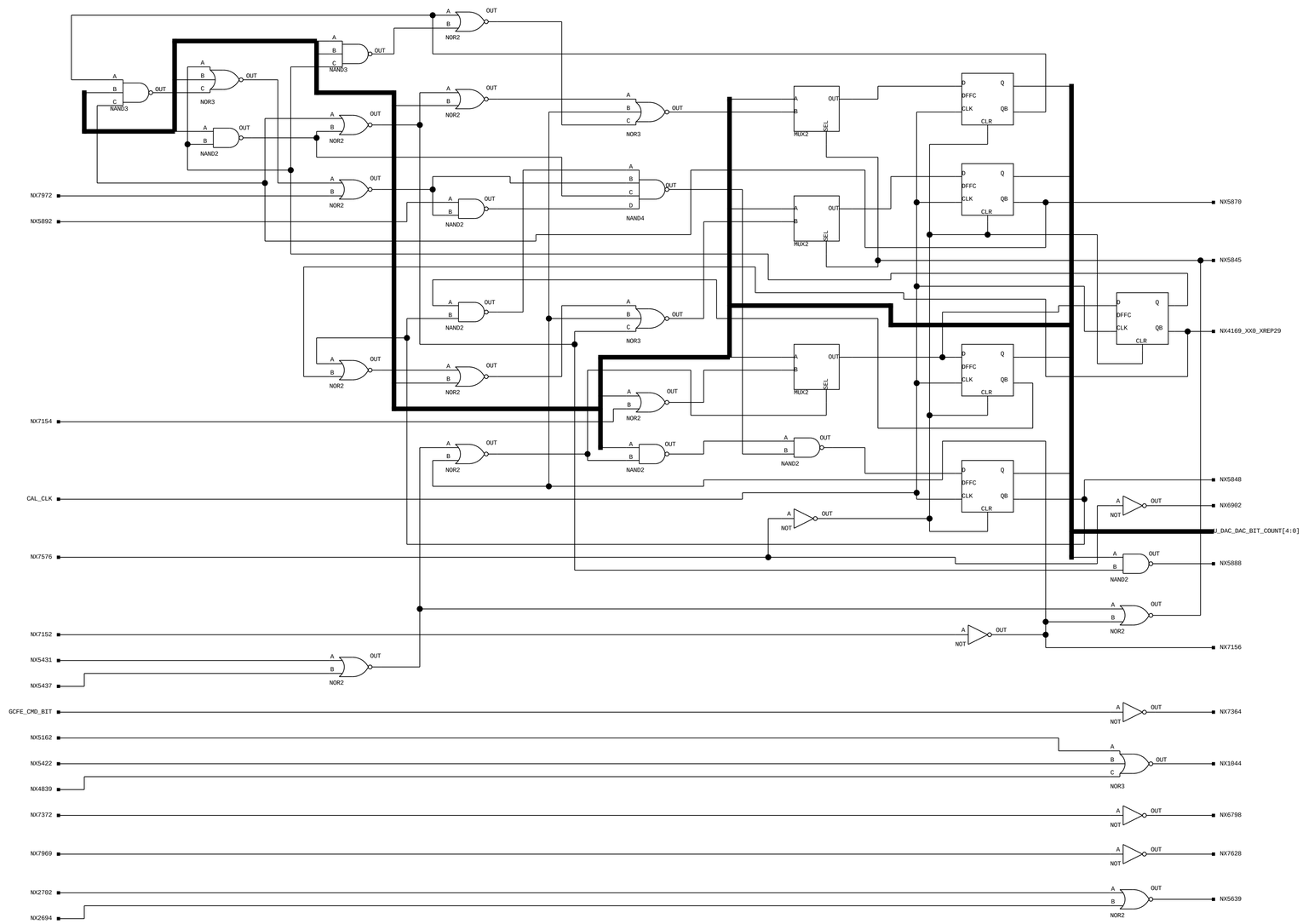
SHEET OF 60

GCRC_CHIP

VERSION 5

LAT-DS-03519-50 D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



STANFORD LINEAR ACCELERATOR CENTER
U.S. DEPARTMENT OF ENERGY
STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

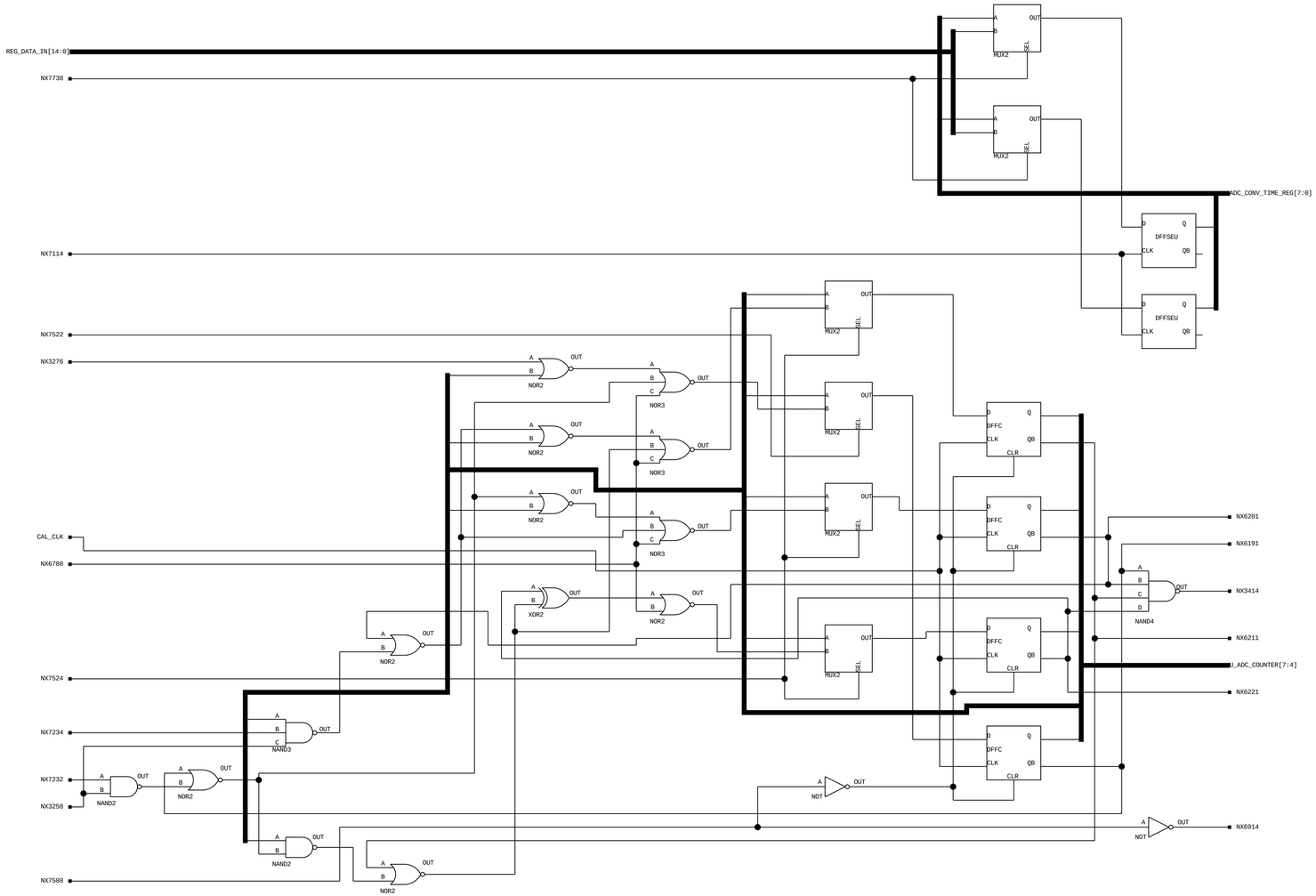
		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

GCRC_CHIP

VERSION 5

LAT-DS-03519-50 D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



STANFORD LINEAR ACCELERATOR CENTER
U.S. DEPARTMENT OF ENERGY
STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

SHEET OF 60

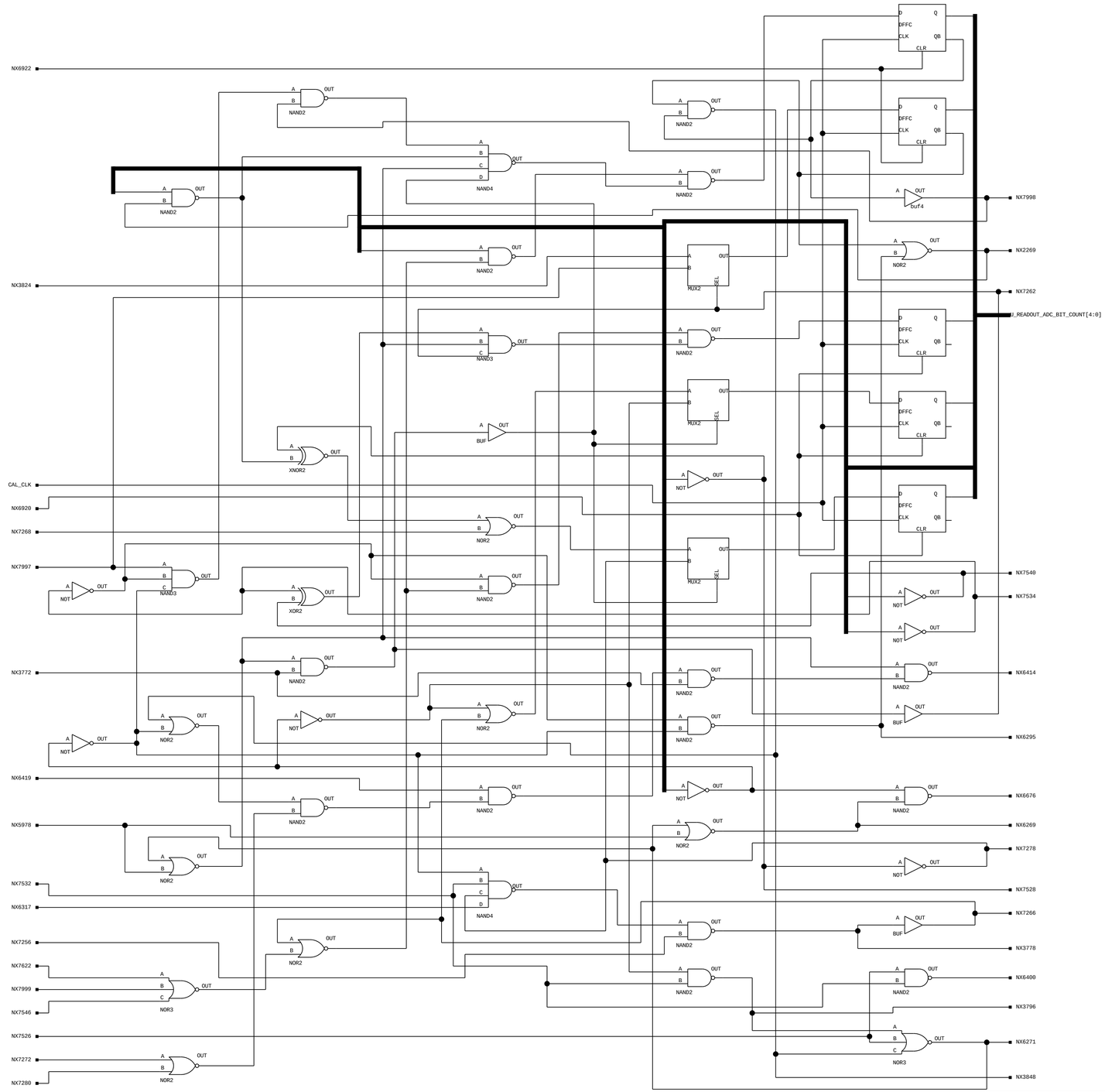
GCRC_CHIP

VERSION 5

LAT-DS-03519-50

D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



STANFORD LINEAR ACCELERATOR CENTER
U.S. DEPARTMENT OF ENERGY
STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Amye - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

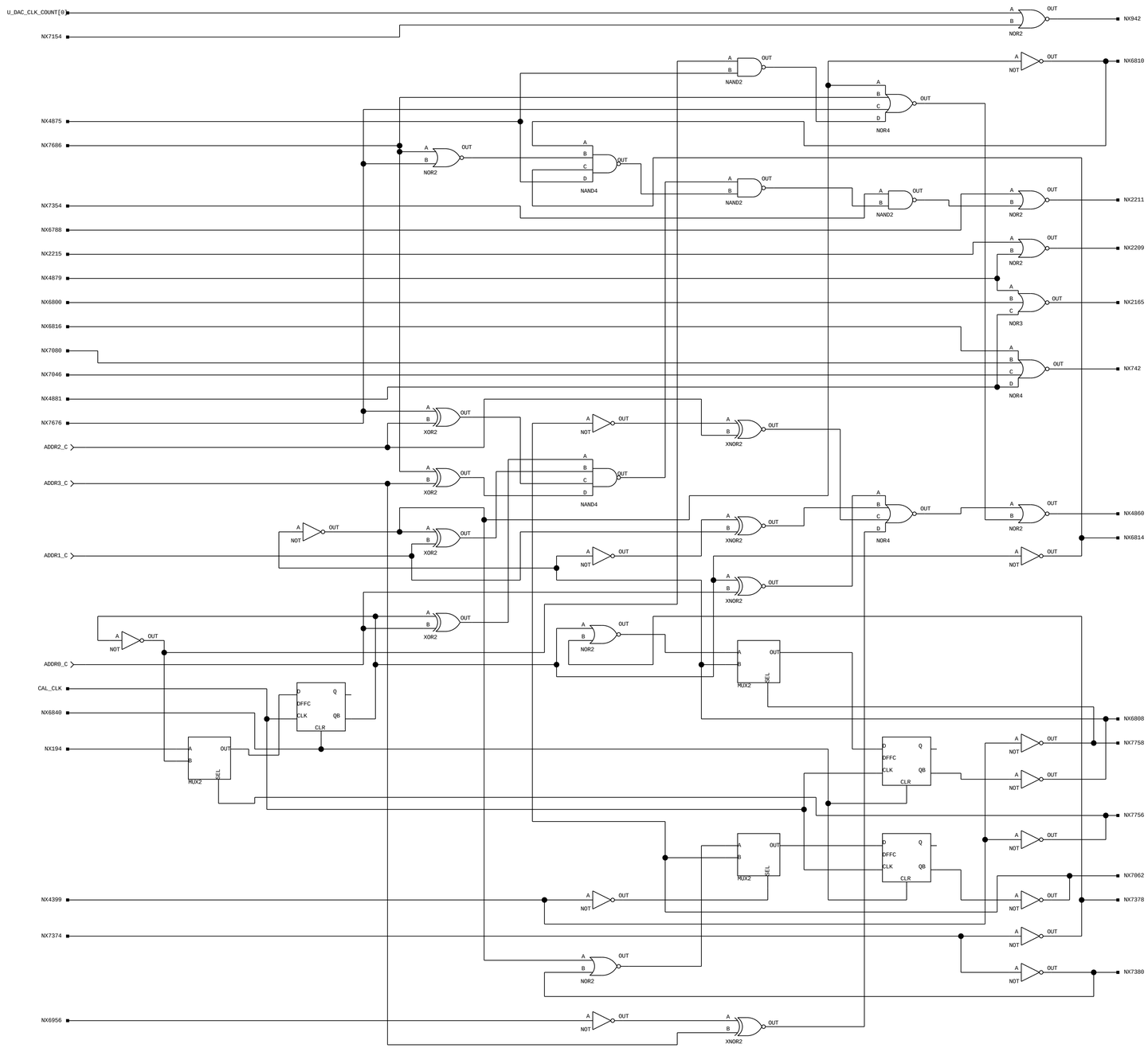
SHEET OF 60

GCRC_CHIP

VERSION 5

LAT-DS-03519-50 D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY — STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

SHEET OF 60

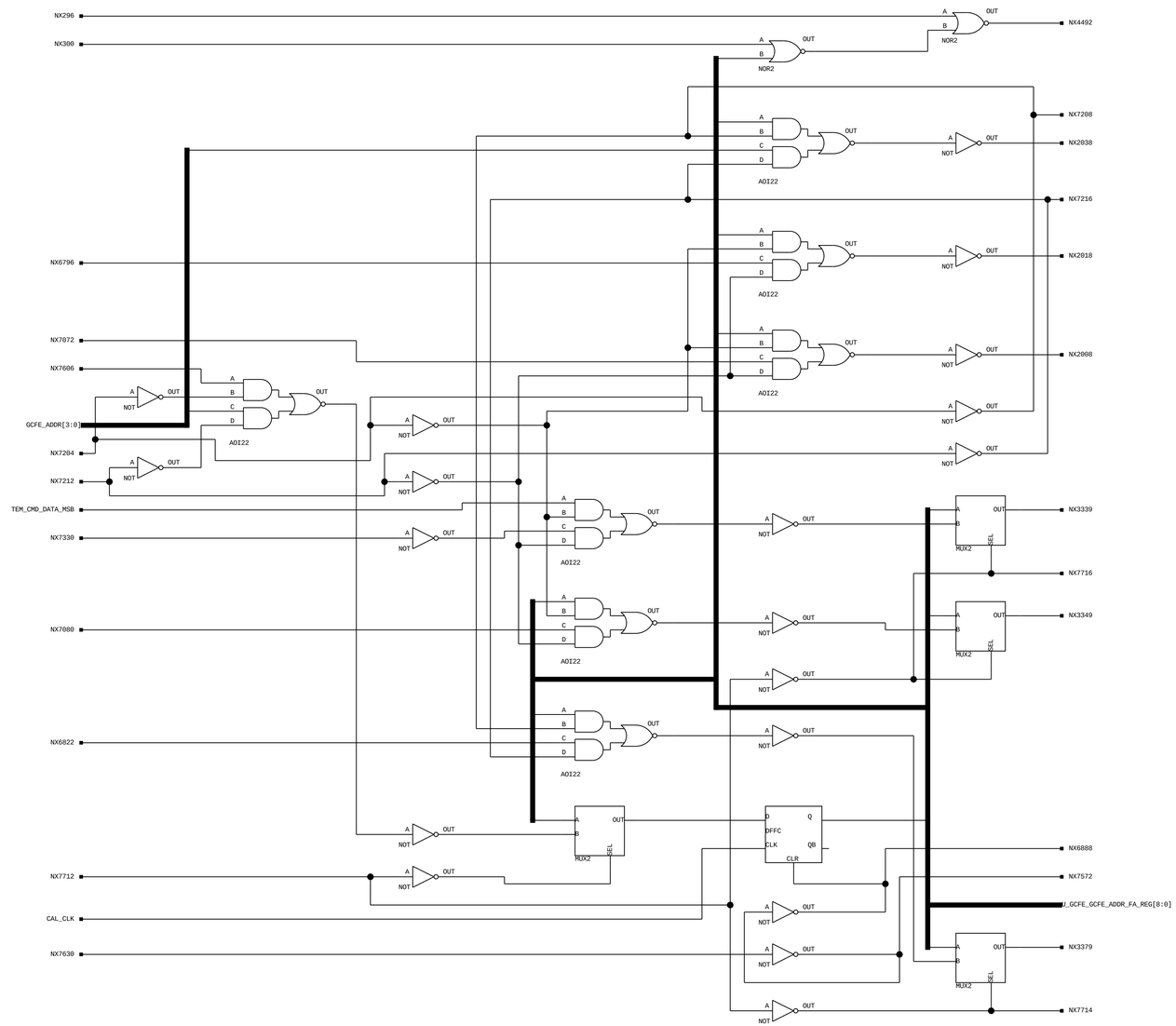
GCRC_CHIP

VERSION 5

LAT-DS-03519-50

D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



STANFORD LINEAR ACCELERATOR CENTER
U.S. DEPARTMENT OF ENERGY
STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

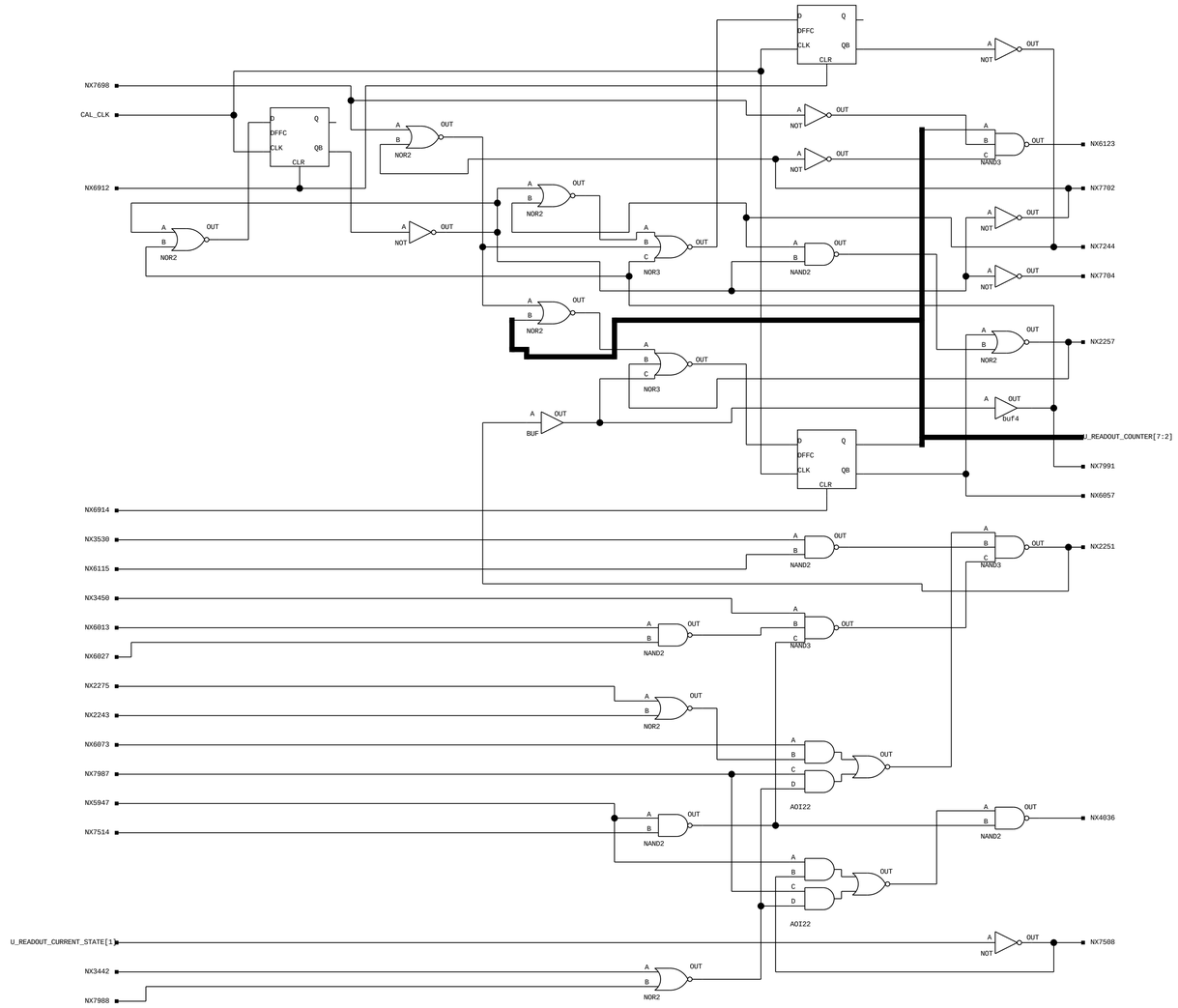
		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

SHEET OF 60

GCRC_CHIP

VERSION 5

LAT-DS-03519-50 D



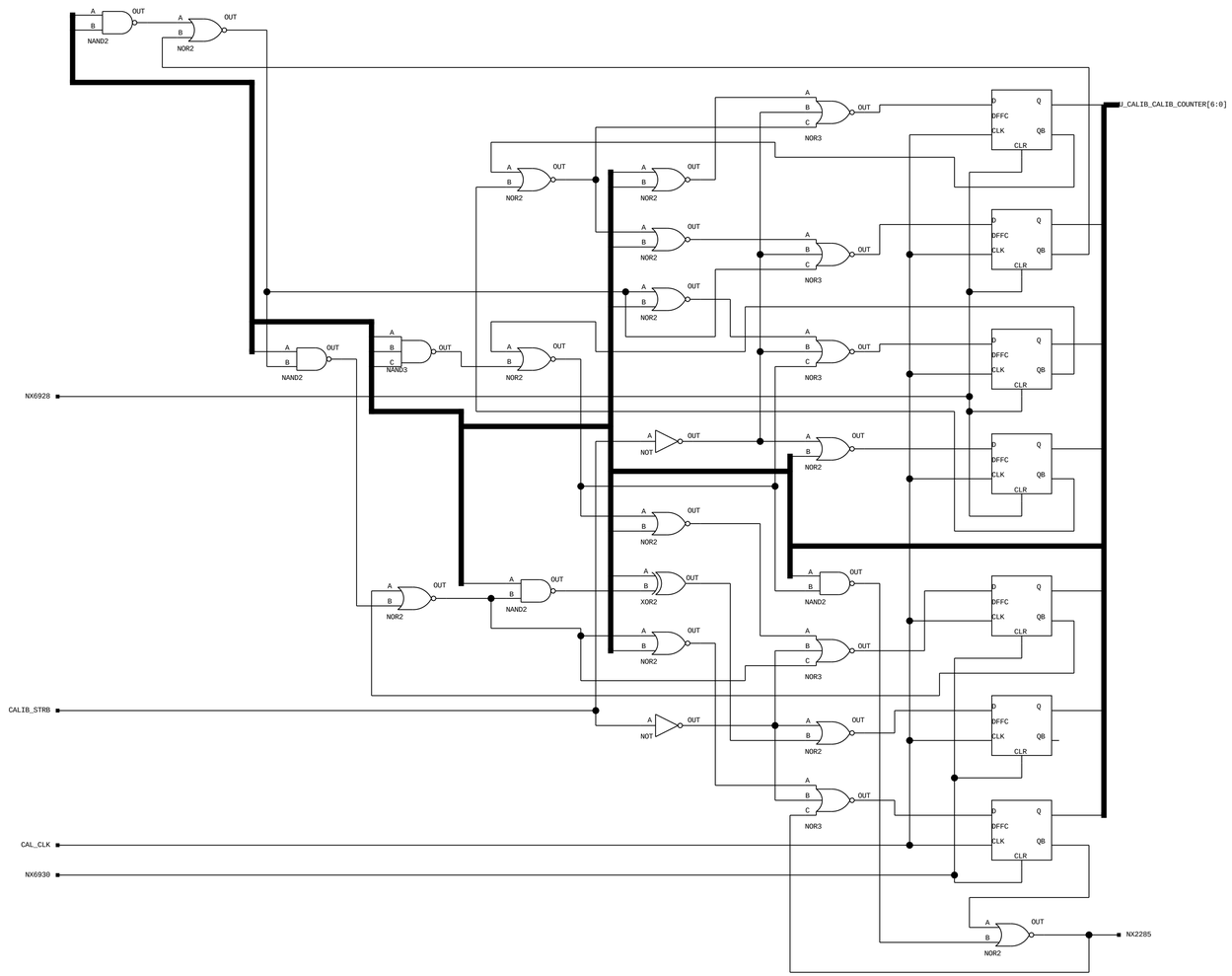
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 8 OF 51

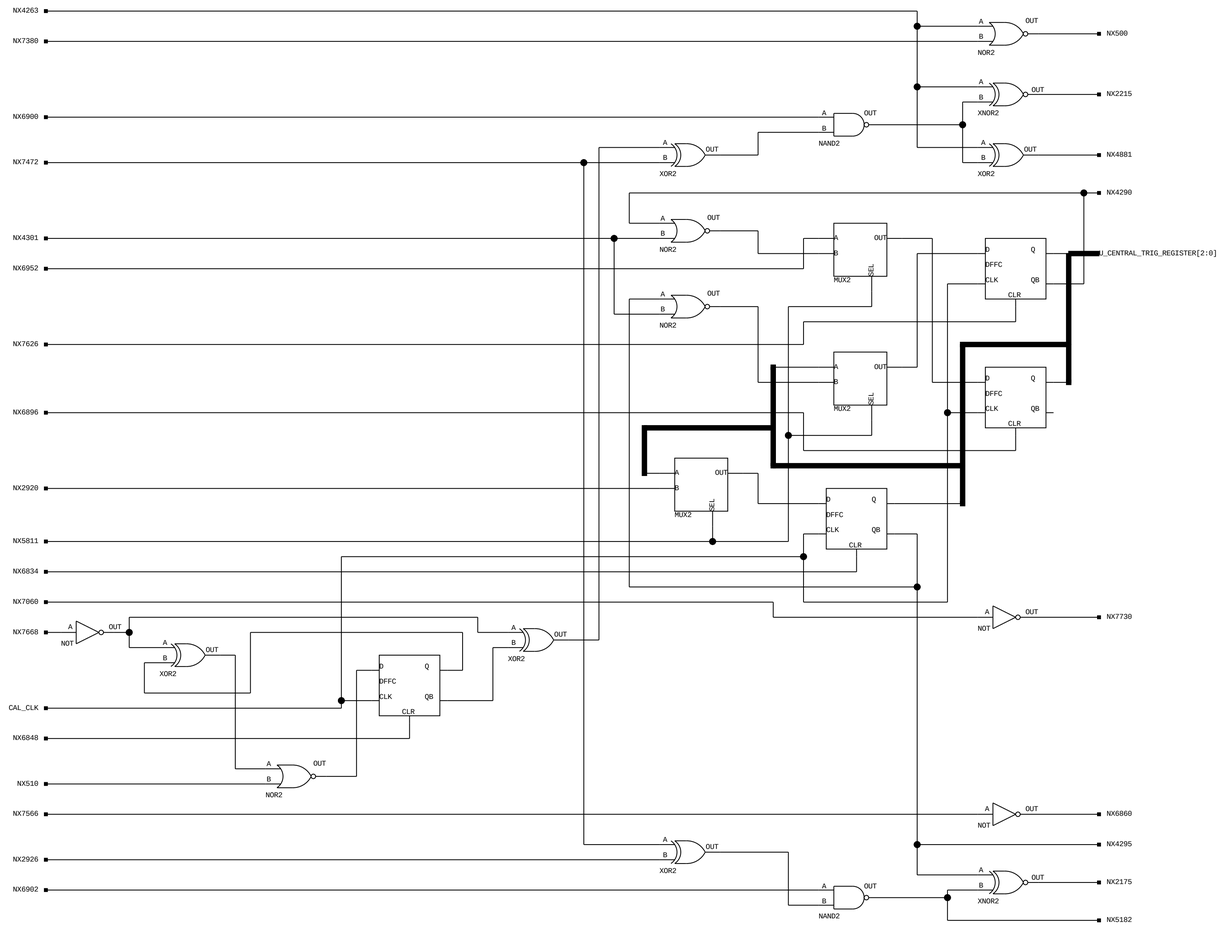


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

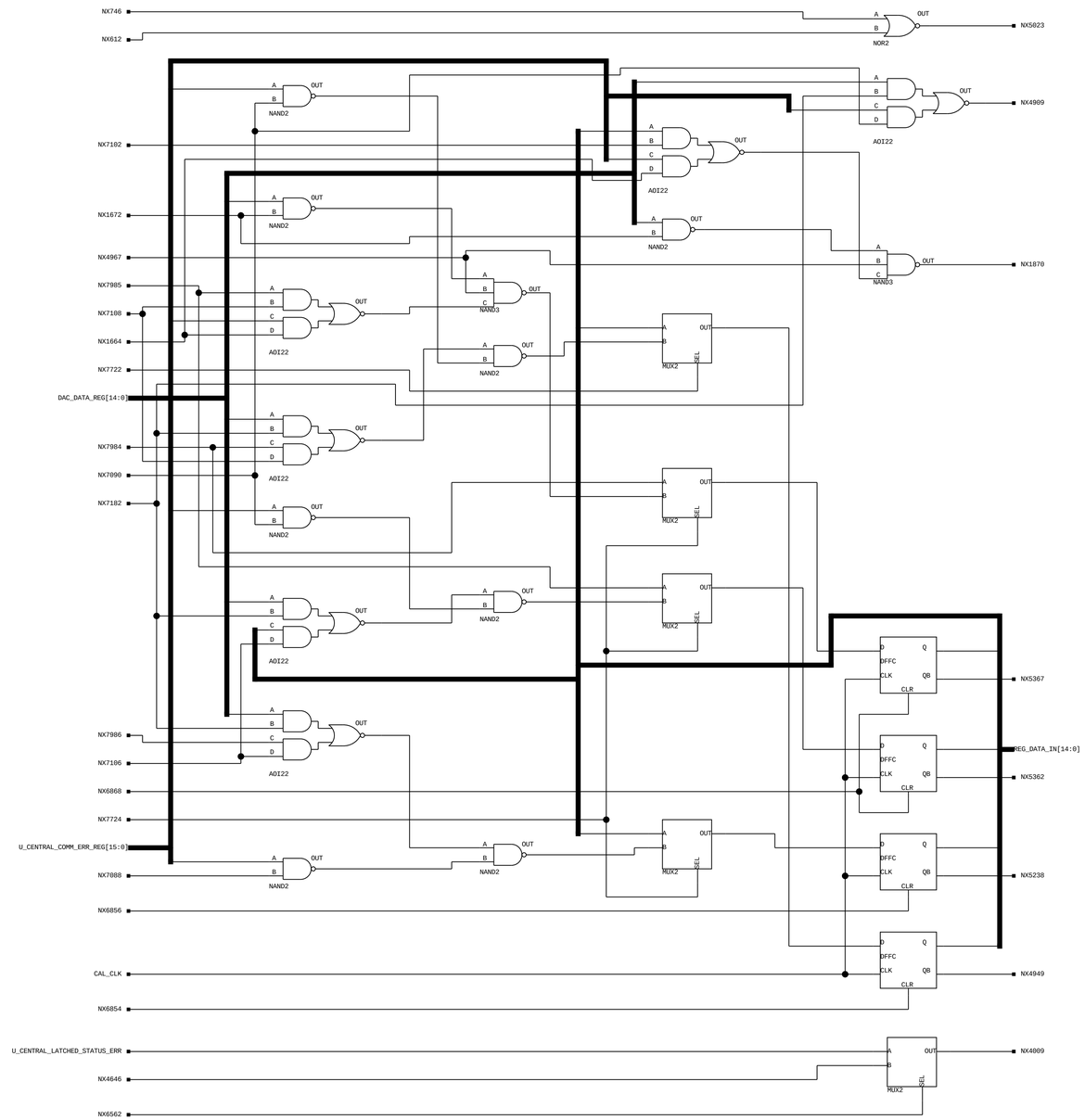


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44



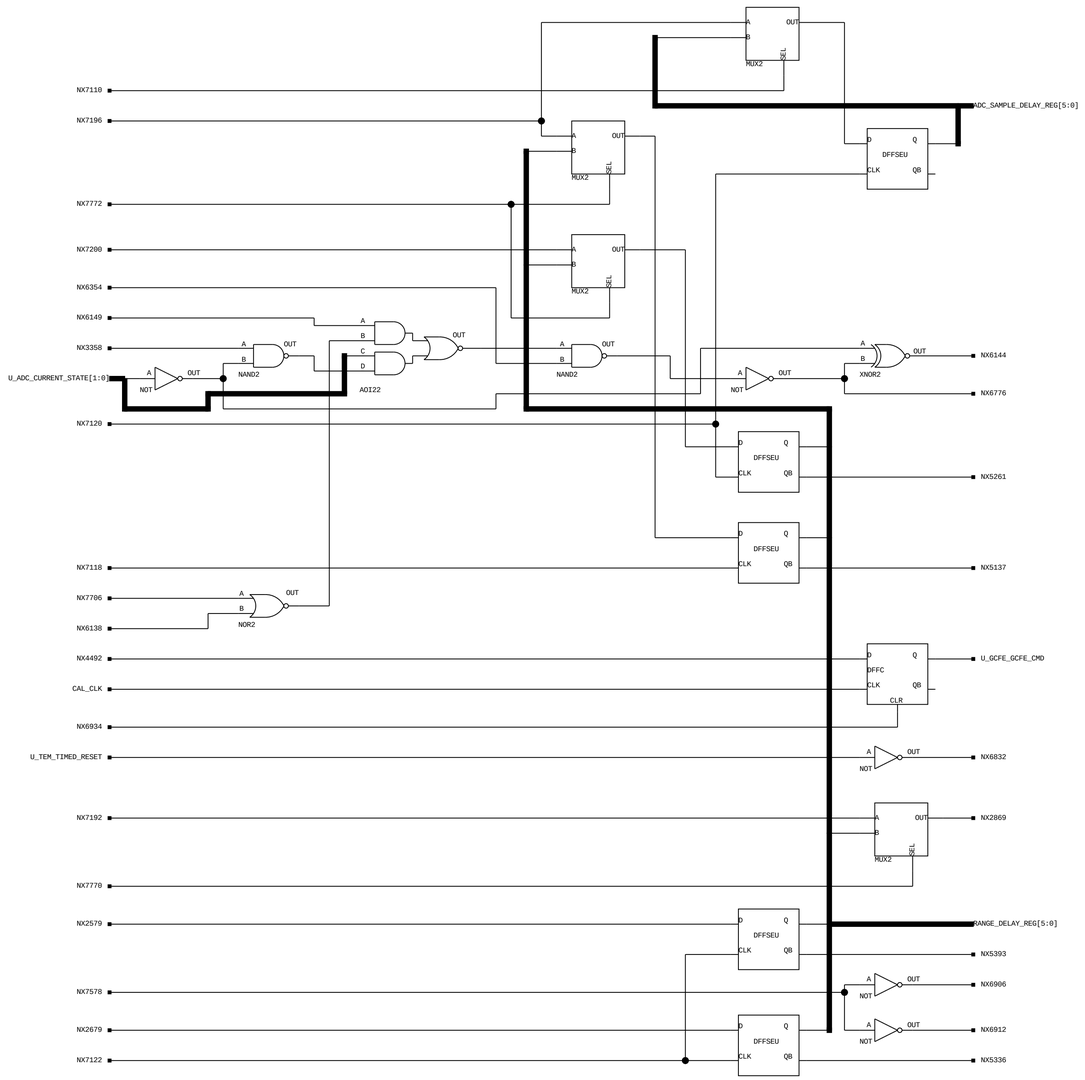
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 11 OF 51



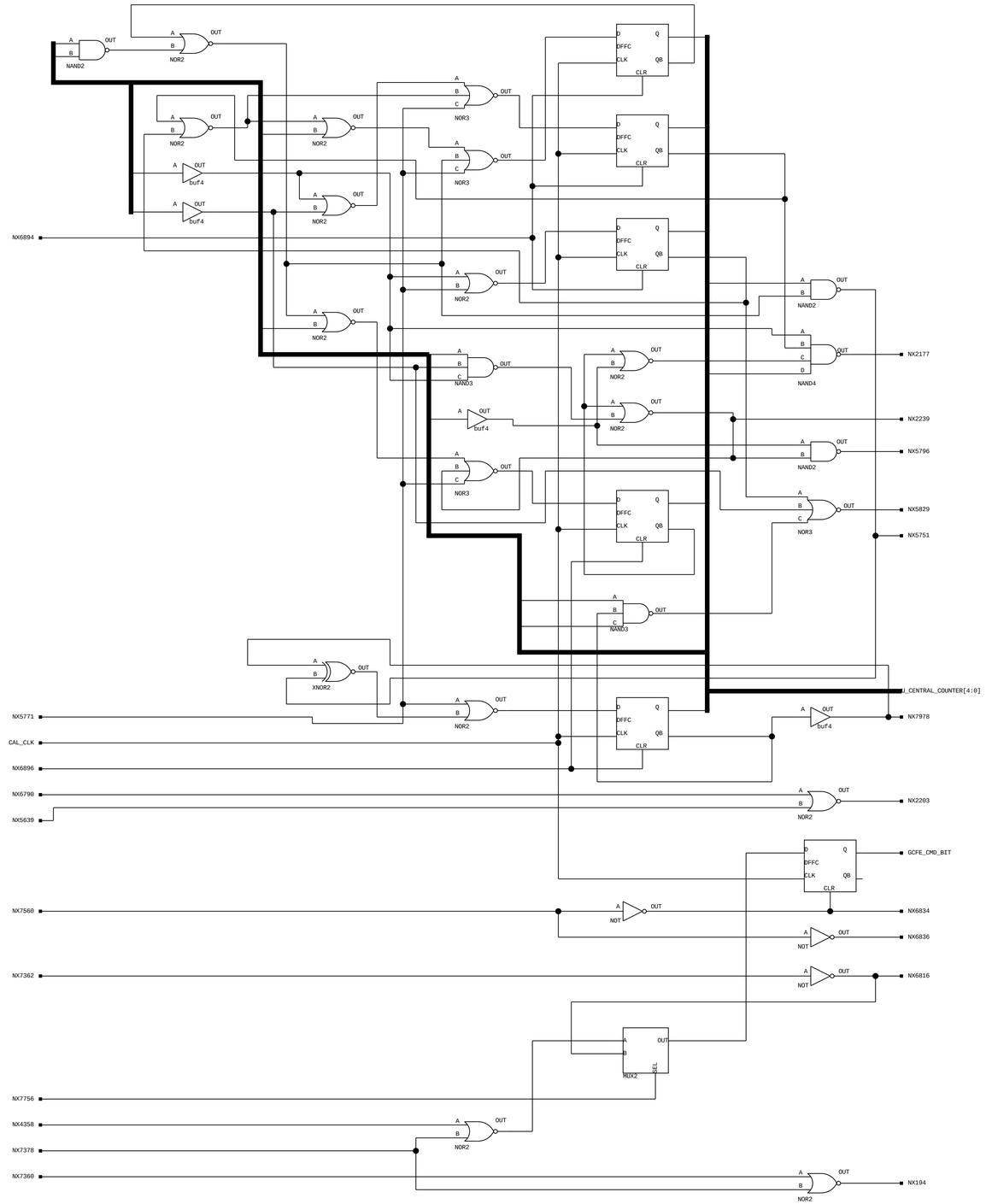
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 12 OF 51

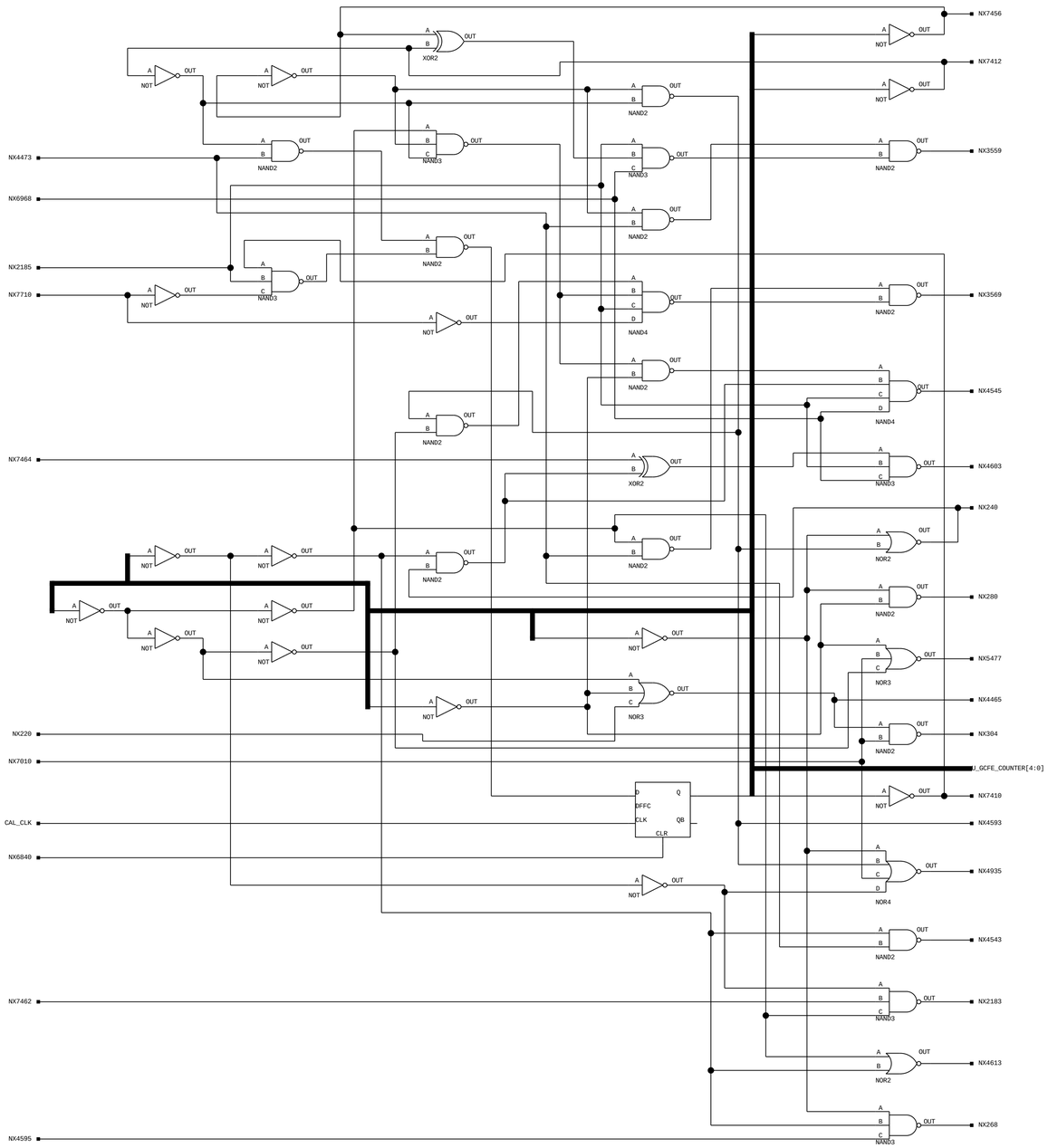


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44



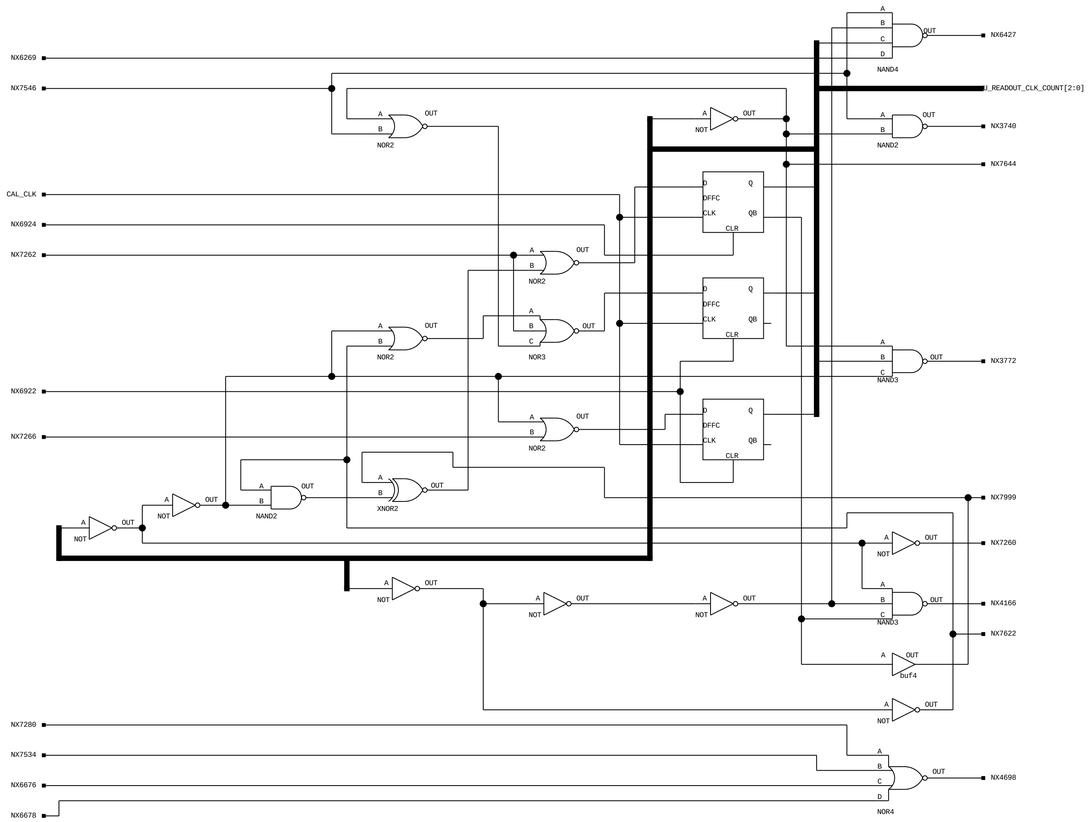
gcrc_core

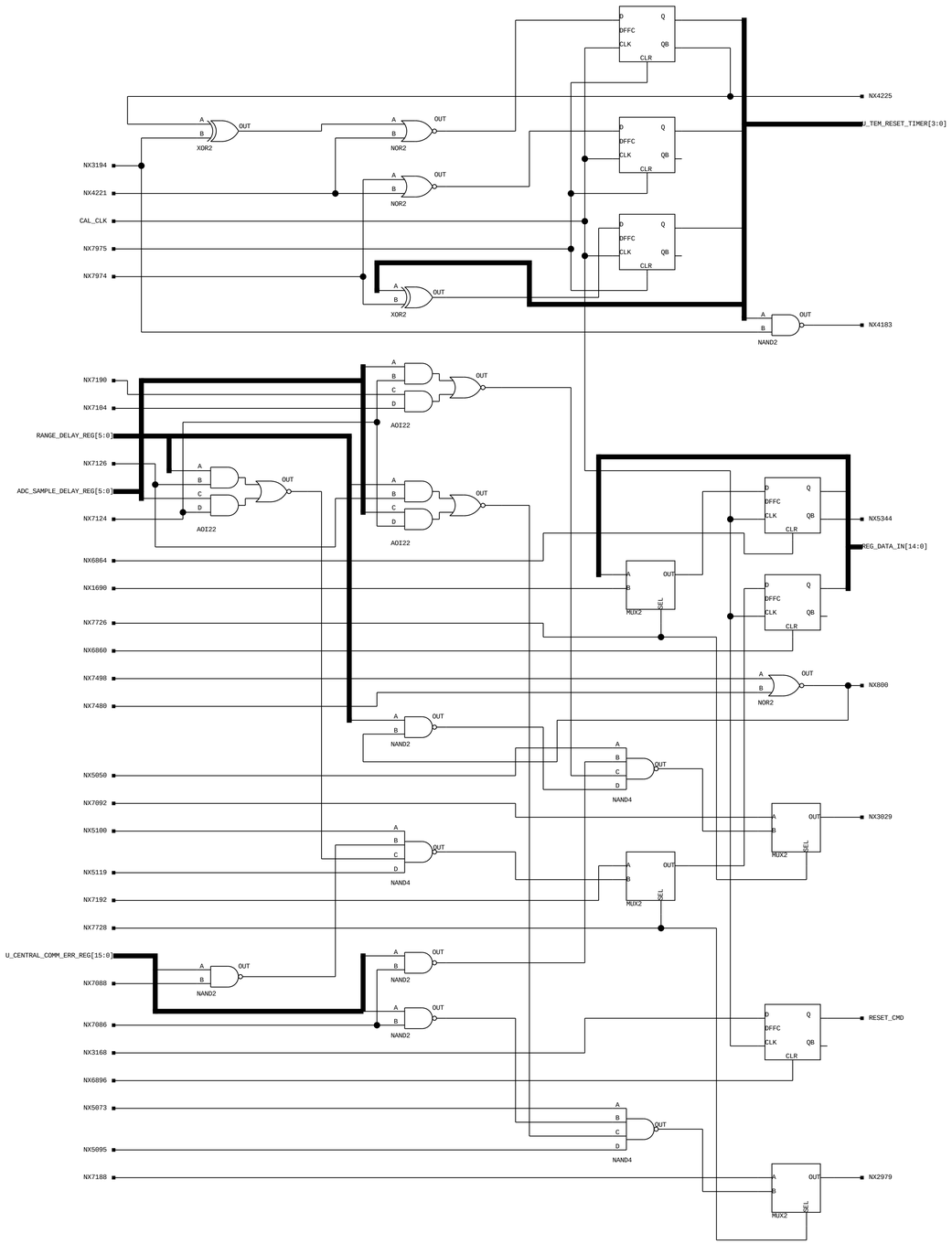
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 14 OF 51





