University *of* Glasgow

INDUSTRIAL PROJECT EE5

Design of electronic modules for the low-level RF systems at CERN.

With particular regard to a new trigger unit for the Super Proton Synchrotron.

Thomas E. Levens

0602332

Submitted in fulfilment of the requirements for the

Degree of Master of Engineering

School of Engineering,

University *of* Glasgow

Completed at the European Organization for Nuclear Research

Supervised by Dr. Daniel Valuch and Prof. Andrew Knox

January 2011

CERN

Abstract

This report presents the work completed while the author was working for the BE-RF-FB group at the European Organization for Nuclear Research during the period of June to December 2010. The placement was completed as part of the University of Glasgow course 'Industrial Project EE5' which is requirement during the final year of the Degree of Master of Engineering. The report will pay particular attention to the hardware and firmware design of the 'Dual Trigger Unit', a new electronic module for the low-level RF system of the Super Proton Synchrotron accelerator which generates delayed timing pulses in order to trigger other hardware. In addition to this, the report will cover other projects completed during the period, including work on a prototype of the 'VME Peak Detector' card for the Large Hadron Collider beam observation system.

Table of Contents

Preface

This report presents the outcome of work that has been completed for the University of Glasgow course 'Industrial Project EE5' [1] which is a requirement during the fifth year of the degree of Master of Engineering and is completed while working in an industrial context. During the period from June to December 2010, I worked for the BE-RF-FB[1] group at the European Organization for Nuclear Research (CERN) under the supervision of Dr. Daniel Valuch.

Particular attention will be paid to the design of a new electronic module for the low-level RF (LLRF) system of the Super Proton Synchrotron, a 6.9 km particle accelerator in CERN's accelerator complex. The 'Dual Trigger Unit' was the main project completed during my time at CERN and will be covered in Part I of the report. I also had time to work on another project, the 'VME Peak Detector' which will be covered in Part II and some additional work, related to these projects, is included in Part III.

The introduction, which makes up Section 1 of the report, is written to provide a basic overview of CERN and particle acceleration in general. Readers who are familiar with the subject area may wish to skip this section.

Acknowledgements

I would like to thank a number of people for their invaluable advice and support throughout this course. First and foremost, my supervisor at CERN, Dr. Daniel Valuch and my project supervisor at the University of Glasgow, Professor Andrew Knox. I would also like to thank my advisor of studies, Dr. Nick Bailey and my second project supervisor Dr. Martin Macauley. Finally, I would like to thank the rest my colleagues at CERN and especially Wolfgang Höfle, John Molendijk, Urs Wehrle, Thomas Böhl and Grégoire Hagmann for their help and support throughout my time with the organisation.

[1] Beams Department, Radio Frequency Group, RF Feedbacks and Beam Control Section.

List of Figures

List of Tables

List of Acronyms and Abbreviations

General engineering terms

AC alternating current

ADC analogue-to-digital converter

CMOS complimentary metal-oxide-semiconductor

CRC cyclic redundancy check

DC direct current

ECL emitter-coupled logic

eV electron volt

FPGA field-programmable gate array

IC integrated circuit

I/O input/output

JTAG Joint Test Action Group

LED light-emitting diode

LVDS low-voltage differential signalling

LVTTL low-voltage TTL

MISO master input, slave output

MOSFET metal-oxide-semiconductor field-effect transistor

MOSI master output, slave input

NIM nuclear instrumentation module

NMOS n-channel MOSFET

OLED organic light-emitting diode

PCB printed circuit board

Q quality factor

RAM random-access memory

RF radio frequency

SMA subminiature version A

SMC subminiature version C

SPI serial peripheral interface bus

SRAM static RAM

SRFF set/reset flip-flop

SSH secure shell

TTL transistor–transistor logic

VHDL VHSIC hardware description language

VHSIC very-high-speed integrated circuit

VME Versa Module Eurocard bus

Terms specifically related to CERN

AD Antiproton Decelerator

BE-RF-FB Beams Department, Radio Frequency Group, RF Feedbacks and Beam Control Section.

CERN the European Organization for Nuclear Research

CLIC Compact Linear Collider

COMPASS Common Muon and Proton Apparatus for Structure and Spectroscopy

CNGS CERN Neutrinos to Gran Sasso

CTF3 CLIC Test Facility 3

CTRV Controls Timing Receiver VME

DTU Dual Trigger Unit

$\mathbf{F_{REV}}$ the revolution frequency

ISOLDE On-Line Isotope Mass Separator

LEIR the Low-Energy Ion Ring

LEP the Large Electron–Positron Collider

LHC Large Hadron Collider

LINAC linear accelerator

LLRF low-level RF

MMI man–machine interface

n_TOF neutron time-of-flight

PS Proton Synchrotron

PSB Proton Synchrotron Booster

RTC Real Time Computer

SLR Serial Link Router

SLT Serial Link Tester

SPS Super Proton Synchrotron

VHI Versatile Human Interface

VPD VME Peak Detector

1 Introduction

1.1 About CERN

Founded in 1953, by a convention between twelve European countries, the European Organization for Nuclear Research (CERN) is one of the world's largest and most respected centres for scientific research [2]. Today, it is lead by the CERN Council which is made up twenty European member states[2] and further countries and organisations have 'observer' status which entitles them to attend council meetings[3]. The physics programme at CERN currently attracts contributions from around 580 universities around the world with over 8000 visiting scientists performing research [3].

Straddling the Franco-Swiss border, just outside Geneva, CERN is host to a chain of 'particle accelerators' that are used to increase the energy of charged particles and bring them close to the speed of light. At high energies, beams of particles are smashed into each other and, in the collision, other types of particles can be formed. Detectors are constructed around the collision points to observe the debris of the collisions and examine any particles that may be produced. Since it's first accelerator was commissioned in 1957, CERN has been the host to a number of significant scientific discoveries including neutral currents (1973), the $W^{\pm}$ and Z^0 bosons (1983) and the first observations of antihydrogen (1995). It is also the birth-place of the world-wide-web (1990).

1.2 Particle acceleration

Early experiments to determine the structure of the atom were conducted using naturally occurring radiation. Ernest Rutherford first disintegrated a nitrogen nucleus to form oxygen in 1919 by using alpha particles from radioactive decay. However, only light elements can be disintegrated using this method due to the limited energy available from natural sources. A charged particle, such as an alpha particle, that is placed in an electric field experiences a force which could be used to accelerate it. In a 1927 speech to the Royal Society, Rutherford proposed 'accelerators' capable of disintegrating heavy elements [4]. In 1932 Cockcroft and Walton succeeded in disintegrating lithium using protons accelerated to 400 keV[4] by a DC voltage multiplier. However, acceleration with a DC voltage is limited to a few MeV due to the break down of air under high voltage.

Rather than a high voltage, DC, field, a lower AC voltage can be used to accelerate as this can be synchronised to the particle's movement such that the polarity of field always accelerates them. In a linear accelerator (LINAC), particles are accelerated in a straight line through the centre of a series of 'drift tubes' that alternate in polarity. The AC accelerating voltage, which is usually sinusoidal and in the radio frequency (RF) range, is synchronised so that as particles pass through a gap between two adjacent tubes they sense an accelerating voltage. The equipotential walls of the drift tube shield the particles

[2] Austria, Belgium, Bulgaria, the Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Italy, the Netherlands, Norway, Poland, Portugal, the Slovak Republic, Spain, Sweden, Switzerland and the United Kingdom.

[3] The European Commission, India, Israel, Japan, the Russian Federation, Turkey, UNESCO and the USA

[4] The electron volt (eV) is a standard unit used to measure energy. It is equivalent to the kinetic energy gained by a single electron when it is accelerated by a potential difference of 1 V and has a value of 1.602×10^{-19} J. In accelerator physics, the units of eV are also used to the quantify mass ($eV/c_0{}^2$) and momentum (eV/c_0). In both cases these can be normalised by the speed of light (by making $c_0 = 1$) to get a unit of eV.

from the negative, decelerating, portion of the RF waveform and when they have reached the next gap, the phase of the sinusoidal voltage has varied by 180° and the same accelerating field is present again. As they are straight, LINACs have the disadvantage of being a 'single pass' structure, as the particles can only pass through each accelerating gap once. Because of the limited acceleration per gap, a machine has to be physically long to achieve a large acceleration gradient.

Large accelerators are now primarily 'synchrotrons' in which electromagnets are used to bend the particles' trajectory around a ring while they are accelerated and, because of their layout, the same accelerating gap can be to increase the energy of the particles on multiple passes. Generally, the accelerating gap is built using a resonant 'cavity' that is tuned to the frequency of the RF voltage used for acceleration.

In a synchrotron, the frequency of the RF voltage is selected so that a particle travelling around machine will be in phase with the voltage each time it passes the accelerating gap. To remain in phase with the electric field, the particle must pass after an integer number of RF periods. An ideal particle that travels around the centre of the machine, and is completely synchronous with the RF voltage, is called a 'synchronous particle' whose rotational frequency is called the the revolution frequency (F_{REV}) of the machine. If the particle passes at a zero of the RF voltage it will not be accelerated, whereas a positive field in it's direction of travel will accelerate it and a negative field will decelerate it.

Rather than accelerating single particles, it is desirable to use groups of particles in order to increase the chances of collisions occurring when the beams are crossed. A group of particles will have a distribution of energies and this will cause some of the particles to arrive back at the accelerating gap later (or earlier) than the synchronous particle. As they are now out of phase, the RF voltage will give them an accelerating (or decelerating) kick resulting in them arriving earlier (or later) on the next pass of the gap. Over many machine turns, an equilibrium is reached where a group of particles oscillates around the centre point of a synchronous particle as a 'bunch' and this phenomenon is called 'synchrotron oscillation' or 'synchrotron motion'. As each bunch of particles passes the accelerating gap after an integer number of RF periods, the maximum number of stable bunches that can be accelerated is limited to one per synchronous particle, or one per RF period. By offsetting the phase of the RF voltage so that the synchronous particle senses a positive accelerating gradient, it is possible to accelerate the whole bunch of particles.

1.3 CERN's accelerators

As it is impossible to accelerate particles from rest to a high energy in a single synchrotron, CERN operate a chain of accelerators that are used to gradually increase the energy of particles. Once accelerated in a particular machine, particles are either transferred to a larger machine or supplied to any one of a number of experiments. CERN's accelerator complex consists of eight accelerators and a de-accelerator shown in Figure 1 on page xv. Currently, two types of particles are accelerated for physics: protons (H^+), which are the nuclei of hydrogen atoms; and lead ions (Pb^{82+}). The initial production and acceleration procedure is different for protons and ions, however both start with a source that strips the electrons from atoms to produce positively charged ions. In both cases, initial acceleration of the particles is performed using a LINAC: LINAC2 is used for protons, accelerating them to 50 MeV; and LINAC3 is used for ions,

accelerating them to 4.2 MeV/u. Protons are then accelerated to 1.4 GeV by a small, 25 m, synchrotron called the Proton Synchrotron Booster (PSB) before being passed into the Proton Synchrotron (PS). Ions from LINAC3 are accumulated in a storage ring, called the Low-Energy Ion Ring (LEIR), and are accelerated to 72 MeV/u before being passed to the PS [6]. The PSB also provides low energy protons for CERN's ISOLDE radioactive isotope facility.

The PS is a 628 m, 26 GeV, synchrotron and is the oldest machine in CERN's complex having been commissioned in 1959 [7]. As well as accelerating the particles delivered by the PSB and LEIR, it performs complex RF 'gymnastics' to split the bunches of particles and ensure they have the correct size and shape before injection into the Super Proton Synchrotron (SPS) and eventually the Large Hadron Collider (LHC) [8]. As well as serving as an injector for the larger accelerators, the PS provides beam for fixed target experiments in the East Area and n_TOF as well a producing antiprotons for deceleration in the AD.

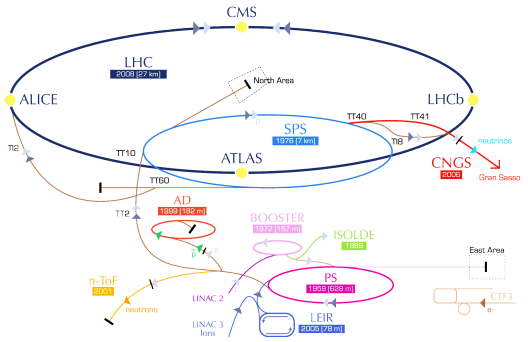


Figure 1: The current accelerator complex at CERN [5].

The SPS is a 6.9 km synchrotron straddling the Franco-Swiss border [9]. It was commissioned in 1976, as a 300 GeV proton accelerator for fixed-target physics. Since then, it has been used as a proton–antiproton collider (as the SpS), served as the final injector for the Large Electron–Positron Collider (LEP) and now provides protons and ions to the LHC alongside fixed-target experiments including CNGS, COMPASS and the North Area.

The LHC is largest particle accelerator in the world and is the primary focus of the current physics programme at CERN [10]. Located in a 27 km tunnel, an average of 100 m beneath the French countryside, the LHC is host to four major detectors along with a number of smaller experiments. Particles are injected into two counter rotating beams from the SPS at 450 GeV and are currently accelerated up to 3.5 TeV per beam before collision.

CERN is also home to the CLIC test facility (CTF3) to investigate the feasibility of a new linear, multi-TeV, electron-positron collider [11].

1.4 Low-level RF

The LLRF systems at CERN are responsible for generating the low-voltage RF signal that is amplified (by the 'high-level RF' system) before being fed to the accelerating cavities. A number of feedback and feedforward loops are used to ensure that the generated signal has the right amplitude and phase to correctly accelerate the particles within the machine. The LLRF system is responsible for ensuring beam stability by damping several types of oscillations and contains equipment used for beam monitoring and diagnostics.

The LLRF systems that are currently in use at CERN are a mixture of analogue and digital systems that have been developed over the last 50 years [12]. Every system is built in a modular nature to al-

low upgrades of individual parts of the system to be performed easily. Current development trends have moved towards systems that use field-programmable gate arrays (FPGAs) to control all digital and analogue hardware in the module. As the control logic is implemented in a programmable device, the functionality of the module can be modified without having to make hardware changes and this results in a much more flexible system that is capable of upgrades and future development.

The 'firmware' that controls the FPGA is written using VHSIC hardware description language (VHDL), allowing the system to be coded using higher-level programming concepts such as conditional statements and loops. The BE-RF group use 'Visual Elite' from Mentor Graphics [13] to provide an integrated development environment that handles the structural VHDL and allows 'blocks' of code to be reused easily. The environment offers a visual coding paradigm by using block diagrams, state machines and truth tables that are automatically converted into VHDL code.

The section operates a design repository, called *CommonVisual*, to allow commonly used VHDL blocks to be shared between designers. Based on a revision control system, the system allows a reduced design effort for new projects through reuse of pre-tested blocks that were built for written for previous modules in the system.

Part I

Dual Trigger Unit

2 Introduction

The aim of this project was to design a new 'trigger unit' for the LLRF system of the SPS accelerator. In this machine, the LLRF system is largely based around nuclear instrumentation module (NIM) modules that implement specific functional blocks and are connected to form the system as a whole. Individual modules are synchronised using timing connections that carry a digital pulse train. For example, a 1 kHz clock signal is distributed to the equipment in order to keep it locked to the master clock. Trigger units are used to generate a delayed timing pulse after an 'event' which could be, for example, the injection of particles into the machine. The delayed pulse can then be used to trigger another device, such as an oscilloscope, to observe the injection of the particles.

Figure 2: Final Dual Trigger Unit module hardware.

The requirement for a new trigger unit has arisen due to a number of deficiencies in the modules that are currently in use. The foremost problem arises from the fact that the SPS is a 'cycling' machine, meaning that it operates by sending particles to different destinations in a sequence. The pattern of the RF electric field and magnetic field intensity required to deliver the particles to each destination is called a 'cycle' of the machine and these cycles are arranged into a repeating pattern called a 'super-cycle'. Currently, the super-cycle composition can change frequently, due to the requirements imposed by the filling of the LHC and the need to deliver particles to other experiments concurrently.

The current 'Triple Timing Unit with Trigger Selection' was designed during the LEP-era when the SPS was used to accelerate leptons[5]. The module offers three delays that can be triggered from one of two 'start' inputs. Generally, one of the 'start' inputs is strobed at the beginning of each super-cycle and the time delay required to trigger in a specific cycle is calculated from this point. The reliance on each cycle being located a fixed amount of time from the beginning of the super-cyle presents a problem when the arrangement of the cycles is changed, as all trigger units need to have their time delays reset. Therefore, a major requirement for the new trigger unit is the ability to trigger at a fixed time from the start of any specific cycle, which will allow the unit to adjust automatically to any change in the super-cycle composition.

[5] Electrons (e^-) and positrons (e^+).

3 Specification

The specification document for the Dual Trigger Unit (DTU), included in Appendix A, calls for a 1L[6] wide NIM module containing two independent trigger units. Each trigger unit should receive the 'RTC Stack' serial link that broadcasts the upcoming cycle's 'MMI Target' 500 ms before the start of the cycle. If the received value matches the one programmed into the unit, it should delay for between −500 ms and 100 s, and this delay should be adjustable with millisecond granularity. Once the delay timeout is complete, a pulse should be generated at the module's outputs. The width of the output pulse should be configurable on a microsecond scale, and should, nominally, be 20 µs long. Each trigger unit should have two low-voltage TTL (LVTTL) outputs, for triggering oscilloscopes, and two optocoupler-isolated outputs for connection to other equipment in the SPS. The unit should have a timing input capable of receiving the 1 kHz clock signal, and the main delay counter should be synchronised to this clock if it is available.

Both trigger units should have a Versatile Human Interface (VHI) module on their front panel to allow configuration of the unit's parameters by the operator. The unit should indicate that a trigger has occurred by flashing a red LED and should optionally emit an audible beep.

During the DTU's design phase, and after further consultation with the operators, the following additions were made to the specification. Firstly, to offer backwards compatibility with the infrastructure provided for current timing units, the DTU should include two timing inputs, selectable alongside the MMI Target values, that can be used to start the delay counter. Secondly, the unit should have an option to synchronise it's output to the machine's F_{REV} pulse[7] that, when enabled, causes the generation of the unit's output pulse to be delayed until the next F_{REV} pulse has occured.

4 Hardware Design

The hardware design, shown in Figure 3 on the next page, is based around a central FPGA that implements the digital logic used to control the module. External circuitry is limited to interfacing the FPGA to the various electronic standards that are required for the input and output connections. Each major sub-system of the design will be discussed in the following sections. A complete schematic for the unit is included in Appendix C.1.

4.1 FPGA and supporting hardware

The core of the module is the FPGA, which is used to implement all of the control logic. An Xilinx Spartan 3AN [DS1] with 400 000 logic gates is used due to it's low cost and internal configuration memory. The FPGA is clocked from a 50 MHz crystal oscillator and can be programmed via a JTAG connection that is provided on the module's PCB.

When power is first applied to the module, the startup of the FPGA is sequenced by a pair of Maxim MAX6304 [DS2] controllers that provide time delayed pulses. The first MAX6304 keeps the FPGA in an

[6] NIM modules are built in multiples of 1.35″ wide (1L, 2L, etc) and have a standard height of 8.75″.
[7] The LLRF system generates the F_{REV} synchronisation pulse to mark the position of the first RF bucket within the machine.

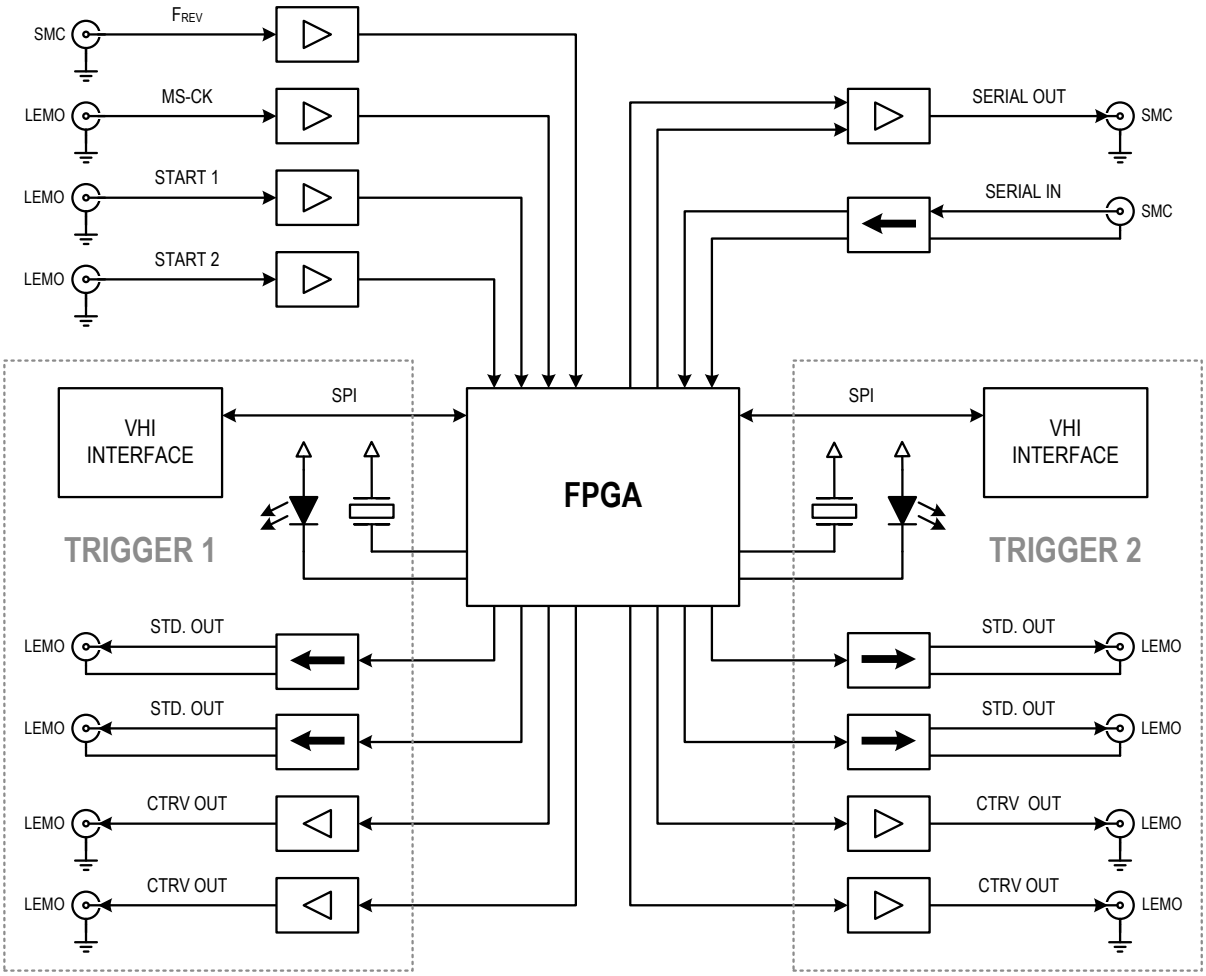


Figure 3: A simplified hardware layout of the Dual Trigger Unit. Only major functional blocks are included, auxiliary components such as the power supplies are omitted for clarity.

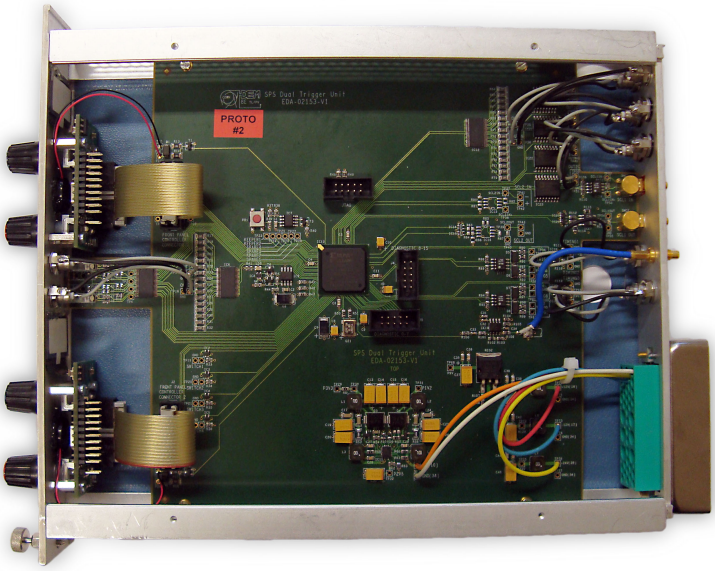


Figure 4: Interior of the Dual Trigger Unit module. The VHI modules can be seen on the left hand side of the module, connected by gold cables to the main PCB. The module's FPGA is the large IC in the centre of the board.

idle state for 250 ms after power is applied, by holding the chip's 'PROG_B' line low, to allow time for the power supply voltages to stabilise. Once this line is raised, the FPGA begins configuration from it's internal flash memory and the 'DONE' line is raised when it is complete. A second MAX6304 is then triggered, which holds the 'RST' line high for a short period of time to activate the reset pins of all logic implemented inside the FPGA.

4.2 *Power supplies*

A standard, 42 pin, connector on the NIM backplane provides the module with power supplies at ±6 V, ±12 V and ±24 V and all voltages required by the module are derived from these. The ±12 V supply lines are used as bipolar op-amp supplies and the 12 V line is regulated down to 9 V for the VHI module. The 24 V supply line is used directly for the audible signal. Finally, the 6 V line supplies a TPS75003 [DS3] switched-mode supply that generates the 3.3 V and 1.2 V supplies required by the FPGA using two buck converters.

All supply rails from the NIM backplane are filtered using a passive network, consisting of an inductor and three capacitors, to ensure that incoming supply noise is minimised and that digital switching noise does not leak out of the module.

4.3 *Versatile Human Interface*

The Versatile Human Interface (VHI) was designed for the SPS Phase Shifter [14, 15] module to improve the way it is configured. As NIM modules in the SPS have become more complex, they require an increased number of parameters to be configured for correct operation. Due to the limited space available on a 1L width NIM panel, traditional input methods, such as single digit rotary controls and switches,

severely limit the number of functions that can be provided.

The VHI module consists of a small, 128 by 64 pixel, OLED display and two rotary encoders that allow any number of parameters to be configured through an on-screen menu. Each parameter can be a numeric value of up to 9 digits, a list of predefined constants or a boolean selector. The VHI also offers read-back parameters which can be used to display values calculated by the module. The value entered for each parameter is stored in an internal, non-volatile memory, to ensure consistency across power cycles and is transmitted to the FPGA via a SPI connection.

Figure 5: This photo shows the VHI interface when displaying an editable list-type parameter.

For the DTU, the use of the VHI solves a number of problems. It allows the parameters required to be configured easily by the operator. It also offers the ability to pre-store the names of the MMI Targets as a selectable list eliminating the need for the user to remember which number corresponds to each target. As the parameters are defined in the VHI's firmware, extra parameters can be added in the future. Coupled with the FPGA's reconfigurability, this makes the DTU easily expandable and reconfigurable in the future, should the need arise.

To keep each interface simple, the DTU uses one VHI per trigger unit. To connect the VHI requires a single connector on the PCB that carries the SPI signals to the FPGA along with the 3.3 V and 9 V supplies required by the module.

4.4 Trigger outputs and timing inputs

As 'timing' connections run between different modules in the LLRF system, which may be physically located in different places, the connections are isolated on one end, using an optocoupler, to reduce the risk of ground loops being formed.

In the SPS, the connection standard uses an active-low signal generated by an open-collector transistor optocoupler that pulls the line to ground when active, and provides isolation from the module's ground. At the other end of the transmission line, a pull-up resistor to the receiving module's supply voltage keeps the line high when idle. The connection scheme is known as a 'standard-type' timing connection and is shown in Figure 7 on page 7. The SPS uses a lot of legacy hardware, which can operate at either 5 V TTL or 3.3 V LVTTL signal levels, so the connection scheme has the advantage of allowing the connections between both electrical standards. It is also capable of handling short circuits without any problems. However, the minimum pulse duration is limited by the switching speed of the transistor output optocoupler, which is in the order of a few microseconds.

For the LHC project, the timing connection standard was updated to use a standard 50 Ω line driver to generate an active-high pulse on the line. The connection is isolated using a logic-output optocoupler on the input module and the use of a high speed optocoupler is possible as it can receive the supply voltage of the module. The connection scheme is known as a 'CTRV-type' timing connection, as it is implemented by the commonly used CTRV timing card. The connection is shown in Figure 8 on page 7.

4.4.1 50 Ω LVTTL (CTRV-type) outputs

In the SPS LLRF system, the DTU's CTRV-type outputs are primarily used to drive oscilloscopes and other devices that are not equipped with the pull-up resistor that is required for a standard-type output. They also provide forward compatibility for use with hardware that has been designed for the LHC, and hence expects a CTRV-type output, which is becoming more common as systems in the SPS are updated.

To implement the 50 Ω line driver, a 74LCX16244 16 bit buffer [DS4] is used to drive two outputs. As the supply voltage of the module is 3.3 V, the driver must be able to source a minimum current of 66 mA to drive a 50 Ω load. However, the outputs must also be able to handle a short circuit in case of a fault, so a 8 Ω source resistance has been added to limit the short circuit current to 412.5 mA and maintain the high-level output voltage above 2.8 V. To meet the short circuit current requirement, eight channels of the 74LCX16244 chip are connected in parallel, with a 68 Ω resistor on each channel, to give a 8 Ω total source resistance. 250 mW rated source resistors are used, as a continuous short circuit would cause dissipation of approximately 180 mW of power in each resistor. Each channel of the driver is capable of sourcing 50 mA and, while this is slightly under the amount required to drive a direct short circuit, when combined with short pulses generated by the DTU the driver should not be overly stressed. The implemented driver circuitry is shown in Figure 9(a) on the facing page.

Figure 6: Rear panel of the Dual Trigger Unit. From the top are the module's outputs, serial link I/O, timing inputs and power connector.

Each trigger unit has two CTRV-type outputs, one on the rear panel and one on the front panel to allow easy access when connecting the unit for triggering oscilloscopes. Both outputs use LEMO 00 push-pull connectors, which are standard for all timing connections in the SPS LLRF system.

4.4.2 Isolated (standard-type) outputs

The specification for the DTU calls for output whose pulse width can be adjusted from a minimum of 1 µs upwards. Therefore, a standard phototransistor optocoupler, such as the Fairchild CNY173M [DS5], is too slow to use, with turn-on and -off times that are in the order of tens of microseconds. Faster optocouplers are available, as used for the CTRV-type timing inputs, however these require a power supply for the logic gate at the output. As the optocoupler needs to be isolated from the module's power supply and ground, the direct use of this type of optocoupler is not possible. The use of a small DC-DC converter, such as the TES 1 series from Traco Power [DS6], would enable this type of optocoupler to be used by providing isolation from the module's power supply. These DC-DC converters use a small transformer and a pulse-width modulated signal to send power across an isolated gap. For the purpose of isolation, this component would be suitable, however the added complexity and cost of having a DC-DC converter for every optocoupled output is a major disadvantage.

The Analog Devices ADuM5200 [DS7] digital isolator offers a suitable compromise, as it is capable

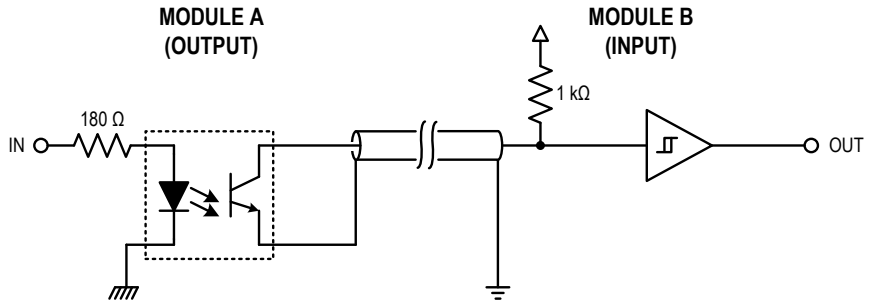


Figure 7: Schematic for the standard-type timing I/O as per the specification.

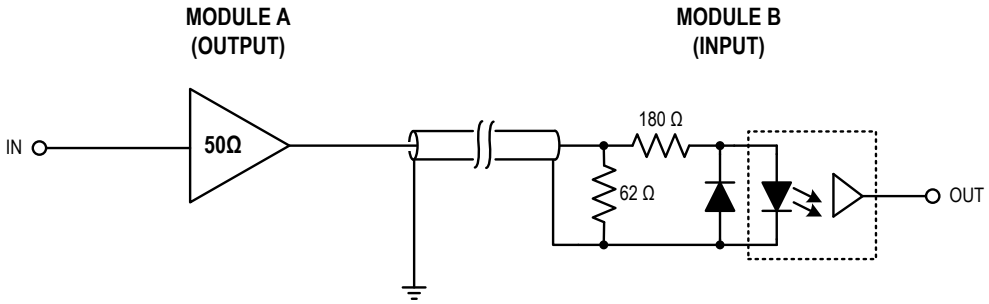
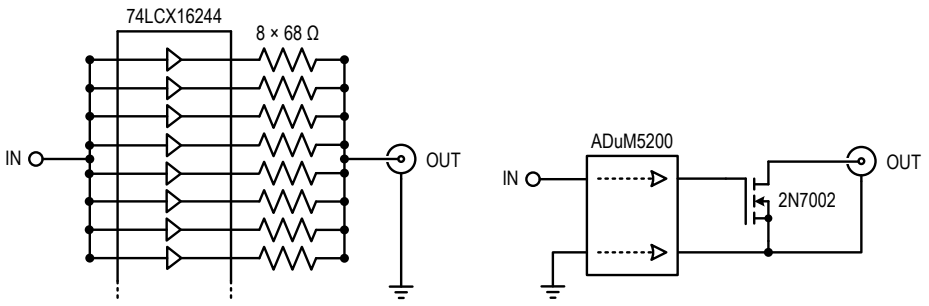


Figure 8: Schematic for the CTRV-type timing I/O as per the specification.



(a) CTRV-type (b) Standard-type

Figure 9: CTRV- and standard-type outputs as implemented in the Dual Trigger Unit.

of high speeds and offers an integrated DC-DC converter to power it's output section. The component is available in speed grades starting at 1 MHz, and these are suitable for the outputs of the DTU. To provide an open-collector-style output, as is expected, a standard 2N7002 [DS8] NMOS transistor is connected to the isolated output of the ADuM5200. The implemented circuit is shown in Figure 9(b) on the previous page.

Each trigger unit has two standard-type outputs, both on the rear panel. As with the CTRV-type outputs, both use LEMO 00 connectors with the addition of plastic washers to isolate the connector from the module's metal chassis.

4.4.3 Timing inputs

The DTU has both standard- and CTRV-type timing inputs provided on the PCB. The standard-type timing inputs consist of a 1 kΩ pull-up resistor to module's 3.3 V supply and a SN74LVC1G17 [DS9] logic buffer to provide a signal with fast edges to the FPGA. The CTRV-type input uses a HCPL-063L [DS10] optocoupler with a 180 Ω current limiting resistor to protect the internal LED. A 62 Ω parallel resistance is used to give a total load resistance of around 50 Ω and a diode is connected in anti-parallel with the optocoupler's LED to protect against reverse-polarity connection.

Three of the standard-type timing inputs are wired to the module's rear panel, all using LEMO 00 connectors. A fourth standard-type input is available on the PCB, along with four CTRV-type timing inputs. The back panel LEMO connectors have isolation washers fitted to allow them to be converted from standard-type to CTRV-type if this is needed.

4.5 Serial link

The serial link protocol, named 'SerialLink[16]', was developed for the SPS in the early 1990s as a way of sending data up to 100 m over a single coaxial cable. The packet size is traditionally 16 bit, however some, extended, 32 bit connections are now also in use. The serial output is driven by a LT1223 [DS11] op-amp which amplifies two signals from the FPGA: a start pulse, which is inverted to form a -2 V pulse; and a series of data bits which form a series of 2 V (for logic high) and 0 V (for logic low) pulses. As with the CTRV-type timing connection, the serial link is optoisolated at it's input side and connects the LEDs of two HCPL-063L optocouplers in anti-parallel. In this configuration, one is activated by the -2 V start pulse, and the other is activated by the 2 V data bits, decoding the single serial line back into the two control signals that can be processed by the FPGA.

4.6 F_{REV} input

The F_{REV} signal is generated by the LLRF system to indicate the position of the first RF bucket in the machine. Unlike the 'slow' timing signals, it is a fast, 2 V, signal that has a pulse width of around 5 ns making it impossible to capture directly using the synchronous logic inside the FPGA. The method employed by other LLRF modules is to use the F_{REV} pulse to set an external ECL set/reset flip-flop (SRFF) that is fast enough to be triggered by the F_{REV} pulse. The output of the SRFF is detected by the FPGA

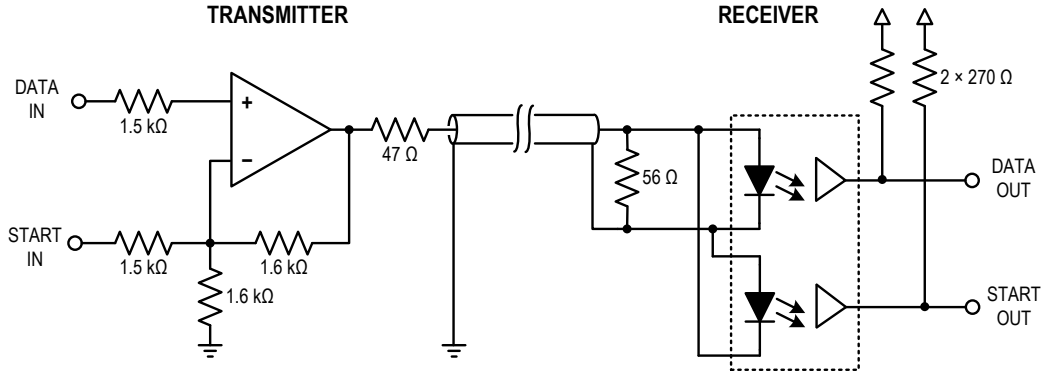


Figure 10: SerialLink[16] transmitter and receiver as defined by the specification.

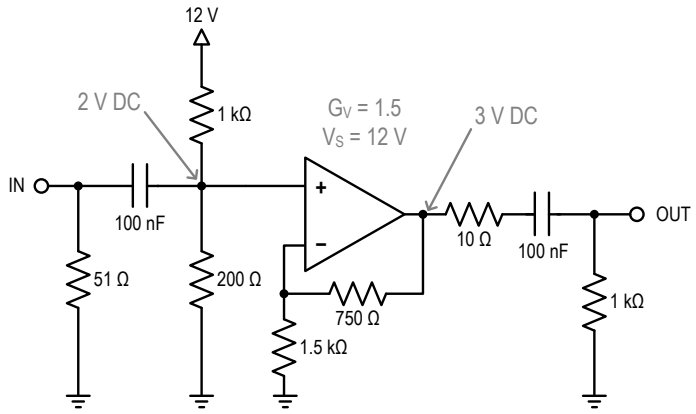


Figure 11: Amplifier setup to scale the F_{REV} input from a 2 V pulse to a 3 V pulse.

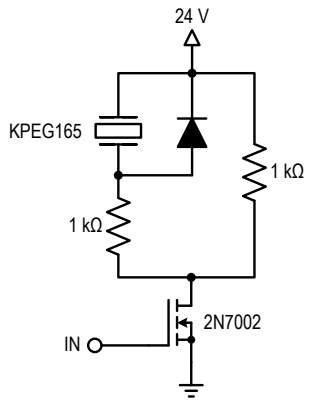


Figure 12: The audible signal piezo driver circuitry. The protection diode is a 15MQ040N.

on it's next clock cycle, and a hand-shake signal is sent to clear the SRFF. While the system works well, it is complex to implement, as ECL logic uses voltage levels that are incompatible with the CMOS logic of the FPGA, requiring additional 'glue' logic in between. The F_{REV} input level also has to be carefully scaled to meet the input requirements of the SRFF.

For the DTU, a simpler solution is presented. Instead of using an external ECL SRFF, a flip-flop inside the FPGA is used to capture the signal. The Spartan 3AN data sheet [DS1] states a minimum clock high time of 0.75 ns for the internal synchronous logic, which is enough to capture the fast F_{REV} signal. The operation of the logic inside the FPGA is fully described in Section 6.3.

Therefore, the only hardware requirement is to amplify the 2 V F_{REV} pulse to a level compatible with the 3.3 V CMOS logic inside the FPGA. A high-bandwidth THS3202D [DS12] op-amp is used, however it does not support rail-to-rail output levels, so operating it from a single-ended power supply is not possible. It also does not support $\pm$12 V supplies, which are the only differential supplies available on the DTU's PCB. To allow operation from a single 12 V supply, the op-amp is AC-coupled on it's input and output. A potential divider sets the input bias to 2 V DC and the gain of the op-amp is set to 1.5, amplifying the DC-bias to 3 V at the output. The 2 V F_{REV} pulse is also amplified to 3 V and is AC-coupled out with a reference to ground in order to give a pulse that can be fed directly to the FPGA. The circuitry can also be adapted to a LHC style F_{REV} pulse, which is 1 V, if this required by changing the resistors in the feedback network.

4.7 User feedback

4.7.1 Audible signal

The audible signal is implemented using a piezo sounder driven by a square wave to provide a beep that can be heard when the DTU triggers. The main challenge has been finding a sounder meets three criteria: it has to be loud enough to be heard over a high level of background noise; it has to have a reasonable frequency range to allow selection of the beep frequency; and it has to be small enough for two to fit in the constrained amount of space available on the module's front panel. The Kingstate KPEG165 [DS13] is the only sounder readily available that meets these criteria, the limiting factor being the space available on the front panel. The KPEG165 is small enough to fit in the space between the encoders of the VHI module and is mounted on flying leads for connection to the PCB.

The sounder is driven from a 24 V supply using a 2N7002 transistor as a low-side switch. A 1 kΩ pull-up resistor and a 1 kΩ current-limiting series resistor are used, as shown in Figure 12 on the preceding page. The sounder's supply voltage is maximised to increase the sound level of the beep. The available supply voltages were tested using a sound level meter placed 1 m from the sounder. With the transistor driven by a 4.8 kHz square wave, the sound level produced was: 78 dB at 24 V; 70 dB at 12 V; and 64 dB at 6 V. Therefore, maximising the voltage supplied to the circuit leads to a noticeable increase in output volume of the sounder to ensure that it can be heard over a high level of background noise.

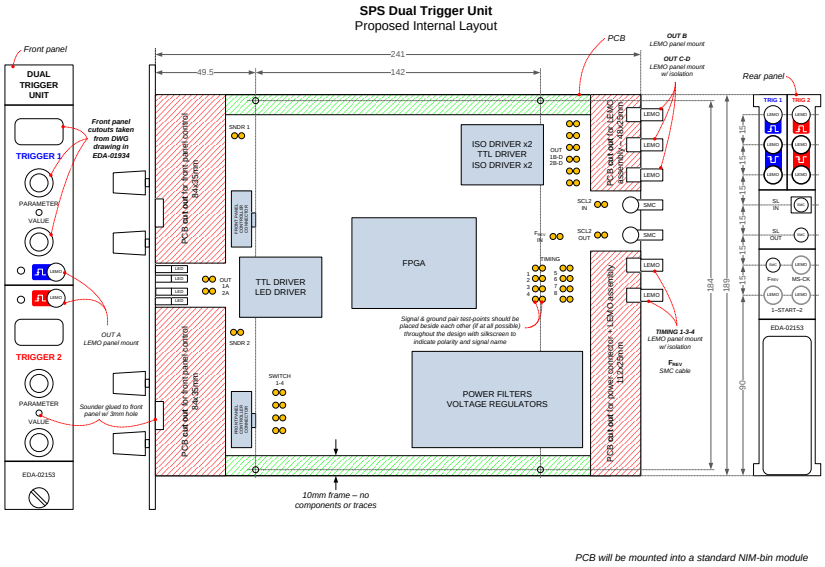


Figure 13: Diagram submitted to the design service of the proposed layout of the Dual Trigger Unit.

4.7.2 LEDs

Each trigger has a tri-colour, red-green-blue, LED to provide a visual indication of the unit's status. The LEDs are driven by a 74LCX16244 using a single line per colour. Although only six lines are utilised for the two LEDs, all sixteen lines of the driver are wired to the FPGA, in case they are needed for future upgrades.

5 PCB and Mechanical Design

The PCB layout and mechanical design for the module were completed by CERN's 'Development of Electronic Modules' design service [16]. Schematics for the module were prepared and submitted to the service, along with a diagram to describe the component placement and external layout required, as shown in Figure 13. The PCB layout and mechanical part drawings were produced by Pascal Vulliez and were received as 'Gerber' files that could be sent directly to the manufacturer of the module. The designs, along with the schematic, are included in Appendix C.

Significant time has been spent devising a usable layout for the module. The main limitation imposed upon the design is the space available on the front and rear panels of the module. The physical layout of VHI module cannot be adjusted, so these consume a significant portion of the available front panel space. The two output connectors are placed together, in the middle of the front panel, so that a cable can be swapped from one to the other with minimal effort. A red/blue colour scheme is used for the two triggers to enable easy identification of which rear panel outputs are associated with each trigger. The timing inputs on the rear panel have isolation washers fitted and, while these are not required for the standard-type timing inputs, their inclusion allows the flexibility to convert to a CTRV-type input in the future.

6 FPGA Firmware Design

The top level of the DTU firmware design, included in Appendix D.1, is based around two identical 'trigger unit' blocks as shown in Figure 14 on the facing page. Each block implements all of the logic required for a single trigger unit and their abstraction as a block ensures that updates are applied to both triggers concurrently.

Each trigger unit consists of a delay timer that triggers a pulse generator. The architecture of this block is illustrated in Figure 15 on the next page and the firmware block diagram is included in Appendix D.7. The delay timer can be started by one of two sources: a serial packet containing the correct cycle number; or an external timing pulse at one of the two 'start' inputs. The trigger source is received as a value from the VHI and is decoded to select the correct source. The first 32 values (0–31) select a serial packet whilst values 32 and 33 select the start inputs. Once the delay time has expired, a pulse generator is triggered to generate the output pulse.

A second operating mode exists, where the output pulse is synchronised to the LLRF system's F_{REV} pulse. In this mode, once the delay timer has expired, the unit waits until a F_{REV} pulse has been received and then generates the output pulse as normal. The operation of this mode will be described in Section 6.5.

6.1 Serial input

The serial start and data inputs are decoded by the SerialLink[16] receiver block that is described in Section 19. The MMI Target is transmitted over a 16 bit serial link as a 5 bit number plus an odd parity bit. The value received in the serial packet is compared to the value programmed into the 'Trigger Source' parameter of the VHI and, if they match, a strobe is sent to start the delay counter. If the parity bit is set incorrectly, or any of the upper bits of the serial packet are set, then a special value of 20_{16} is sent to the VHI as the current target read-back to display a message on the VHI display and let the operator know that there is an error.

6.2 Timing inputs and clock generation

The module's back panel timing inputs are synchronised into the FPGA's clock domain using the ClkSync block, included in Appendix N.3. The block implements a two flip-flop synchroniser and uses a third flip-flop to provide positive and negative edge detection strobes of a single FPGA clock width.

If the external 1 kHz clock signal is not available, an internally generated 1 kHz signal can be substituted. The clock is synthesised by dividing the the 50 MHz quartz oscillator input by 50 000 using a ClkGate block, included in Appendix N.2. The generated clock is resynchronised by the negative edge of the serial start pulse, or the timing input edge used to trigger the delay, to maximises the accuracy the delay duration. If resynchronisation is not performed, up to 1 ms of additional jitter can appear on the pulse output as there would be no way to know where the start pulse occurs in relation to the generated clock period.