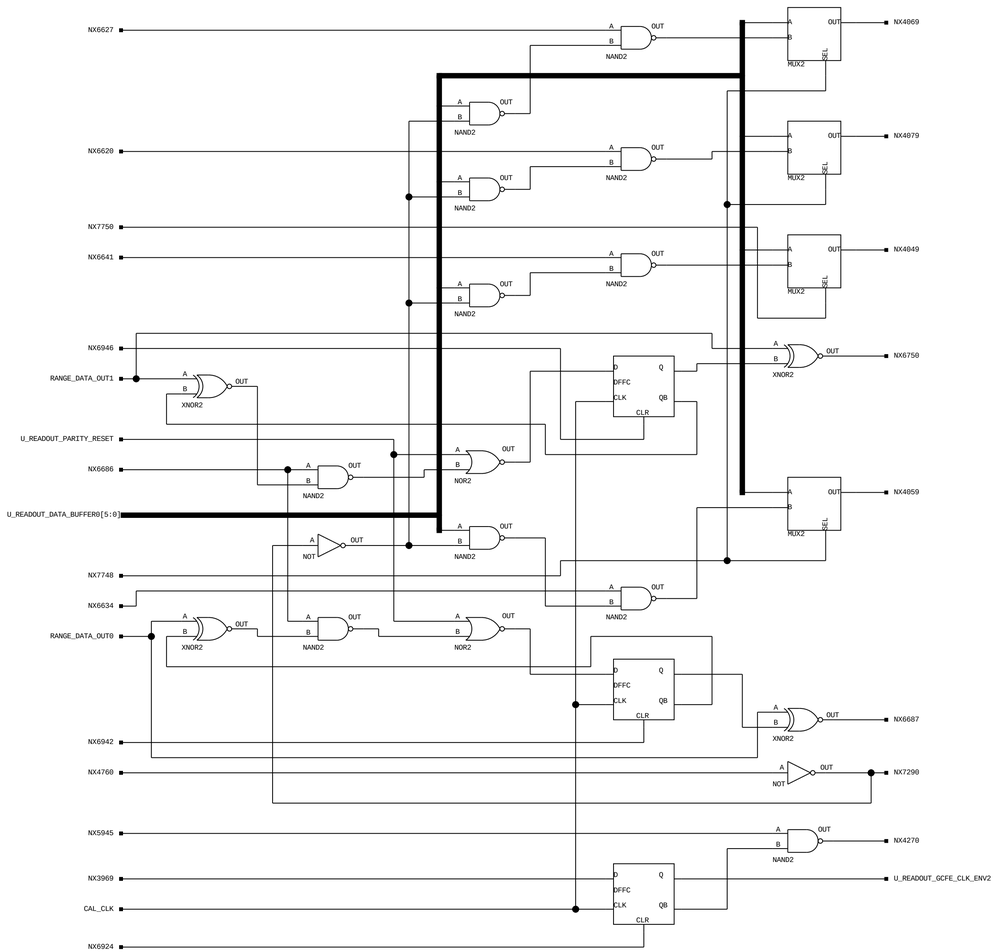
gcrc_core

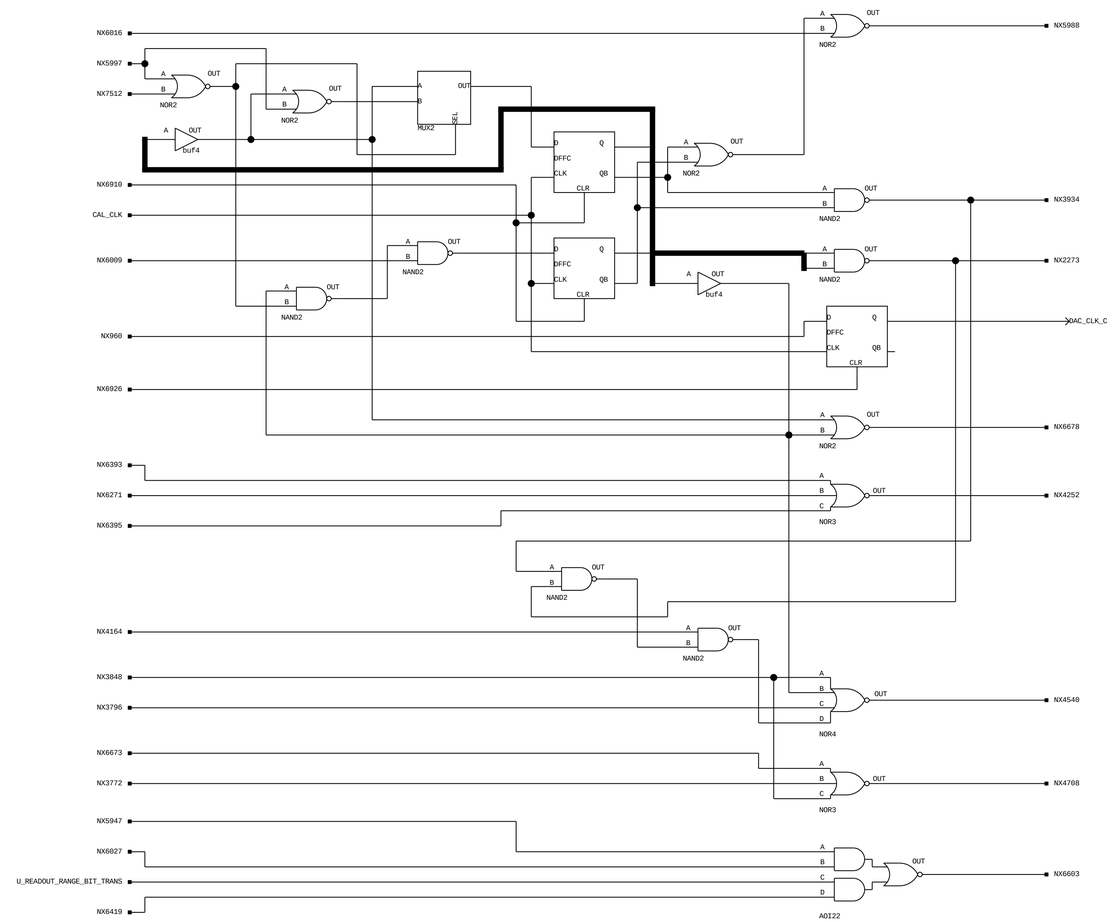
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 15 OF 51



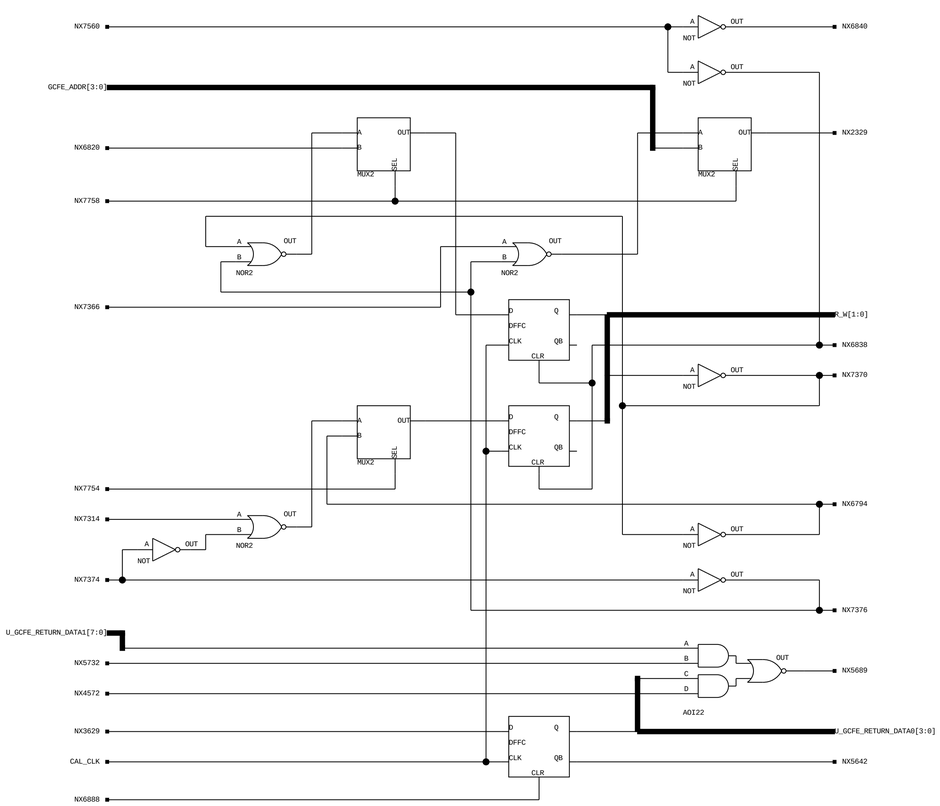


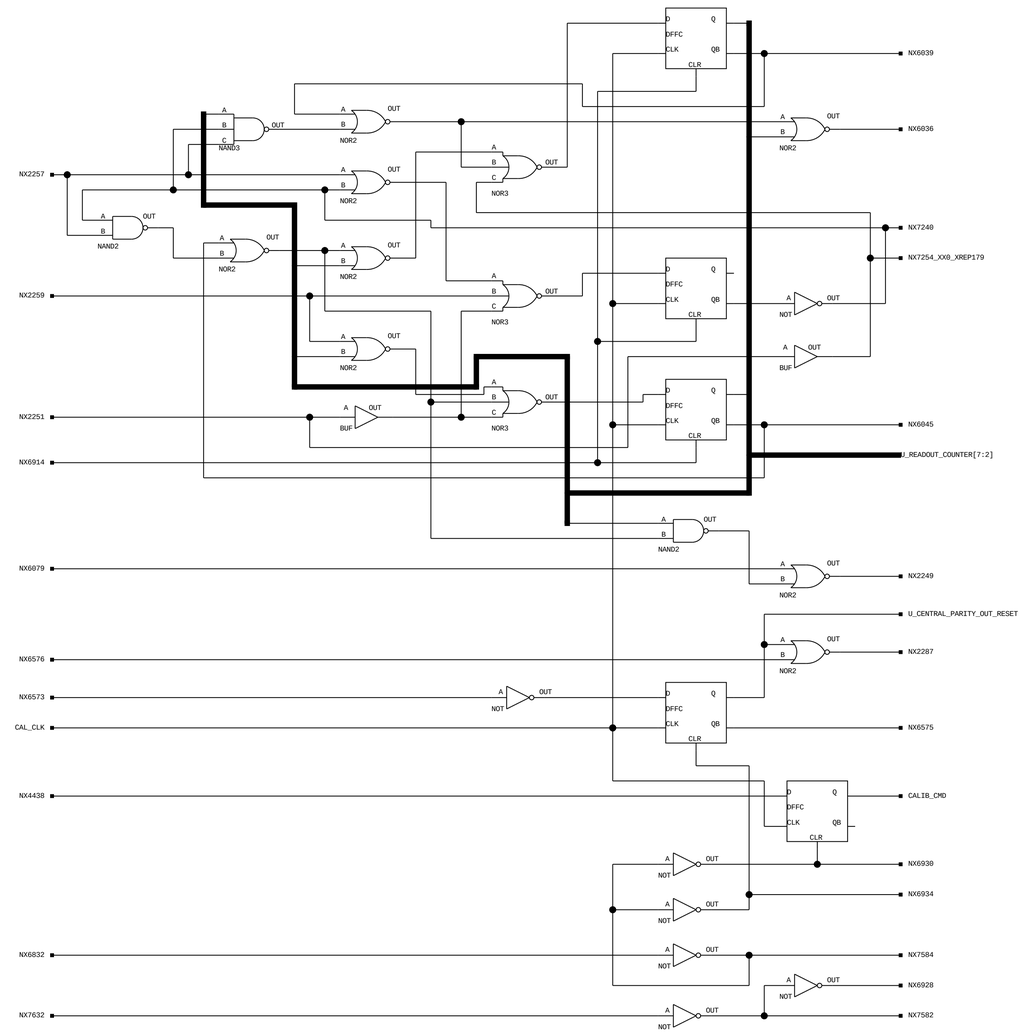
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44





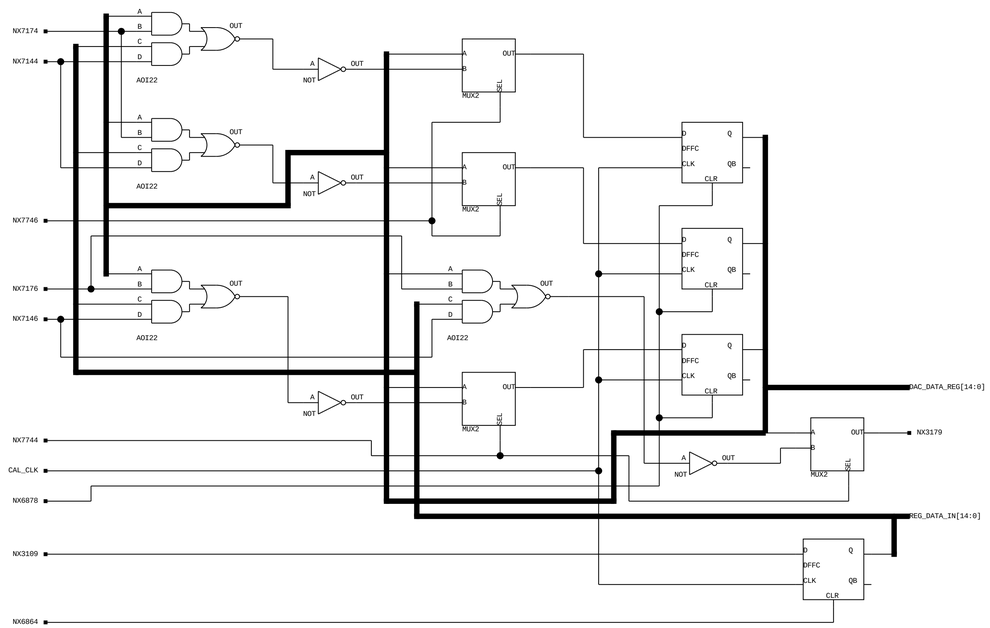
gcrc_core

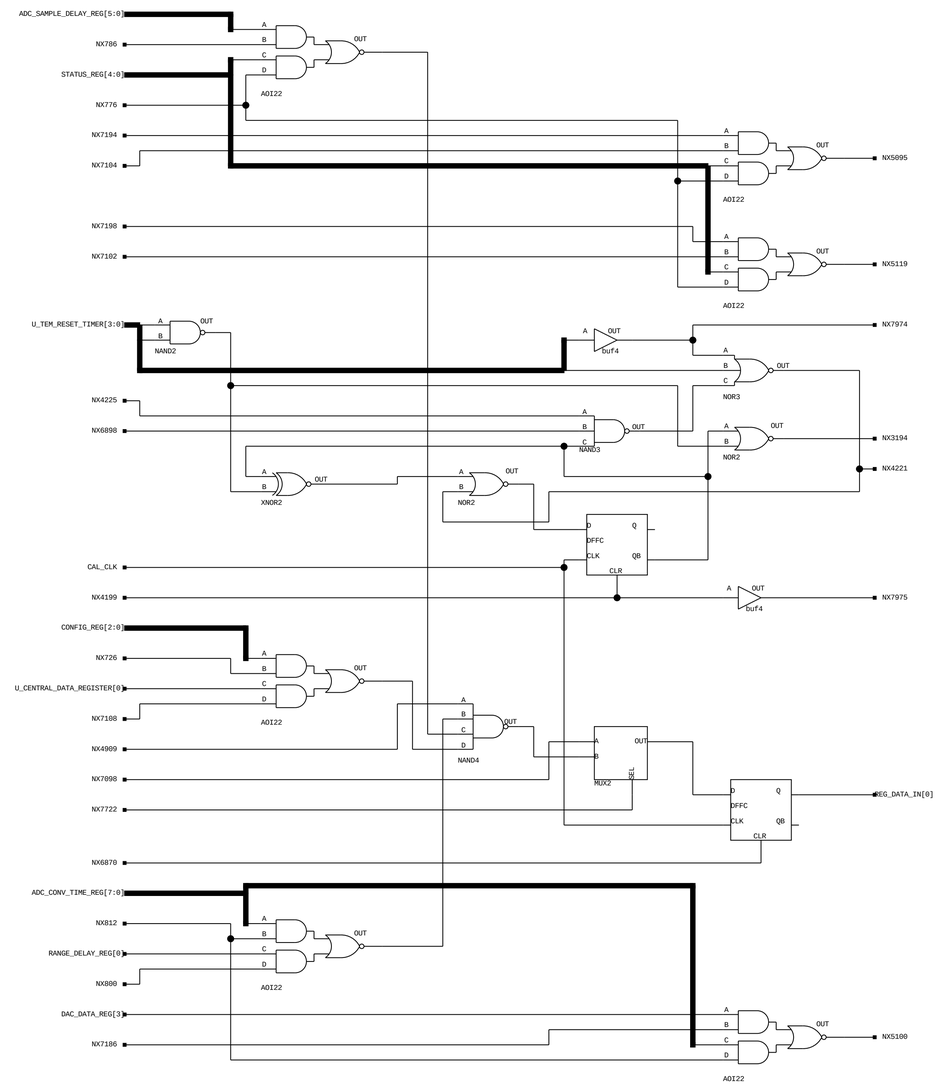
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 17 OF 51



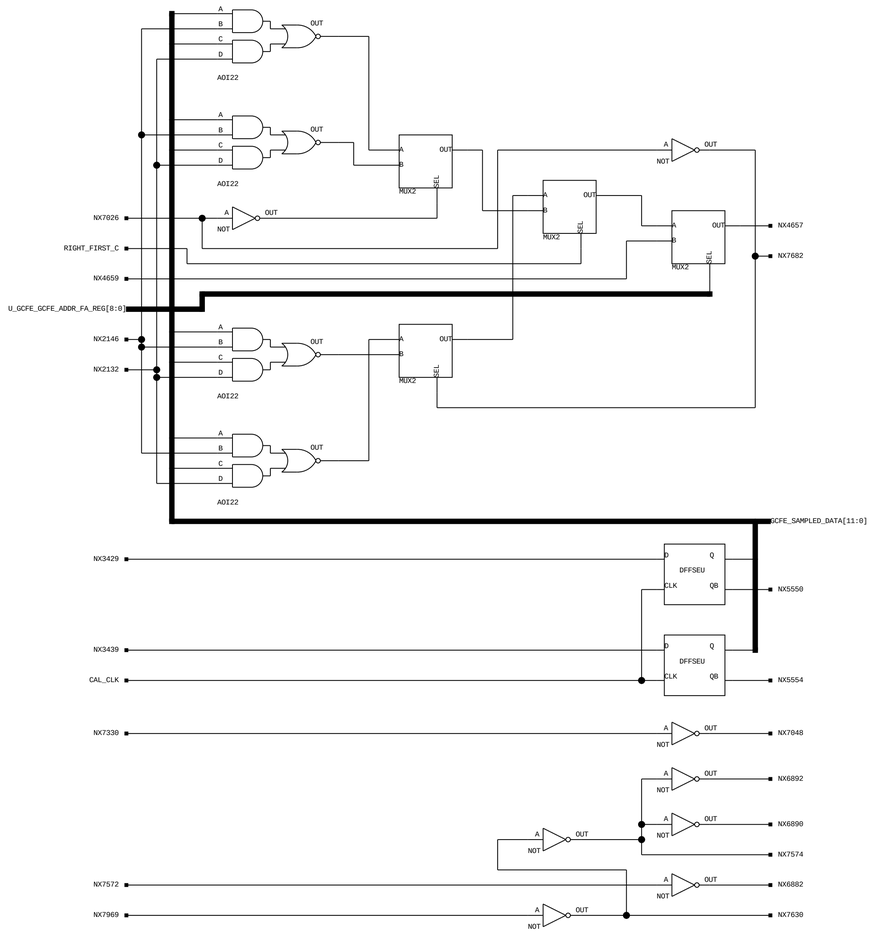


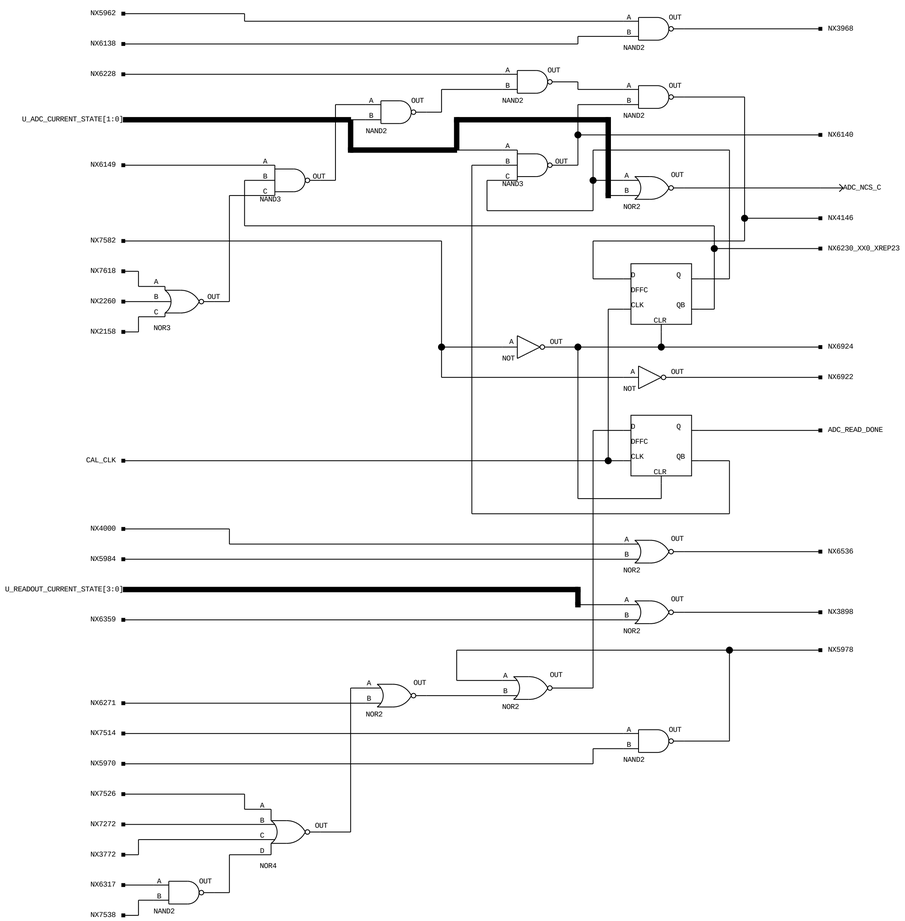
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44	SHEET 18 OF 51





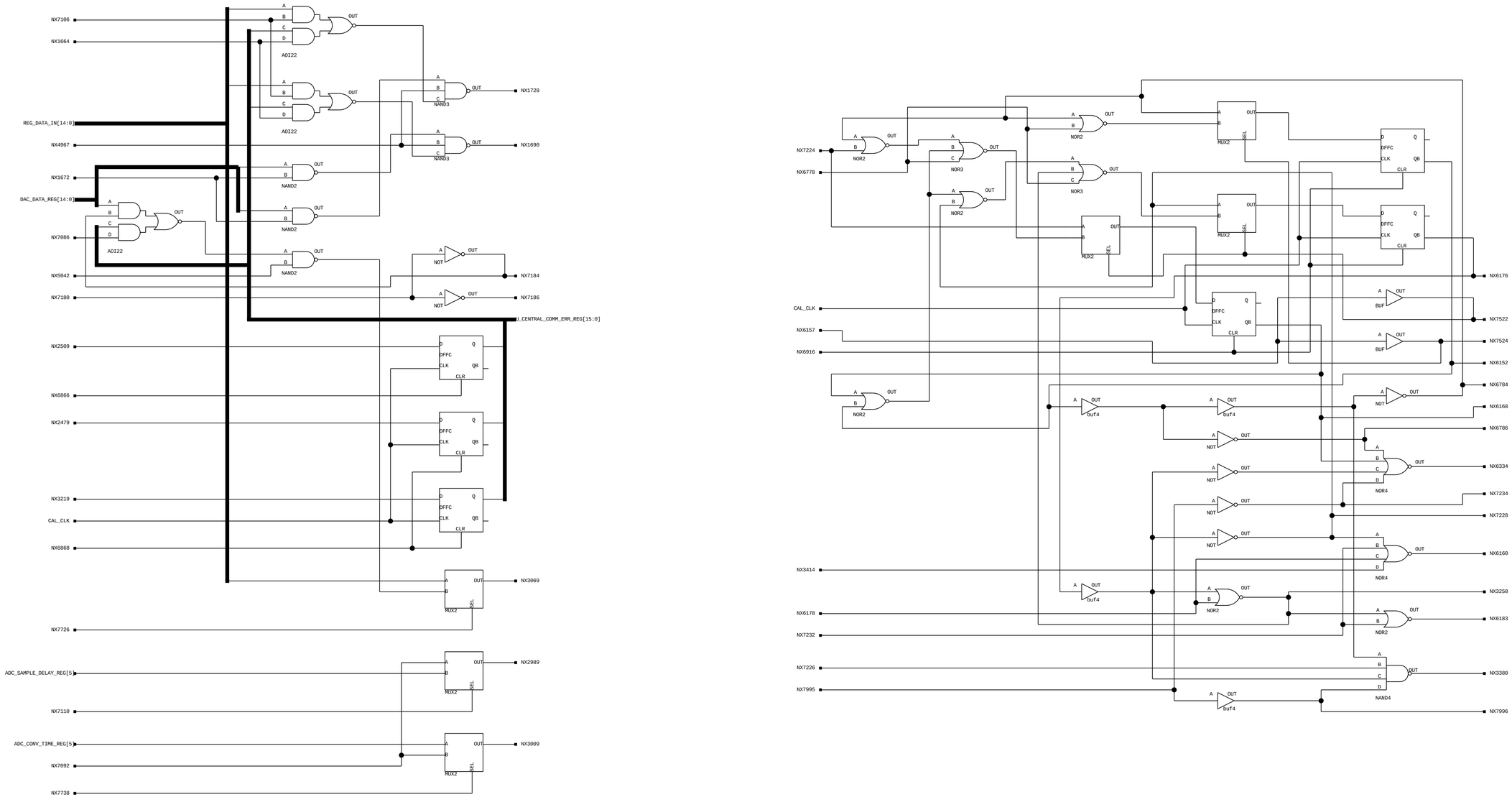
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 19 OF 51



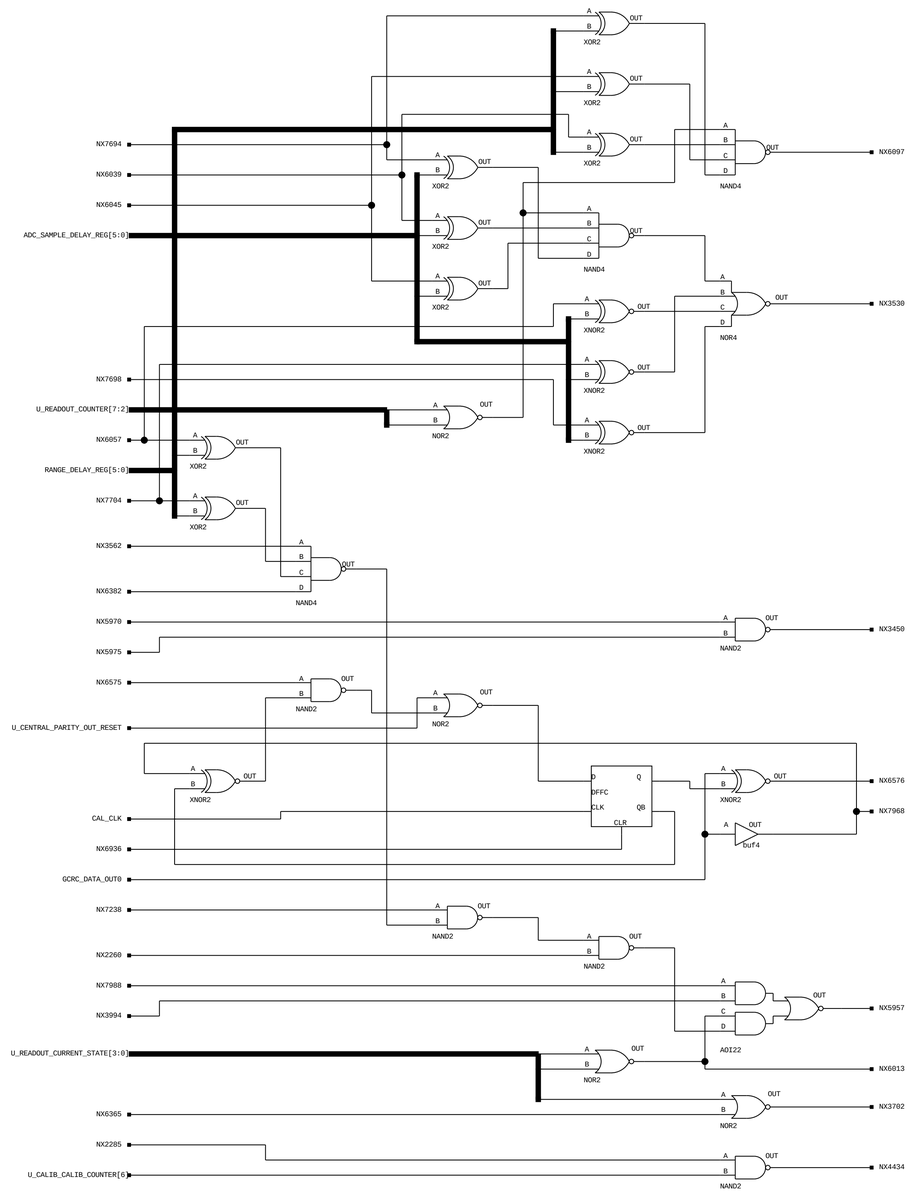
gcrc_core

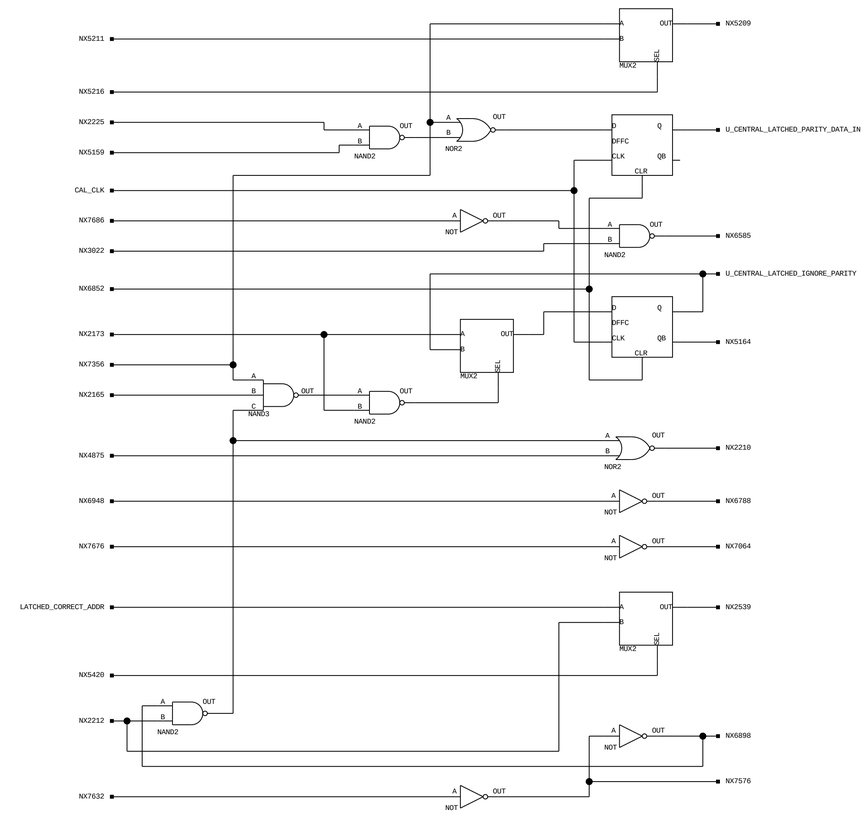
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 20 OF 51





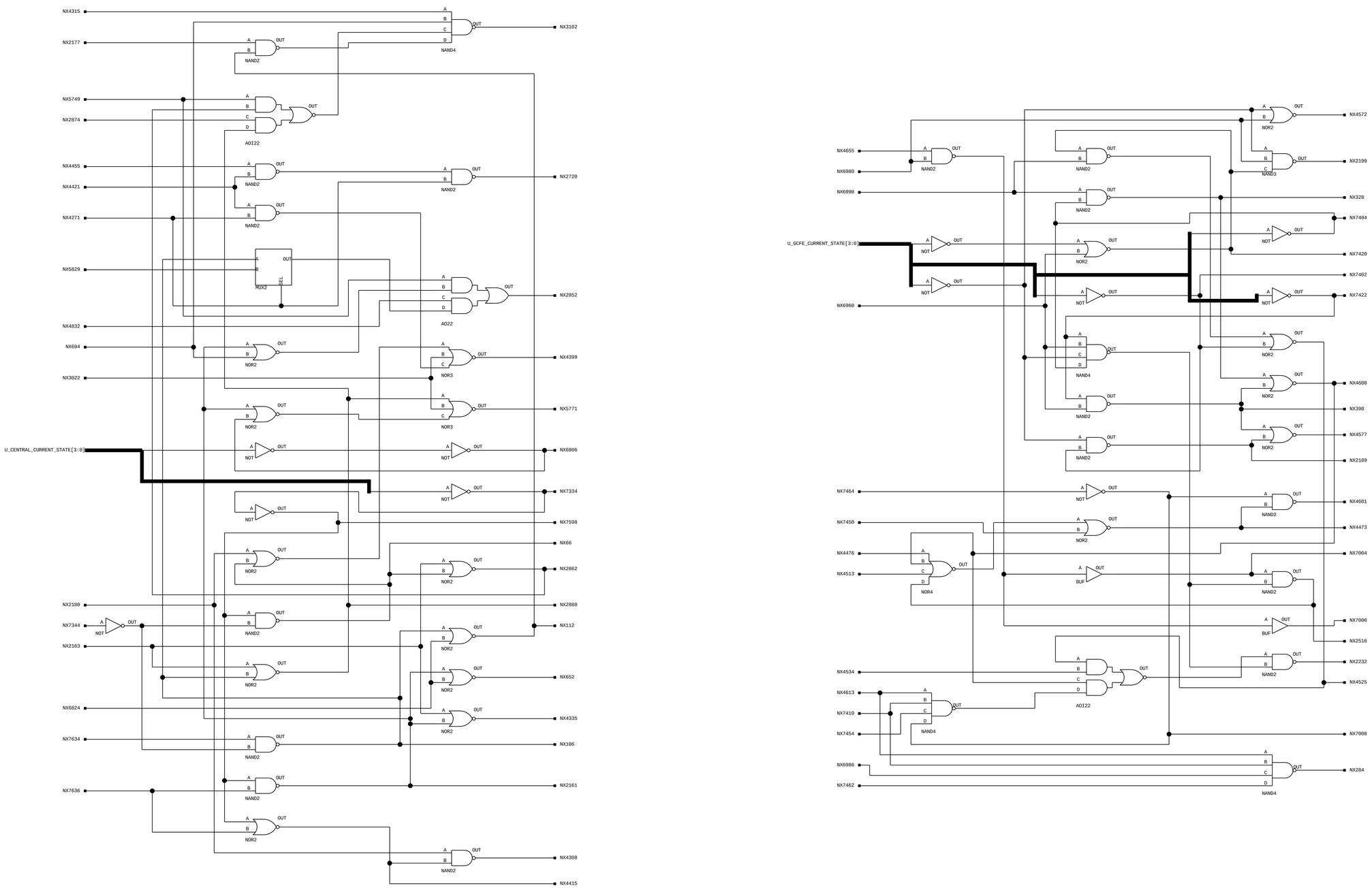
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 21 OF 51



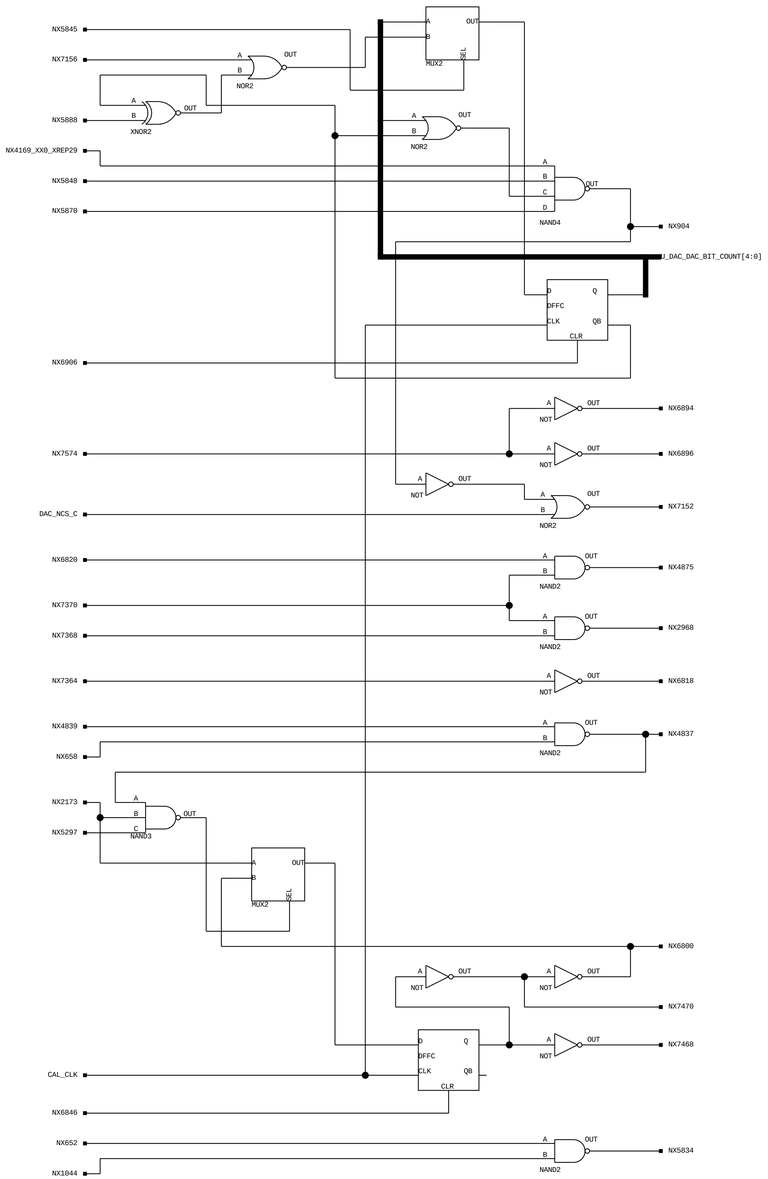
gcrc_core

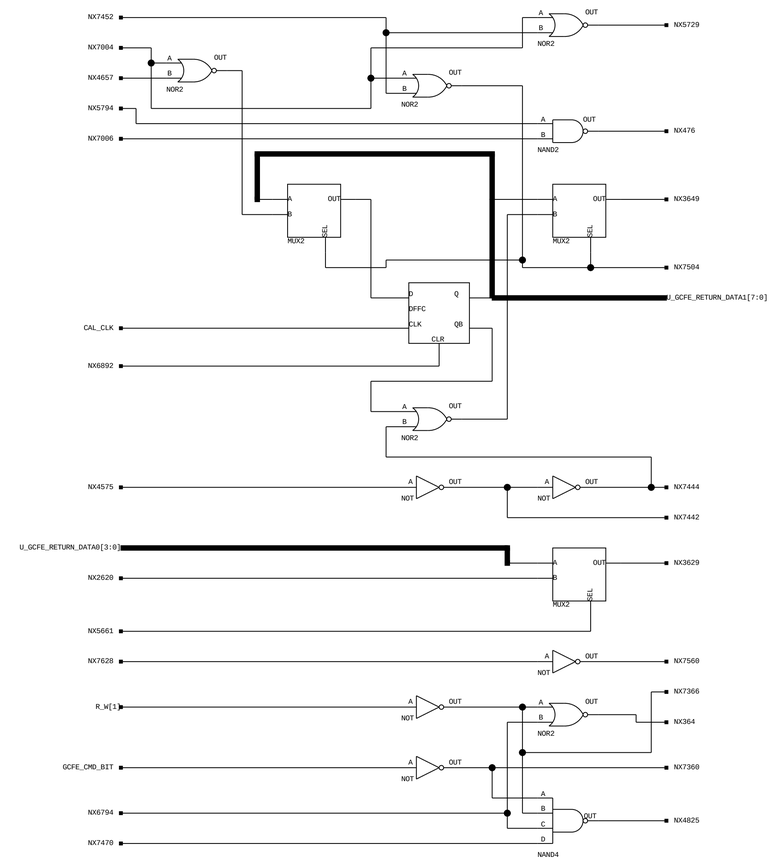
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 22 OF 51





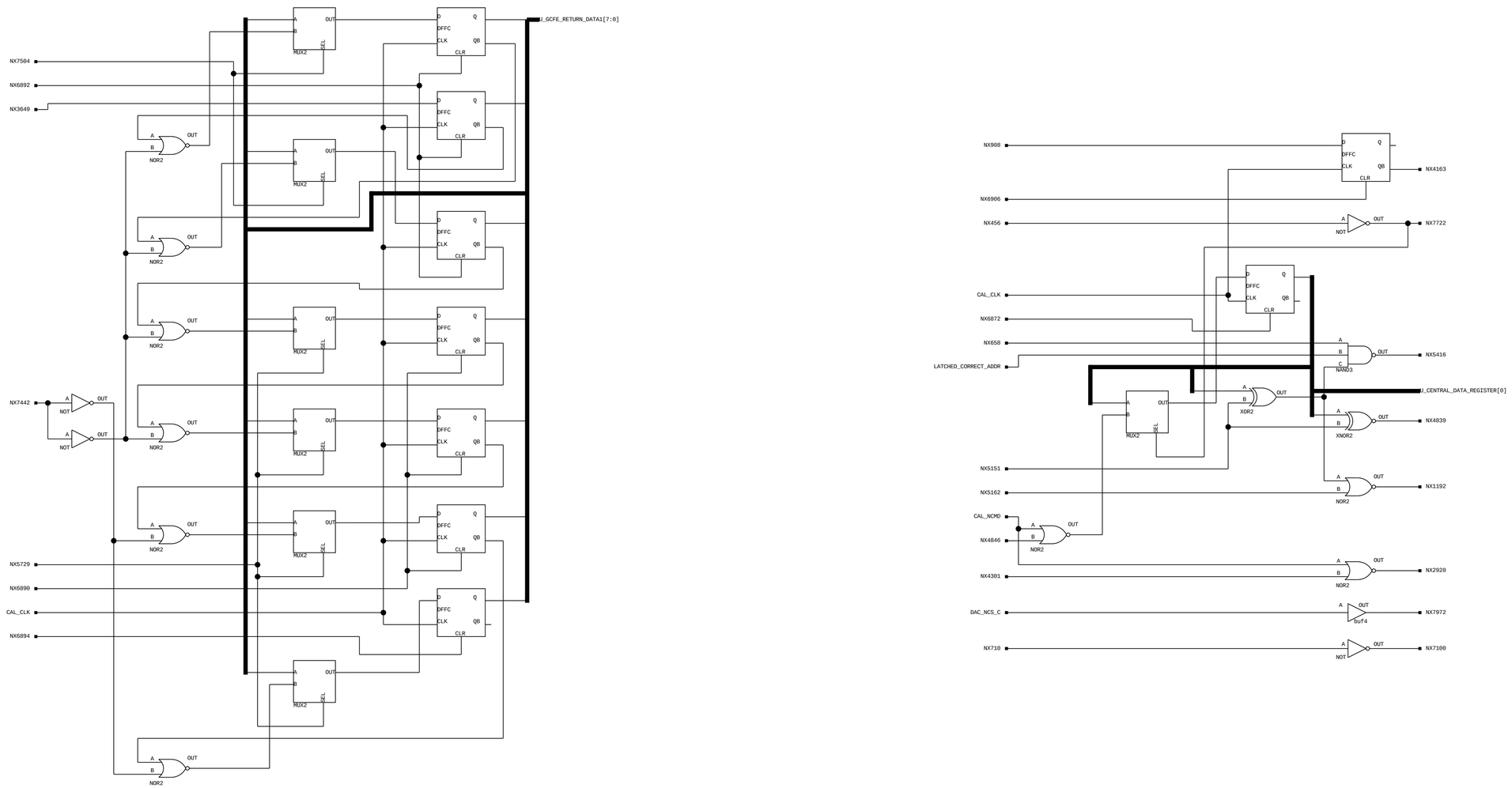
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 23 OF 51