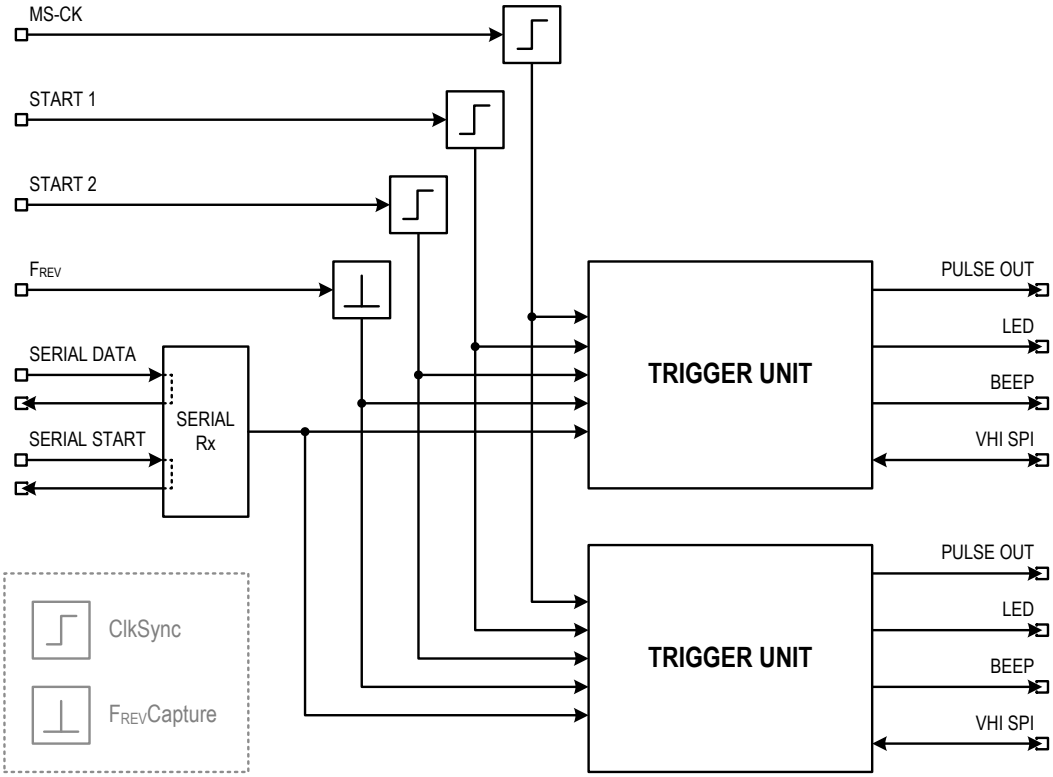


Figure 14: Top level firmware block diagram for the DTU.

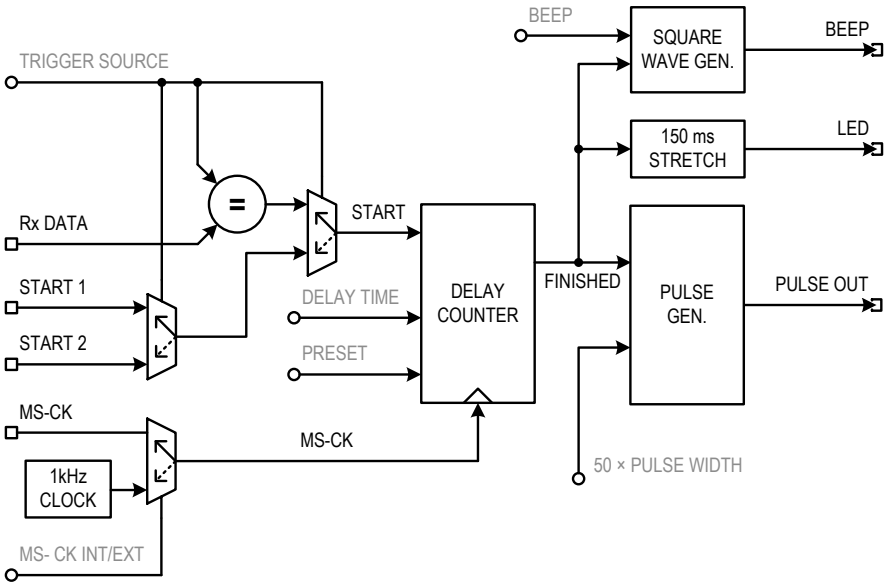


Figure 15: Simplified block diagram of each 'trigger unit' within the DTU. This diagram excludes the F_{REV} syncing ability. Grey text represents a register updated from the VHI interface. Red text represents I/O.

6.3 F_{REV} input

As mentioned in Section 4.6 the DTU uses an internal flip-flop to catch the fast F_{REV} pulse. The logic used is illustrated in Figure 16(a) on the facing page and the firmware block diagram is included in Section D.5. The F_{REV} input pulse is connected to the clock input of a D-type flip-flop to clock in a static high value. The latched F_{REV} pulse is synchronised into the FPGA's clock domain using a ClkSync block and the synchronised output is fed back, after a single clock delay, to clear the input flip-flop by activating it's asynchronous reset pin.

The output of the ClkSync block is used as a clock-synchronous F_{REV} signal and the output of the initial flip-flop is used as an asynchronous signal. The timing of these signals, with reference to the input pulse, is shown in Figure 16(b) on the next page. The reset of the input flip-flop is delayed by one clock cycle to ensure that the asynchronous F_{REV} pulse is kept high for the duration of the synchronous pulse.

6.4 Delay counter

The DlyCounter block, included in Appendix D.4, implements a signed up-counter with preset and maximum values. When the start strobe is received, the counter is loaded with the value at it's 'Preset' input. When using a serial trigger, the preset value is taken from a VHI register and when using an external trigger, the counter is preset with zero. The counter increments on each 1 kHz clock edge until the maximum value has been reached and then a strobe is generated to trigger the pulse generator.

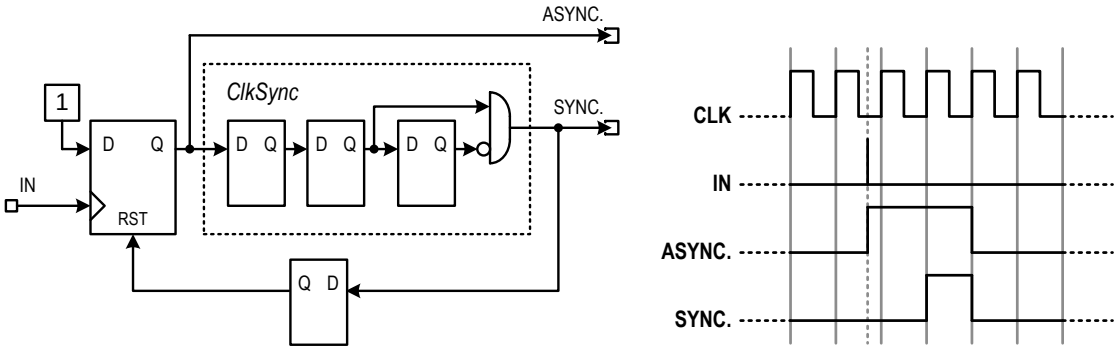
In the LLRF system, the start of the serial packet is well aligned to the 1 kHz clock and it's reception takes around 11 µs. The 1 kHz clock period that the serial packet ends in is considered as the initial millisecond of the delay, as shown in Figure 17 on the facing page. A special case exists for where the maximum value set to be less than, or equal to, the preset value. Instead of waiting for the next clock edge, in this case the finished strobe is generated as soon as the start strobe is received.

6.5 Pulse generator

The PulseGen block, included in Appendix N.6, generates a pulse of a fixed duration by holding it's output high for a certain number of clock periods. In the DTU, the output pulse can be triggered in one of two ways, depending on whether the unit's output is synchronised to F_{REV} or not. In the non-synchronised mode, the output pulse is triggered directly by the finished strobe of the delay counter. Alternatively, once the delay has elapsed, the output pulse is delayed until the next F_{REV} pulse is received. The delay finished strobe is used to set a SRFF and it's output enables an AND gate. This lets the next synchronous F_{REV} pulse through to trigger the output pulse and is fed back to clear the SRFF. The asynchronous F_{REV} pulse is also used to set the output pulse immediately via an OR gate at the output to minimise jitter, which be described in Section 8.3.

6.6 User feedback

A second PulseGen block is used to generate a 150 ms long pulse at the same time as the output pulse is generated and this is used to flash a front panel LED red. The same LED is also illuminated green while



(a) Firmware for F_{REV} capture (b) Timing of capture

Figure 16: The fast F_{REV} signal is capture in firmware using a quasi-SRFF and synchroniser. The timing of the inputs and outputs is also shown.

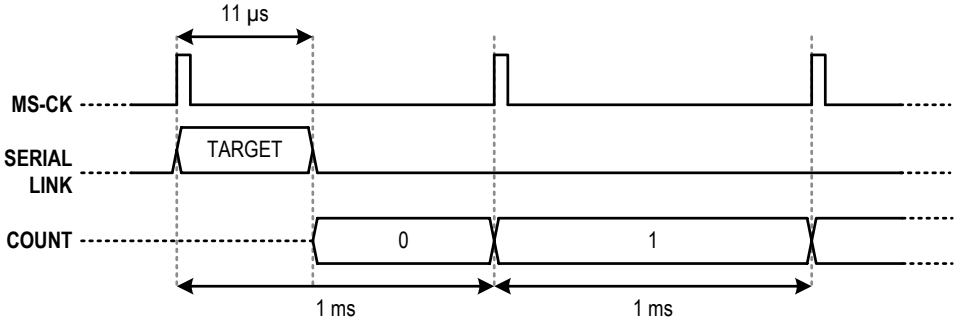


Figure 17: Timing diagram of the serial packet, 1 kHz clock and resultant delay count (not to scale).

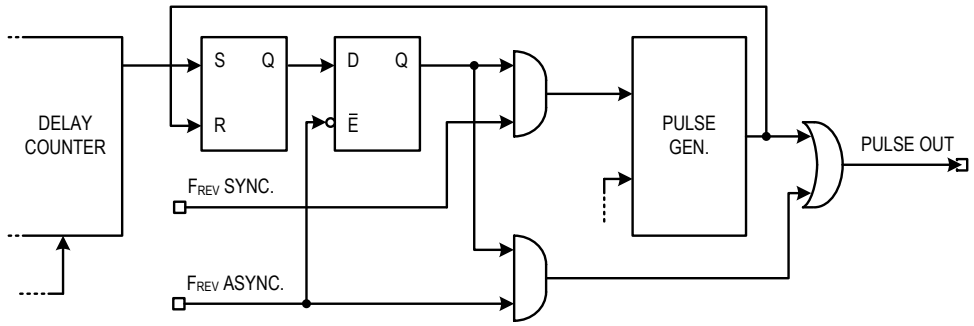


Figure 18: The connection between the delay counter and pulse generator is modified when F_{REV} synchronisation is enabled. When the delay counter is finished, the output pulse is triggered on the next F_{REV} pulse. The asynchronous F_{REV} pulse is used to set the pulse output immediately in order to reduce jitter.

the delay counter is running to provide an indication that the unit has triggered when using long delay times.

To check for the presence of the 1 kHz clock and F_{REV} pulse, these inputs are connected to a ClkChk block which counts the number of 50 MHz clock periods since it received the last clock strobe. If another strobe does not occur within a defined period, set to ± 5 % of the nominal clock period, then the signal is considered to be missing. These two checks are used to illuminate the front panel LED blue and indicate to the user that there is a problem. However, during normal operation of the SPS, the regular F_{REV} pulse stops between each cycle, and this would result in a short, regular, flash while the F_{REV} synchronisation mode is enabled, even though there is nothing wrong. Rather than setting the front panel LED to be continuously illuminated by the lack of an input signal, a short timeout of 5 s is generated by the operator enabling the use of the external 1 kHz clock or the F_{REV} synchronisation. If the relevant input signal is missing during this timeframe then the LED is illuminated to alert them. Otherwise it is only illuminated when a trigger occurs and the relevant signal is not available.

The audible piezo beeper is driven by a square wave that is generated by a ClkGate block. The 150 ms pulse used to flash the LED is also used to enable the square wave generation. Four preset output frequencies are programmed into the module's firmware and can be selected using a parameter in the VHI.

7 VHI Firmware

The firmware for the VHI module has been developed by Markus Labs [17] and is written in C for the Atmel microcontroller present on the VHI module. It is designed to allow all parameter configuration through a single header file that is compiled and flashed to the module via a JTAG connection.

For the DTU, parameters for all user configurable values, such as the trigger source, delay time, output pulse width, F_{REV} and 1 kHz clock synchronisation and audible beep are provided. A hidden parameter is also provided to allow the main counter preset value to be set. Having this as a VHI parameter, rather than hardcoded into the FPGA, saves having to modify the FPGA's firmware if the value is changed from the nominal value of 500 ms. Finally, a read-back of the active MMI Target and the unit's delay time are provided. The header file that defines all of the parameters is included in Appendix E. The list of MMI Target names is taken from a separate header file using a preprocessor macro, allowing the same values to be included for both the trigger source and active MMI Target read-back parameters. Doing this will save the user having to update the two lists separately if a MMI Target name changes.

8 Problems Encountered

8.1 Slow rising edge of 1 kHz clock

A problem was encountered during initial testing of the module where, occasionally, the output pulse would be generated 1 ms too early. The problem is shown in Figure 20 on the next page, which was recorded over a number of output cycles, and where there is an early output pulse highlighted in purple.

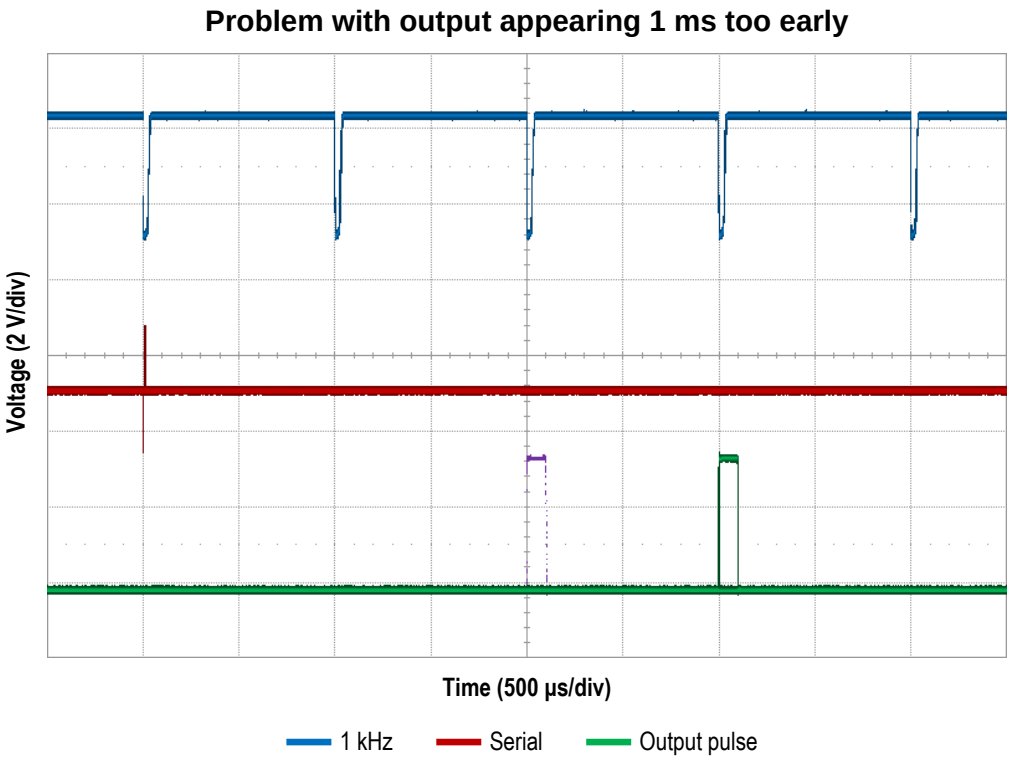


Figure 19: Problem with output pulse appearing 1 ms too early. The blue trace shows the raw 1 kHz clock. The red trace is the serial packet indicating the start of the delay. The green trace is the output pulse. Notice the erroneous output pulse which is highlighted in purple.

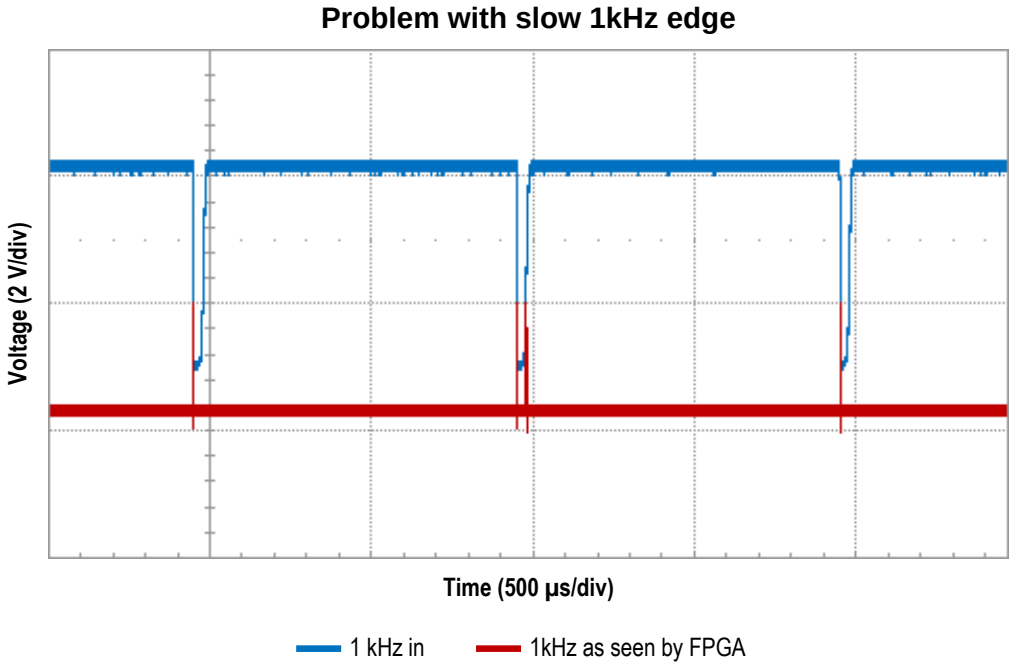


Figure 20: Test to confirm source of problem shown in Figure 19. The blue trace is the raw 1 kHz clock. The red trace is output from the FPGA's ClkSync block. On the middle clock, a second pulse is generated due to a double-crossing of the low-high transition level.

Further testing was performed which determined that this glitch only occured when using the external 1 kHz clock signal. It was noticed that, on one occurrence, the output pulse was aligned with the rising edge of the 1 kHz clock signal rather than the falling edge and this led to the theory that the slow rising edge of the clock signal was causing a second strobe to be generated. To test the hypothesis, the output of the ClkSync block, indicating the negative edge of the external clock, was routed to a diagnostic port on the FPGA. The signal was viewed on an oscilloscope alongside the raw 1 kHz clock signal. The results are shown in Figure 20 on the preceding page and confirm that the rising edge of the clock can cause the input logic's low-high threshold point to be crossed a second time which causes an extra clock to be registered by the FPGA.

The timing input circuitry initially used a Texas Instruments SN74LVC1G125 [DS14] buffer with a 3.3 kΩ pull-up resistor. To remedy the problem, the buffer has been changed for a SN74LVC1G17 [DS9] which has a Schmitt-trigger input. A 1 kΩ resistor has also been substituted to decrease the rise time of the optocoupler.

8.2 Output pulse jitter

Since the output pulse only needs a 1 µs resolution, the output pulse generator was initially driven by a 1 MHz clock that was derived from the 50 MHz oscillator. However, it was found that doing so introduced extra jitter between the output pulse and the external 1 kHz clock. As there is no fixed phase relationship between the external clock and the internally derived clock, the pulse generator could be triggered at any point during the 1 MHz clock period. The output pulse can not be set until the next rising edge of the 1 MHz clock and hence there is up to 1 µs of jitter on the output pulse. To correct this, the output pulse generator is now clocked directly by the 50 MHz clock and a 50x multiplication factor is applied to the pulse width parameter by the VHI interface. The result is the required 1 µs granularity on the parameter input while minimising jitter on the pulse output.

8.3 F_{REV} jitter

When generation of the output pulse is performed in a way that is completely synchronous to the FPGA's clock, up to 20 ns of jitter can occur due to the lack of a phase relationship between the FPGA's clock and the triggering F_{REV} pulse. Because of this, the F_{REV} pulse can occur at any point during the FPGA's clock period and the output will only set on the rising edge of the clock. To reduce the jitter, the output pulse is set, asynchronously, as soon as the F_{REV} pulse is captured by the logic inside the FPGA. While this reduces the jitter on the rising edge of the output pulse, it has the downside of adding the same amount of jitter to the falling edge. In an operational situation, the falling edge jitter will not pose a problem, as only the rising edge of the output pulse is used to trigger other devices. A comparison of the jitter observed when using the synchronous and asynchronous pulse outputs is shown in Figure 21 on the next page.

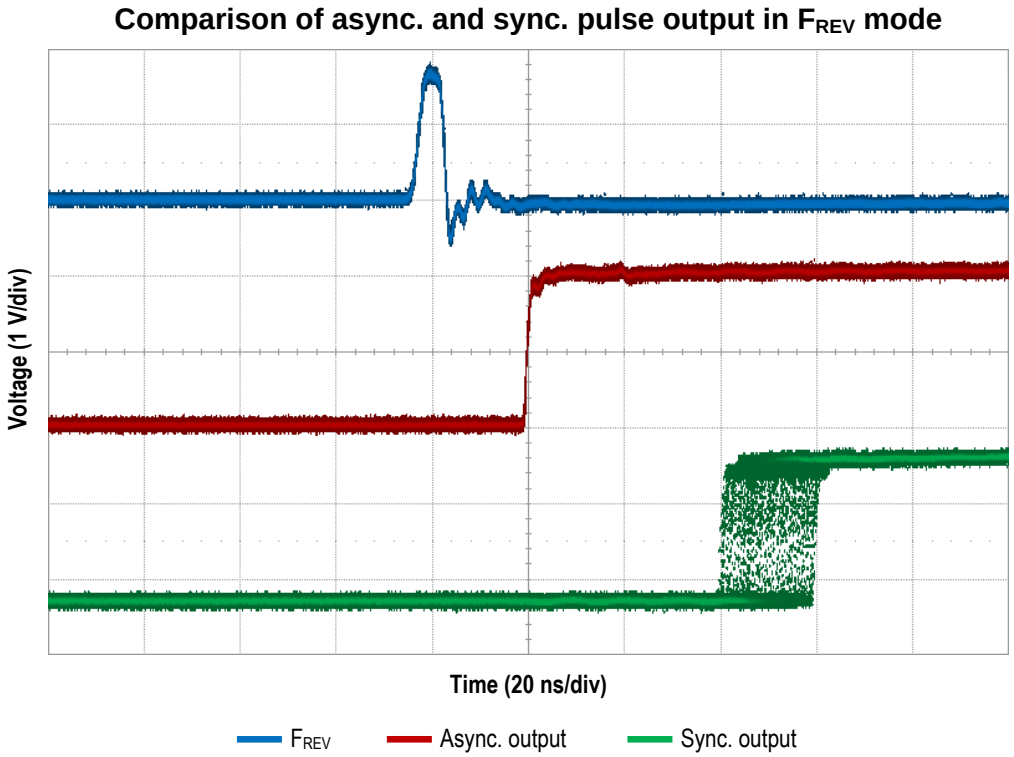


Figure 21: Comparison of async. and sync. pulse output in F_{REV} mode. For this plot both output pulses are attenuated by 3 dB and the F_{REV} pulse is attenuated by 2 dB. The image was recorded from 300 superimposed cycles.

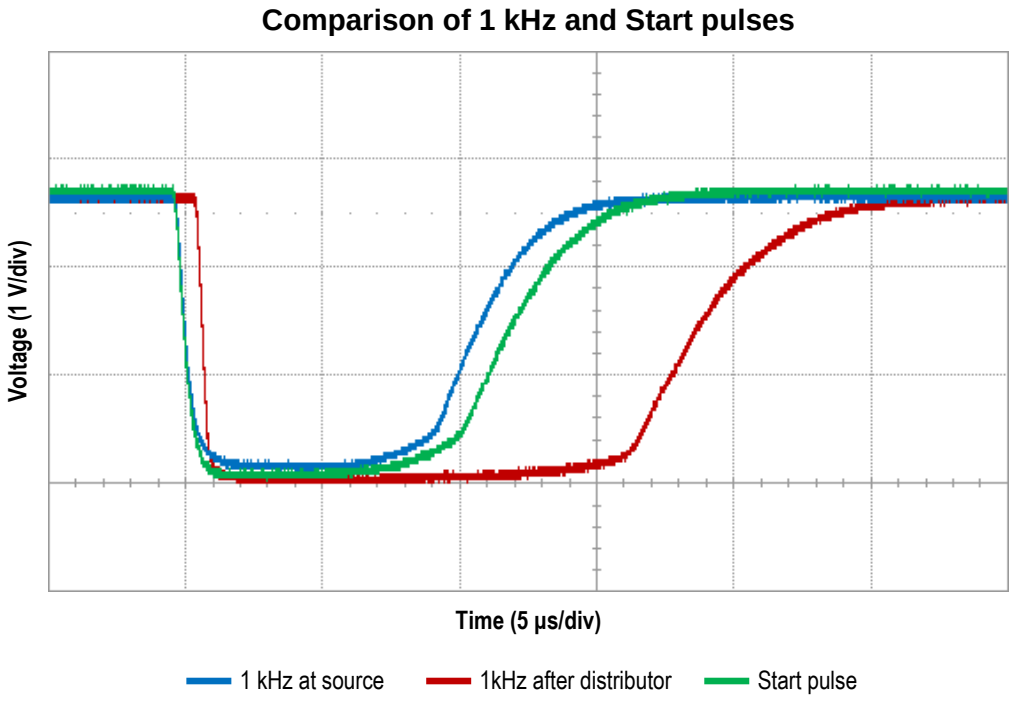


Figure 22: Plot showing the relationship between 1 kHz clock and external start pulses. The blue trace is the external start pulse. The green trace shows the 1 kHz clock at it's source. The red trace shows the 1 kHz clock after a clock distributor. This distributor adds approximately 2 µs of delay to falling (active) edge of the clock signal.

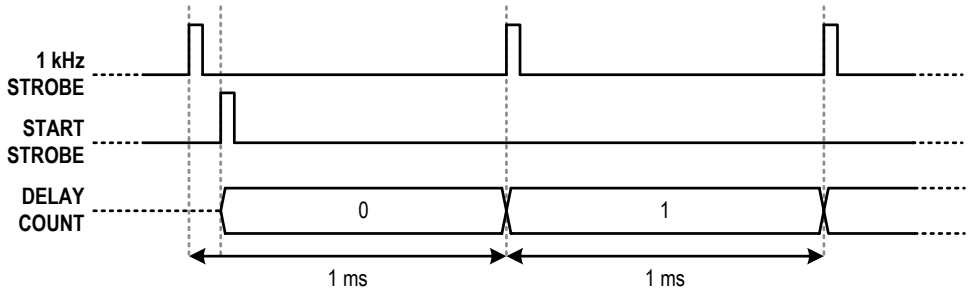


Figure 23: Normal operation of counter with external start pulses (not to scale). In this case, the start pulse occurs some time after the 1 kHz clock and the count is incremented correctly.

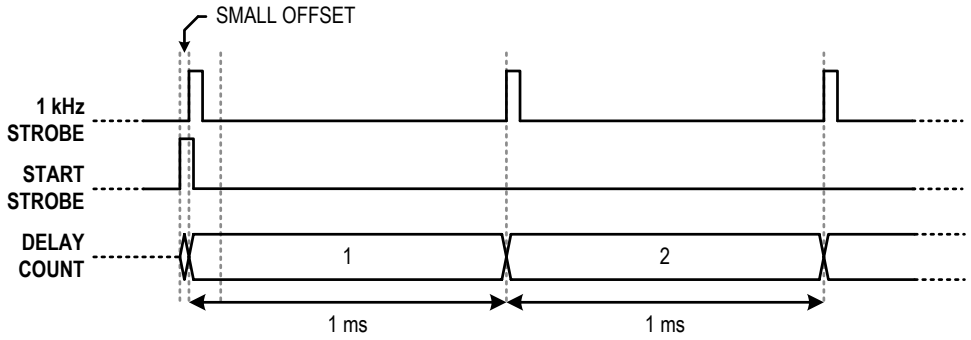


Figure 24: Problem with external start pulses occurring just before the clock (not to scale). In this case, the start pulse occurs just before the 1 kHz clock and the count is incremented by one incorrectly.

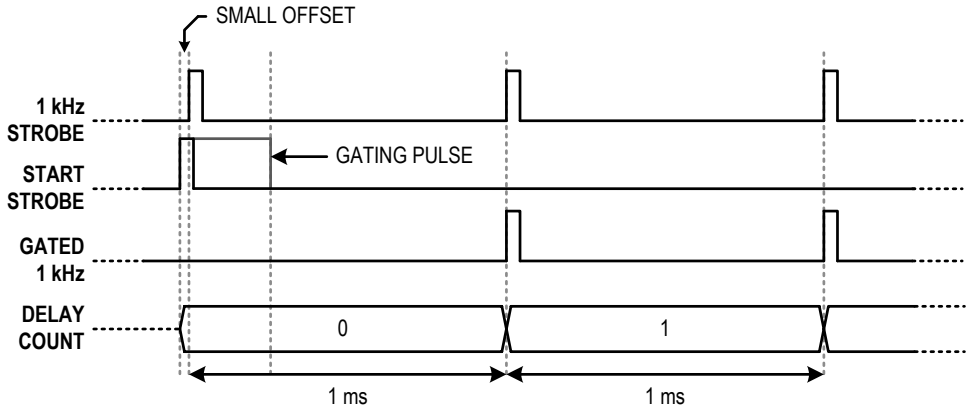


Figure 25: Fix for external trigger problem (not to scale). The high period of the start signal, marked in blue and nominally 20 µs in length is used to gate the 1 kHz clock. This removes the incorrect increment and the count is correct again.

8.4 *External triggers*

A final problem arose when triggering the unit using the external start inputs and using the external clock which caused the output pulse to be generated 1 ms too early. However, the same problem did not occur when using a serial trigger or the internally generated 1 kHz clock. To trace the source of the problem, the external 1 kHz clock and start signals were observed using an oscilloscope. A measurement was also made at the source of the the 1 kHz clock and the results are shown in Figure 22 on page 19. From this measurement it is evident that falling edge of the start pulse, as seen by the module, occurs around 1 µs before the falling edge of the 1 kHz clock. However, at their sources, the start pulse and clock are well synchronised. The additional 1 µs delay on the clock signal is added by a signal distributor box that passes the signal through an additional optocoupler.

The slight delay in the 1 kHz clock is a problem because of the assumption that the start pulse will always appear within the first 1 kHz clock period as illustrated in Figure 23 on the preceding page. However, due to the slight delay in the clock, the situation illustrated in Figure 24 occurs. In this case, there is a brief count of zero before the 1 kHz clock edge which causes the count at the next clock to be too high.

As no more than a couple of microseconds of delay can be added by distributor boxes, a solution has been devised to allow start pulses that occurred within a few microseconds of the 1 kHz clock to be handled correctly. The low-portion of the external start signal, which is nominally 20 µs long, is used to gate the 20 ns strobe generated on the negative edge of the 1 kHz clock signal as shown in Figure 25. With this additional gating and nominal input pulses, a start pulse that occurs within 20 µs of the 1 kHz clock edge, will be handled correctly as the first 1 kHz strobe will be ignored.

9 Testing and Verification

9.1 *Testing during development*

The DTU has been continually tested during development to ensure that the design operated within the specification provided. A modified SPS Phase Shifter module, shown in Figure 26 was utilised as a development platform for the hardware and firmware. It was chosen as a significant portion of hardware functionality required by the DTU is present. The module uses the same FPGA ass the DTU and features a VHI module, serial links and standard-type timing inputs. A F_{REV} input and piezo sounder driver were constructed on prototype boards and added to the module. The Phase Shifter's second serial output was converted to use as a LVTTL driver by connecting the, inverted, start line to ground and using the data line as an output. Although this method did not provide a LVTTL-compatible output voltage, it was sufficient for testing the operation of the module. The prototype also

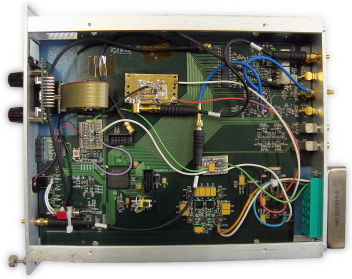


Figure 26: Prototype Dual Trigger Unit built from a modified SPS Phase Shifter module.

allowed firmware development and testing to continue while the DTU's PCB was designed and prototypes were manufactured.

Before testing on hardware, individual sections of the DTU's firmware design were simulated using ModelSim, which is available within the Visual Elite development environment. Thorough simulation allowed each individual block of the firmware to be tested and it's behaviour verified correct, enabling testing on hardware to concentrate on issues related to interfacing the module with the LLRF system.

Testing the prototype in the SPS's LLRF system with 'real' signals proved invaluable in discovering the issues discussed in Section 8. Having a prototype in place also gave the operators a chance to test the unit and give feedback on it's operation and the configuration interface. The prototyping process meant that the majority of issues had been resolved by the time the initial prototype of the DTU's PCBs arrived. Since this point, very few changes have been required to the firmware.

9.2 Final testing

Final testing, to verify the performance of the unit met specs, was performed by connecting the prototype to the timing signals in the LLRF system. The outputs of the module and the relevant timing signals were connected to an LeCroy WaveRunner 104MXi oscilloscope that has four input channels with 1 GHz bandwidth and offers 'infinite persistance' to allow traces to be recorded over a long period of time. Further, the oscilloscope can directly provide statistical information about the traces that have been recorded.

9.2.1 Ouput pulse jitter from 1 kHz clock

The plot, shown in Figure 27 on the facing page, is the result of 1915 output cycles recorded using the oscilloscope's infinite persistance. The blue and red traces show the DTU's output when using the external and internally generated 1 kHz clock signals. The purple trace is the standard-type, active-low, 1 kHz clock signal as received by the DTU from a signal distribution box which has a reasonably slow falling edge due to the optocoupled output. To provide a fast edge for the oscilloscope to trigger on, the same 1 kHz clock signal, taken from the signal distributor's test-point, is plotted as the green trace and all time delays are measured from this signal. The measurements were taken with the DTU set to produce a nominal pulse width of 20 µs and to trigger from a serial packet at a time of 0 ms, in other words 500 ms after the serial packet is received.

The results are presented in Table 1 on the next page. From these, it is clear that the output pulse width produced by the DTU is very accurate with jitter of around 50 ps. However, this is of little importance due to the fact that only the initial edge of the pulse is used to trigger other devices. Due to this accuracy, and the configurable pulse width, the possibility of using the output as a gating pulse for other signals is available.

The results for the jitter between the output pulse and the 1 kHz clock are more complicated to interpret, due to the fact that the 1 kHz clock passes through a signal distributor box before reaching the module, adding 1.30 µs of delay and 16.35 ns of jitter to the signal, as shown by Measurement C in Ta-

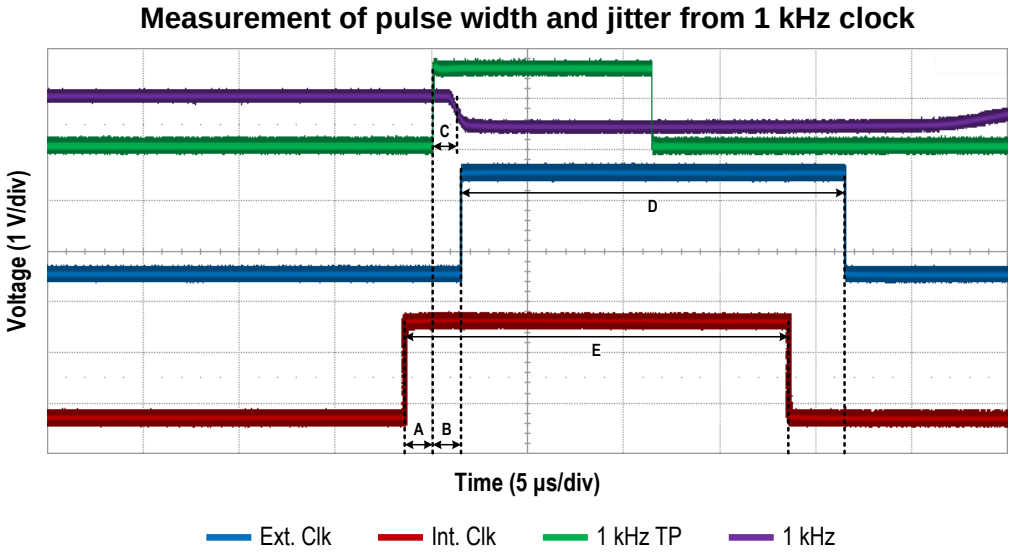


Figure 27: Measurement of the DTU's output pulse width and jitter compared to the 1 kHz clock. The blue and red traces are the output pulse when using the external and internal 1 kHz clocks respectively. The purple trace is the standard-type optocoupled 1 kHz clock. as received by the DTU. The green trace is the same clock before the distribution box which provides sharper edges to measure from.

	Minimum	Maximum	Mean	Std. Dev.
A	1.34 µs	1.61 µs	1.41 µs	61.89 ns
B	1.45 µs	1.47 µs	1.46 µs	5.99 ns
C	1.25 µs	1.38 µs	1.30 µs	16.35 ns
D	20.00 µs	20.00 µs	20.00 µs	49.00 ps
E	20.00 µs	20.00 µs	20.00 µs	47.09 ps

Table 1: Results from DTU output pulse testing. Letters in the measurement column refer to the markers in Figure 27.

ble 1. Taking this into account, the mean delay between the 1 kHz clock being received by the module is around 150 ns which matches the processing delay from the module's synchronous logic. As the 1 kHz clock signal cannot be guaranteed to have a fixed phase relationship with the internal 50 MHz quartz oscillator, it is reasonable to expect jitter up to the 20 ns period of the FPGA's clock. Therefore the 6 ns jitter measured seems erroneous, especially considering the 16 ns of jitter on the 1 kHz clock signal received by the module. A reasonable explanation is that over the number of output samples taken, any jitter present on the module's output is compensating for some of the jitter on the 1 kHz clock input.

Since the DTU will be mainly used to trigger devices to a millisecond level of accuracy, these measurements confirm that the unit performs to specification.

9.2.2 Ouput pulse jitter from F_{REV}

A second test was performed for the operational mode where the output is synchronised to the F_{REV} pulse. This was conducted in an identical manner to the measurement of jitter from the 1 kHz clock aside from the F_{REV} synchronisation mode being enabled. The jitter between the F_{REV} pulse and the output pulse was measured and was found to have a mean of 25.71 ns and a standard deviation of 73.78 ps.

10 Suggestions For Future Work

While the unit performs to specification, there are a couple of areas that have been identified where improvements could be made to the DTU and it's supporting infrastructure. These will be discussed in this section.

10.1 MMI Target serial link

The first area for improvement concerns the serial packets that are used to trigger the unit. Currently, there is not a one-to-one mapping between the cycle playing in the SPS and MMI Target number that is generated, for example, some cycles share a MMI Target. For the operator, this poses a problem if they wants the unit to trigger on the combined cycles independently.

This is not a simple problem to remedy, as other devices in the LLRF system rely on the current MMI Target-to-cycle mapping. The simplest solution would be to add a second serial link dedicated to the DTUs that could offer a one-to-one mapping. However, adding this would require an investment in extra hardware, in the form of CTRV cards, a serial encoder and additional cabling. It would have the benefit of being transparent to existing hardware that could continue to operate with the current serial link.

A second solution would be to implement a one-to-one mapping between the MMI Target and the cycle playing in the machine. Implementing this solution would require no extra hardware or cabling, but would require current hardware to be updated.

Either way, the DTU's hardware is generic, as all logic related to the reception of serial packets is implemented in the FPGA, making it possible to adapt the unit to any changes that are made the MMI Target system in the future.

10.2 MMI Target names stored in the VHI

To update the list of MMI Target names, used by the VHI, currently requires the the firmware on each VHI to be re-flashed. To access the JTAG connectors on the VHI involves removing the DTU from it's crate. The original specification for the DTU requested a port on the front panel, however no suitable connector was found to fit in the constrained space available on the module's front panel. As the names of the MMI Targets may change every year, keeping all DTUs up to date could be a time consuming process with the current setup.

A possible solution would be to use specially encoded serial packets to update the list of names stored in the VHI interfaces. To implement this would require a major upgrade to the VHI firmware with two features. Firstly, the name of each item in a 'list' type parameter would have to be addressable by the FPGA. For simplicity, these could share the seven bit address space that the current parameters use. However, this would limit the total number of parameters and list items to 128. An alternative would be to assign each list an address in this space and then have a sub-address to access each item in that list. Secondly, the FPGA is currently unable to send data to the VHI other than for read-back type parameters. If it were able to send data for any parameter, then it would be possible, combined with having the list items addressable as parameters, to update the lists from the FPGA.

Secondly, there would need to be a method to transmit the cycle names over the serial link which could be done in a number of ways. The simplest would be a serial link 'master' device sitting between the DTUs and the current serial chain. A possible solution would be a NIM module, based on the Serial Link Router hardware discussed in Section 20, whose VHI had a parameter to set each of the target names. These could be sent as specially encoded serial packets from which the DTUs could update their locally stored lists.

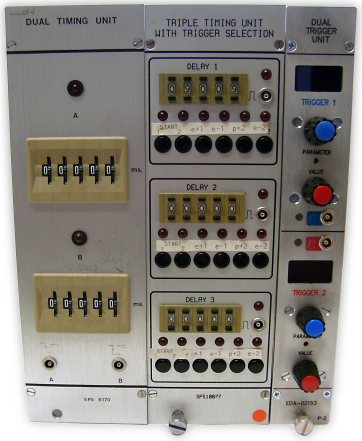


Figure 28: The Dual Trigger Unit next to two older models of trigger units.

A second solution would be to design a new VME card that could be controlled by software to send the correct packets. The card could also listen to the timing network and generate the serial packets automatically based on the upcoming cycles. With this ability, it could solve the previously discussed problem of MMI Target-to-cycle mapping as well. However, this would require new hardware to be designed and constructed whereas the previous solution would be able to utilise current hardware.

11 Conclusion

A modern solution to producing delayed trigger pulses in the low-level RF system for the Super Proton Synchrotron is presented here. The new Dual Trigger Unit has a number of advantages over the modules that it replaces, shown in Figure 28. The most significant of these is the ability to automatically handle changes in the SPS's super-cycle structure. Previous trigger units use delays that are counted from the

start of the super-cycle, so if the structure of this is altered then the delays need to be reprogrammed. The Dual Trigger Unit listens to the MMI Target serial link which broadcasts the number of the next cycle playing in the machine. By using this as a trigger source, the unit is unaffected by changes the super-cycle structure and will trigger in the correct place regardless of any changes.

Due to the unit being based on a FPGA, it also offers the scope to be updated in order to deal with any changes that are made to the LLRF system in the future. In the coming years there are plans for significant upgrades to the SPS so this flexibility could be vital.

While there are still some areas in which the infrastructure supporting the unit could be improved, the current solution is already huge benefit to the operators, who have responded very positively to the new units. Twenty modules have been produced so far, and these are currently awaiting installation in the Super Proton Synchrotron.

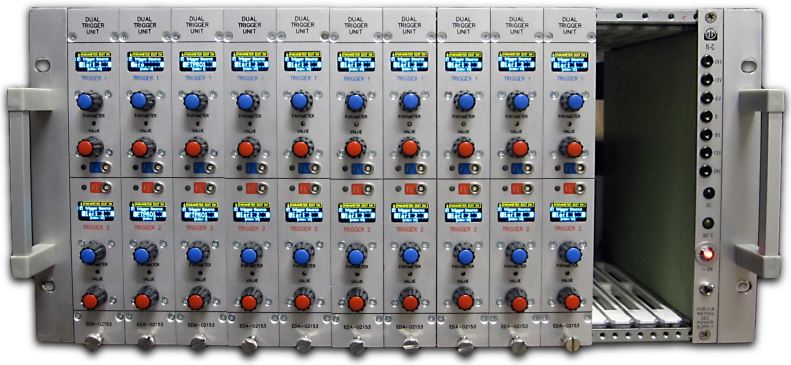


Figure 29: The first ten, production, Dual Trigger Units in a NIM crate.

Part II

VME Peak Detector

12 Introduction

The VME Peak Detector (VPD) is a VME card for the LHC's beam observation system. The card receives an input signal from a beam pickup that is of the 'wall current monitor' design [18] and consists of a resistive gap in the beam pipe. When the charged particles move past the gap, a voltage is induced across it. Each bunch of particles has a nominally Gaussian distribution and, because of this, the voltage sensed by the card has a Gaussian shape in the time domain. The card calculates and stores the peak value of the pickup voltage and allows an operator to view this information over a long period of time. By observing the variations in the peak value of the signal during this time, the stability of the beam can be inferred. If the beam is unstable, then the peak value will oscillate over a number of turns due to movement of the particles within the bunch [19, 18].

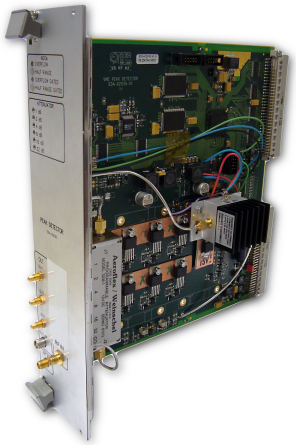


Figure 30: Modified VME Peak Detector prototype.

The VPD card has been in development for a long period of time and has been worked on by a number of designers within the BE-RF-FB group. The hardware had already been designed and an initial prototype had been constructed during 2009. The hardware design consists of a diode peak detector circuit that is fed with the pickup signal. The output of this detector is digitised and stored in memory, where it can be read out by the VME host computer. While the hardware was, essentially, complete, a number of changes were required due to problems found during testing of the prototype. Initial work on the firmware had been started by another designer but changes in the operational requirements of the card have required a complete rewrite.

13 Hardware Overview

The VPD's hardware design can be broadly broken into two parts: an analogue RF frontend, consisting of the diode detector, amplifier and attenuator; and a digital backend, consisting of the analogue-to-digital converter (ADC), FPGA, external memory and interface to the VME bus.

13.1 RF frontend

The RF frontend is responsible for conditioning the input signal and is illustrated in Figure 31 on page 29. The signal from the beam pickup is received on a front panel SMA connector that is wired directly to an Aeroflex programmable attenuator [DS15] which allows the input signal level to be reduced by between 0 dB and 63 dB in 1 dB steps. Due to the different intensities of beam that can be present in the LHC,

this range is necessary in an operational context to provide protection to the rest of the card against the possibility of a large input voltage.

After the attenuator, the signal passes through a Mini-Circuits KSWHA-1-20+ RF switch [DS16] that is digitally controlled by the FPGA, and can be toggled on a bunch-by-bunch basis. Using this switch, the input signal can be let through to the peak detector to pick out a single bunch of particles in the machine and allow it to be studied, in isolation, on multiple passes of the pickup. After the switch, the signal passes through a Mini-Circuits ZFL-2500VH+ [DS17] RF amplifier to increase the signal level by approximately 25 dB and provide an optimal signal level to the diode peak detector. The detector consists of a HSMS-282C RF Schottky diode [DS18] which, when positively biased by the input signal, charges a 940 pF capacitance. Once the input signal has dropped back to zero, the capacitance is allowed to discharge through a 1 MΩ resistor giving a time constant of just under 1 ms. The peak detected signal is provided via buffered outputs on the front panel of the card, allowing the signal to be analysed by external equipment. The signal is also fed to the ADC to be digitised and stored in memory.

13.2 Digital backend

A simplified schematic of the VPD's digital backend is illustrated in Figure 32 on the next page. The peak detected signal is digitised by a 14 bit Analog Devices AD9240 [DS19] ADC that runs at 3.3 Msamp/s, so that a sample is taken once every twelve RF bunches[8], or approximately once every 300 ns. Although the VPD is designed to be able to observe any bunch in the machine, the time constant of the diode detector is just under 1 ms and, because of this it is not necessary for the ADC to sample in every bucket. Even if an ADC sample is only taken an entire turn (3654 RF bunches or approximately 90 µs) after the input bunch then there will be approximately 90 % of the initial signal present at the output of the detector.

The digitised signal is read by the module's FPGA, an Xilinx Spartan XC3S700A [DS20] which implements the card's functionality as digital logic. Selected samples from the ADC are stored into a pair of onboard 4 MiB static RAM (SRAM) chips [DS21] that are double banked to provide storage space for up to two million samples between readouts. The module is capable of providing continuous acquisition, as one memory bank can be written to while the other is read from. The operation of the memory controller will be described in Section 15.4.

The card's FPGA communicates with a host computer over a VME bus. The host computer reads to and writes from configuration registers in the FPGA that control the card's operation. The VME bus also allows the contents of the SRAM to be read out by the host computer.

The FPGA uses two clock sources, a 40 MHz, bunch synchronous, clock received over the VME backplane and a local 50 MHz crystal oscillator. The 40 MHz clock is divided by twelve to provide the ADC clock and since it is synchronous to the bunches in the machine, this ensures that the ADC samples at fixed points in each turn of the machine which is essential to provide accurate readings of the peak detected signal value.

[8]The LHC uses a 400 MHz RF system which results in a total of 35 640 possible RF buckets. For operational reasons, only every tenth bucket is filled which gives the possibility of 3564 RF bunches in the machine, each spaced by 25 ns. However, due to requirements for beam injection and extraction, a maximum of 2808 bunches can be filled.

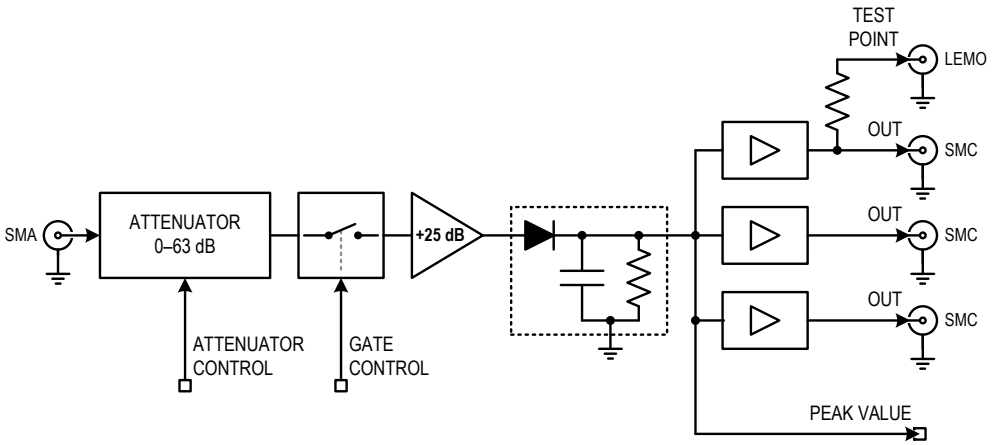


Figure 31: Simplified RF frontend of the VME Peak Detector.

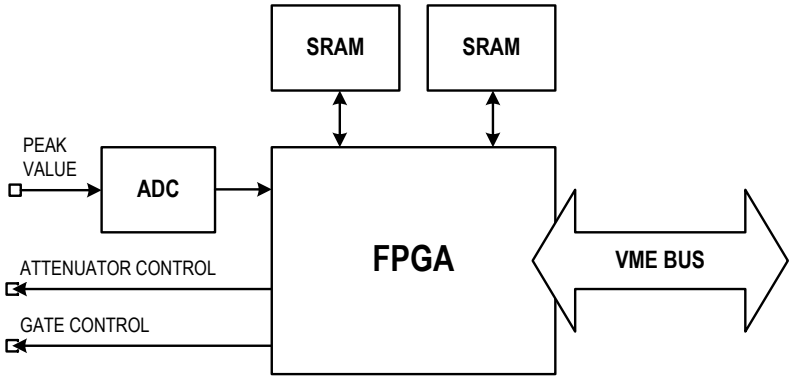


Figure 32: Simplified digital backend of the VME Peak Detector.

13.3 LHC LLRF VME standard

The LHC LLRF system is built around cards based on the VME electrical standard that are built in Eurocard-standard dimensions. The VME bus is an extension of the Motorola 68000 microprocessor's bus protocol [23]. Each card is connected to a 'backplane' which propagates an address bus, a data bus, interrupt lines and other signals between the cards.

The LHC LLRF system uses custom VME crates[9] that offer four 32 bit slots for commerical VME cards and fifteen 16 bit slots for cards that are designed specifically for the LLRF system [22]. The custom cards plug into a backplane which uses a standard VME 'P1' connector to provide a 24 bit address bus and 16 bit data bus to all modules. A custom 'P2' connector provides distribution of power supplies, clocks, timing signals, a LVDS data bus, a serial link and has a built in JTAG chain to allow remote programming of the cards. The custom cards are deeper than standard VME cards at 220 mm compared to 160 mm and the extra depth allows space on the board for large RF components such as amplifiers and filters.

In the standard crate configuration, a host computer running Linux sits in the first slot of the 32 bit section and is the master device. The computer is connected to the network and runs software that controls all other cards in the crate by writing into registers in each card over the VME bus. Each card be configured, in real time, by the LHC's software control system. The VME bus' 24 bit address space is mapped to cover the address space of all card in the system. The top four bits of the address bus select a card, while the remaining twenty bits select a register or memory within this card giving each card a 1 MiB memory space to store registers and provide access to external memory.

14 Hardware Changes

A number of hardware changes have been required since the prototype of the VPD card was produced and these will be described in the following section. The prototype schematic is included in Appendix F.1 in an unmodified form for reference. The modifications to the RF frontend are included in Appendix F.2. Some changes are required to other parts of the schematic, but none of these are major and these have not been included.

14.1 Changes to RF frontend

The prototype for the VPD originally contained two independent diode peak detectors. The input signal was split after the attenuator and one branch was fed to a peak detector via a gating switch, while the other was fed directly. Both branches used a Mini-Circuits ZPUL-30P [DS22] amplifier with a 700 MHz bandwidth and identical peak detectors. During testing of the prototype, it was discovered that the bandwidth of the amplifier was not high enough to properly observe fast oscillations in the beam. The replacement of a ZFL-2500VH+, which has a 2.5 GHz bandwidth, had already been chosen by the previous designer. It was also found that the current drawn by two amplifiers is too much for the power supply of the VME card. To reduce the load, the non-gated detector and it's associated amplifier have

[9] A rack mounted enclosure that holds multiple VME cards.

been removed, leaving a single, gated, peak detector on the board.

The omission of the second amplifier has a secondary benefit. The pair of ZPUL-30P amplifiers had to be mounted on standoffs above the card's PCB to allow the board space underneath them to be utilised for other components. Despite using a low-profile heat-sink, the height of the amplifiers required the card to be 12TE[10] wide. A single ZFL-2500VH+ amplifier has a smaller footprint than a single ZPUL-30P and this will allow the amplifier to be mounted directly to the PCB. By using a low-profile heat-sink, such as the Fischer Elektronik SK 81/50 [DS23], the reduction in height of mounting the amplifier directly to the PCB will allow the card to be reduced to a 8TE wide front panel.

After making the aforementioned changes, the signal at the output of the the gating switch had a small amount of 500 MHz noise and this remained even when the input to the switch was directly terminated with a 50 Ω resistor. The switch is driven by a pair of high bandwidth THS3202D op-amps which invert and amplify the CMOS outputs of the FPGA to meet the switch's requirement of a pair of differential drive signals at −8 V and 0 V. The outputs of these op-amps were found to be oscillating and this signal was coupled into the main RF signal path resulting in the 500 MHz noise at the output. The oscillation is caused by the capacitive load at the op-amp's output caused by the gate of the MOSFET switch. As it takes time to charge the capacitance, this alters the voltage sensed by the op-amp's feedback loop and, due to it's high bandwidth and fast slew rate, the it attempts to compensate for the lower feedback voltage which causes the output to oscillate. A 51 Ω resistor has been added in series with the output of the op-amp to provide decoupling from the load. The value of the feedback resistors has also been increased slightly, and these two measures are enough to stabilise the amplifiers and eliminate the noise.

14.2 Other changes

The ZPUL-30P required a 24 V power supply that was generated by an on-board boost converter stepping up the card's 12 V power supply. As the replacement ZFL-2500VH+ amplifier requires a 15 V supply, the regulator has been modified by altering the feedback resistors to provide the correct output voltage. As the supply operates as a boost converter, when it is deactivated the 12 V DC voltage passes directly through the inductor and is present at the output. To allow the RF amplifier to be deactivated an IPS7081 [DS24] high-side MOSFET switch has been added at the output of the supply and is driven from the FPGA via a 2N7002 transistor.

The 15 V supply is sampled by a 'diagnostic' ADC and this reading is used to ensure that the supply voltage to the amplifier is correct. As the MAX1270 ADC [DS25] has an input range of 0 V to 5 V, the voltage is reduced using a resistor divider network to provide a 2.5 V output for the nominal 15 V supply. However, when testing, it was found that the input current of the ADC causes a the voltage out of the potential divider to sag and decrease the voltage sensed by the ADC. As this measurement needs to be fairly accurate to be useful, a unity gain op-amp buffer has been added between the potential divider and the ADC's input in order to supply the current required.

[10] 1TE is approximately equal to 5 mm. Hence, 12TE is approximately 60 mm wide. VME cards are built with front panels that are multiples of 4TE wide.

15 Firmware Design

The VPD's firmware architecture is shown in Figure 33 on the facing page. The control logic is split into two clock domains: a bunch synchronous 40 MHz clock; and a 50 MHz crystal oscillator. The 40 MHz domain is used for all operations that have to be synchronised to the bunches of particles in the machine, such as the ADC sampling control and the bunch gate control. The 50 MHz domain is used for communication over the VME bus and register control. As the ADC only samples once every twelve bunches, there is time for the data from the ADC to be resynchronised into the 50 MHz domain immediately and this allows the memory controller to be simplified significantly as both the VME bus and ADC access to memory is performed in the same clock domain. Individual blocks of the firmware will be discussed in the following section and the VHDL code and block diagrams are included in Appendix G.

15.1 VME interface

Communication with the VME bus is handled by a controller block that is common to all cards developed for the LHC LLRF system. The block provides address and data busses, along with read and write strobes, that are used to interface with control registers defined within the FPGA. Each card has 2^{20} memory addresses available, as the top four bits of the 24 bit VME address bus are used to select one of the 15 cards in the crate. However, the VME standard defines that each address refers to an 8 bit memory location. The VME bus is 16 bit wide, so the memory is addressed with bit zero of the address bus held at zero and this results in 2^{19}, 16 bit, memory addresses being available to each card.

The software that runs on the VME host computer and controls the card is written by a separate designer, so a 'memory map' has to be defined for the card. The document, included in Appendix H, defines all registers that are used along with their function. A number of standard registers are implemented identically in all LLRF cards in order to simplify the development of the control software. Although some of the functionality of these registers is not used for the VPD, they are all implemented fully.

The card's external SRAM chips can be accessed using the top half of the card's address space which gives a 512 KiB area of memory. To map the whole 4 MiB external memory to this area, a 'view port selector' is defined which allows different slices of the external SRAM to be connected to the VME bus.

Control registers are defined using standard blocks: CtrlRegN, for normal control registers; and RMWReg for special registers that use the upper eight bits as a write mask for the lower eight bits. The VME write data bus is distributed to all registers and a write selector, CRegWrSel, sends the write strobe on the correct register. The control registers, read-only registers and external SRAM are multiplexed to the VME bus in a three stage process, as shown in Figure 34. First, the outputs from the control registers are multiplexed together using the block CRegRdMux and the output from this multiplexer is combined with the output from any read back only registers using the block RegRdMux. Finally, the output from the registers is multiplexed with the external SRAM using the block VmeMux which also distributes the read and write strobes to the correct destination and returns the confirmation strobes to the VME controller.

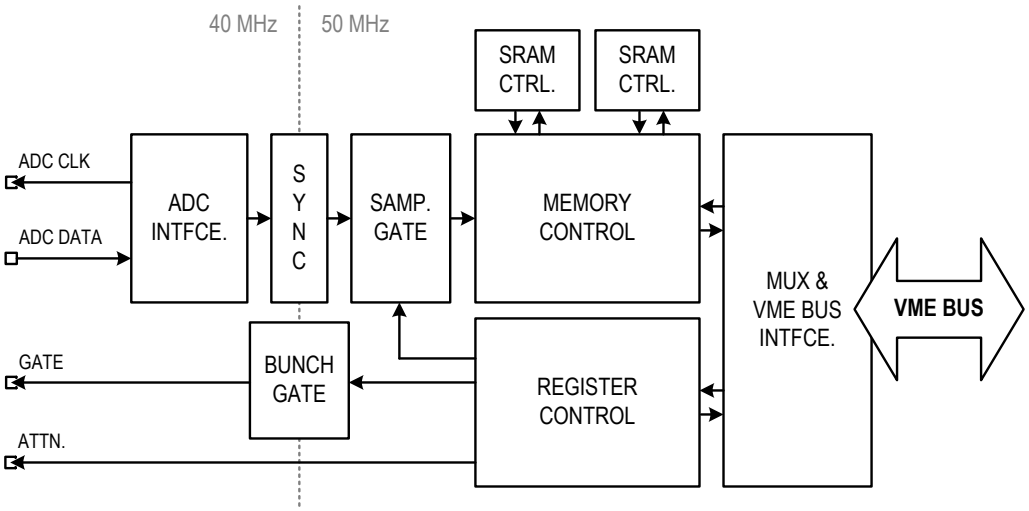


Figure 33: Simplified block diagram of the VPD's firmware.

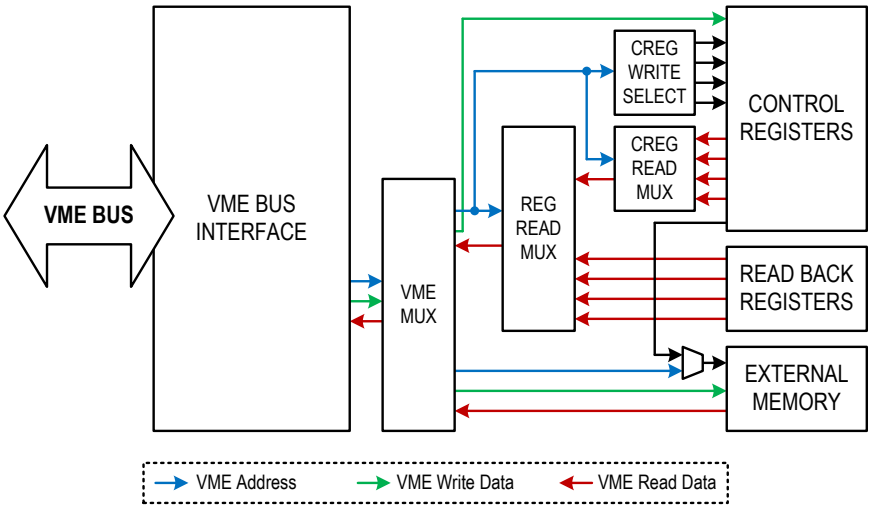


Figure 34: Simplified block diagram of the three stage multiplexing scheme used to connect the VPD's external memory and registers to the VME bus.

15.2 ADC acquisition

The ADC clock is generated using a CLKGEN block that divides the bunch synchronous 40 MHz clock by twelve to generate a 3.3 MHz clock resulting in the ADC sampling once every twelve bunches. The rising edge of the clock is resynchronised by the F_{REV} pulse to ensure that a sample is always taken in bunch zero. The edge of the ADC clock is delayed by half a 40 MHz clock cycle in order that the sample is taken in the middle of each bunch.

A sample counter is incremented on every positive edge of the ADC clock and reset when the F_{REV} pulse is received. The output from this counter is clocked through three flip-flops, on the positive edge of the ADC clock, to compensate for the ADC's pipeline delay. The sample number and ADC data are combined and latched into a flip-flop by the negative edge of the ADC clock, which is when the data is valid.

15.3 Bunch and sample gating

Two 'masks' are implemented to allow control of the RF switch (the 'bunch mask') and the ADC sample storage (the 'sample mask') over a full machine turn. The masks are implemented as 'bit-arrays' with a each bit in the array representing a single bunch or sample. The bit-array is iterated through and wraps back to the start once the end has been reached. The bunch mask is 3564 bit long, corresponding to one bit for each bunch in the machine. When there is a one in the mask for a particular bunch, the RF switch is enabled and allows signal through to the diode peak detector. The sample mask is 297 bit long, as an ADC sample is only taken once every twelve bunches, and controls whether a sample is stored in SRAM memory or discarded.

Both masks are implemented using the FPGA's 'block RAM' which is an area of the FPGA dedicated to memory functionality. Each mask is configured as a double-banked, 16 bit, memory and is 256 words deep. One bank is 'active', and is used to read back the mask, while the second is 'inactive' and can have it's contents updated by the VME host computer. The banks are switched by writing to a bit in a control register and this allows an entire new mask to be set up without disturbing the active mask. A twelve bit address is used to access the individual bits within the mask. The top eight bits are used to select a word from the block RAM while the bottom four bits are used to select an individual bit within this word. A counter can then be used to increment the mask address and retrieve the single bit mask from memory. Due to the architecture of the block RAM there is a single clock delay between asserting an address and receiving the mask out.

The single clock delay is not a problem for the sample mask, as a flip-flop can be used to delay the ADC data by one clock cycle in order to synchronise it with the mask. However, for the bunch mask, the delay would result in the mask being applied one bunch too late as shown in Figure 35(a) on the next page. In this example, the mask address is reset to zero by the F_{REV} pulse and, because of the delay, the mask appears one bunch too late. The solution to the problem is shown in Figure 35(b). In this case, the address counter is loaded with a value of one by the F_{REV} pulse and wraps back to zero after it reaches 3563. As the address trails the bunch by one sample, the one clock delay causes the mask to be correctly

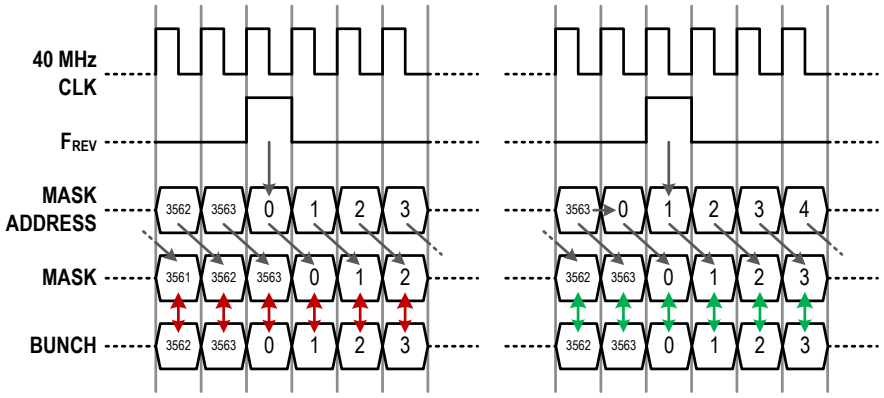


Figure 35: Alignment of bunch mask with and without correction.

aligned with the bunch. The only case where this method would fail is if the F_{REV} pulse occurred at some point other than every 3564th 40 MHz clock period. If this were to happen, the first mask would be incorrectly applied, although this will never happen while the machine is in operation.

In reality, a two flip-flop resynchronisation is used on the mask output to remove glitches that are caused when the block RAM word is changed. As the resynchronisation adds a two clock delay to the mask output, the address counter is loaded with a value of three rather than one by the F_{REV} pulse.

15.4 External memory control

The FPGA stores samples from the ADC in a pair of external SRAM chips that are double banked. As with the mask RAMs, one chip is always active and is used to record ADC samples, while the other chip is inactive and can be read over the VME bus. Memory access is controlled by the firmware block MemCtrl, which handles the overall read and write process and bank selection. The SRAM chip that is used requires that it's data bus is delayed by one clock cycle compared to it's address bus, so each memory chip has an independent controller, implemented by the block SramCtrl, to synchronise it's address bus, data bus and control lines.

A standard has been developed for LLRF cards whereby each card has an 'observation' memory and a 'post-mortem' memory. The observation memory is used to retrieve data from the card during normal operation while the post-mortem memory is kept as a record of the last couple of seconds of operation. If there is a problem with the machine, the post-mortem memory of each card can analysed to determine it's source. In most LLRF cards, each type of memory is continually updated until the VME host computer issues a 'freeze' signal. At this point, acquisition ceases and the memory chip is connected to the VME bus to enable it's contents to be read out. The VPD does not operate in the standard manner, as it does not have a post-mortem memory and uses a double banked observation memory to provide continuous acquisition. However, the memory has been designed in a way that makes this transparent to the host computer and, because of this, the memory is controlled using the standard observation memory registers. Two operating modes exist, which are defined by whether the observation memory is frozen by the card's FPGA or by the host computer.

In the standard operating mode, the FPGA will write samples from the ADC into memory until a certain number have been recorded. Once this limit is reached, the observation memory is frozen automatically and an interrupt is sent to the host computer to inform it that data is ready. Meanwhile, the FPGA switches to the other memory bank and continues to record samples. Once the host computer has finished reading back the samples in the inactive bank, it must release the observation memory. If this has not been done by the time the FPGA is due to switch banks again, the active bank will be discarded and overwritten to give time for the VME host computer to catch up. If this occurs, an error flag is set to warn the user that data has been discarded.

A second operating mode exists, where the host computer is responsable for manually requesting an observation freeze, which is more akin to a standard LLRF card. In this mode, the FPGA will continue to record data until a freeze is requested or the memory chip has been filled with samples, at which point recording pauses. Once an observation freeze has occurred, a bank switch takes place and acquisition will restart. In both operating modes, the MemAddrCnt block is responsible for incrementing the memory address after each successful write and triggering a observation freeze when appropriate.

Finally, a diagnostic mode exists where the memory chips can be directly read from and written to by the VME bus to serve as a test vector and allow verification that the memory chips are working correctly. In this mode, the MemAddrCnt controller is bypassed by the MemWrSw block and the VME bus address and data lines are connected directly to the memory. The bank can then be manually switched using a register bit, in order that both memory chips can be tested.

16 Testing and Verification

16.1 Testing of the RF frontend

Initial testing of the modified RF frontend has been performed. To check it's gain, the RF amplifier was directly connected between a pulse generator and an oscilloscope and the peak value at the output was measured and compared to the peak value at the input. The amplifier was found to have a gain of 21.75 dB, which is in the expected range of 20 dB to 25 dB.

To find the 1 dB compression point of the amplifier, the pulse generator was then connected to the card's front panel input with a variable attenuator which was varied with a range of -30 dB to -15 dB to give a peak voltage deviation of 0.16 V to 0.9 V at the input of the amplifier. The output of the RF amplifier was measured with an oscilloscope, as before, and the peak value recorded. The results are shown in Figure 36 on the facing page and suggest that a usable peak voltage range at the input of the module is 0.4 V to 0.6 V.

The analogue output of the peak detector was also compared to current peak detectors that are adapted from the SPS. These are purely analogue units that do not offer the digitalisation and storage that the VPD offers. The results of this test are shown in Figure 37 on the next page. From this we can see that the VPD and the SPS peak detectors both have a similar response to a single bunch input. However, further testing will need to be performed to ensure that the VPD now has sufficient bandwidth.

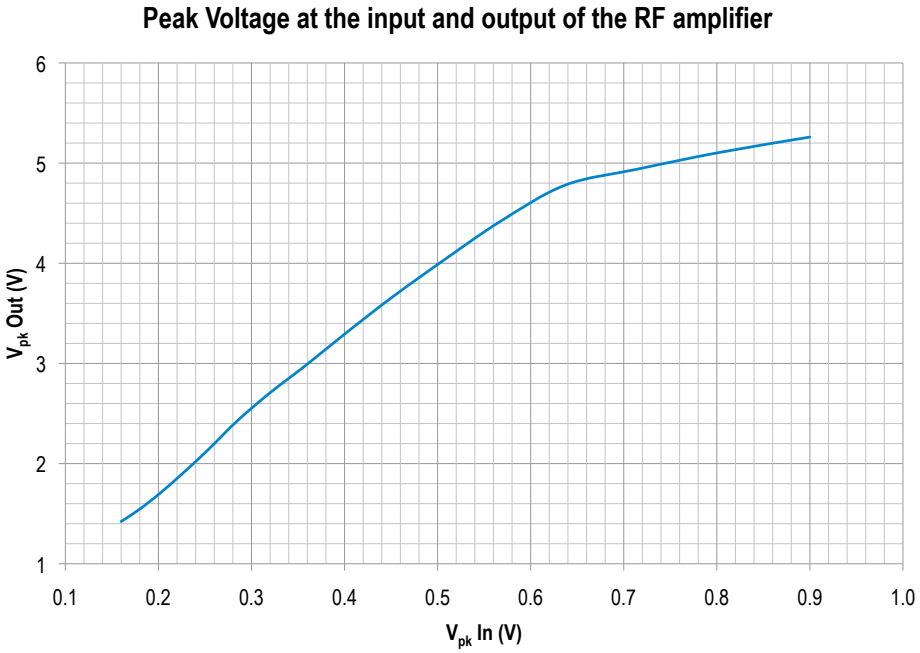


Figure 36: Voltage in and out of RF amp.

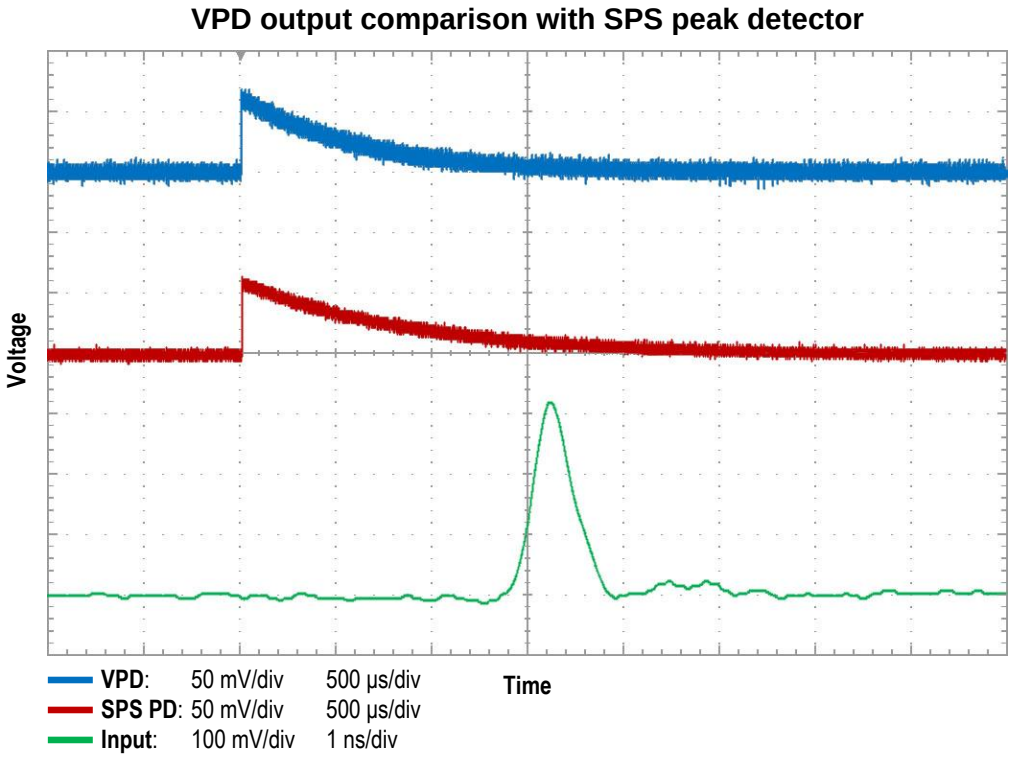


Figure 37: Comparison of the outputs of the VME Peak Detector and the currently used SPS peak detector. Also shown is the input pulse used to generate the output trace. Note that this was recorded on a separate occasion and has no relationship in the time domain to the output pulses shown on the graph.

16.2 Testing of the ADC

To test the operation of the ADC, a DC voltage was directly injected into the diode detector. The input voltage was varied and the ADC value was read over the VME bus. To test the full ADC range, the input voltage was ramped from 0 V to 6 V and the voltage produced at the output was measured. The test setup is shown in Figure 38 on the facing page and the results are shown in Figure 39 on the next page with the graph normalised to 5 V and the ADC's full scale value. The voltage to ADC value mapping is as expected, with a linear increase in voltage and saturation above the reference voltage of 5 V. To achieve the same value out of the ADC, the input voltage is required to be slightly higher than the output voltage. This is also as expected, due to the voltage dropped by the diode in the peak detector. From this graph, it is clear that the ADC is working correctly.

16.3 Testing of the firmware

The operation of the firmware was tested using a SSH connection to the VME host computer in order to write to, and read from, the VME registers of the card. Testing from the command line was necessary because the card's control software has not been written yet. Signals in the FPGA were patched out and observed using a LeCroy Mixed Signal Oscilloscope. With this, the synchronisation of the 40 MHz clock, ADC clock, bunch and sample mask and F_{REV} pulse were verified. The command line interface also allowed the operation of the memory controller to be checked by manually writing values into the memory control registers and observing the memory contents when a known voltage was applied to the ADC's input. Further and more extensive testing to the modules firmware will be carried out once the control software has been written. A graphical user interface will allow more extensive testing to be performed without the need to type individual commands into a command line, a process which is slow and prone to errors.

17 Suggestions For Future Work

As of the time of writing, the VPD is still awaiting final testing of the hardware modifications, as described in Section 14. Once further testing has been completed, and any final changes made, a second prototype PCB will be commissioned from the CERN Design Office and further testing will have to be completed to ensure that the hardware and PCB designs are working correctly. The firmware will also have to be completely tested once the drivers and software to control it have been written.

18 Conclusion

While the VME Peak Detector is still at a prototype stage, significant progress has been made to the project. A number of hardware changes have been made to the initial prototype PCB and testing on these is well advanced. The firmware for the card has been rewritten from scratch and is also undergoing testing while awaiting the card's control software.

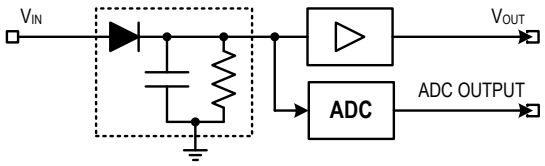


Figure 38: Hardware setup for VME Peak Detector ADC test. The ADC value is read back via the VME bus.

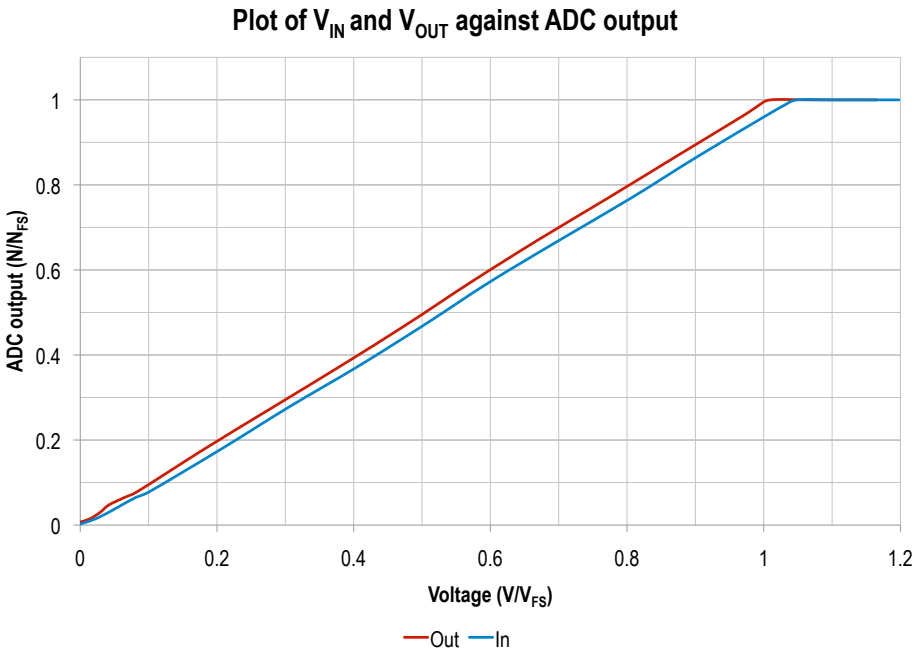


Figure 39: Results of the VPD ADC test. Both V_{IN} and V_{OUT} are plotted against the output from the ADC. Values are normalised to $V_{FS} = 5\,V$ and $N_{FS} = 2^{14} - 1$

Part III

Other Work

19 SerialLink[16] Controllers

A proprietary serial link is used for inter-module communication in the LLRF systems of the LHC and SPS accelerators [20]. The standard, called 'SerialLink[16]', was developed in the early 1990s to allows data to be sent up to 100 m over a single coaxial cable. The data is sent as a series of bits with 2 V representing a logic high and 0 V representing logic low. Each transmission packet is preceded by a -2 V start bit and a short recovery pulse. Most SerialLink[16] connections use a 16 bit packet size, however some now use a elongated 32 bit packet as well. Since a serial packet can, theoretically, pass through many modules, each with their own clock domain, the serial packet must be re-synthesised by each module to keep the integrity of the bit timing.

For the LHC, the standard was redeveloped to improve speed and reliability. The new standard, known as the 'Serial Control Link', is broadly similar to the 'classic' link, but features shorter bit times and has a 16 bit cyclic redundancy check (CRC) appended to each packet. A summary of the difference between the two standards is shown in Table 2 on the following page. Figure 40 shows the the packet timing of SerialLink[16] and Figure 41 shows the Serial Control Link.

While a VHDL block that implements the Serial Control Link protocol exists for inclusion in LLRF module firmware, no such block was available for SerialLink[16]. Some previous modules for the SPS used a modified version of the Serial Control Link blocks, however these did not adhere strictly to the standard. Therefore, after discussion with John Molendijk, the designer of the serial standards and the Serial Control Link blocks, it was decided to adapt the current controller blocks to be usable for both connection standards.

The controller blocks for the Serial Control Link consist of a state-machine, to handle the packet timing, that is connected to an external shift register and CRC generator. To be useable for both standards, 'generic' parameters have been added to the state-machines. The timing of the link is generated by counting 50 MHz clock cycles meaning that the only change required is to make the number of cycles counted variable by a generic. Since the Serial Control Link state-machine uses an external CRC generator block, this can been omitted from the SerialLink[16] controller.

For the transmitter side, the addition of generics to cover the differences in bit timing, recovery time and packet bit-length, is sufficient to obtain a working solution. However, for the receiver, things are slightly complicated by the fact that the receiver has to re-generate the data before repeating it. In the existing Serial Control Link blocks, the receiver starts repeating 40 ns after the rising edge of the incoming start pulse. The length of the recovery pulse is designed so that, using this method, the sampling point is set at 260 ns into the 300 ns data bit, as shown in Figure 42 on page 44. Due to the longer bit time of the SerialLink[16] standard, coupled with the shorted recovery pulse, this method will not work correctly. The solution is to add a certain amount of pre-delay between the rising edge of the start pulse

	SerialLink[16]	Serial Control Link
Start Bit Time	240 ns	240 ns
Recovery Time	80 ns	140 ns
Data Bit Time	640 ns	300 ns
Stop Bit Time	320 ns	320 ns
Data Length	16 bit	32 bit
CRC Length	—	16 bit
Packet Time	10.9 µs	15.1 µs

Table 2: Comparison of SerialLink[16] and Serial Control Link.

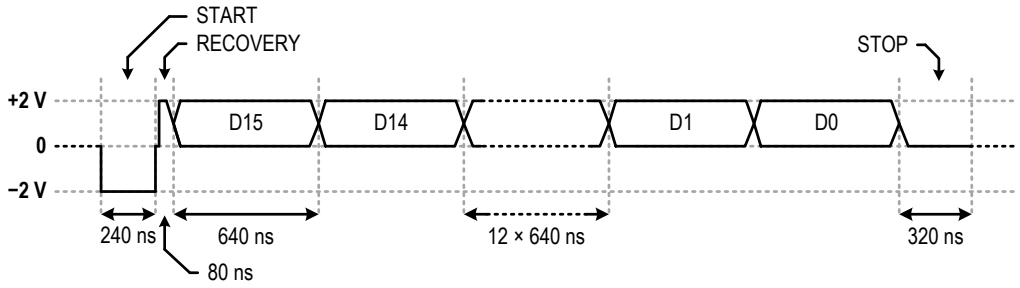


Figure 40: SerialLink[16] timing diagram.

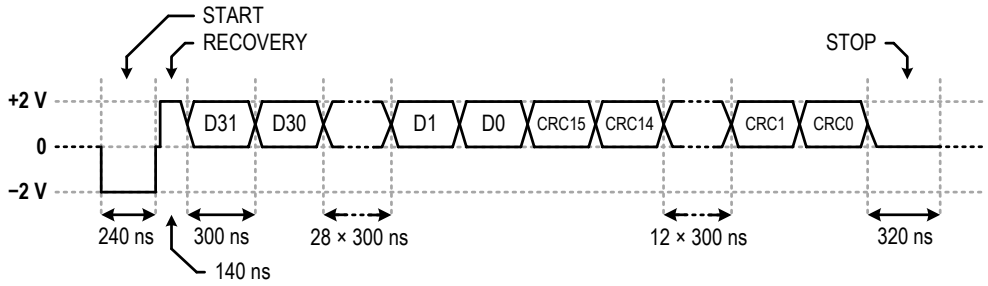


Figure 41: Serial Control Link timing diagram.

and the beginning of the repeated data, as shown in Figure 43 on the following page. The amount of delay added is calculated automatically to keep the sampling point 40 ns before the end of the each bit.

However, the addition of the pre-delay also adds a further complication. The 320 ns stop bit of the Serial Control Link transmitter is sufficient that, even if the transmitter sends packets back-to-back, the receiver has completed repeating the packet before it has to begin receive a new one. Due to the longer bit time of the SerialLink[16] standard, and the added pre-delay, the 320 ns stop bit is not sufficiently long. Two options exist for fixing this: increase the length of the stop bit in the transmitter; or be able to cope with a new packet beginning before the repeater has finished. To retain compatibility with current transmitters, the first option is not viable and, therefore, the second method is implemented. The worst case scenario would be if the transmitter sent packets back-to-back. In this case, the time between end of the last data bit and the rising edge of the next start pulse is 560 ns. The repeated data trails the incoming data by 620 ns due to the 380 ns pre-delay and the repetition of the 240 ns start pulse. Therefore, even in the worse case scenario, the rising edge of the start pulse can only occur during the final 60 ns of repetition as shown in Figure 44 on the next page. During the last 80 ns of repetition, the start line is observed and if it's active on any clock edge, a counter is incremented. If it is not active at all during this 80 ns period, or remains active after it, then this counter is ignored and the state-machine returns to an idle state to await the rising edge of the start pulse. However, if the rising edge occurs during this period then this counter is used to decrement the value of the pre-delay. This ensures that the repetition works correctly, even if the start pulse for a second packet is received while repetition of a previous packet is ongoing.

Throughout development of the updated controllers, the state-machines were simulated and their output compared to a unmodified Serial Control Link controller. This was vital to ensure that the new controller behaved exactly the same as the old version, which is used widely in LLRF modules in the LHC.

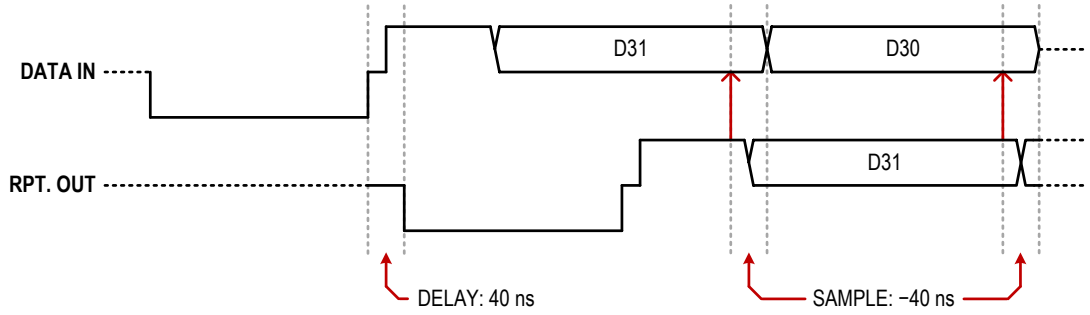


Figure 42: The Serial Control Link receiver's sampling point, 40 ns before the end of the data in bit, is marked with the red arrow. This results from the relationship between timing of the start, recovery and data bit lengths.

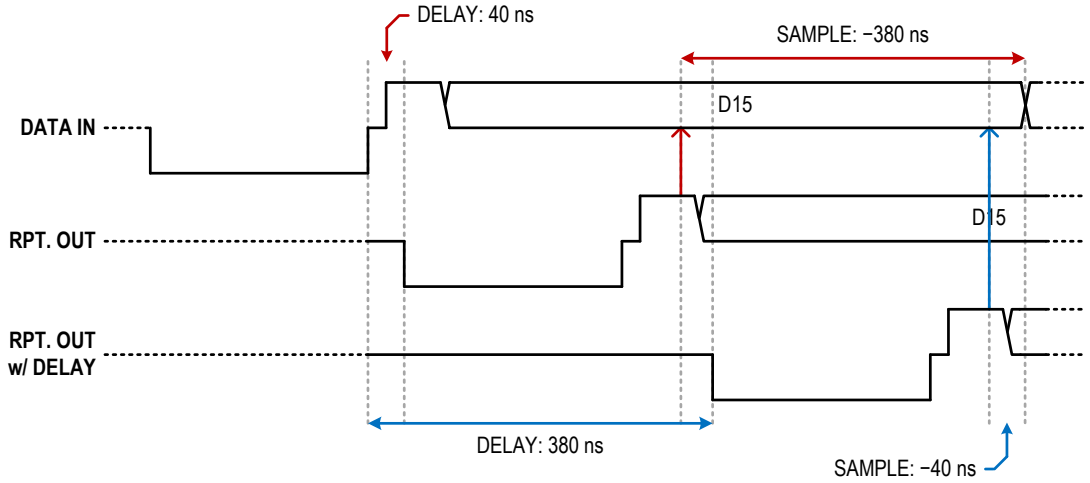


Figure 43: If the Serial Control Link method of determining the sampling point is used for SerialLink[16], then the input data is sampled too early, as shown by the red arrow. A delay is added to obtain the optimum sampling point, as shown by the blue arrow.

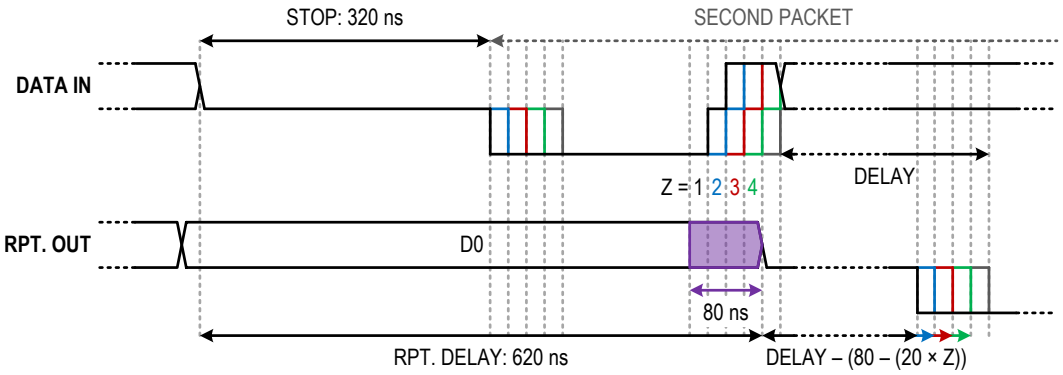


Figure 44: During the last 80 ns of repetition, marked with purple, the active portion of the start pulse is counted (Z) and this used to adjust the delay time.

20 Serial Link Router and Tester

When developing the Dual Trigger Unit, described in Part I, a secondary use for the same PCB emerged. A new tester module for the SerialLink[16] connections in the SPS LLRF system is required which would have similar hardware requirements to the DTU. The main limitation with current serial testers modules is that they only support 16 bit serial connections and do not support the 32 bit serial connections that are now being used. Currently, a modified SPS Phase Shifter has been re-commissioned as a temporary 32 bit serial tester, but more units are needed for permanent installation. Some additional features have also been requested, such as a 'hold' function which would allow the observation of rapidly changing serial data.

Figure 45: Final Serial Link Router (left) and Serial Link Tester (right) hardware.

A second, serial-related, module has been requested to allow multiple serial signals to be routed to a single output which will be used for testing cards in the LHC transverse damper system. The unit would be required to loop up to four serial signals from an input to an output and also have the ability to route any one of these to a secondary set of outputs. To accomodate both these designs it was decided that, rather than reusing the DTU design, a new PCB should be designed to have four serial inputs and eight serial outputs and omit unnecessary hardware. This new PCB will be used for the base of the new Serial Link Router (SLR) and Serial Link Tester (SLT) modules.

The majority of the hardware design is identical to the DTU with the module utilising the same FPGA, power supply and timing inputs. The major differences are: the lack of the LVTTL and isolated outputs; the lack of drivers for the piezo beeper and related 24 V power supply; and the lack of connections for a second VHI module. The module also has a total four serial inputs and eight serial outputs and eight red-green LEDs on the front panel for status indication. The schematic for this module is included in Appendix I.1 and the PCB in Appendix I.2.

While both modules share an identical PCB design, the SLR and SLT have different mechanical designs. Both are mounted in NIM chassises, but feature a different arrangement of I/O ports. The SLR variation has eight serial outputs on the back panel while the SLT replaces four of these with timing inputs. The SLT also has two push buttons added to the front panel.

Firmware for both modules is reasonably simple, and is mostly based around pre-developed blocks, such as the SerialLink[16] blocks described in Section 19. A major feature of both designs is the ability to be configured for operation with either SPS or LHC systems without having to re-flash the FPGA. To make this possible, the FPGA firmware supports both connections and selection can be made using registers in the VHI interface. These registers have the option of being available for configuration directly from the VHI menu or they can be hidden and set when the module is programmed.

The main functionality of the SLR is to allow serial input to output routing. It's four serial inputs are each directly repeated to a dedicated output. It also features an additional four outputs which, via the VHI interface, can be configured to mirror any of the four inputs. The data packets received on each of the serial inputs can be viewed via the VHI and four of the front panel LEDs show link activity. The final four LEDs can either be configured to show link activity on the routing outputs, or which input is routed to one of these outputs.

The main functionality of the SLT is to be able to view packets received on the serial link and send new packets. Only the first two the serial input are used in this configuration, while the other two simply repeat their input. Two registers in the VHI interface for each serial link allow the input data to be viewed and the output data to be programmed. A serial packet can be sent on the link by two buttons on the front panel, or by two back panel timing inputs. The received packets can also be looped to the output, to enable observation while not disrupting the serial link. A hold mode is also implemented and, when enabled, the data displayed on the VHI is frozen until a pulse is received at a timing input. Once this happens the unit is 'armed' and the displayed data will be updated once the next serial packet is received. If loop-through mode is enabled, this is not disrupted by the hold mode.

21 Maxim DS18B20 Controller

Maxim's DS18B20 [DS26] digital thermometer is used on the VME Peak Detector and will be included in many future VME cards for the LHC LLRF system. The chip serve a dual purpose of providing a temperature measurement and a unique 'silicon serial number' which can be used to identify each individual card. VME cards have previously used a Maxim DS2401 [DS27] chip to provide a serial number, however these chips do not offer a temperature sensor. Both chips communicate using Maxim's '1-Wire® Bus' protocol allowing bi-directional communication with multiple devices over a single wire.

To read the temperature and serial number from a VME card's FPGA, a VHDL controller block is required which has been split between two state-machines. The first is a dedicated 1-Wire® Bus sequencer, included in Appendix P.3, which connects directly to a bidirectional I/O bus and controls the bus timing. The sequencer implements bus timings as per the recommended values in Maxim Application Note AN126 [21].

A second state-machine, included in Appendix P.2, sequences the transmission process required to communicate with the DS18B20 chip. This block triggers the 1-Wire® Bus sequencer to perform operations in the correct order. On a bus with multiple devices, each chip can be addressed using it's serial number, however, this requires a complex initialisation procedure to scan the bus to determine the serial numbers of the devices present. Alternately, all devices on the bus can be addressed concurrently. If multiple devices were present, this would result in a bus clash when the devices tried to respond to the commands. However, for a single device, this addressing mode simplifies the communication process significantly and is implemented by the command sequencer.

A major benefit of implementing this controller as a pair of state-machines is that the 1-Wire® Bus sequencer can be reused for other chips and will significantly reduce the design effort required to write controllers for any 1-Wire® Bus devices in the future.

22 VHI SPI Controller

For communication with the VHI interface a SPI bus controller and parameter register set have been written. The SPI bus controller, included in Appendix Q.2, implements the slave side of the SPI communication protocol. Each VHI interface communicates with the FPGA as the master on a dedicated, four wire, SPI bus. Each SPI packet consists of an 8 bit header followed by up to 32 data bits. The first bit of the header is a read/write flag and the remain seven bits are the address of a register within the FPGA. The SPI controller is ignorant to the length of the data packet and will continue to receive data bits as long as the VHI continues to assert the clock signal, up to the maximum packet size. For connection to internal registers the SPI controller has an address bus and a pair of data I/O busses and a read/write line is used to strobe the data into the registers.