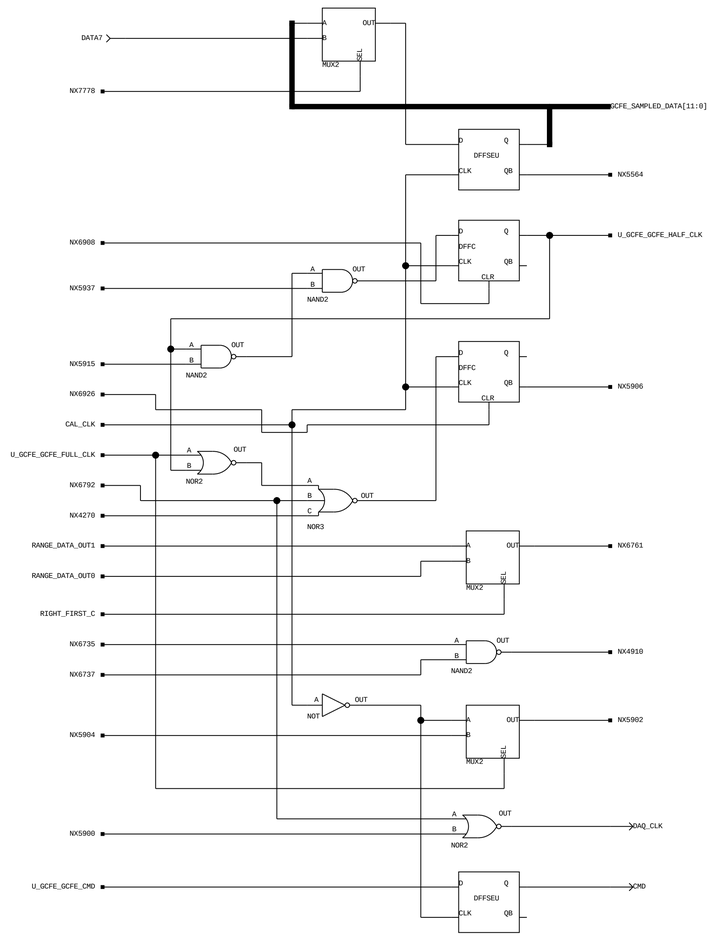
gcrc_core

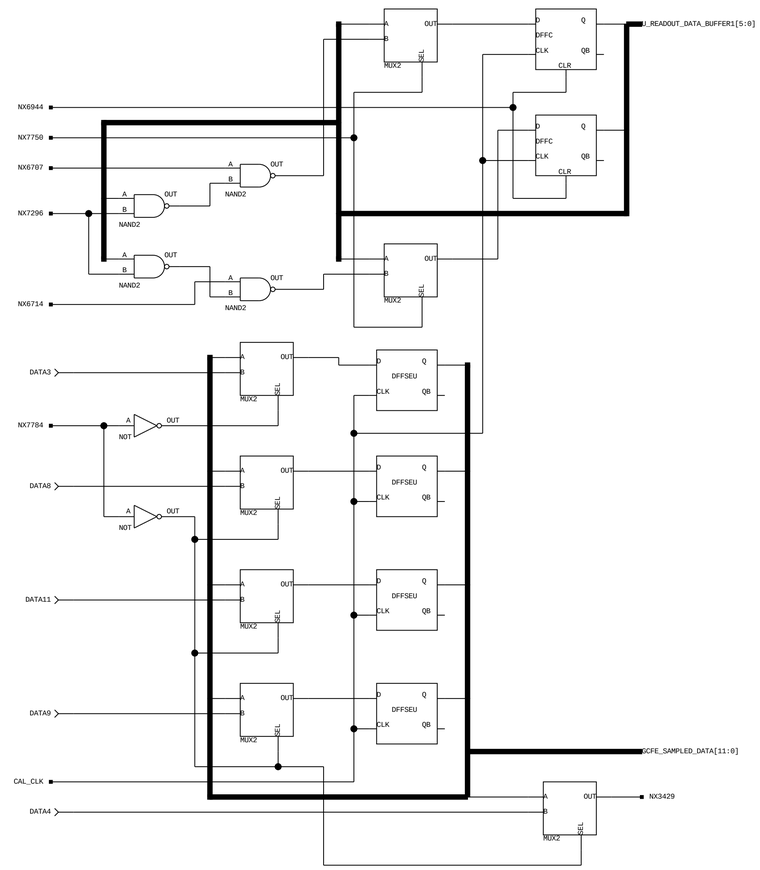
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 24 OF 51





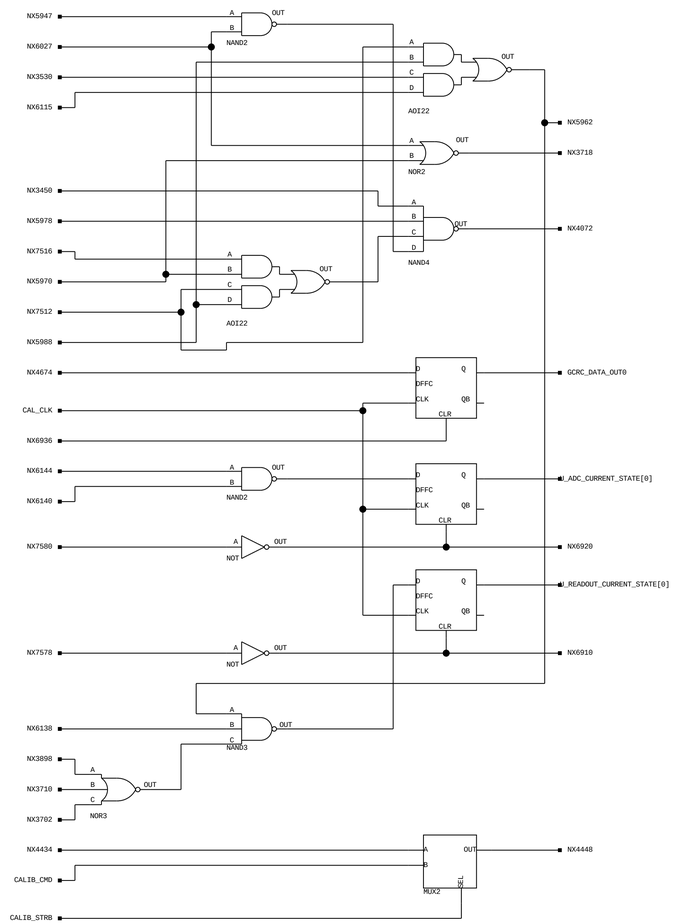
gcrc_core

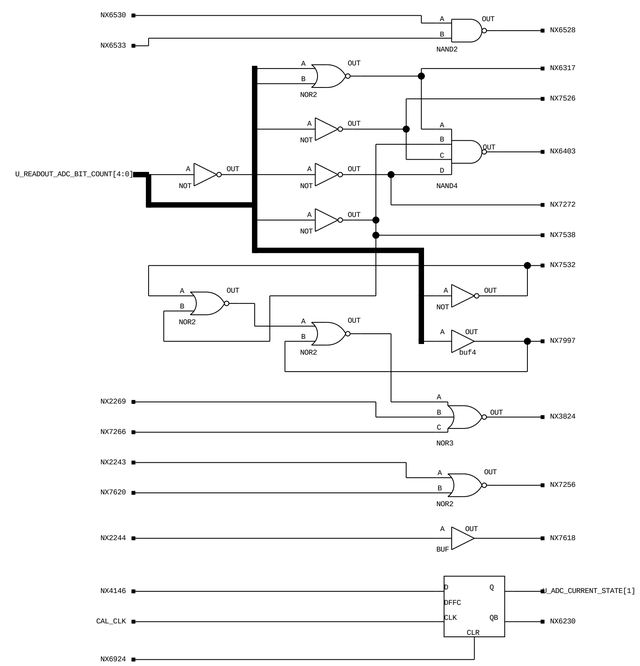
WIR:gcrc_chip

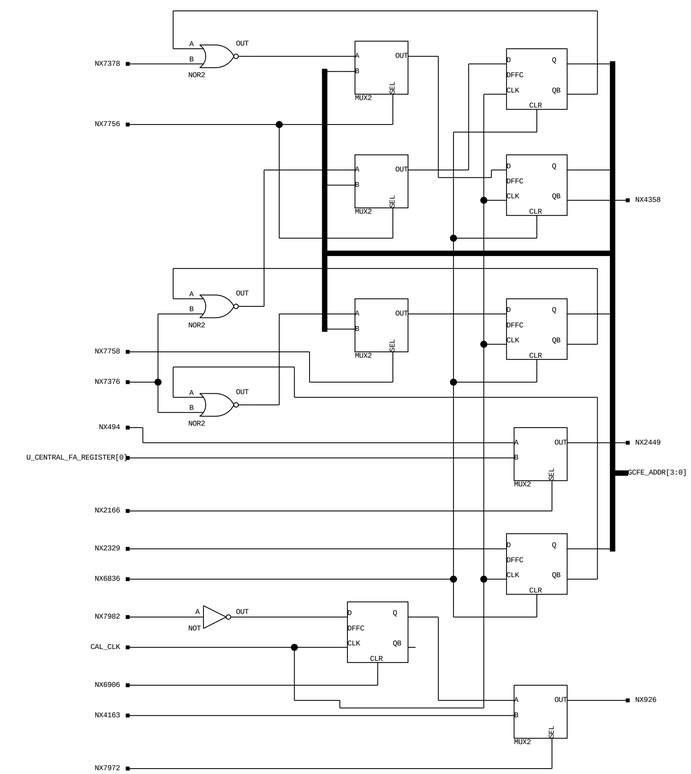
SCH:gcrc_core

20 Dec 02 17:44

SHEET 25 OF 51







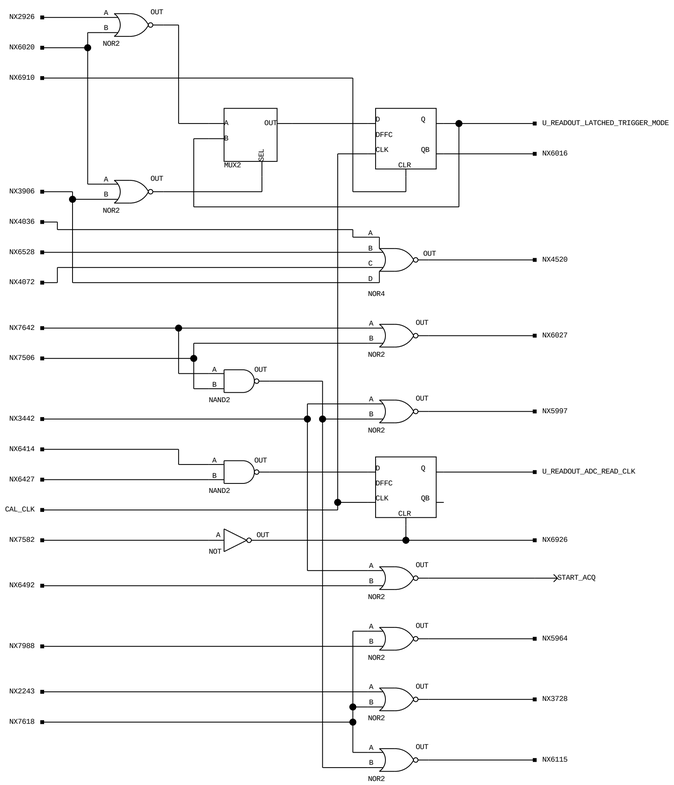
gcrc_core

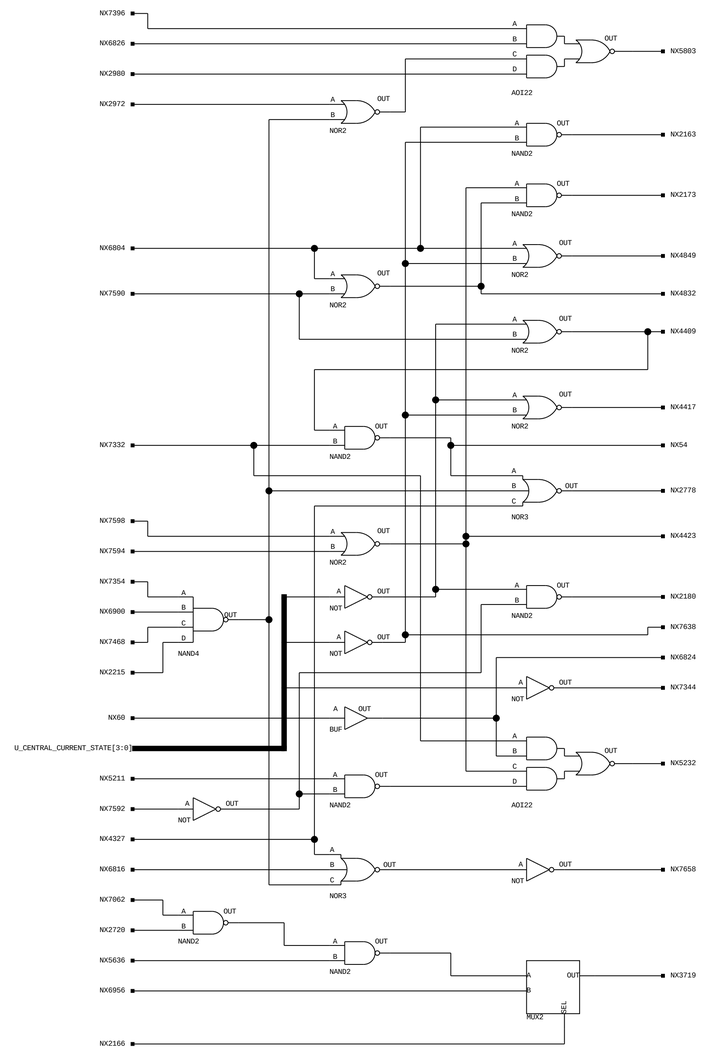
WIR:gcrc_chip

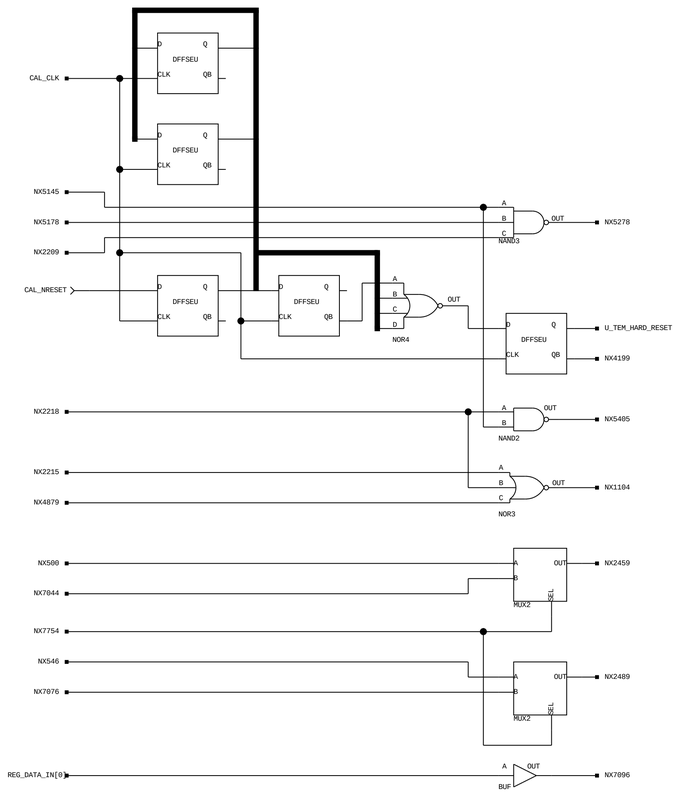
SCH:gcrc_core

20 Dec 02 17:44

SHEET 26 OF 51







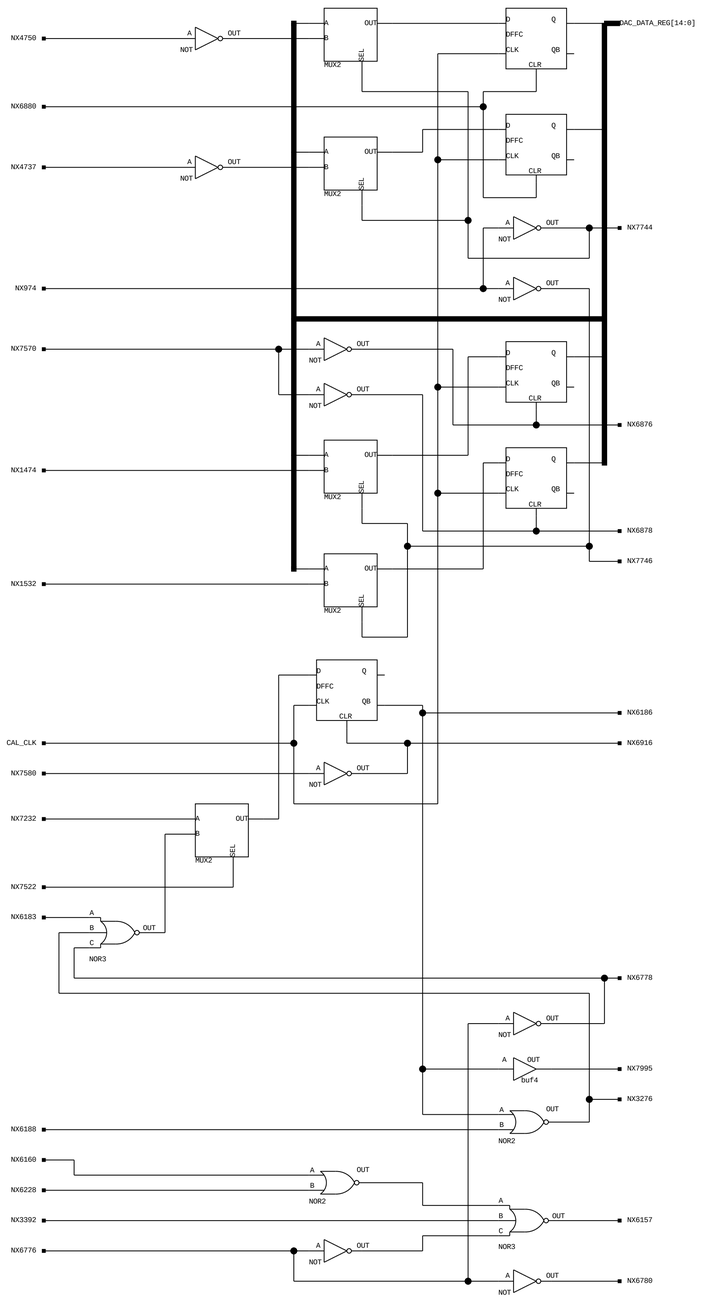
gcrc_core

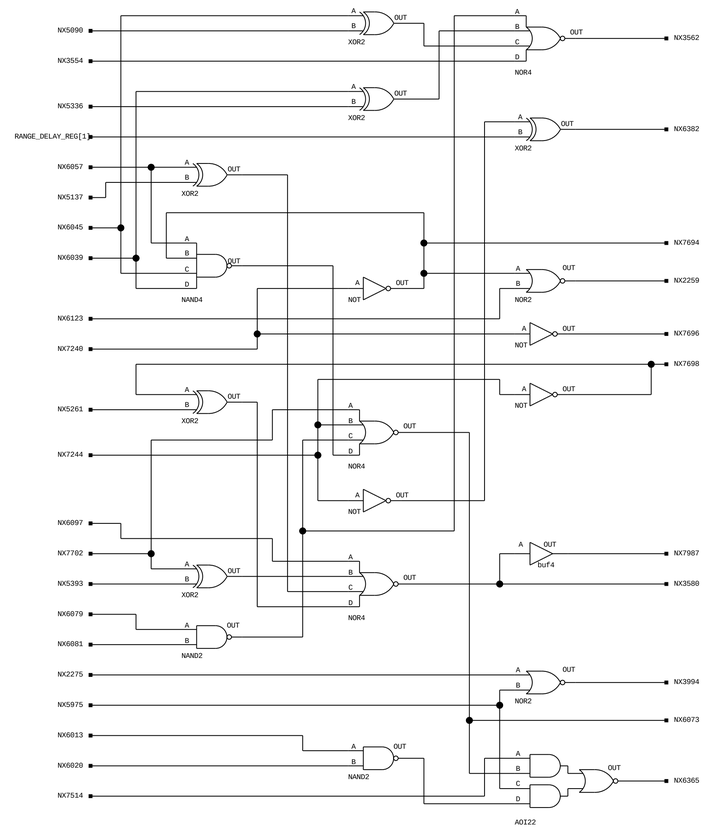
WIR:gcrc_chip

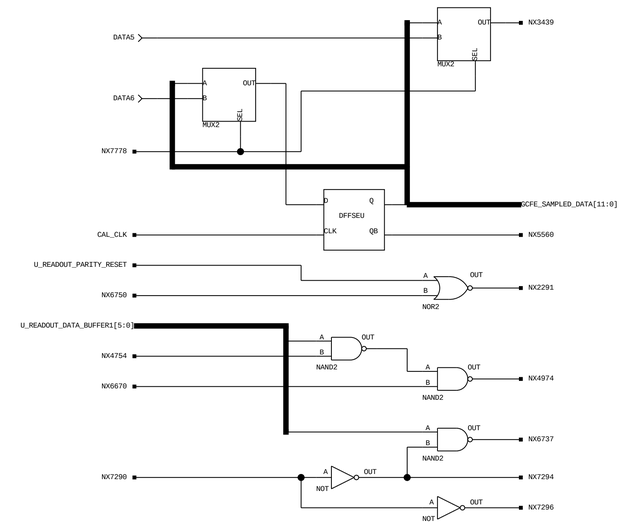
SCH:gcrc_core

20 Dec 02 17:44

SHEET 27 OF 51







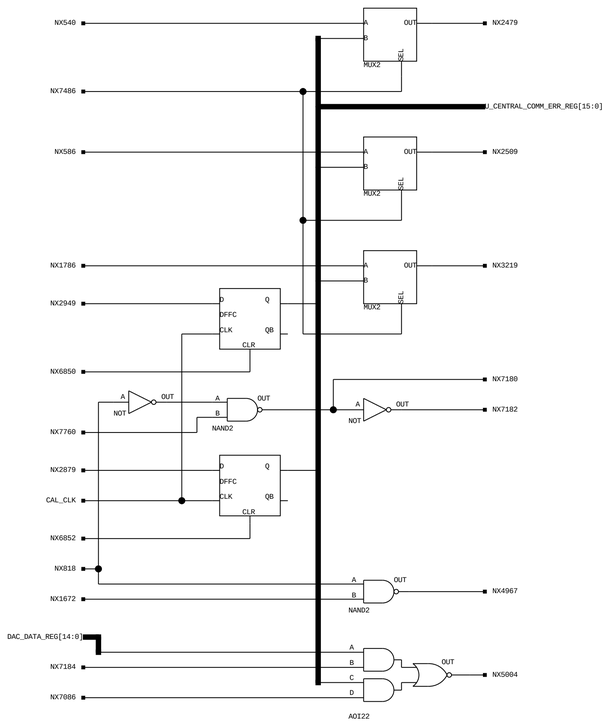
gcrc_core

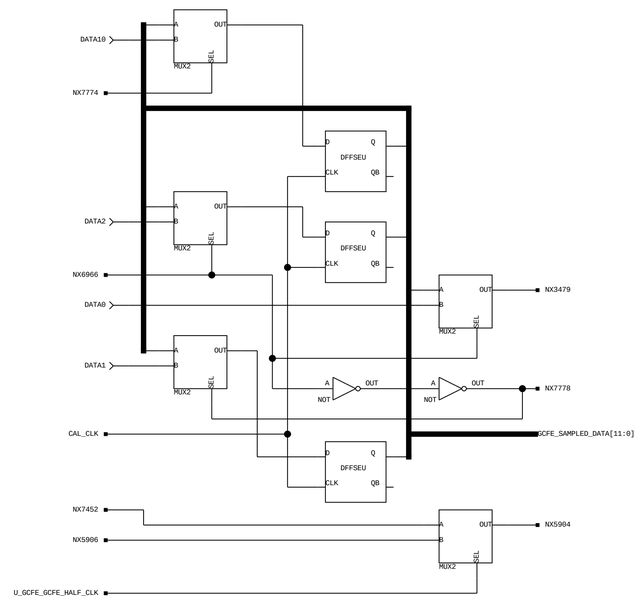
WIR:gcrc_chip

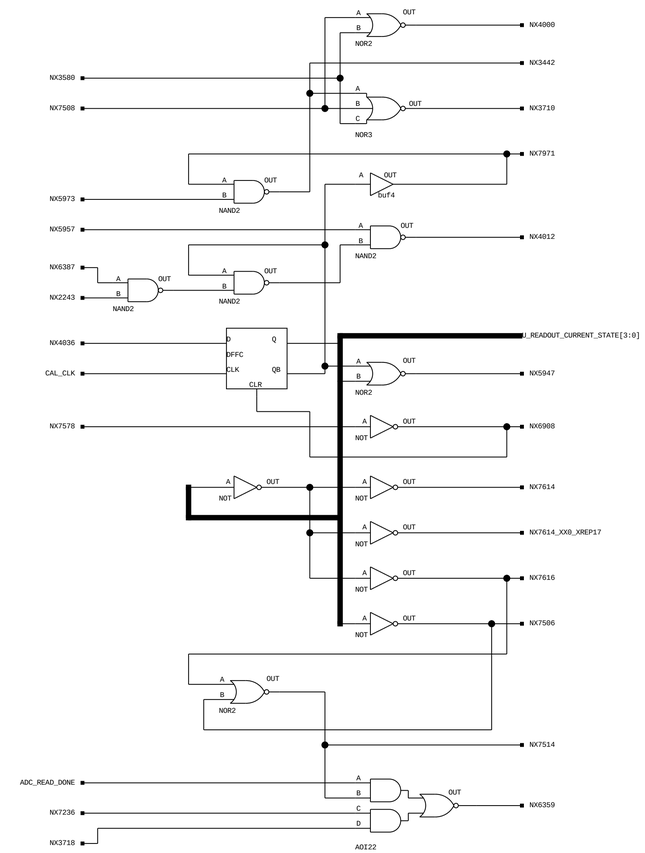
SCH:gcrc_core

20 Dec 02 17:44

SHEET 28 OF 51







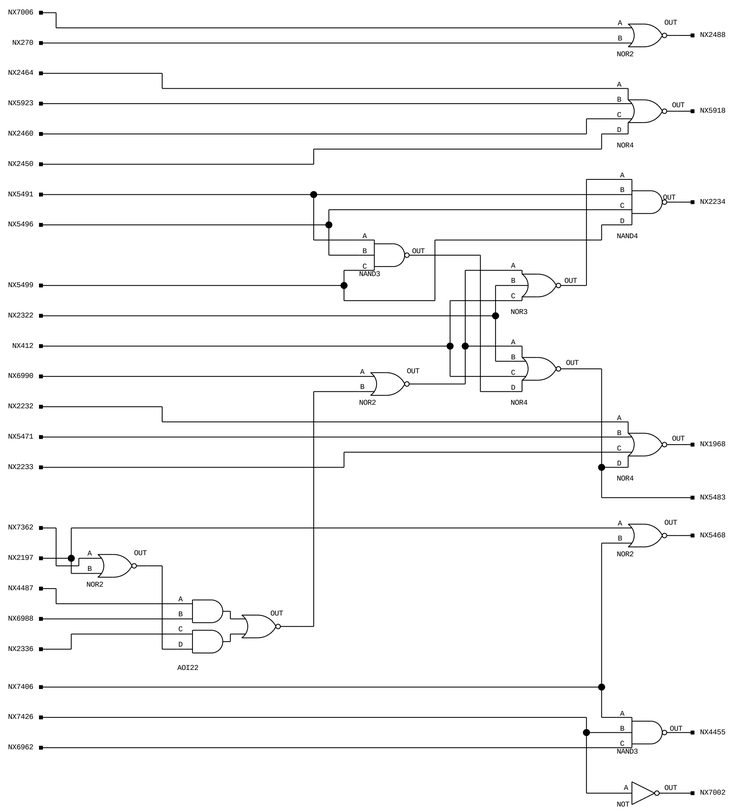
gcrc_core

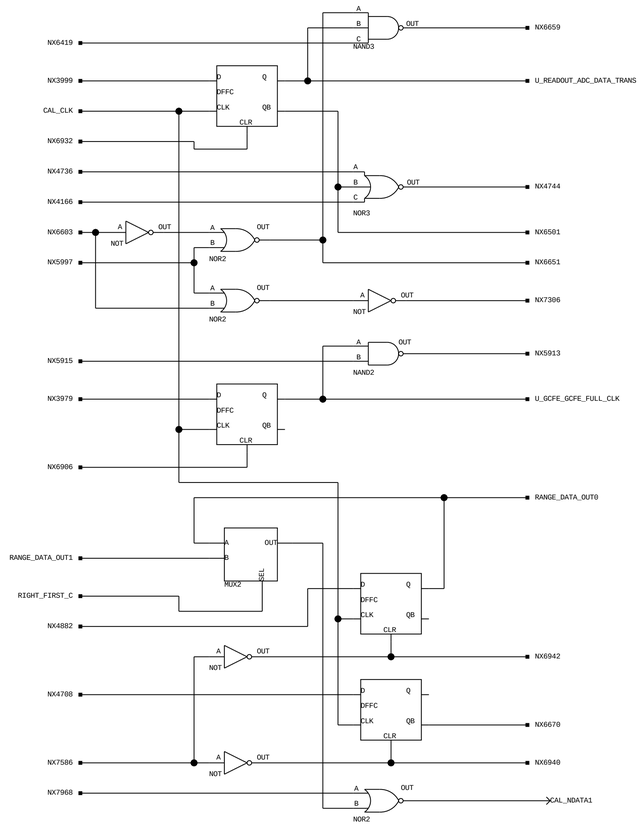
WIR:gcrc_chip

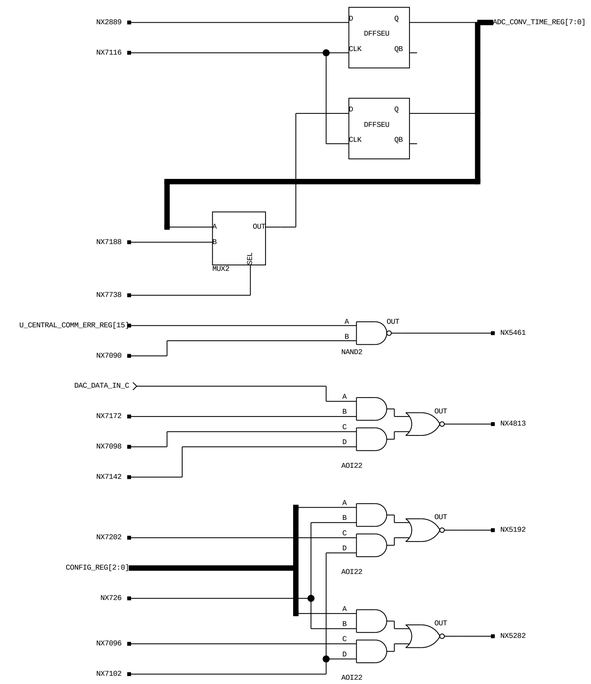
SCH:gcrc_core

20 Dec 02 17:44

SHEET 29 OF 51







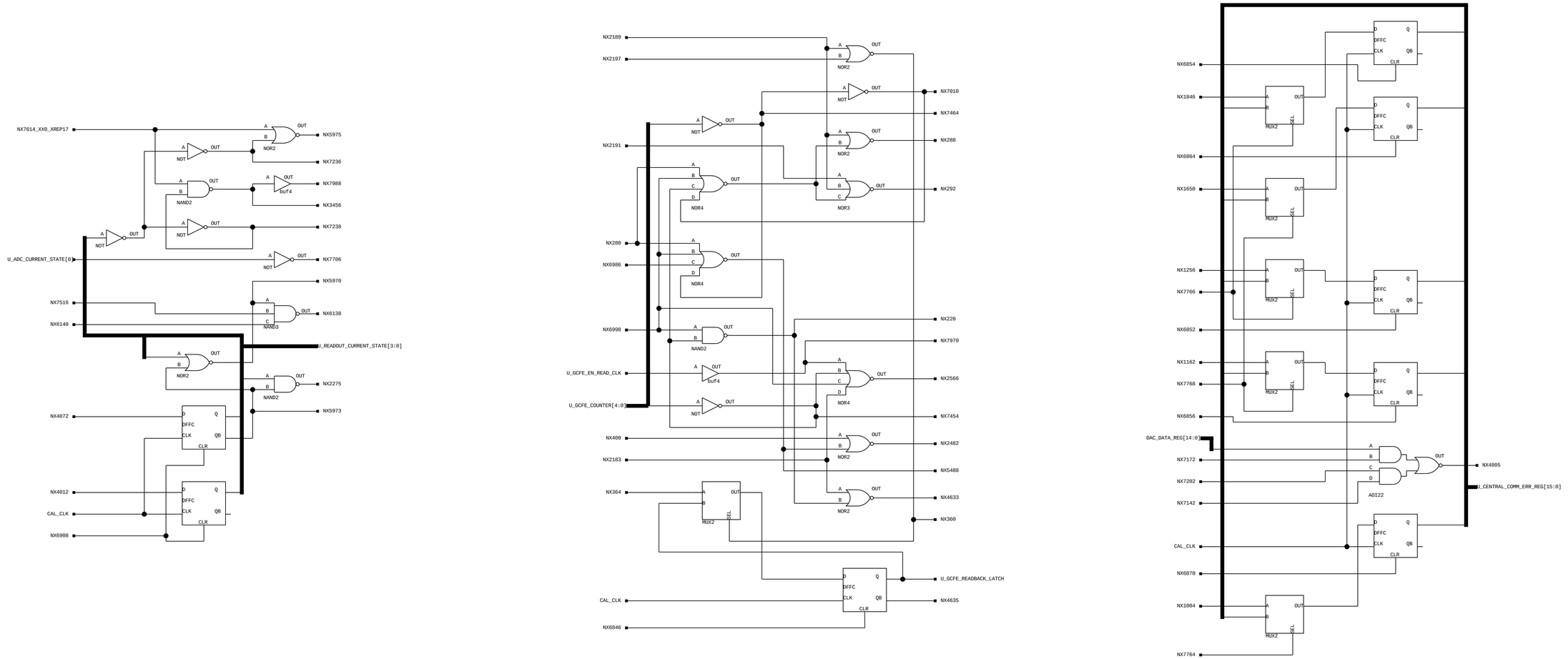
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 30 OF 51



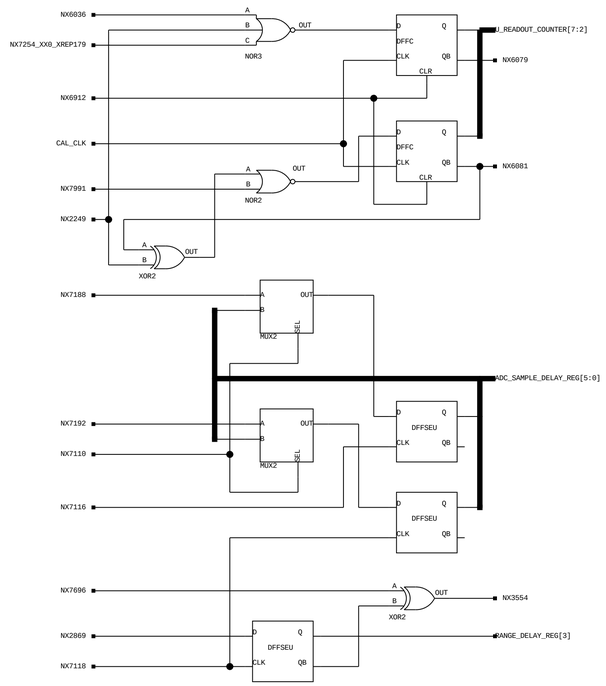
gcrc_core

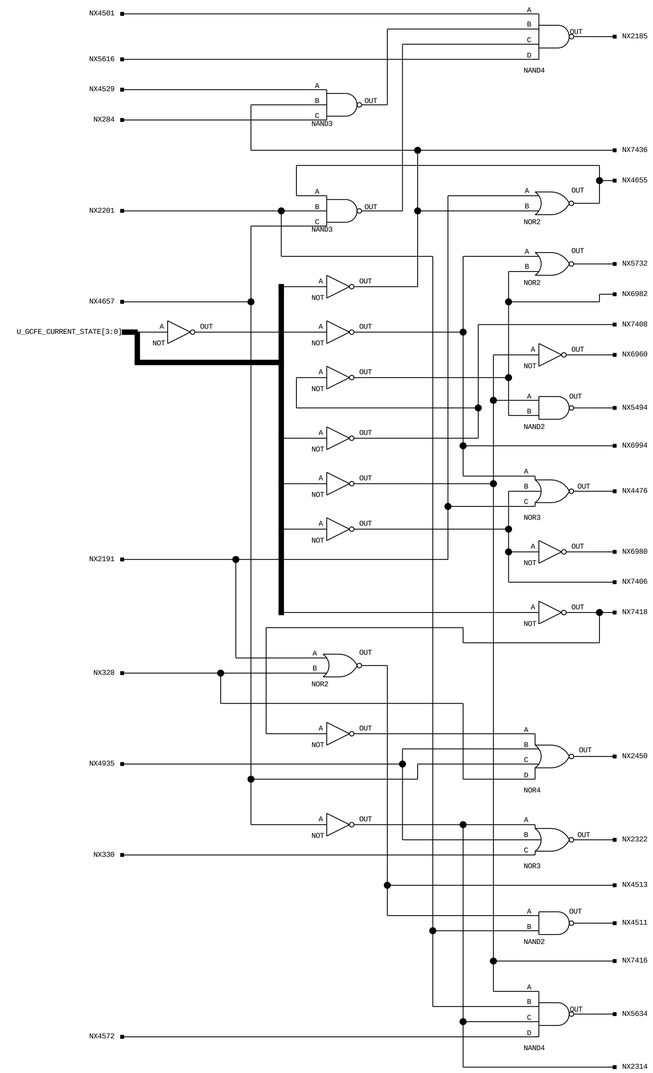
WIR:gcrc_chip

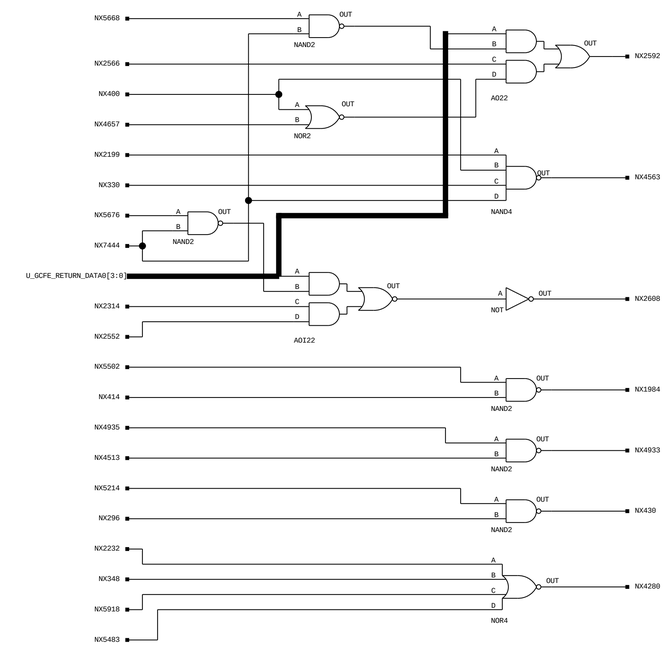
SCH:gcrc_core

20 Dec 02 17:44

SHEET 31 OF 51





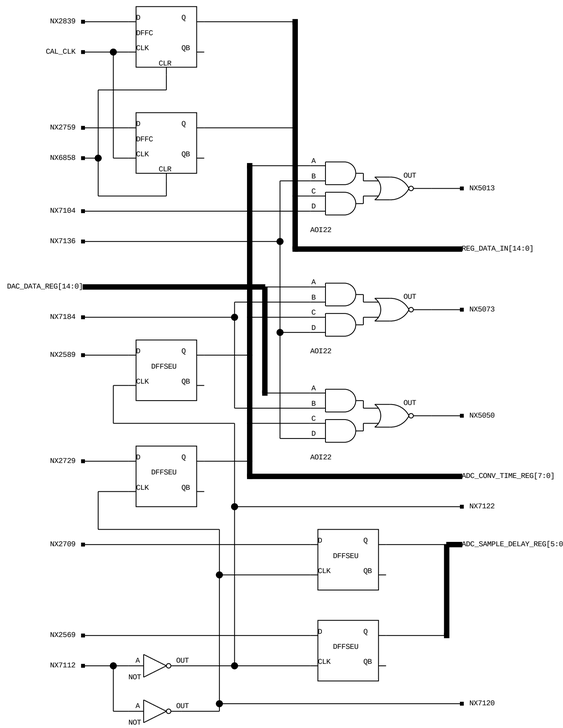


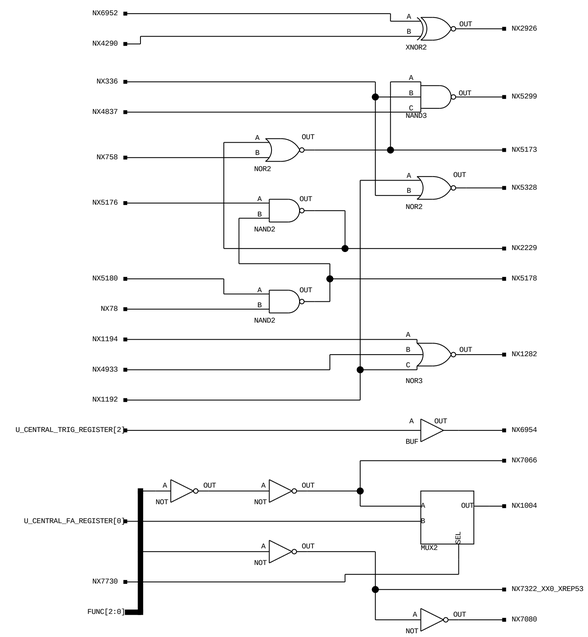
gcrc_core

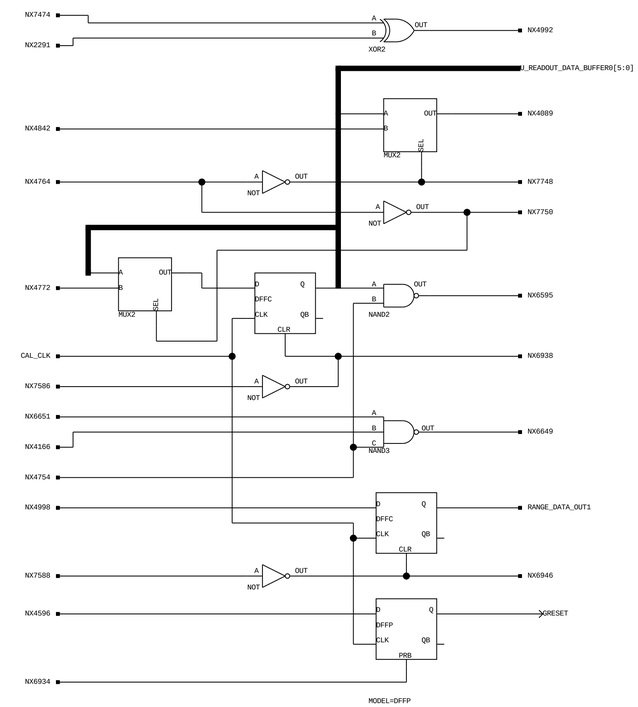
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44	SHEET 32 OF 51





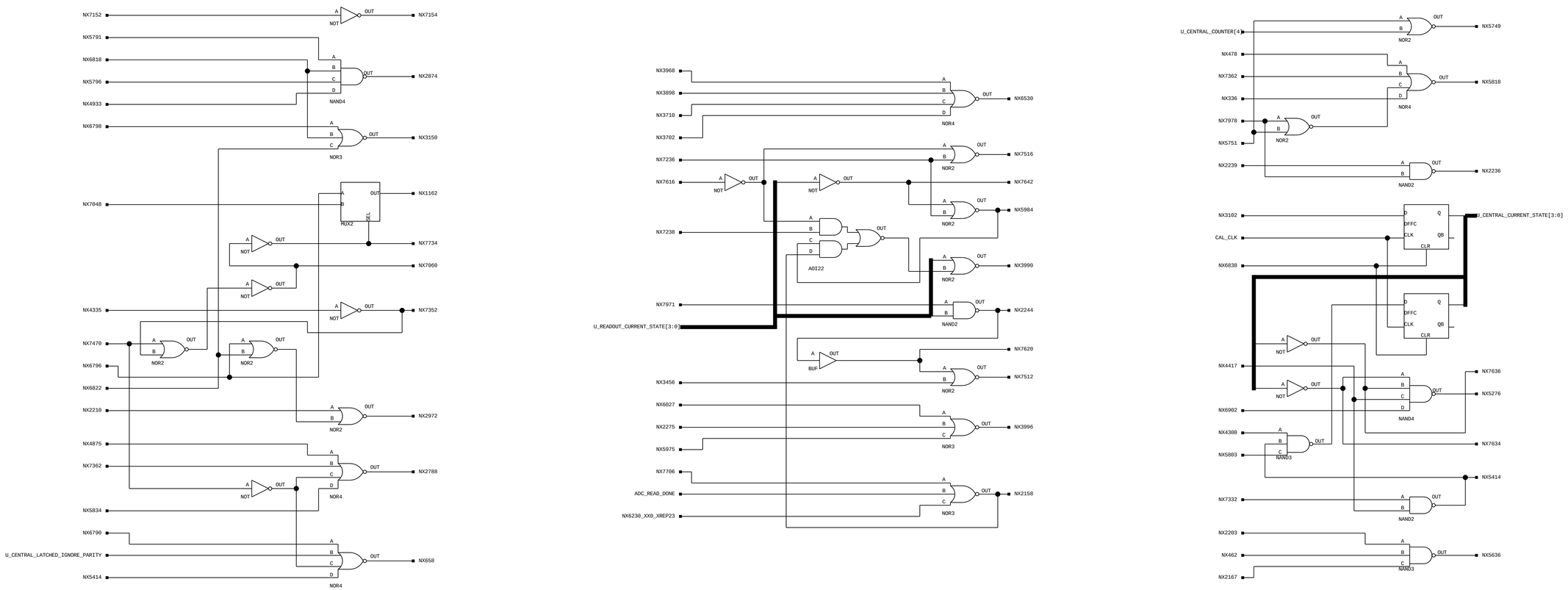


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44



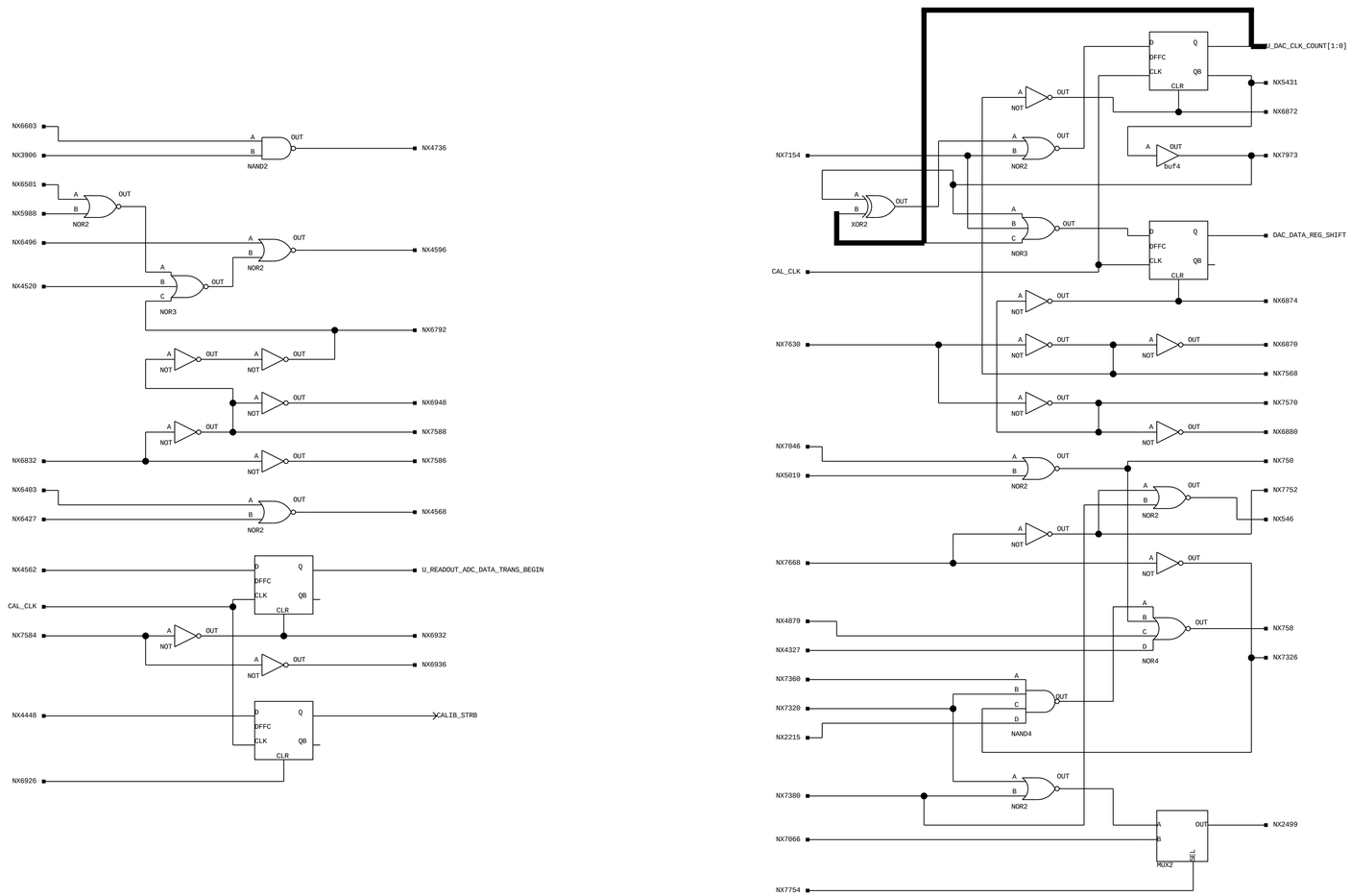
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 34 OF 51

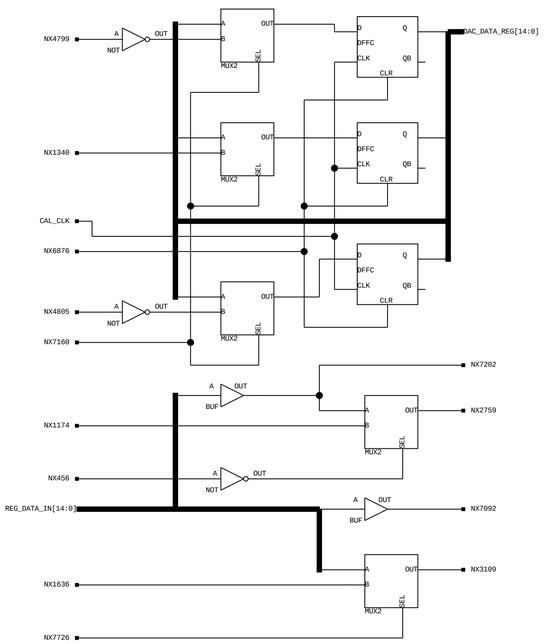


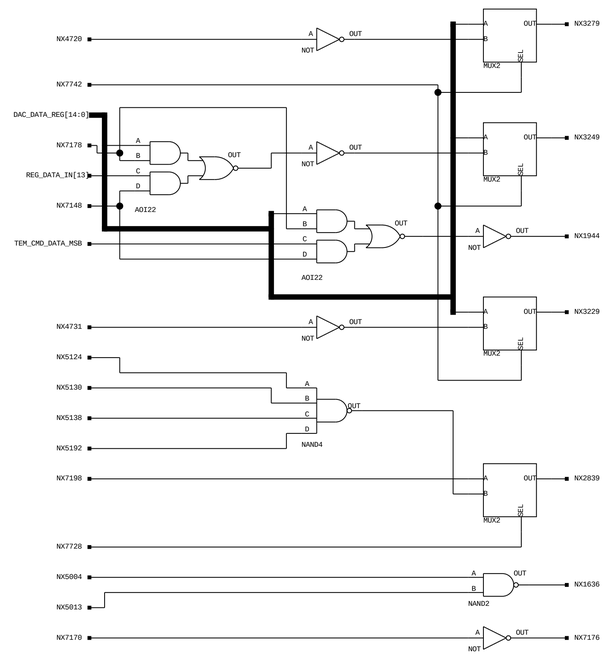
gcrc_core

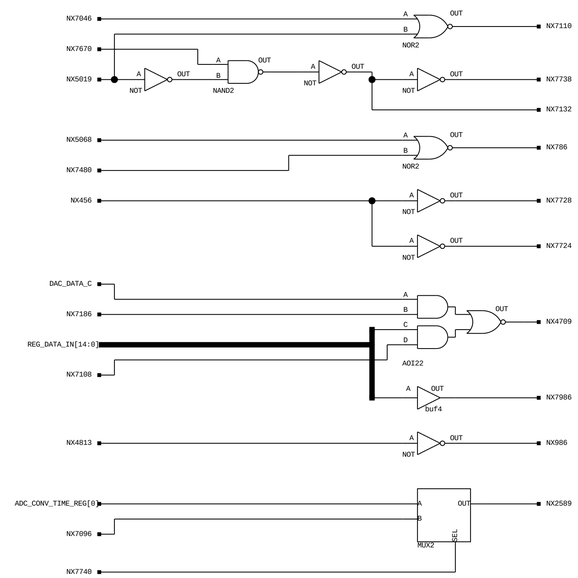
WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44







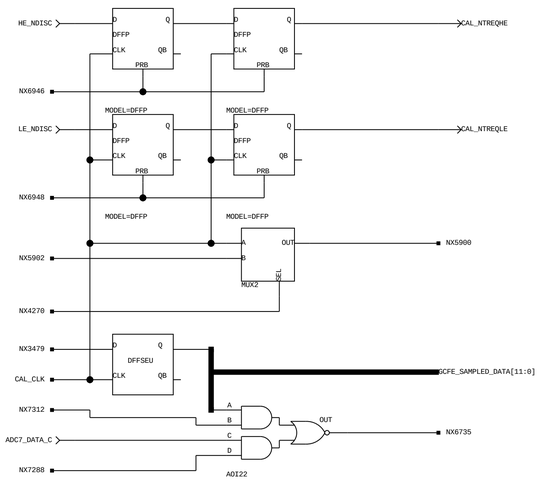
gcrc_core

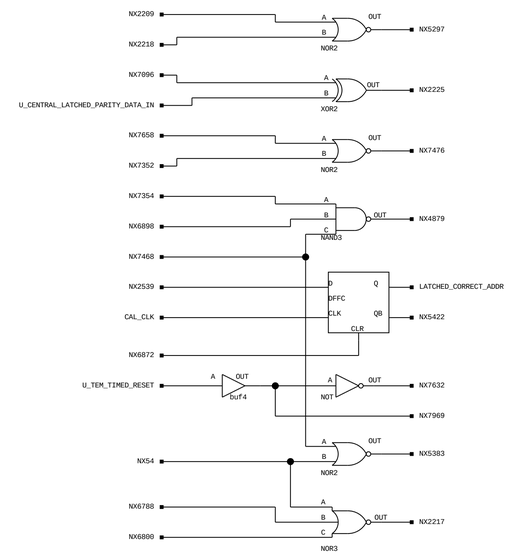
WIR:gcrc_chip

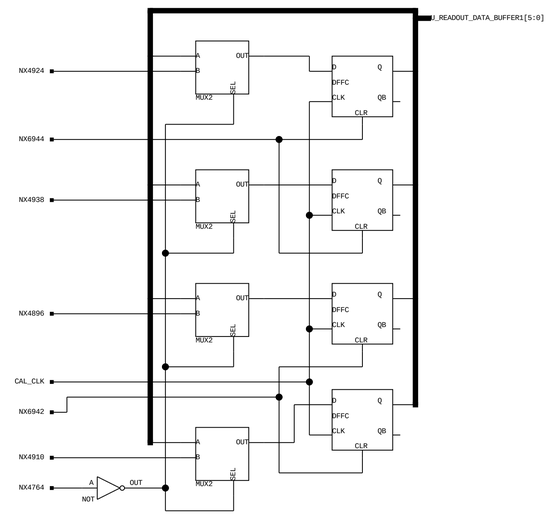
SCH:gcrc_core

20 Dec 02 17:44

SHEET 36 OF 51





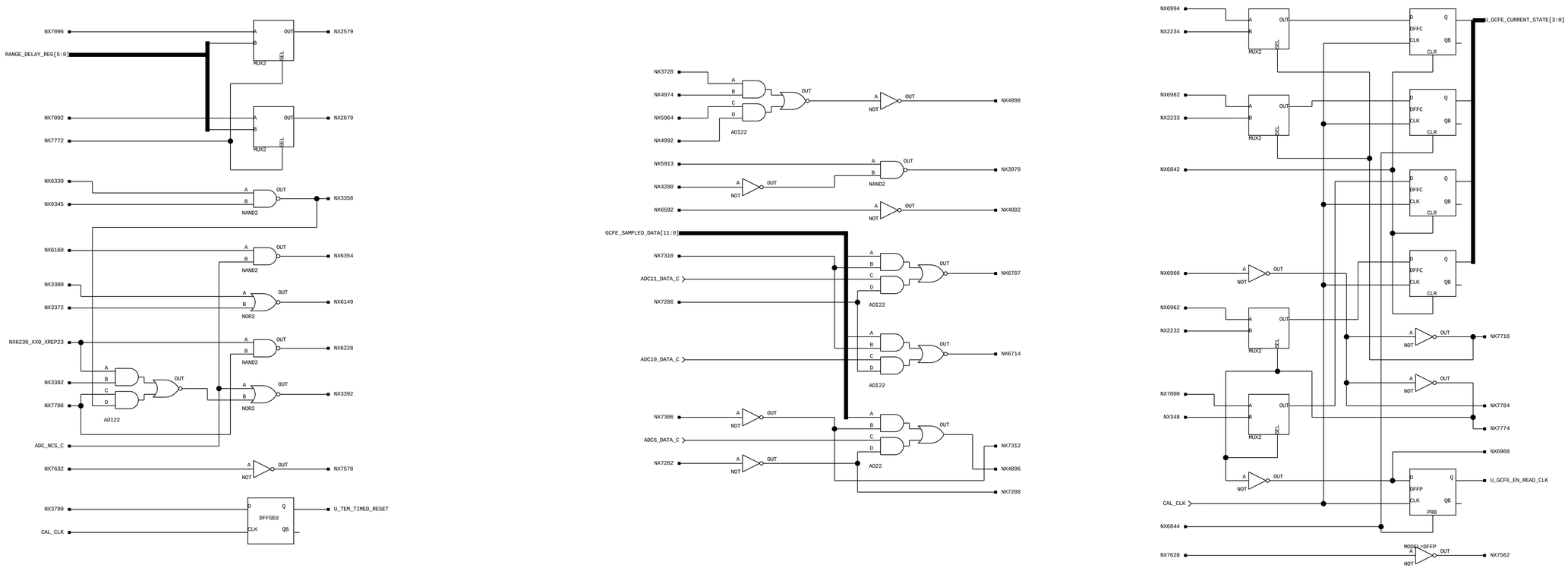


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44



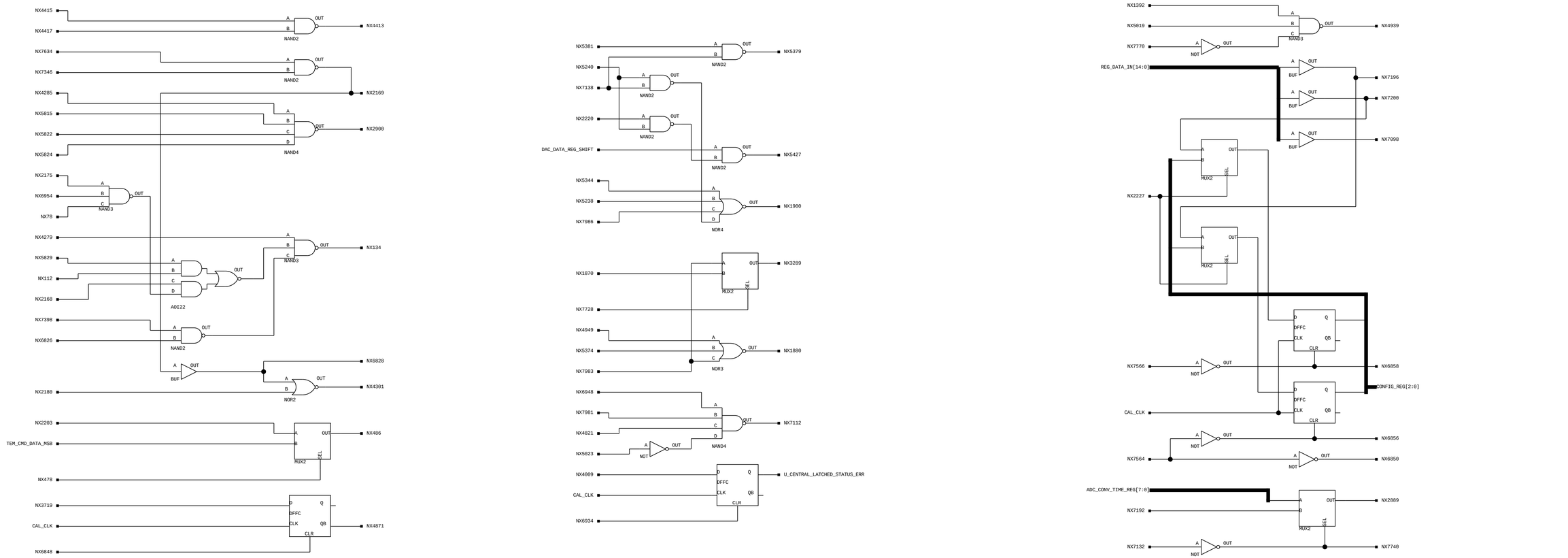
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 38 OF 51



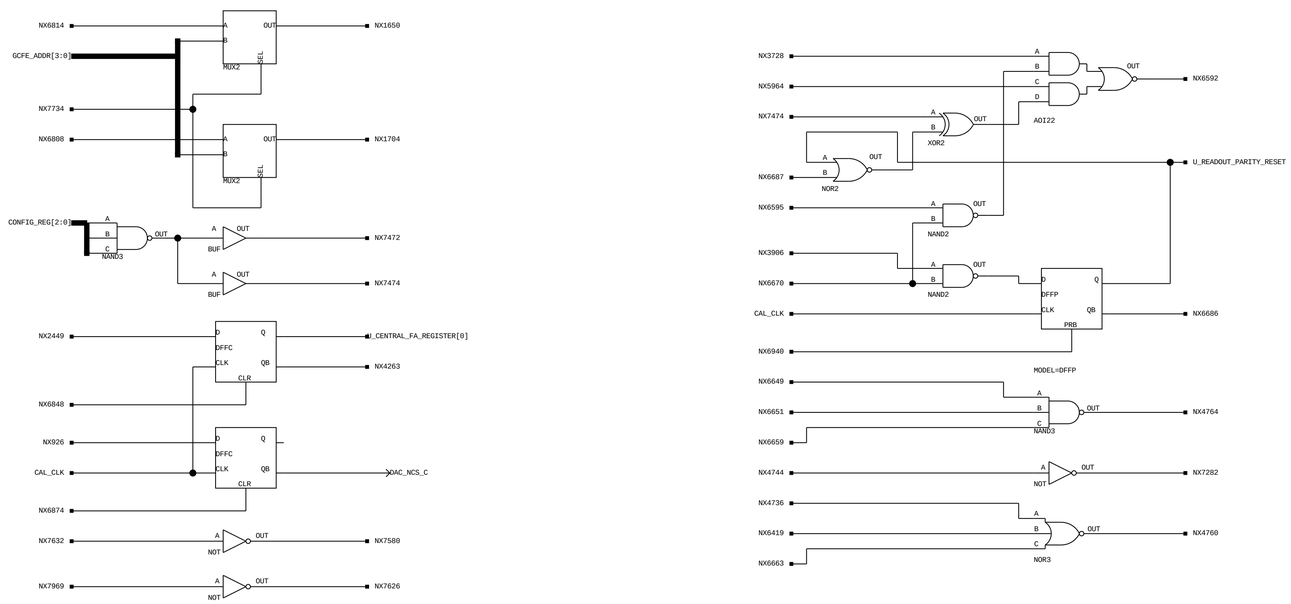
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 39 OF 51



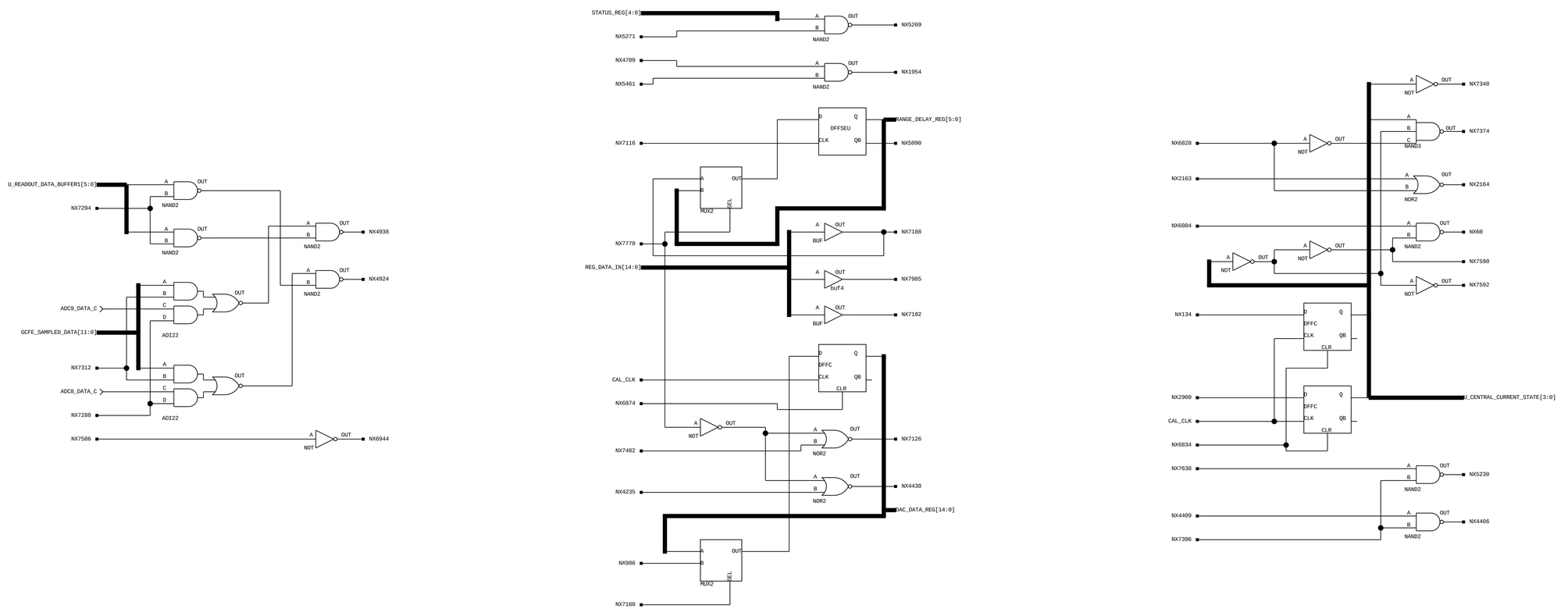
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 40 OF 51



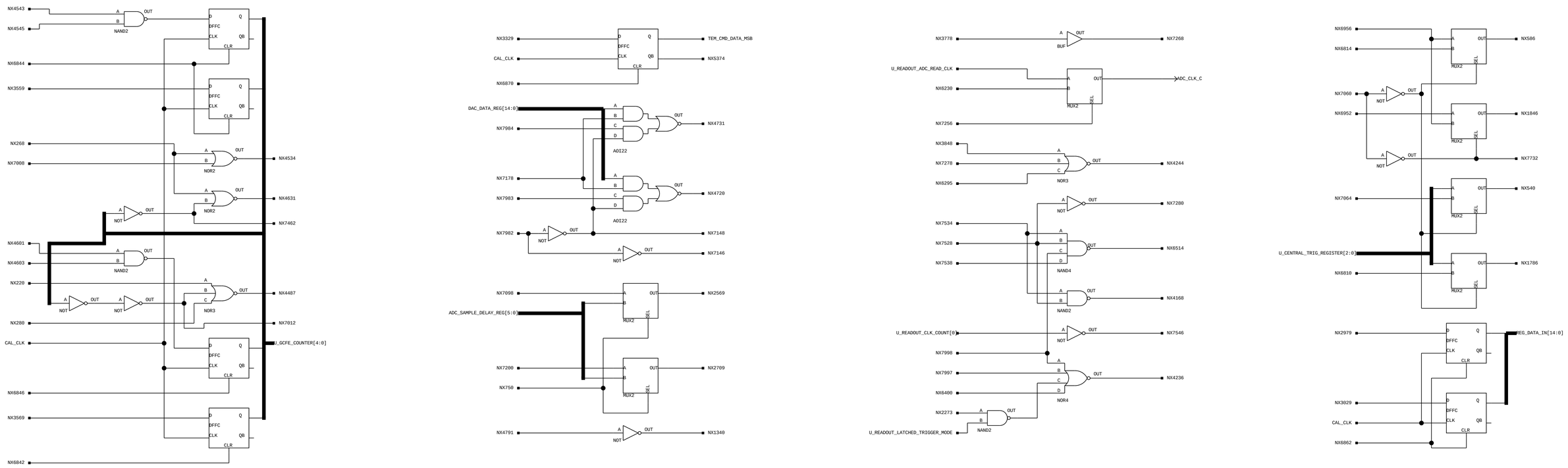
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 41 OF 51



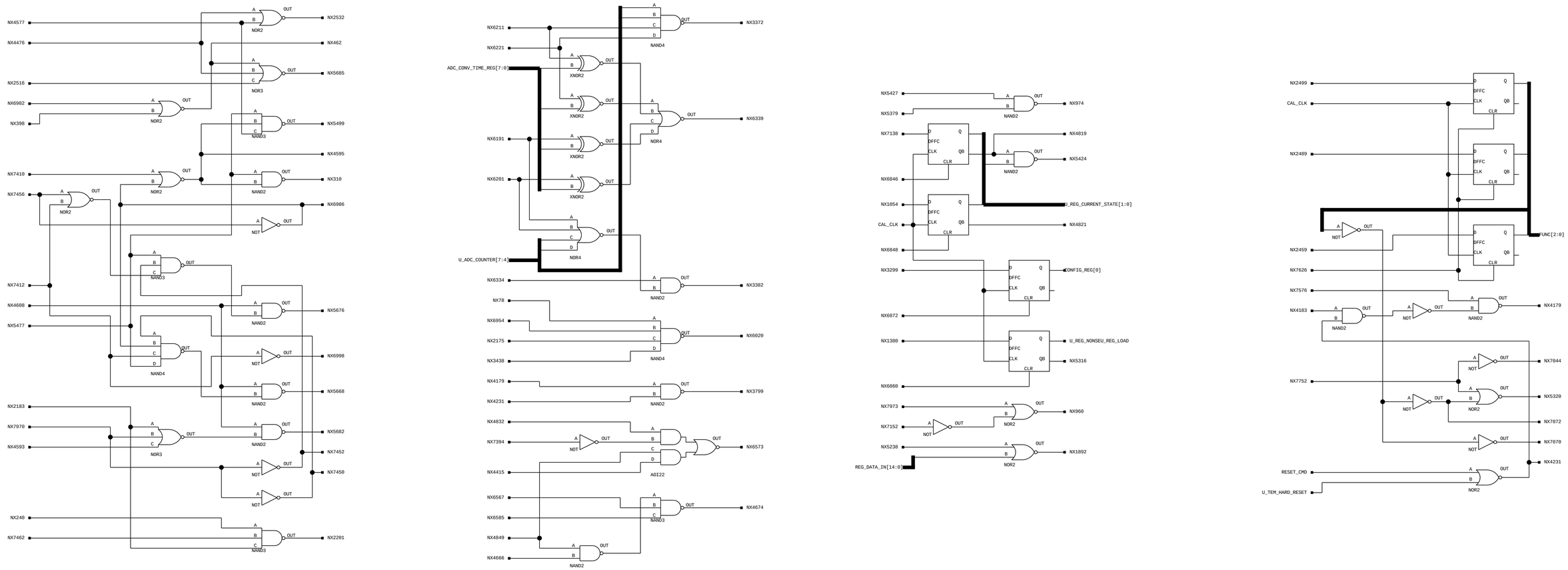
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 42 OF 51



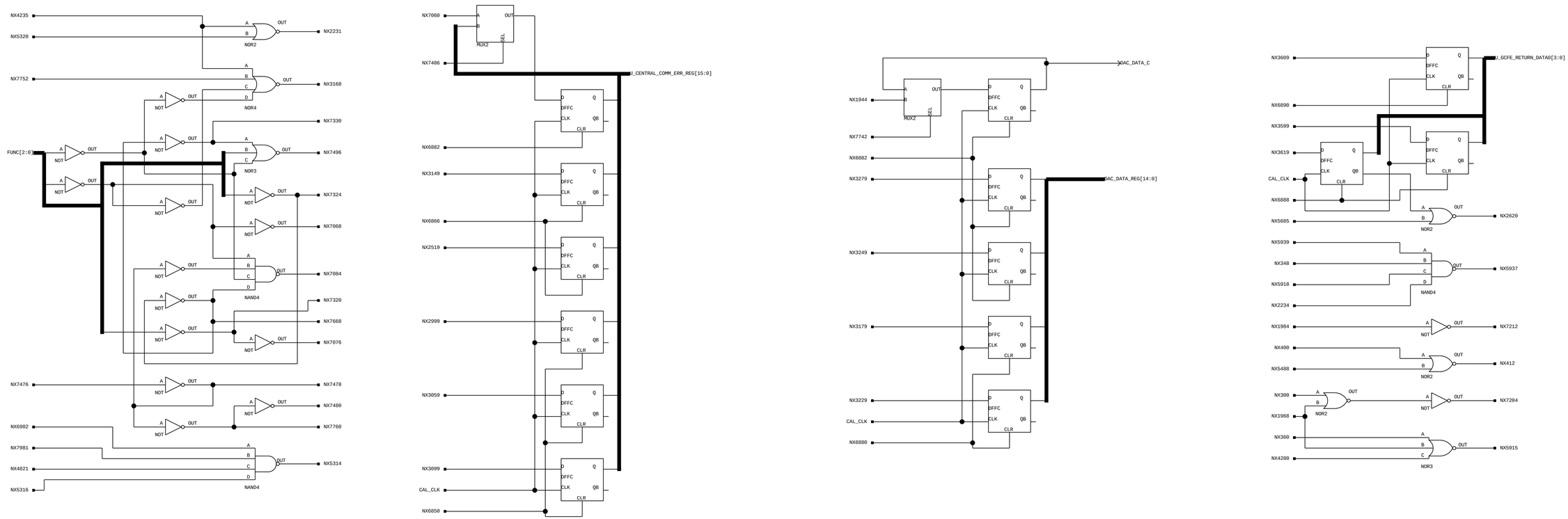
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 43 OF 51



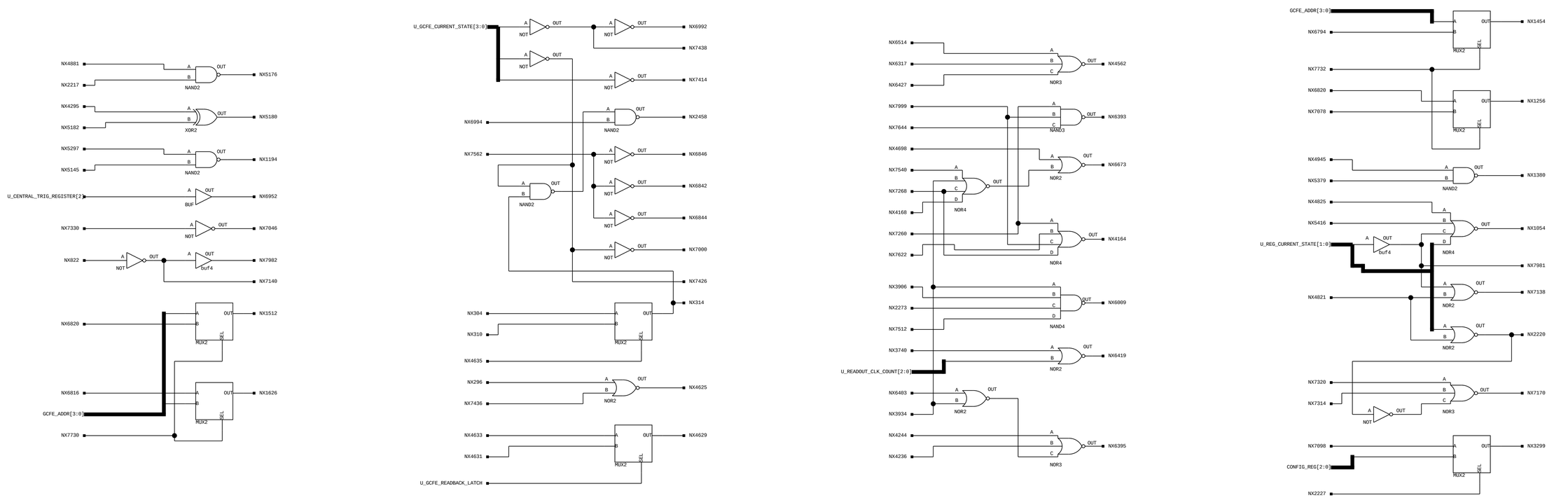
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 44 OF 51



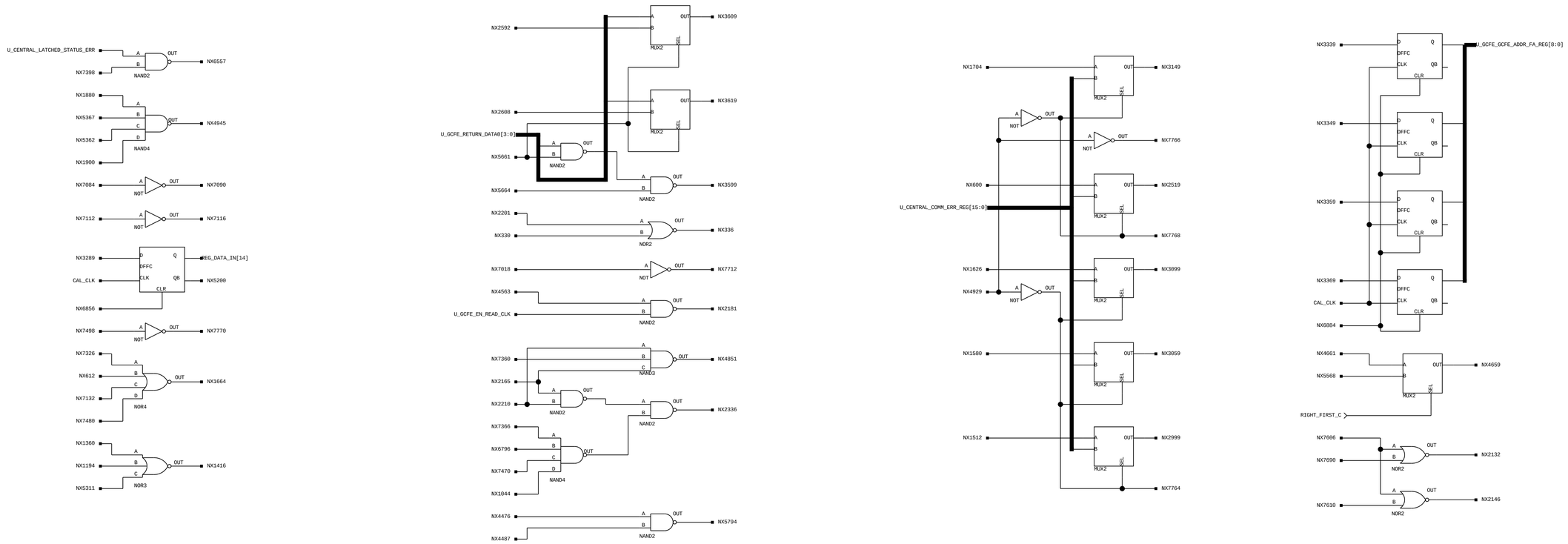
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 45 OF 51

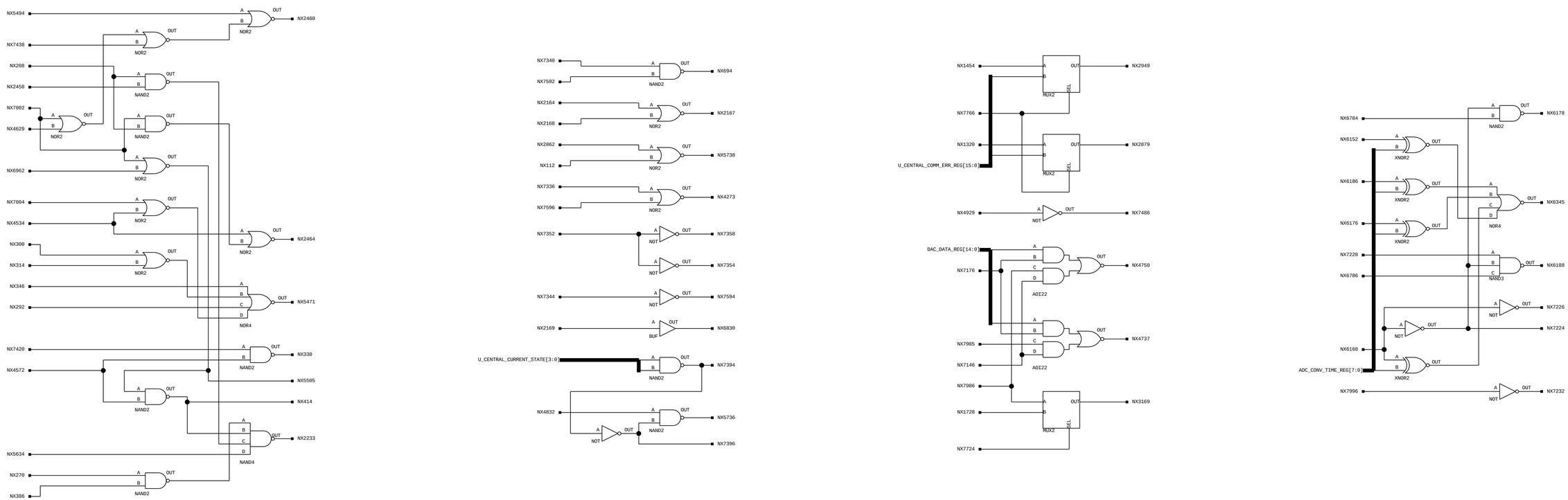


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

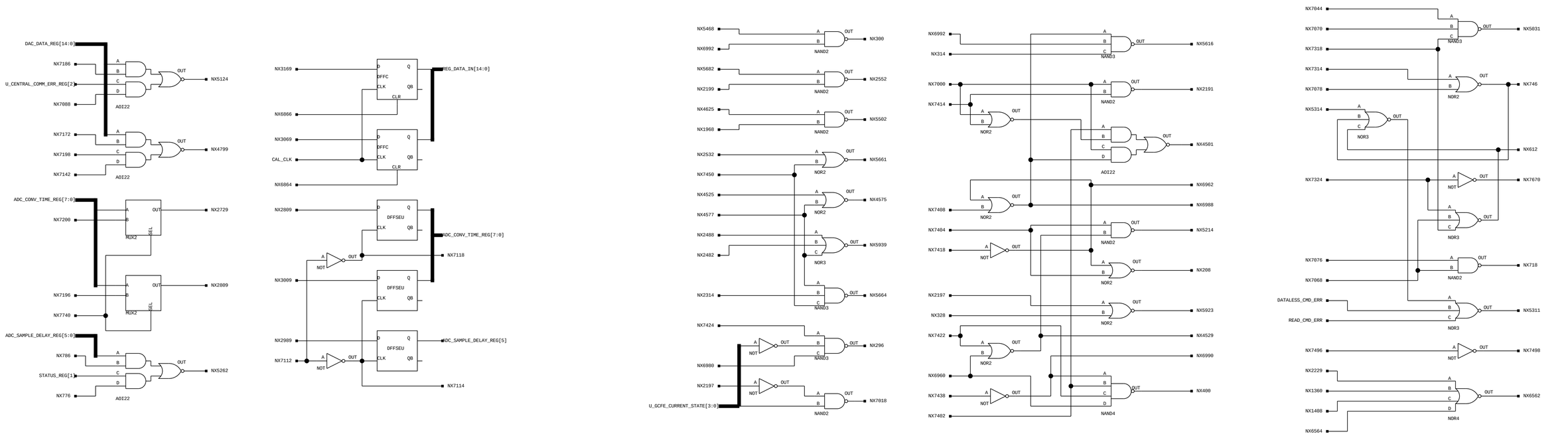


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44



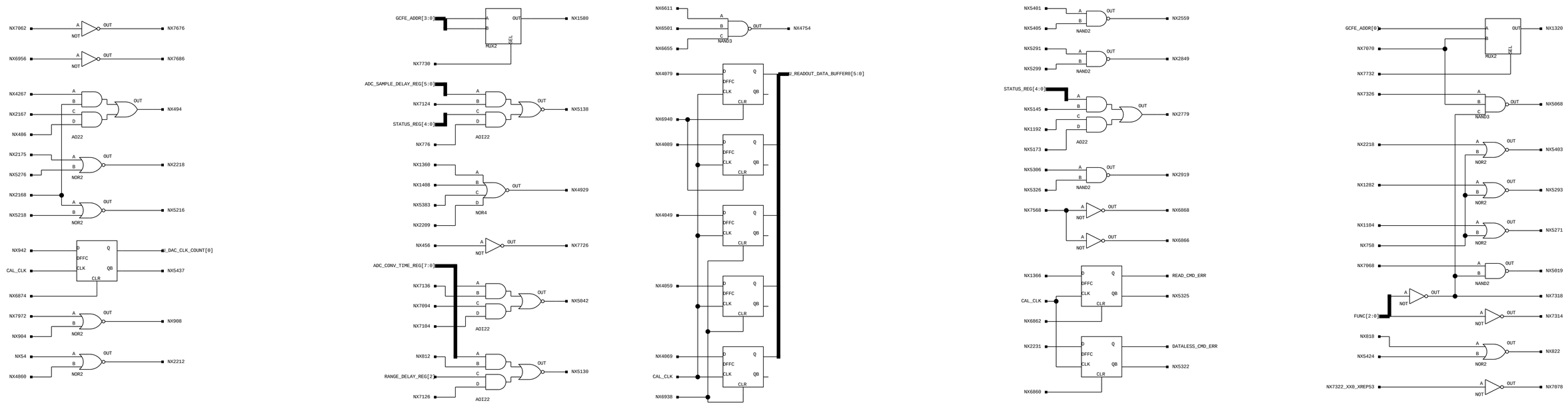
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 48 OF 51

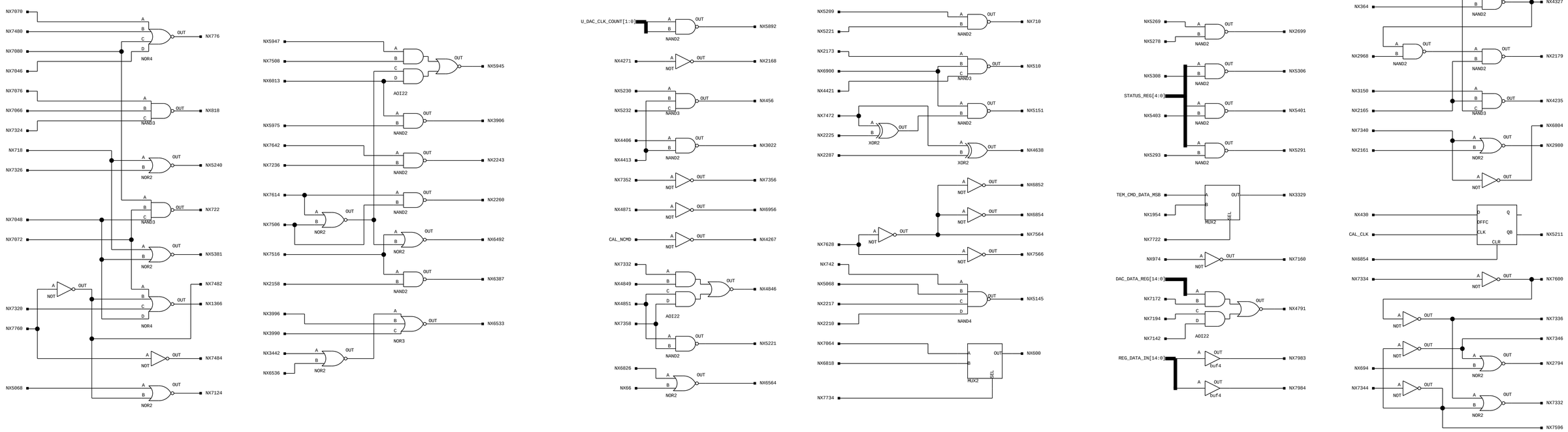


gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44



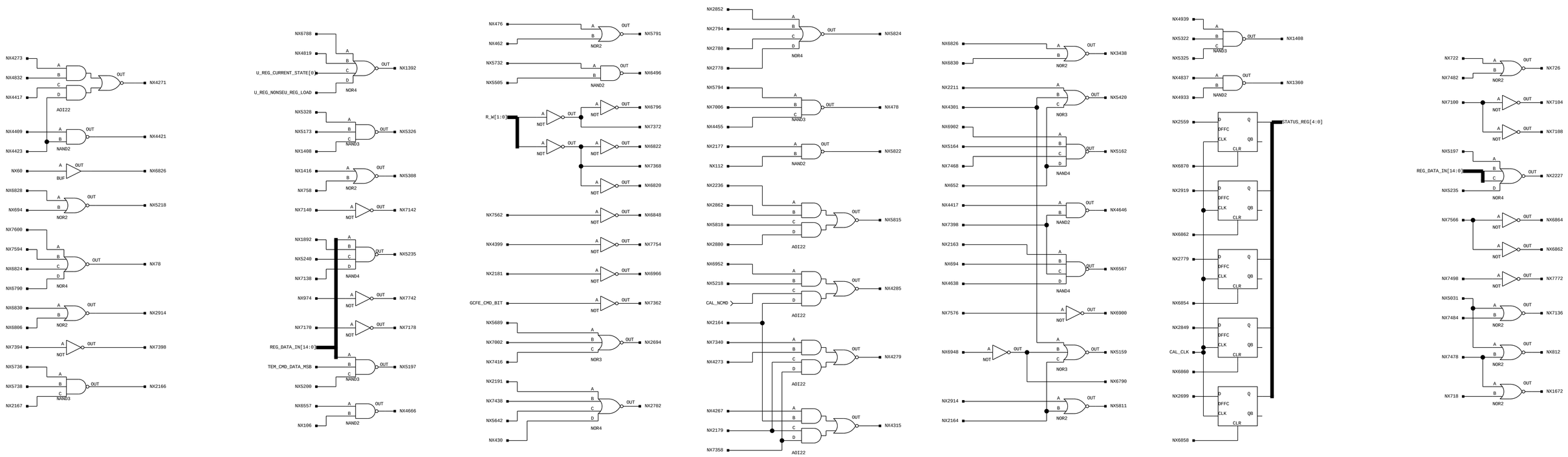
gcrc_core

WIR:gcrc_chip

SCH:gcrc_core

20 Dec 02 17:44

SHEET 50 OF 51



gcrc_core

WIR:gcrc_chip

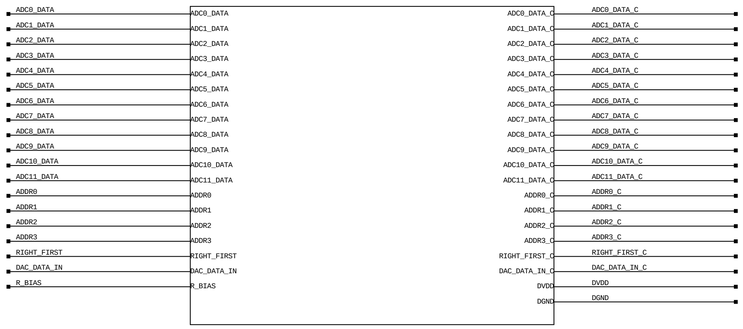
SCH:gcrc_core

20 Dec 02 17:44

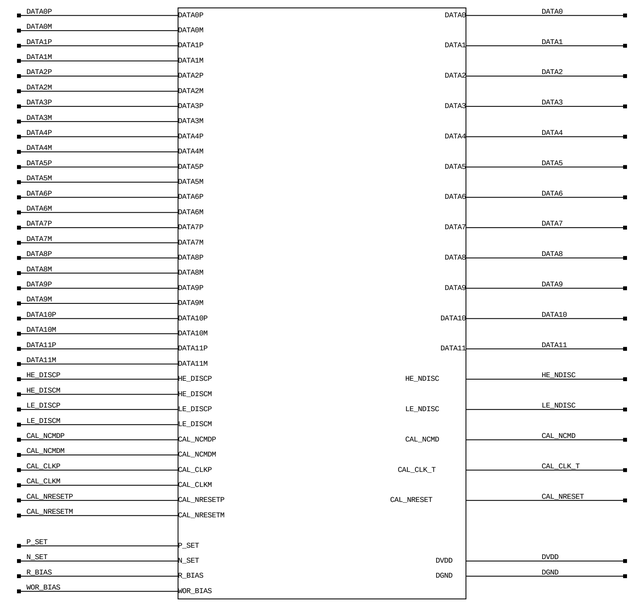
SHEET 51 OF 51

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE

CMOS RECEIVERS



LVDS RECEIVERS and WOR RECEIVERS



GCRC_INPUTS

2-24-2003_11:31

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

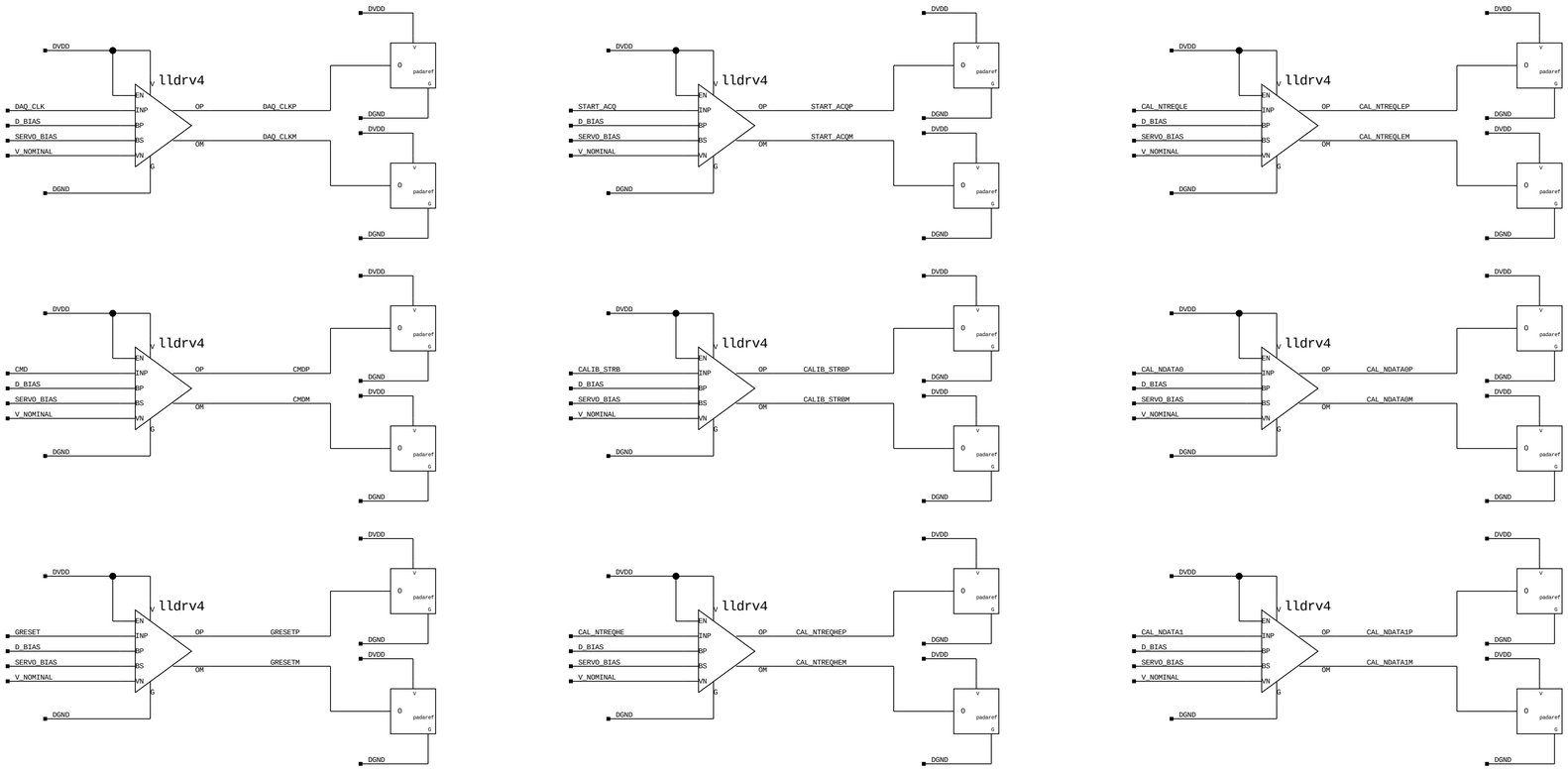
GCRC_CHIP

INPUT PADS

VERSION 5

LAT-DS-03519-50 D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



LVDS_TRANSMITTERS

2-24-2003_11:37

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

GCRC_CHIP

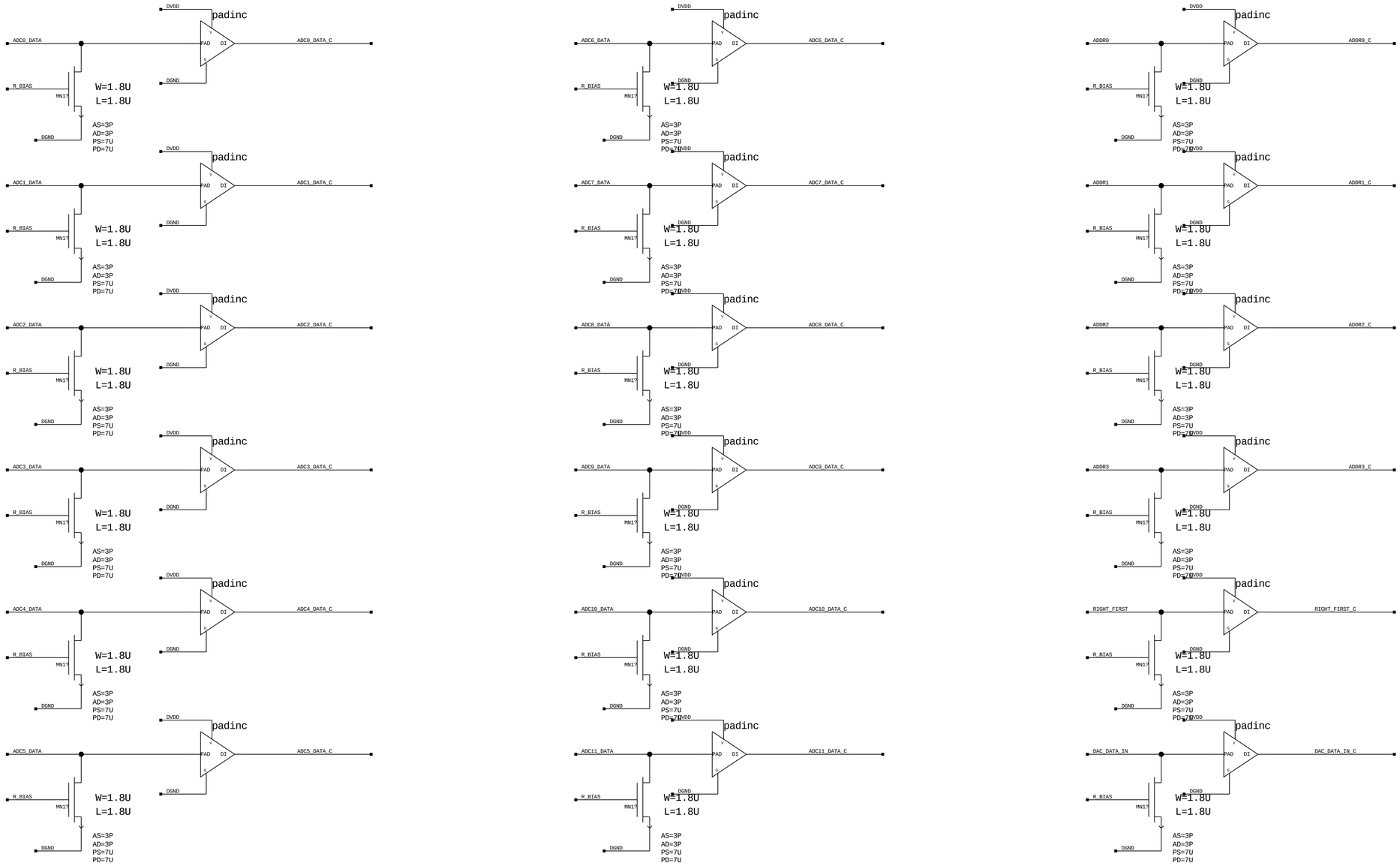
LVDS DRIVERS AND BIASES

VERSION 5

LAT-DS-03519-50

D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE



CMOS_RECEIVERS

2-24-2003_11:29

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/94	
ENGR	Dieter Freytag - SLAC	4/5/94	
ENGR	Mark Freytag - SLAC	4/5/94	
DFTR	Mark Freytag - SLAC	4/5/94	
CHKR			