The register block, included in Appendix Q.1, handles parameter storage within the FPGA. The block uses 'generic' parameters to configure the address and bit length of the register. The VHI only supports data transfers of 8 bit, 16 bit or 32 bit length. To allow the use of other data bus widths inside the FPGA, the register block can be set to any size in the range of 1 bit to 32 bit and the data will be padded to nearest valid packet size before transmission to the VHI. A final generic parameter controls whether this padding should be carried out as a signed operation, by repeating the sign bit, or an unsigned operation by adding zeros. The output of the register is implemented as a tri-state driver and is keep in high impedance mode when the register is not being addressed. Because of this, multiple registers can be connected a common bus with the SPI controller which eliminates the need to use external multiplexers when using multiple registers.

References

[1] University of Glasgow, *Course Catalog: Industrial Project EE5 MQSY*. Available: http://www.gla.ac.uk/coursecatalogue/course/?code=MQSY [Accessed: Oct 30, 2010].

[2] CERN, *CERN history highlights*. Available: http://public.web.cern.ch/public/en/about/About-en.html [Accessed: Oct 30, 2010].

[3] CERN, *CERN — A global endeavour*. Available: http://public.web.cern.ch/public/en/About/Global-en.html [Accessed: Oct 30, 2010].

[4] M. Vretenar, "RF for Accelerators — an introduction," in *CAS — CERN Accelerator School, Denmark 2010*, June 2010. Available: http://cas.web.cern.ch/cas/Denmark-2010/Lectures/Vretenar-1.pdf [Accessed: Jan 05, 2011].

[5] C. Lefèvre, "The CERN accelerator complex. Complexe des accélérateurs du CERN," Dec 2008. Available: http://cdsweb.cern.ch/record/1260465 [Accessed: Jan 07, 2011].

[6] CERN, *The accelerator complex*. Available: http://public.web.cern.ch/public/en/Research/AccelComplex-en.html [Accessed: Jan 07, 2011].

[7] CERN, *PS – the Proton Synchrotron*. Available: http://public.web.cern.ch/public/en/Research/PS-en.html [Accessed: Jan 07, 2011].

[8] R. Garoby, "Multiple bunch-splitting in the ps: results and plans," Tech. Rep. CERN-PS-2001-004-RF, CERN, Geneva, Feb 2001. Available: http://cdsweb.cern.ch/record/488252 [Accessed: Jan 05, 2011].

[9] CERN, *SPS — the Super Proton Synchrotron*. Available: http://public.web.cern.ch/public/en/Research/SPS-en.html [Accessed: Jan 07, 2011].

[10] CERN, *The Large Hadron Collider*. Available: http://public.web.cern.ch/public/en/LHC/LHC-en.html [Accessed: Jan 07, 2011].

[11] CERN, *CLIC Test Facility 3*. Available: http://ctf3.home.cern.ch/ctf3/CTFindex.htm [Accessed: Jan 07, 2011].

[12] D. Valuch, "Radio frequency systems of the CERN synchrotron accelerators," in *Radioelektronika, 2009. 19th International Conference*, pp. 17–24, April 2009. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5158756 [Accessed: Dec 11, 2010].

[13] Mentor Graphics, *Visual Elite — Advanced TLM-RTL Design and Integration Platform*. Available: http://www.mentor.com/products/fv/visual_elite/ [Accessed: Oct 30, 2010].

[14] J.-M. Combe and D. Valuch, *EDMS Web Navigator: EDA-01934 SPS Phase Shifter*. CERN. Available: https://edms.cern.ch/nav/EDA-01934-V2-0 [Accessed: Oct 30, 2010].

[15] D. Valuch, *Human interface for the SPS Phase Shifter module*. CERN. Available: https://edms.cern.ch/file/983502/1/Human_interface_for_SPS_phase_shifter.pdf [Accessed: Nov 7, 2010].

[16] CERN, *TS-DEM — Development of Electronic Modules*. Available: http://ts-dep-dem.web.cern.ch/ts-dep-dem/ [Accessed: Oct 31, 2010].

[17] Markus Labs, *Website*. Available: http://markuslabs.com/ [Accessed: Nov 14, 2010].

[18] T. Bohl and J. F. Malo, "The APWL wideband wall current monitor," Tech. Rep. CERN-BE-2009-006, CERN, Feb 2009. Available: http://cdsweb.cern.ch/record/1164165?ln=en [Accessed: Dec 11, 2010].

[19] T. Bohl. Private communication.

[20] J. C. Molendijk, "Serial link solutions (draft)." Unpublished document.

[21] Maxim, *1-Wire® Communication Through Software*. Available: http://pdfserv.maxim-ic.com/en/an/AN126.pdf [Accessed: Nov 5, 2010].

[22] J. C. Molendijk, *LHC RF Low Level backplane specification*. CERN. Available: https://edms.cern.ch/document/603466 [Accessed: Nov 20, 2010].

[23] VITA, *VME Technology Frequently Asked Questions*. Available: http://www.vita.com/vmefaq.html [Accessed: Dec 1, 2010].

[24] CERN, *EDMS Portal*. Available: https://edms.cern.ch/ [Accessed: Dec 08, 2010].

[25] J. Hoult, "Teachers' Notes on Particle Accelerators." Available: http://teachers.web.cern.ch/teachers/archiv/HST2001/accelerators/teachers%20notes/Teachers%20Notes%20on%20Accelerators.doc [Accessed: Jan 07, 2011].

[26] P. Vulliez and T. Levens, *EDMS Web Navigator: EDA-02153-V1-2 Dual Trigger Unit*. CERN. Available: https://edms.cern.ch/nav/EDA-02153-V1-2 [Accessed: Dec 08, 2010].

[27] J.-M. Combe and D. Valuch, *EDMS Web Navigator: EDA-02035-V1-0 VME Peak Detector*. CERN. Available: https://edms.cern.ch/nav/EDA-02035-V1-0 [Accessed: Dec 08, 2010].

[28] J.-M. Combe, T. Levens, and D. Valuch, *EDMS Web Navigator: EDA-02163-V2-0 Serial Link Router*. CERN. Available: https://edms.cern.ch/nav/EDA-02163-V2-0 [Accessed: Dec 08, 2010].

[29] J.-M. Combe, T. Levens, and D. Valuch, *EDMS Web Navigator: EDA-02163-V2-1 Serial Link Tester*. CERN. Available: https://edms.cern.ch/nav/EDA-02163-V2-1 [Accessed: Dec 08, 2010].

Data Sheets

[DS1] Xilinx, *Spartan-3AN FPGA Family Data Sheet*. Available: http://www.xilinx.com/support/documentation/data_sheets/ds557.pdf [Accessed: Oct 30, 2010].

[DS2] Maxim Integrated Products, *+5V, Low-Power μP Supervisory Circuits with Adjustable Reset/Watchdog*. Available: http://datasheets.maxim-ic.com/en/ds/MAX6301-MAX6304.pdf [Accessed: Oct 30, 2010].

[DS3] Texas Instruments Inc, *TPS75003 Triple-Supply Power Management IC for Powering FPGAs and DSPs*. Available: http://focus.ti.com/lit/ds/symlink/tps75003.pdf [Accessed: Oct 30, 2010].

[DS4] Fairchild Semiconducter Inc, *74LCX16244 Low Voltage 16-Bit Buffer/Line Driver with 5V Tolerant Inputs and Outputs*. Available: http://www.fairchildsemi.com/ds/74%2F74LCX16244.pdf [Accessed: Oct 30, 2010].

[DS5] Fairchild Semiconducter Inc, *CNY17XM, CNY17FXM, MOC810XM – Phototransistor Optocouplers*. Available: http://www.fairchildsemi.com/ds/CN/CNY173M.pdf [Accessed: Oct 30, 2010].

[DS6] Traco Electronic AG, *DC/DC Converters, TES 1 Series, 1 Watt*. Available: http://www.tracopower.com/fileadmin/medien/dokumente/pdf/datasheets/tes1.pdf [Accessed: Oct 30, 2010].

[DS7] Analog Devices Inc, *ADuM5200/ADuM5201/ADuM5202 Dual Channel Isolators with Integrated DC-to-DC Converter*. Available: http://www.analog.com/static/imported-files/data_sheets/ADuM5200_5201_5202.pdf [Accessed: Oct 30, 2010].

[DS8] Fairchild Semiconducter Inc, *2N7000/2N7002/NDS7002A N-Channel Enhancement Mode Field Effect Transistor*. Available: http://www.fairchildsemi.com/ds/2N/2N7000.pdf [Accessed: Oct 30, 2010].

[DS9] Texas Instruments Inc, *SN74LVC1G17 Single Schmitt-Trigger Buffer*. Available: http://focus.ti.com/lit/ds/symlink/sn74lvc1g17.pdf [Accessed: Oct 30, 2010].

[DS10] Avago Technologies, *HCPL-260L/060L/263L/063L High Speed LVTTL Compatible 3.3 Volt Optocouplers*. Available: http://www.avagotech.com/docs/AV02-0616EN [Accessed: Oct 30, 2010].

[DS11] Linear Technology, *LT1223 100MHz Current Feedback Amplifier*. Available: http://cds.linear.com/docs/Datasheet/1223fb.pdf [Accessed: Oct 30, 2010].

[DS12] Texas Instruments Inc, *THS3202D 2-GHz, Low Distortion, Dual Current-Feedback Amplifiers*. Available: http://focus.ti.com/lit/ds/symlink/ths3202.pdf [Accessed: Oct 31, 2010].

[DS13] Kingstate, *KPEG165 Piezo Transducer (External Circuitry Type)*. Available: http://www.kingstate.com.tw/eng/Products/Products/ProductItem/tabid/186/rtab/185/Default.aspx?ItemId=255 [Accessed: Oct 31, 2010].

[DS14] Texas Instruments Inc, *SN74LVC1G125 Single Bus Buffer Gate with 3-State Output*. Available: http://focus.ti.com/lit/ds/symlink/sn74lvc1g125.pdf [Accessed: Oct 30, 2010].

[DS15] Aeroflex Inc, *Model 3200 Series/Model 3200 (E Series) Programmable Attenuators with optional TTL Interface*. Available: http://www.aeroflex.com/ams/weinschel/pdfiles/wmod3200.pdf [Accessed: Dec 11, 2010].

[DS16] Mini-Circuits, *KSWHA-1-20+ Surface Mount High Isolation Switch*. Available: http://minicircuits.com/pdfs/KSWHA-1-20+.pdf [Accessed: Nov 20, 2010].

[DS17] Mini-Circuits, *ZFL-2500VH+ Coaxial Amplifier*. Available: http://minicircuits.com/pdfs/ZFL-2500VH+.pdf [Accessed: Nov 20, 2010].

[DS18] Avago Technologies, *HSMS-282x Surface Mount RF Schottky Barrier Diodes*. Available: http://www.avagotech.com/docs/AV02-1320EN [Accessed: Nov 20, 2010].

[DS19] Analog Devices Inc, *AD9240 Complete 14-Bit, 10 MSPS Monolithic A/D Converter*. Available: http://www.analog.com/static/imported-files/data_sheets/AD9240.pdf [Accessed: Nov 20, 2010].

[DS20] Xilinx, *Spartan-3A FPGA Family: Data Sheet*. Available: http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf [Accessed: Nov 20, 2010].

[DS21] Cypress Semiconductor Corporation, *CY7C1461AV33, CY7C1463AV33, CY7C1465AV33 36 Mbit (1M x 36/2 M x 18/512K x 72) Flow-Through SRAM with NoBL™ Architecture*. Available: http://www.cypress.com/?docID=21327 [Accessed: Dec 11, 2010].

[DS22] Mini-Circuits, *ZPUL-30P Coaxial Pulse Amplifier*. Available: http://minicircuits.com/pdfs/ZPUL-30P.pdf [Accessed: Nov 20, 2010].

[DS23] Fischer Elektronik GmbH, *Standard extruded heatsinks — SK 81*. Available: http://www.fischerelektronik.de/index.php?id=114&L=1&article=CA@fcool@PR@SK81_ [Accessed: Dec 11, 2010].

[DS24] International Rectifier, *IPS7081(R)(S)PbF Intelligent Power High Side Switch*. Available: http://www.irf.com/product-info/datasheets/data/ips7081.pdf [Accessed: Dec 1, 2010].

[DS25] Maxim Integrated Products, *MAX1270/MAX1271 Multirange, +5V, 8-Channel, Serial 12-Bit ADCs*. Available: http://datasheets.maxim-ic.com/en/ds/MAX1270-MAX1271B.pdf [Accessed: Dec 1, 2010].

[DS26] Maxim Integrated Products, *DS18B20 Programmable Resolution 1-Wire Digital Thermomete*. Available: http://pdfserv.maxim-ic.com/en/ds/DS18B20.pdf [Accessed: Nov 5, 2010].

[DS27] Maxim Integrated Products, *DS2401 Silicon Serial Number*. Available: http://pdfserv.maxim-ic.com/en/ds/DS2401.pdf [Accessed: Nov 5, 2010].

Appendices

A Dual Trigger Unit — Specification

- NIM module 1L.
- Two independent front panel controls and displays.
- The serial **Cycle Data** signal transmits 500 ms before injection a binary value of the RTC Stack which corresponds to an **MMI Target**.
- There are 31 RTC Stacks (01 hex to 1F hex).
- Delay: −500 ms to 99 999 ms (output pulse = Cycle Data strobe + 500 − Delay).
- The Cycle Data strobe need eventually to be synchronised with the MS-CK.
- Menu-switchable beep.

Front Panel:

- **OUTPUT 1** and **OUTPUT 2** each:
  - Front Panel Controller: display, parameter button, value button.
  - LEMO 00 connector: output TTL 50 Ω programmable (20 µs) positive pulse (rising edge).
  - LED (red) 150 ms active.
- Eventually: connector to update (with PC) the display firmware (target names).

Rear Panel:

- **SD IN:** Serial Cycle Data input (isolated).
- **SD OUT:** Serial Cycle Data output.
- **MS-CK IN:** Timing input (open collector driven).
- **OUT 1a OC:** Output opto coupler programmable (20 µs) closing (falling edge).
- **OUT 1b OC:** Output opto coupler programmable (20 µs) closing (falling edge).
- **OUT 1c TTL:** Output TTL 50 Ω programmable (20 µs) positive pulse (rising edge).
- **OUT 2a OC:** Output opto coupler programmable (20 µs) closing (falling edge).
- **OUT 2b OC:** Output opto coupler programmable (20 µs) closing (falling edge).
- **OUT 2c TTL:** Output TTL 50 Ω programmable (20 µs) positive pulse (rising edge).

B Dual Trigger Unit — User Guide

B.1 Functional description

The Dual Trigger Unit is a 1L NIM module that contains two identical trigger units. Each unit is capable of triggering independently from any MMI Target received over the RTC stack serial link and, for backwards compatibility, has two 'start' timing inputs on the rear panel which can also be used as a trigger source.

On the front panel, each trigger unit has a VHI user interface that is used to configure each parameter of the DTU. A list of the parameters and their functions is included in Section B.4.

In the centre of the front panel are a LVTTL 50 Ω output and a LED for each trigger. The LED will illuminate green while a trigger delay is in progress and flash red for 150 ms when the delay is complete and a pulse is generated. In the VHI, a audible beep can be enabled and this occurs at the same time as the red LED flash.

Each unit has both LVTTL 50 Ω and optocoupled outputs on the back panel. The outputs associated with 'Trigger 1' are marked with blue tabs and those associated with 'Trigger 2' are marked with red tabs to match the front panel. The LVTTL outputs can be identified by the LEMO connectors at the top of the panel that are not fitted with isolation washers. The optocoupled outputs are below these and are fitted with black isolation washers.

The unit is capable of operating with or without an external 1 kHz clock signal. The 'MS-CK Int/Ext' parameter in the VHI interface the clock source to be selected. When the 'Sync to F_{REV}' parameter is enabled the unit triggers and counts it's delay as normal and, once elapsed, the unit will wait until the next F_{REV} pulse is received before outputting a pulse.

B.2 Problems

If the front panel LED illuminates blue for 5 s after either the F_{REV} synchronisation is enabled or the external MS-CK is selected then the relevant clock signal is not connected or being received properly.

When the external MS-CK is selected and the unit triggers, the LED will illuminate blue immediately if the MS-CK is not received. This can be verified by checking current delay time with the VHI parameter 'RDBK: Trig. Time'. If there is a problem with the MS-CK this will not count upwards. In this case, check that the MS-CK is connected or switch to the internally generated clock.

If the delay time elapses as normal, but the LED illuminates blue rather than flashing red, the F_{REV} signal is not being received. In this case, check the connections or disable F_{REV} synchronisation.

B.3 Inputs and outputs

B.3.1 Front panel

LVTTL 50 Ω output

One LVTTL 50 Ω output per trigger unit with a LEMO connector. Output pulse has a positive polarity. Connectors are marked with a red or blue tab to indicate which trigger they belong to.

B.3.2 Rear panel

LVTTL 50 Ω output

One LVTTL 50 Ω output per trigger unit with a LEMO connector. Output pulse has a positive polarity. Connectors are marked with a red or blue tab to indicate which trigger they belong to.

Optocoupled output

Two open-collector optocoupled outputs per trigger unit with LEMO connectors. Connectors have black isolation washers. Output pulse has a negative polarity. Connectors are marked with a red or blue tab to indicate which trigger they belong to.

SL IN and SL OUT

16 bit SerialLink[16] connection for MMI target reception via RTC stack serial link. SMC connectors as per usual. Input has red isolation washer.

F_{REV} input

Input for 2 V, 5 ns F_{REV} pulse. SMC connector.

MS-CK input

SPS-type timing input for the 1 kHz clock signal.

START 1 and START 2

SPS-type timing input for external triggers.

B.4 VHI parameters

Trigger Source

This parameter allows the trigger source to be selected. This is a list parameter which contains the 32 MMI targets and the two external start inputs, 'Start 1' and 'Start 2'. A full list is provided in Section B.6.

Delay Time

This parameter sets the time delay between receiving a trigger and outputting a pulse. The value correlates to the delay in milliseconds and has a range of -500 ms to 100 s.

RDBK: Cur. Cycle

This parameter gives a read back of the MMI target that's currently playing in the machine.

RDBK: Trig. Time

When the unit is triggered, this read back parameter shows the progress of the delay in milliseconds.

Pulse Width

This parameter sets the width of the output pulse in microseconds. It is configurable with a range of 1 µs to 100 µs.

Sync to F_{REV}

When this boolean parameter is enabled, the output pulse will be delayed until the next F_{REV} pulse is received at the F_{REV} input.

Beep

This parameter configures the audible feedback beep. When set to zero the beep is disabled. When set to a value between one and four the beep is enabled. A different frequency beep is produced for each value:

0. Beep Off
1. 4.8kHz
2. 3.5kHz
3. 2.3kHz
4. 1.4kHz

MS-CK Int/Ext

This boolean parameter selects whether the internally generated 1 kHz clock should be used, or whether the external 1 kHz clock signal, connected to the MS-CK input should be used.

Supply	Idle Current
6 V	95 mA
12 V	110 mA
−12 V	10 mA
24 V	5 mA

Table 3: DTU power supply requirements.

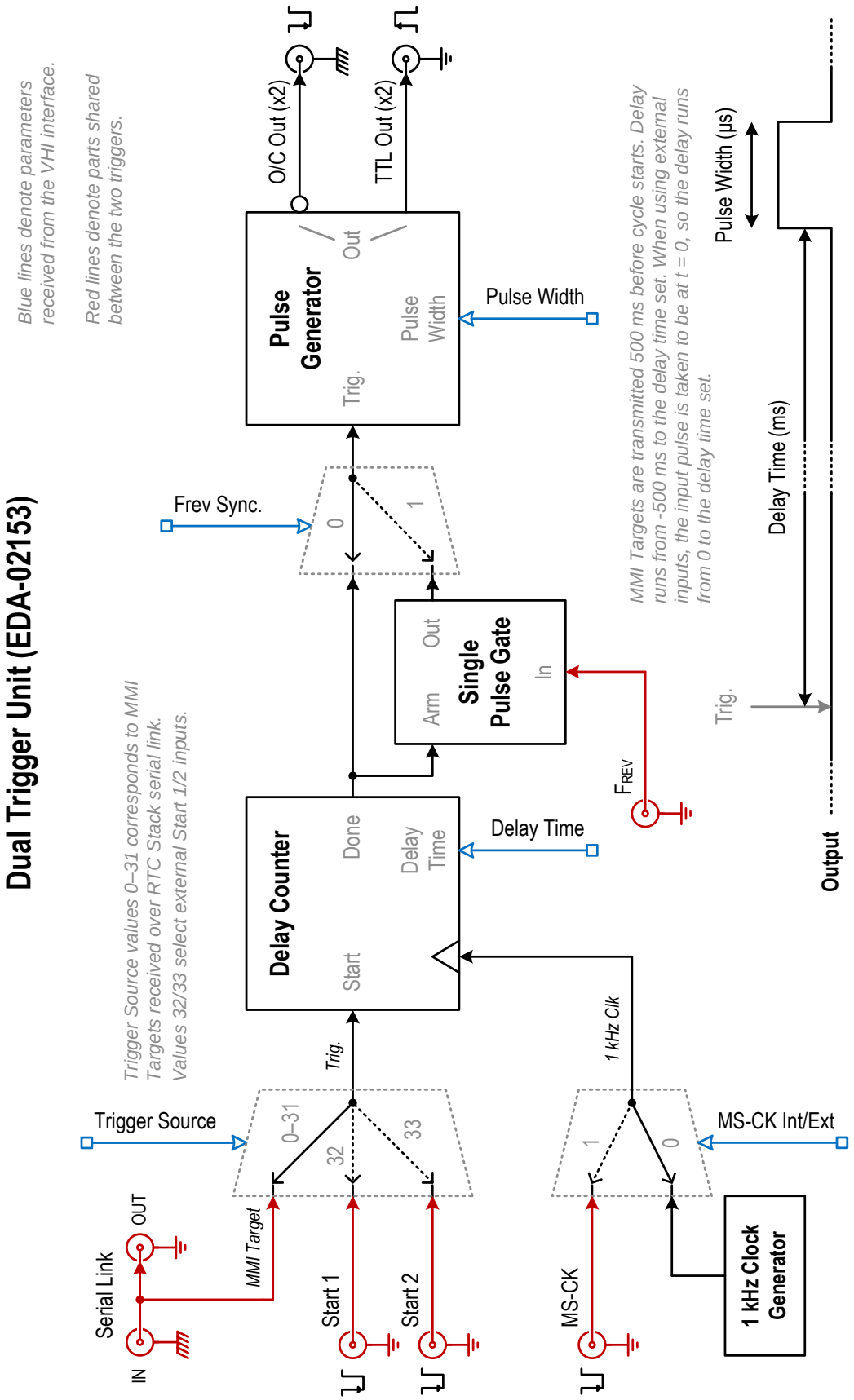
B.5 Power supply requirements

Table 3 lists the power supplies required in the NIM crate for complete operation. Nominal current when the unit is idle is also indicated. Each LVTTL output loaded with 50 Ω will increase the current draw on the 6 V line by approximately 50 mA for the duration of the output pulse. Each piezo sounder, when enabled, will draw approximately 20 mA (average) for the duration of the 150 ms beep.

B.6 Current list of trigger sources

0. Zero
1. SFTPRO1
2. SFTPRO2
3. SFT25 ns
4. MD1
5. MD2
6. IonsFixTrg
7. IonsRecapt
8. Res Stk 08
9. Res Stk 09
10. LHCPILOT
11. LHCMONO
12. LHC12BU
13. LHC25 ns
14. LHC75 ns
15. LHCMD
16. LHCSCRUB
17. LHCION
18. Res Stk 18
19. Res Stk 19
20. CNGS1
21. CNGS2
22. CNGS3
23. Res Stk 23
24. Res Stk 24
25. Res Stk 25
26. Res Stk 26
27. Res Stk 27
28. Res Stk 28
29. Res Stk 29
30. Res Stk 30
31. Res Stk 31
32. Start 1
33. Start 2

B.7 Block diagram

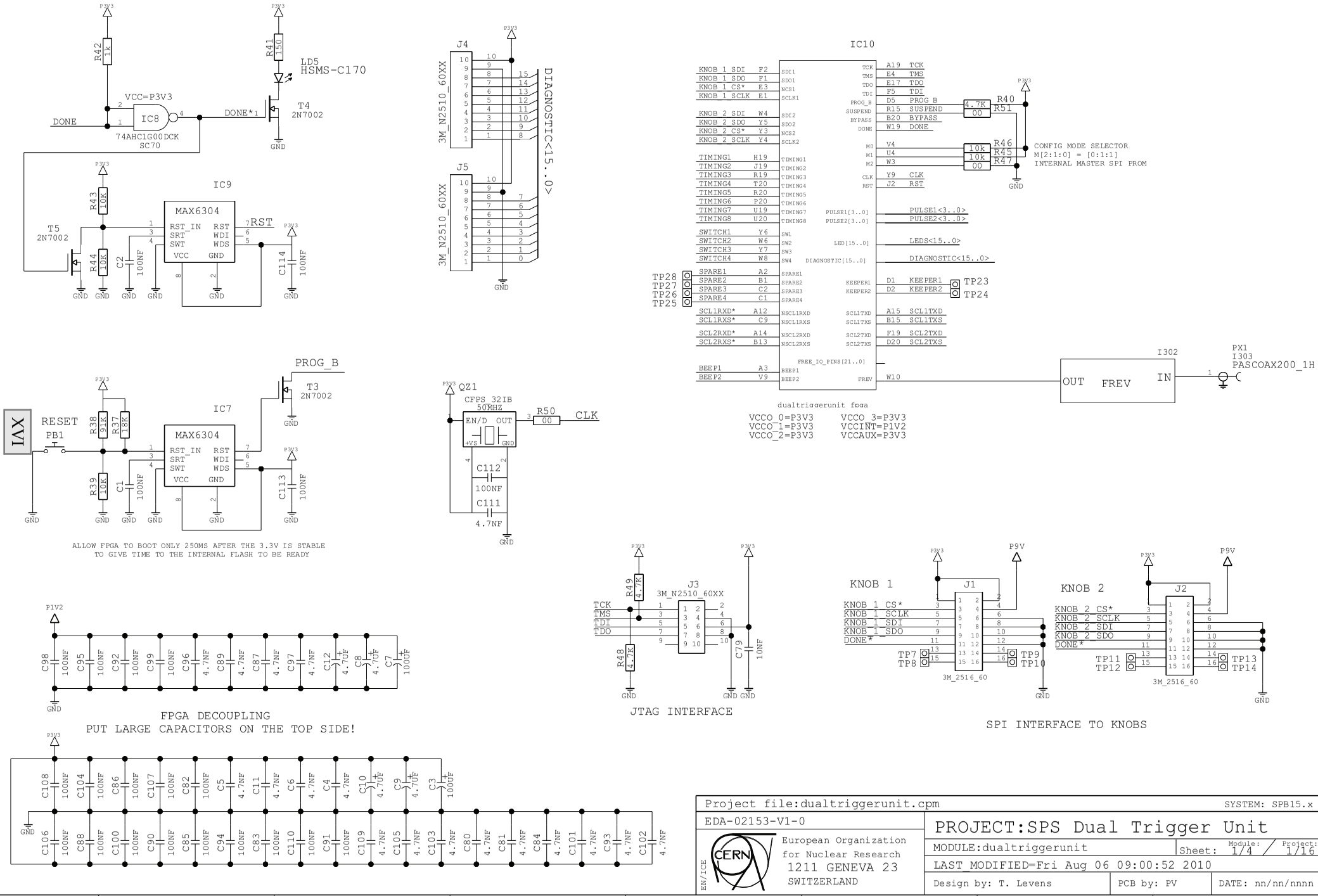


C Dual Trigger Unit — Design Files

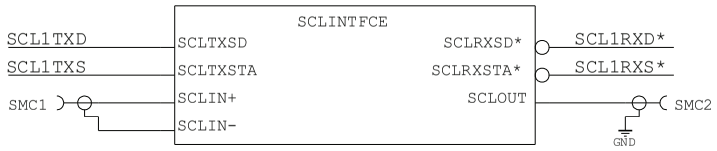
This appendix contains the schematic, PCB and mechanical parts for the Dual Trigger Unit module. The following documents are included:

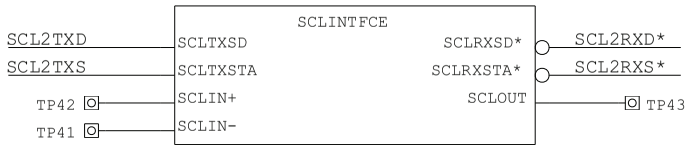
The PCB and panels were prepared by Mr. Pascal Vulliez in the CERN TE-DEM department. Duplicate pages of the schematic are omitted for clarity. For the PCB: green lines indicate tracks on the top layer and red indicates tracks on the bottom.

All files related the the DTU are published on CERN's EDMS [24] service with the identifier EDA-02153-V1-2 [26].

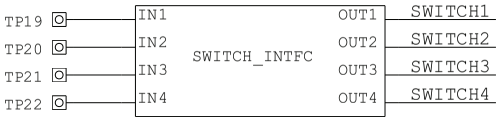


Project file:dualtriggerunit.cpm		SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:dualtriggerunit	Sheet: 1/4 Module: Project: 1/16
EN/ICE	LAST_MODIFIED=Fri Aug 06 09:00:52 2010	
	Design by: T. Levens	PCB by: PV DATE: nn/nn/nnnn

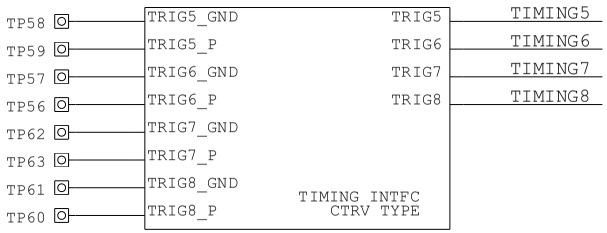


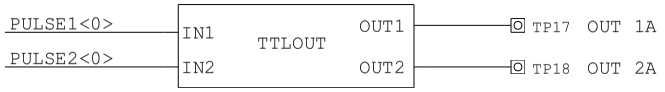


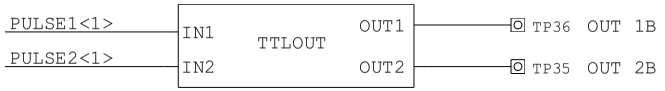
CONNECTORS FOR SCL2, DO NOT ASSEMBLE
PREPARE COAX PINS TO SOLDER CABLE



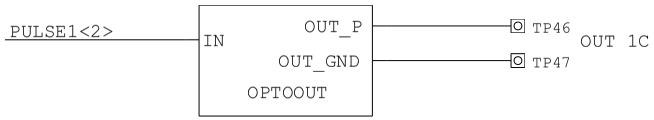
SWITCH INTERFACE

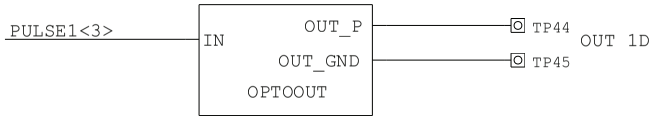


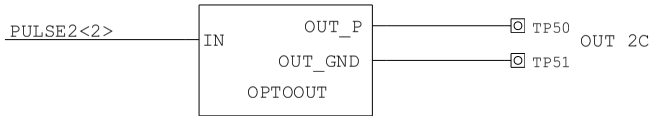


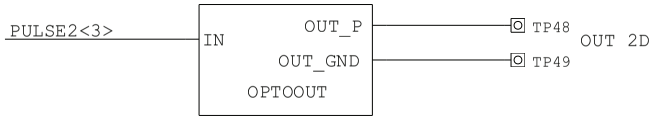


TTL OUTPUTS 1/2
OUT A FRONT PANEL ISOLATED LEMO
OUT B BACK PANEL ISOLATED LEMO
SOLDERED ON CABLES





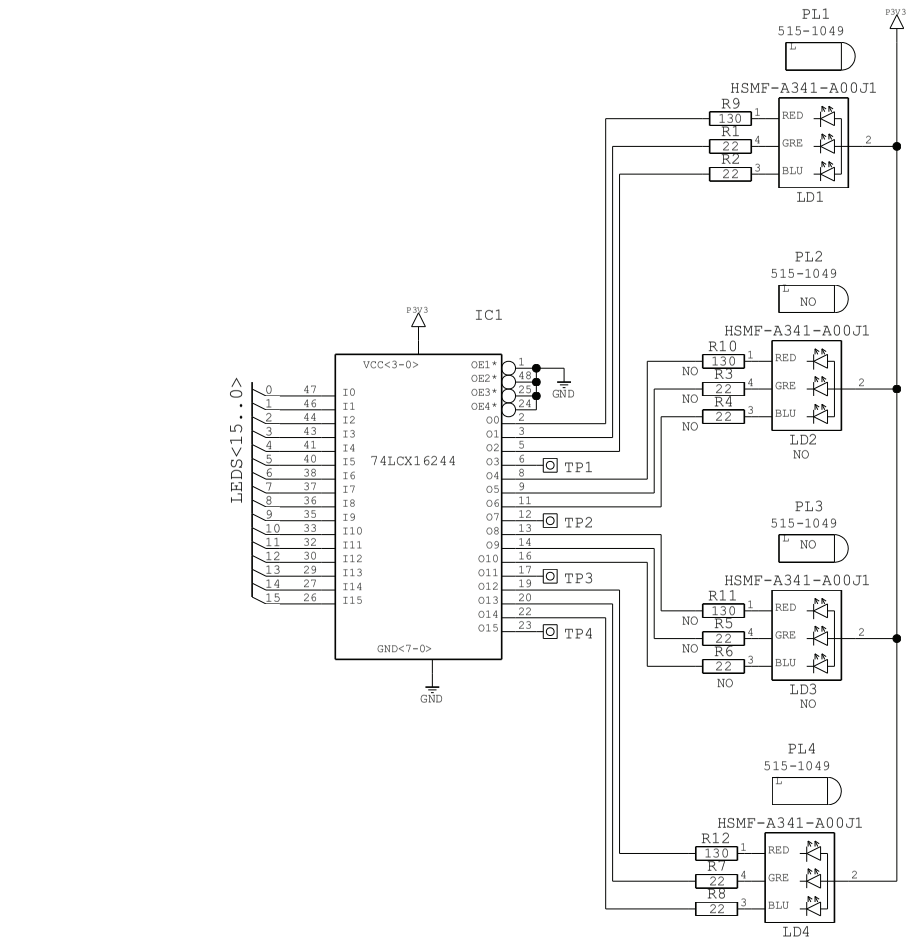


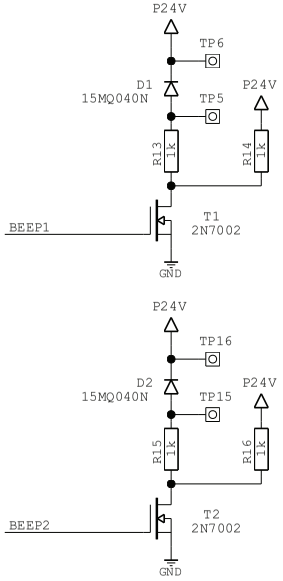


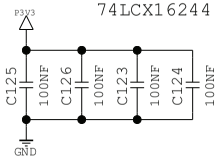
ISOLATED OUTPUTS 1/2
OUT C/D BACK PANEL ISOLATED LEMO
SOLDERED ON CABLES

XVII

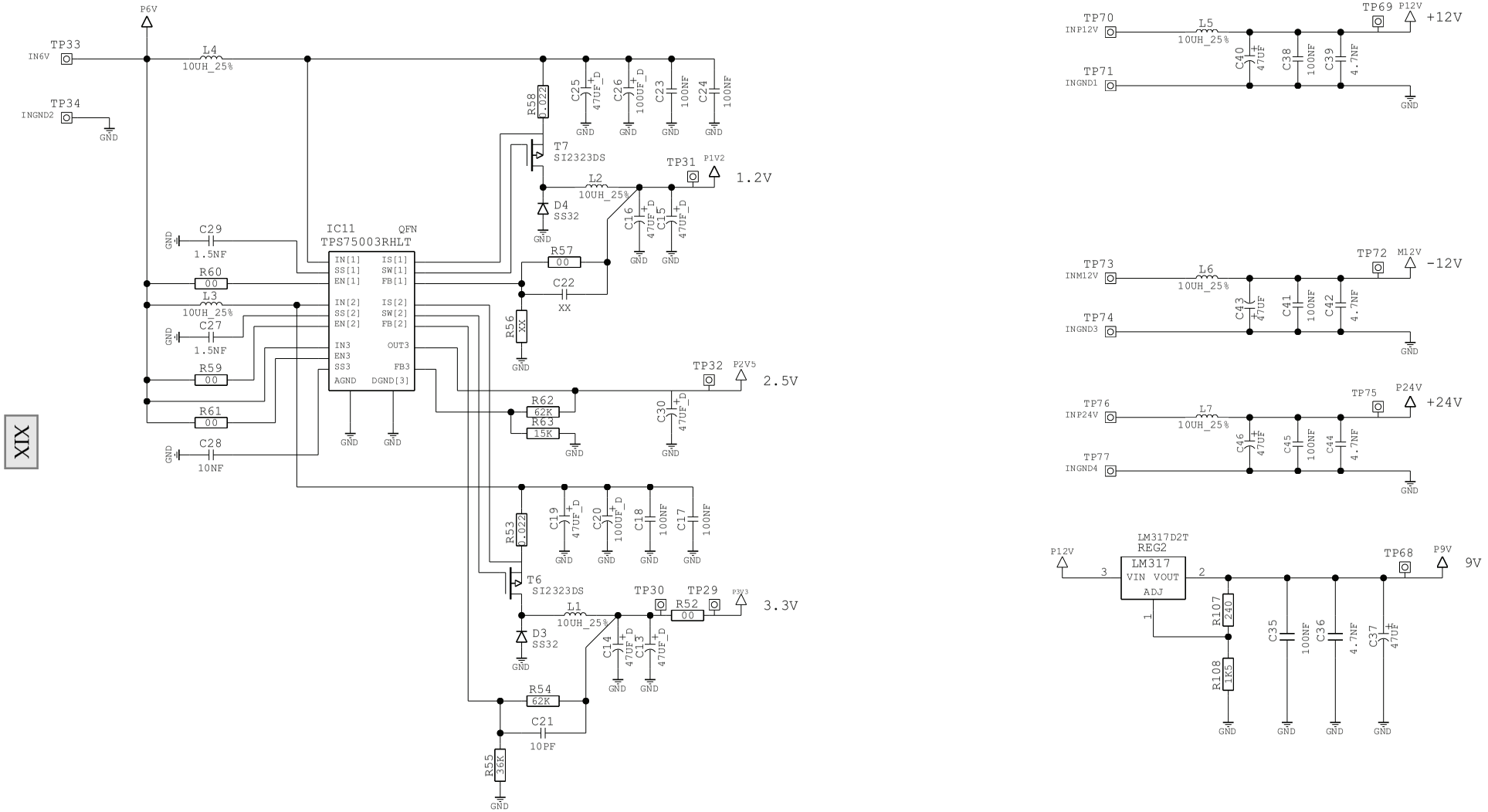
Project file:dualtriggerunit.cpm			SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit		
EN/ICE CERN European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:dualtriggerunit	Sheet: 2/4	Module: 2/4 / Project: 2/16
	LAST_MODIFIED=Fri Aug 06 09:00:51 2010		
	Design by: T. Levens	PCB by: PV	DATE: nn/nn/nnnn





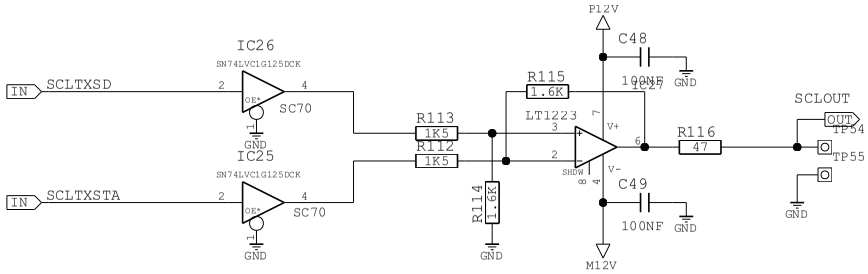


Project file:dualtriggerunit.cpm		SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:dualtriggerunit	Sheet: 3/4 Module: / Project: 3/16
	LAST_MODIFIED=Fri Aug 06 09:00:52 2010	
	Design by: T. Levens	PCB by: PV DATE: nn/nn/nnnn



Project file:dualtriggerunit.cpm			SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit		
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:dualtriggerunit	Sheet: 4/4	Module: 4/4 / Project: 4/16
	LAST_MODIFIED=Fri Aug 06 09:00:53 2010		
	Design by: T. Levens	PCB by: PV	DATE: nn/nn/nnnn

SERIAL CONTROL LINK TRANSMITTER
3.3V VERSION



SERIAL LINK OUT:

BACK PANEL SMC

PCB CABLE TERMINAL

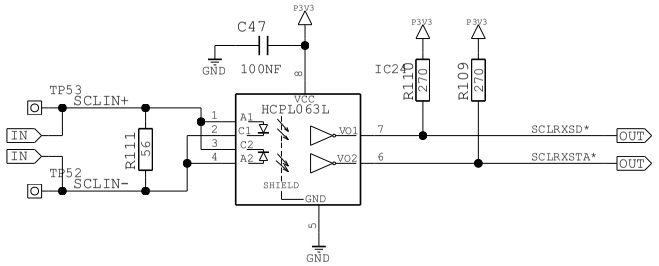
XX

SERIAL CONTROL LINK RECEIVER

SERIAL LINK IN:

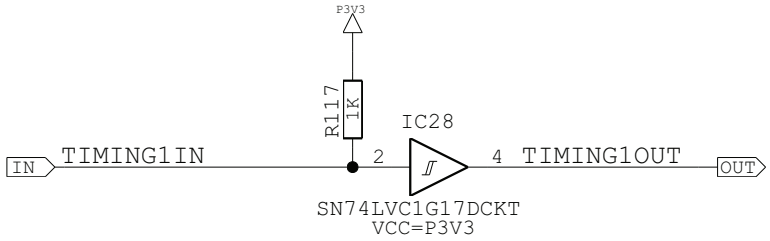
BACK PANEL SMC

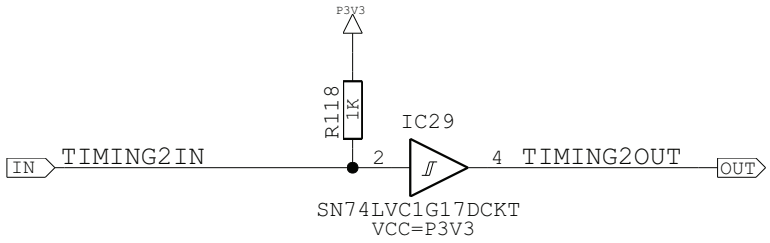
PCB CABLE TERMINAL

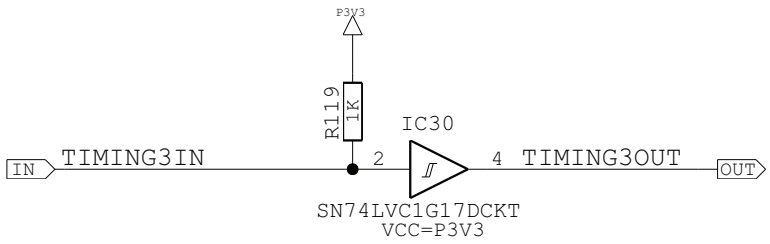


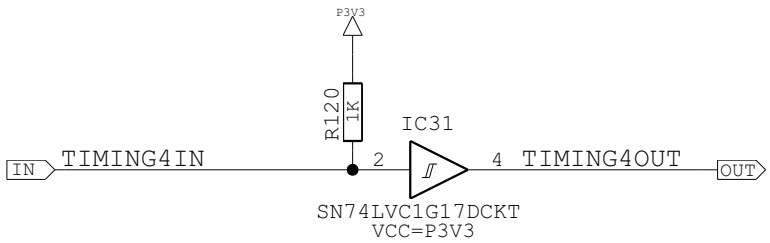
1G125 DECOUPLING

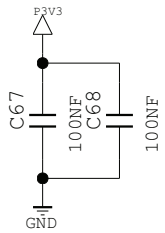
Project file:dualtriggerunit.cpm		SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:sclintfce	Sheet: 1/1 Module: Project: 5/16
EN/ICE	LAST_MODIFIED=Fri Aug 06 09:00:54 2010	
	Design by: T. Levens / PCB by: PV	DATE: nn/nn/nnnn











Project file:dualtriggerunit.cpm SYSTEM: SPB15.x

EDA-02153-V1-0

European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND
TS/DEM

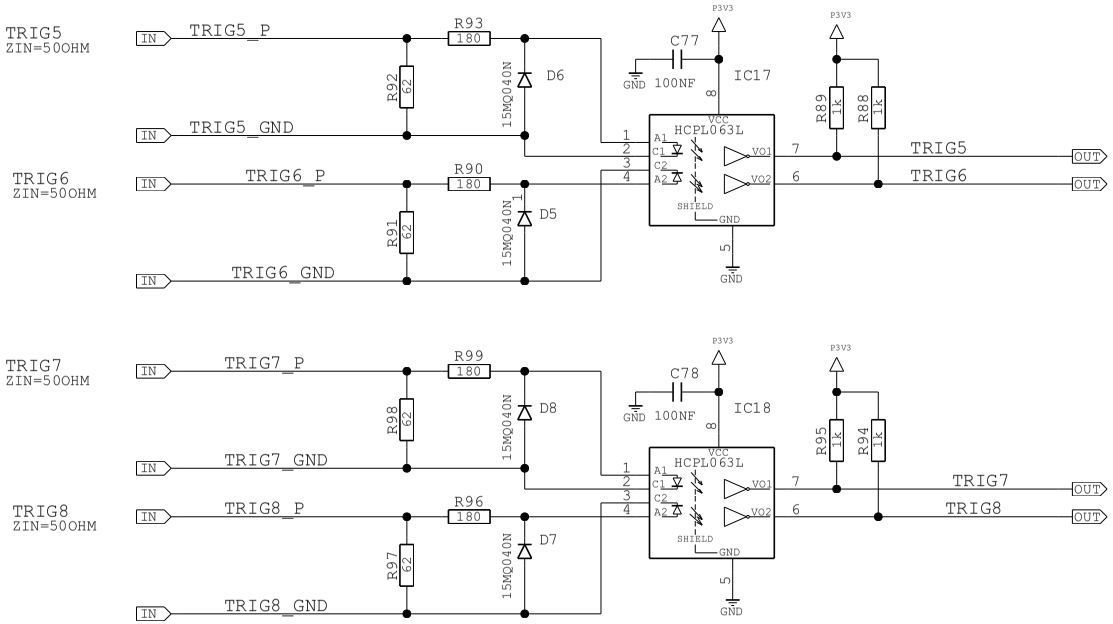
PROJECT:SPS Dual Trigger Unit

MODULE:timing_rcv | Sheet: Module: 1/1 / Project: 7/16

LAST_MODIFIED=Fri Aug 06 09:00:55 2010

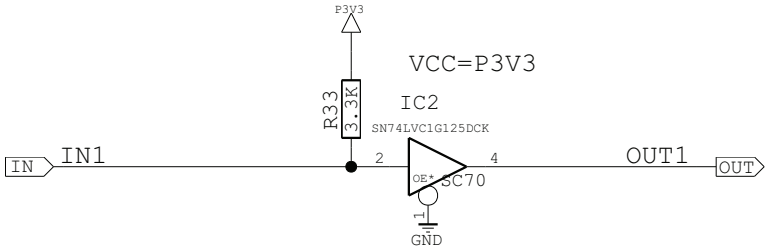
Design by: [illegible]vens | PCB by: PV | DATE: nn/nn/nnnn