SHEET 4 OF 60

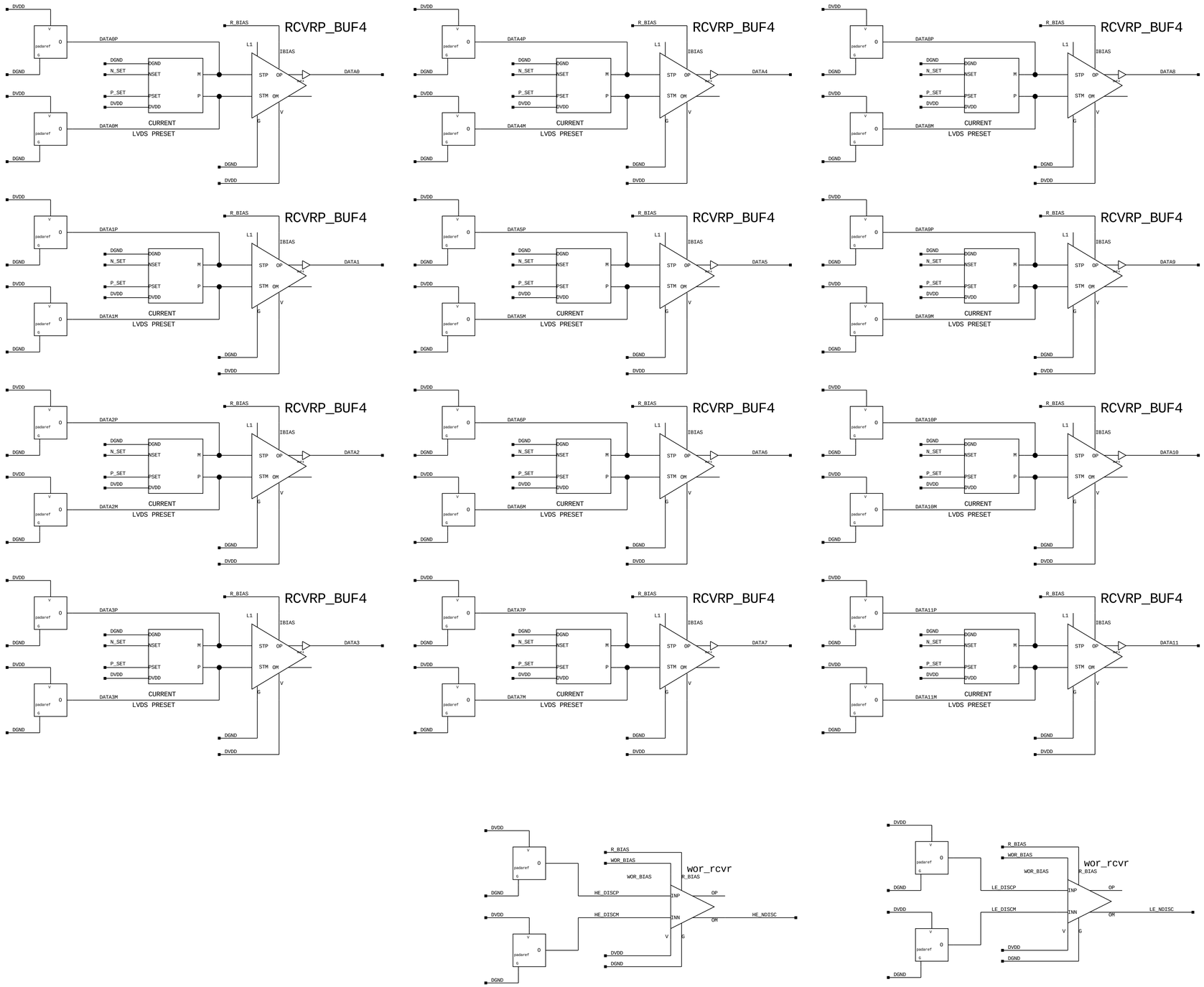
GCRC_CHIP

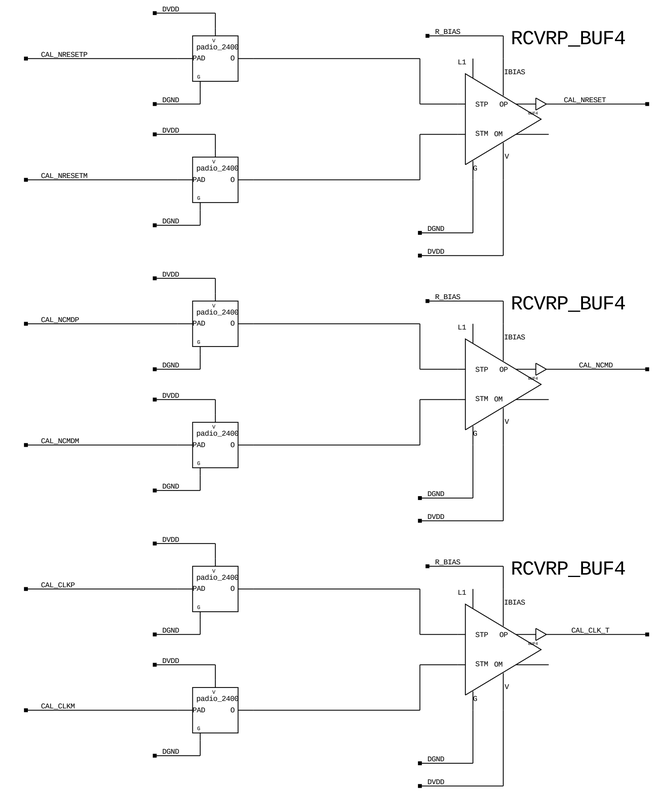
VERSION 5

LAT-DS-03519-50

D

REV	DESCRIPTION	DFTR	CHKR	ENGR APPV	DATE





LVDS_RECEIVERS

2-24-2003_11:30

STANFORD LINEAR ACCELERATOR CENTER

U.S. DEPARTMENT OF ENERGY

STANFORD UNIVERSITY — STANFORD, CALIFORNIA

PROPRIETARY DATA OF STANFORD UNIVERSITY AND/OR U.S. DEPARTMENT OF ENERGY. RECIPIENT SHALL NOT PUBLISH THE INFORMATION WITHIN UNLESS GRANTED SPECIFIC PERMISSION OF STANFORD UNIVERSITY.

		DATE	APPROVALS
ENGR	Jim Ampe - NRL	4/5/04	
ENGR	Dieter Freytag - SLAC	4/5/04	
ENGR	Mark Freytag - SLAC	4/5/04	
DFTR	Mark Freytag - SLAC	4/5/04	
CHKR			

GCRC_CHIP

LVDS RECEIVERS

VERSION 5

LAT-DS-03519-50

D

GLAST LAT SPECIFICATION	Document # **LAT-DS-03519-50**	Date Effective 12 April 2004
	Prepared by(s) Jim Ampe	Supersedes None
	Subsystem/Office Calorimeter Subsystem	
Document Title **Calorimeter Readout Control ASIC – VHDL Code**		

Gamma-ray Large Area Space Telescope (GLAST)

Large Area Telescope (LAT)

Calorimeter GCRC5 Flight ASIC VHDL Code

DOCUMENT APPROVAL

Prepared by:

James Ampe Naval Research Lab	Date

Approved by:

Gunther Haller LAT Electronics Manager	Date

CHANGE HISTORY LOG

Revision	Effective Date	Description of Changes
1	4/12/04	Created document from GCRC ver 5 flight VHDL Code

Table of Contents

Table of Contents 4
List of Figures 4
1 PURPOSE 5
2 DEFINITIONS 5
2.1 Acronyms 5
3 APPLICABLE DOCUMENTS 5
4 GCRC OVERVIEW 5
5 GCRC5 ASIC MODULES 7
5.1 GCRC_CHIP.VHD 7
5.2 GCRC_CENTRAL.VHD 18
5.3 GCRC_REG.VHD 35
5.4 GCRC_DAC.VHD 40
5.5 GCRC_GCFE.VHD 43
5.6 GCRC_CALIB.VHD 56
5.7 GCRC_ADC.VHD 58
5.8 GCRC_READOUT.VHD 61
5.9 GCRC_TRIG.VHD 74
5.10 GCRC_TEM.VHD 76
6 GCRC VHDL Revision Notes 78

List of Figures

Figure 1. GCRC Functional Block Diagram 6

1 PURPOSE

This document documents the VHDL code in the GLAST Large Area Telescope (LAT) flight Calorimeter Readout Control version 5 (GCRC5) ASIC.

2 DEFINITIONS

2.1 Acronyms

GLAST – Gamma-ray Large Area Space Telescope

GCFE – Glast Calorimeter Front-end Electronics

GCRC – Glast Calorimeter Readout Control electronics

LAT – Large Area Telescope

CAL – Calorimeter Detector

TREQ – Level 1 Trigger Request

TEM – Tower Electronics Module

3 APPLICABLE DOCUMENTS

[1] Calorimeter Readout Control ASIC – Conceptual Design, Submission 5, LAT-SS-00208-05.

4 GCRC OVERVIEW

One of the two custom ASICs required for the GLAST Calorimeter is the Glast Calorimeter Readout Control (GCRC) Application Specific Integrated Circuit (ASIC). The basic function of the GCRC is the interface between the Tower Electronics Module (TEM) and the Glast Calorimeter Front End (GCFE) ASIC. The GCRC passes commands from the TEM to GCFEs, controls event readout of the log ends, and passes data back to the TEM. The GCRC is a digital chip with Low Voltage Differential Signalling (LVDS) Inputs and Outputs (I/O). The GCRC ASIC is fabricated in the processes for the 0.5 um Agilent CMOS process.

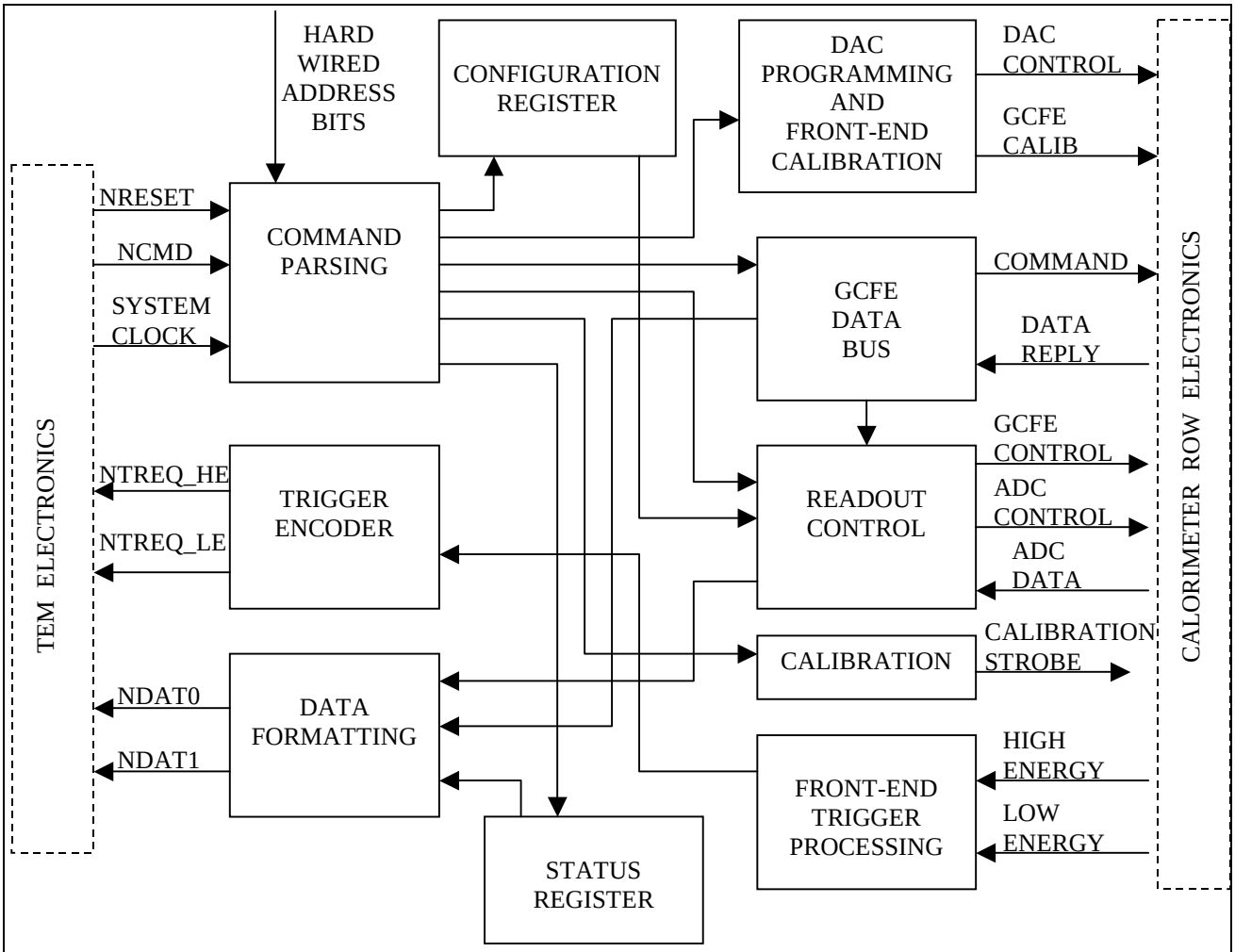


Figure 1. GCRC Functional Block Diagram

5 GCRC5 ASIC MODULES

5.1 GCRC_CHIP.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_CHIP.vhd
--
--Created by Jim Ampe / Michael Condon, Naval Research Lab
--
--Revision: 1.0 (11-12-01)
--          First version.
--     1.0 Rev B (11-14-01), Deleted signal not used.
--     1.0 Rev D (11-29-01), Modified chip signal names to be more
--       consistant with GCFE signal naming and documentation.
--     2.0 (1-2-02), Added signals for added Status Register bit and
--       added GCRC Config Register bit.  Also added signals for
--       Read_Cmd_Err and Write_Cmd_err.
--     3.0 (2-20-02), Changed LE & HE DISC signal names to indicate
--       asserted low signals, ie. LE_NDISC.
--     4.0 (3-5-02) Added code to pass clocked GCFE data to READOUT module.
--       Delay of GCFE data was causing problems. Rename CMOS level I/O
--       signals to have "_C" suffix.
--     4.0 Rev A (3-29-02) Passed Latched_Correct_Addr and Correct_Addr
--       to GCRC_GCFE module.
--     5.0 (9-10-02) Changed TEM command decoding.  Added TEM Reset command.
--     5.1 No Change.
--     5.2 No Change.
--     5.3 (12-11-02) Added signals between GCFE module and Central module.
--
--Function: Top level design file for GCRC.
--
--
-----------------------------------------------------------------------------------------

--library GCRC;
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
--use gcrc.gcrc_pkg.all;


entity GCRC_CHIP is
  port (
    ADDR0_C: in STD_LOGIC;
    ADDR1_C: in STD_LOGIC;
    ADDR2_C: in STD_LOGIC;
    ADDR3_C: in STD_LOGIC;
    RIGHT_FIRST_C: in STD_LOGIC;
    CAL_NRESET: in STD_LOGIC;
    CAL_NTREQLE: buffer STD_LOGIC;
    CAL_NTREQHE: buffer STD_LOGIC;
    CAL_NCMD: in STD_LOGIC;
    CAL_CLK: in STD_LOGIC;
    CAL_NDATA0: buffer STD_LOGIC;
    CAL_NDATA1: buffer STD_LOGIC;
    ADC0_DATA_C: in STD_LOGIC;  --12 ADC Data Lines
    ADC1_DATA_C: in STD_LOGIC;
```

```
    ADC2_DATA_C: in STD_LOGIC;
    ADC3_DATA_C: in STD_LOGIC;
    ADC4_DATA_C: in STD_LOGIC;
    ADC5_DATA_C: in STD_LOGIC;
    ADC6_DATA_C: in STD_LOGIC;
    ADC7_DATA_C: in STD_LOGIC;
    ADC8_DATA_C: in STD_LOGIC;
    ADC9_DATA_C: in STD_LOGIC;
    ADC10_DATA_C: in STD_LOGIC;
    ADC11_DATA_C: in STD_LOGIC;
    LE_NDISC: in STD_LOGIC;
    HE_NDISC: in STD_LOGIC;
    DATA0: in STD_LOGIC;
    DATA1: in STD_LOGIC;   --12 GCFE Data Lines
    DATA2: in STD_LOGIC;
    DATA3: in STD_LOGIC;
    DATA4: in STD_LOGIC;
    DATA5: in STD_LOGIC;
    DATA6: in STD_LOGIC;
    DATA7: in STD_LOGIC;
    DATA8: in STD_LOGIC;
    DATA9: in STD_LOGIC;
    DATA10: in STD_LOGIC;
    DATA11: in STD_LOGIC;
    DAC_DATA_IN_C : in STD_LOGIC;
    DAQ_CLK: buffer STD_LOGIC;
    CMD: buffer STD_LOGIC;
    GRESET: buffer STD_LOGIC;
    START_ACQ: buffer STD_LOGIC;
    CALIB_STRB: buffer STD_LOGIC;
    ADC_NCS_C: buffer STD_LOGIC;
    ADC_CLK_C: buffer STD_LOGIC;
    DAC_NCS_C: buffer STD_LOGIC;
    DAC_CLK_C: buffer STD_LOGIC;
    DAC_DATA_C: buffer STD_LOGIC
  );
end GCRC_CHIP;


architecture top_level of GCRC_CHIP is
signal NRESET         : std_logic;
signal CLK            : std_logic;
signal TEM_CMD          : std_logic;
signal CONFIG_PARITY       : std_logic; -- From GCRC register.
signal WIRED_ADDR       : std_logic_vector(3 downto 0);-- Hardwired GCRC address
signal STATUS_REPLY_ERR    : std_logic; -- GCFE not reply, timeout error.
signal R_W            : std_logic_vector(1 downto 0); -- Read/Write not, TEM command bit
signal FUNC           : std_logic_vector(2 downto 0); -- TEM function bits.
signal GCFE_CLK_ENV     : std_logic; -- DAQ_CLK envelope from GCRC READOUT module.
signal GCFE_RESET_ENV   : std_logic;   -- RESET envelope from GCRC READOUT module.
signal GCFE_DELAYED_RTN_DATA : std_logic; -- Serial data returned from 1 of 12 GCFEs.
signal GCFE_TRANS       : std_logic;    -- GCFE is transmitting data to GCRC.
signal GCFE_WRITE       : std_logic; -- GCRC is writing to GCFE.
signal GCFE_CMD_BIT     : std_logic;    -- TEM GCFE command bit
signal GCFE_ADDR        : std_logic_vector(3 downto 0); -- Address in GCFE command.
signal GCFE_DATA      : std_logic_vector(11 downto 0); -- 12 GCFE serial data inputs.
signal TEM_CMD_DATA_MSB    : std_logic; -- TEM command data most significant bit.
signal GCRC_TRANS_RECV     : std_logic; -- GCRC is transmitting to the TEM
signal GCRC_DATA_OUT0      : std_logic; -- GCRC serial data out0
```

```
signal GCRC_DATA_OUT1      : std_logic; -- GCRC serial data out0
signal GCRC_IN_READOUT     : std_logic; -- GCRC is performing range readout.
signal GCRC_DAC_WRITE   : std_logic;  -- GCRC is writing to the DAC, to GCRC_TRIG.
signal STATUS_TRIG_ERR     : std_logic; -- Bit to be placed in status register.
signal STATUS_FA_ERR       : std_logic; --  Bit to be placed in status register.
signal STATUS_DATA_ERR     : std_logic; -- Bit to be placed in status register.
signal STATUS_REG       : std_logic_vector(4 downto 0); -- Status error register.
signal RANGE_DELAY_REG     : std_logic_vector(5 downto 0); -- Peak hold settling time.
signal ADC_SAMPLE_DELAY_REG : std_logic_vector(5 downto 0); -- Analog Mux settling
time.
signal ADC_CONV_TIME_REG   : std_logic_vector(7 downto 0);  -- ADC Conversion time.
signal DAC_DATA_REG     : std_logic_vector(15 downto 0); -- Onboard DAC data.
signal CONFIG_REG       : std_logic_vector(2 downto 0);-- GCRC configuration register.
--signal DEAD_TIME_REG     : std_logic_vector(7 downto 0); -- GCFE trigger dead time
register.
signal GCFE_SAMPLED_DATA   : std_logic_vector(11 downto 0); -- Clocked GCFE data.
signal CALIB_CMD      : std_logic;    -- Decoded calibration command.
signal CORRECT_FA_PARITY   : std_logic; -- Correct TEm function & address parity flag.
signal CORRECT_DATA_PARITY : std_logic; -- Correct TEM command data parity.
signal TEM_PARITY_ERR   : std_logic; -- Any TEM command parity error.
signal CORRECT_ADDR     : std_logic; -- TEM command matches GCRC wired address.
signal LATCHED_CORRECT_ADDR  : std_logic; -- Latched version of correct address.
signal TRIGGER        : std_logic; -- TEM commanded trigger, to GCRC_READOUT module.
signal TRIGGER_MODE     : std_logic; -- Single range readout = '0'.
signal REG_DATA_IN         : std_logic_vector (15 downto 0);--input parallel data
signal DAC_DATA_REG_SHIFT  : std_logic; -- Shift DAC data by one, to GCRC_DAC module.
signal DAC_WRITE      : std_logic; -- Onboard DAC is being written to.
signal ADC_DATA            : std_logic_vector (11 downto 0); -- Serial data from 12
ADc's
signal adc_read_begin   : std_logic; -- Begin ADC readout, from GCRC_ADC module
signal ADC_SAMPLE_CLK   : std_logic;  -- ADC clock from GCRC_ADC module
signal adc_read_done    : std_logic;   -- ADC has completed readout, to GCRC_ADC module
signal adc_sample       : std_logic; --  Begin ADC conversion, to GCRC_ADC module
signal RANGE_DATA_OUT0     : std_logic;   -- Combined ADC and GCFE data to the TEM
signal RANGE_DATA_OUT1     : std_logic;   -- Combined ADC and GCFE data to the TEM
signal WRITE_CMD_ERR    : std_logic; -- TEM Write command error, no action taken.
signal READ_CMD_ERR     : std_logic; -- TEM Read command error, not correct read.
signal RIGHT_FIRST      : std_logic; -- Left-right swapping control.
signal DAC_DATA_IN      : std_logic; -- Onboard DAC data.
signal ADC_NCS        : std_logic; -- ADC chip select, asserted low.
signal ADC_CLK        : std_logic; -- ADC clock signal.
signal DAC_NCS        : std_logic; -- Onboard DAC chip select, asserted low.
signal DAC_CLK        : std_logic; -- Onboard DAC clock.
signal DAC_DATA         : std_logic; -- Onboard DAC serial data signal.
signal RESET_CMD      : std_logic; -- Commanded reset from TEM.
signal DATALESS_CMD_ERR    : std_logic; -- TEM Dataless Cmd Not recognized. Added for
Ver 5.
signal GCFE_TRANS_LOWER    : std_logic; -- Ver 5.2.  Shifting lower GCFE data bits to
GCRC_Central module.


  Component GCRC_TEM is
  port (
    GCRC_DATA_OUT0 : in std_logic; -- Command reply data from GCRC_CENTRAL module.
    GCRC_DATA_OUT1 : in std_logic; -- Command reply data from GCRC_CENTRAL module.
    RANGE_DATA_OUT0  : in std_logic; -- Range readout data from GCRC_READOUT module.
    RANGE_DATA_OUT1  : in std_logic; -- Range readout data from GCRC_READOUT module.
    CAL_NCMD    : in std_logic; -- Command line to TEM.
    CAL_CLK     : in std_logic; -- System clock from TEM
```

```
    CAL_NRESET     : in std_logic; -- System RESET from TEM, asserted low.
    RIGHT_FIRST    : in std_logic;  -- Readout left-right switch control.
    RESET_CMD   : in std_logic; -- Commanded reset from TEM.
    CAL_NDATA0     : buffer std_logic; -- Output data line to TEM
    CAL_NDATA1     : buffer std_logic; -- Output data line to TEM
    TEM_CMD        : buffer std_logic; -- TEM command to GCRC_CENTRAL module.
    CLK         : buffer std_logic;-- Feedthrough clock.
    NRESET      : buffer std_logic -- Feedthrough RESET.
       );
  end component;


  Component GCRC_CENTRAL is
    port(
      NRESET            : in std_logic; -- Global RESET.
      CLK               : in std_logic; -- System 20 MHz clk.
      CONFIG_PARITY     : in std_logic; -- From GCRC register.
      WIRED_ADDR        : in std_logic_vector(3 downto 0);  -- Hardwired GCRC address
      TEM_CMD           : in std_logic; -- Serial command bits from the TEM.
      GCFE_DELAYED_RTN_DATA : in std_logic; -- Serial data returned from 1 of 12 GCFEs.
      GCFE_TRANS      : in std_logic;   -- GCFE is transmitting data to GCRC.
      GCFE_WRITE      : in std_logic; -- GCRC is writing to GCFE.
      STATUS_REPLY_ERR  : in std_logic; -- GCFE not reply, timeout error.
      WRITE_CMD_ERR   : in std_logic;  -- TEM command error, no GCRC action taken.
      STATUS_REG      : in std_logic_vector(4 downto 0); -- Register from GCRC_REG
module.
      RANGE_DELAY_REG : in std_logic_vector(5 downto 0); -- Register from GCRC_REG
module.
      ADC_SAMPLE_DELAY_REG : in std_logic_vector(5 downto 0); -- Register from GCRC_REG
module.
      ADC_CONV_TIME_REG : in std_logic_vector(7 downto 0); -- Register from GCRC_REG
module.
      DAC_DATA_REG    : in std_logic_vector(15 downto 0); -- Register from GCRC_REG
module.
      CONFIG_REG      : in std_logic_vector(2 downto 0); -- Register from GCRC_REG
module.
      GCFE_TRANS_LOWER  : in std_logic; -- Added Ver 5.2
      R_W             : buffer std_logic_vector(1 downto 0); -- Read/Write not, TEM
command bit
      FUNC         : buffer std_logic_vector(2 downto 0); -- TEM function bits.
      GCFE_CMD_BIT    : buffer std_logic; -- TEM GCFE command bit
      GCFE_ADDR       : buffer std_logic_vector(3 downto 0); -- Address in GCFE command.
      TEM_CMD_DATA_MSB  : buffer std_logic; -- TEM command data most significant bit.
      REG_DATA_IN     : buffer std_logic_vector(15 downto 0); -- TEM command data.
      GCRC_TRANS_RECV : buffer std_logic; -- GCRC is transmitting to the TEM
      GCRC_DATA_OUT0    : buffer std_logic; -- GCRC serial data out0
      GCRC_DATA_OUT1    : buffer std_logic; -- GCRC serial data out0
      STATUS_TRIG_ERR   : buffer std_logic; -- Bit to be placed in status register.
      STATUS_FA_ERR     : buffer std_logic; --  Bit to be placed in status register.
      STATUS_DATA_ERR   : buffer std_logic; -- Bit to be placed in status register.
      READ_CMD_ERR    : buffer std_logic; -- Correct parity from TEM, but Invalid
Command.
      CALIB_CMD       : buffer std_logic; -- Decoded calibration command.
      CORRECT_FA_PARITY : buffer std_logic; -- Correct TEm function & address parity
flag.
      CORRECT_DATA_PARITY : buffer std_logic; -- Correct TEM command data parity.
      TEM_PARITY_ERR    : buffer std_logic; -- Any TEM command parity error.
      CORRECT_ADDR    : buffer std_logic; -- TEM command matches GCRC wired address.
      LATCHED_CORRECT_ADDR : buffer std_logic; -- Latched version of correct address.
```

```
      TRIGGER       : buffer std_logic; -- TEM commanded trigger, to GCRC_READOUT
module.
      TRIGGER_MODE    : buffer std_logic; -- 1 or 4 range readout mode, to GCRC_READOUT.
      RESET_CMD       : buffer std_logic; -- Commanded reset from TEM.
      DATALESS_CMD_ERR  : buffer std_logic -- TEM Dataless Cmd not recognized.
         );
   end component;

   Component GCRC_REG is
   port(
      NRESET               : in std_logic; -- Chip wide RESET
      CLK            : in std_logic; -- 20MHz system clock
      GCFE_CMD_BIT     : in std_logic; -- Decoded TEM command is for the GCFE
      R_W            : in std_logic_vector(1 downto 0); -- Write to register = 0, Read
from register = 1
      FUNC        : in std_logic_vector(2 downto 0); -- TEM command function
      REG_DATA_IN         : in std_logic_vector(15 downto 0);--input parallel data
      CORRECT_FA_PARITY  : in std_logic; -- Function and address of TEM command
      CORRECT_DATA_PARITY: in std_logic;  -- Data of TEM command
      TEM_PARITY_ERR   : in std_logic; -- Any TEM command parity error.
      CORRECT_ADDR     : in std_logic; -- TEM command matches GCRC wired address.
      LATCHED_CORRECT_ADDR : in std_logic; -- Latched version of correct address.
      STATUS_TRIG_ERR  : in std_logic; -- Trigger parity error
      STATUS_FA_ERR      : in std_logic; -- Function address parity error
      STATUS_DATA_ERR  : in std_logic; -- Data parity error
      STATUS_REPLY_ERR : in std_logic; -- Error, GCFE not reply befor timeout.
      READ_CMD_ERR     : in std_logic; -- Correct parity from TEM, but Invalid Command.
      DAC_DATA_REG_SHIFT : in std_logic; -- Shift DAC data by one, to GCRC_DAC module.
      DAC_DATA_IN      : in std_logic; -- Data shifted out of onboard DAC.
      DATALESS_CMD_ERR : in std_logic; -- Dataless Cmd not recognized.
      DAC_WRITE      : buffer std_logic; -- Onboard DAC is being written to.
      DAC_DATA       : buffer std_logic; -- Most sig bit of reg, to onboard DAC.
      CONFIG_PARITY       : buffer std_logic; -- Parity mode
      WRITE_CMD_ERR    : buffer std_logic;-- TEM command error, no GCRC action taken.

      STATUS_REG       : buffer std_logic_vector(4 downto 0); -- Register to GCRC_CENTRAL
module.
      RANGE_DELAY_REG     : buffer std_logic_vector(5 downto 0); -- Peak hold settling
time.
      ADC_SAMPLE_DELAY_REG : buffer std_logic_vector(5 downto 0); -- Analog Mux settling
time.
      ADC_CONV_TIME_REG     : buffer std_logic_vector(7 downto 0);  -- ADC Conversion
time.
      DAC_DATA_REG     : buffer std_logic_vector(15 downto 0); -- Register to
GCRC_Central module.
      CONFIG_REG       : buffer std_logic_vector(2 downto 0)  -- Register to GCRC_Central
module.
      );
   end component;

   Component GCRC_GCFE
   port(
      NRESET           : in std_logic; -- Global RESET.
      CLK              : in std_logic; -- System 20 MHz clk.
      GCFE_ADDR        : in std_logic_vector(3 downto 0); -- GCFE address sent by TEM.
      GCFE_CMD_BIT     : in std_logic; -- TEM command is for GCFE.
      R_W            : in std_logic_vector(1 downto 0); -- Read/Write not
      FUNC        : in std_logic_vector(2 downto 0); --  TEM function bits
```

```
     TEM_CMD_DATA_MSB : in std_logic; -- TEM cmd data most significant bit, from
CENTRAL_CONTROL.
     CORRECT_FA_PARITY  : in std_logic; -- TEM command function and addrss has correct
parity.
     CORRECT_DATA_PARITY: in std_logic; -- TEM command data has correct parity.
     TEM_PARITY_ERR   : in std_logic; -- Latched value of any TEM command parity error.
     GCFE_DATA        : in std_logic_vector(11 downto 0); -- Serial data bits from 12
GCFEs.
     GCFE_CLK_ENV     : in std_logic; -- DAQ_CLK envelope from GCRC READOUT module.
     GCFE_RESET_ENV   : in std_logic;  -- RESET envelope from GCRC READOUT module.
     RIGHT_FIRST      : in std_logic; -- Right first, log end addressing switch.
     LATCHED_CORRECT_ADDR : in std_logic; -- Latched correct GCRC address.
     CORRECT_ADDR     : in std_logic; -- Correct GCRC Address.
     STATUS_REPLY_ERR : buffer std_logic; -- Error, GCFE not reply prior to timeout.
     DAQ_CLK          : buffer std_logic;-- Clk out to GCFEs.
     GCFE_DELAYED_RTN_DATA : buffer std_logic; -- Mux selected 1 of 12 GCFE data bits.
     GCFE_TRANS       : buffer std_logic; -- GCFE is transmitting data to GCRC.
     GCFE_WRITE       : buffer std_logic; -- GCRC is writing to GCFE.
     CMD              : buffer std_logic;-- Serial command and data to the GCFE.
     GRESET           : buffer std_logic; -- RESET command to GCFE as data envelope.
     GCFE_TRANS_LOWER : buffer std_logic; -- Added ver 5.2.
     GCFE_SAMPLED_DATA  : buffer std_logic_vector( 11 downto 0) -- Clocked GCFE_DATA.
     );
   end component;

   Component GCRC_READOUT is
     port (
      NRESET             : in std_logic;  --Chipwide RESET, asserted low
      CLK                : in std_logic; --20 MHz System Clock
      TRIGGER        : in std_logic; -- Readout command has been received
      TRIGGER_MODE     : in std_logic;  --- "0" is 1 range readout, "1" is 4 range
readout"
      GCFE_SAMPLED_DATA  : in std_logic_vector (11 downto 0); -- Clocked data from 12
GCFE's
      ADC_DATA          : in std_logic_vector (11 downto 0); -- Serial data from 12 ADc's
      ADC_READ_BEGIN   : in std_logic; -- Begin ADC readout, from GCRC_ADC module
      ADC_SAMPLE_CLK   : in std_logic;  -- ADC clock from GCRC_ADC module
      RANGE_DELAY_REG    : in std_logic_vector (5 downto 0);
      ADC_SAMPLE_DELAY_REG : in std_logic_vector (5 downto 0);
      CONFIG_PARITY    : in std_logic;    -- Register parity mode, '0' is odd parity.
      GCFE_CLK_ENV     : buffer std_logic; -- GCFE clock envelope
      START_ACQ      : buffer std_logic; -- GCFE readout state machine start
      GCFE_RESET_ENV   : buffer std_logic; -- GCFE command envelope
      ADC_READ_DONE    : buffer std_logic;-- ADC has completed readout, to GCRC_ADC
module
      ADC_SAMPLE       : buffer std_logic; --  Begin ADC conversion, to GCRC_ADC module
      ADC_CLK        : buffer std_logic;-- ADC Clock to ADC chips
      RANGE_DATA_OUT0    : buffer std_logic;-- Combined ADC and GCFE data to the TEM
      RANGE_DATA_OUT1    : buffer std_logic;-- Combined ADC and GCFE data to the TEM
      GCRC_IN_READOUT    : buffer std_logic  -- GCRC is performing range readout.
      );
   end component;

   Component GCRC_ADC
     port (
      NRESET             : in std_logic;     --Chipwide RESET, asserted low
      CLK              : in std_logic;     --20 MHz System Clock
      ADC_SAMPLE       : in std_logic;     -- Notification to begin ADC conversion, from
GCRC_READOUT
```

```
     ADC_READ_DONE    : in std_logic;    -- ADC data has been read, from GCRC_READOUT
     ADC_CONV_TIME_REG  : in std_logic_vector(7 downto 0);  -- ADC conversion time
stored in register
     ADC_READ_BEGIN   : buffer std_logic; -- Begin ADC readout, to GCRC_READOUT
     ADC_NCS             : buffer std_logic;  --Control and Shutdown line
     ADC_SAMPLE_CLK   : buffer std_logic  --Clock to the ADC, filtered by GCRC_READOUT
       );
  end component;

  Component GCRC_DAC
  port (
     NRESET        : in std_logic;  -- Global RESET.
     CLK           : in std_logic;  -- System 20 Mhz clk.
     DAC_WRITE     : in std_logic; -- DAC write request from GCRC_REG module.
     DAC_DATA_REG_SHIFT : buffer std_logic;  -- Command DAC data to be shifted, to
GCRC_REG.
     DAC_NCS            : buffer STD_LOGIC; -- Chip SELECT for the DAC
     DAC_CLK       : buffer std_logic; -- Clock to DAC
     GCRC_DAC_WRITE   : buffer std_logic -- GCRC is writing to the DAC, to GCRC_TRIG
module.
       );
  end component;

  Component GCRC_TRIG
    port (
     NRESET             : in std_logic;      --Chipwide RESET, asserted low
     CLK              : in std_logic;      --20 MHz System Clock
     LE_NDISC       : in STD_LOGIC; -- Wired 'OR' low energy trigger from GCFEs,
asserted low.
     HE_NDISC       : in STD_LOGIC;  -- Wired 'OR' high energy trigger from GCFEs,
asserted low.
     CAL_NTREQLE      : buffer STD_LOGIC; -- Low energy trigger request to the TEM,
asserted low.
     CAL_NTREQHE      : buffer STD_LOGIC -- High energy trigger request to the TEM,
asserted low.
        );
  end component;

  Component GCRC_CALIB
    port (
     NRESET             : in std_logic;   -- Chipwide RESET, asserted low.
     CLK              : in std_logic;   -- 20 MHz System Clock.
     CALIB_CMD      : in std_logic;   -- Decode Calibration command from TEM.
     CALIB_STRB       : buffer std_logic  -- Calibration line to the GCFEs.
        );
  end component;

  begin

  -- Signal renaming additions for GCRC Ver 4.
  -- Inputs.
  RIGHT_FIRST <= RIGHT_FIRST_C;
  DAC_DATA_IN <= DAC_DATA_IN_C;
  -- Outputs.
  ADC_NCS_C <= ADC_NCS;
  ADC_CLK_C <= ADC_CLK;
  DAC_NCS_C <= DAC_NCS;
  DAC_CLK_C <= DAC_CLK;
  DAC_DATA_C <= DAC_DATA;
```

```
-- GCRC Ver 4 has added "_C" suffix to below signals.
WIRED_ADDR(0) <= ADDR0_C;
WIRED_ADDR(1) <= ADDR1_C;
WIRED_ADDR(2) <= ADDR2_C;
WIRED_ADDR(3) <= ADDR3_C;

  ADC_DATA(0) <= ADC0_DATA_C;
  ADC_DATA(1) <= ADC1_DATA_C;
  ADC_DATA(2) <= ADC2_DATA_C;
  ADC_DATA(3) <= ADC3_DATA_C;
  ADC_DATA(4) <= ADC4_DATA_C;
  ADC_DATA(5) <= ADC5_DATA_C;
  ADC_DATA(6) <= ADC6_DATA_C;
  ADC_DATA(7) <= ADC7_DATA_C;
  ADC_DATA(8) <= ADC8_DATA_C;
  ADC_DATA(9) <= ADC9_DATA_C;
  ADC_DATA(10) <= ADC10_DATA_C;
  ADC_DATA(11) <= ADC11_DATA_C;

GCFE_DATA(0) <= DATA0;
GCFE_DATA(1) <= DATA1;
GCFE_DATA(2) <= DATA2;
GCFE_DATA(3) <= DATA3;
GCFE_DATA(4) <= DATA4;
GCFE_DATA(5) <= DATA5;
GCFE_DATA(6) <= DATA6;
GCFE_DATA(7) <= DATA7;
GCFE_DATA(8) <= DATA8;
GCFE_DATA(9) <= DATA9;
GCFE_DATA(10) <= DATA10;
GCFE_DATA(11) <= DATA11;



U_TEM : gcrc_tem
   port map
      (GCRC_DATA_OUT0 => GCRC_DATA_OUT0,
      GCRC_DATA_OUT1 => GCRC_DATA_OUT1,
      RANGE_DATA_OUT0 => RANGE_DATA_OUT0,
      RANGE_DATA_OUT1 => RANGE_DATA_OUT1,
      CAL_NCMD => CAL_NCMD,
      CAL_CLK => CAL_CLK,
      CAL_NRESET => CAL_NRESET,
      RIGHT_FIRST => RIGHT_FIRST,
      RESET_CMD => RESET_CMD,
      CAL_NDATA0 => CAL_NDATA0,
      CAL_NDATA1 => CAL_NDATA1,
      TEM_CMD => TEM_CMD,
      CLK => CLK,
      NRESET => NRESET
      );


U_CENTRAL : gcrc_central
   port map
      (NRESET => NRESET,
      CLK => CLK,
      CONFIG_PARITY => CONFIG_PARITY,
```

```
      WIRED_ADDR => WIRED_ADDR,
      TEM_CMD => TEM_CMD,
      GCFE_DELAYED_RTN_DATA => GCFE_DELAYED_RTN_DATA,
      GCFE_TRANS => GCFE_TRANS,
      GCFE_WRITE => GCFE_WRITE,
      STATUS_REPLY_ERR => STATUS_REPLY_ERR,
      WRITE_CMD_ERR => WRITE_CMD_ERR,
      STATUS_REG => STATUS_REG,
      RANGE_DELAY_REG => RANGE_DELAY_REG,
      ADC_SAMPLE_DELAY_REG => ADC_SAMPLE_DELAY_REG,
      ADC_CONV_TIME_REG => ADC_CONV_TIME_REG,
      DAC_DATA_REG => DAC_DATA_REG,
      CONFIG_REG => CONFIG_REG,
      GCFE_TRANS_LOWER => GCFE_TRANS_LOWER,
      R_W => R_W,
      FUNC => FUNC,
      GCFE_CMD_BIT => GCFE_CMD_BIT,
      GCFE_ADDR => GCFE_ADDR,
      TEM_CMD_DATA_MSB => TEM_CMD_DATA_MSB,
      REG_DATA_IN => REG_DATA_IN,
      GCRC_TRANS_RECV=> GCRC_TRANS_RECV,
      GCRC_DATA_OUT0 => GCRC_DATA_OUT0,
      GCRC_DATA_OUT1 => GCRC_DATA_OUT1,
      STATUS_TRIG_ERR => STATUS_TRIG_ERR,
      STATUS_FA_ERR => STATUS_FA_ERR,
      STATUS_DATA_ERR => STATUS_DATA_ERR,
      READ_CMD_ERR => READ_CMD_ERR,
      CALIB_CMD => CALIB_CMD,
      CORRECT_FA_PARITY => CORRECT_FA_PARITY,
      CORRECT_DATA_PARITY => CORRECT_DATA_PARITY,
      TEM_PARITY_ERR => TEM_PARITY_ERR,
      CORRECT_ADDR => CORRECT_ADDR,
      LATCHED_CORRECT_ADDR => LATCHED_CORRECT_ADDR,
      TRIGGER=> TRIGGER,
      TRIGGER_MODE => TRIGGER_MODE,
      RESET_CMD => RESET_CMD,
      DATALESS_CMD_ERR => DATALESS_CMD_ERR
       );


  U_REG : gcrc_reg
    port map
      (NRESET => NRESET,
      CLK => CLK,
      GCFE_CMD_BIT => GCFE_CMD_BIT,
      R_W => R_W,
      FUNC => FUNC,
      REG_DATA_IN => REG_DATA_IN,
      CORRECT_FA_PARITY => CORRECT_FA_PARITY,
      CORRECT_DATA_PARITY => CORRECT_DATA_PARITY,
      TEM_PARITY_ERR => TEM_PARITY_ERR,
      CORRECT_ADDR => CORRECT_ADDR,
      LATCHED_CORRECT_ADDR => LATCHED_CORRECT_ADDR,
      STATUS_TRIG_ERR => STATUS_TRIG_ERR,
      STATUS_FA_ERR => STATUS_FA_ERR,
      STATUS_DATA_ERR => STATUS_DATA_ERR,
      STATUS_REPLY_ERR => STATUS_REPLY_ERR,
      READ_CMD_ERR => READ_CMD_ERR,
      DAC_DATA_REG_SHIFT => DAC_DATA_REG_SHIFT,
```

```
      DAC_DATA_IN => DAC_DATA_IN,
      DATALESS_CMD_ERR => DATALESS_CMD_ERR,
      DAC_WRITE => DAC_WRITE,
      DAC_DATA => DAC_DATA,
      CONFIG_PARITY => CONFIG_PARITY,
      WRITE_CMD_ERR => WRITE_CMD_ERR,
      STATUS_REG => STATUS_REG,
      RANGE_DELAY_REG => RANGE_DELAY_REG,
      ADC_SAMPLE_DELAY_REG => ADC_SAMPLE_DELAY_REG,
      ADC_CONV_TIME_REG => ADC_CONV_TIME_REG,
      DAC_DATA_REG => DAC_DATA_REG,
      CONFIG_REG => CONFIG_REG
      );

  U_GCFE : gcrc_gcfe
    port map
      (NRESET => NRESET,
      CLK => CLK,
      GCFE_ADDR => GCFE_ADDR,
      GCFE_CMD_BIT => GCFE_CMD_BIT,
      R_W => R_W,
      FUNC => FUNC,
      TEM_CMD_DATA_MSB => TEM_CMD_DATA_MSB,
      CORRECT_FA_PARITY => CORRECT_FA_PARITY,
      CORRECT_DATA_PARITY => CORRECT_DATA_PARITY,
      TEM_PARITY_ERR => TEM_PARITY_ERR,
      GCFE_DATA => GCFE_DATA,
      GCFE_CLK_ENV => GCFE_CLK_ENV,
      GCFE_RESET_ENV => GCFE_RESET_ENV,
      RIGHT_FIRST => RIGHT_FIRST,
      LATCHED_CORRECT_ADDR => LATCHED_CORRECT_ADDR,
      CORRECT_ADDR => CORRECT_ADDR,
      STATUS_REPLY_ERR => STATUS_REPLY_ERR,
      DAQ_CLK => DAQ_CLK,
      GCFE_DELAYED_RTN_DATA => GCFE_DELAYED_RTN_DATA,
      GCFE_TRANS => GCFE_TRANS,
      GCFE_WRITE => GCFE_WRITE,
      CMD => CMD,
      GRESET => GRESET,
      GCFE_TRANS_LOWER => GCFE_TRANS_LOWER,
      GCFE_SAMPLED_DATA => GCFE_SAMPLED_DATA );