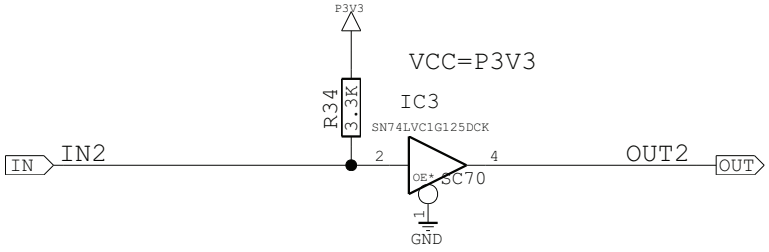
OPTO-ISOLATION FOR CTRV TRIGGERS

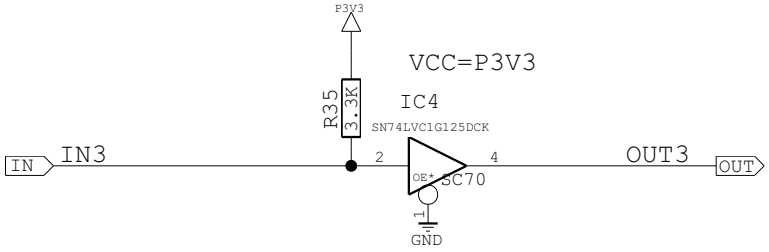


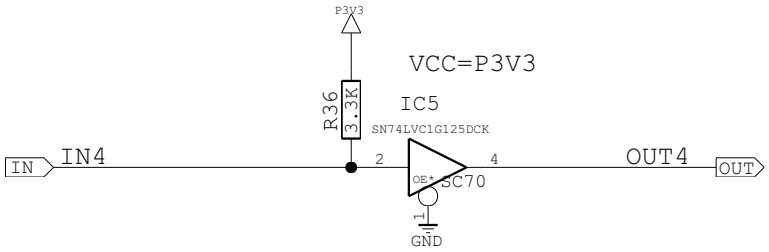
XXII

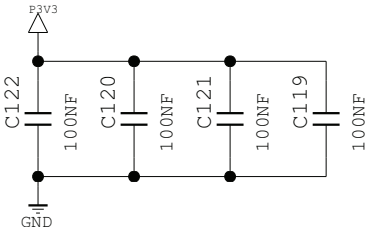
Project file:dualtriggerunit.cpm		SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND (EN/ICE)	MODULE:fpintfce	Sheet: Module: 1/1 Project: 8/16
	LAST_MODIFIED=Fri Aug 06 09:00:56 2010	
	Design by: T. Levens / PCB by: PV	DATE: nn/nn/nnnn











Project file:dualtriggerunit.cpm SYSTEM: SPB15.x

EDA-02153-V1-0

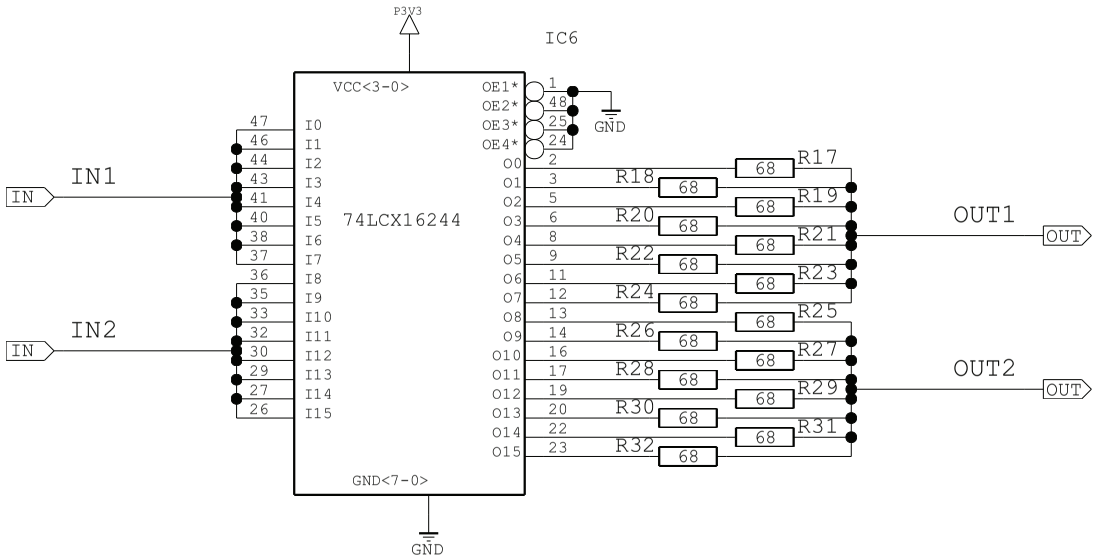
European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND
TS/DEM

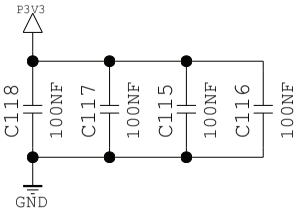
PROJECT:SPS Dual Trigger Unit

MODULE:switch_intfc Sheet: Module: 1/1 Project: 9/16

LAST_MODIFIED=Fri Aug 06 09:00:57 2010

Design by: T. Levens PCB by: PV DATE: nn/nn/nnnn





XXIV

Project file:dualtriggerunit.cpm SYSTEM: SPB15.x

EDA-02153-V1-0

TS/DEM

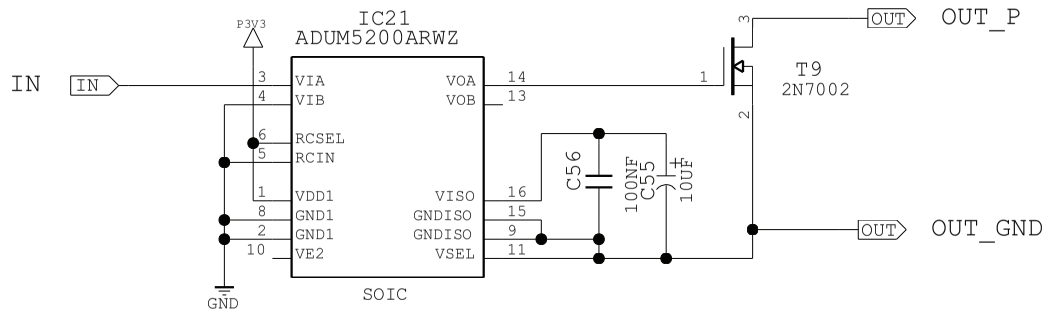
European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND

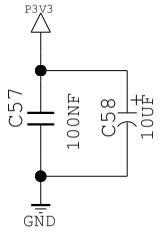
PROJECT:SPS Dual Trigger Unit

MODULE:ttlout Sheet: Module: 1/1 / Project: 10/16

LAST_MODIFIED=Fri Aug 06 09:00:58 2010

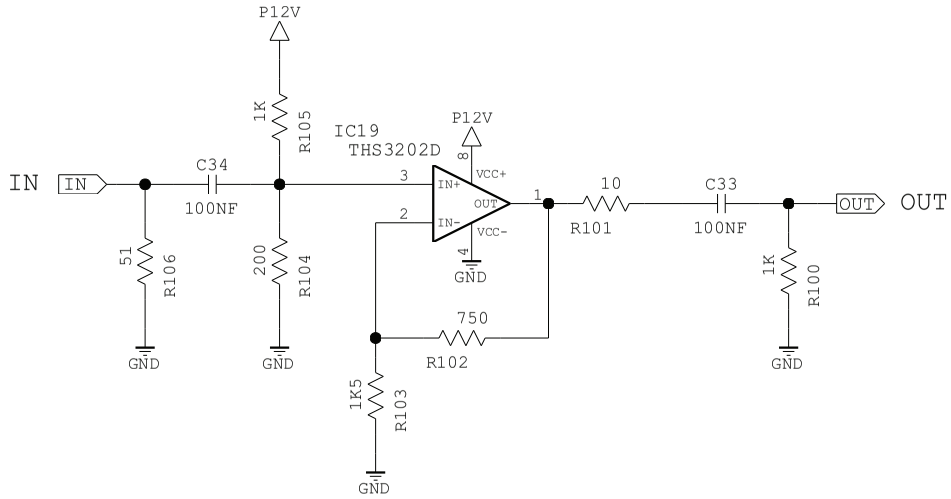
Design by: T. Levens | PCB by: PV | DATE: nn/nn/nnnn

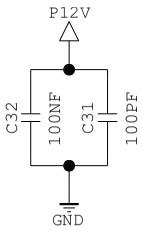




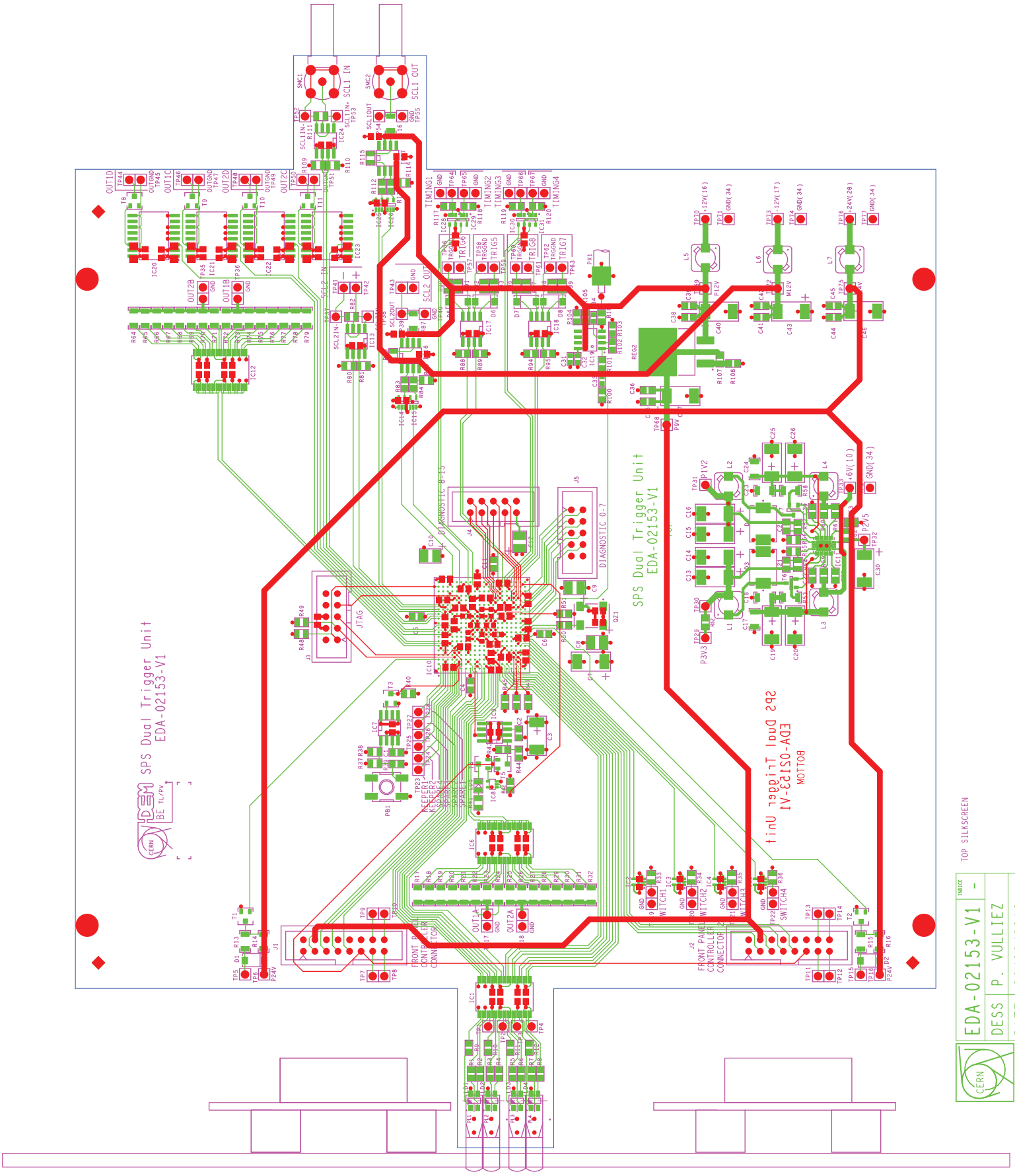
Project file:dualtriggerunit.cpm		SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit	
TS/DEM — European Organization for Nuclear Research, 1211 GENEVA 23, SWITZERLAND	MODULE:optoout	Sheet: Module: 1/1 / Project: 12/16
	LAST_MODIFIED=Fri Aug 06 09:00:59 2010	
	Design by: T. Levens	PCB by: PV — DATE: nn/nn/nnnn

FREV AMPLIFIER
2V TO 3V3





Project file:dualtriggerunit.cpm		SYSTEM: SPB15.x
EDA-02153-V1-0	PROJECT:SPS Dual Trigger Unit	
TS/DEM CERN European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:frev	Sheet: Module: 1/1 / Project: 16/16
	LAST_MODIFIED=Fri Aug 06 09:00:59 2010	
	Design by: T. Levens	PCB by: PV / DATE: nn/nn/nnnn


SPS Dual Trigger Unit
EDA-02153-V1
TOP SILKSCREEN
EDA-02153-V1
DESS P. VULLIEZ
DATE 06/08/2010
EN/ICE

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES
SELON NORMES ISO
DRAWING, RUGOSITY, TOLERANCES
ACCORDING TO ISO STANDARDS

PROJECTION

EDA-02153 DUAL TRIGGER UNIT

FRONT PANEL
ARRANGEMENT

SCALE 1:1

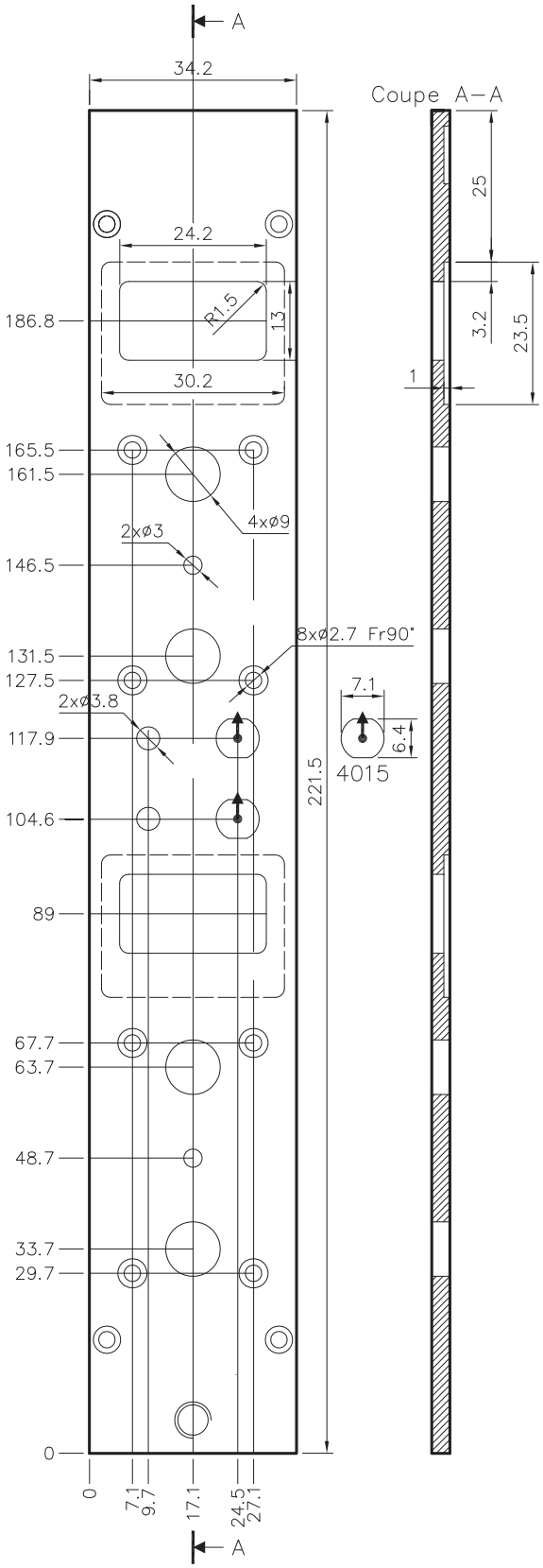
DES	P Vulliez	01/10/2010
MOD.-		
N° EDA-02153-V1-2_fp.dwg		

EDA-02153-V1-2_fp-pres 4

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON NORMES ISO
DRAWING, RUGOSITY, TOLERANCES ACCORDING TO ISO STANDARDS

PROJECTION



PANNEAU FRONTAL 5Hx1L
(CHASSIS CIM 8905)
EPAISSEUR 3mm
SCEM: 06.61.53.551.1

Traitement de Surface : Oxydation anodique (Anodisation)

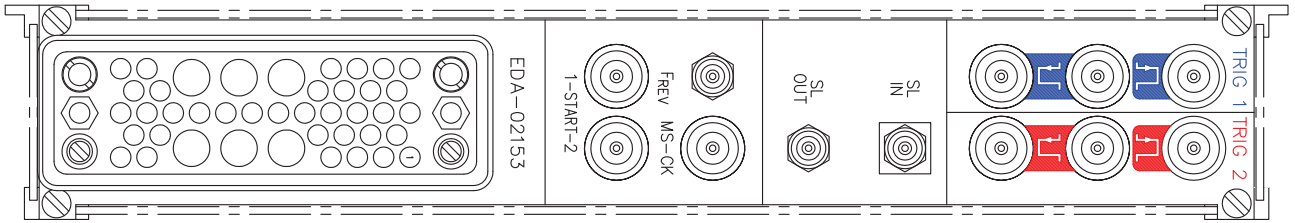
EDA-02153 DUAL TRIGGER UNIT			
FRONT PANEL MECHANICAL	SCALE 1:1	DES P Vulliez	01/10/2010
		MOD.-	
		N° EDA-02153-V1-2_fp.dwg	
EDA-02153-V1-2_fp-mech			4

PROJECTION

DESSIN, RUGOSITE, TOLERANCES
SELON NORMES ISO
DRAWING, RUGOSITY, TOLERANCES
ACCORDING TO ISO STANDARDS

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

XXX



EDA-02153 DUAL TRIGGER UNIT

REAR PANEL ARRANGEMENT

SCALE 1:1

DES P Vuillez 01/10/2010

MOD.

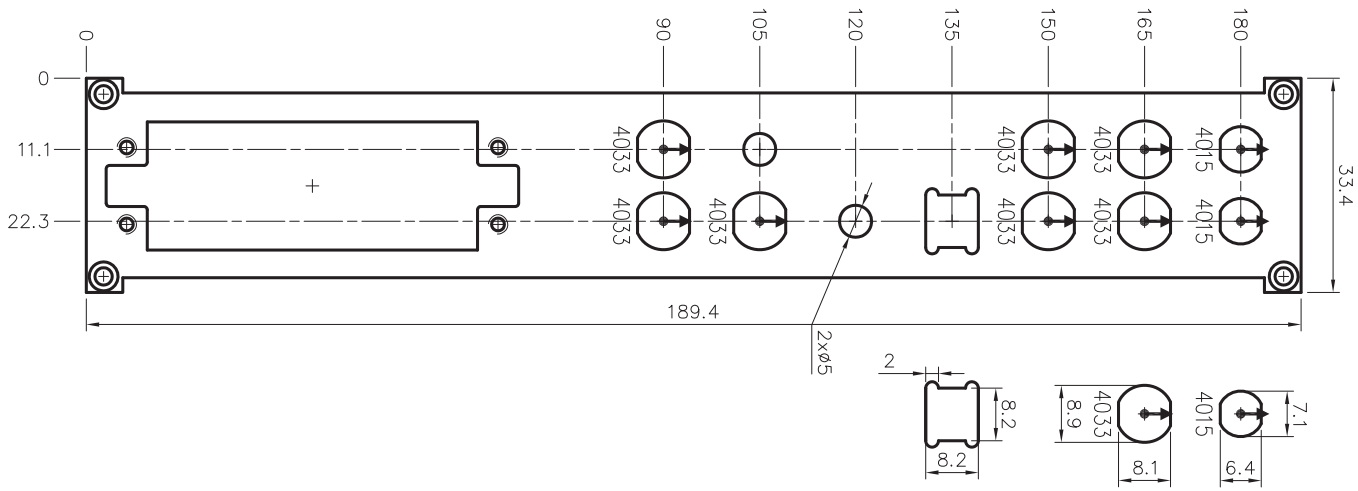
N° EDA-02053-V1-2_rp.dwg

EDA-02153-V1-2_rp-pres 4

PROJECTION

DESSIN, RUGOSITE, TOLERANCES
SELON NORMES ISO
DRAWING, RUGOSITY, TOLERANCES
ACCORDING TO ISO STANDARDS

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10



EDA-02153 DUAL TRIGGER UNIT

REAR PANEL
MECHANICAL

SCALE 1:1

DES P Vuilliez 01/10/2010

MOD.

N° EDA-02153-V1-2_rp.dwg

EDA-02153-V1-2_rp-pres | 4

D Dual Trigger Unit — Firmware

This appendix contains the firmware for the Dual Trigger Unit module. Only blocks that are unique to this project are included, and are as follows:

DualTriggerUnit_Top XXXIV

Top level of the firmware.

BeepDecoder XXXVI

Mapping from beep selection register to frequency.

DualTriggerUnit_Pack XXXVII

Register address definitions for the VHI.

DlyCounter XXXVIII

VHDL block which implements a more advanced delay counter.

FrevCapture XL

Block to capture and stretch the fast F_{REV} pulse.

KnobRegisters XLI

Registers for VHI communication.

SingleTriggerUnit XLII

Complete, single, trigger unit which implements all functionality shared between units.

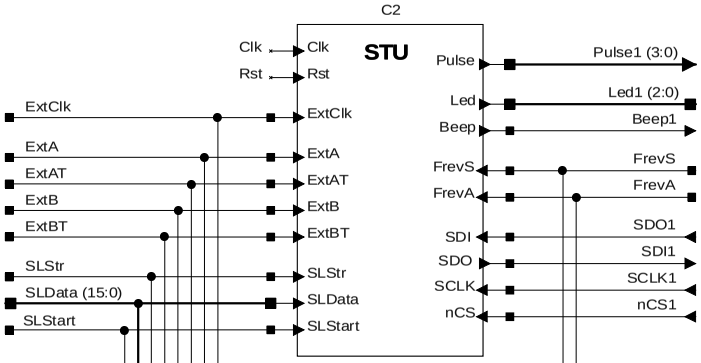
TimClkChk XLVI

Provides a pulse for a defined period of time if a clock signal is not available.

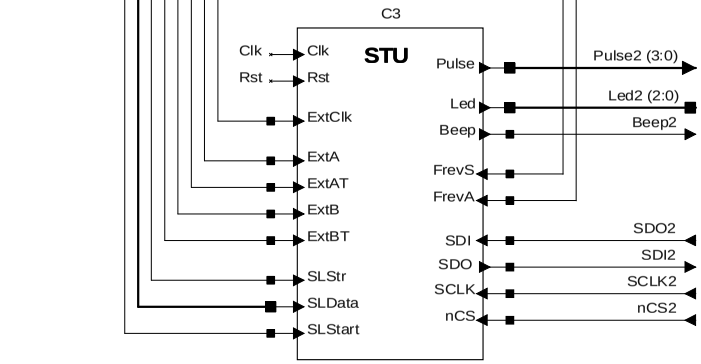
This firmware for this module also relies upon the VHI SPI Controller (Appendix Q), the SerialLink[16] Controller (Appendix O) and blocks common to all modules (Appendix N). Any blocks not included from any of these sources are recycled from other projects and are not the work of the author.

Dual Trigger Unit

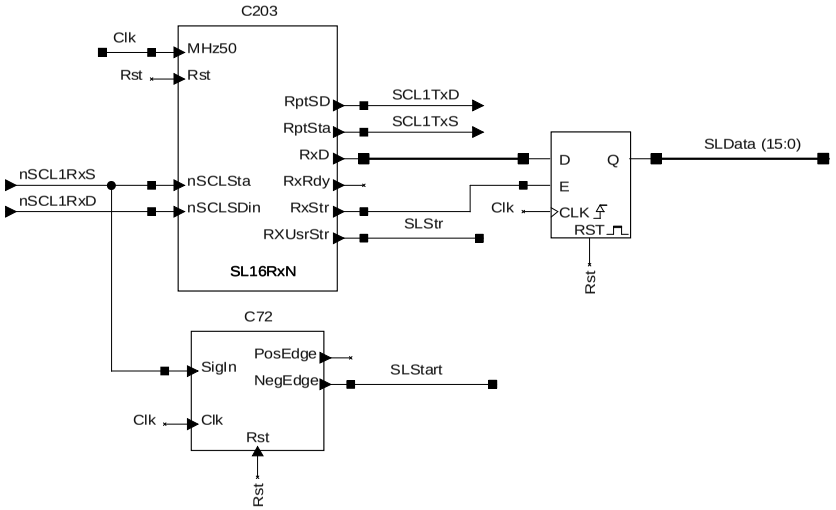
TRIGGER 1



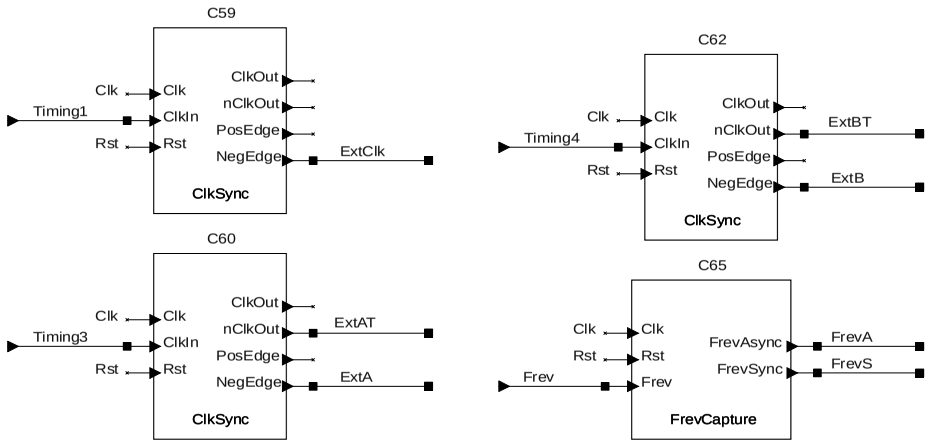
TRIGGER 2



SERIAL INPUT

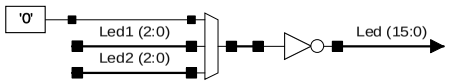


TIMING INPUTS



LED DRIVERS

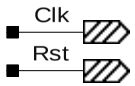
(ACTIVE LOW)



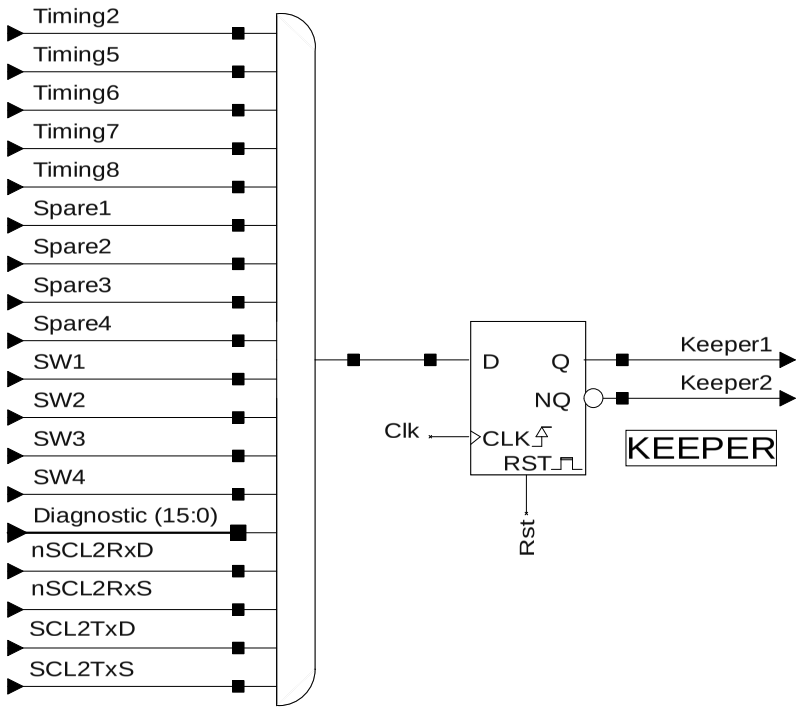
XXXIV

Design: DualTriggerUnit:DualTriggerUnit_Top	Date: Thu Dec 30 14:07:27 2010	Page: 1

KEEPER FOR UNUSED I/O



XXXV



Design: DualTriggerUnit:DualTriggerUnit_Top[2]	Date: Sun Dec 05 13:57:02 2010	Page: 1 (1,1)

Truth Table: BeepDecoder

Beep	EnableOut	Divider
X”1”	EnableIn	Beep1Div
X”2”	EnableIn	Beep2Div
X”3”	EnableIn	Beep3Div
X”4”	EnableIn	Beep4Div
	'0'	X”0000”

Packages Used
ieee.STD_LOGIC_1164
work.DualTriggerUnit_Pack

Interface
EnableIn : in std_logic ;
Beep : in std_logic_vector (3 downto 0);
EnableOut : out std_logic ;
Divider : out std_logic_vector (15 downto 0);

D.3 DualTriggerUnit_Pack

```
-----------------------------------------
-----------------------------------------
-- Date        : Mon Jun 07 10:25:14 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : KNOB memory map and other constants for the Dual
--               Trigger Unit.
--
-----------------------------------------
-----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

package  DualTriggerUnit_Pack  is
  -- Firmware version code.
  constant    C_FirmwareVersion   : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(20101214, 32));

  -- KNOB memory-map version code.
  constant    C_MemoryMapVersion  : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(20100806, 32));

  -- KNOB write registers.
  constant    TargetSelectReg     : std_logic_vector(6 downto 0)  := "0010000";   -- 10h
  constant    DelayTimeReg        : std_logic_vector(6 downto 0)  := "0010001";   -- 11h
  constant    StartOffsetReg      : std_logic_vector(6 downto 0)  := "0010010";   -- 12h
  constant    PulseLengthReg      : std_logic_vector(6 downto 0)  := "0010011";   -- 13h
  constant    FrevSyncReg         : std_logic_vector(6 downto 0)  := "0010100";   -- 14h
  constant    BeepReg             : std_logic_vector(6 downto 0)  := "0010101";   -- 15h
  constant    ClkSelectReg        : std_logic_vector(6 downto 0)  := "0010110";   -- 16h

  -- KNOB read-back registers.
  constant    FirmwareVersionReg  : std_logic_vector(6 downto 0)  := "0000000";   -- 00h
  constant    MemoryMapVersionReg : std_logic_vector(6 downto 0)  := "0000001";   -- 01h
  constant    CurrentTargetReg    : std_logic_vector(6 downto 0)  := "0010111";   -- 17h
  constant    TargetTimeReg       : std_logic_vector(6 downto 0)  := "0011000";   -- 18h

  -- 50MHz clock dividers for different beep "tones".
  constant    Beep1Div            : std_logic_vector(15 downto 0) := X"28B1";     -- 4.8kHz
  constant    Beep2Div            : std_logic_vector(15 downto 0) := X"37CE";     -- 3.5kHz
  constant    Beep3Div            : std_logic_vector(15 downto 0) := X"54EB";     -- 2.3kHz
  constant    Beep4Div            : std_logic_vector(15 downto 0) := X"8B82";     -- 1.4kHz

  -- Clock checks. Added 5% to the frequency as a margin for error.
  constant    FrevChkMin          : std_logic_vector(31 downto 0) := X"0000043A"; -- 44kHz
    (0470h) + 5% = 46.2kHz
  constant    FrevChkMax          : std_logic_vector(31 downto 0) := X"000004E5"; -- 42kHz (04
    A6h) - 5% = 39.9kHz
  constant    MsClkChkMin         : std_logic_vector(15 downto 0) := X"BA30";     -- 1kHz  (
    C350h) + 5% = 1.05kHz
  constant    MsClkChkMax         : std_logic_vector(15 downto 0) := X"CD97";     -- 1kHz  (
    C350h) - 5% = 0.95kHz

  -- Other constants.
  constant    MsClkDiv            : std_logic_vector(15 downto 0) := X"C350";     -- 1kHz
  constant    LedFlashDur         : std_logic_vector(7 downto 0)  := X"96";       -- 150ms
  constant    ClkChkDur           : std_logic_vector(15 downto 0) := X"1388";     -- 5s

end;
```

D.4 DlyCounter

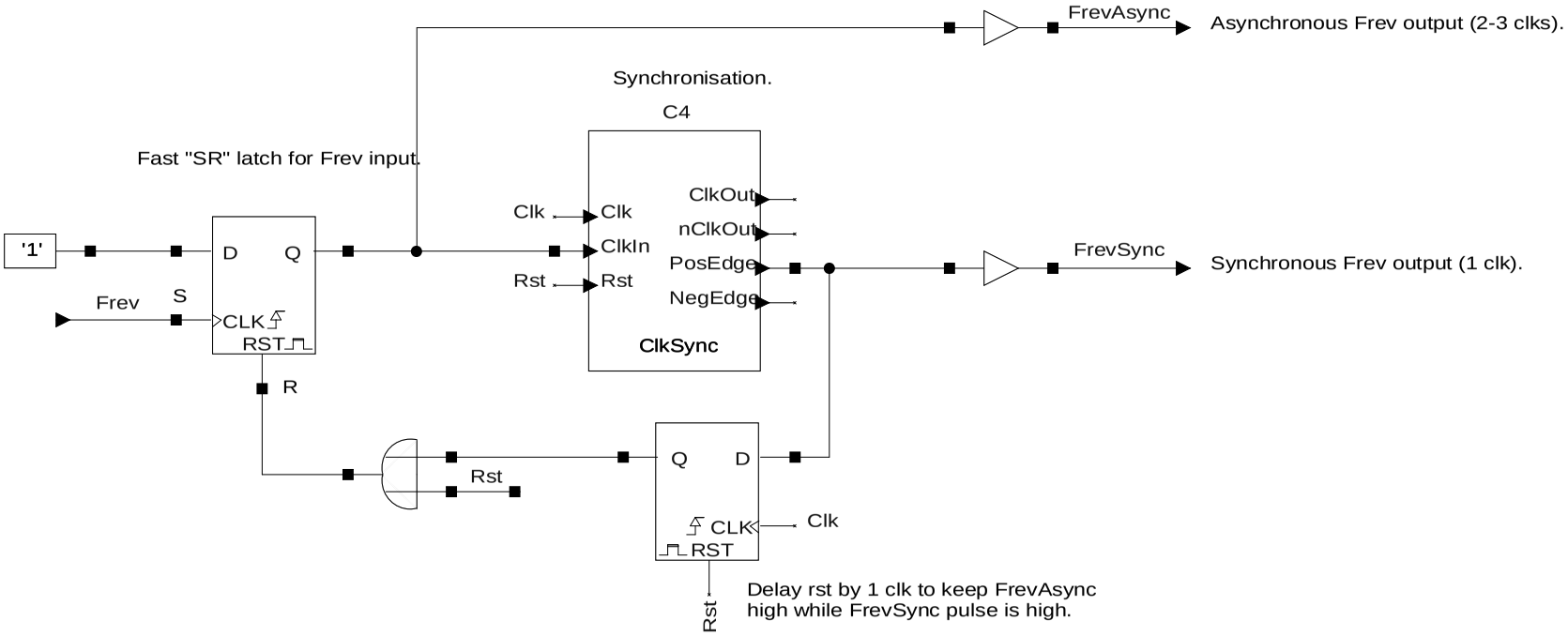
```
----------------------------------------
----------------------------------------
-- Date        : Mon Jun 07 14:39:10 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Signed upward counter with maximum value and preset.
--
--               Bit length of counter can be set with generic "N".
--
--               Enable pin can be left floating (default to '1').
--
-- Changelog   : 20101115 (1.2)
--                 Fixed bug where count output was not updated in the case of
--                 zero delay time (i.e. Preset = Maximum).
--
--               20100903 (1.1)
--                 Initial revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity  DlyCounter  is
  generic (
    N         : natural := 16
  );

  port (
    Start,
    Clk,
    Rst       : in  std_logic;
    Enable    : in  std_logic   := '1';

    Maximum,
    Preset    : in  std_logic_vector(N - 1 downto 0);

    Running,
    Finished  : out std_logic;

    Count     : out std_logic_vector(N - 1 downto 0)
  );
end;

architecture  V1  of  DlyCounter  is

signal  Loc_Count   : signed(N - 1 downto 0);
signal  Loc_Started : std_logic;

begin
  process (Clk, Rst) begin
    -- Asynchronous reset.
    if Rst = '1' then
      Loc_Count   <= signed(Preset);
      Finished    <= '0';
      Loc_Started <= '0';
    elsif rising_edge(Clk) then
      -- Special case of zero delay time (i.e. preset equal to maximum).
      -- Also applies if Maximum incorrectly set less than Preset.
      -- Pulse is sent immediately as long as counter not already started.
      if signed(Maximum) <= signed(Preset) then
        if Start = '1' and Loc_Started = '0' then
          Loc_Count <= signed(Preset);
          Finished <= '1';
        else
```

```
          Finished <= '0';
        end if;
      -- Otherwise, begin pulse at next clock cycle.
      else
        -- Start?
        if Start = '1' and Loc_Started = '0' then
          Loc_Count   <= signed(Preset);
          Finished    <= '0';
          Loc_Started <= '1';
        -- Started and enable?
        elsif Enable = '1' and Loc_Started = '1' then
          if Loc_Count >= (signed(Maximum) - 1) then
            Finished    <= '1';
            Loc_Started <= '0';
          else
            Finished    <= '0';
            Loc_Started <= '1';
          end if;

          Loc_Count <= Loc_Count + 1;
        -- Otherwise, turn the pulse off to ensure single cycle length.
        else
          Finished <= '0';
        end if;
      end if;
    end if;
  end process;

  Count    <= std_logic_vector(Loc_Count);
  Running  <= Loc_Started;
end;

--EOF
```

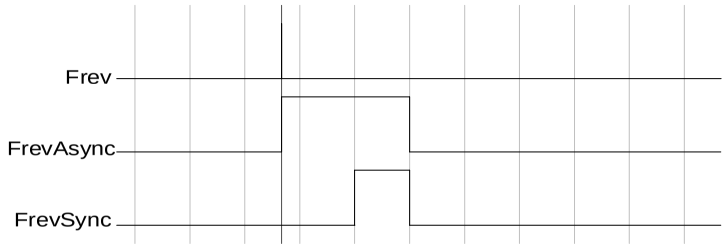
Clk

Rst

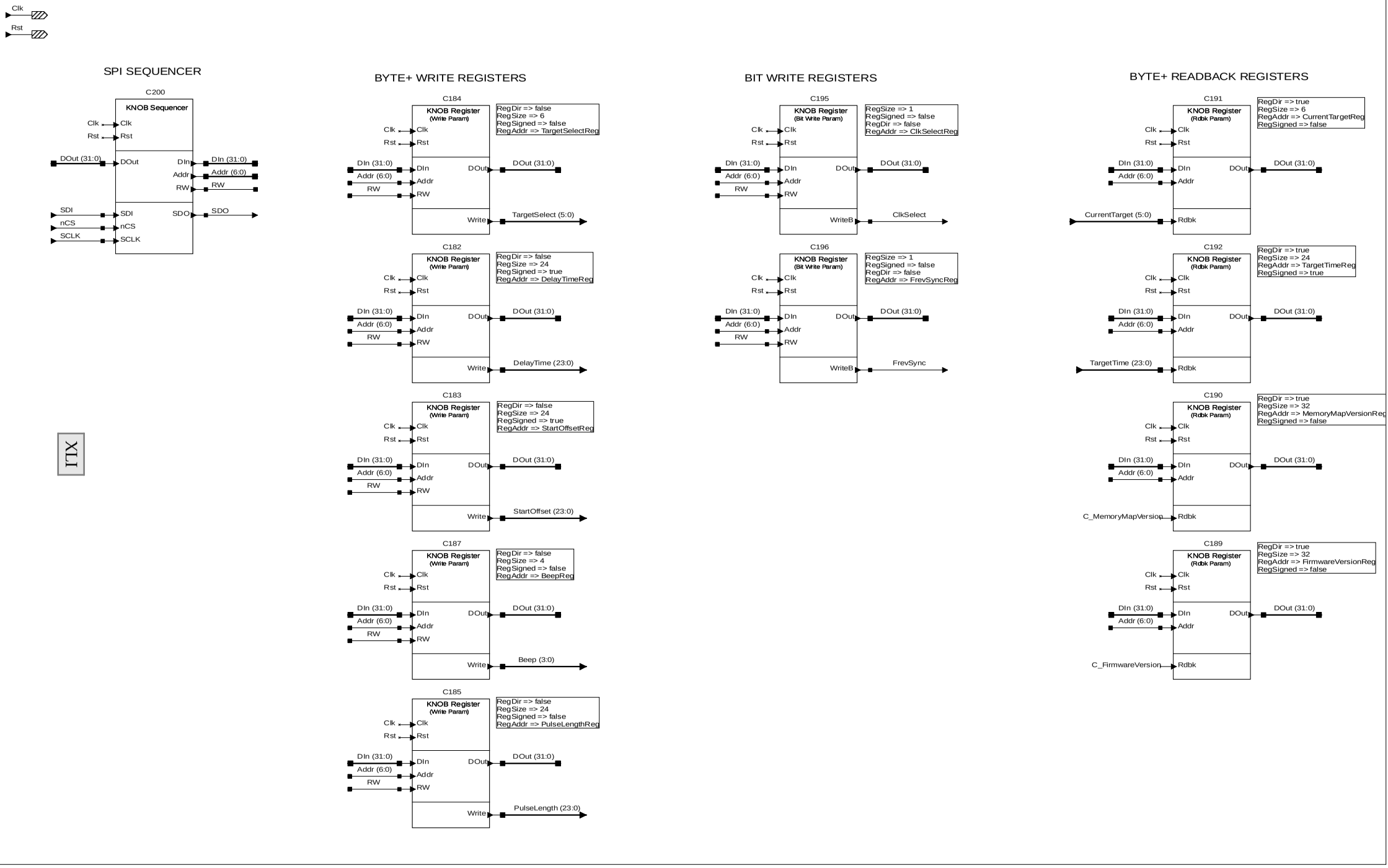
ASYNCHRONOUS F $_{REV}$ CAPTURE



XL

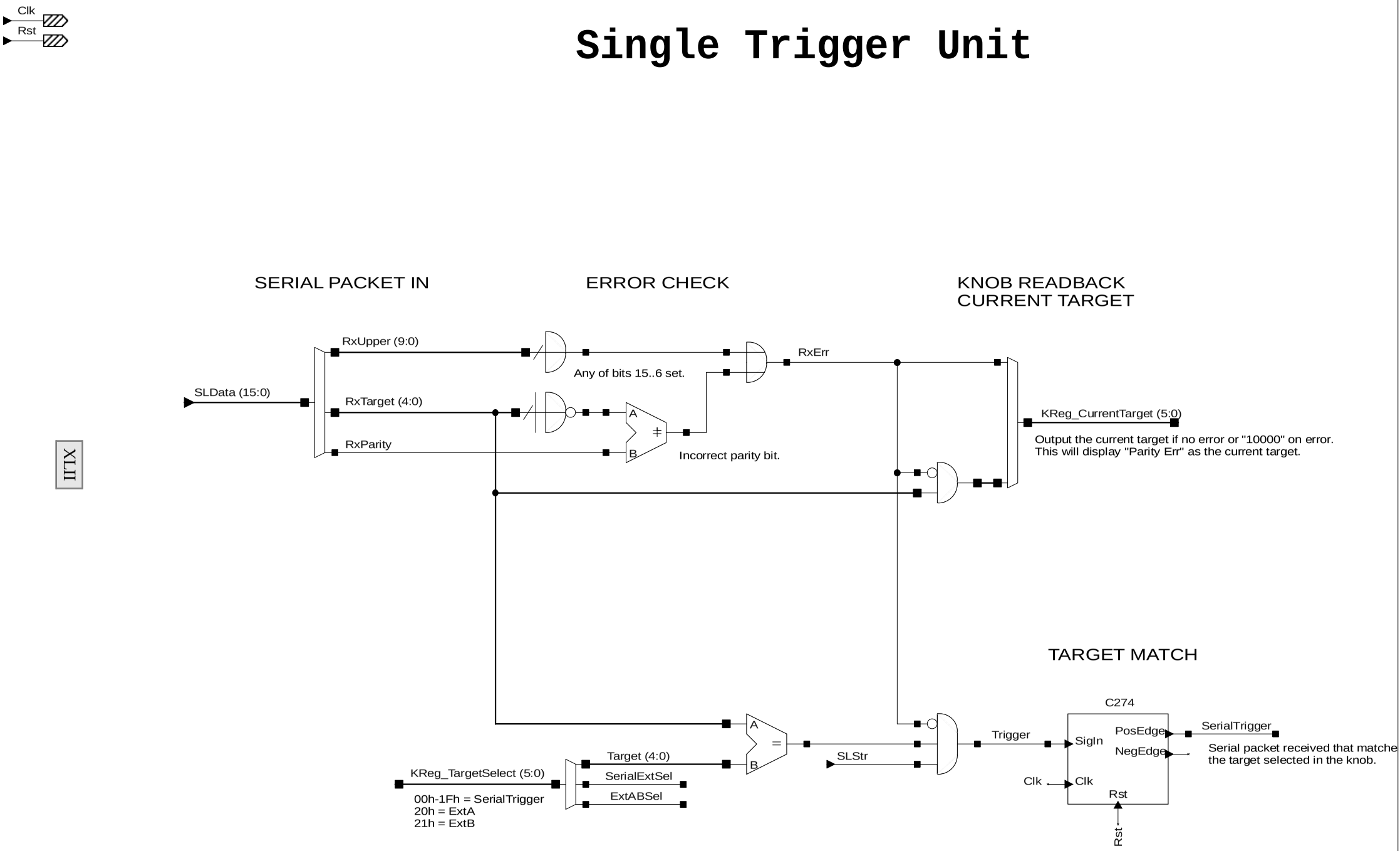


Design: DualTriggerUnit:FrevCapture	Date: Fri Dec 03 17:13:01 2010	Page: 1



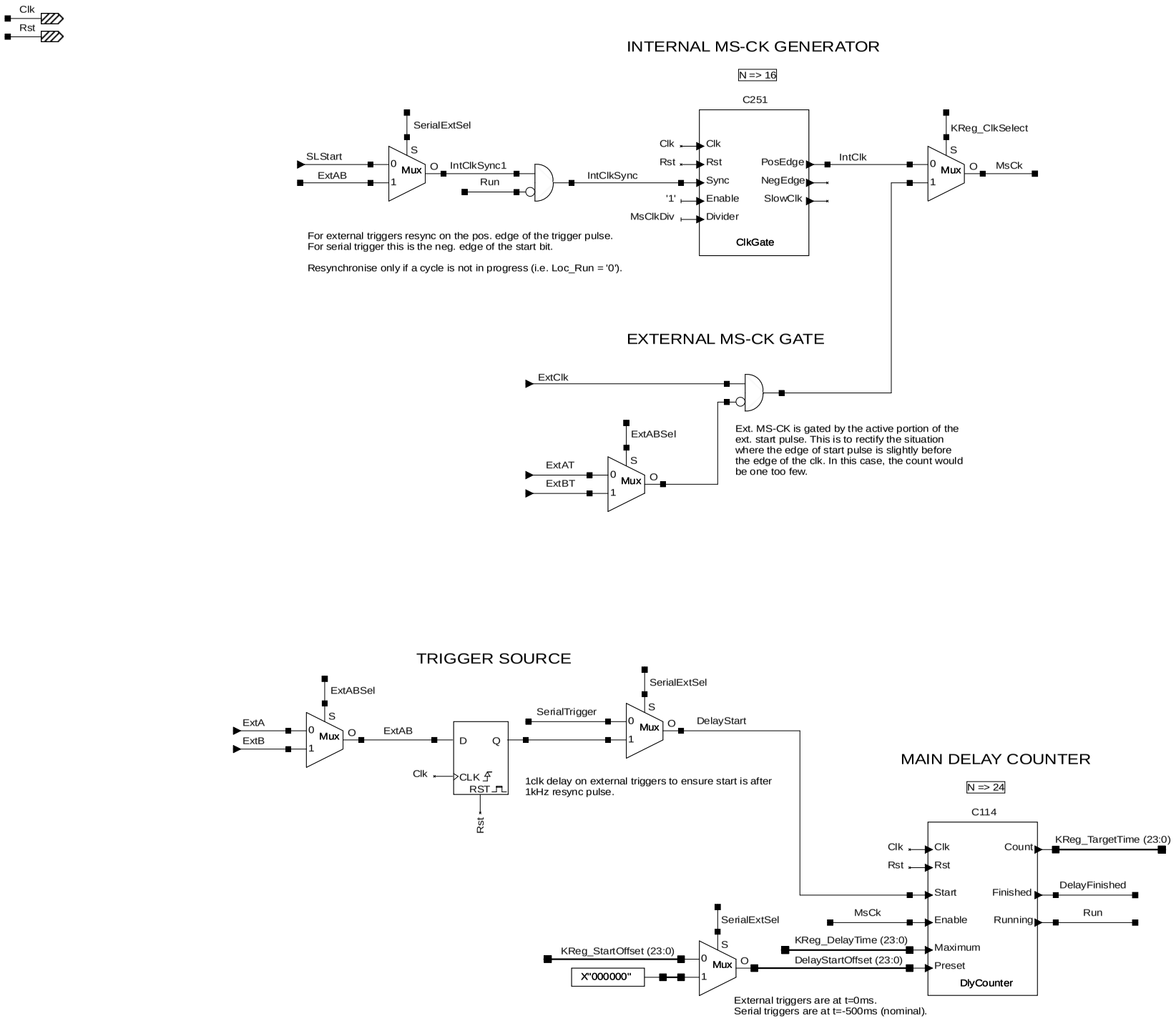
Design: DualTriggerUnit:KnobRegisters	Date: Fri Dec 03 17:13:30 2010	Page: 1

Single Trigger Unit



XLII

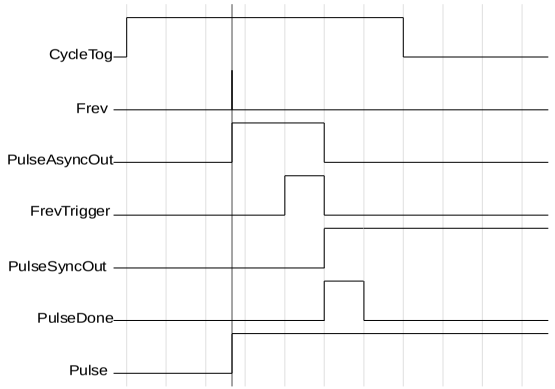
Design: DualTriggerUnit:SingleTriggerUnit (Sheet 1 of 4)	Date: Thu Dec 30 14:17:46 2010	Page: 1



XLIII

Design: DualTriggerUnit:SingleTriggerUnit (Sheet 2 of 4)	Date: Thu Dec 30 14:17:46 2010	Page: 2

When CycleT is high (i.e. the delay is complete), the timing should be as shown to the left. Frev is asynchronous to the clk, as is FrevAT. FrevPulseOut is the sum of FrevAT (to have the output pulse set as soon as Frev occurs) and PulseOut from the pulse generator.

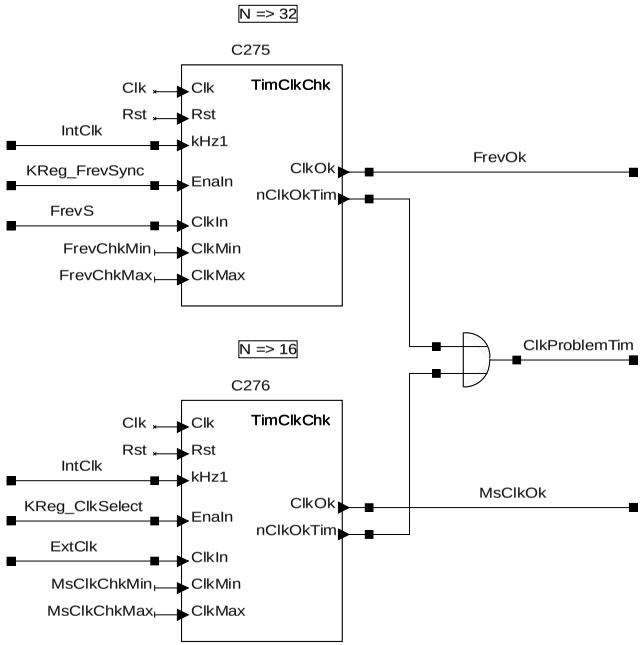


To avoid jitter with ext clock, this generator runs at 50MHz clock speed (SlowClk = 1). Knob has a 50x setting on parameter to select correct duration with microsecond granularity.

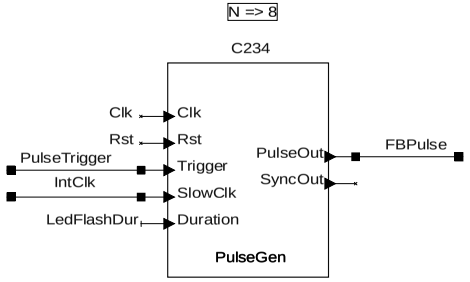
Design: DualTriggerUnit:SingleTriggerUnit (Sheet 3 of 4)	Date: Thu Dec 30 14:17:46 2010	Page: 3

Clk
Rst

CLOCK CHECKS

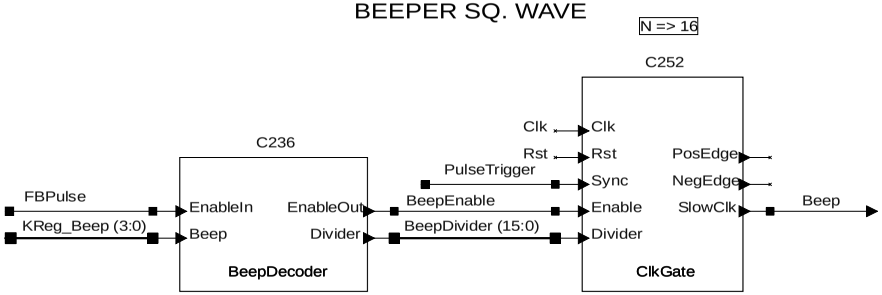


FEEDBACK PULSE

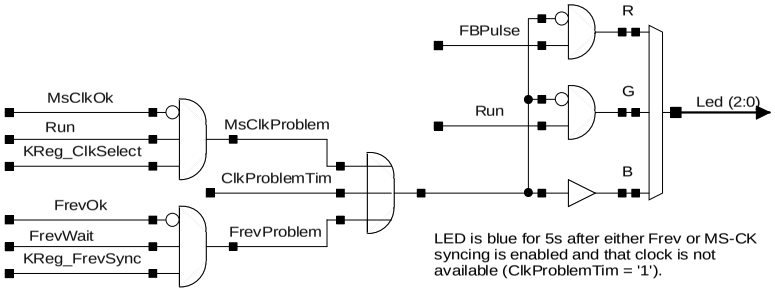


150ms duration for LED and BEEP.
No need for precise timing, so runs off int clk.

BEEPER SQ. WAVE



LED OUTPUT



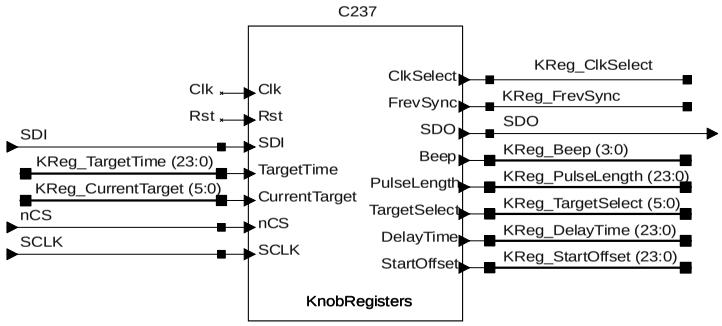
LED is blue for 5s after either Frev or MS-CK syncing is enabled and that clock is not available (ClkProblemTim = '1').

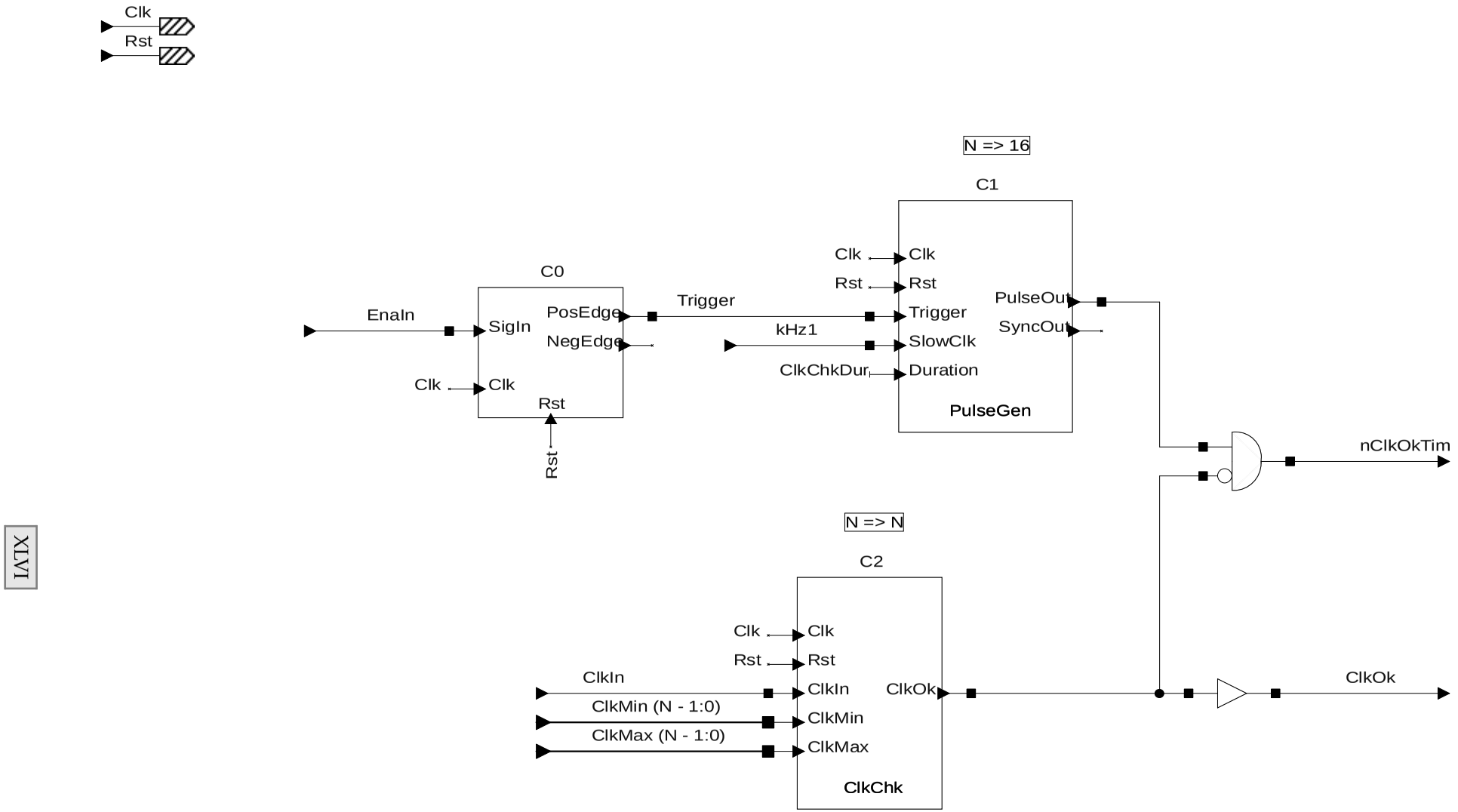
LED is blue when counter is running, MS-CK is selected and not available (MsClkProblem = '1').

LED is blue when awaiting Frev and Frev is not available (FrevProblem = '1').

Otherwise, show red/green LEDs as normal.

KNOB REGISTERS





Design: DualTriggerUnit:TimClkChk	Date: Thu Dec 30 14:09:45 2010	Page: 1 (1,1)

E Dual Trigger Unit — VHI Parameter Definition

E.1 DualTriggerUnit.h

```
#include "parameters.h"

const plist_item vlist1[] = {
  LIST_STARTING_STRING,
  #include "DualTriggerUnit_MMI.h"
  "Start 1 ",       // 20
  "Start 2 ",       // 21
  LIST_TERMINATION_STRING
};

const plist_item vlist2[] = {
  LIST_STARTING_STRING,
  #include "DualTriggerUnit_MMI.h"
  "Parity Err",       // 20
  LIST_TERMINATION_STRING
};

volatile parameter p0 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Trigger Source",           // unsigned char name[16]
  "",                         // unsigned char comment[16]
  10,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                         // number_of_bits
  0x10,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist1[0])      // address of the referenced list array
};

volatile parameter p1 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  SIGNED,                     // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Delay Time",               // unsigned char name[16]
  "[ms]",                     // unsigned char comment[16]
  6,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  -500.0,                     // range_min (FLOAT)
  99999.0,                    // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x11,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter p2 = {
  PARAM_INVISIBLE,            // basis (PARAM_EDIT, PARAM_READBACK)
  SIGNED,                     // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Start Offset",             // unsigned char name[16]
  "[ms]",                     // unsigned char comment[16]
  5,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  -1000.0,                    // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x12,                       // address (INT8)
  -500.0,                     // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};
```

```
volatile parameter p3 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  UNSIGNED,                   // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Pulse Width",              // unsigned char name[16]
  "[us]",                     // unsigned char comment[16]
  6,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  1.0,                        // range_min (FLOAT)
  99999.0,                    // range_max (FLOAT)
  50.0,                       // multiplier (double)
  32,                         // number_of_bits
  0x13,                       // address (INT8)
  20.0,                       // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter p4 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  BOOLEAN,                    // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Sync to Frev",             // unsigned char name[16]
  "",                         // unsigned char comment[16]
  1,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  1.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x14,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter p5 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  UNSIGNED,                   // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Beep",                     // unsigned char name[16]
  "[0=Off, 1..4=On]",         // unsigned char comment[16]
  1,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  4.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x15,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter p6 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  BOOLEAN,                    // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "MS-CK Int/Ext",            // unsigned char name[16]
  "[0=Int, 1=Ext]",           // unsigned char comment[16]
  1,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  1.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x16,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r0 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "RDBK: Cur. Cycle",         // unsigned char name[16]
```

```
  "",                         // unsigned char comment[16]
  10,                         // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x17,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  (unsigned)(&vlist2[0])      // address of the referenced list array
};

volatile parameter r1 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  SIGNED,                     // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "RDBK: Trig. Time",         // unsigned char name[16]
  "[ms]",                     // unsigned char comment[16]
  6,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  -500.0,                     // range_min (FLOAT)
  99999.0,                    // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x18,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r2 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  UNSIGNED,                   // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Firmware Version",         // unsigned char name[16]
  "",                         // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x00,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r3 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  UNSIGNED,                   // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Mem Map Version",          // unsigned char name[16]
  "",                         // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x01,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Keep this parameter for proper MENU FPGA WRITE definition.
// Mandatory setting is only BASIS = PARAM_MENU, other vars are ignored for now
// (KNOB version 1.0)
volatile parameter write_menu = {
  PARAM_MENU,                 // basis (PARAM_EDIT, PARAM_READBACK)
  FPGA_WRITE,                 // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "FPGA STORE",               // unsigned char name[16]
  "Push to store parameters", // unsigned char comment[16]
```

```
  0,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  0.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x00,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#define MAX_PARAMETERS  (sizeof(plist)/4)

// parameters list
volatile parameter *plist[] = { &p0, &p1, &r0, &r1, &p2, &p3, &p4, &p5, &p6,
                                &write_menu,
                                &r2 };

//EOF
```

E.2 DualTriggerUnit_MMI.h

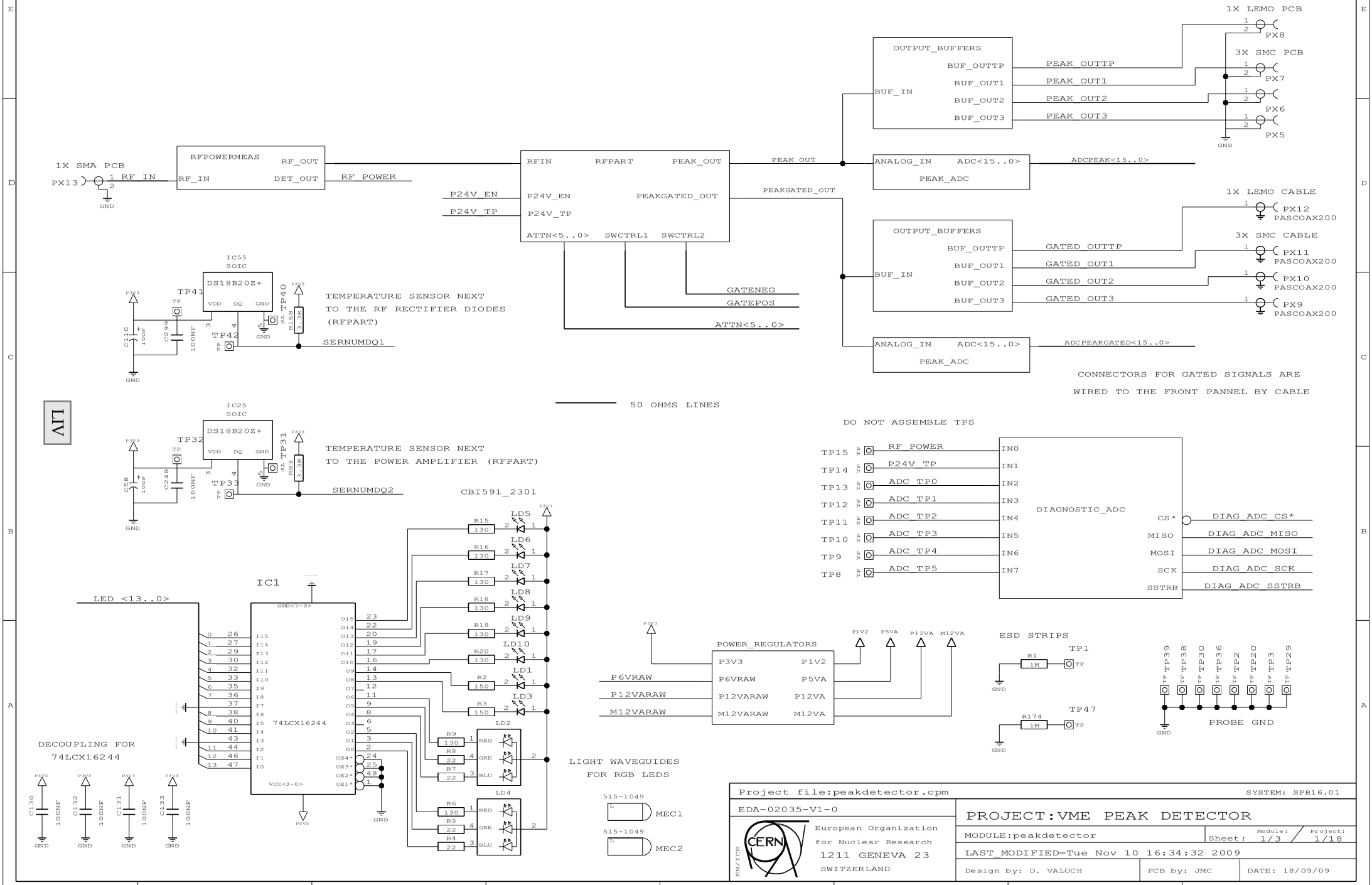
```
/*

  The following list defines the 31 MMI targets for the "Trigger Source" and
  "Current MMI Targ" lists.

  Notes:
  - Each target name is 10 characters (maximum).
  - Please make sure all 31 targets are present.
  - This list is #included into an array definition so please maintain the
    formatting!

*/

//"1234567890"
  "Zero      ",     // 0x00
  "SFTPRO1   ",     // 0x01
  "SFTPRO2   ",     // 0x02
  "SFT25NS   ",     // 0x03
  "MD1       ",     // 0x04
  "MD2       ",     // 0x05
  "IonsFixTrg",     // 0x06
  "IonsRecapt",     // 0x07
  "Res Stk 08",     // 0x08
  "Res Stk 09",     // 0x09
  "LHCPILOT  ",     // 0x0A
  "LHCMONO   ",     // 0x0B
  "LHC12BU   ",     // 0x0C
  "LHC25NS   ",     // 0x0D
  "LHC75NS   ",     // 0x0E
  "LHCMD     ",     // 0x0F
  "LHCSCRUB  ",     // 0x10
  "LHCION    ",     // 0x11
  "Res Stk 18",     // 0x12
  "Res Stk 19",     // 0x13
  "CNGS1     ",     // 0x14
  "CNGS2     ",     // 0x15
  "CNGS3     ",     // 0x16
  "Res Stk 23",     // 0x17
  "Res Stk 24",     // 0x18
  "Res Stk 25",     // 0x19
  "Res Stk 26",     // 0x1A
  "Res Stk 27",     // 0x1B
  "Res Stk 28",     // 0x1C
  "Res Stk 29",     // 0x1D
  "Res Stk 30",     // 0x1E
  "Res Stk 31",     // 0x1F
//"1234567890"

//EOF
```

F VME Peak Detector — Design Files

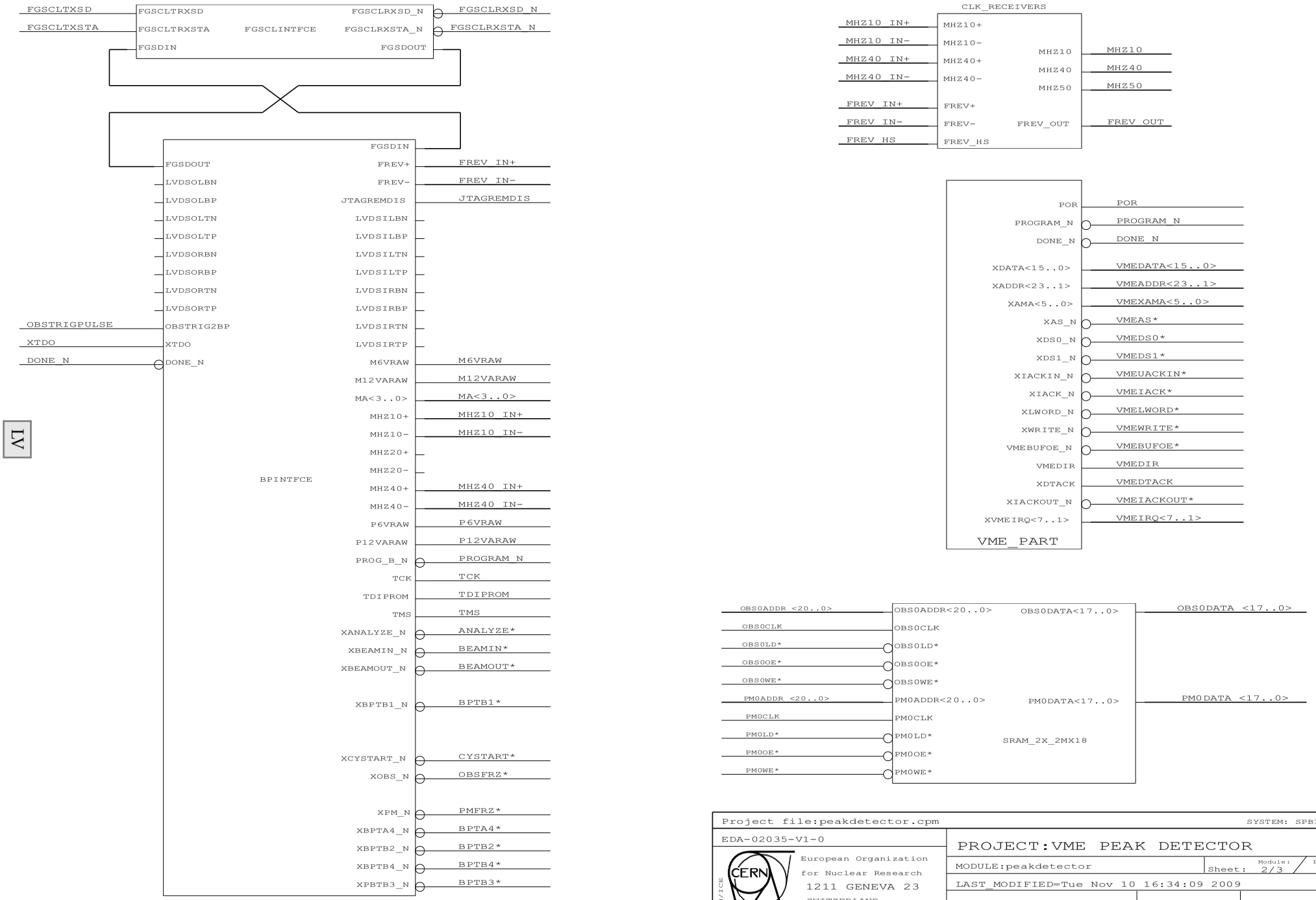
This appendix contains the schematic for the VME Peak Detector. It also includes the modifications made to the RF part of the schematic. The following documents are included:

The original schematics were prepared by Daniel Valuch. Duplicate pages of the schematic are omitted for clarity. Further small modifications are required to the non-RF parts of the schematic, but these are not included.

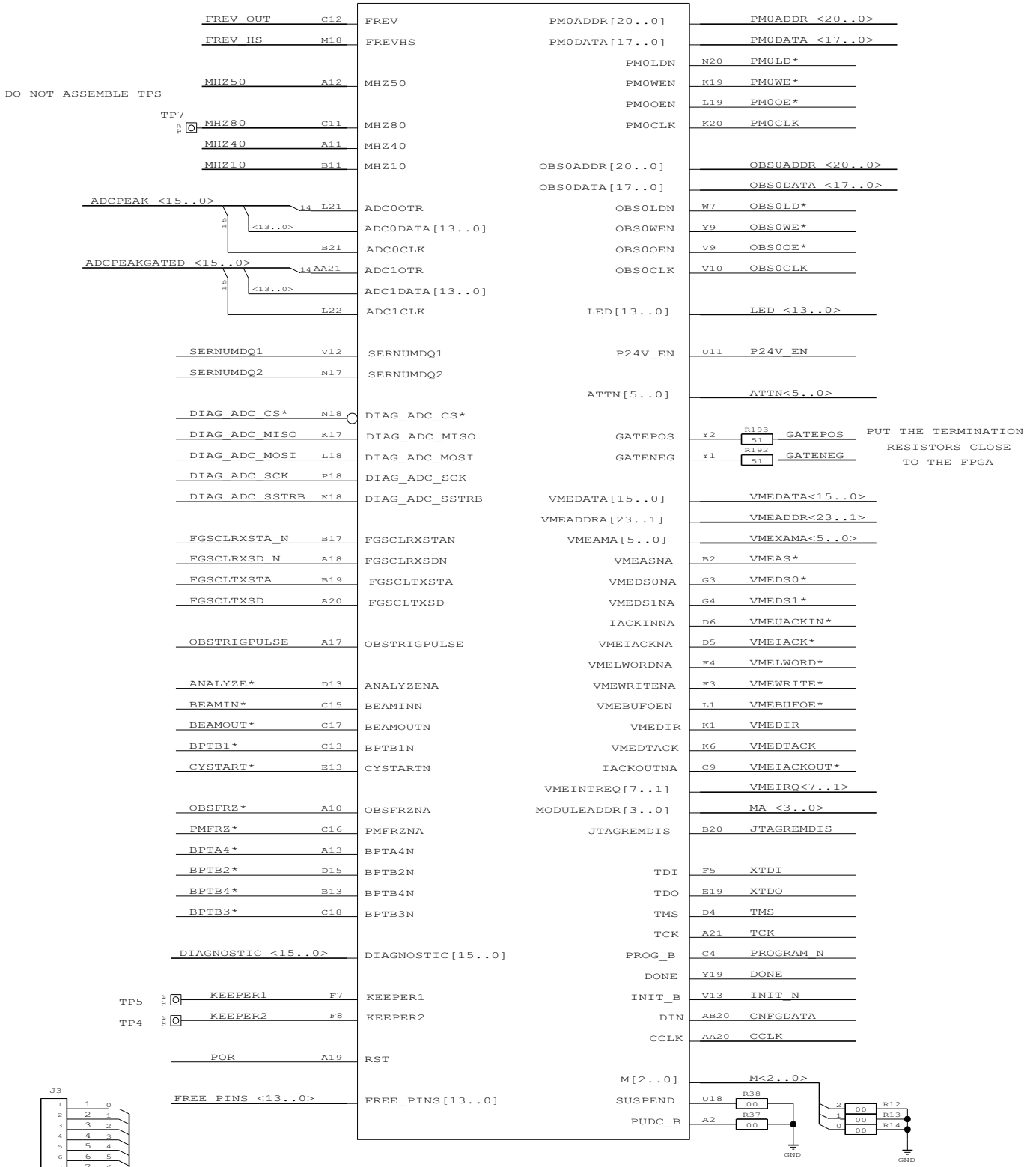
All files related the the VPD are published on CERN's EDMS service with the identifier EDA-02035-V1-0 [27].



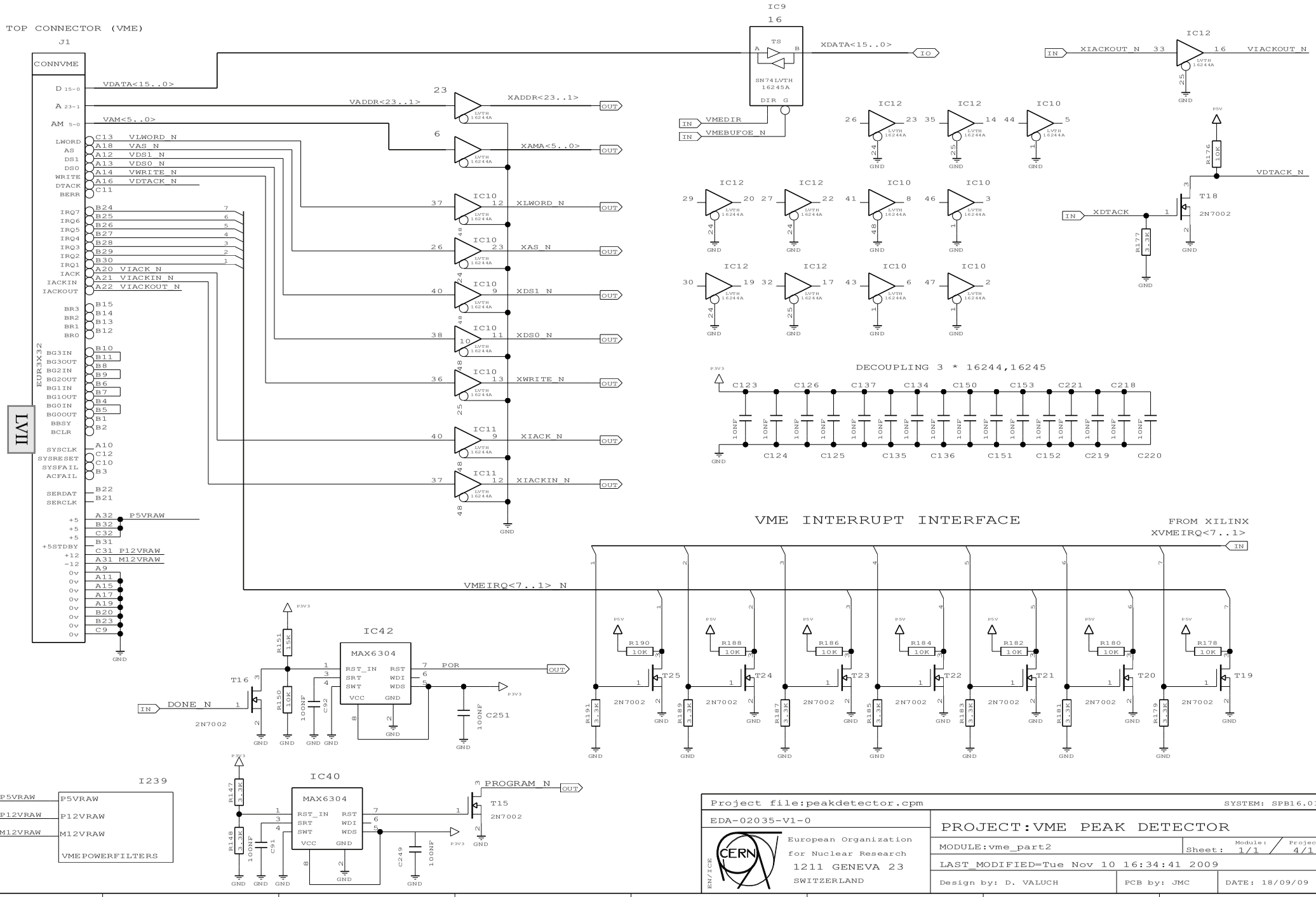
Project file:peakdetector.cpm			SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR		
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:peakdetector	Sheet: 1/3	Module: / Project: 1/18
	LAST_MODIFIED=Tue Nov 10 16:34:32 2009		
	Design by: D. VALUCH	PCB by: JMC	DATE: 18/09/09

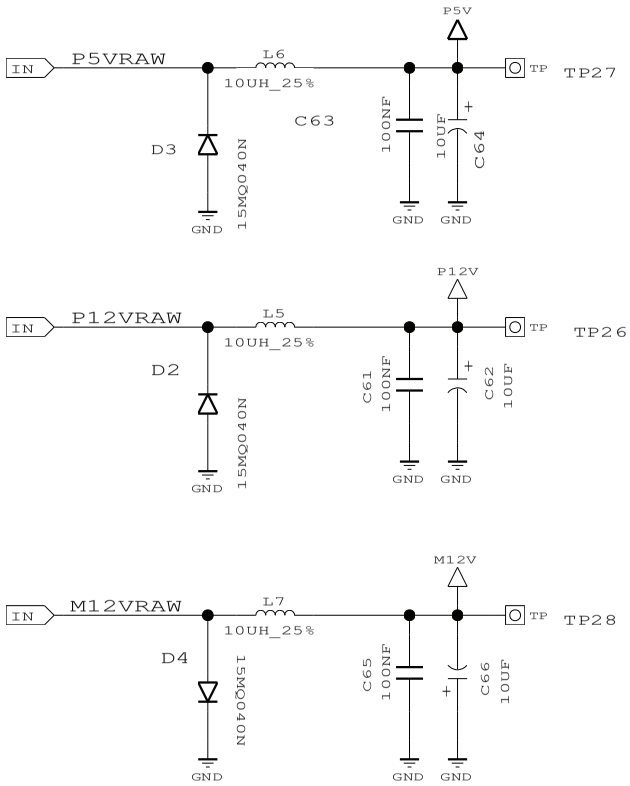

FGSCLTXSD
FGSCLTXSTA
FGSCLTRXSD
FGSCLTRXSTA
FGSDIN
FGSCLINTFCE
FGSCLRXSD_N
FGSCLRXSTA_N
FGSDOUT
FGSCLRXSD_N
FGSCLRXSTA_N
FGSDIN
FGSDOUT
LVDSOLBN
LVDSOLBP
LVDSOLTN
LVDSOLTP
LVDSORBN
LVDSORBP
LVDSORTN
LVDSORTP
OBSTRIGPULSE
OBSTRIG2BP
XTDO
XTDO
DONE_N
DONE_N
BPINTFCE
FREV+
FREV-
JTAGREMDIS
LVDSILBN
LVDSILBP
LVDSILTN
LVDSILTP
LVDSIRBN
LVDSIRBP
LVDSIRTN
LVDSIRTP
M6VRAW
M12VARAW
MA<3..0>
MHZ10+
MHZ10-
MHZ20+
MHZ20-
MHZ40+
MHZ40-
P6VRAW
P12VARAW
PROG_B_N
TCK
TDIPROM
TMS
XANALYZE_N
XBEAMIN_N
XBEAMOUT_N
XBPTB1_N
XCYSTART_N
XOBS_N
XPM_N
XBPTA4_N
XBPTB2_N
XBPTB4_N
XPBTB3_N
FREV_IN+
FREV_IN-
JTAGREMDIS
M6VRAW
M12VARAW
MA<3..0>
MHZ10_IN+
MHZ10_IN-
MHZ40_IN+
MHZ40_IN-
P6VRAW
P12VARAW
PROGRAM_N
TCK
TDIPROM
TMS
ANALYZE*
BEAMIN*
BEAMOUT*
BPTB1*
CYSTART*
OBSFRZ*
PMFRZ*
BPTA4*
BPTB2*
BPTB4*
BPTB3*
LV
CLK_RECEIVERS
MHZ10_IN+
MHZ10_IN-
MHZ40_IN+
MHZ40_IN-
FREV_IN+
FREV_IN-
FREV_HS
MHZ10+
MHZ10-
MHZ40+
MHZ40-
FREV+
FREV-
FREV_HS
MHZ10
MHZ40
MHZ50
FREV_OUT
MHZ10
MHZ40
MHZ50
FREV_OUT
POR
PROGRAM_N
DONE_N
XDATA<15..0>
XADDR<23..1>
XAMA<5..0>
XAS_N
XDS0_N
XDS1_N
XIACKIN_N
XIACK_N
XLWORD_N
XWRITE_N
VMEBUFOE_N
VMEDIR
XDTACK
XIACKOUT_N
XVMEIRQ<7..1>
VME_PART
POR
PROGRAM_N
DONE_N
VMEDATA<15..0>
VMEADDR<23..1>
VMEXAMA<5..0>
VMEAS*
VMEDS0*
VMEDS1*
VMEUACKIN*
VMEIACK*
VMELWORD*
VMEWRITE*
VMEBUFOE*
VMEDIR
VMEDTACK
VMEIACKOUT*
VMEIRQ<7..1>
OBS0ADDR <20..0>
OBS0CLK
OBS0LD*
OBS0OE*
OBS0WE*
PM0ADDR <20..0>
PM0CLK
PM0LD*
PM0OE*
PM0WE*
OBS0ADDR<20..0>
OBS0CLK
OBS0LD*
OBS0OE*
OBS0WE*
PM0ADDR<20..0>
PM0CLK
PM0LD*
PM0OE*
PM0WE*
OBS0DATA<17..0>
PM0DATA<17..0>
SRAM_2X_2MX18
OBS0DATA <17..0>
PM0DATA <17..0>
Project file:peakdetector.cpm
SYSTEM: SPB16.01
EDA-02035-V1-0
European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND
EN/ICE
PROJECT:VME PEAK DETECTOR
MODULE:peakdetector
Module: Sheet: 2/3
Project: 2/18
LAST_MODIFIED=Tue Nov 10 16:34:09 2009
Design by: D. VALUCH
PCB by: JMC
DATE: 18/09/09

IC8
peakdetector_fpga_top



Project file:peakdetector.cpm			SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR		
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND (CERN, LVI, TS/DEM)	MODULE:peakdetector	Sheet: 3/3	Module: / Project: 3/18
	LAST_MODIFIED=Tue Nov 10 16:34:26 2009		
	Design by: D. VALUCH	PCB by: JMC	DATE: 18/09/09

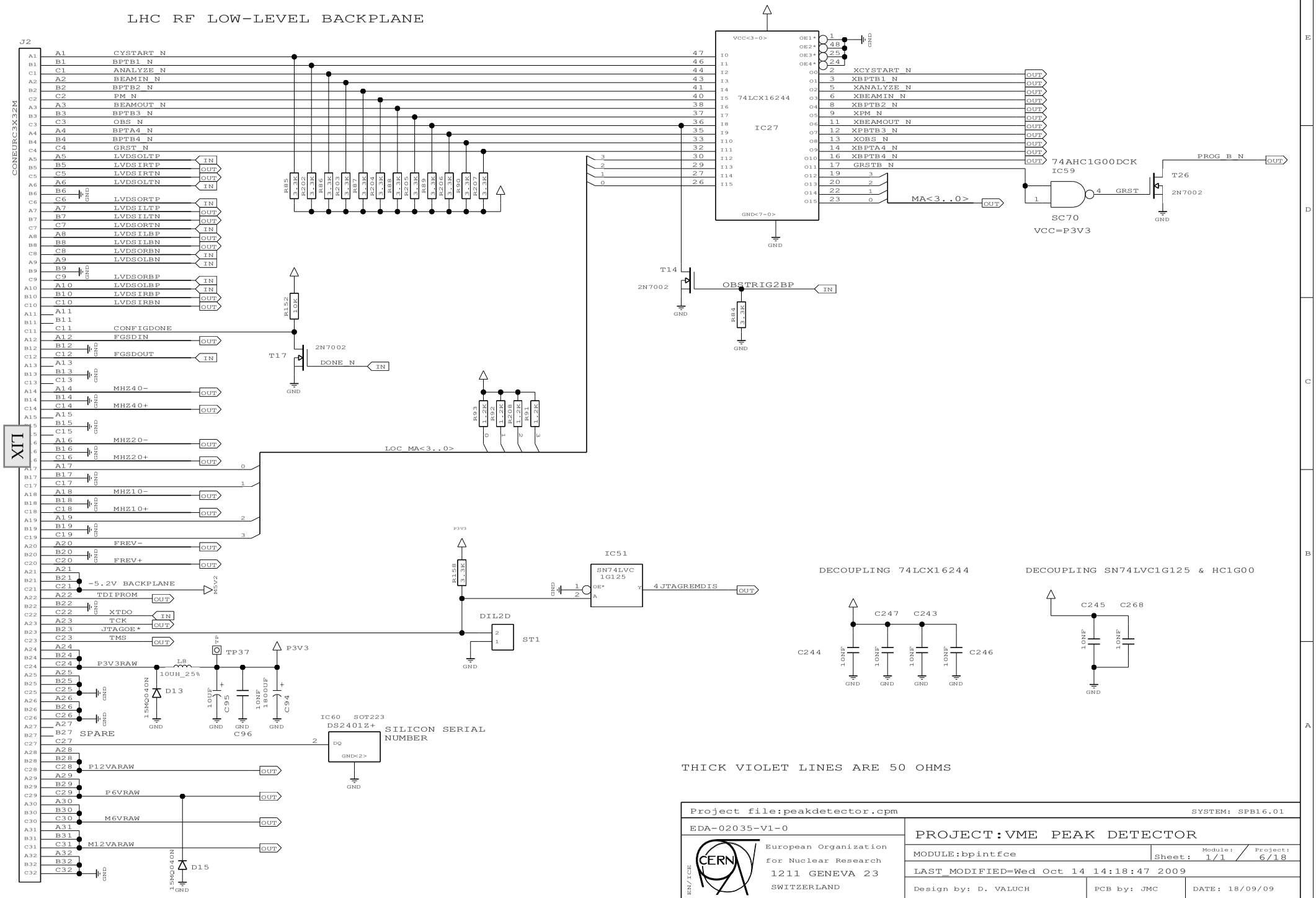

TOP CONNECTOR (VME)
J1
CONNVME
VDATA<15..0>
VADDR<23..1>
VAM<5..0>
VLWORD_N
VAS_N
VDS1_N
VDS0_N
VWRITE_N
VDTACK_N
VIACK_N
VIACKIN_N
VIACKOUT_N
P5VRAW
P12VRAW
M12VRAW
EUR3X32
LVIII
IC9
SN74LVTH 16245A
XDATA<15..0>
VMEDIR
VMEBUFOE_N
XADDR<23..1>
XAMA<5..0>
XLWORD_N
XAS_N
XDS1_N
XDS0_N
XWRITE_N
XIACK_N
XIACKIN_N
IC12
XIACKOUT_N
VIACKOUT_N
IC10
R176
VDTACK_N
T18
2N7002
XDTACK
R177
DECOUPLING 3 * 16244,16245
C123
C126
C137
C134
C150
C153
C221
C218
C124
C125
C135
C136
C151
C152
C219
C220
VME INTERRUPT INTERFACE
FROM XILINX
XVMEIRQ<7..1>
VMEIRQ<7..1>_N
T25
T24
T23
T22
T21
T20
T19
IC42
MAX6304
POR
DONE_N
T16
IC40
PROGRAM_N
T15
I239
VMEPOWERFILTERS
Project file:peakdetector.cpm
SYSTEM: SPB16.01
EDA-02035-V1-0
European Organization for Nuclear Research
1211 GENEVA 23
SWITZERLAND
PROJECT:VME PEAK DETECTOR
MODULE:vme_part2
Sheet: 1/1
Module: 1/1
Project: 4/18
LAST_MODIFIED=Tue Nov 10 16:34:41 2009
Design by: D. VALUCH
PCB by: JMC
DATE: 18/09/09



LVIII

Project file:peakdetector.cpm			SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR		
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND (TS/DEM)	MODULE:vmepowerfilters	Sheet: 1/1 (Module: / Project: 5/18)	
	LAST_MODIFIED=Wed Oct 14 14:18:41 2009		
	Design by: D. VALUCH	PCB by: JMC	DATE: 18/09/09

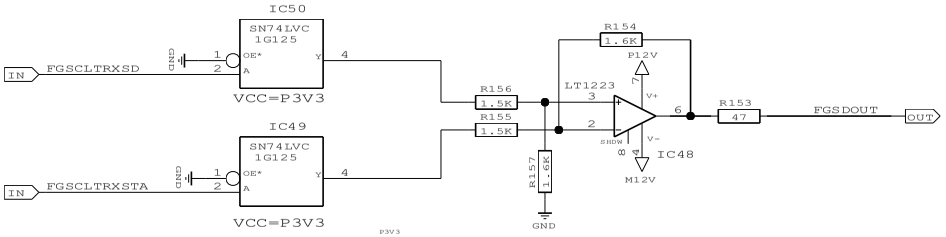
LHC RF LOW-LEVEL BACKPLANE



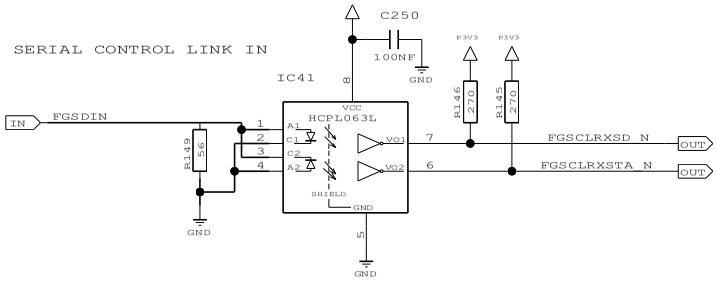
Project file:peakdetector.cpm		SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:bpintfce	Module: 1/1 Project: 6/18
EN/ICE	LAST_MODIFIED=Wed Oct 14 14:18:47 2009	
	Design by: D. VALUCH	PCB by: JMC DATE: 18/09/09

XT

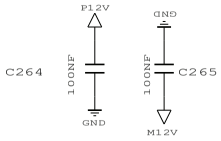
SERIAL CONTROL LINK OUT



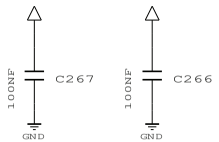
SERIAL CONTROL LINK IN



DECOUPLING LT1223



DECOUPLING 74*125

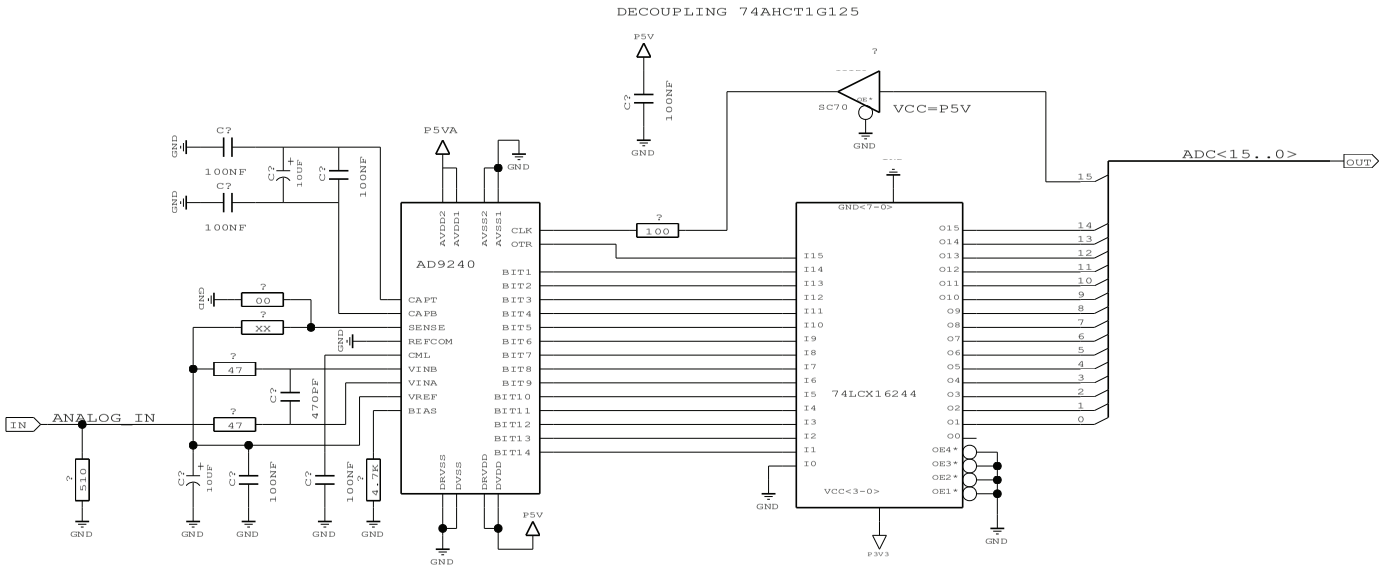


THICK VIOLET LINES ARE 50 OHMS

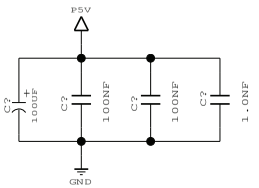
Project file:peakdetector.cpm	SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND (EN/ICE)	MODULE:fgsclintfce — Sheet: 1/1 — Module: 1/1 — Project: 7/18
	LAST_MODIFIED=Wed Oct 14 14:18:55 2009
	Design by: D. VALUCH — PCB by: JMC — DATE: 18/09/09