  U_READOUT : gcrc_readout
    port map
      (NRESET => NRESET,
      CLK => CLK,
      TRIGGER => TRIGGER,
      TRIGGER_MODE => TRIGGER_MODE,
      GCFE_SAMPLED_DATA => GCFE_SAMPLED_DATA,
      ADC_DATA => ADC_DATA,
      adc_read_begin => adc_read_begin,
      ADC_SAMPLE_CLK => ADC_SAMPLE_CLK,
      RANGE_DELAY_REG => RANGE_DELAY_REG,
      ADC_SAMPLE_DELAY_REG => ADC_SAMPLE_DELAY_REG,
      CONFIG_PARITY => CONFIG_PARITY,
      gcfe_clk_env => gcfe_clk_env,
      START_ACQ => START_ACQ,
      GCFE_RESET_ENV => GCFE_RESET_ENV,
```

```
        adc_read_done => adc_read_done,
        adc_sample => adc_sample,
        ADC_CLK => ADC_CLK,
        RANGE_DATA_OUT0 => RANGE_DATA_OUT0,
        RANGE_DATA_OUT1 => RANGE_DATA_OUT1,
        GCRC_IN_READOUT => GCRC_IN_READOUT );

   U_ADC : gcrc_adc
      port map
        (NRESET => NRESET,
        CLK => CLK,
        ADC_SAMPLE => ADC_SAMPLE,
        ADC_READ_DONE => ADC_READ_DONE,
        ADC_CONV_TIME_REG => ADC_CONV_TIME_REG,
        ADC_READ_BEGIN => ADC_READ_BEGIN,
        ADC_NCS => ADC_NCS,
        ADC_SAMPLE_CLK => ADC_SAMPLE_CLK );

   U_DAC : gcrc_dac
      port map
        (NRESET => NRESET,
        CLK => CLK,
        DAC_WRITE => DAC_WRITE,
        DAC_DATA_REG_SHIFT => DAC_DATA_REG_SHIFT,
        DAC_NCS => DAC_NCS,
        DAC_CLK => DAC_CLK,
        GCRC_DAC_WRITE => GCRC_DAC_WRITE );

   U_TRIG : gcrc_trig
      port map
      (
      NRESET => NRESET,
      CLK => CLK,
      CAL_NTREQLE => CAL_NTREQLE,
      CAL_NTREQHE => CAL_NTREQHE,
      LE_NDISC => LE_NDISC,
      HE_NDISC => HE_NDISC );


   U_CALIB : gcrc_calib
      port map
      (NRESET => NRESET,
       CLK => CLK,
      CALIB_CMD => CALIB_CMD,
      CALIB_STRB => CALIB_STRB );


end top_level;
```

5.2 GCRC_CENTRAL.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--Filename: GCRC_CENTRAL.vhd
--Created by Jim Ampe / Michael Condon, Naval Research Lab
--Revision: 1.0 (11-12-01)
--          First version.
--     1.0 Rev A (11-14-01) swapped GCRC and GCFE TEM command address decoding.
--     1.0 Rev C (11-16-01) Incorporated register readout to reduce flip flops.
--       Also ignore TEM trigger if trigger parity is incorrect.  Also corrected
--       Status Error Bit to TEM being cleared in Idle state, not after transmission
--       of bit to TEM.  Also not look for TEM data for a Calib command.
--     2.0 (1-2-02) Added code to correct read command to incorrect
--       address trying to act on the read command. Also added Read_Cmd_Err signal,
--       latched by Status Register.  Also modified code for TEM GCFE write commands
--       to function correctly spaced closer together.
--     2.0 Rev B (1-16-02) Added code to save TEM command for GCFE timeout.
--     4.0 Rev A (3-29-02) Added code to main state machine looking for
Latched_Correct_Addr.
--     5.0  ( 11-13-02) Added code for commanded reset function.
--       Changed addressing to two bit read/write field.
--       Lowest 3 function/address bits unchanged.
--       8 MS bits of GCRC Configuration register read back GCRC version number.
--       Added status register error bit control for incorrect TEM dataless commands.
--       Modified Last Command Error Register definition:
--          MSB = 0: Err Register other bits are Trigger reg and 12 FA register bits.
--          MSB = 1: Err Register other bits are 15 FA register bits.
--     5.1 (12-09-02) Changed GCRC readback version from '50' to '51'.
--     5.2 No Change.
--     5.3 (12-12-02) Changed GCRC version to '53'.  Also modified FA_Register
manipulation
--          to accept late coming GCFE reply data to the end of the FA_Register.
--
--
--Function: Top level design file for GCRC.
--
--
-----------------------------------------------------------------------------------------

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;


entity GCRC_CENTRAL is
    port(
      NRESET            : in std_logic; -- Global RESET.
      CLK               : in std_logic; -- System 20 MHz clk.
      CONFIG_PARITY     : in std_logic; -- From GCRC register.
      WIRED_ADDR        : in std_logic_vector(3 downto 0);  -- Hardwired GCRC address
      TEM_CMD           : in std_logic; -- Serial command bits from the TEM.
      GCFE_DELAYED_RTN_DATA : in std_logic; -- Serial data returned from 1 of 12 GCFEs.
      GCFE_TRANS      : in std_logic;   -- GCFE is transmitting data to GCRC.
      GCFE_WRITE      : in std_logic; -- GCRC is writing to GCFE.
      STATUS_REPLY_ERR  : in std_logic; -- GCFE not reply, timeout error.
      WRITE_CMD_ERR   : in std_logic;  -- TEM write command error, no GCRC action taken.
      STATUS_REG      : in std_logic_vector(4 downto 0); -- Register from GCRC_REG
module.
```

```
      RANGE_DELAY_REG : in std_logic_vector(5 downto 0); -- Register from GCRC_REG
module.
      ADC_SAMPLE_DELAY_REG : in std_logic_vector(5 downto 0); -- Register from GCRC_REG
module.
      ADC_CONV_TIME_REG : in std_logic_vector(7 downto 0); -- Register from GCRC_REG
module.
      DAC_DATA_REG    : in std_logic_vector(15 downto 0); -- Register from GCRC_REG
module.
      CONFIG_REG      : in std_logic_vector(2 downto 0); -- Register from GCRC_REG
module.
      GCFE_TRANS_LOWER  : in std_logic; -- Shifting lower GCFE return data bits.
      R_W             : buffer std_logic_vector(1 downto 0); -- Read/Write not, TEM
command bit
      FUNC         : buffer std_logic_vector(2 downto 0); -- TEM function bits.
      GCFE_CMD_BIT    : buffer std_logic; -- TEM GCFE command bit
      GCFE_ADDR       : buffer std_logic_vector(3 downto 0); -- Address in GCFE command.
      TEM_CMD_DATA_MSB  : buffer std_logic; -- TEM command data most significant bit.
      REG_DATA_IN     : buffer std_logic_vector(15 downto 0); -- TEM command data.
      GCRC_TRANS_RECV : buffer std_logic; -- GCRC is transmitting to the TEM
      GCRC_DATA_OUT0    : buffer std_logic; -- GCRC serial data out0
      GCRC_DATA_OUT1    : buffer std_logic; -- GCRC serial data out0
      STATUS_TRIG_ERR   : buffer std_logic; -- Bit to be placed in status register.
      STATUS_FA_ERR     : buffer std_logic; --  Bit to be placed in status register.
      STATUS_DATA_ERR   : buffer std_logic; -- Bit to be placed in status register.
      READ_CMD_ERR    : buffer std_logic; -- Correct parity from TEM, but Invalid
Command.
      CALIB_CMD       : buffer std_logic; -- Decoded calibration command.
      CORRECT_FA_PARITY : buffer std_logic; -- Correct TEM function & address parity
flag.
      CORRECT_DATA_PARITY : buffer std_logic; -- Correct TEM command data parity.
      TEM_PARITY_ERR    : buffer std_logic; -- Any TEM command parity error.
      CORRECT_ADDR    : buffer std_logic; -- TEM command matches GCRC wired address.
      LATCHED_CORRECT_ADDR : buffer std_logic; -- Latched version of correct address.
      TRIGGER       : buffer std_logic; -- TEM commanded trigger, to GCRC_READOUT
module.
      TRIGGER_MODE    : buffer std_logic; -- 1 or 4 range readout mode, to GCRC_READOUT.
      RESET_CMD       : buffer std_logic;  -- TEM commanded reset to GCRC_TEM module.
      DATALESS_CMD_ERR  : buffer std_logic -- Incorrect dataless command from TEM.
         );
end GCRC_CENTRAL;



architecture GCRC_CENTRAL of GCRC_CENTRAL is
--  Ver 5 added GCRC version Constant, to changed for each ASIC manufactured.  Initial
value '50' for '5.0'
--  Ver 5.1  changed to '51' for version 5.1
--  Ver 5.3  changed to '53' for version 5.3.
Constant GCRC_VERSION     : std_logic_vector (7 downto 0) := "01010011";

Constant FA_Count              : std_logic_vector (4 downto 0):= "01101";
Constant Data_Count          : std_logic_vector (4 downto 0):= "01111";
Constant FA_Out_Count        : std_logic_vector (4 downto 0):= "01101";
Constant Data_Out_Count      : std_logic_vector (4 downto 0):= "01111";
Constant WAIT_TIMEOUT     : std_logic_vector (4 downto 0) := "11111";
Constant IDLE           : std_logic_vector (3 downto 0):= "0000";
Constant WAIT_START       : std_logic_vector (3 downto 0) := "0001";
Constant TEM_TRIG_INPUT      : std_logic_vector (3 downto 0) := "0010";
Constant TEM_TRIG_CHECK      : std_logic_vector (3 downto 0) := "0011";
```

```
Constant TEM_FA_INPUT      : std_logic_vector (3 downto 0) := "0100";
Constant TEM_FA_CHECK      : std_logic_vector (3 downto 0) := "0101";
Constant TEM_DATA_INPUT      : std_logic_vector (3 downto 0) := "0110";
Constant TEM_DATA_CHECK      : std_logic_vector (3 downto 0) := "0111";
Constant DELAY_TO_IDLE       : std_logic_vector (3 downto 0) := "1000";
Constant GCRC_WAIT_TRANS     : std_logic_vector (3 downto 0) := "1001";
Constant GCRC_TRANS_START    : std_logic_vector (3 downto 0) := "1010";
Constant GCRC_TRANS_FA       : std_logic_vector (3 downto 0) := "1011";
Constant GCRC_TRANS_FA_PARITY  : std_logic_vector (3 downto 0) := "1100";
Constant GCRC_TRANS_DATA     : std_logic_vector (3 downto 0) := "1101";
Constant GCRC_TRANS_ERR      : std_logic_vector (3 downto 0) := "1110";
Constant GCRC_TRANS_PARITY   : std_logic_vector (3 downto 0) := "1111";


signal GCRC_ADDR                    : std_logic_vector(3 downto 0);
signal TRIG_REGISTER              : std_logic_vector(2 downto 0);
signal FA_REGISTER                : std_logic_vector(14 downto 0);
signal DATA_REGISTER              : std_logic_vector(16 downto 0);
signal COMM_ERR_REG          : std_logic_vector(15 downto 0);
signal COUNTER                 : std_logic_vector(4 downto 0);
signal COUNTER_EN            : std_logic;
signal CURRENT_STATE, NEXT_STATE  : std_logic_vector(3 downto 0);
signal CORRECT_TRIG_PARITY          : std_logic;
signal TRANS_TRIG_PARITY       : std_logic; -- Transmitted parity bit
signal TRANS_FA_PARITY         : std_logic;
signal TRANS_DATA_PARITY       : std_logic;
signal COMP_PARITY_TRIG_IN     : std_logic; -- Computed parity value
signal COMP_PARITY_FA_IN       : std_logic;
signal COMP_PARITY_DATA_IN     : std_logic;
signal LATCHED_STATUS_ERR      : std_logic; -- Register for TEM error bit transmission
signal PARITY_TRIG           : std_logic;
signal PARITY_FA_IN          : std_logic;
signal PARITY_DATA_IN        : std_logic;
signal PARITY_RESET_FA         : std_logic;
signal PARITY_RESET_DATA       : std_logic;
signal LATCHED_PARITY_FA_IN       : std_logic;
signal LATCHED_PARITY_DATA_IN     : std_logic;
signal XOR_OUT             : std_logic;
signal LATCHED_XOR_OUT         : std_logic;
signal PARITY_OUT            : std_logic;
signal PARITY_OUT_RESET        : std_logic;
signal IGNORE_PARITY         : std_logic;
signal LATCHED_IGNORE_PARITY   : std_logic;


begin

   -- Ver 5 changed FUNC from 4 bits to 3 bits.  Changed R_W from 1 bit to 2 bits.
  FUNC <= FA_REGISTER(3 downto 1);
   R_W <= FA_REGISTER(5 downto 4);
  GCRC_ADDR <= FA_REGISTER(14 downto 11);
  GCFE_ADDR <= FA_REGISTER(9 downto 6);
  GCFE_CMD_BIT <= FA_REGISTER(10);  --if '1' then GCFE function if '0' then GCRC
function
   REG_DATA_IN <= DATA_REGISTER(16 downto 1);
   TRANS_Trig_Parity <= TRIG_REGISTER(0);
   TRANS_FA_Parity <= FA_REGISTER(0);
   TRANS_DATA_PARITY <= DATA_REGISTER(0);
   TEM_CMD_DATA_MSB <= DATA_REGISTER(16);
```

```
   -- State machine designed so that all GCFEs per calorimeter side go through states
   --     concurrently as much as possible.  Only check GCRC address for read command.
   -- Also, if any parity error occurs, assume command is long command sending data,
   --   flow through state machine reading all bits but not processing command.
   -- If trig parity occurs, assume 1 range readout, but also flow through state
machine.
   FSM_STATE: process (CURRENT_STATE, COUNTER, TRIG_REGISTER, CORRECT_ADDR,
   LATCHED_CORRECT_ADDR, CORRECT_TRIG_PARITY, CORRECT_FA_PARITY, CORRECT_DATA_PARITY,
   TEM_PARITY_ERR, GCFE_CMD_BIT, GCFE_TRANS, R_W, STATUS_REPLY_ERR, TEM_CMD, CALIB_CMD)
   begin
        case CURRENT_STATE is
          when IDLE =>  -- State 0
          COUNTER_EN <= '0';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
          NEXT_STATE <= WAIT_START;

          -- Wait_start waits for the TEM command start bit.
          when WAIT_START =>  -- State 1
          COUNTER_EN <= '0';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
          if ( TEM_CMD = '1' ) then -- This is the start bit
            NEXT_STATE <= TEM_TRIG_INPUT ;
          else
            NEXT_STATE <= WAIT_START;
          end if;

          -- TEM trig input waits for the TEM command start bit to get to
          --   the end of the register.
          when TEM_TRIG_INPUT =>  -- State 2
          COUNTER_EN <= '0';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
          if ( TRIG_REGISTER(2) = '1' ) then -- This is the start bit
            NEXT_STATE <= TEM_TRIG_CHECK ;
          else
            NEXT_STATE <= TEM_TRIG_INPUT;
          end if;

          -- If correct trigger parity and TEM trigger command, assert Trigger flags
          --     to GCRC_READOUT module and go back to idle.
          -- If incorrect trigger parity, flow through command decoding to get past all
          --     possible incoming data bits.
          when TEM_TRIG_CHECK =>  -- State 3
          COUNTER_EN <= '0';
          IGNORE_PARITY <= '0';
          if ( CORRECT_TRIG_PARITY = '1' ) then
            if ( TRIG_REGISTER(2) = '1') then
              NEXT_STATE <= IDLE ;
              TRIGGER <= '1' ;
              if( TRIG_REGISTER(1) = '1') then
                TRIGGER_MODE <= '1' ; -- 4 range readout
              else
```

```
                TRIGGER_MODE <= '0' ; -- 1 range readout
              end if;
            else
              TRIGGER <= '0';  -- No trigger, data command.
              TRIGGER_MODE <= '0';
              NEXT_STATE <= TEM_FA_INPUT;
            end if;
          else -- Incorrect trigger parity.
            -- Flow through remaining states to get past possible incoming data bits.
            TRIGGER <= '0' ;
            TRIGGER_MODE <= '0' ;
            NEXT_STATE <= TEM_FA_INPUT ;
          end if;

          when TEM_FA_INPUT =>  -- State 4
          COUNTER_EN <= '1';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
          if ( COUNTER = FA_COUNT ) then
            NEXT_STATE <= TEM_FA_CHECK;
          else
            NEXT_STATE <= TEM_FA_INPUT;
          end if;

          when TEM_FA_CHECK =>  -- State 5
          COUNTER_EN <= '0';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          -- If (parity correct and TEM read command and Correct Address)
          --   go to WAIT_TRANS.
          -- If (parity correct and TEM read command but not Correct Address),
          --   flow through TEM_DATA_INPUT to ignore possible TEM command
          --   data bits.  Set flag Ignore_Parity so as not to not set status
          --   error if TEM command data parity is incorrect.
          -- If TEM Parity Err or FA parity incorrect then assume the longest command
          -- to follow, needing to flow through TEM_DATA_READ so as not to confuse
          -- following bits with trigger command.
          -- Ver 5, code re-worked.
          if ( CORRECT_FA_PARITY = '1' and TEM_PARITY_ERR = '0' and CORRECT_ADDR = '0')
then
            IGNORE_PARITY <= '1';  -- If no errors and incorrect address, ignore future
parity errors.
          else
            IGNORE_PARITY <= '0';
          end if;
            if ( CORRECT_FA_PARITY = '1' and TEM_PARITY_ERR = '0') then
            if ( R_W = "00" ) then   -- If dataless command, go back to idle regardless
of address.
              NEXT_STATE <= DELAY_TO_IDLE;
            elsif ( R_W = "10" and CORRECT_ADDR = '1') then -- If reading and address
correct
              NEXT_STATE <= GCRC_WAIT_TRANS;
            else
              NEXT_STATE <= TEM_DATA_INPUT;  -- If writing or incorrect address, read
through TEM data.
            end if;
          else
```

```
            NEXT_STATE <= TEM_DATA_INPUT; -- If any errors read through TEM command
data
          end if;


          when TEM_DATA_INPUT =>  -- State 6
          COUNTER_EN <= '1';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
          if ( COUNTER = DATA_COUNT ) then
            NEXT_STATE <= TEM_DATA_CHECK;
          else
            NEXT_STATE <= TEM_DATA_INPUT;
          end if;

          --  If a parity had occurred anywhere in the TEM command,
          --      go back to idle from here.
          --  Go to transmit state if transmitting data to GCFE or
          --      reading GCFE register.
          --  Else if writing to the GCFE go back to idle.
          -- Ver 4, Added check for Latched_Correct_Addr.
          when TEM_DATA_CHECK =>  -- State 7
          COUNTER_EN <= '0';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
            if (CORRECT_DATA_PARITY = '1' and TEM_PARITY_ERR = '0') then
            if ( GCFE_CMD_BIT = '1' and R_W = "10" and LATCHED_CORRECT_ADDR = '1') then

              NEXT_STATE <= GCRC_WAIT_TRANS ;
            else
              NEXT_STATE <= DELAY_TO_IDLE ; -- Writing to GCRC, go back to idle.
            end if;
          else -- Parity error, go back to idle
            NEXT_STATE <= DELAY_TO_IDLE ;
          end if;

          -- Added for Ver 2.
          -- Delays one clock cycle so that Trig Reg can be saved in Cmd Err Reg
          --   if Write Command Error is asserted.
          when DELAY_TO_IDLE =>  -- State 8
          COUNTER_EN <= '0';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
          NEXT_STATE <= IDLE ;

          -- Wait for data to begin transmitting back to the TEM.
          -- If waiting for GCFE data, have timeout counter so do not get stuck here.
          when GCRC_WAIT_TRANS =>  -- State 9
          COUNTER_EN <= '1';
          TRIGGER <= '0';
          TRIGGER_MODE <= '0';
          IGNORE_PARITY <= '0';
          -- Read from GCRC or Reading from GCFE and have started receiving GCFE data.
          if ( GCFE_CMD_BIT = '0' or GCFE_TRANS = '1' ) then
            NEXT_STATE <= GCRC_TRANS_START ;
          -- No reply from GCFE, or GCRC_GCFE module
```

```
elsif ( STATUS_REPLY_ERR = '1' or COUNTER = WAIT_TIMEOUT ) then
   NEXT_STATE <= GCRC_TRANS_START ;
else
   NEXT_STATE <= GCRC_WAIT_TRANS;
end if;

when GCRC_TRANS_START =>  -- State 10
COUNTER_EN <= '0';
TRIGGER <= '0';
TRIGGER_MODE <= '0';
IGNORE_PARITY <= '0';
NEXT_STATE <= GCRC_TRANS_FA ;

-- Transmit Address and Function bits to TEM.
when GCRC_TRANS_FA =>  -- State 11
COUNTER_EN <= '1';
TRIGGER <= '0';
TRIGGER_MODE <= '0';
IGNORE_PARITY <= '0';
if ( COUNTER = FA_OUT_COUNT ) then
   NEXT_STATE <= GCRC_TRANS_FA_PARITY ;
else
   NEXT_STATE <= GCRC_TRANS_FA;
end if;

when GCRC_TRANS_FA_PARITY =>  -- State 12
COUNTER_EN <= '0';
TRIGGER <= '0';
TRIGGER_MODE <= '0';
IGNORE_PARITY <= '0';
NEXT_STATE <= GCRC_TRANS_DATA ;

when GCRC_TRANS_DATA =>    -- State 13
COUNTER_EN <= '1';
TRIGGER <= '0';
TRIGGER_MODE <= '0';
IGNORE_PARITY <= '0';
if ( COUNTER = DATA_OUT_COUNT ) then
   NEXT_STATE <= GCRC_TRANS_ERR ;
else
   NEXT_STATE <= GCRC_TRANS_DATA;
end if;

-- Transmit error bit to TEM.
when GCRC_TRANS_ERR =>    -- State 14
COUNTER_EN <= '0';
TRIGGER <= '0';
TRIGGER_MODE <= '0';
IGNORE_PARITY <= '0';
NEXT_STATE <= GCRC_TRANS_PARITY;

when GCRC_TRANS_PARITY =>  -- State 15
COUNTER_EN <= '0';
TRIGGER <= '0';
TRIGGER_MODE <= '0';
IGNORE_PARITY <= '0';
NEXT_STATE <= IDLE;

when others =>
```

```
        COUNTER_EN <= '0';
        TRIGGER <= '0';
        TRIGGER_MODE <= '0';
        IGNORE_PARITY <= '0';
        NEXT_STATE <= IDLE;

      end case;
  end process;

    FSM_CHANGE: process (NRESET, CLK)is
    begin
        if (NRESET = '0')then
            CURRENT_STATE <= IDLE;
        elsif(CLK'event and CLK = '1')then
       CURRENT_STATE <= NEXT_STATE ;
     end if;
  end process;


  -- Data Register used for both loading in TEM command data and for
  --     shifting out register read data.
  -- For Ver 2 added Read_Cmd_Error.
    DATA_LOAD: process (CLK, NRESET)
    begin
        if (NRESET = '0')then
            DATA_REGISTER <= (others => '0');
       READ_CMD_ERR <= '0';
        elsif(CLK'event and CLK = '1')then
       case (CURRENT_STATE) is
       when IDLE =>
         READ_CMD_ERR <= '0';
         if( GCFE_WRITE = '1' ) then -- Shift out MSB to GCRC_GCFE module.
                DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
                DATA_REGISTER(0) <= '0' ;
         else
            DATA_REGISTER <= DATA_REGISTER;
         end if;
       when WAIT_START =>
         READ_CMD_ERR <= '0';
         if( GCFE_WRITE = '1' ) then -- Shift out MSB to GCRC_GCFE module.
                DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
                DATA_REGISTER(0) <= '0' ;
         else
            DATA_REGISTER <= DATA_REGISTER;
         end if;
       when TEM_TRIG_INPUT =>
         READ_CMD_ERR <= '0';
         if( GCFE_WRITE = '1' ) then -- Shift out MSB to GCRC_GCFE module.
                DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
                DATA_REGISTER(0) <= '0' ;
         else
            DATA_REGISTER <= (others => '0');
         end if;

       -- Added for Ver 2..
       when TEM_TRIG_CHECK =>
         READ_CMD_ERR <= '0';
         if( GCFE_WRITE = '1' ) then -- Shift out MSB to GCRC_GCFE module.
                DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
```

```
                DATA_REGISTER(0) <= '0' ;
          else
            DATA_REGISTER <= (others => '0');
          end if;
        -- Added for Ver 2.
        when TEM_FA_INPUT =>
          READ_CMD_ERR <= '0';
          if( GCFE_WRITE = '1' ) then -- Shift out MSB to GCRC_GCFE module.
                DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
                DATA_REGISTER(0) <= '0' ;
          else
            DATA_REGISTER <= (others => '0');
          end if;

        when TEM_FA_CHECK =>
          if ( TEM_PARITY_ERR = '0' and CORRECT_FA_PARITY = '1' and R_W = "10" and
GCFE_CMD_BIT = '0' and
          CORRECT_ADDR = '1' ) then
          -- Register read command, transfer register to Data_Register.
            case FUNC is
              when "000" =>     -- Status Register, 5 bits
                DATA_REGISTER(5 downto 1) <= STATUS_REG ;
                DATA_REGISTER(16 downto 6) <= (others => '0');
                DATA_REGISTER(0) <= '0';
                READ_CMD_ERR <= '0';
                when  "001" =>     -- Last error command register, 16 bits
                     DATA_REGISTER(16 downto 1) <= Comm_Err_REG ;
                DATA_REGISTER(0) <= '0';
                READ_CMD_ERR <= '0';
                  when "011" =>  -- Peak hold to GCFE range decision delay, 6 bits
                DATA_REGISTER(6 downto 1) <= RANGE_DELAY_REG ;
                DATA_REGISTER(16 downto 7) <= (others => '0');
                DATA_REGISTER(0) <= '0';
                READ_CMD_ERR <= '0';
                  when "100" =>   -- GCFE Range decision to ADC sample delay, 6 bits
                DATA_REGISTER(6 downto 1) <= ADC_SAMPLE_DELAY_REG ;
                DATA_REGISTER(16 downto 7) <= (others => '0');
                DATA_REGISTER(0) <= '0';
                READ_CMD_ERR <= '0';
                when "101" =>   -- ADC conversion time delay, 8 bits
                DATA_REGISTER(8 downto 1) <= ADC_CONV_TIME_REG ;
                DATA_REGISTER(16 downto 9) <= (others => '0');
                DATA_REGISTER(0) <= '0';
                READ_CMD_ERR <= '0';
                when "110" =>     -- Onboard digital to analog converter, 16 bits
                       DATA_REGISTER(16 downto 1) <= DAC_DATA_REG(15 downto 0) ;
                DATA_REGISTER(0) <= '0';
                READ_CMD_ERR <= '0';
              when "111" =>     -- GCRC configuration register, 3 bits + GCRC Version
                DATA_REGISTER(3 downto 1) <= CONFIG_REG ;
                DATA_REGISTER(16 downto 9) <= GCRC_VERSION(7 downto 0);
                DATA_REGISTER(8 downto 4) <= (others => '0');
                DATA_REGISTER(0) <= '0';
                READ_CMD_ERR <= '0';
                when others =>
                DATA_REGISTER <= (others => '0');
                READ_CMD_ERR <= '1';
            end case;
```

```
        else -- Else not register read, begin shifting in TEM command data.
               DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
               DATA_REGISTER(0) <= TEM_CMD ;
          READ_CMD_ERR <= '0';
        end if; -- End if Current_State is Tem_FA_Check

      -- Shift in TEM command data.
      when TEM_DATA_INPUT =>
        READ_CMD_ERR <= '0';
              DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
              DATA_REGISTER(0) <= TEM_CMD ;

      -- Shift possible register data out.
      when GCRC_TRANS_FA =>
        READ_CMD_ERR <= '0';
        DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
              DATA_REGISTER(0) <= '0' ;
      when GCRC_TRANS_FA_PARITY =>
        READ_CMD_ERR <= '0';
        DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
              DATA_REGISTER(0) <= '0' ;
      when GCRC_TRANS_DATA =>
        READ_CMD_ERR <= '0';
        DATA_REGISTER(16 downto 1) <= DATA_REGISTER(15 downto 0) ;
              DATA_REGISTER(0) <= '0' ;

      when others =>
        DATA_REGISTER <=  DATA_REGISTER;
        READ_CMD_ERR <= '0';
          end case;
       end if;
    end process;

  TRIG_REG_LOAD: process (NRESET, CLK)
    begin
       if (NRESET = '0')then
          TRIG_REGISTER <= (others => '0');
       elsif(CLK'event and CLK = '1')then
      if ( CURRENT_STATE = IDLE ) then
        TRIG_REGISTER <= (others => '0');
          elsif ( CURRENT_STATE = WAIT_START or CURRENT_STATE = TEM_TRIG_INPUT )then
              TRIG_REGISTER( 2 downto 1) <= TRIG_REGISTER(1 downto 0) ;
              TRIG_REGISTER(0) <= TEM_CMD ;
      else
              TRIG_REGISTER <= TRIG_REGISTER;
          end if;
       end if;
    end process;


  -- Function and address register is used for both reading and writing.
  -- Transmitting to the TEM draws in data bits into LSB of FA register.
  FA_REG_READ_WR: process (CLK, NRESET)
    begin
       if (NRESET = '0')then
        FA_REGISTER <= (others => '0');
       elsif(CLK'event and CLK = '1')then
      if ( CURRENT_STATE = WAIT_START ) then
        FA_REGISTER <= (others => '0');
```

```
            elsif( CURRENT_STATE = TEM_TRIG_CHECK or CURRENT_STATE = TEM_FA_INPUT )then
          -- Shift TEM data into FA Reg
                FA_REGISTER(14 downto 1) <= FA_REGISTER(13 downto 0);
                FA_REGISTER(0) <= TEM_CMD ;
        elsif( CURRENT_STATE = GCRC_TRANS_FA or CURRENT_STATE = GCRC_TRANS_FA_PARITY
          or CURRENT_STATE = GCRC_TRANS_DATA )then
          -- Shift FA reg data out to TEM
          -- Changed ver 5.3  Insert lower return data if indicated by GCFE_Trans_Lower.
          if ( GCFE_TRANS_LOWER = '1' ) then
            FA_REGISTER(14) <= GCFE_DELAYED_RTN_DATA;
          else
            FA_REGISTER(14) <= FA_REGISTER(13);
          end if;
          FA_REGISTER(13 downto 1) <= FA_REGISTER(12 downto 0);
          -- Shift return data into vacated FA Reg slot
          if ( GCFE_TRANS = '1' ) then
            FA_REGISTER(0) <= GCFE_DELAYED_RTN_DATA; -- GCFE data
          else
            FA_REGISTER(0) <= DATA_REGISTER(16); -- Register data
          end if;
        else
          FA_REGISTER <= FA_REGISTER;
            end if;
        end if;
    end process;

  -- All purpose counter.
    Count: process (CLK, NRESET)
    begin
        if (NRESET = '0') then
            Counter <= (others => '0');
        elsif(CLK'event and CLK = '1')then
            if ( COUNTER_EN = '1') then
                Counter <= Counter + 1 ;
            else
                Counter <= (others => '0') ;
            end if;
        end if;
    end process;

  -- Dataless TEM command decoding.
  -- Added Ver 5.  Decode TEM command for reset.  Also added Dataless Command Error.
  DATALESS_DECODE: process(NRESET, CLK)
  begin
    if (NRESET = '0') then
      CALIB_CMD <= '0';
      RESET_CMD <= '0';
      DATALESS_CMD_ERR <= '0';
    elsif ( CLK'event and CLK = '1') then
      if (TEM_PARITY_ERR = '0' and CORRECT_FA_PARITY = '1' and R_W = "00" and
        GCFE_CMD_BIT = '0' and CORRECT_ADDR = '1') then
        case FUNC is
          when "011" =>
          CALIB_CMD <= '1';
          RESET_CMD <= '0';
          DATALESS_CMD_ERR <= '0';
          when "001" =>
          RESET_CMD <= '1';
          CALIB_CMD <= '0';
```

```
            DATALESS_CMD_ERR <= '0';
            when others =>
            CALIB_CMD <= '0';
            RESET_CMD <= '0';
            DATALESS_CMD_ERR <= '1';
          end case;
        else
          CALIB_CMD <= '0';
          RESET_CMD <= '0';
          DATALESS_CMD_ERR <= '0';
        end if;
      end if;
    end process;

    -- Matches TEM commanded GCRC address with the hard wired GCRC pins.
    -- Correct Address used for determining Correct Function Address parity.
    ADDRESS: process (NRESET, COUNTER, GCRC_ADDR, WIRED_ADDR, R_W, CURRENT_STATE)
    begin
        if (NRESET = '0') then
            CORRECT_ADDR <= '0';
        elsif (CURRENT_STATE = TEM_FA_CHECK ) then
        if ( GCRC_ADDR = WIRED_ADDR or (GCRC_ADDR = "1111" and R_W /= "10"))then
            CORRECT_ADDR <= '1' ;
        else
          CORRECT_ADDR <= '0';
        end if;
      else
          CORRECT_ADDR <= '0' ;
        end if;
    end process;

    -- Latches correct address value for determining Correct Data Parity.
    LATCHED_ADDRESS: process (NRESET, CLK)
    begin
        if (NRESET = '0') then
            LATCHED_CORRECT_ADDR <= '0';
      elsif ( CLK'event and CLK = '1') then
          if (CORRECT_ADDR = '1') then
            LATCHED_CORRECT_ADDR <= '1' ;
        elsif ( CURRENT_STATE = IDLE or CURRENT_STATE = TEM_DATA_CHECK ) then
          LATCHED_CORRECT_ADDR <= '0' ;
        else
            LATCHED_CORRECT_ADDR <= LATCHED_CORRECT_ADDR;
        end if;
        end if;
    end process;


    --   Latch if status error occurred, for Error Bit transmission to TEM.
    --   Clear latched bit following Error Bit transmission to the TEM.
    -- For Ver 2 added check on Read and Write Command error.
    LATCHED_STATUS_ERROR: process (NRESET, CLK)
    begin
      if (NRESET = '0') then
        LATCHED_STATUS_ERR <= '0';
      elsif(CLK'event and CLK = '1')then
        if ( CURRENT_STATE = GCRC_TRANS_PARITY ) then
          LATCHED_STATUS_ERR <= '0';
        elsif ( STATUS_REPLY_ERR = '1' or  STATUS_TRIG_ERR = '1' or
```

```
          STATUS_FA_ERR = '1' or STATUS_DATA_ERR = '1' or READ_CMD_ERR = '1' or
          WRITE_CMD_ERR = '1' or DATALESS_CMD_ERR = '1') then
          LATCHED_STATUS_ERR <= '1';
        else
          LATCHED_STATUS_ERR <= LATCHED_STATUS_ERR;
        end if;
      end if;
    end process;

    --   Latch if read command occured and GCRC address not correct.
    --  Clear if in IDLE state.
    LATCHED_IGNORE_PAR: process (NRESET, CLK)
    begin
      if (NRESET = '0') then
        LATCHED_IGNORE_PARITY <= '0';
      elsif(CLK'event and CLK = '1')then
        if ( CURRENT_STATE = IDLE ) then
          LATCHED_IGNORE_PARITY <= '0';
        elsif ( IGNORE_PARITY = '1') then
          LATCHED_IGNORE_PARITY <= '1';
        else
          LATCHED_IGNORE_PARITY <= LATCHED_IGNORE_PARITY;
        end if;
      end if;
    end process;

    -- Saves command which resulted in parity error.
    -- Ver2 RevB add code to save command for GCFE timeout.
    -- Ver 5 modify code to save First 15 bits of TEM cmd if Trig error,
    --   save Address and Function bits if non-trig error.
    LAST_ERR_CMD: process (NRESET, CLK)
    begin
      if (NRESET = '0') then
        COMM_ERR_REG <= (others => '0');
      elsif(CLK'event and CLK = '1')then
        -- If a parity error occured on a read command, control state machine
        --  will flow through TEM data read and TEM data check.
        -- A Read or Write Command Error can be asserted after FA Register is loaded.
        if ( CURRENT_STATE = TEM_FA_CHECK and TEM_PARITY_ERR = '1' and STATUS_FA_ERR =
'0') then -- Trig error
          COMM_ERR_REG(15) <= '0';
          COMM_ERR_REG(14 downto 12) <= TRIG_REGISTER;
          COMM_ERR_REG(11 downto 0) <= FA_REGISTER(14 downto 3);
        elsif ( STATUS_FA_ERR = '1' or DATALESS_CMD_ERR = '1' or READ_CMD_ERR = '1' or
WRITE_CMD_ERR = '1' or
          STATUS_DATA_ERR = '1' or STATUS_REPLY_ERR = '1' ) then --  Errors that are not
trig error
          COMM_ERR_REG(15) <= '1';
          COMM_ERR_REG(14 downto 0) <= FA_REGISTER;
        else
          COMM_ERR_REG <= COMM_ERR_REG;
        end if;
      end if;
    end process;

    -- Notifies GCRC_TRIG that GCRC is receiving from or transmitting to the TEM.
    GCRC_TRANSMIT: process (NRESET, CLK)
    begin
         if (NRESET = '0')then
```

```
            GCRC_TRANS_RECV <= '0';
     elsif( CLK'event and CLK = '1') then
       case CURRENT_STATE is
       when IDLE =>
       GCRC_TRANS_RECV <= '0';
       when WAIT_START =>
       GCRC_TRANS_RECV <= '0';
       when others =>
       GCRC_TRANS_RECV <= '1';
       end case;
     end if;
   end process;

   -- Outputs serial data to the TEM.
     OUTPUT_GCRC_DATA: process (NRESET, CLK)
     begin
         if (NRESET = '0')then
             GCRC_DATA_OUT0 <= '0' ;
             GCRC_DATA_OUT1 <= '0' ;
         elsif(CLK'event and CLK = '1')then
       case ( CURRENT_STATE ) is
             when GCRC_TRANS_START =>
                 GCRC_DATA_OUT0 <= '1' ;
                 GCRC_DATA_OUT1 <= '1' ;
       when GCRC_TRANS_FA =>
                 GCRC_DATA_OUT0 <=  FA_REGISTER(14) ;
                 GCRC_DATA_OUT1 <= FA_REGISTER(14) ;
       when GCRC_TRANS_DATA  =>
                 GCRC_DATA_OUT0 <=  FA_REGISTER(14) ;
                 GCRC_DATA_OUT1 <= FA_REGISTER(14) ;
       when GCRC_TRANS_ERR =>
                 GCRC_DATA_OUT0 <= LATCHED_STATUS_ERR ;
                 GCRC_DATA_OUT1 <= LATCHED_STATUS_ERR ;
       when GCRC_TRANS_FA_PARITY =>
                 GCRC_DATA_OUT0 <= PARITY_OUT ;
                 GCRC_DATA_OUT1 <= PARITY_OUT ;
       when GCRC_TRANS_PARITY =>
                 GCRC_DATA_OUT0 <= PARITY_OUT ;
                 GCRC_DATA_OUT1 <= PARITY_OUT ;
       when others =>
                 GCRC_DATA_OUT0 <= '0' ;
                 GCRC_DATA_OUT1 <= '0' ;
       end case;
         end if;
     end process;


   -- Computed parity (odd) is inverted if even parity (CONFIG_PARITY = 1) is stored.
   PARITY_LEVEL: process (NRESET, CONFIG_PARITY, PARITY_TRIG, PARITY_FA_IN,
PARITY_DATA_IN)
     begin
         if (NRESET = '0')then
       COMP_PARITY_TRIG_IN <= '0';
       COMP_PARITY_FA_IN <= '0';
       COMP_PARITY_DATA_IN <= '0';
         elsif ( CONFIG_PARITY = '1' ) then-- Even parity.
          COMP_PARITY_TRIG_IN <= PARITY_TRIG;
          COMP_PARITY_FA_IN <= PARITY_FA_IN;
          COMP_PARITY_DATA_IN <= PARITY_DATA_IN;
```

```
    else                -- Normal odd parity.
         COMP_PARITY_TRIG_IN <= not PARITY_TRIG;
         COMP_PARITY_FA_IN <= not PARITY_FA_IN;
         COMP_PARITY_DATA_IN <= not PARITY_DATA_IN;
    end if;
  end process;

  -- If any TEM command parity is detected, TEM_PARITY_ERR is set.  Tem Parity Err
  --   is RESET at the beginning of each TEM command.
  LATCHED_PARITY_ERROR: process (NRESET, CLK)
  begin
    if (NRESET = '0') then
      TEM_PARITY_ERR <= '0';
    elsif ( CLK'event and CLK = '1') then
      if ( CURRENT_STATE = IDLE ) then
        TEM_PARITY_ERR <= '0';
      elsif ( STATUS_TRIG_ERR= '1' or STATUS_FA_ERR= '1' or STATUS_DATA_ERR= '1') then
        TEM_PARITY_ERR <= '1';
        else
        TEM_PARITY_ERR <= TEM_PARITY_ERR;
      end if;
    end if;
  end process;


  --  Compares transmitted with computed parity bits.
  --  If trigger parity occurs, FA parity and data parity errors are not checked.
  --  IF FA parity error occurs, data parity error is not checked.
  --  Ver 5 added check for Ignore Parity to FA Check.
  PARITY_CHECK: process (NRESET, CURRENT_STATE, TRANS_TRIG_PARITY, TRANS_FA_PARITY,
  TRANS_DATA_PARITY, COMP_PARITY_TRIG_IN, COMP_PARITY_FA_IN, COMP_PARITY_DATA_IN,
  TEM_PARITY_ERR, LATCHED_IGNORE_PARITY, IGNORE_PARITY)
   begin
       if (NRESET = '0' or CURRENT_STATE = IDLE ) then
      STATUS_TRIG_ERR <= '0';
      STATUS_FA_ERR <= '0';
      STATUS_DATA_ERR <= '0';
      CORRECT_TRIG_PARITY <= '0';
      CORRECT_FA_PARITY <= '0';
      CORRECT_DATA_PARITY <= '0';
      elsif ( CURRENT_STATE = TEM_TRIG_CHECK ) then
        STATUS_FA_ERR <= '0';
      STATUS_DATA_ERR <= '0';
      CORRECT_FA_PARITY <= '0';
      CORRECT_DATA_PARITY <= '0';
        if ( TRANS_TRIG_PARITY =  COMP_PARITY_TRIG_IN) then
        STATUS_TRIG_ERR <= '0' ;
             CORRECT_TRIG_PARITY <= '1' ;
      else -- Incorrect trigger parity
        STATUS_TRIG_ERR <= '1' ;
        CORRECT_TRIG_PARITY <= '0' ;
      end if;
      elsif ( CURRENT_STATE = TEM_FA_CHECK and TEM_PARITY_ERR = '0') then
      STATUS_TRIG_ERR <= '0';
      STATUS_DATA_ERR <= '0';
      CORRECT_TRIG_PARITY <= '0';
      CORRECT_DATA_PARITY <= '0';
        if ( TRANS_FA_PARITY =  COMP_PARITY_FA_IN ) then
        STATUS_FA_ERR <= '0' ;
```

```
            CORRECT_FA_PARITY <= '1' ;
      else -- Incorrect function and address parity
        STATUS_FA_ERR <= '1' ;
        CORRECT_FA_PARITY <= '0' ;
      end if;
      elsif ( CURRENT_STATE = TEM_DATA_CHECK and TEM_PARITY_ERR = '0' and
         LATCHED_IGNORE_PARITY = '0') then
      STATUS_TRIG_ERR <= '0';
      STATUS_FA_ERR <= '0';
      CORRECT_TRIG_PARITY <= '0';
      CORRECT_FA_PARITY <= '0';
        if ( TRANS_DATA_PARITY =  COMP_PARITY_DATA_IN) then
        STATUS_DATA_ERR <= '0' ;
            CORRECT_DATA_PARITY <= '1' ;
      else -- Incorrect trigger parity
        STATUS_DATA_ERR <= '1' ;
        CORRECT_DATA_PARITY <= '0' ;
      end if;
    else
      STATUS_TRIG_ERR <= '0';
      STATUS_FA_ERR <= '0';
      STATUS_DATA_ERR <= '0';
      CORRECT_TRIG_PARITY <= '0';
      CORRECT_FA_PARITY <= '0';
      CORRECT_DATA_PARITY <= '0';
    end if;
  end process;

  -- Incoming parity
  PARITY: process (NRESET, TRIG_REGISTER, FA_REGISTER, DATA_REGISTER,
  LATCHED_PARITY_FA_IN, LATCHED_PARITY_DATA_IN, CURRENT_STATE)
  begin
    PARITY_TRIG <=  TRIG_REGISTER(1) xor TRIG_REGISTER(2) ;
    PARITY_FA_IN <=  FA_REGISTER(1) xor LATCHED_PARITY_FA_IN ;
    PARITY_DATA_IN <=  DATA_REGISTER(1) xor LATCHED_PARITY_DATA_IN ;
    if (NRESET = '0') then
      PARITY_RESET_FA <= '1';
      PARITY_RESET_DATA <= '1';
    elsif ( CURRENT_STATE = IDLE or CURRENT_STATE = WAIT_START) then
      PARITY_RESET_FA <= '1';
      PARITY_RESET_DATA <= '1';
    elsif ( CURRENT_STATE = TEM_FA_CHECK ) then
      PARITY_RESET_FA <= '0';
      PARITY_RESET_DATA <= '1';
    else
      PARITY_RESET_FA <= '0';
      PARITY_RESET_DATA <= '0';
    end if;
  end process;

  -- Register for incoming parity.
  LATCHED_PAR: process (NRESET, CLK)
  begin
    if (NRESET = '0') then
      LATCHED_PARITY_FA_IN <= '0';
      LATCHED_PARITY_DATA_IN <= '0';
    elsif ( CLK'event and CLK = '1') then
      if ( PARITY_RESET_FA = '1') then
        LATCHED_PARITY_FA_IN <= '0';
```

```
        else
          LATCHED_PARITY_FA_IN <= PARITY_FA_IN ;
        end if;
        if ( PARITY_RESET_DATA = '1') then
          LATCHED_PARITY_DATA_IN <= '0';
        else
          LATCHED_PARITY_DATA_IN <= PARITY_DATA_IN ;
        end if;
      end if;
   end process;

   -- RESET output parity before function-address and data transmission.
   PAR_OUT_RESET: process( NRESET, CLK)
   begin
      if (NRESET = '0') then
        PARITY_OUT_RESET <= '0';
      elsif ( CLK'event and CLK = '1') then
        if ( CURRENT_STATE = GCRC_TRANS_START or CURRENT_STATE = GCRC_TRANS_FA_PARITY)
then
          PARITY_OUT_RESET <= '1' ;
        else
          PARITY_OUT_RESET <= '0' ;
        end if;
      end if;
   end process;

   -- Invert Parity_out if odd parity (Config parity = 0).
   PAR_OUT: process( NRESET, CLK, XOR_OUT, CONFIG_PARITY, LATCHED_XOR_OUT,
GCRC_DATA_OUT0,
   PARITY_OUT_RESET )
   begin
      XOR_OUT <= (GCRC_DATA_OUT0 xor LATCHED_XOR_OUT) and (not PARITY_OUT_RESET) ;
      PARITY_OUT <= XOR_OUT xnor CONFIG_PARITY;
      if (NRESET = '0') then
        LATCHED_XOR_OUT <= '0';
      elsif ( CLK'event and CLK = '1') then
        if( PARITY_OUT_RESET = '1') then
          LATCHED_XOR_OUT <= '0' ;
        else
          LATCHED_XOR_OUT <= XOR_OUT ;
        end if;
      end if;
   end Process;

end GCRC_CENTRAL;
```

5.3 GCRC_REG.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_REG.vhd
--
--Created by Michael Condon / Jim Ampe, Naval Research Lab
--
--
--Revision: 1.0 (11-12-01)
--           First version.
--     1.0 Rev C (11-16-01), moved reading of registers to GCRC_Cental module to
--       use Data_Register, minimizing flip flops.
--     1.0 Rev D (11-29-01), use upper data word "A5" as a key for changing Parity.
--     2.0  (1/2/02) Make some registers non-resetable and only clocked during load.
--       This is so that SEU flip-flops can be used to implement the registers.
--       Add a status register bit for TEM Command Read/Write error.  Also
--       added a third bit to the GCRC Config register for Parity voting.
--     5.0 (11-08-02) Changed TEM decoding as listed in GCRC_Central module header.
--       Also added Dataless_Cmd_err into status register error bit.
--     5.1 No Change.
--     5.2 No Change.
--     5.3 No Change.
--
--Function: Handles writing to registers.
--
--
-----------------------------------------------------------------------------------------