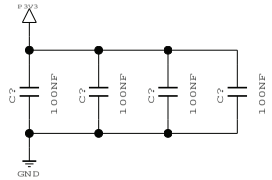
LXI

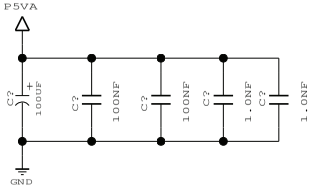
INPUT RANGE
0 TO +5V



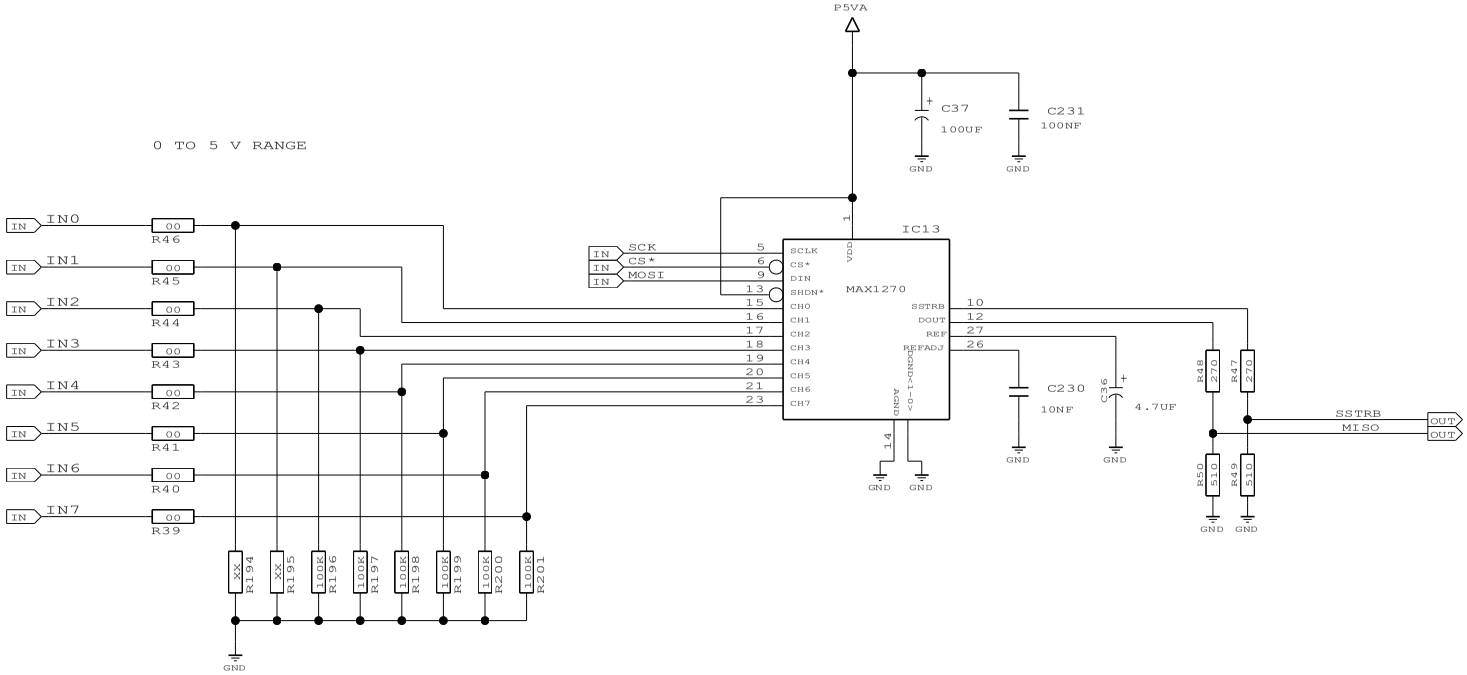
INPUT SPAN
2V - CONNECT SENSE TO VREF
5V - CONNECT SENSE TO REFCOM



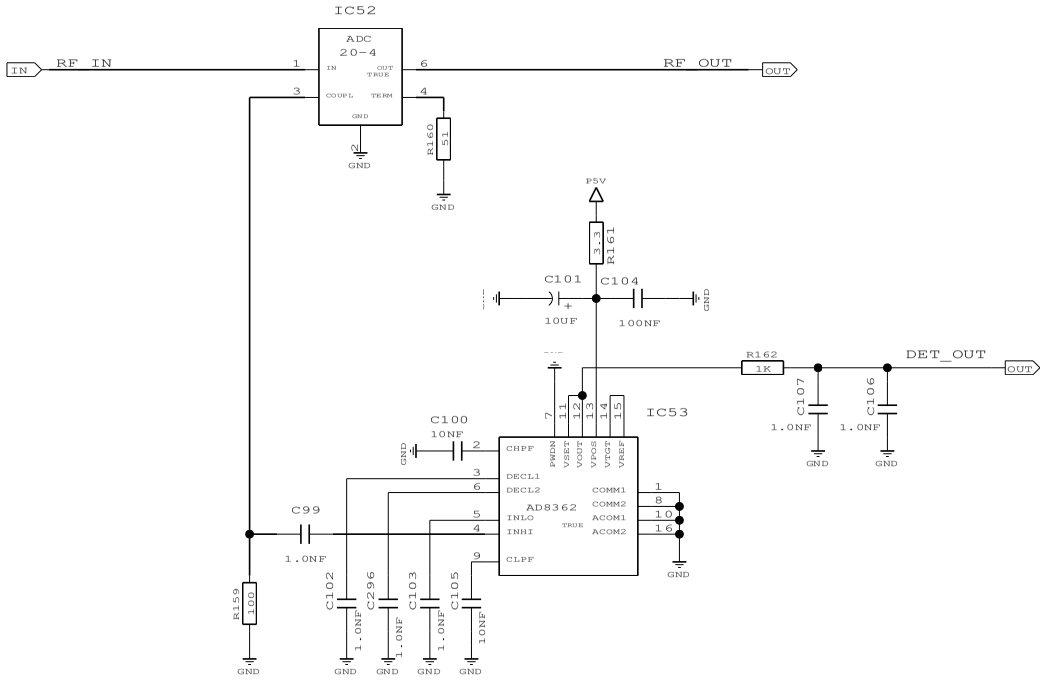




Project file:peakdetector.cpm		SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT: VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND (EN/ICE)	MODULE:peak_adc	Sheet: 1/1 (Module: / Project: 8/18)
	LAST_MODIFIED=Wed Oct 14 14:17:45 2009	
	Design by: D. VALUCH	PCB by: JMC / DATE: 18/09/09



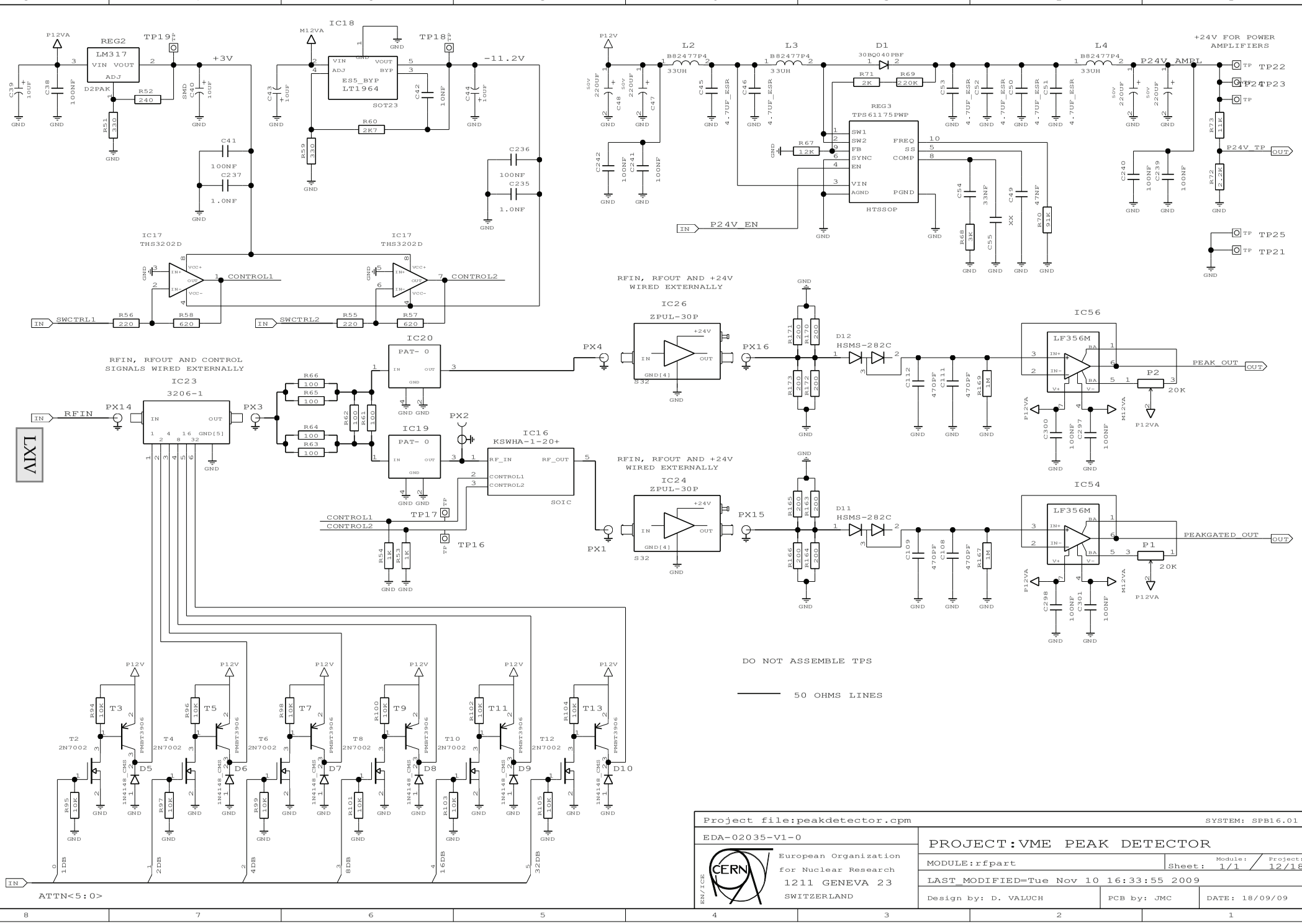
Project file:peakdetector.cpm			SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR		
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:diagnostic_adc	Sheet: 1/1	Module: 1/1 / Project: 10/18
EN/ICE	LAST_MODIFIED=Wed Oct 14 14:17:53 2009		
	Design by: D. VALUCH	PCB by: JMC	DATE: 18/09/09

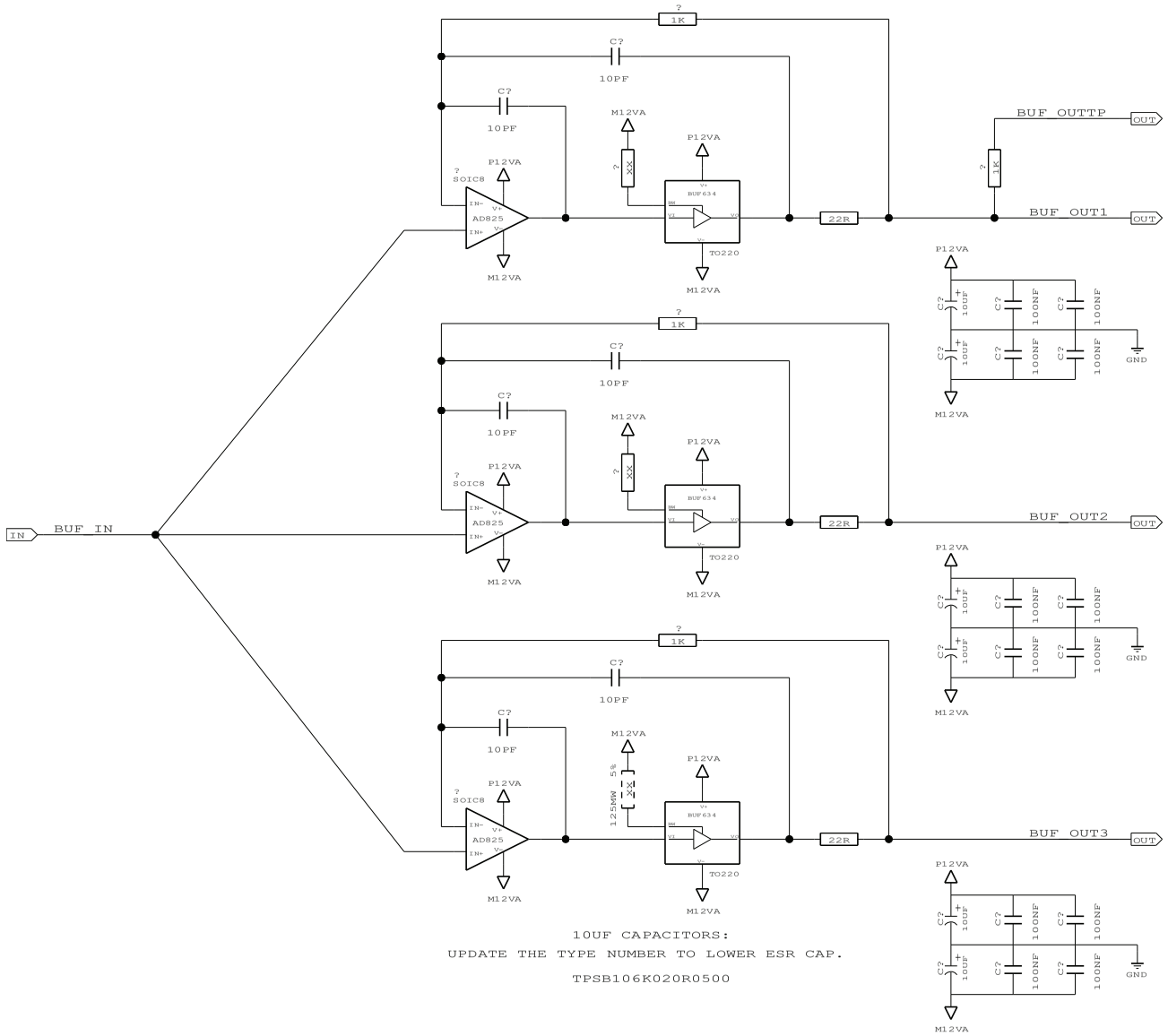


LXIII

THICK PURPLE LINES ARE 50OHM

Project file:peakdetector.cpm		SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND (EN/ICE)	MODULE:rfpowermeas	Sheet: Module: 1/1 Project: 11/18
	LAST_MODIFIED=Wed Oct 14 14:17:58 2009	
	Design by: D. VALUCH	PCB by: JMC DATE: 18/09/09


REG2
LM317
VIN VOUT
ADJ
D2PAK
+3V
TP19
IC18
LT1964
ES5_BYP
SOT23
-11.2V
TP18
L2
L3
L4
B82477P4
33UH
D1
30BQ040PBF
+24V FOR POWER AMPLIFIERS
P24V_AMPL
REG3
TPS61175PWP
HTSSOP
P24V_EN
P24V_TP
TP22
TP24
TP23
TP25
TP21
IC17
THS3202D
CONTROL1
CONTROL2
SWCTRL1
SWCTRL2
RFIN, RFOUT AND +24V WIRED EXTERNALLY
IC26
ZPUL-30P
IC24
IC56
IC54
LF356M
D12
D11
HSMS-282C
PEAK_OUT
PEAKGATED_OUT
P2
P1
20K
RFIN, RFOUT AND CONTROL SIGNALS WIRED EXTERNALLY
IC23
3206-1
IC20
IC19
PAT- 0
IC16
KSWHA-1-20+
SOIC
RFIN
PX1
PX2
PX3
PX4
PX14
PX15
PX16
TP16
TP17
CONTROL1
CONTROL2
LXIV
T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13
2N7002
PMBT3906
1N4148_CMS
D5 D6 D7 D8 D9 D10
ATTN<5:0>
1DB
2DB
4DB
8DB
16DB
32DB
DO NOT ASSEMBLE TPS
50 OHMS LINES
Project file:peakdetector.cpm
SYSTEM: SPB16.01
EDA-02035-V1-0
European Organization for Nuclear Research
1211 GENEVA 23
SWITZERLAND
EN/ICE
PROJECT:VME PEAK DETECTOR
MODULE:rfpart
Sheet: 1/1
Module: 1/1
Project: 12/18
LAST_MODIFIED=Tue Nov 10 16:33:55 2009
Design by: D. VALUCH
PCB by: JMC
DATE: 18/09/09

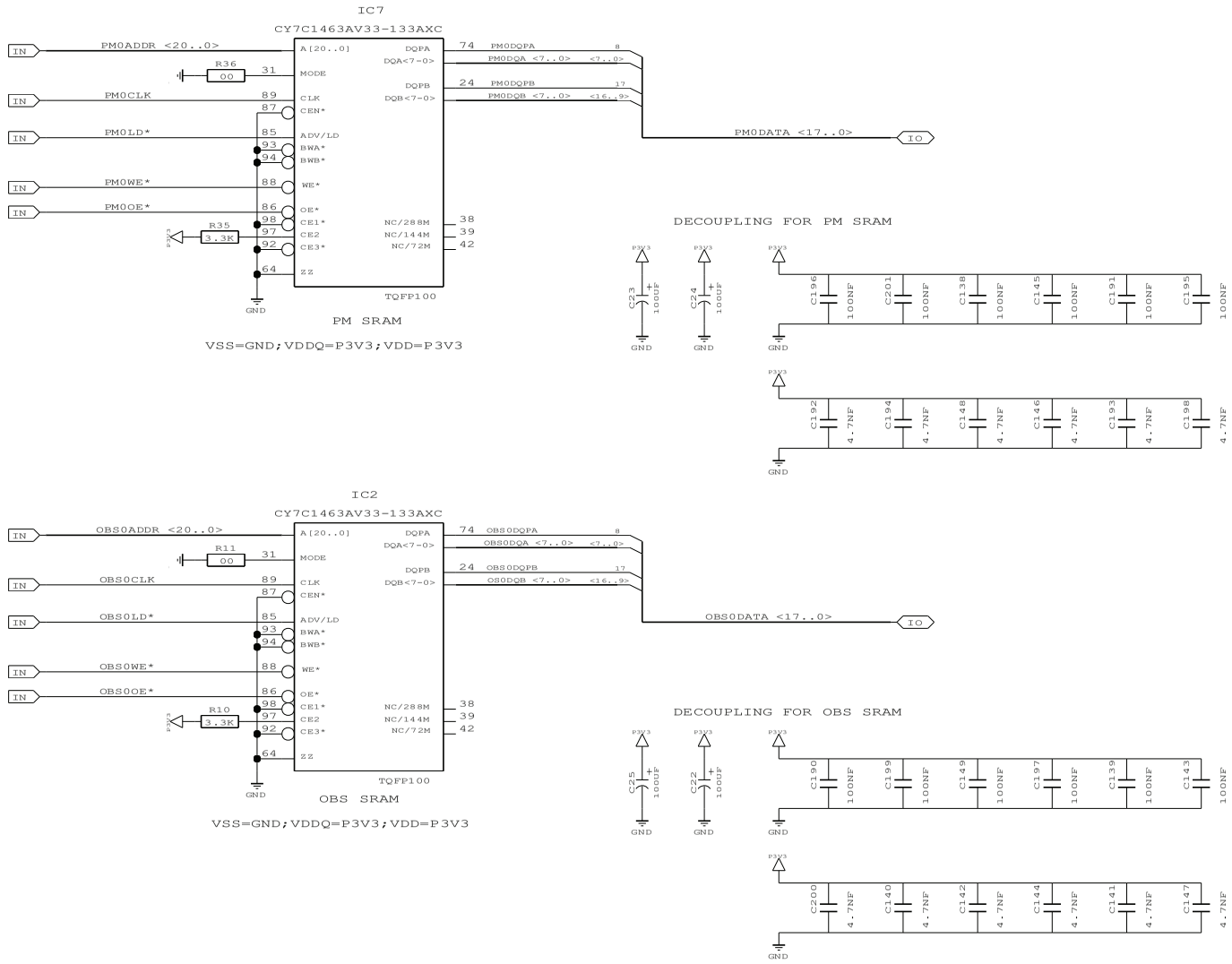


10UF CAPACITORS:

UPDATE THE TYPE NUMBER TO LOWER ESR CAP.

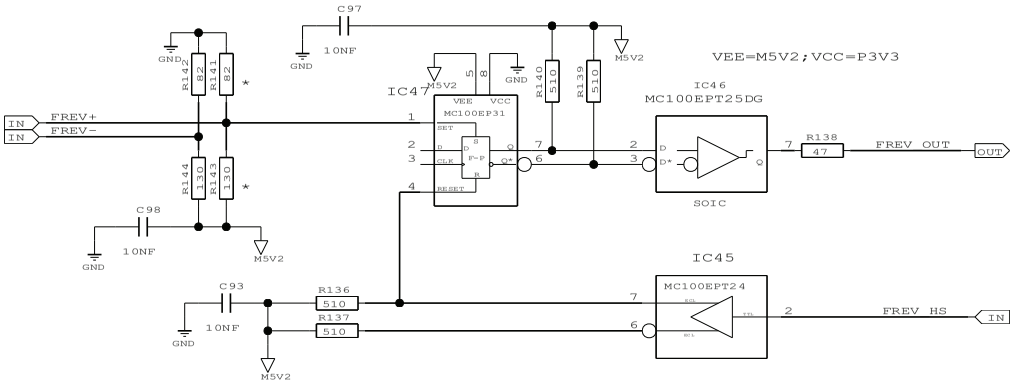
TPSB106K020R0500

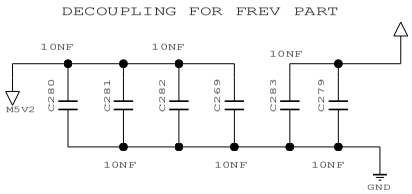
Project file:peakdetector.cpm		SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:output_buffers	Sheet: 1/1 (Module: / Project: 13/18)
EN/ICE	LAST_MODIFIED=Wed Oct 14 14:18:15 2009	
	Design by: D. VALUCH	PCB by: JMC / DATE: 18/09/09

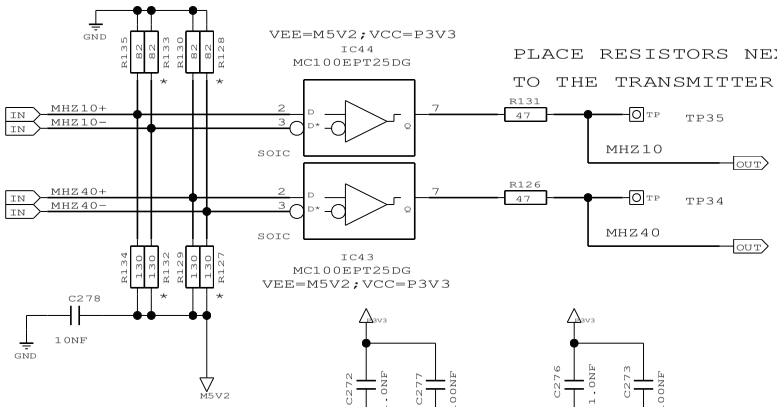


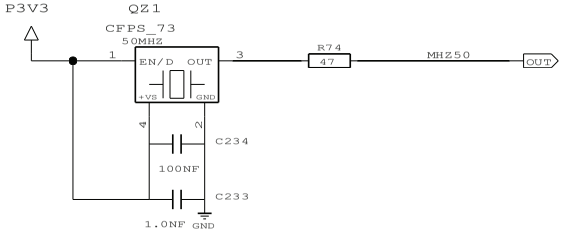
LXVI

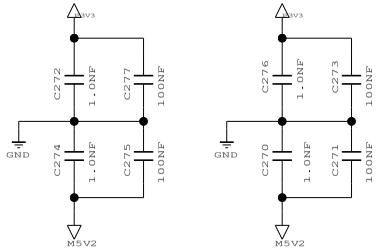
Project file:peakdetector.cpm			SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT: VME PEAK DETECTOR		
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:sram_2x_2mx18	Sheet: 1/1	Module: 1/1 / Project: 15/18
	LAST_MODIFIED=Tue Nov 10 16:34:19 2009		
	Design by: D. VALUCH	PCB by: JMC	DATE: 18/09/09









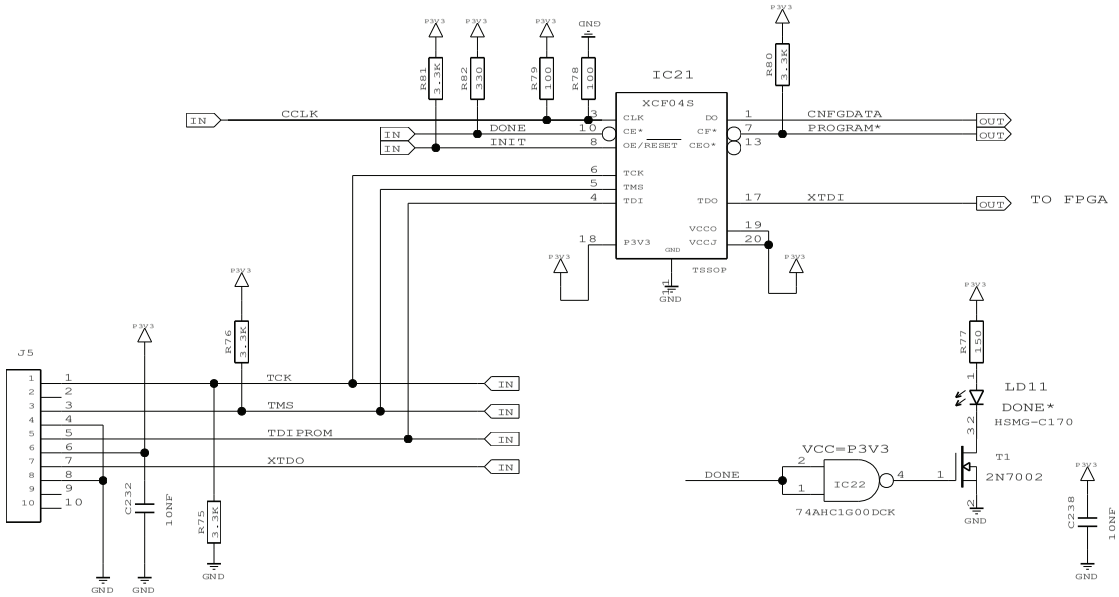


THICK VIOLET LINES ARE 50 OHM

* PLACE NETWORKS AT RECEIVING END

Project file:peakdetector.cpm		SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND (EN/ICE)	MODULE:clk_receivers	Sheet: 1/1 (Module: / Project: 16/18)
	LAST_MODIFIED=Tue Oct 20 11:33:05 2009	
	Design by: D. VALUCH	PCB by: JMC / DATE: 18/09/09

TERMINATION FOR FPGA CCLK
NEAR TO THE CONFIG ROM



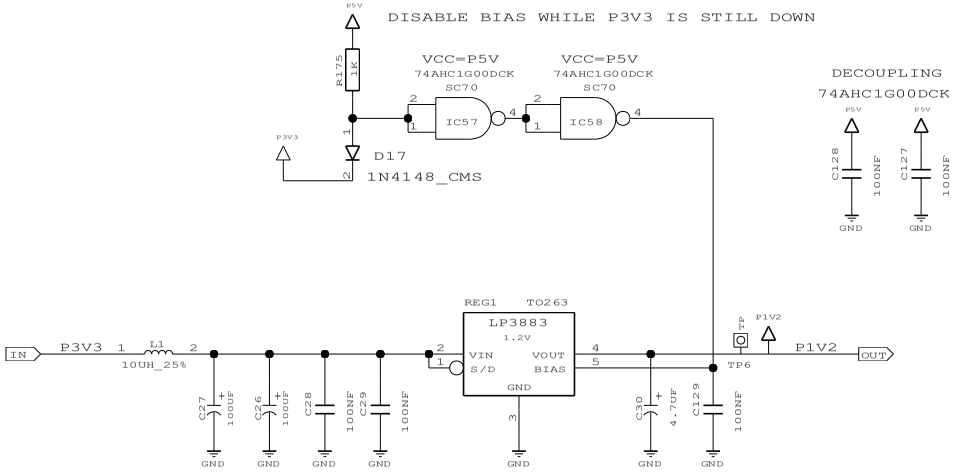
THICK VIOLET LINES ARE 50 OHMS

Project file:peakdetector.cpm		SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT: VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:jtag_cfg_xcf04s	Sheet: Module: 1/1 Project: 17/18
EN/ICE	LAST_MODIFIED=Wed Oct 14 14:19:18 2009	
	Design by: D. VALUCH	PCB by: JMC DATE: 18/09/09

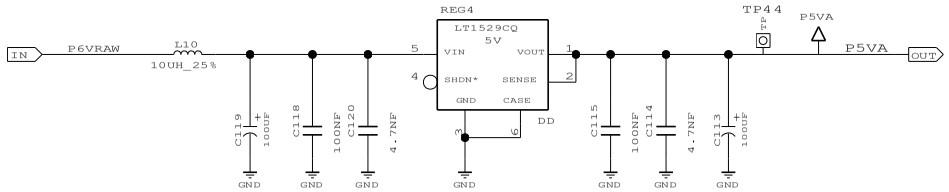
+1.2V/3A CORE VOLTAGE

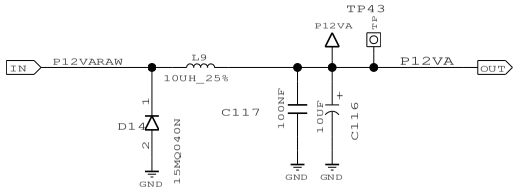
P5V FROM VME

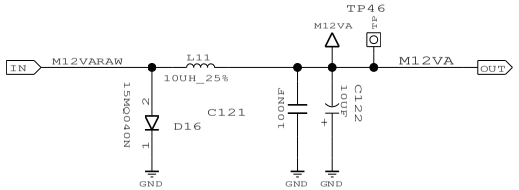
DISABLE BIAS WHILE P3V3 IS STILL DOWN



DECOUPLING
74AHC1G00DCK

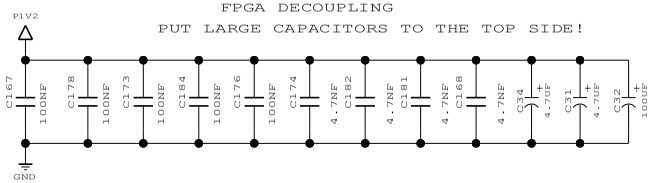




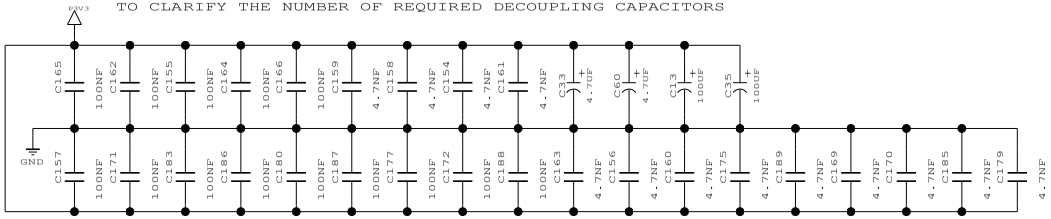


DO NOT ASSEMBLE TPS

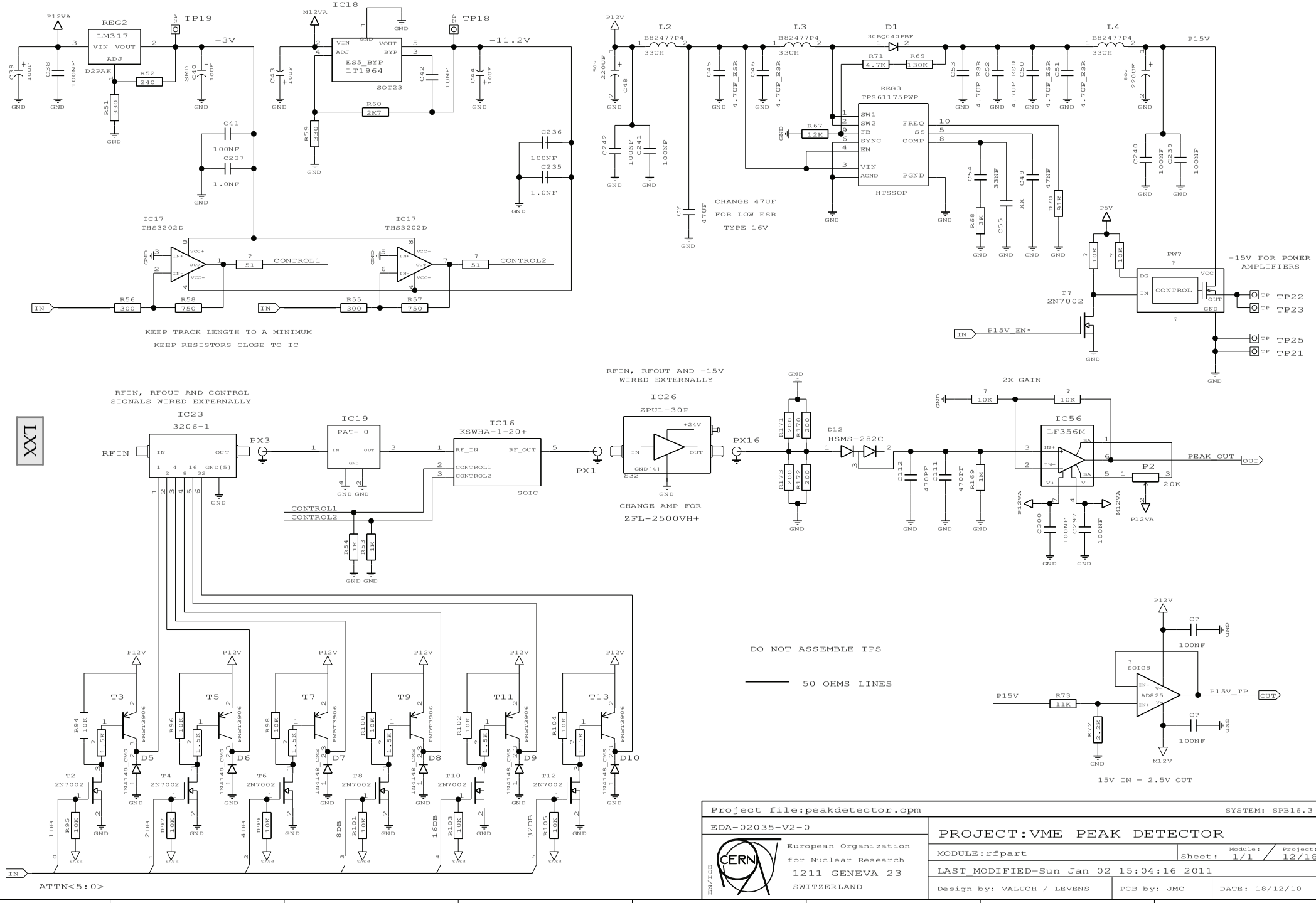
FPGA DECOUPLING
PUT LARGE CAPACITORS TO THE TOP SIDE!



PCB DESIGNER SHOULD CONTACT THE MODULE DESIGNER
TO CLARIFY THE NUMBER OF REQUIRED DECOUPLING CAPACITORS



Project file:peakdetector.cpm		SYSTEM: SPB16.01
EDA-02035-V1-0	PROJECT:VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:power_regulators	Sheet: 1/1 Module: Project: 18/18
EN/ICE	LAST_MODIFIED=Wed Oct 14 14:18:21 2009	
	Design by: D. VALUCH	PCB by: JMC DATE: 18/09/09



Project file:peakdetector.cpm		SYSTEM: SPB16.3
EDA-02035-V2-0	PROJECT: VME PEAK DETECTOR	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:rfpart	Module: 1/1 Project: 12/18
	LAST_MODIFIED=Sun Jan 02 15:04:16 2011	
	Design by: VALUCH / LEVENS	PCB by: JMC DATE: 18/12/10

G VME Peak Detector — Firmware

This appendix contains the firmware for the VME Peak Detector module. Only blocks that are unique to this project are included, and are as follows:

RegCntrl XCVII

Top level of the register controller.

RegRdMux CIV

Read back multiplexer for the read only registers and the control registers.

SampleGating CVI

Interface to mask the ADC data strobe if it should not be recorded to memory.

SerNumValidSeq CVII

Sequencer to initiate silicon serial number reads.

SramCtrl CVIII

Controller for each external SRAM chips.

StretchNHold CIX

Block to take one pulse and both stretch it to a certain duration and to hold it until cleared.

VmeMemMap CX

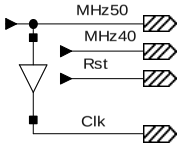
VME register memory map.

VmeMux CXII

Multiplexer between VME access to memory and registers.

This firmware for this module also relies upon some blocks common to all modules (Appendix N). Any blocks not included from any of these sources are recycled from other projects and are not the work of the author.

VME PEAK DETECTOR V2



50MHz Clocked
40MHz Clocked

LXXIII

VME INTERFACE

C0

nVmeBufOE → VmeBufOeN
VmeDir → VmeDir
nVmeAS → VmeAsNA
nVmeDS1 → VmeDs1NA
nVmeDS0 → VmeDs0NA
VmeData (15:0) → VmeData
nVmeWrite → VmeWriteNA
nVmeLWord → VmeLwordNA
nVmeIAck → VmeIackNA
VmeXAmA (5:0) → VmeAmA (4:0)
VmeAmA_2 → VmeAmA
VmeDTAck → vmeDtackN
VmeIntReq (7:1) → VmeIntReqN
VmeAddrIn (23:1) → VmeAddrA
VmeModuleAddr (3:0) → ModuleAddr
Clk → Clk
Rst → RstN

IackOutNA → nVmeIAckOut
IackInNA → nVmeIAckIn
DataFromMem → VmeRdData (15:0)
DataReadDone → VmeRdDone
AddrMem → VmeAddr (19:0)
ReadMem → VmeRdStr
WriteMem → VmeWrStr
DataToMem → VmeWrData (15:0)
DataWriteDone → VmeWrDone
IntProcessed → VmeIntProcessed
UserIntReqN → nVmeUserIntReq
IRQLevelReg → VmeIRQStatID (15:0)
IRQStatusIDReg

Vme_intfce

IRQ INTERFACE

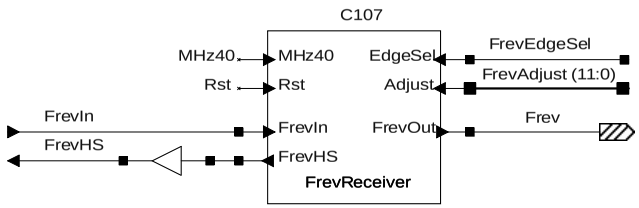
C92

IRQHandler

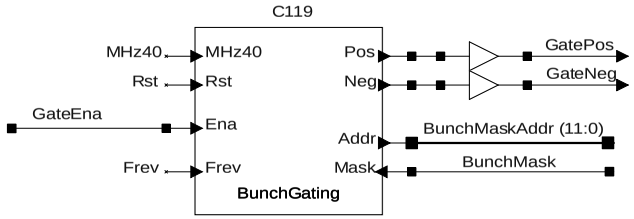
Clk → Clk
Rst → Rst
User
EnableIrq → enableIRQ
VmeInterrupt → TestIRQ
IrqReq → UserIRQ
ClrIrqErr → ClrIRQMem
VmeIRQLvl (2:0) → IRQLevelReg
VME
IntProcessed → UserIntReqN
IRQOverRun → IrqOverRun
IRQTimeOut → IrqTimeOut

Design: PeakDetector:PeakDetector_Top (Sheet 1 of 5)	Date: Fri Dec 03 17:34:11 2010	Page: 1

Frev RECEIVER

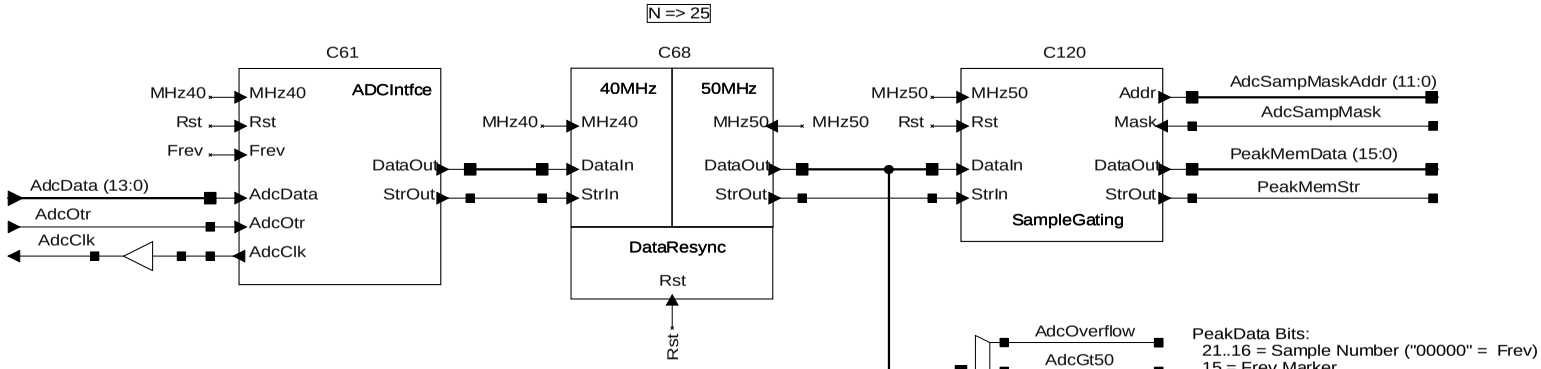


BUNCH GATING INTERFACE

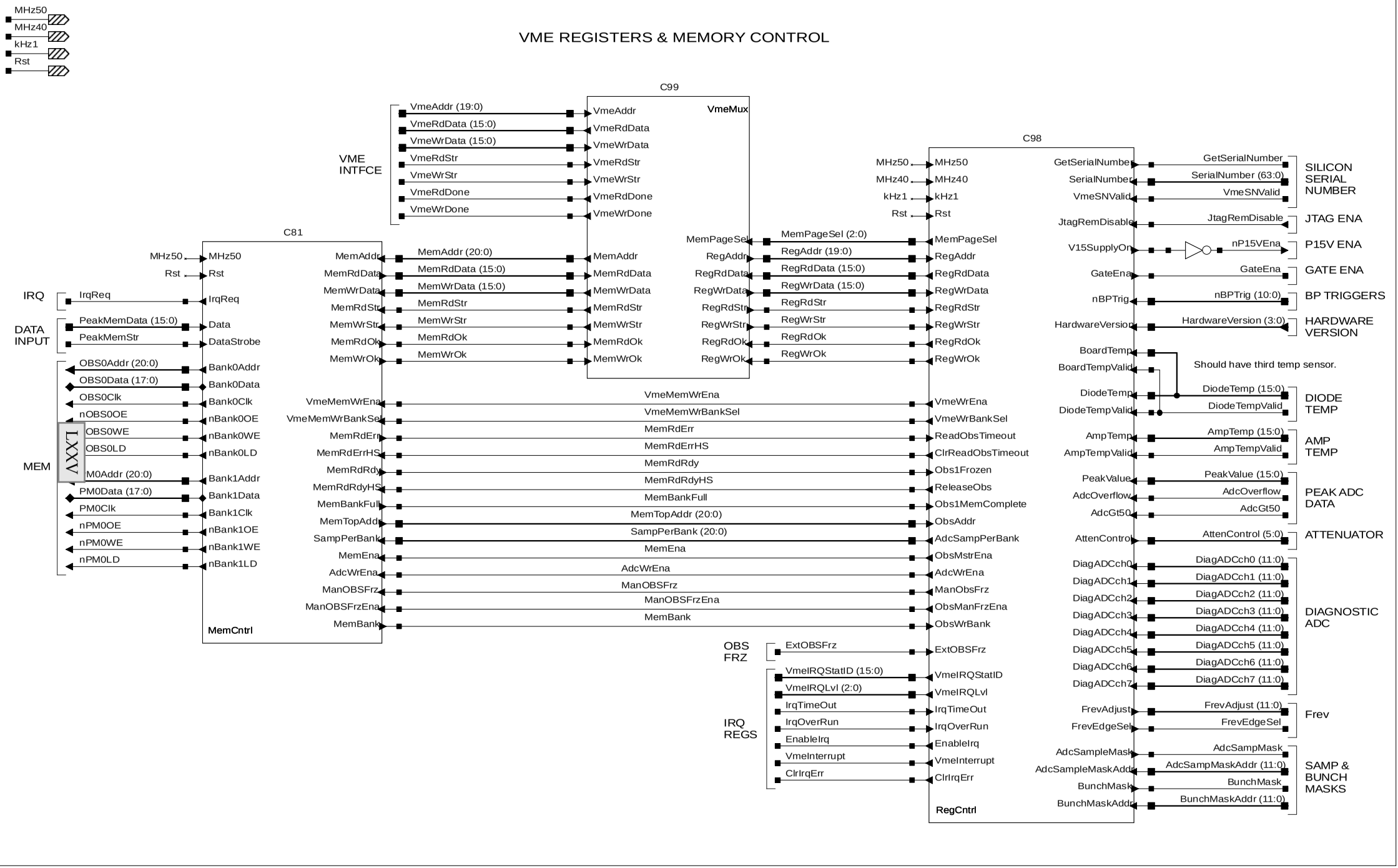


LXXIV

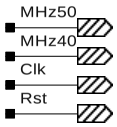
PEAK-VALUE ADC ACQUISITION PATH



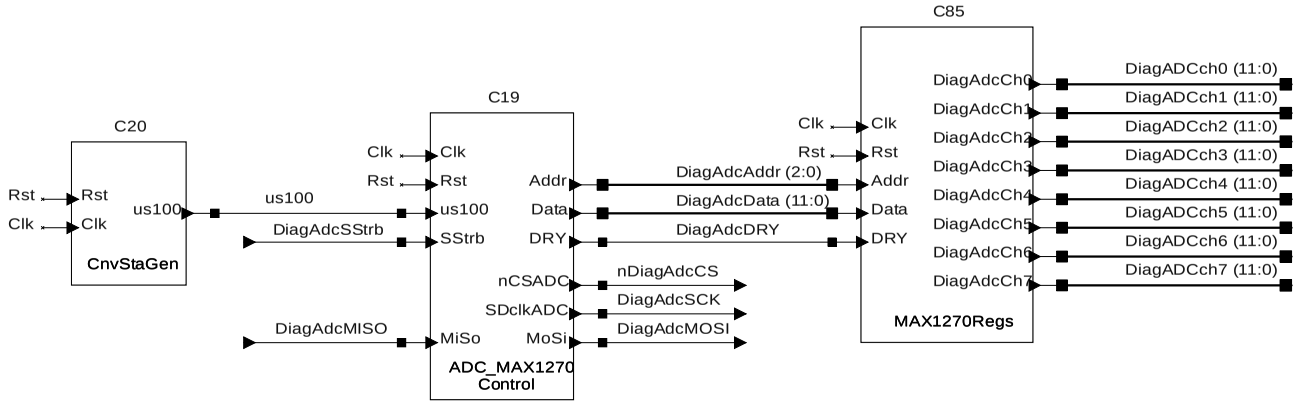

```
AdcOverflow <= PeakData(14);
AdcGt50 <= PeakData(13);
PeakValue(15) <= PeakData(15);
PeakValue(14) <= PeakData(14);
PeakValue(13) <= PeakData(13);
PeakValue(12 downto 0) <= PeakData(12 downto 0);
```



Design: PeakDetector:PeakDetector_Top (Sheet 3 of 5)	Date: Fri Dec 03 17:34:11 2010	Page: 3