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity GCRC_REG is
  port(
    NRESET              : in std_logic; -- Chip wide RESET
    CLK            : in std_logic; -- 20MHz system clock
    GCFE_CMD_BIT     : in std_logic; -- Decoded TEM command is for the GCFE
    R_W            : in std_logic_vector (1 downto 0); -- Write register = 01, Read
register = 10
    FUNC        : in std_logic_vector (2 downto 0); -- TEM command function
    REG_DATA_IN        : in std_logic_vector (15 downto 0);--input parallel data
    CORRECT_FA_PARITY  : in std_logic; -- Function and address of TEM command
    CORRECT_DATA_PARITY: in std_logic;  -- Data of TEM command
    TEM_PARITY_ERR   : in std_logic; -- Any TEM command parity error.
    CORRECT_ADDR     : in std_logic; -- TEM command matches GCRC wired address.
    LATCHED_CORRECT_ADDR : in std_logic; -- Latched version of correct address.
    STATUS_TRIG_ERR  : in std_logic; -- Trigger parity error
    STATUS_FA_ERR      : in std_logic; -- Function address parity error
    STATUS_DATA_ERR  : in std_logic; -- Data parity error
    STATUS_REPLY_ERR : in std_logic; -- Error, GCFE not reply befor timeout.
    READ_CMD_ERR     : in std_logic; -- Correct parity from TEM, but Invalid Command.
    DAC_DATA_REG_SHIFT : in std_logic; -- Shift DAC data by one, to GCRC_DAC module.
    DAC_DATA_IN      : in std_logic; -- Data shifted out of onboard DAC.
    DATALESS_CMD_ERR : in std_logic; -- Dataless Cmd not recognized.
    DAC_WRITE      : buffer std_logic; -- Onboard DAC is being written to.
    DAC_DATA       : buffer std_logic; -- Most sig bit of reg, to onboard DAC.
    CONFIG_PARITY      : buffer std_logic; -- Parity mode
```

```
     WRITE_CMD_ERR    : buffer std_logic;-- TEM command error, no GCRC action taken.
     STATUS_REG       : buffer std_logic_vector(4 downto 0); -- Register to GCRC_CENTRAL
module.
     RANGE_DELAY_REG    : buffer std_logic_vector(5 downto 0); -- Peak hold settling
time.
     ADC_SAMPLE_DELAY_REG : buffer std_logic_vector(5 downto 0); -- Analog Mux settling
time.
     ADC_CONV_TIME_REG    : buffer std_logic_vector(7 downto 0);  -- ADC Conversion
time.
     DAC_DATA_REG     : buffer std_logic_vector(15 downto 0); -- Register to
GCRC_Central module.
     CONFIG_REG       : buffer std_logic_vector(2 downto 0)  -- Register to GCRC_Central
module.
     );
end GCRC_REG;


architecture GCRC_REG of GCRC_REG is

constant IDLE      : std_logic_vector(1 downto 0) := "00";
constant DELAY     : std_logic_vector(1 downto 0) := "01";
constant LOAD      : std_logic_vector(1 downto 0) := "10";
signal WRITE_KEY   : std_logic;
signal SEU_REG_CLK      : std_logic;
signal NONSEU_REG_LOAD  : std_logic;
signal CURRENT_STATE, NEXT_STATE : std_logic_vector(1 downto 0);

begin
  -- Config_parity normally set to '0', odd parity.  Both bits must be set to '1' for
even parity.
  CONFIG_PARITY <= (CONFIG_REG(0) and CONFIG_REG(1) and CONFIG_REG(2));
  DAC_DATA <= DAC_DATA_REG(15);
  --  Write_key asserted '1' when upper data word = A5, used as extra security.
  WRITE_KEY <= REG_DATA_IN(15) and REG_DATA_IN(13) and REG_DATA_IN(10) and
  REG_DATA_IN(8) and ( not(REG_DATA_IN(14) or REG_DATA_IN(12) or
  REG_DATA_IN(11) or REG_DATA_IN(9) ) );

  -- Clear status register if it is read out, otherwise set a bit if an error occurs.
    STATUS_REGISTER: process (NRESET, CLK)
    begin
        if (NRESET = '0')then
            STATUS_REG <= (others => '0');
    elsif(CLK'EVENT and CLK = '1')then
       -- Read from registers.
       if (CORRECT_FA_PARITY = '1' and R_W = "10" and GCFE_CMD_BIT = '0' and
          CORRECT_ADDR = '1' and FUNC = "000") then
          STATUS_REG <= (others => '0'); -- Clear status when read.
       elsif (STATUS_TRIG_ERR = '1') then  -- Latch status error bits.
            STATUS_REG(0) <= '1';
          STATUS_REG(4 downto 1) <= STATUS_REG(4 downto 1);
       elsif (STATUS_FA_ERR = '1') then
          STATUS_REG(1) <= '1' ;
          STATUS_REG(0) <= STATUS_REG(0);
          STATUS_REG(4 downto 2) <= STATUS_REG(4 downto 2);
       elsif (STATUS_DATA_ERR = '1') then
            STATUS_REG(2) <= '1' ;
          STATUS_REG(1 downto 0) <= STATUS_REG(1 downto 0);
          STATUS_REG(4 downto 3) <= STATUS_REG(4 downto 3);
       elsif (STATUS_REPLY_ERR = '1') then
```

```
          STATUS_REG(3) <= '1' ;
          STATUS_REG(2 downto 0) <= STATUS_REG(2 downto 0);
          STATUS_REG(4) <= STATUS_REG(4);
        elsif (WRITE_CMD_ERR = '1' or READ_CMD_ERR = '1' or DATALESS_CMD_ERR = '1') then
          STATUS_REG(4) <= '1' ;
          STATUS_REG(3 downto 0) <= STATUS_REG(3 downto 0);
        else
          STATUS_REG(4 downto 0) <= STATUS_REG(4 downto 0);
        end if;
      end if; -- If CLK'event
    end process;

    FSM_REG_WRITE: process (CURRENT_STATE, TEM_PARITY_ERR, FUNC,
    CORRECT_DATA_PARITY, R_W, GCFE_CMD_BIT, LATCHED_CORRECT_ADDR )
    begin
      case CURRENT_STATE is
        when IDLE =>  -- State 0
        if (TEM_PARITY_ERR = '0' and CORRECT_DATA_PARITY = '1' and R_W = "01"
        and GCFE_CMD_BIT = '0' and LATCHED_CORRECT_ADDR = '1') then
          NEXT_STATE <= DELAY;
        else
          NEXT_STATE <= IDLE;
        end if;

        when DELAY =>  -- State 1
        NEXT_STATE <= LOAD;

        when LOAD =>   -- State 2
        NEXT_STATE <= IDLE;

        when others => -- State 3
        NEXT_STATE <= IDLE;

      end case;
    end process;


    FSM_CONTROL: process (NRESET, CLK)
    begin
      if (NRESET = '0') then
        CURRENT_STATE <= IDLE;
      elsif (CLK'EVENT and CLK = '1') then
        CURRENT_STATE <= NEXT_STATE;
      end if;
    end process;

    --  Clock SEU registers if write to Time Delay Reg 1,2 or 3.
    SEU_REG_CLOCK: process (NRESET, CURRENT_STATE, FUNC )
    begin
      if ( NRESET = '0') then
        SEU_REG_CLK <= '0';
      elsif (CURRENT_STATE = LOAD and
        ( FUNC = "011" or FUNC = "100" or FUNC = "101") ) then
          SEU_REG_CLK <= '1';
      else
          SEU_REG_CLK <= '0';
      end if;
    end process;
```

```
    WRITE_SEU_REGISTERS: process (SEU_REG_CLK )
    begin
     if ( SEU_REG_CLK'EVENT and SEU_REG_CLK = '1')then
       case FUNC is
       -- "000" Status register, no write command

       -- "001" Last error command register, no write

       when "011" =>  -- Peak hold to GCFE range decision delay.
            RANGE_DELAY_REG <= REG_DATA_IN(5 downto 0) ;
       ADC_SAMPLE_DELAY_REG <= ADC_SAMPLE_DELAY_REG;
       ADC_CONV_TIME_REG <= ADC_CONV_TIME_REG;

       when "100" =>   -- GCFE range decision to ADC sample delay
            ADC_SAMPLE_DELAY_REG <= REG_DATA_IN(5 downto 0) ;
       RANGE_DELAY_REG <= RANGE_DELAY_REG;
       ADC_CONV_TIME_REG <= ADC_CONV_TIME_REG;

       when "101" =>   -- ADC conversion time delay
            ADC_CONV_TIME_REG <= REG_DATA_IN(7 downto 0) ;
       RANGE_DELAY_REG <= RANGE_DELAY_REG;
       ADC_SAMPLE_DELAY_REG <= ADC_SAMPLE_DELAY_REG;

       when others =>
       RANGE_DELAY_REG <= RANGE_DELAY_REG;
       ADC_SAMPLE_DELAY_REG <= ADC_SAMPLE_DELAY_REG;
       ADC_CONV_TIME_REG <= ADC_CONV_TIME_REG;
       end case;
     end if;
    end process;

    WRITE_NON_SEU_REG: process (NRESET, CLK)
    begin
        if (NRESET = '0')then
            CONFIG_REG <= (others => '0');
       DAC_DATA_REG <= (others => '0');
       NONSEU_REG_LOAD <= '0';
       DAC_WRITE <= '0';

     elsif(CLK'EVENT and CLK = '1')then

       if ( CURRENT_STATE = DELAY ) then
         case FUNC is
         when "111" =>  -- Write to config register.
         DAC_WRITE <= '0';
         DAC_DATA_REG <= DAC_DATA_REG;
         if  (WRITE_KEY = '1') then -- Use extra check to change parity or GCFE read
config.
                  CONFIG_REG <= REG_DATA_IN(2 downto 0) ;
           NONSEU_REG_LOAD <= '1';
         else
           CONFIG_REG <= CONFIG_REG;
           NONSEU_REG_LOAD <= '0';
         end if;

         when "110" =>  -- Onboard digital to analog converter
         DAC_DATA_REG <= REG_DATA_IN ;
         DAC_WRITE <= '1';
         NONSEU_REG_LOAD <= '1';
```

```
          when others =>
          CONFIG_REG <= CONFIG_REG;
          DAC_WRITE <= '0';
          NONSEU_REG_LOAD <= '0';
          if ( DAC_DATA_REG_SHIFT = '1') then
            DAC_DATA_REG(15 downto 1) <= DAC_DATA_REG(14 downto 0);
            DAC_DATA_REG(0) <= DAC_DATA_IN;
          else
            DAC_DATA_REG <= DAC_DATA_REG;
          end if;

          end case;

        -- Pass shifted dac_data_reg to gcrc_dac module.
        elsif ( DAC_DATA_REG_SHIFT = '1') then
          DAC_DATA_REG(15 downto 1) <= DAC_DATA_REG(14 downto 0);
          DAC_DATA_REG(0) <= DAC_DATA_IN; -- Take in data shifted out of dac.
          CONFIG_REG <=  CONFIG_REG;
          DAC_WRITE <= '0';
          NONSEU_REG_LOAD <= '0';
        else
          CONFIG_REG <= CONFIG_REG;
          DAC_DATA_REG <= DAC_DATA_REG;
          DAC_WRITE <= '0';
          NONSEU_REG_LOAD <= '0';
        end if;
      end if;
    end process;

    -- Error given if write gcrc command is to correct address,
    --     but no register write is performed.
    WRITE_CMD_ERROR: process (NRESET, CURRENT_STATE, SEU_REG_CLK, NONSEU_REG_LOAD)
    begin
      if (NRESET = '0') then
        WRITE_CMD_ERR <= '0';
      elsif ( CURRENT_STATE = LOAD and NONSEU_REG_LOAD = '0' and SEU_REG_CLK = '0') then
        WRITE_CMD_ERR <= '1';
      else
        WRITE_CMD_ERR <= '0';
      end if;
    end process;


end GCRC_REG;
```

5.4 GCRC_DAC.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_DAC.vhd
--
--Created by Jim Ampe / Michael Condon, Naval Research Lab
--
--Revision: 1.0 (11-12-01)
--          First version.
--     5.0 No change.
--     5.1 No Change.
--     5.2 No Change.
--     5.3 No Change.
--
--Function: Controls Function and Signals between the onboard digital to analog
--     converter and the GCRC.  Module GCRC_REG outputs the DAC data values.
--
--
-----------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity GCRC_DAC is
  port (
  NRESET        : in std_logic;  -- Global RESET.
  CLK           : in std_logic;  -- System 20 Mhz clk.
  DAC_WRITE     : in std_logic; -- DAC write request from GCRC_REG module.
  DAC_DATA_REG_SHIFT : buffer std_logic;  -- Command DAC data to be shifted, to
GCRC_REG.
  DAC_NCS            : buffer STD_LOGIC; -- Chip SELECT for the DAC
  DAC_CLK       : buffer std_logic; -- Clock to DAC
  GCRC_DAC_WRITE   : buffer std_logic -- GCRC is writing to the DAC, to GCRC_TRIG
module.
     );
end GCRC_DAC;


architecture GCRC_DAC of GCRC_DAC is
  Constant IDLE                   :std_logic := '0';
  Constant WRITE                  :std_logic := '1';
  signal DAC_WRITE_DONE      : std_logic;
  signal DAC_BIT_COUNT       : std_logic_vector(4 downto 0);
  signal CLK_COUNT        : std_logic_vector(1 downto 0);
  signal CURRENT_STATE,NEXT_STATE: std_logic;

begin

  FSM_State: process (CURRENT_STATE, DAC_WRITE, DAC_WRITE_DONE)
    begin
          case Current_State is

            when IDLE =>
            DAC_NCS <= '1';
            GCRC_DAC_WRITE <= '0';
            if ( DAC_WRITE = '1' ) then
            NEXT_STATE <= WRITE ;
```

```
        else
          NEXT_STATE <= IDLE;
        end if;

          when WRITE =>
        DAC_NCS <= '0' ;
        GCRC_DAC_WRITE <= '1';
        if (DAC_WRITE_DONE = '1' ) then
          NEXT_STATE <= IDLE ;
        else
          NEXT_STATE <= WRITE;
        end if;

          when others=>
        DAC_NCS <= '1';
        NEXT_STATE <= IDLE;
        GCRC_DAC_WRITE <= '0';
        end case;
  end process;

FSM_STATE_CHANGE:process (NRESET, CLK)
begin
  if (NRESET = '0') then
    CURRENT_STATE <= IDLE;
  elsif( CLK'event and CLK = '1') then
    CURRENT_STATE <= NEXT_STATE;
  end if;
end process;

DAC_SERIAL_WRITE: process(NRESET, CLK)
begin
  if (NRESET = '0') then
    DAC_CLK <= '0';
    CLK_COUNT <= (others => '0');
    DAC_BIT_COUNT <= (others => '0');
    DAC_WRITE_DONE <= '0';
    DAC_DATA_REG_SHIFT <= '0';
  elsif( CLK'event and CLK = '1') then
    if ( CURRENT_STATE = WRITE ) then
      if ( DAC_BIT_COUNT = "10000") then
        DAC_WRITE_DONE <= '1' ;
        CLK_COUNT <= (others => '0');
        DAC_CLK <= '0';
        DAC_BIT_COUNT <= (others => '0');
        DAC_DATA_REG_SHIFT <= '0';
      else
        CLK_COUNT <= CLK_COUNT + 1 ;
        DAC_WRITE_DONE <= '0';
        case CLK_COUNT is
          when "00" =>
          DAC_CLK <= '0' ;
          DAC_DATA_REG_SHIFT <= '0';
          DAC_BIT_COUNT <= DAC_BIT_COUNT;

          when "01" =>
          DAC_CLK <= '0' ;
          DAC_DATA_REG_SHIFT <= '0';
          DAC_BIT_COUNT <= DAC_BIT_COUNT;
```

```
              when "10" =>
              DAC_CLK <= '1' ;
              DAC_DATA_REG_SHIFT <= '1' ;
              DAC_BIT_COUNT <= DAC_BIT_COUNT;

              when "11" =>
              DAC_CLK <= '1' ;
              DAC_DATA_REG_SHIFT <= '0' ;
              DAC_BIT_COUNT <= DAC_BIT_COUNT + 1 ;

              when others =>
              DAC_CLK <= '0';
              DAC_DATA_REG_SHIFT <= '0';
              DAC_BIT_COUNT <= DAC_BIT_COUNT;
            end case;
          end if; -- If Dac_bit_count
        else -- If current_state = write.
          DAC_WRITE_DONE <= '0';
          CLK_COUNT <= (others => '0');
          DAC_CLK <= '0';
          DAC_BIT_COUNT <= (others => '0');
          DAC_DATA_REG_SHIFT <= '0';
        end if;
      end if; -- If clk'event

   end process;

end GCRC_DAC;
```

5.5 GCRC_GCFE.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--Filename: GCRC_GCFE.vhd
--Created by Jim Ampe / Michael Condon, Naval Research Lab.
--Revision: 1.0 (11-12-01)
--            First version.
--     1.0 Rev C (11-16-01), fixed problems in forwarding GCFE returned data.
--       Also fixed glitches on GCFE RESET line.
--     2.0 RevB (1-16-02) modified GCFE_WRITE signal assignment.
--     4.0 (3-05-02) Add flip flops to Clock in GCFE data prior to multiplexer.
--     4.0 Rev A (3-29-02) Added check for Latched_Correct_Addr and Correct_Addr
--       to get out of IDLE state.
--     5.0 (11-08-02) Changed TEM command decoding as stated in GCRC_Cental header
notes.
--       Changed GCFE write
--       from: GCFE Data transmitted on pos Clk, Daq_Clk inverted from Clk,
--       to: GCFE Data transmitted on positive Clk, Daq_Clk inverted.
--       Changed GCFE read
--       from: Read GCFE data on pos Clk, Daq_Clk non-inverted
--       to: Read GCFE data on pos Clk, Daq_Clk inverted.
--     5.1 (12-09-02) Changed Clocking in GCFE communications
--       GCFE write
--         Data transmitted on falling system clk, Daq_Clk asserted high on rising
system clk.
--       GCFE read
--         Daq_Clk asserted high on rising system clk, Return data latched on rising
system clk.
--       Readout Clocking
--         Daq_Clk asserteed high on falling system clk.
--     5.2  (12-10-02) Change GCFE clocking to Daq_Clk in phase with system clk for
register reading and writing.
--         Data transmitted out of phase and returned data is latched inphase with
system clk.
--     5.3 (12-12-02) Changed GCFE clocking communicattion to read back GCFE data at
half the system clk rate.
--       GCFE write
--         Data transmitted on falling system clk, Daq_Clk asserted high on rising
system clk.
--       GCFE read
--         Daq_Clk asserted high on ever-other rising system clk, Return data latched
on same rising half clk.
--       Readout Clocking
--         Daq_Clk asserteed high on falling system clk.
--       Method of reading in GCFE data at half clk rate and sending out data at full
clk rate.
--         Not having a FIFO, the implementation is as follows:
--         1) Two GCFE data bits are read in at half clk rate.
--         2) Central Control shifts out the stored 14 FA bits + parity +  2 GCFE data
bits at
--           full clk rate while GCFE module is reading 8 more data bits at half rate.
--         3) Cental Control shifts out the 8 GCFE data bits at full clk rate while
GCFE module
--           reads 4 GCFE data bits at half rate.
--         4) Central control shifts out the 4 data bits at full clk rate, + last two
GCFE data bits
--           and appends error and parity bits.
--
```

```
--
--Function:  Handles communication to the external GCFE chips.
--
-------------------------------------------------------------------------------------------

library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity GCRC_GCFE is
  port(
  NRESET           : in std_logic; -- Global RESET.
  CLK              : in std_logic; -- System 20 MHz clk.
  GCFE_ADDR        : in std_logic_vector(3 downto 0); -- GCFE address sent by TEM.
  GCFE_CMD_BIT     : in std_logic; -- TEM command is for GCFE.
  R_W            : in std_logic_vector(1 downto 0); -- Read/Write not
  FUNC         : in std_logic_vector(2 downto 0); --  TEM function bits
  TEM_CMD_DATA_MSB : in std_logic; -- TEM cmd data most significant bit, from
CENTRAL_CONTROL.
  CORRECT_FA_PARITY  : in std_logic; -- TEM command function and addrss has correct
parity.
  CORRECT_DATA_PARITY: in std_logic; -- TEM command data has correct parity.
  TEM_PARITY_ERR   : in std_logic; -- Latched value of any TEM command parity error.
  GCFE_DATA      : in std_logic_vector(11 downto 0); -- Serial data bits from 12 GCFEs.
  GCFE_CLK_ENV     : in std_logic; -- DAQ_CLK envelope from GCRC READOUT module.
  GCFE_RESET_ENV   : in std_logic;  -- RESET envelope from GCRC READOUT module.
  RIGHT_FIRST      : in std_logic; -- Right first, log end addressing switch.
  LATCHED_CORRECT_ADDR : in std_logic; -- Latched correct GCRC address.
  CORRECT_ADDR     : in std_logic; -- Correct GCRC Address.
  STATUS_REPLY_ERR : buffer std_logic; -- Error, GCFE not reply prior to timeout.
  DAQ_CLK          : buffer std_logic;-- Clk out to GCFEs.
  GCFE_DELAYED_RTN_DATA : buffer std_logic; -- Mux selected 1 of 12 GCFE data bits.
  GCFE_TRANS       : buffer std_logic; -- GCFE is transmitting data to GCRC.
  GCFE_WRITE       : buffer std_logic; -- GCRC is writing to GCFE.
  CMD            : buffer std_logic;-- Serial command and data to the GCFE.
  GRESET           : buffer std_logic; -- RESET command to GCFE as data envelope.
  GCFE_TRANS_LOWER : buffer std_logic; -- GCFE readback data available to Central
module.
  GCFE_SAMPLED_DATA  : buffer std_logic_vector(11 downto 0) -- Clocked in GCFE data
line.
  );
end GCRC_GCFE;


architecture GCFE_CNTRL_ENTITY of GCRC_GCFE is
Constant GCFE_EXTRA_CLKS     : std_logic_vector(4 downto 0) := "00010";
-- Ver 5.1  Increment GCFE_Cmd_Length and GCFE_Short_Cmd_Length by one due to extra clk
given at beginning of write.
Constant GCFE_CMD_LENGTH     : std_logic_vector(4 downto 0) := "11001";
-- Ver 5.3 Increment Short Cmd Length by 3 to give one more fast clk prior to reading
at half clk rate.
Constant GCFE_SHORT_CMD_LENGTH : std_logic_vector(4 downto 0) := "01101";
Constant GCFE_REPLY_DATA_LENGTH: std_logic_vector(4 downto 0) := "10000";
Constant GCFE_REPLY_HALF_LENGTH: std_logic_vector(4 downto 0) := "01000";
-- Ver 5.3 Halve Gcfe_Reply_Timeout due to counting at half system clk rate.
Constant GCFE_REPLY_TIMEOUT    : std_logic_vector(4 downto 0) := "00111";
Constant IDLE                : std_logic_vector(3 downto 0) := "0000";
Constant RESET_ENV             : std_logic_vector(3 downto 0) := "0001";
Constant START_BIT             : std_logic_vector(3 downto 0) := "0010";
```

```
Constant WRITE                : std_logic_vector(3 downto 0) := "0011";
Constant EXTRA_CLKS             : std_logic_vector(3 downto 0) := "0100";
Constant REPLY_WAIT           : std_logic_vector(3 downto 0) := "0101";
Constant READ0             : std_logic_vector(3 downto 0) := "0110"; -- Ver 5.3 changed
from Read to Read0.
Constant READ1           : std_logic_vector(3 downto 0) := "0111"; -- Ver 5.3 added.
Constant READ2           : std_logic_vector(3 downto 0) := "1000"; -- Ver 5.3 added.
Constant READ3           : std_logic_vector(3 downto 0) := "1001"; -- Ver 5.3 added.

signal Counter             : std_logic_vector(4 downto 0);
signal COUNTER_EN               : std_logic;
signal READBACK_LATCH      : std_logic;
signal GCFE_CMD            : std_logic; -- Command bit to GCFE, renamed CMD.
signal GCFE_RESET          : std_logic; -- Command envelope RESET to GCFE, renamed
GRESET.
signal GCFE_RTN_DATA       : std_logic;
signal EN_READ_CLK         : std_logic; -- Ver 5.3 added.  Enable for half rate clock.
signal DAQ_CLK_INPHASE       : std_logic; -- Ver 5.3 added.
signal REG_DAQ_CLK_INPHASE   : std_logic; -- Ver 5.3 added.
signal GCFE_FULL_CLK       : std_logic; -- Ver 5.3 added.  Enable for Daq_Clk at system
Clk rate.
signal GCFE_HALF_CLK       : std_logic; -- Ver 5.3 added.  Enable for Daq_Clk at half
system Clk rate.
signal GCFE_ADDR_FA_REG      : std_logic_vector(8 downto 0);
signal CURRENT_STATE       : std_logic_vector(3 downto 0);
signal NEXT_STATE          : std_logic_vector(3 downto 0);
signal RETURN_DATA0        : std_logic_vector(3 downto 0);  -- Ver 5.3.  Temporary GCFE
data holding register.
signal RETURN_DATA1        : std_logic_vector(7 downto 0); -- Ver 5.3.Temporary GCFE
data holding register.

begin


  -- CMD <= GCFE_CMD;  -- Ver. 5.1, changed from direct assignment to transmitted on
falling clk.
  GRESET <= GCFE_RESET; -- Rename RESET line to GCFE chips.

  FSM_STATE: process (R_W, GCFE_CMD_BIT, CORRECT_FA_PARITY, CORRECT_DATA_PARITY,
  CURRENT_STATE, COUNTER, READBACK_LATCH, TEM_PARITY_ERR, GCFE_RTN_DATA,
  LATCHED_CORRECT_ADDR, CORRECT_ADDR )
  begin
       case CURRENT_STATE is
       -- Ver 4B, Added Correct_Addr and Latched_Correct_Addr to IDLE logic.
       when IDLE =>  -- State 0
         COUNTER_EN <= '0';
         STATUS_REPLY_ERR <= '0';
         GCFE_TRANS <= '0';
         GCFE_TRANS_LOWER <= '0';
         if ( GCFE_CMD_BIT = '1' and TEM_PARITY_ERR = '0' and
         ( (R_W = "10" and CORRECT_FA_PARITY = '1' and CORRECT_ADDR = '1') or
         ( R_W = "01" and CORRECT_DATA_PARITY = '1' and LATCHED_CORRECT_ADDR = '1')) )
then
           NEXT_STATE <= RESET_ENV ;
         else
           NEXT_STATE <= IDLE;
         end if;

       when RESET_ENV =>   -- State 1
```

```
      COUNTER_EN <= '0';
      STATUS_REPLY_ERR <= '0';
      GCFE_TRANS <= '0';
      NEXT_STATE <= START_BIT;
      GCFE_TRANS_LOWER <= '0';

    when START_BIT =>   -- State 2
      COUNTER_EN <= '0';
      STATUS_REPLY_ERR <= '0';
      GCFE_TRANS <= '0';
      NEXT_STATE <= WRITE;
      GCFE_TRANS_LOWER <= '0';

    when WRITE =>        -- State 3
      STATUS_REPLY_ERR <= '0';
      GCFE_TRANS <= '0' ;
      GCFE_TRANS_LOWER <= '0';
      if ( READBACK_LATCH = '1' ) then
        if ( COUNTER = GCFE_SHORT_CMD_LENGTH ) then
          NEXT_STATE <= REPLY_WAIT ;
          COUNTER_EN <= '0' ;
        else
          NEXT_STATE <= WRITE;
          COUNTER_EN <= '1';
        end if;
      elsif ( COUNTER = GCFE_CMD_LENGTH ) then
        COUNTER_EN <= '0';
        NEXT_STATE <= EXTRA_CLKS;
      else
        NEXT_STATE <= WRITE ;
        COUNTER_EN <= '1';
      end if;

    when EXTRA_CLKS =>   -- State 4
      STATUS_REPLY_ERR <= '0';
      GCFE_TRANS <= '0';
      GCFE_TRANS_LOWER <= '0';
      if( COUNTER = GCFE_EXTRA_CLKS )then
        COUNTER_EN <= '0';
        NEXT_STATE <= IDLE;
      else
        NEXT_STATE <= EXTRA_CLKS;
        COUNTER_EN <= '1';
      end if;

    when REPLY_WAIT =>   -- State 5
      GCFE_TRANS <= '0';
      GCFE_TRANS_LOWER <= '0';
      if ( COUNTER = GCFE_REPLY_TIMEOUT ) then
        NEXT_STATE <= IDLE;
        COUNTER_EN <= '0';
        STATUS_REPLY_ERR <= '1';
      elsif ( GCFE_RTN_DATA = '1' ) then
        NEXT_STATE <= READ0;
        COUNTER_EN <= '0' ;
        STATUS_REPLY_ERR <= '0';
      else
        NEXT_STATE <= REPLY_WAIT;
        COUNTER_EN <= '1';
```

```
            STATUS_REPLY_ERR <= '0';
          end if;

        -- Ver 5.3, Renamed from Read to Read0.  State for first 2 returned GCFE data
bits.
        when READ0 =>     -- State 6
          COUNTER_EN <= '1';
          STATUS_REPLY_ERR <= '0';
          GCFE_TRANS_LOWER <= '0';
          if( COUNTER = "00001" )then
            NEXT_STATE <= READ1;
            GCFE_TRANS <= '1';
          else
            NEXT_STATE <= READ0 ;
            GCFE_TRANS <= '0';
          end if;

        -- Added Ver 5.3. State for next 8 returned GCFE data bits.
        when READ1 =>     -- State 7
          COUNTER_EN <= '1' ;
          STATUS_REPLY_ERR <= '0';
          GCFE_TRANS <= '1';
          GCFE_TRANS_LOWER <= '0';
          if( COUNTER = "01001" )then
            NEXT_STATE <= READ2;
          else
            NEXT_STATE <= READ1;
          end if;

        -- Added Ver 5.3. State for next 4 returned GCFE data bits.
        when READ2 =>     -- State 8
          GCFE_TRANS_LOWER <= '1';
          COUNTER_EN <= '1' ;
          STATUS_REPLY_ERR <= '0';
          GCFE_TRANS <= '1';
          if( COUNTER = "01101" )then
            NEXT_STATE <= READ3 ;
          else
            NEXT_STATE <= READ2 ;
          end if;

        -- Added Ver 5.3. State for last 2 returned GCFE data bit and transferring data
to Central Module.
        when READ3 =>     -- State 9
          GCFE_TRANS_LOWER <= '1';
          COUNTER_EN <= '1' ;
          STATUS_REPLY_ERR <= '0';
          GCFE_TRANS <= '1';
          if( COUNTER = "10000" )then
            NEXT_STATE <= IDLE ;
          else
            NEXT_STATE <= READ3 ;
          end if;

        when others =>
          NEXT_STATE <= IDLE;
          COUNTER_EN <= '0';
          STATUS_REPLY_ERR <= '0';
          GCFE_TRANS <= '0';
```

```
          GCFE_TRANS_LOWER <= '0';
        end case;
   end process;

   FSM_CONTROL: process (NRESET, CLK)
   begin
     if (NRESET = '0') then
        CURRENT_STATE <= IDLE;
     elsif(CLK'event and CLK = '1') then
        -- If reading back data at half the clock rate, update the state only on DAQ_Clk
rising edges.
        if ( CURRENT_STATE=REPLY_WAIT or CURRENT_STATE=READ0 or CURRENT_STATE=READ1 or
CURRENT_STATE=READ2
          or CURRENT_STATE=READ3 ) then
          if ( EN_READ_CLK = '0' ) then
            CURRENT_STATE <= NEXT_STATE;
          else
            CURRENT_STATE <= CURRENT_STATE;
          end if;
        else
          CURRENT_STATE <= NEXT_STATE;
        end if;
     end if;
   end process;

   -- Ver 5.1 Changed
   --  From: Daq_Clk Inverted from Clk on GCFE write, Daq_Clk positive with Clk on GCFE
Read
   --  To: Daq_Clk inverted from Clk on Log-accept and Range bit readouts
   --   Daq_Clk Not inverted on both GCFE write and read.
   -- Ver 5.3  Add code for Daq_Clk to be half the system clk rate for GCFE data
readback.
   GCFE_CLK: process (NRESET, CLK, GCFE_CLK_ENV, EN_READ_CLK, REG_DAQ_CLK_INPHASE,
GCFE_FULL_CLK, GCFE_HALF_CLK )
   begin
     if (NRESET = '0') then
        DAQ_CLK <= '0';
        DAQ_CLK_INPHASE <= '0';
     -- Inverted clock for clocking out log-accept and range readout bits.
     elsif ( GCFE_CLK_ENV = '1' ) then
        DAQ_CLK <= not CLK;
        DAQ_CLK_INPHASE <= '0';
     -- Non- inverted clock for GCFE writing
     elsif ( GCFE_FULL_CLK = '1') then
        DAQ_CLK <= CLK;
        DAQ_CLK_INPHASE <= '1';
     -- Half rate clock for GCFE data readback.
     elsif ( GCFE_HALF_CLK = '1' ) then
        DAQ_CLK_INPHASE <= '1';
        if ( EN_READ_CLK = '1' ) then
          DAQ_CLK <= '1';
        else
          DAQ_CLK <= '0';
        end if;
     else
        DAQ_CLK_INPHASE <= '0';
        -- Does not glitch Daq_Clk when turning off.  Holds Daq_Clk high for one clock
cycle when completing.
        if ( REG_DAQ_CLK_INPHASE = '1' ) then
```

```
          DAQ_CLK <= '1';
        else
          DAQ_CLK <= '0';
        end if;
      end if;
    end process;

    -- Enable for full or half rate Daq_Clk to GCFE.
    GCFE_CLK_EN: process ( NRESET, CLK )
    begin
      if (NRESET = '0') then
        GCFE_FULL_CLK <= '0';
        GCFE_HALF_CLK <= '0';
      elsif ( CLK'event and CLK = '1') then
        if ( NEXT_STATE = WRITE ) then
          GCFE_FULL_CLK <= '1';
          GCFE_HALF_CLK <= '0';
        elsif ( NEXT_STATE = REPLY_WAIT ) then
          GCFE_FULL_CLK <= '0';
          GCFE_HALF_CLK <= '1';
        elsif ( CURRENT_STATE = IDLE ) then
          GCFE_FULL_CLK <= '0';
          GCFE_HALF_CLK <= '0';
        else
          GCFE_FULL_CLK <= GCFE_FULL_CLK;
          GCFE_HALF_CLK <= GCFE_HALF_CLK;
        end if;
      end if;
    end process;

    -- Ver 5.3 Register added to remember if Daq_Clk was in phase or out-of-phase with
system clk.
    -- In phase Daq_Clks will produce glitch clk when turned off.
    REG_DAQ_CLK_PHASE: process(NRESET, CLK)
    begin
      if (NRESET = '0') then
        REG_DAQ_CLK_INPHASE <= '0';
      elsif ( CLK'event and CLK='1' ) then
        REG_DAQ_CLK_INPHASE <= DAQ_CLK_INPHASE;
      end if;
    end process;

    --  Ver 5.3 added.  This determines when the half rate clock is asserted, and
correspondingly
    --    when the data is latched in.
    HALF_CLK_ENABLE: process (NRESET, CLK)
    begin
      if (NRESET = '0') then
        EN_READ_CLK <= '1';
      elsif ( CLK'event and CLK = '1') then
        if ( CURRENT_STATE=REPLY_WAIT or CURRENT_STATE=READ0 or CURRENT_STATE=READ1 or
          CURRENT_STATE = READ2 or CURRENT_STATE = READ3 )then
          EN_READ_CLK <= not EN_READ_CLK;
        else
          EN_READ_CLK <= '1';
        end if;
      end if;
    end process;
```

```
-- GCFE RESET is controlled in this module and in the GCRC_READOUT module.
GCFE_RESET_CONTROL: process (NRESET, CLK)
begin
  if (NRESET = '0') then
    GCFE_RESET <= '1';
  elsif ( CLK'event and CLK = '1' ) then
    if ( CURRENT_STATE /= IDLE or GCFE_RESET_ENV = '0') then
      GCFE_RESET <= '0';
    else
      GCFE_RESET <= '1';
    end if;
  end if;
end process;

-- GCFE Write disables triggers and informs GCRC_Central to shift TEM cmd data.
GCFE_WR: process (NRESET, CLK)
begin
  if (NRESET = '0') then
    GCFE_WRITE <= '0';
  elsif ( CLK'event and CLK = '1' ) then
    case CURRENT_STATE is
      when START_BIT =>
      GCFE_WRITE <= '1';
      when WRITE =>
      GCFE_WRITE <= '1';
      when EXTRA_CLKS =>
      GCFE_WRITE <= '1';
      when REPLY_WAIT =>
      GCFE_WRITE <= '1';
      when others =>
      GCFE_WRITE <= '0';
    end case;
  end if;
end process;

-- Remember if expecting returned data from the GCFE
GCFE_READBACK_LATCH: process (NRESET, CLK)
begin
  if ( NRESET = '0') then
    READBACK_LATCH <= '0';
  elsif ( CLK'event and CLK = '1' ) then
    if ( CURRENT_STATE = IDLE ) then
      if ( R_W = "10") then
        READBACK_LATCH <= '1';
      else
        READBACK_LATCH <= '0';
      end if;
    else
      READBACK_LATCH <= READBACK_LATCH;
    end if;
  end if;
end process;

-- Clocks in data from GCFE.  1 clock delay in GCFE data.
-- No Reset condition so Xilinx implements in IOB FF.
GCFE_DATA_SAMPLE: process ( CLK)
begin
  if ( CLK'event and CLK = '1') then
    -- If reading back register data, sample at half system clock rate.
```

```
      if ( CURRENT_STATE=REPLY_WAIT or CURRENT_STATE=READ0 or CURRENT_STATE=READ1
        or CURRENT_STATE=READ2 or CURRENT_STATE=READ3) then
        if ( EN_READ_CLK <= '0' ) then
          GCFE_SAMPLED_DATA <= GCFE_DATA;
        else
          GCFE_SAMPLED_DATA <= GCFE_SAMPLED_DATA;
        end if;
      else
        -- Read data at full clock rate.
        GCFE_SAMPLED_DATA <= GCFE_DATA;
      end if;
    end if;
  end process;

  GCFE_DATA_MUX: process(GCFE_ADDR_FA_REG, RIGHT_FIRST, GCFE_SAMPLED_DATA)
  begin
    if ( RIGHT_FIRST = '0' ) then -- No reversing of addressing.
    case GCFE_ADDR_FA_REG(8 downto 5) is  --  Register bits containing GCFE address.
      when "0000" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(0) ;
      when "0001" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(1) ;
      when "0010" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(2) ;
      when "0011" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(3) ;
      when "0100" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(4) ;
      when "0101" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(5) ;
      when "0110" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(6) ;
      when "0111" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(7) ;
      when "1000" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(8) ;
      when "1001" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(9) ;
      when "1010" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(10) ;
      when "1011" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(11) ;
      when others =>
      GCFE_RTN_DATA <= '0' ;
    end case;
    else -- Right_first = 1, swithch addressing.
    case GCFE_ADDR_FA_REG(8 downto 5) is
      when "0000" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(11) ;
      when "0001" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(10) ;
      when "0010" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(9) ;
      when "0011" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(8) ;
      when "0100" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(7) ;
      when "0101" =>
      GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(6) ;
```

```
        when "0110" =>
        GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(5) ;
        when "0111" =>
        GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(4) ;
        when "1000" =>
        GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(3) ;
        when "1001" =>
        GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(2) ;
        when "1010" =>
        GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(1) ;
        when "1011" =>
        GCFE_RTN_DATA <= GCFE_SAMPLED_DATA(0) ;
        when others =>
        GCFE_RTN_DATA <= '0' ;
      end case;
      end if; -- If Right_first.
   end process;

   -- GCFE Data is sent to GCRC_Central module
   -- Ver 5.3  If Half_Clock readback mode, send data from registers
   GCFE_RTN_DATA_DELAY: process (NRESET, CURRENT_STATE, RETURN_DATA0, RETURN_DATA1 )
   begin
      if (NRESET = '0') then
         GCFE_DELAYED_RTN_DATA <= '0';
      else
         case CURRENT_STATE is
         when READ1 =>
         GCFE_DELAYED_RTN_DATA <= RETURN_DATA0(3);

         when READ2 =>
         GCFE_DELAYED_RTN_DATA <= RETURN_DATA1(7);

         when READ3 =>
         GCFE_DELAYED_RTN_DATA <= RETURN_DATA0(3);

         when others =>
         GCFE_DELAYED_RTN_DATA <= '0';

         end case;
      end if;
   end process;

   -- Ver 5.3 Added.  Put GCFE reply data into one of two shift registers.
   --   First shift register can be sending data back to the TEM at full system clock
rate
   --   while the second register is storing the GCFE data at a slower clock rate.
   GCFE_DATA_REGISTER: process (NRESET, CLK)
   begin
      if (NRESET = '0') then
         RETURN_DATA0 <= (others => '0');
         RETURN_DATA1 <= (others => '0');
      elsif ( CLK'event and CLK = '1' ) then
           case CURRENT_STATE is
           -- Read 3 GCFE return data bits into Return_Data0.
           when READ0 =>
           if ( EN_READ_CLK = '0') then
              RETURN_DATA0(3) <= RETURN_DATA0(2);
              RETURN_DATA0(2) <= GCFE_RTN_DATA;
              RETURN_DATA0(1 downto 0) <= (others => '0');
```

```
          else
            RETURN_DATA0 <= RETURN_DATA0;
          end if;
          RETURN_DATA1 <= (others => '0');

          -- Read 9 GCFE return data bits into Return_Data1, Shift bits in Return_Data0
to Central Module.
          when READ1 =>
          RETURN_DATA0(3 downto 1) <= RETURN_DATA0(2 downto 0);
          RETURN_DATA0(0) <= '0';
          if ( EN_READ_CLK = '0' ) then
            RETURN_DATA1(7 downto 1) <= RETURN_DATA1(6 downto 0);
            RETURN_DATA1(0) <= GCFE_RTN_DATA;
          else
            RETURN_DATA1 <= RETURN_DATA1;
          end if;

          -- Read 4 GCFE return data bits in Return_Data0, Shift bits in Return_data1 to
Central Module.
          when READ2 =>
          IF ( EN_READ_CLK = '0' ) then
            RETURN_DATA0(3 downto 1) <= RETURN_DATA0(2 downto 0);
            RETURN_DATA0(0) <= GCFE_RTN_DATA;
          else
            RETURN_DATA0 <= RETURN_DATA0;
          end if;
          RETURN_DATA1(7 downto 1) <= RETURN_DATA1(6 downto 0);
          RETURN_DATA1(0) <= '0';

          -- Finish transferring data, while read in last 2 GCFE returned data bits.
          when READ3 =>
          if ( COUNTER = "01110" and EN_READ_CLK = '0') then
            RETURN_DATA0(1) <= GCFE_RTN_DATA;
          else
            RETURN_DATA0(1) <= RETURN_DATA0(0);
          end if;
          if ( COUNTER = "01111" and EN_READ_CLK = '0') then
            RETURN_DATA0(2) <= GCFE_RTN_DATA;
          else
            RETURN_DATA0(2) <= RETURN_DATA0(1);
          end if;
          RETURN_DATA0(3) <= RETURN_DATA0(2);
          RETURN_DATA0(0) <= '0';
          RETURN_DATA1 <= (others => '0');

          when others =>
          RETURN_DATA0 <= (others => '0');
          RETURN_DATA1 <= (others => '0');
          end case;
     end if;
  end process;

  CNTRL_COUNTER: process (NRESET, CLK)
  begin
     if (NRESET = '0')then
        COUNTER <= (others => '0');
     elsif(CLK'event and CLK = '1')then
        -- If reading GCFE data back at half clock rate, do not count every-other system
clk.
```

```
      if ( not((CURRENT_STATE= REPLY_WAIT or CURRENT_STATE = READ0 or
CURRENT_STATE=READ1 or
         CURRENT_STATE=READ2 or CURRENT_STATE=READ3) and EN_READ_CLK = '1') ) then
         if (COUNTER_EN = '1')then
           COUNTER <= COUNTER + 1;
         else
           COUNTER <= (others => '0') ;
         end if;
      else
         COUNTER <= COUNTER;
      end if;
    end if;
  end process;

  -- Transmitting TEM commanded address, function and data to the GCFEs.
  -- Data bits, TEM_CAMD_DATA_MSB, are shifted in from GCRC_CENTRAL_CONTROL module.
  -- For reading GCFEs, data bits sent directly to GCRC_CENTRAL_CONTROL module.
  DATA_CMD: process(NRESET, CLK)
  begin
    if (NRESET = '0') then
           GCFE_CMD <= '0';
      GCFE_ADDR_FA_REG <= (others => '0');
    elsif( CLK'event and CLK = '1') then
      case CURRENT_STATE is
         when RESET_ENV =>
         GCFE_CMD <= '0' ;
         GCFE_ADDR_FA_REG <= GCFE_ADDR & R_W & FUNC; -- Grab address and function bits.

         when START_BIT =>
         GCFE_CMD <= '1' ;    --start bit to GCFE
         GCFE_ADDR_FA_REG <= GCFE_ADDR_FA_REG;

         when WRITE =>
         GCFE_CMD <= GCFE_ADDR_FA_REG(8) ;
         if ( NEXT_STATE = REPLY_WAIT ) then  -- Relatch GCFE address for data mux.
           GCFE_ADDR_FA_REG <= GCFE_ADDR & R_W & FUNC ;
         else
           GCFE_ADDR_FA_REG(8 downto 1) <= GCFE_ADDR_FA_REG(7 downto 0);
           GCFE_ADDR_FA_REG(0) <= TEM_CMD_DATA_MSB;
         end if;

         when others =>
         GCFE_CMD <= '0';
         GCFE_ADDR_FA_REG <= GCFE_ADDR_FA_REG;
      end case;
    end if;
  end process;

  -- Send data to GCFE on falling system clock.  No reset condition so implemented in
Xilinx I/O module.
  -- Half clock cycle delay between GCFE_CMD and CMD.
  DATA_OUT: process(CLK)
  begin
    if ( CLK'event and CLK = '0') then
      CMD <= GCFE_CMD;
    end if;
  end process;
```

```
end GCFE_CNTRL_ENTITY;
```

5.6 GCRC_CALIB.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_CALIB.vhd
--
--Created by Jim Ampe / Michael Condon, Naval Reserach Lab
--
--Revision:   1.0 (11-12-01)
--            First version.
--       4.0 (3-5-02) Remove 3.2 usec calibration pulse delay.
--       5.0  No change.
--       5.1 No change.
--       5.2 No Change.
--       5.3 No Change.
--
--Function: Asserts the CALIB line to the GCFE's for a fixed pulse width, when
commanded by
--     the TEM.
--
-----------------------------------------------------------------------------------------

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity GCRC_CALIB is
  port (
    NRESET             : in std_logic;   -- Chipwide RESET, asserted low.
    CLK              : in std_logic;   -- 20 MHz System Clock.
    CALIB_CMD      : in std_logic;   -- Decode Calibration command from TEM.
    CALIB_STRB       : buffer std_logic  -- Calibration line to the GCFEs.
    );
end GCRC_CALIB;

architecture GCRC_CALIB of GCRC_CALIB is

  constant CALIB_PULSE             : std_logic_vector(6 downto 0) := "1111111";
  --constant CALIB_DELAY         : std_logic_vector(6 downto 0) := "0111111";
  constant IDLE             : std_logic := '0';
  constant PULSE            : std_logic := '1';
  signal CALIB_COUNTER         : std_logic_vector(6 downto 0);
  signal CALIB_COUNTER_EN        : std_logic;
  signal NEXT_STATE, CURRENT_STATE  : std_logic;

begin

  FSM_STATE: process (CURRENT_STATE, CALIB_CMD, CALIB_COUNTER)is
  begin
    case CURRENT_STATE is
      when IDLE =>
      CALIB_COUNTER_EN <= '0';
      CALIB_STRB <= '0';
      if ( CALIB_CMD = '1') then
        NEXT_STATE <= PULSE;
      else
        NEXT_STATE <= IDLE;
      end if;
```

```
        when PULSE =>
        CALIB_COUNTER_EN <= '1';
        CALIB_STRB <= '1';
        if ( CALIB_COUNTER = CALIB_PULSE) then
          NEXT_STATE <= IDLE;
        else
          NEXT_STATE <= PULSE;
        end if;

        when others =>
        CALIB_COUNTER_EN <= '0';
        CALIB_STRB <= '0';
        NEXT_STATE <= IDLE;

      end case;
    end process;


    FSM_CONTROL: process (NRESET, CLK)is
    begin
      if (NRESET = '0')then
        CURRENT_STATE <= IDLE;
      elsif(CLK'event and CLK = '1') then
        CURRENT_STATE <= NEXT_STATE;
      end if;
    end process;


    CALIB_COUNT: process (NRESET, CLK)is
    begin
      if (NRESET = '0')then
        CALIB_COUNTER <= (others => '0');
      elsif(CLK'event and CLK = '1')then
        if ( CALIB_COUNTER_EN = '0' ) then
          CALIB_COUNTER <= (others => '0') ;
        else
          CALIB_COUNTER <= CALIB_COUNTER + 1 ;
        end if;
      end if;
    end process;

end GCRC_CALIB;
```

5.7 GCRC_ADC.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_ADC.vhd
--
--Created by Jim Ampe / Michael Condon, Naval Research Lab
--
--Revision:  1.0 (11-12-01)
--          First Version
--     5.0  No change
--     5.1 No change.
--     5.2 No Change.
--     5.3 No Change.
--
--Function: Controls the state of the onboard ADCs.  Does not readout the ADC data.
--
--
-----------------------------------------------------------------------------------------

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity GCRC_ADC is
    port (
     NRESET             : in std_logic;     --Chipwide RESET, asserted low
     CLK              : in std_logic;     --20 MHz System Clock
     ADC_SAMPLE       : in std_logic;     -- Notification to begin ADC conversion, from
GCRC_READOUT
     ADC_READ_DONE    : in std_logic;    -- ADC data has been read, from GCRC_READOUT
     ADC_CONV_TIME_REG  : in std_logic_vector(7 downto 0);  -- ADC conversion time
stored in register
     ADC_READ_BEGIN   : buffer std_logic; -- Begin ADC readout, to GCRC_READOUT
     ADC_NCS            : buffer std_logic; -- Chip Sel line to Analog to Digital
Converter.
     ADC_SAMPLE_CLK   : buffer std_logic  --Clock to the ADC, filtered by GCRC_READOUT
       );
end GCRC_ADC;

architecture GCRC_ADC of GCRC_ADC is

constant SLEEP_TIME     : std_logic_vector(2 downto 0) := "011";
constant WAKE_TIME      : std_logic_vector(5 downto 0) := "110010";--2.5 usec min wake
time
constant SLEEP          : std_logic_vector(1 downto 0) := "00";
constant WAKE           : std_logic_vector(1 downto 0) := "01";
constant CONV           : std_logic_vector(1 downto 0) := "10";
constant TRANS          : std_logic_vector(1 downto 0) := "11";
signal COUNTER          : std_logic_vector(7 downto 0); --8 bit binary counter
signal COUNTER_EN     : std_logic; -- counter enable
signal COUNTER_RESET  : std_logic; -- counter RESET
signal current_state, next_state : std_logic_vector(1 downto 0);--States of FSM

begin
    FSM_STATES: process (CURRENT_STATE, COUNTER, ADC_SAMPLE, ADC_READ_DONE,
ADC_CONV_TIME_REG)is
    begin
```

```
 case current_state is

     when SLEEP =>  -- State 0
     ADC_NCS <= '1';
     ADC_READ_BEGIN <= '0' ;
     ADC_SAMPLE_CLK <= '1' ;
if (COUNTER = SLEEP_TIME ) then
   NEXT_STATE <= WAKE ;
   COUNTER_EN <= '0' ;
   COUNTER_RESET <= '1' ;
else
   NEXT_STATE <= SLEEP ;
   COUNTER_EN <= '1' ;
   COUNTER_RESET <= '0' ;
end if;

     when WAKE =>   --  State 1
     ADC_NCS <= '0' ;
     ADC_READ_BEGIN <= '0' ;
     ADC_SAMPLE_CLK <= '1' ;
if ( COUNTER = WAKE_TIME ) then
   if ( ADC_SAMPLE = '1' ) then
     NEXT_STATE <= CONV ;
     COUNTER_EN <= '0' ;
     COUNTER_RESET <= '1' ;
   else
     NEXT_STATE <= WAKE ;
     COUNTER_EN <= '0' ;
     COUNTER_RESET <= '0' ;
   end if;
else
   NEXT_STATE <= WAKE;
   COUNTER_EN <= '1';
   COUNTER_RESET <= '0';
end if;

     when CONV =>   --  State 2
     ADC_NCS <= '0' ;
ADC_READ_BEGIN <= '0' ;
     ADC_SAMPLE_CLK <= '0' ;
if ( COUNTER = ADC_CONV_TIME_REG ) then
   NEXT_STATE <= TRANS ;
   COUNTER_EN <= '0';
   COUNTER_RESET <= '1' ;
else
   NEXT_STATE <= CONV ;
   COUNTER_EN <= '1';
   COUNTER_RESET <= '0' ;
end if;

     when TRANS =>   --  State 3
     ADC_NCS <= '0' ;
     ADC_SAMPLE_CLK <= '0' ;
     COUNTER_EN <= '0';
     COUNTER_RESET <= '1' ;
if ( ADC_READ_DONE = '1' ) then
   NEXT_STATE <= SLEEP ;
   ADC_READ_BEGIN <= '0' ;
```

```
       else
         NEXT_STATE <= TRANS ;
         ADC_READ_BEGIN <= '1' ;
       end if;

            when others =>
            ADC_NCS <= '1' ;
            ADC_SAMPLE_CLK <= '1' ;
       ADC_READ_BEGIN <= '0' ;
            NEXT_STATE <= SLEEP ;
       COUNTER_EN <= '0';
       COUNTER_RESET <= '1' ;
        end case;
    end process;

    FSM_CHANGE: process (CLK, NRESET)is
    begin
        if (NRESET = '0')then
            CURRENT_STATE <= SLEEP ;
        elsif(CLK'event and CLK = '1')then
       CURRENT_STATE <= NEXT_STATE ;
     end if;
  end process;

    TIME_COUNTER: process (NRESET, CLK)is
    begin
        if (NRESET = '0')then
            COUNTER <= (others => '0');
        elsif(CLK'event and CLK = '1')then
       if (COUNTER_RESET = '1') then
         COUNTER <= (others => '0') ;
       elsif ( COUNTER_EN = '1' ) then
         COUNTER <= COUNTER + 1 ;
       else
         COUNTER <= COUNTER;
       end if;
     end if;
  end process;


end GCRC_ADC;
```

5.8 GCRC_READOUT.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_READOUT.vhd
--
--Created by Jim Ampe / Michael Condon, Naval Research Lab.
--
--Revision: 1.0 (11-12-01)
--           First Version.
--     1.0 Rev B (11-14-01),  Range readout skipped ADC LSB, is fixed.
--     3.0 (2-13-02), latch in log accept bits after they are output by GCFEs.
--       Also deassert GCFE Resest line following last Range Bit read.
--     5.0 (11-08-02) Add another clock cycle delay, a minimum two clock cycles delay
       between asserting DAQ_CLK to GCFE and reading back data from GCFE chip.
--     5.1 No Change.
--     5.2 No Change.
--     5.3 No Change.
--
--
--Function: Controls readout state machine of GCFE, collects range data from GCFE
--        and ADC pulse height data, serial transmits bits to the TEM.
--
--
-----------------------------------------------------------------------------------------

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity GCRC_READOUT is
    port (
     NRESET           : in std_logic;  --Chipwide RESET, asserted low
     CLK             : in std_logic; --20 MHz System Clock
     TRIGGER     : in std_logic; -- Readout command has been received
     TRIGGER_MODE   : in std_logic;  --- "0" is 1 range readout, "1" is 4 range readout"
     GCFE_SAMPLED_DATA : in std_logic_vector (11 downto 0); -- Clocked data from 12
GCFE's
     ADC_DATA        : in std_logic_vector (11 downto 0); -- Serial data from 12 ADc's
     ADC_READ_BEGIN : in std_logic; -- Begin ADC readout, from GCRC_ADC module
     ADC_SAMPLE_CLK  : in std_logic;  -- ADC clock from GCRC_ADC module
     RANGE_DELAY_REG  : in std_logic_vector (5 downto 0);
     ADC_SAMPLE_DELAY_REG : in std_logic_vector (5 downto 0);
     CONFIG_PARITY  : in std_logic;    -- Register parity mode, '0' is odd parity.
     GCFE_CLK_ENV   : buffer std_logic; -- GCFE clock envelope
     START_ACQ   : buffer std_logic; -- GCFE readout state machine start signal.
     GCFE_RESET_ENV : buffer std_logic; -- GCFE command envelope
     ADC_READ_DONE  : buffer std_logic;-- ADC has completed readout, to GCRC_ADC module
     ADC_SAMPLE    : buffer std_logic; --  Begin ADC conversion, to GCRC_ADC module
     ADC_CLK     : buffer std_logic;-- ADC Clock to ADC chips
     RANGE_DATA_OUT0  : buffer std_logic;-- Combined ADC and GCFE data to the TEM
     RANGE_DATA_OUT1  : buffer std_logic;-- Combined ADC and GCFE data to the TEM
     GCRC_IN_READOUT  : buffer std_logic  -- GCRC is perfoming range readout, to
GCRC_TRIG module.
     );
end GCRC_READOUT;

architecture GCRC_READOUT of GCRC_READOUT is
```

```
constant IDLE           : std_logic_vector(3 downto 0) := "0000";
constant START_BIT      : std_logic_vector(3 downto 0) := "0001";
constant PEAK_HOLD        : std_logic_vector(3 downto 0) := "0010";
constant HOLD_DELAY     : std_logic_vector(3 downto 0) := "0011";
constant SEL_RANGE        : std_logic_vector(3 downto 0) := "0100";
constant LOG_ACCEPT_BIT   : std_logic_vector(3 downto 0) := "0101";
constant LOG_BIT_DELAY    : std_logic_vector(3 downto 0) := "0110";
constant LOG_ACCEPT_READ  : std_logic_vector(3 downto 0) := "0111";
constant RANGE_DELAY    : std_logic_vector(3 downto 0) := "1000";
constant CONVERSION       : std_logic_vector(3 downto 0) := "1001";
constant READOUT      : std_logic_vector(3 downto 0) := "1010";
constant READOUT_PARITY   : std_logic_vector(3 downto 0) := "1011";
-- Ver 5 Added constant for two clock cycle delay in reading of GCFE data.
constant GCFE_READ_DELAY  : std_logic_vector(3 downto 0) := "0001";

signal DATA_BUFFER0     : std_logic_vector (5 downto 0);
signal DATA_BUFFER1     : std_logic_vector (5 downto 0);
signal COUNTER            : std_logic_vector(7 downto 0);--6 bit binary counter
signal RANGE_COUNT      : std_logic_vector(1 downto 0);
signal ADC_BIT_COUNT    : std_logic_vector(4 downto 0);
signal CLK_COUNT      : std_logic_vector(2 downto 0);
signal READOUT_TRANS_BEGIN : std_logic; -- Transmission to TEM has started, place start
bit.
signal ADC_READ_CLK     : std_logic;
signal COUNTER_EN       : std_logic;
signal COUNTER_RESET    : std_logic;
signal READOUT_DONE     : std_logic;
signal GCFE_CLK_ENV1    : std_logic;
signal GCFE_CLK_ENV2    : std_logic;
signal XOR_OUT0         : std_logic;
signal LATCHED_XOR_OUT0   : std_logic;
signal PARITY_OUT0      : std_logic;
signal XOR_OUT1         : std_logic;
signal LATCHED_XOR_OUT1   : std_logic;
signal PARITY_OUT1      : std_logic;
signal PARITY_RESET     : std_logic;
signal LOG_ACCEPT_TRANS_BEGIN  : std_logic;
signal RANGE_BIT_TRANS_BEGIN : std_logic;
signal ADC_DATA_TRANS_BEGIN    : std_logic;
signal LOG_ACCEPT_TRANS   : std_logic;
signal RANGE_BIT_TRANS    : std_logic;
signal ADC_DATA_TRANS   : std_logic;
signal LATCHED_TRIGGER_MODE : std_logic; -- Register to remember if 4 range readout.
signal current_state, next_state : std_logic_vector(3 downto 0);--States of FSM

begin

    FSM_STATES: process (CURRENT_STATE, COUNTER, TRIGGER, LATCHED_TRIGGER_MODE,
  ADC_READ_BEGIN, READOUT_DONE, ADC_SAMPLE_DELAY_REG, RANGE_DELAY_REG, RANGE_COUNT)is
    begin
        case CURRENT_STATE is
            when IDLE =>    -- STATE 0
       GCFE_CLK_ENV1 <= '0';
       START_ACQ <= '0' ;
       ADC_SAMPLE <= '0' ;
       COUNTER_EN <= '0' ;
       COUNTER_RESET <= '1' ;
       if ( TRIGGER = '1' ) then
```

```
      NEXT_STATE <= START_BIT ;
    else
      NEXT_STATE <= IDLE ;
    end if;

    when START_BIT =>    -- STATE 1
    GCFE_CLK_ENV1 <= '0'  ;
    ADC_SAMPLE <= '0' ;
    COUNTER_EN <= '0' ;
    COUNTER_RESET <= '1' ;
    START_ACQ <= '1' ;
    NEXT_STATE <= PEAK_HOLD ;

    when PEAK_HOLD =>  -- STATE 2
    START_ACQ <= '1' ;
    GCFE_CLK_ENV1 <= '1' ;
    ADC_SAMPLE <= '0' ;
    COUNTER_EN <= '0' ;
    COUNTER_RESET <= '1' ;
    NEXT_STATE <= HOLD_DELAY ;

    when HOLD_DELAY =>   -- STATE 3
    START_ACQ <= '0' ;
    GCFE_CLK_ENV1 <= '0' ;
    ADC_SAMPLE <= '0' ;
    if ( COUNTER = RANGE_DELAY_REG ) then
      NEXT_STATE <= SEL_RANGE ;
      COUNTER_EN <= '0' ;
      COUNTER_RESET <= '1' ;
    else
      NEXT_STATE <= HOLD_DELAY ;
      COUNTER_EN <= '1' ;
      COUNTER_RESET <= '0' ;
    end if;

    when SEL_RANGE =>    -- STATE 4
    START_ACQ <= '0' ;
    GCFE_CLK_ENV1 <= '1' ;
    ADC_SAMPLE <= '0' ;
    COUNTER_EN <= '0' ;
    COUNTER_RESET <= '1' ;
    NEXT_STATE <= LOG_ACCEPT_BIT ;

    when LOG_ACCEPT_BIT =>      -- STATE 5
    START_ACQ <= '0' ;
    GCFE_CLK_ENV1 <= '1' ;
    ADC_SAMPLE <= '0' ;
    COUNTER_EN <= '0' ;
    COUNTER_RESET <= '1' ;
    NEXT_STATE <= LOG_BIT_DELAY;

    -- Ver 5, changed 1 clock cycle delay to 2 clock cycle delay.
    when LOG_BIT_DELAY =>     -- STATE 6
    START_ACQ <= '0' ;
    GCFE_CLK_ENV1 <= '0' ;
    ADC_SAMPLE <= '0' ;
    if ( COUNTER = GCFE_READ_DELAY ) then
      NEXT_STATE <= LOG_ACCEPT_READ;
      COUNTER_EN <= '0' ;
```

```
      COUNTER_RESET <= '1' ;
    else
      NEXT_STATE <= LOG_BIT_DELAY;
      COUNTER_EN <= '1' ;
      COUNTER_RESET <= '0' ;
    end if;

    when LOG_ACCEPT_READ =>     -- STATE 7
    START_ACQ <= '0' ;
    GCFE_CLK_ENV1 <= '0' ;
    ADC_SAMPLE <= '0' ;
    COUNTER_EN <= '0' ;
    COUNTER_RESET <= '1' ;
    NEXT_STATE <= RANGE_DELAY ;

    when RANGE_DELAY =>   -- STATE 8
    START_ACQ <= '0' ;
    GCFE_CLK_ENV1 <= '0' ;
    ADC_SAMPLE <= '0' ;
    if ( COUNTER = ADC_SAMPLE_DELAY_REG ) then
      NEXT_STATE <= CONVERSION ;
      COUNTER_EN <= '0' ;
      COUNTER_RESET <= '1' ;
    else
      NEXT_STATE <= RANGE_DELAY ;
      COUNTER_EN <= '1' ;
      COUNTER_RESET <= '0' ;
    end if;

    when CONVERSION =>    -- STATE 9
    START_ACQ <= '0' ;
    GCFE_CLK_ENV1 <= '0' ;
    COUNTER_EN <= '0' ;
    COUNTER_RESET <= '1' ;
    if ( ADC_READ_BEGIN = '1') then
      NEXT_STATE <= READOUT ;
      ADC_SAMPLE <= '0' ;
    else
      NEXT_STATE <= CONVERSION ;
      ADC_SAMPLE <= '1' ;
    end if;


    when READOUT =>      -- STATE 10
    START_ACQ <= '0' ;
    ADC_SAMPLE <= '0' ;
    COUNTER_EN <= '0' ;
    COUNTER_RESET <= '1' ;
    GCFE_CLK_ENV1 <= '0';
    if ( READOUT_DONE = '1') then
      NEXT_STATE <= READOUT_PARITY ;
    else
      NEXT_STATE <= READOUT;
    end if;

    when READOUT_PARITY =>     -- STATE 11
    START_ACQ <= '0' ;
    ADC_SAMPLE <= '0' ;
    COUNTER_EN <= '0' ;
```

```
      COUNTER_RESET <= '1' ;
      GCFE_CLK_ENV1 <= '0';
      if (LATCHED_TRIGGER_MODE = '0') then -- 1 RANGE READOUT.
        NEXT_STATE <= IDLE ;
      elsif (RANGE_COUNT = "11") then  -- RANGE COUNT INCREMENTED NEXT CLOCK EDGE.
        NEXT_STATE <= IDLE ;  -- COMPLETED 4 RANGE READOUT
      else
        NEXT_STATE <= CONVERSION ; -- NOT COMPLETED 4 RANGE REDOUT
      end if;

      when others =>
      START_ACQ <= '0';
      GCFE_CLK_ENV1 <= '0';
      ADC_SAMPLE <= '0';
      NEXT_STATE <= IDLE;
      COUNTER_EN <= '0';
      COUNTER_RESET <= '1';

       end case;
    end process;

  FSM_CHANGE: process (NRESET, CLK)is
  begin
      if ( NRESET = '0') then
         CURRENT_STATE <= IDLE;
       elsif(CLK'EVENT and CLK = '1')then
        CURRENT_STATE <= NEXT_STATE ;
      end if;
  end process;

  IN_READOUT: process (NRESET, CURRENT_STATE ) is
  begin
    if ( NRESET = '0' ) then
      GCRC_IN_READOUT <= '0';
    elsif ( CURRENT_STATE = IDLE ) then
      GCRC_IN_READOUT <= '0';
    else
      GCRC_IN_READOUT <= '1';
    end if;
  end process;

  LATCHED_TRIG_MODE: process (NRESET, CLK) is
  begin
    if( NRESET = '0') then
      LATCHED_TRIGGER_MODE <= '0';
    elsif( CLK'EVENT and CLK = '1') then
      if ( CURRENT_STATE = IDLE and TRIGGER = '1') then
        LATCHED_TRIGGER_MODE <= TRIGGER_MODE ;
      else
        LATCHED_TRIGGER_MODE <= LATCHED_TRIGGER_MODE;
      end if;
    end if;
  end process;

  RANGE_COUNTER: process (NRESET, CLK)
  begin
    if (NRESET = '0') then
      RANGE_COUNT <= (others => '0');
    elsif ( CLK'EVENT and CLK = '1') then
```

```
      if ( CURRENT_STATE = IDLE ) then
        RANGE_COUNT <= (others => '0');
      elsif ( CURRENT_STATE = READOUT_PARITY) then
        RANGE_COUNT <= RANGE_COUNT + 1 ;
      end if;
    end if;
  end process;

  -- Request Reset line to gcfe is asserted low for readout.
  --    Reset_envelope signal goes through gcrc_gcfe module, where it is sampled
  --     by the system clock.  Assert Reset High to GCFE during readout after
  --   completing last range bit reads.
  GCFE_RESET_ENVELOPE: process (NRESET, CURRENT_STATE, NEXT_STATE, ADC_DATA_TRANS,
    LATCHED_TRIGGER_MODE, RANGE_COUNT )
  begin
       if (NRESET = '0') then
           GCFE_RESET_ENV <= '1';
    elsif ( (CURRENT_STATE = IDLE and NEXT_STATE = IDLE) or
      -- End of GCFE read for single range readout.
      ( ADC_DATA_TRANS = '1' and LATCHED_TRIGGER_MODE = '0') or
      -- End of GCFE read for 4 range readout.
      ( ADC_DATA_TRANS = '1' and LATCHED_TRIGGER_MODE = '1' and RANGE_COUNT = "11")
)then
      GCFE_RESET_ENV <= '1';
    else
      GCFE_RESET_ENV <= '0';
    end if;
  end process;


   TIME_COUNTER: process (NRESET, CLK)
   begin
       if (NRESET = '0')then
           COUNTER <= (others => '0');
       elsif(CLK'EVENT and CLK = '1')then
      if (COUNTER_RESET = '1') then
        COUNTER <= (others => '0') ;
      elsif ( COUNTER_EN = '1' ) then
        COUNTER <= COUNTER + 1 ;
      else
        COUNTER <= COUNTER;
      end if;
    end if;
  end process;

  -- The MAX145 ADC uses the ADC CLK transition to first sample the analog signal
  --   then shift out the data bits.
  ADC_CLOCK: process (CURRENT_STATE, ADC_SAMPLE_CLK, ADC_READ_CLK)is
   begin
    if (CURRENT_STATE = READOUT) then
      ADC_CLK <= ADC_READ_CLK;
    else
      ADC_CLK <= ADC_SAMPLE_CLK;
    end if;
  end process;

  -- The GCFE Clock Envelope is passed to the GCRC GCFE Module to
  --   select a DAQ_CLK transition to the GCFE chips.
  GCFE_CLOCK_ENV: process (GCFE_CLK_ENV1, GCFE_CLK_ENV2)is
```

```
    begin
        if (GCFE_CLK_ENV1 = '1' or GCFE_CLK_ENV2 = '1') then
       GCFE_CLK_ENV <= '1';
     else
       GCFE_CLK_ENV <= '0';
     end if;
  end process;

  -- Process of forming the serial readout data to the TEM.
  ADC_READOUT: process (NRESET, CLK)is
    begin
     if (NRESET = '0') then
       ADC_BIT_COUNT <= (others => '0');
       CLK_COUNT <= (others => '0');
       ADC_READ_DONE <= '0';
       READOUT_DONE <= '0';
       ADC_READ_CLK <= '0';
       GCFE_CLK_ENV2 <= '0';
       READOUT_TRANS_BEGIN <= '0';
       LOG_ACCEPT_TRANS_BEGIN <= '0';
       RANGE_BIT_TRANS_BEGIN <= '0';
       ADC_DATA_TRANS_BEGIN <= '0';
     elsif (CLK'EVENT and CLK='1') then
       if (CURRENT_STATE = READOUT) then
         -- adc uses internal conversion clock, applied ADC clock clocks out data bits.
         -- MAXIM MAX145 ADC readout clock rate is 5 mhz.
         -- Interleave ADC clocking with shifting data bits out.
           if ( ADC_BIT_COUNT /= "10001" ) then
             case CLK_COUNT is

               when "000" =>
               ADC_BIT_COUNT <= ADC_BIT_COUNT;
               GCFE_CLK_ENV2 <= '0';
               READOUT_DONE <= '0';
               ADC_READ_DONE <= '0';
               READOUT_TRANS_BEGIN <= '0';
               LOG_ACCEPT_TRANS_BEGIN <= '0';
               RANGE_BIT_TRANS_BEGIN <= '0';
               ADC_DATA_TRANS_BEGIN <= '0';
               if (ADC_BIT_COUNT = "00000") then
                 ADC_READ_CLK <= '0' ;
               else
                 ADC_READ_CLK <= '1';
               end if;

               when "001" =>
               ADC_BIT_COUNT <= ADC_BIT_COUNT ;
               ADC_READ_CLK <= '0' ;
               READOUT_DONE <= '0';
               ADC_READ_DONE <= '0';
               READOUT_TRANS_BEGIN <= '0';
               LOG_ACCEPT_TRANS_BEGIN <= '0';
               RANGE_BIT_TRANS_BEGIN <= '0';
               ADC_DATA_TRANS_BEGIN <= '0';
               if ( RANGE_COUNT = "00" and ADC_BIT_COUNT = "0010" ) then
                 GCFE_CLK_ENV2 <= '1';  -- Clock GCFE range bit 1
               elsif ( ADC_BIT_COUNT = "0011") then
                 GCFE_CLK_ENV2 <= '1';  -- Clock GCFE range bit 0
               elsif ( (ADC_BIT_COUNT = "01000" or ADC_BIT_COUNT = "1001") and
```

```
   ( RANGE_COUNT <= "10") and (LATCHED_TRIGGER_MODE = '1') ) then
   GCFE_CLK_ENV2 <= '1';  -- SWITCH GCFE ANALOG MUX AND
else             --   GET RANGE BIT 1 if 4 range readout.
   GCFE_CLK_ENV2 <= '0';
end if;

when "010" =>
ADC_BIT_COUNT <= ADC_BIT_COUNT ;
ADC_READ_CLK <= '0' ;
GCFE_CLK_ENV2 <= '0';
READOUT_DONE <= '0';
ADC_READ_DONE <= '0';
READOUT_TRANS_BEGIN <= '0';
LOG_ACCEPT_TRANS_BEGIN <= '0';
RANGE_BIT_TRANS_BEGIN <= '0';
ADC_DATA_TRANS_BEGIN <= '0';

when "011" =>
ADC_BIT_COUNT <= ADC_BIT_COUNT ;
ADC_READ_CLK <= '0' ;
GCFE_CLK_ENV2 <= '0';
READOUT_DONE <= '0';
ADC_READ_DONE <= '0';
READOUT_TRANS_BEGIN <= '0';
LOG_ACCEPT_TRANS_BEGIN <= '0';
RANGE_BIT_TRANS_BEGIN <= '0';
ADC_DATA_TRANS_BEGIN <= '0';

when "100" =>
ADC_BIT_COUNT <= ADC_BIT_COUNT;
ADC_READ_CLK <= '1';
GCFE_CLK_ENV2 <= '0';
READOUT_DONE <= '0';
ADC_READ_DONE <= '0';
READOUT_TRANS_BEGIN <= '0';
case ADC_BIT_COUNT is
   when "00001" =>
   if ( RANGE_COUNT = "00" ) then
      LOG_ACCEPT_TRANS_BEGIN <= '1';
   else
      LOG_ACCEPT_TRANS_BEGIN <= '0';
   end if;
   RANGE_BIT_TRANS_BEGIN <= '0';
   ADC_DATA_TRANS_BEGIN <= '0';

   when "00010" =>
   LOG_ACCEPT_TRANS_BEGIN <= '0';
   RANGE_BIT_TRANS_BEGIN <= '1';
   ADC_DATA_TRANS_BEGIN <= '0';

   when "00100" =>
   LOG_ACCEPT_TRANS_BEGIN <= '0';
   RANGE_BIT_TRANS_BEGIN <= '0';
   ADC_DATA_TRANS_BEGIN <= '1';

   when others =>
   LOG_ACCEPT_TRANS_BEGIN <= '0';
   RANGE_BIT_TRANS_BEGIN <= '0';
   ADC_DATA_TRANS_BEGIN <= '0';
```

```
        end case; -- End case ADC bit count.

        when "101" =>
        ADC_BIT_COUNT <= ADC_BIT_COUNT + 1 ;
        ADC_READ_CLK <= '1' ;
        GCFE_CLK_ENV2 <= '0';
        LOG_ACCEPT_TRANS_BEGIN <= '0';
        RANGE_BIT_TRANS_BEGIN <= '0';
        ADC_DATA_TRANS_BEGIN <= '0';
        case ADC_BIT_COUNT is
          when "00001" =>
          if ( RANGE_COUNT = "00" ) then
            READOUT_TRANS_BEGIN <= '1';
          else
            READOUT_TRANS_BEGIN <= '0';
          end if;
          READOUT_DONE <= '0';
          ADC_READ_DONE <= '0';

          when "00010" =>
          if ( RANGE_COUNT = "00" ) then
            READOUT_TRANS_BEGIN <= '0';
          else
            READOUT_TRANS_BEGIN <= '1';
          end if;
          READOUT_DONE <= '0';
          ADC_READ_DONE <= '0';

          when "10000" =>
          READOUT_TRANS_BEGIN <= '0';
          READOUT_DONE <= '1';
          ADC_READ_DONE <= '1';

          when others =>
          READOUT_TRANS_BEGIN <= '0';
          READOUT_DONE <= '0';
          ADC_READ_DONE <= '0';
        end case; -- End case ADC bit count.

        when others => -- Others Clk Count.
        READOUT_TRANS_BEGIN <= '0';
        ADC_BIT_COUNT <= ADC_BIT_COUNT ;
        ADC_READ_CLK <= '0' ;
        GCFE_CLK_ENV2 <= '0';
        READOUT_DONE <= '0';
        ADC_READ_DONE <= '0';
        LOG_ACCEPT_TRANS_BEGIN <= '0';
        RANGE_BIT_TRANS_BEGIN <= '0';
        ADC_DATA_TRANS_BEGIN <= '0';
      end case;

      if (CLK_COUNT = "101") then
        CLK_COUNT <= "000" ;
      else
        CLK_COUNT <= CLK_COUNT + 1 ;
      end if;

    else  -- Else if ADC_BIT_COUNT = "10001".
      READOUT_TRANS_BEGIN <= '0';
```

```
              ADC_READ_DONE <= '1';
              READOUT_DONE <= '1';
              GCFE_CLK_ENV2 <= '0';
              ADC_READ_CLK <= '0';
              LOG_ACCEPT_TRANS_BEGIN <= '0';
              RANGE_BIT_TRANS_BEGIN <= '0';
              ADC_DATA_TRANS_BEGIN <= '0';
              ADC_BIT_COUNT <= (others => '0');
              CLK_COUNT <= (others => '0');
            end if;

       else  -- Current state not readout.
         ADC_BIT_COUNT <= (others => '0');
         CLK_COUNT <= (others => '0');
         READOUT_TRANS_BEGIN <= '0';
         ADC_READ_DONE <= '0';
         ADC_READ_CLK <= '0';
         READOUT_DONE <= '0';
         LOG_ACCEPT_TRANS_BEGIN <= '0';
         RANGE_BIT_TRANS_BEGIN <= '0';
         ADC_DATA_TRANS_BEGIN <= '0';

       end if; -- End if CURRENT_STATE = READOUT.
     end if; --End else if clk'event.
   end process; -- Process BIT_READOUT.


   -- Data Buffer which serializes the GCFE data and ADC data.  Data is first
   --  loaded in parallel, then shifted out to form serial data stream.
   DATA_BUF: process( NRESET, CURRENT_STATE, CLK)
   begin
     if ( NRESET = '0' ) then
          DATA_BUFFER0 <= (others => '0');
          DATA_BUFFER1 <= (others => '0');
     elsif ( CLK'event and CLK = '1') then
       if ( CURRENT_STATE = IDLE ) then
         DATA_BUFFER0 <= (others => '0');
         DATA_BUFFER1 <= (others => '0');
       -- Read into buffer log accept bit or range bits.
       elsif ( ( RANGE_BIT_TRANS = '1' and CLK_COUNT = "000") or
         (CURRENT_STATE = LOG_ACCEPT_READ ) ) then
         DATA_BUFFER0(5 downto 0) <= GCFE_SAMPLED_DATA(5 downto 0);
         DATA_BUFFER1(5 downto 0) <= GCFE_SAMPLED_DATA(11 downto 6);
       -- Read into buffer ADC bits.
       elsif ( ADC_DATA_TRANS = '1' and CLK_COUNT = "000" ) then
         DATA_BUFFER0(5 downto 0) <= ADC_DATA(5 downto 0);
         DATA_BUFFER1(5 downto 0) <= ADC_DATA(11 downto 6);
       -- Shift Buffer Data Bits.
       elsif ( (LOG_ACCEPT_TRANS = '1' or RANGE_BIT_TRANS = '1' or ADC_DATA_TRANS =
'1')
         and CLK_COUNT /= "000") then
         DATA_BUFFER0(4 downto 0) <= DATA_BUFFER0(5 downto 1) ;
         DATA_BUFFER1(5 downto 1) <= DATA_BUFFER1(4 downto 0) ;
         DATA_BUFFER0(5) <= '0';
         DATA_BUFFER1(0) <= '0';
       else
         DATA_BUFFER0(5 downto 0) <= DATA_BUFFER0(5 downto 0) ;
         DATA_BUFFER1(5 downto 0) <= DATA_BUFFER1(5 downto 0) ;
       end if;
```

```
    end if;
  end process;

  READOUT_DATA_TRANS: process( NRESET, CLK)
  begin
    if ( NRESET = '0') then
      LOG_ACCEPT_TRANS <= '0';
      RANGE_BIT_TRANS <= '0';
      ADC_DATA_TRANS <= '0';
    elsif ( CLK'event and CLK = '1') then
      if ( CURRENT_STATE /= READOUT ) then
        LOG_ACCEPT_TRANS <= '0';
        RANGE_BIT_TRANS <= '0';
        ADC_DATA_TRANS <= '0';
      elsif ( LOG_ACCEPT_TRANS_BEGIN = '1' ) then
        LOG_ACCEPT_TRANS <= '1';
        RANGE_BIT_TRANS <= '0';
        ADC_DATA_TRANS <= '0';
      elsif ( RANGE_BIT_TRANS_BEGIN = '1' ) then
        LOG_ACCEPT_TRANS <= '0';
        RANGE_BIT_TRANS <= '1';
        ADC_DATA_TRANS <= '0';
      elsif ( ADC_DATA_TRANS_BEGIN = '1' ) then
        LOG_ACCEPT_TRANS <= '0';
        RANGE_BIT_TRANS <= '0';
        ADC_DATA_TRANS <= '1';
      else
        LOG_ACCEPT_TRANS <= LOG_ACCEPT_TRANS;
        RANGE_BIT_TRANS <= RANGE_BIT_TRANS;
        ADC_DATA_TRANS <= ADC_DATA_TRANS;
      end if;
    end if;
  end process;


  -- Output start bit, range readout data, or overall parity bit.
  DATA_OUT: process( NRESET, CLK)
  begin
    if ( NRESET = '0') then
      RANGE_DATA_OUT0 <= '0';
      RANGE_DATA_OUT1 <= '0';
    elsif ( CLK'EVENT and CLK = '1') then
      case CURRENT_STATE is
        when READOUT =>
        if ( READOUT_TRANS_BEGIN = '1' ) then
          RANGE_DATA_OUT0 <= '1' ; -- START BIT.
          RANGE_DATA_OUT1 <= '1' ; -- START BIT.
        elsif ( LOG_ACCEPT_TRANS= '1' or RANGE_BIT_TRANS= '1' or ADC_DATA_TRANS= '1' )
then
          RANGE_DATA_OUT0 <= DATA_BUFFER0(0) ; -- BUFFER0 SHIFTED RIGHT.
          RANGE_DATA_OUT1 <= DATA_BUFFER1(5) ; -- BUFFER1 SHIFTED LEFT.
        else
          RANGE_DATA_OUT0 <= '0';
          RANGE_DATA_OUT1 <= '0';
        end if;

        when READOUT_PARITY =>
        RANGE_DATA_OUT0 <= PARITY_OUT0 ;
        RANGE_DATA_OUT1 <= PARITY_OUT1 ;
```

```
          when others =>
          RANGE_DATA_OUT0 <= '0';
          RANGE_DATA_OUT1 <= '0';
        end case;
      end if;
   end process;

   -- Reset parity computation .
   PAR_RESET: process( NRESET, CLK)
   begin
      if ( NRESET = '0') then
        PARITY_RESET <= '1';
      elsif ( CLK'EVENT and CLK = '1' ) then
        if ( CURRENT_STATE = IDLE or READOUT_TRANS_BEGIN = '1') then
          PARITY_RESET <= '1';
        else
          PARITY_RESET <= '0';
        end if;
      end if;
   end process;

   -- Data stream 0 parity . invert parity_out if odd parity (config parity = 0).
   PAR_OUT0: process( NRESET, CLK, XOR_OUT0, CONFIG_PARITY, LATCHED_XOR_OUT0,
RANGE_DATA_OUT0,
   PARITY_RESET )
   begin
      XOR_OUT0 <= (RANGE_DATA_OUT0 xor LATCHED_XOR_OUT0) and (not PARITY_RESET) ;
      PARITY_OUT0 <= XOR_OUT0 xnor CONFIG_PARITY;
      if (NRESET = '0') then
        LATCHED_XOR_OUT0 <= '0';
      elsif ( CLK'EVENT and CLK = '1') then
        if( PARITY_RESET = '1') then
          LATCHED_XOR_OUT0 <= '0' ;
        else
          LATCHED_XOR_OUT0 <= XOR_OUT0 ;
        end if;
      end if;
   end process;

   -- Data stream 1 parity. invert parity_out if odd parity (config parity = 0).
   PAR_OUT1: process( NRESET, CLK, XOR_OUT1, CONFIG_PARITY, LATCHED_XOR_OUT1,
RANGE_DATA_OUT1,
   PARITY_RESET )
   begin
      XOR_OUT1 <= (RANGE_DATA_OUT1 xor LATCHED_XOR_OUT1) and (not PARITY_RESET) ;
      PARITY_OUT1 <= XOR_OUT1 xnor CONFIG_PARITY;
      if ( NRESET = '0') then
        LATCHED_XOR_OUT1 <= '0';
      elsif ( CLK'EVENT and CLK = '1') then
        if( PARITY_RESET = '1') then
          LATCHED_XOR_OUT1 <= '0' ;
        else
          LATCHED_XOR_OUT1 <= XOR_OUT1 ;
        end if;
      end if;
   end process;
```

```
end GCRC_READOUT;
```

5.9 GCRC_TRIG.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_TRIG.vhd
--
--Created by Jim Ampe / Michael Condon, Naval Research Lab
--
--Revision:  1.0 (11-12-01)
--           First version.
--      2.0 Rev A (1-14-02) Change GCRC_TRANS to being utilized.
--        Also removed negative clk edge trigger detect flip flops.
--      2.0 Rev B (1-16-02) Modified trigger inhibit logic that removes the inhibit
--        only after the incoming LE and HE triggers have been at the state of
--        not triggered for a period of ~15 clocks.  This replaced a counter which
--        previously determined when to remove the trigger inhibit.
--     3.0 (2-20-02) Changed LE and HE trigger input signal names to indicate
--        asserted low signals.  Modify the trigger inhibit not to ignore triggers
--        when Calib_Strobe is asserted high and have not entered Readout.
--     5.0 (10-7-02) Removed trigger pulse stretching and inhibiting.
--     5.1 No Change.
--     5.2 No Change.
--     5.3 No change.
--
--Function: Accepts wired 'OR'ed triggers from the GCFEs and transmits trigger requests
(TREQs) to the TEM.
--
--
-------------------------------------------------------------------------------------


Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity GCRC_TRIG is
    port (
     NRESET             : in std_logic; --Chipwide RESET, asserted low
     CLK              : in std_logic; --20 MHz System Clock
     LE_NDISC       : in STD_LOGIC; -- Wired 'OR' low energy trigger from GCFEs,
asserted low.
     HE_NDISC       : in STD_LOGIC;  -- Wired 'OR' high energy trigger from GCFEs,
asserted low.
     CAL_NTREQLE      : buffer STD_LOGIC; -- Low energy trigger request to the TEM.
     CAL_NTREQHE      : buffer STD_LOGIC -- High energy trigger request to the TEM.
          );
end GCRC_TRIG;

architecture GCRC_TRIG of GCRC_TRIG is

signal HE_TRIG : std_logic;
signal LE_TRIG  : std_logic;

begin
  --  Clock sampled trigger line.
    TRIG_DET: process (NRESET, CLK)
    begin
       if (NRESET = '0')then
```

```
       HE_TRIG <= '1';
       LE_TRIG <= '1';
        elsif(CLK'event and CLK = '1')then
       HE_TRIG <= HE_NDISC;
       LE_TRIG <= LE_NDISC;
     end if;
    end process;

  -- Double clock sampling to reduce metastability.
  TRIG_REQUEST: process (NRESET, CLK)
    begin
       if (NRESET = '0')then
       CAL_NTREQLE <= '1';
       CAL_NTREQHE <= '1';
        elsif(CLK'event and CLK = '1')then
       CAL_NTREQLE <= LE_TRIG;
       CAL_NTREQHE <= HE_TRIG;
     end if;
    end process;

end GCRC_TRIG;
```

5.10 GCRC_TEM.VHD

```
-----------------------------------------------------------------------------------------
--Project name: GLAST GCRC
--
--Filename: GCRC_TEM.vhd
--
--Created by Jim Ampe / Michael Condon, Naval Research Lab
--
--Revision: 1.0 (11-12-01)
--           First version
--     5.0 (9-10-02) Added hard reset deglitch circuit and commanded reset function.
--     5.1 No Change.
--     5.2 No Change.
--     5.3 No Change.
--
--Function:  Handles inversion of data signals to and from the TEM.  Also switches
readout
--        data line depending on Right_First signal.  Performs Reset signal deglitching.
--
-------------------------------------------------------------------------------------
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity GCRC_TEM is
  port (
     GCRC_DATA_OUT0 : in std_logic; -- Command reply data from GCRC_CENTRAL module.
     GCRC_DATA_OUT1 : in std_logic; -- Command reply data from GCRC_CENTRAL module.
     RANGE_DATA_OUT0  : in std_logic; -- Range readout data from GCRC_READOUT module.
     RANGE_DATA_OUT1  : in std_logic; -- Range readout data from GCRC_READOUT module.
     CAL_NCMD    : in std_logic; -- Command line to TEM.
     CAL_CLK     : in std_logic; -- System clock from TEM
     CAL_NRESET    : in std_logic; -- System reset from TEM, asserted low.
     RIGHT_FIRST   : in std_logic;  -- Readout left-right switch control.
     RESET_CMD   : in std_logic; -- Commanded reset from TEM.
     CAL_NDATA0    : buffer std_logic; -- Output data line to TEM
     CAL_NDATA1    : buffer std_logic; -- Output data line to TEM
     TEM_CMD       : buffer std_logic; -- TEM command to GCRC_CENTRAL module.
     CLK         : buffer std_logic;-- Feedthrough clock.
     NRESET      : buffer std_logic -- System reset.
        );
end GCRC_TEM;

architecture GCRC_TEM of GCRC_TEM is

signal RESET_SR         : std_logic_vector( 3 downto 0);
signal RESET_TIMER      : std_logic_vector( 3 downto 0);
signal HARD_RESET       : std_logic;
signal TIMED_RESET      : std_logic;

begin

  TEM_CMD <= not CAL_NCMD;
  CAL_NDATA0 <= not (GCRC_DATA_OUT0 or ((not RIGHT_FIRST) and RANGE_DATA_OUT0) or
     (RIGHT_FIRST and RANGE_DATA_OUT1) );
  CAL_NDATA1 <= not (GCRC_DATA_OUT1 or ((not RIGHT_FIRST) and RANGE_DATA_OUT1) or
     (RIGHT_FIRST and RANGE_DATA_OUT0) );
   CLK <= CAL_CLK;
-- NRESET <= CAL_NRESET;  -- Took out direct hard reset for Ver 5.
```

```
  --  Ver 5 added, Added hard reset deglitching circuit, similar to used in GLAST
Tracker chip.
  --   Asserts chip reset on edge transition of TEM Reset signal.
    hard_reset_detect: process (CLK)
    begin
     if(CLK'event and CLK = '1') then
        RESET_SR(0) <= CAL_NRESET;
        RESET_SR(3 downto 1) <= RESET_SR(2 downto 0);
           if ( RESET_SR = "1000") then  -- Reset Detect
           HARD_RESET <= '1';
        else
           HARD_RESET <= '0';
        end if;
     end if;
    end process;

  -- Ver 5 added, Make sure no circuit glitching initiates timed_reset.
  timed_rset: process(CLK)
  begin
    if(CLK'event and CLK = '1') then
      if ( RESET_CMD = '1' or HARD_RESET = '1' ) then
        TIMED_RESET <= '1';
      elsif ( RESET_TIMER = "1111") then
        TIMED_RESET <= '0';
      else
        TIMED_RESET <= TIMED_RESET;
      end if;
    end if;
  end process;

  -- Ver 5 added, Timer counts up when enabled, count wraps around to zero and stops.
  nreset_timer: process (HARD_RESET,CLK)
  begin
    if( HARD_RESET = '1' ) then
      RESET_TIMER <= (others => '0');
      elsif(CLK'event and CLK = '1') then
      if ( RESET_TIMER /=  "0000" or TIMED_RESET = '1' ) then
          RESET_TIMER <= RESET_TIMER + 1;
      else
        RESET_TIMER <= RESET_TIMER;
      end if;
    end if;
   end process;

  -- Ver 5 added.
  reset: process (TIMED_RESET)
  begin
    if ( TIMED_RESET = '1' ) then
      NRESET <= '0';
    else
      NRESET <= '1';
    end if;
  end process;

end GCRC_TEM;
```

6 GCRC VHDL Revision Notes

GCRC Ver1 RevA Changes (11-14-01):
1) Association of TEM command address bits was incorrect. Last version had GCRC decoding TEM command address bits as first the GCFE address, followed by GCFE command bit, followed by GCRC address. This version correctly follows the GCRC document which is first 4 bits of GCRC address, followed by the GCFE command bit, followed by 4 bits of GCFE address.

GCRC Ver1 RevB Changes (11-14-01):
1) Readout mode was not putting out ADC LSB. GCRC_READOUT module not puts out all 12 ADC bits and adds parity.

2) Readout module had two signal in port list that were unused by other modules. The signals were removed from the port list and the GCRC_Chip file was also updated.

GCRC Ver1 RevC Changes (11-19-01):
1) Incorporated register readout into GCRC_Central to reduce flip flops.

2) Ignore TEM trigger if trigger parity is incorrect.

3) Corrected Status Error Bit to TEM being cleared in Idle state, not after transmission of bit to TEM.

4) Not look for TEM data for a Calib command.

5) Fixed problems in forwarding GCFE returned data.

6) Fixed glitches on GCFE Reset line.

GCRC Ver1 RevD Changes (11-29-01):
1) Use upper data word "A5" as a key for changing Parity in configuration register.

2) Modified chip signal names to be more consistant with GCFE signal naming.

GCRC Ver 2 Changes (1-2-02):
1) Correct problem of GCRC responding to TEM read command to a different address.

2) Add a bit to GCRC status register for invalid TEM read or write command.

3) Add another redundant bit to the GCRC configuration register for parity control.

4) Add code to enable GCRC time delay registers to be implemented with SEU registers.

The SEU registers are clocked only during load, and have no asynchronous reset.

GCRC Ver 2 RevA Changes (1-15-02):
1) Rename global active low RESET to NRESET.

2) Remove negative clock edge flip-flops from GCRC_TRIG module.

3) Correct typo in module GCRC_TRIG so signal GCRC_TRANS gets utilized.

GCRC Ver 2 RevB Changes (1-16-02):
1) Removed unused signals.

2) Changed trigger inhibit logic to re-enable triggers after ~15 clocks of no LE or HE trigger inputs.

3) Added saving TEM command to Err Register for GCFE reply timeouts.

GCRC Ver 3 Changes (2-20-02)
1) Renamed Trigger inputs from GCFEs to HE_NDISC and LE_NDISC, correctly indicating low assertion.

2) Corrected 1 bit shift in data word write to GCFEs.

3) Read Log Accept Bit after GCFEs output the bit.

4) End toggling of CMD and GRESET signals to GCFEs as quickly as possible during readout to maximize settling time caused by those signals transitioning.

GCRC Ver 4 Changes (3-25-02)
1) Removed 3.2 usec delay prior to Calib Assertion.

2) Clock in GCFE data prior to multiplexer selection.

3) Rename CMOS level signals to core, adding "_C" to signal suffix. These are Address inputs, ADC control signals, ADC data inputs, DAC control signals, DAC data input.

GCRC Ver 4 RevA Changes (3-29-02)
1) Fixed problem of GCRC_GCFE module not looking at Correct_Addr for TEM commanded GCFE read and writes.

GCRC Ver 5 Changes (11-14-02)
1) Changed Read/Write address bit to 2 bit addressing. Function codes remain the same.

2) Added TEM reset command functionality. TEM signal line reset occurs on transition from high to low. Chip cannot be put in continous reset.

3) Data readback timing from GCFE changed to allow maximum 1.5 clock cycle round-trip propagation delay.

4) Removed stretching and inhibiting of trigger requests to TEM.

5) Added hard-wired GCRC version, readable at upper 8 MSBs of GCRC configuration register.

6) Changed definition of Command error register to have leading MSB bit indicate whether error register bits start at TEM trigger command (MSB=’0’) or TEM command address bits (MSB=’1’).

GCRC Ver 5.3 Changes (12-16-02)

1) Changed GCFE Read/Write so that GCFE register data is read back at half the system clock rate. This allows two system clock cycles for each GCFE data bit to be read in by the GCRC, with no minimum operating frequency requirement of the system clock. GCFE data is still written at the full system clock rate.