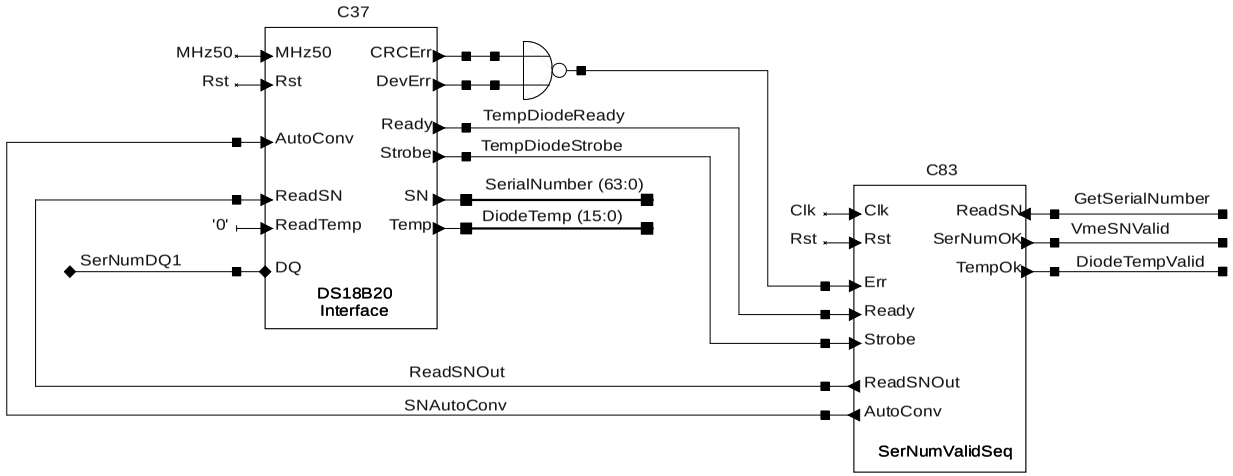


DIAGNOSTIC ADC (MAX1270)

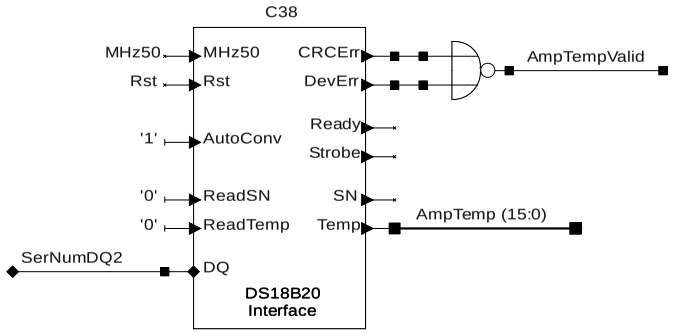


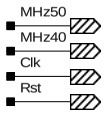
LXXVI

TEMP SENSOR ON DIODES WITH VME SERIAL NUMBER

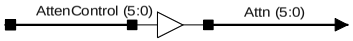


TEMP SENSOR ON RF AMP

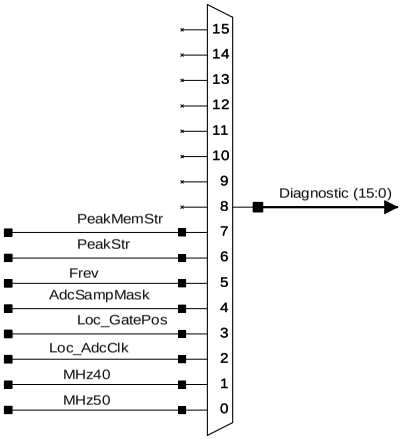




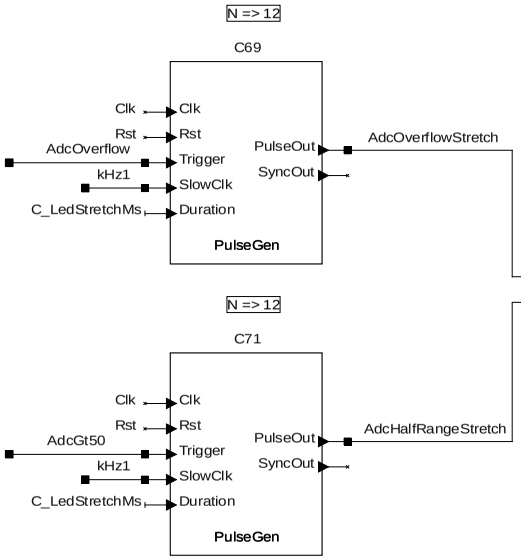
STEP ATTENUATOR INTERFACE



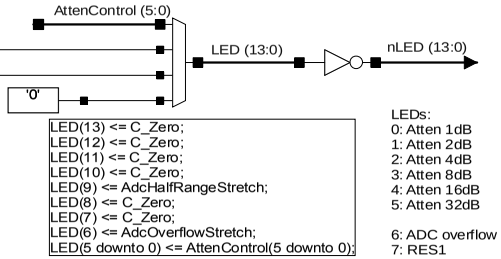
DIAGNOSTICS INTERFACE



LED PULSE STRETCHERS



LED OUTPUT



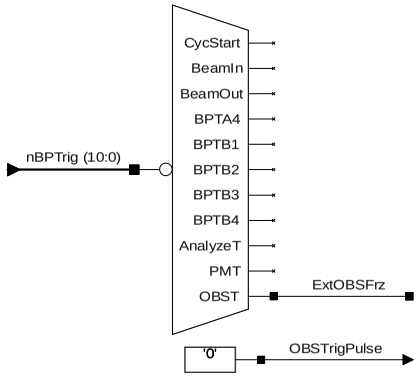
LEDs:
0: Atten 1dB
1: Atten 2dB
2: Atten 4dB
3: Atten 8dB
4: Atten 16dB
5: Atten 32dB

6: ADC overflow
7: RES1

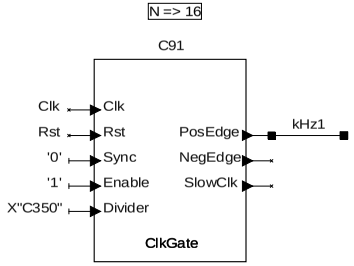
8: ADC half-range [RED]
9: ADC half-range [GRN]
10: ADC half-range [BLU]

11: RES2 [RED]
12: RES2 [GRN]
13: RES2 [BLU]

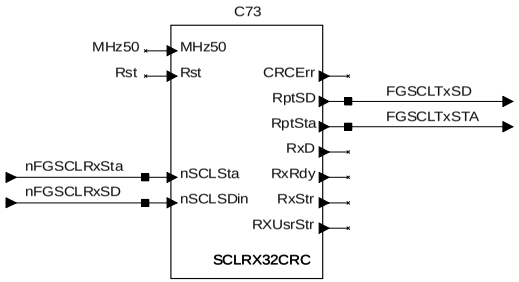
BP TRIGGERS



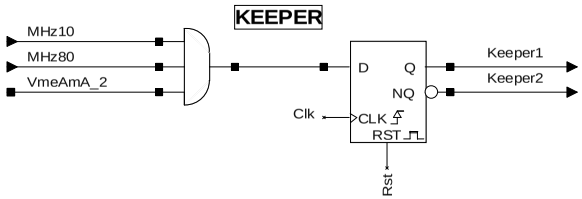
1kHz CLK GEN

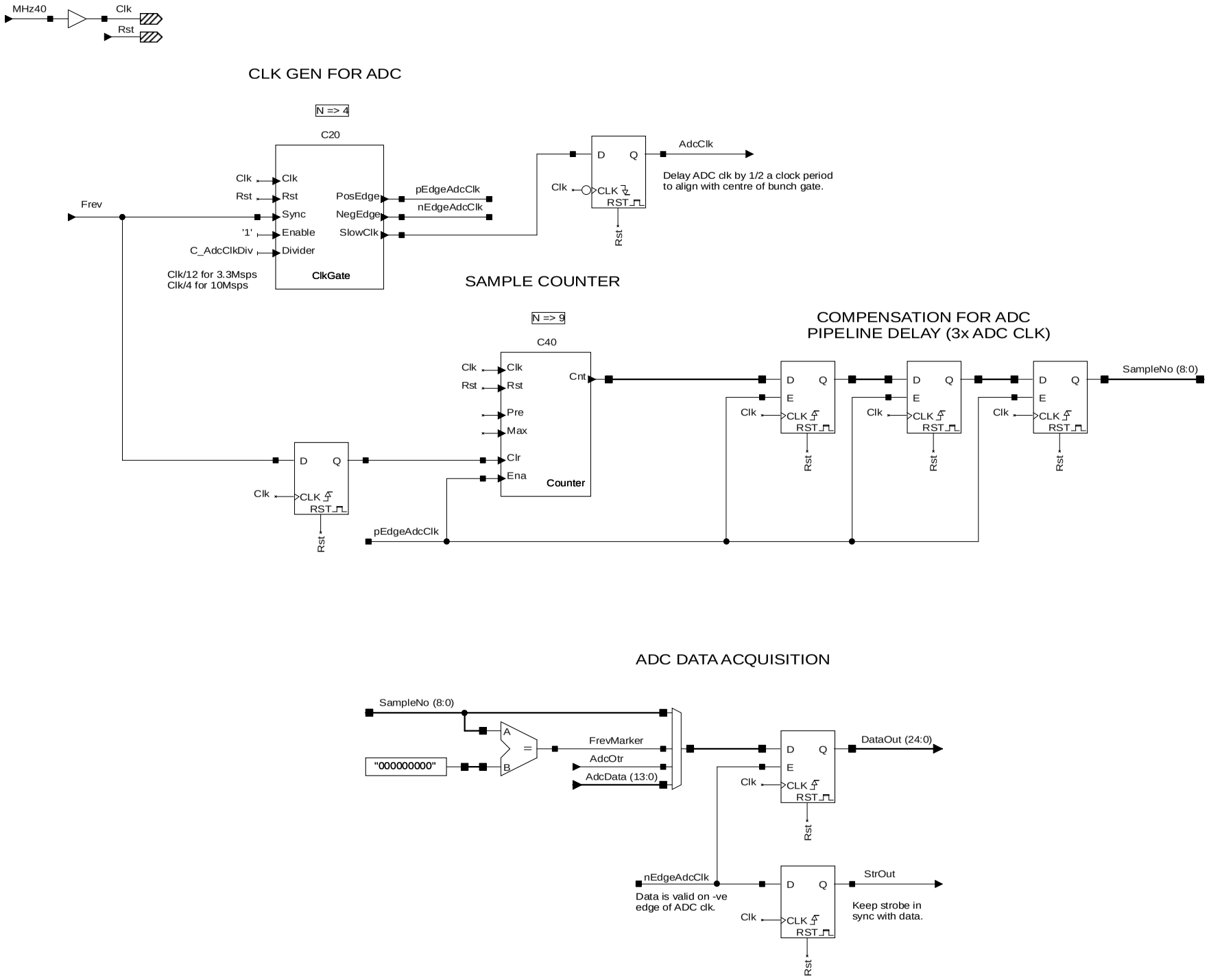


FG-SCL LOOP THRU

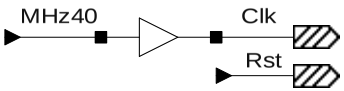


KEEPER

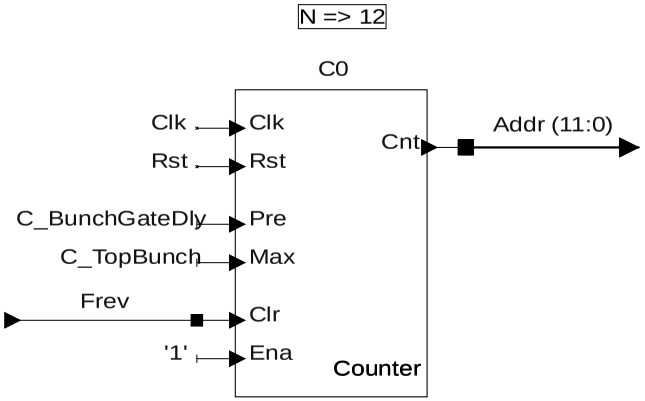




Design: PeakDetector:ADCIntfce	Date: Fri Dec 03 17:30:04 2010	Page: 1

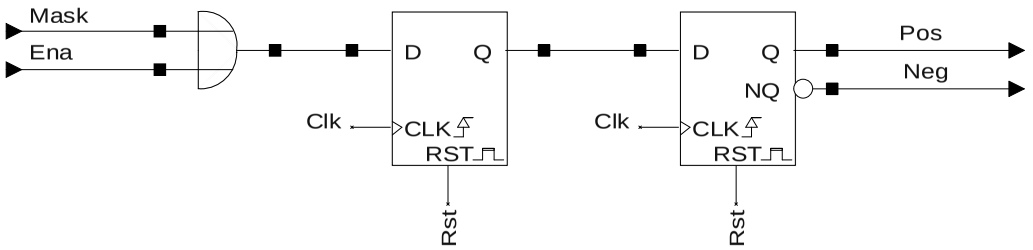


BUNCH COUNTER



Since there is a 1 clk delay between asserting an address and receiving the correct mask from the mask memory this counter is pre-loaded with value 1 on Frev. This means that the correct value is retreived for bucket 1. It also wraps at 3564=0 so that the correct value is retreived for bucket 0 on subsequent turns.

MASKING



Resynchronise to elimitate glitches in the mask when switching words. This necessitates an increase in the preloaded value of the bunch counter by two!

LXXIX

Design: PeakDetector:BunchGating	Date: Fri Dec 03 17:30:39 2010	Page: 1 (1,1)

Truth Table: CRegRdMux

VmeAddr(19 downto 1) & '0'	VmeAddr(19 downto 1) & '0'	CRegRdbk
Reg_Control1		X"00" & Control1
Reg_VmeIRQStatID		VmeIRQStatID
Reg_VmeIRQLvl		VmeIRQLvl
Reg_MemControl1		X"00" & MemControl1
Reg_MemControl2		MemControl2
Reg_MemControl3		X"00" & MemControl3
Reg_Control2		X"00" & Control2
Reg_AttenControl		AttenControl
Reg_FrevAdjust		FrevAdjust
Reg_DiodeMinTemp		DiodeMinTemp
Reg_DiodeMaxTemp		DiodeMaxTemp
Reg_AmpMinTemp		AmpMinTemp
Reg_AmpMaxTemp		AmpMaxTemp
Reg_AdcSampPerBank_0		AdcSampPerBank0
Reg_AdcSampPerBank_1		AdcSampPerBank1
Reg_SysControl		X"00" & SysControl
Reg_TestControl		X"00" & TestControl
>= Reg_AdcSampleMask	<= Reg_AdcSampleMask_End	AdcSampMask
>= Reg_BunchMask	<= Reg_BunchMask_End	BunchMask
		C_VmeNullRdbk

Packages Used
ieee.STD_LOGIC_1164
work.VmeMemMap
ieee.NUMERIC_STD

Interface
VmeAddr : in std_logic_vector (19 downto 0);
CRegRdbk : out std_logic_vector (15 downto 0);
Control1 : in std_logic_vector (7 downto 0);
VmeIRQStatID : in std_logic_vector (15 downto 0);
VmeIRQLvl : in std_logic_vector (15 downto 0);
MemControl1 : in std_logic_vector (7 downto 0);
MemControl2 : in std_logic_vector (15 downto 0);
MemControl3 : in std_logic_vector (7 downto 0);
Control2 : in std_logic_vector (7 downto 0);
AttenControl : in std_logic_vector (15 downto 0);
FrevAdjust : in std_logic_vector (15 downto 0);
DiodeMinTemp : in std_logic_vector (15 downto 0);
DiodeMaxTemp : in std_logic_vector (15 downto 0);
AmpMinTemp : in std_logic_vector (15 downto 0);
AmpMaxTemp : in std_logic_vector (15 downto 0);
AdcSampPerBank0 : in std_logic_vector (15 downto 0);
AdcSampPerBank1 : in std_logic_vector (15 downto 0);
SysControl : in std_logic_vector (7 downto 0);
TestControl : in std_logic_vector (7 downto 0);
AdcSampMask : in std_logic_vector (15 downto 0);
BunchMask : in std_logic_vector (15 downto 0);

Truth Table: CRegWrSel (1/4)

VmeAddr(19 downto 1) & '0'	VmeAddr(19 downto 1) & '0'	CRegRdWrOk	Control1_Sel	VmeIRQStatID_Sel	VmeIRQLvl_Sel	MemControl1_Sel
Reg_Control1		'1'	'1'	'0'	'0'	'0'
Reg_VmeIRQStatID		'1'	'0'	'1'	'0'	'0'
Reg_VmeIRQLvl		'1'	'0'	'0'	'1'	'0'
Reg_MemControl1		'1'	'0'	'0'	'0'	'1'
Reg_MemControl2		'1'	'0'	'0'	'0'	'0'
Reg_MemControl3		'1'	'0'	'0'	'0'	'0'
Reg_Control2		'1'	'0'	'0'	'0'	'0'
Reg_AttenControl		'1'	'0'	'0'	'0'	'0'
Reg_FrevAdjust		'1'	'0'	'0'	'0'	'0'
Reg_DiodeMinTemp		'1'	'0'	'0'	'0'	'0'
Reg_DiodeMaxTemp		'1'	'0'	'0'	'0'	'0'
Reg_AmpMinTemp		'1'	'0'	'0'	'0'	'0'
Reg_AmpMaxTemp		'1'	'0'	'0'	'0'	'0'
Reg_AdcSampPerBank_0		'1'	'0'	'0'	'0'	'0'
Reg_AdcSampPerBank_1		'1'	'0'	'0'	'0'	'0'
Reg_SysControl		'1'	'0'	'0'	'0'	'0'
Reg_TestControl		'1'	'0'	'0'	'0'	'0'
>= Reg_AdcSampleMask	<= Reg_AdcSampleMask_End	'1'	'0'	'0'	'0'	'0'
>= Reg_BunchMask	<= Reg_BunchMask_End	'1'	'0'	'0'	'0'	'0'
		'0'	'0'	'0'	'0'	'0'

Truth Table: CRegWrSel (2/4)

MemControl2_Sel	MemControl3_Sel	Control2_Sel	AttenControl_Sel	FrevAdjust_Sel	DiodeMinTemp_Sel	DiodeMaxTemp_Sel
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'1'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'1'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'1'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'1'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'1'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'1'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'1'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'

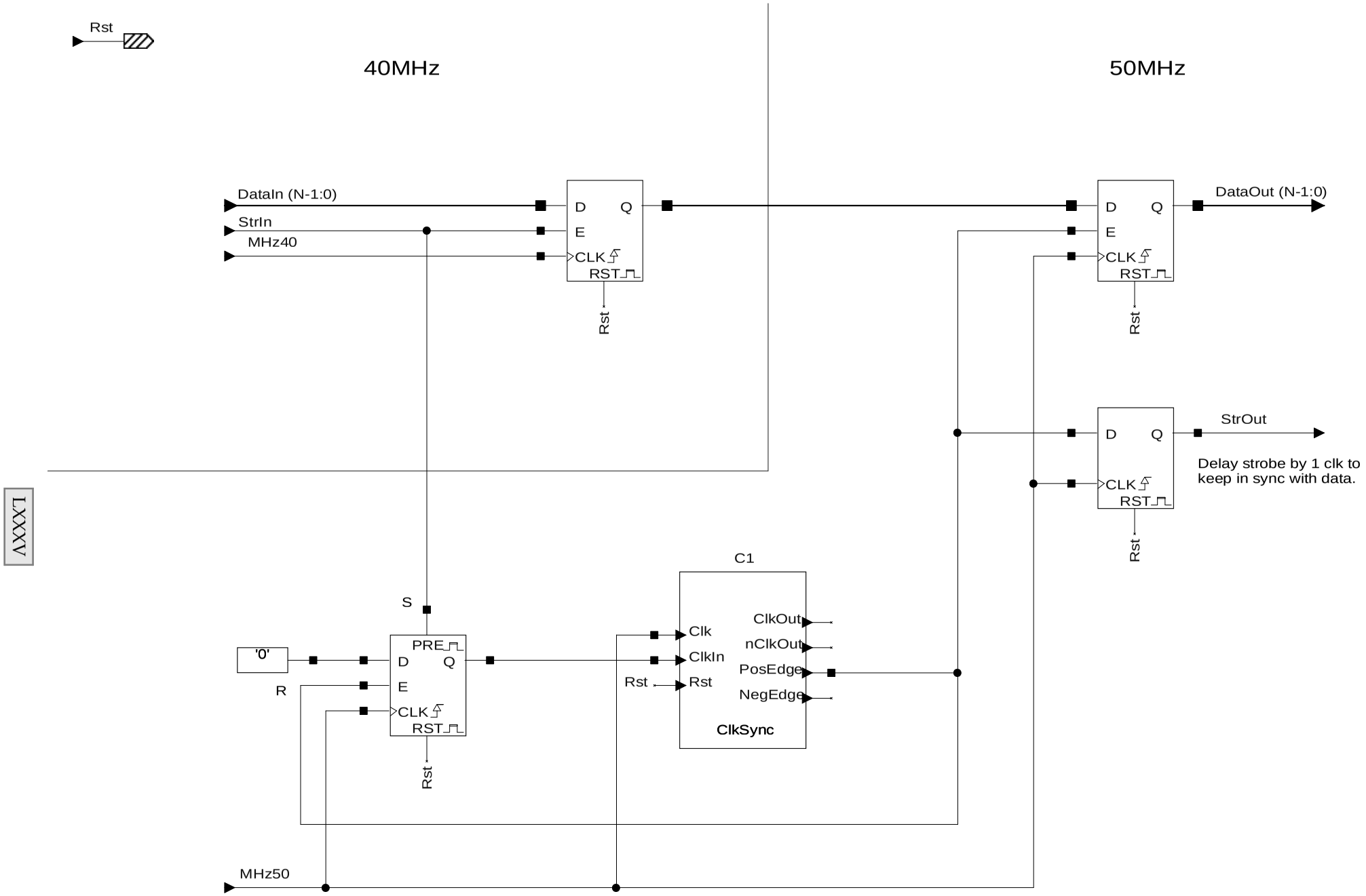
Truth Table: CRegWrSel (3/4)

AmpMinTemp_Sel	AmpMaxTemp_Sel	AdcSampPerBank0_Sel	AdcSampPerBank1_Sel	SysControl_Sel	TestControl_Sel	AdcSampMask_Sel
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'1'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'1'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'1'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'1'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'1'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'1'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'1'
'0'	'0'	'0'	'0'	'0'	'0'	'0'
'0'	'0'	'0'	'0'	'0'	'0'	'0'

Truth Table: CRegWrSel (4/4)

Packages Used
ieee.STD_LOGIC_1164
work.VmeMemMap

Interface
VmeAddr : in std_logic_vector (19 downto 0);
Control1_Sel : out std_logic ;
VmeIRQStatID_Sel : out std_logic ;
VmeIRQLvl_Sel : out std_logic ;
MemControl1_Sel : out std_logic ;
MemControl2_Sel : out std_logic ;
MemControl3_Sel : out std_logic ;
Control2_Sel : out std_logic ;
AttenControl_Sel : out std_logic ;
FrevAdjust_Sel : out std_logic ;
DiodeMinTemp_Sel : out std_logic ;
DiodeMaxTemp_Sel : out std_logic ;
AmpMinTemp_Sel : out std_logic ;
AmpMaxTemp_Sel : out std_logic ;
AdcSampPerBank1_Sel : out std_logic ;
AdcSampPerBank0_Sel : out std_logic ;
SysControl_Sel : out std_logic ;
TestControl_Sel : out std_logic ;
AdcSampMask_Sel : out std_logic ;
BunchMask_Sel : out std_logic ;
CRegRdWrOk : out std_logic ;



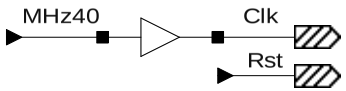
RESYNC FROM 40MHz TO 50MHz CLOCK DOMAINS

Design: PeakDetector:DataResync	Date: Fri Dec 03 17:32:21 2010	Page: 1

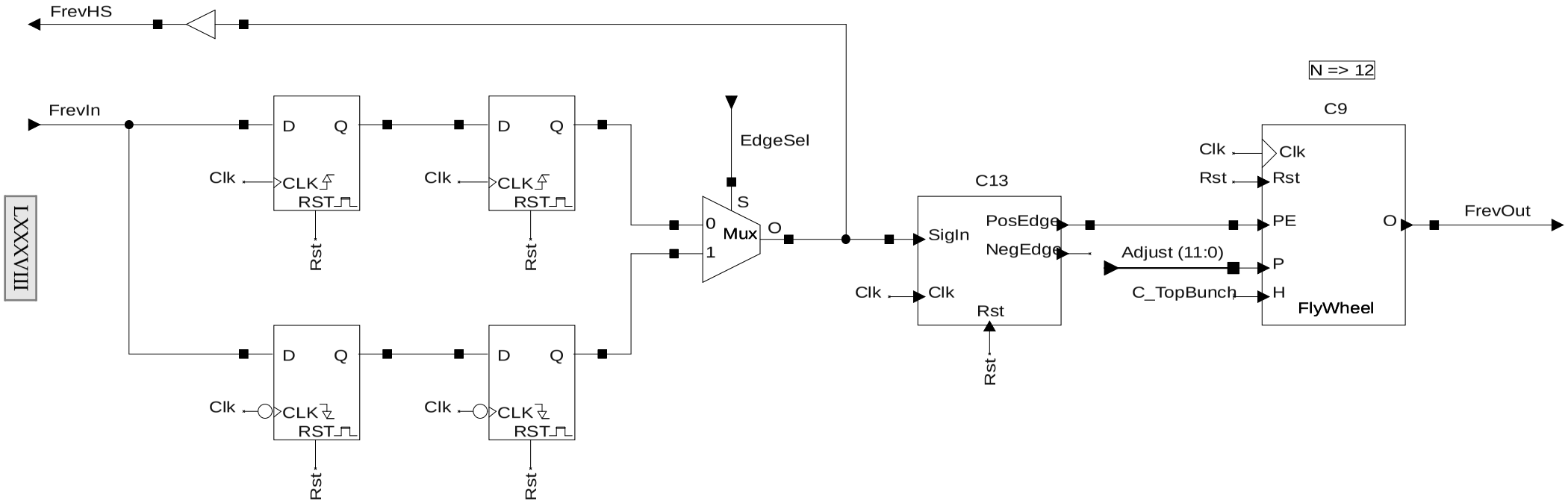
G.7 DPRam256x16b

```
----------------------------------------
----------------------------------------
-- Date        : Wed Oct 27 09:52:26 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Dual port, 256x16b RAM.
--               Single input port (A).
--               Dual output ports (A,B) with separate clocks.
--
-- Changelog   : 20101105
--                 Fixed style so that Synplify correctly generates a dual port
--                 RAM block and not two single port blocks.
--
--               20101027
--                 Initial Revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.std_logic_unsigned.all;

entity  DPRam256x16b  is
  port (
    CLKA,
    WEA,
    ENA   : in  std_logic;

    ADDRA : in  std_logic_vector(7 downto 0);
    DIA   : in  std_logic_vector(15 downto 0);
    DOA   : out std_logic_vector(15 downto 0);

    CLKB  : in  std_logic;

    ADDRB : in  std_logic_vector(7 downto 0);
    DOB   : out std_logic_vector(15 downto 0)
  );
end;

architecture  V1  of  DPRam256x16b  is

-- RAM data type. Array of 256 words.
subtype ramword is std_logic_vector(15 downto 0);
type ram_t is array (0 to 255) of ramword;

-- Instance of RAM array.
signal TheRam : ram_t;

-- Registers to hold address for ports.
signal Loc_ADDRA : std_logic_vector(7 downto 0);
signal Loc_ADDRB : std_logic_vector(7 downto 0);

-- Tell Synplify not to generate RW conflict resolution logic. The RAM will be
-- double banked, so this is not needed.
attribute syn_ramstyle : string;
attribute syn_ramstyle of TheRam : signal is "no_rw_check";

begin
  -- PORT A (Read/Write).
  process (CLKA) begin
    if rising_edge(CLKA) then
      Loc_ADDRA <= ADDRA;
      if ENA = '1' and WEA = '1' then
        TheRam(conv_integer(ADDRA)) <= DIA;
      end if;
    end if;
```

```
  end process;
  DOA <= TheRam(conv_integer(Loc_ADDRA));

  -- PORT B (Read Only).
  process (CLKB) begin
    if rising_edge(CLKB) then
      Loc_ADDRB <= ADDRB;
    end if;
  end process;
  DOB <= TheRam(conv_integer(Loc_ADDRB));
end;

--EOF
```



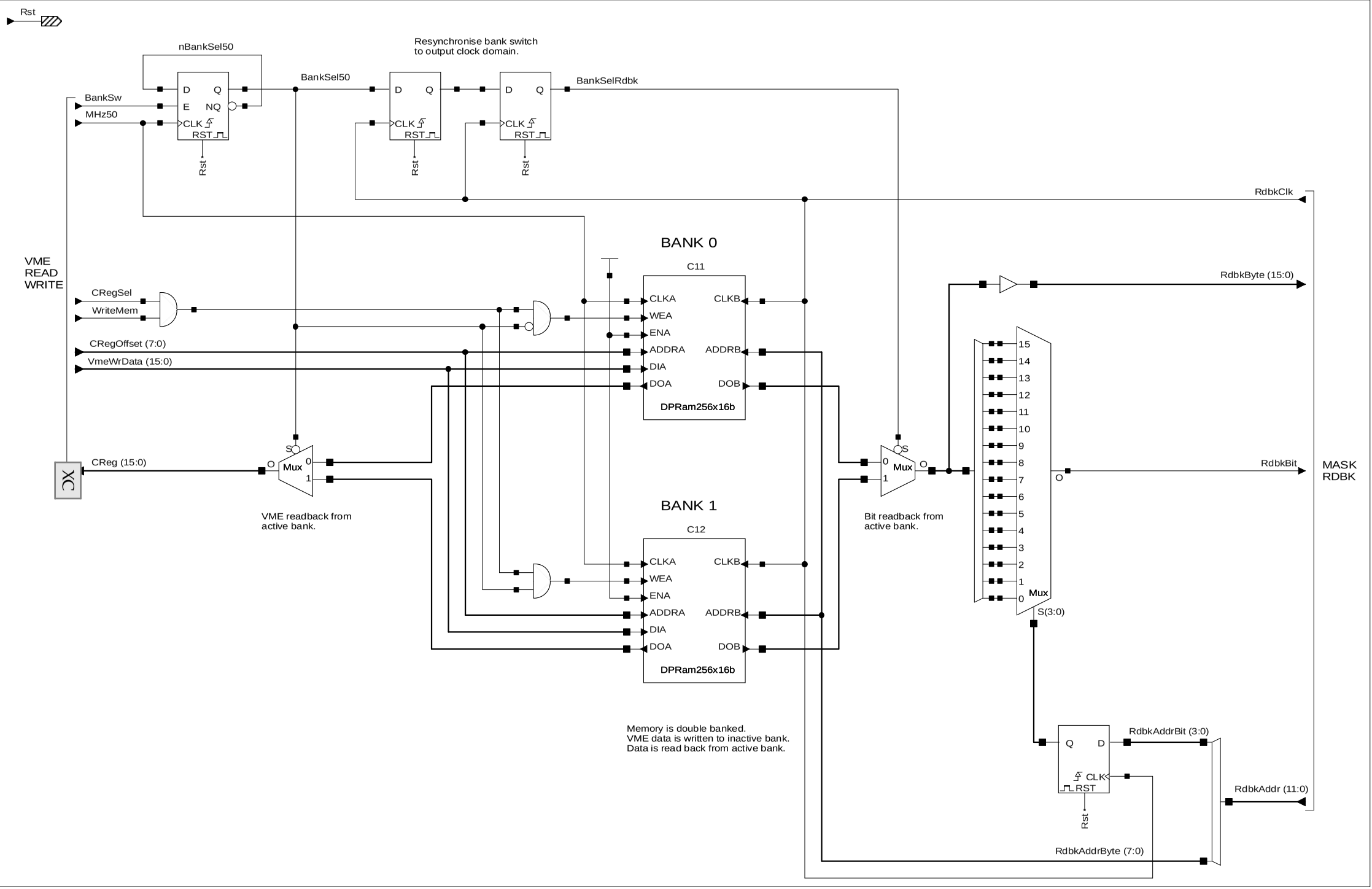
Frev RECEIVER



Design: PeakDetector:FrevReceiver	Date: Mon Dec 13 14:55:44 2010	Page: 1 (1,1)

G.9 MagicConstants

```
------------------------------------------
------------------------------------------
-- Date        : Thu Oct 28 09:23:15 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Magic constants that make everything work.
--
------------------------------------------
------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package MagicConstants  is

  constant C_NoBunches    : std_logic_vector(11 downto 0) := X"DEC"; -- 3564
  constant C_TopBunch     : std_logic_vector(11 downto 0) := X"DEB"; -- 3563
  constant C_BunchGateDly : std_logic_vector(11 downto 0) := X"003";

  constant C_LedStretchMs : std_logic_vector(11 downto 0) := X"064"; -- 100ms
  constant C_VmeStretchMs : std_logic_vector(11 downto 0) := X"535"; -- 1.34s

  constant C_AdcClkDiv    : std_logic_vector(3 downto 0)  := X"C";   -- 12

  constant C_P15VMin      : std_logic_vector(11 downto 0) := X"7D0"; -- 14.65V
  constant C_P15VMax      : std_logic_vector(11 downto 0) := X"830"; -- 15.35V

end;

--EOF
```



Design: PeakDetector:MaskRam	Date: Fri Dec 03 17:33:19 2010	Page: 1

G.11 MemAddrCnt

```
----------------------------------------
----------------------------------------
-- Date        : Tue Oct 19 15:27:36 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : SRAM memory address counter with bank switching.
--
-- Changelog   : 20101123
--                 Changed check from Cnt = SampPerBank to Cnt >= ... so that
--                 if SampPerBank is decreased the switch will happen at once.
--
--               20101122
--                 Bugfix for inability to switch banks when manual freeze was
--                 enabled and the bank was full. MemWrStr is then blocked by
--                 nMemWrBlk so no more MemWrOk strobes are received. Changed
--                 guard in this case to be ManFrzSR, so when next freeze is
--                 triggered, the switch will happen.
--
--               20101119
--                 Addition of the ability to manually freeze the bank. When
--                 ManFrzEna is high, the bank switch will not automatically
--                 occur once SampPerBank is reached. Instead, it will happen
--                 on the sample after ManFrz is received. In this case, the
--                 bank will keep recording samples until it is full. Once this
--                 occurs, data will no longer be stored and the MemBankFull
--                 flag will be set. As with the automatic switching mode, if
--                 a manual freeze is received before the read bank has been
--                 released, it will cause the current bank to be rewritten
--                 from the beginning and the error flag to be set.
--
--               20101019
--                 Inital revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

library CommonVisual;
use CommonVisual.Util.all;

entity  MemAddrCnt  is
  port (
    Clk,
    Rst,
    MemRdRdy,
    MemWrOk,
    ManFrz,
    ManFrzEna       : in  std_logic;

    MemRdRdySet,
    MemRdErrSet,
    MemWrBank,
    MemBankFull,
    nMemWrBlk       : out std_logic;

    SampPerBank     : in  std_logic_vector(20 downto 0);

    MemWrAddr,
    MemTopAddr      : out std_logic_vector(20 downto 0)
  );
end;

architecture  V1  of  MemAddrCnt  is
```

```
signal    Cnt       : unsigned(20 downto 0);
signal    BSel      : std_logic;

signal    Frz       : std_logic;
signal    Samp      : std_logic;
signal    ManFrzSR  : std_logic;

signal    Full      : std_logic;

constant  C_Zero    : unsigned(20 downto 0) := (others => '0');

begin
  -- Mux for freeze (manual or automatic when SampPerBank is reached).
  Frz  <= ManFrzSR when ManFrzEna = '1' else bool_to_stdlogic(Cnt >= unsigned(SampPerBank));
  Samp <= ManFrzSR when (ManFrzEna = '1' and Full = '1') else MemWrOk;

  -- Main counter process.
  process (Clk, Rst) begin
    if Rst = '1' then
      MemRdRdySet <= '0';
      MemRdErrSet <= '0';
      Cnt       <= C_Zero;
      Full      <= '0';
      MemTopAddr  <= std_logic_vector(C_Zero);
      BSel      <= '0';
      ManFrzSR    <= '0';
    elsif rising_edge(Clk) then
      -- Clear by default, make these pulses 1 clk.
      MemRdRdySet <= '0';
      MemRdErrSet <= '0';

      -- SRFF for holding ManFrz until the next MemWrOk.
      if ManFrz = '1' then
        ManFrzSR <= '1';
      end if;

      if Samp = '1' then
        -- Freeze received, toggle banks if read bank is idle, if not
        -- idle then set error flag and rewrite current bank.
        if Frz = '1' then
          ManFrzSR <= '0';

          if MemRdRdy = '0' then
            MemTopAddr  <= std_logic_vector(Cnt + 1);
            BSel      <= BSel xor '1';
            MemRdRdySet <= '1';
          else
            MemRdErrSet <= '1';
          end if;

          Cnt  <= C_Zero;
          Full <= '0';
        -- Else, increment count or toggle bank full flag if at
        -- overflow point.
        else
          if Cnt + 1 = C_Zero then
            Full <= '1';
          else
            Full <= '0';
            Cnt  <= Cnt + 1;
          end if;
        end if;
      end if;
    end if;
  end process;

  MemWrAddr   <= std_logic_vector(Cnt);
  MemWrBank   <= BSel;

  MemBankFull <= Full;
  nMemWrBlk   <= not Full;
end;
```

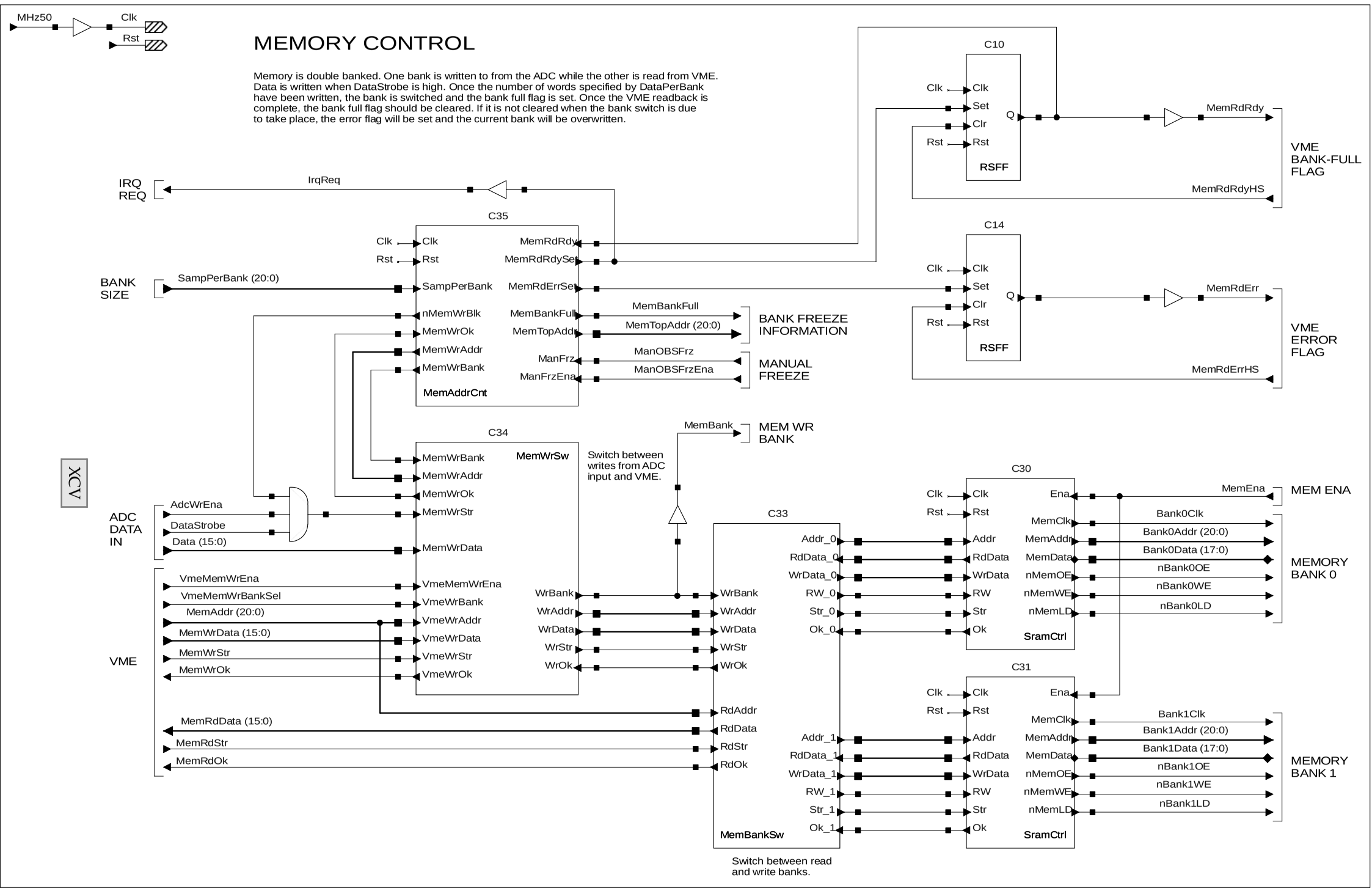
```
--EOF
```

Truth Table: MemBankSw

WrBank	Addr_0	WrData_0	RW_0	Str_0	Addr_1	WrData_1	RW_1	Str_1	WrOk	RdOk
'0'	WrAddr	"00" & WrData	'0'	WrStr	RdAddr	(others => '0')	'1'	RdStr	Ok_0	Ok_1
	RdAddr	(others => '0')	'1'	RdStr	WrAddr	"00" & WrData	'0'	WrStr	Ok_1	Ok_0

Packages Used
ieee.STD_LOGIC_1164

Interface
WrBank : in std_logic ;
Addr_0 : out std_logic_vector (20 downto 0);
Addr_1 : out std_logic_vector (20 downto 0);
WrAddr : in std_logic_vector (20 downto 0);
RdAddr : in std_logic_vector (20 downto 0);
WrData_0 : out std_logic_vector (17 downto 0);
WrData_1 : out std_logic_vector (17 downto 0);
WrData : in std_logic_vector (15 downto 0);
RdData_0 : in std_logic_vector (17 downto 0);
RdData_1 : in std_logic_vector (17 downto 0);
RdData : out std_logic_vector (15 downto 0);
RW_0 : out std_logic ;
RW_1 : out std_logic ;
Str_0 : out std_logic ;
Str_1 : out std_logic ;
RdStr : in std_logic ;
WrStr : in std_logic ;
Ok_0 : in std_logic ;
Ok_1 : in std_logic ;
RdOk : out std_logic ;
WrOk : out std_logic ;



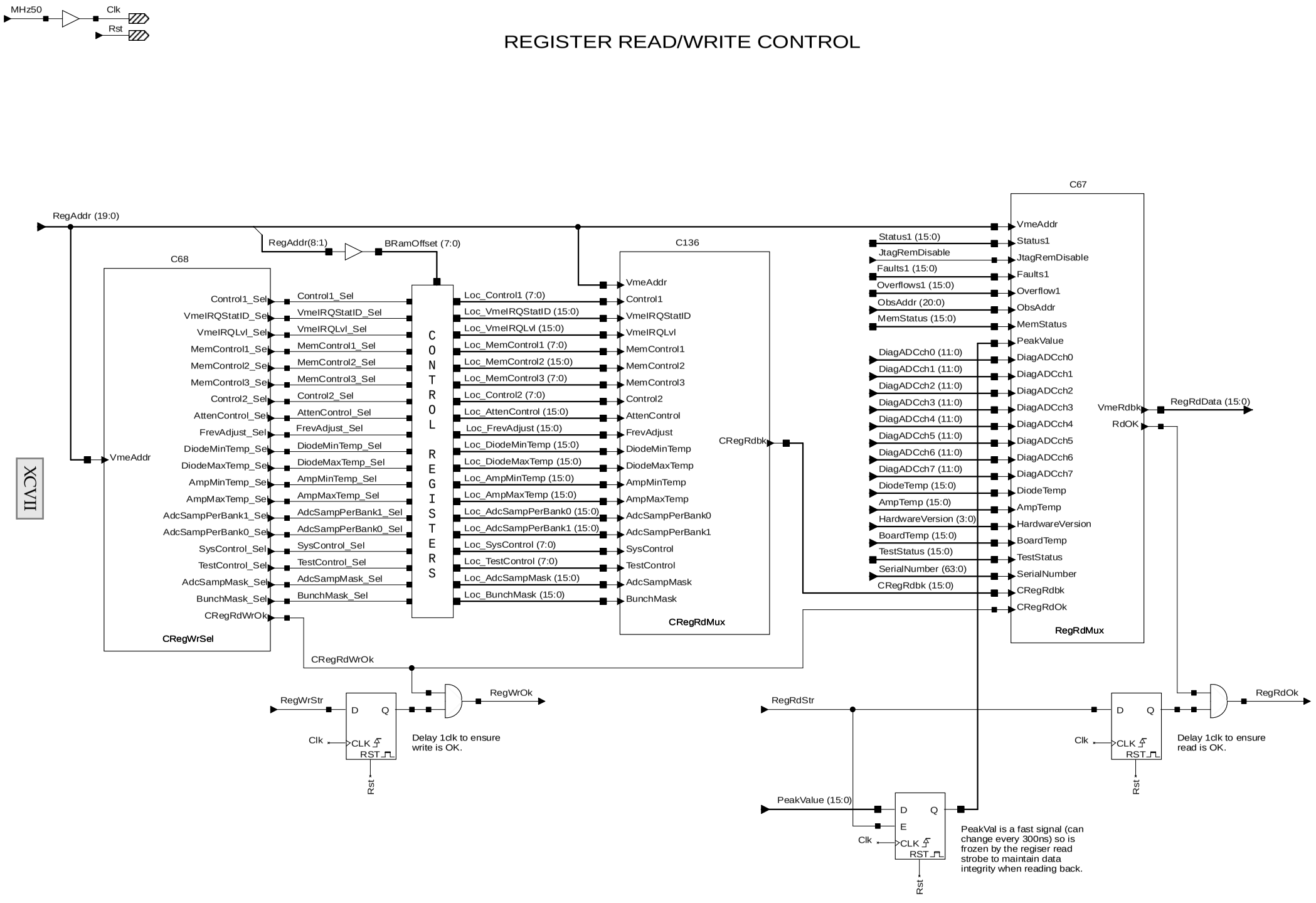
Design: PeakDetector:MemCntrl	Date: Fri Dec 03 17:33:42 2010	Page: 1

Truth Table: MemWrSw

VmeMemWrEna	WrBank	WrAddr	WrData	WrStr	VmeWrOk	MemWrOk
'0'	MemWrBank	MemWrAddr	MemWrData	MemWrStr	'0'	WrOk
	VmeWrBank	VmeWrAddr	VmeWrData	VmeWrStr	WrOk	'0'

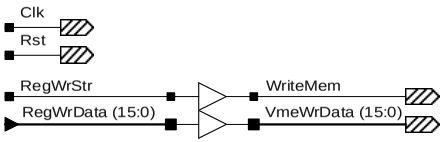
Packages Used
ieee.STD_LOGIC_1164

Interface
VmeMemWrEna : in std_logic ;
MemWrBank : in std_logic ;
VmeWrBank : in std_logic ;
MemWrAddr : in std_logic_vector (20 downto 0);
VmeWrAddr : in std_logic_vector (20 downto 0);
MemWrData : in std_logic_vector (15 downto 0);
VmeWrData : in std_logic_vector (15 downto 0);
MemWrStr : in std_logic ;
VmeWrStr : in std_logic ;
MemWrOk : out std_logic ;
VmeWrOk : out std_logic ;
WrBank : out std_logic ;
WrAddr : out std_logic_vector (20 downto 0);
WrData : out std_logic_vector (15 downto 0);
WrStr : out std_logic ;
WrOk : in std_logic ;

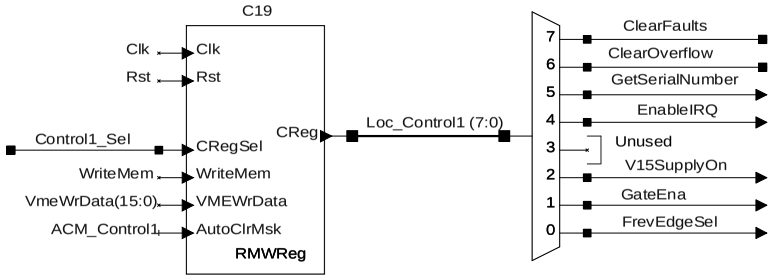


Design: PeakDetector:RegCntrl (Sheet 1 of 7)	Date: Fri Dec 03 17:34:42 2010	Page: 1

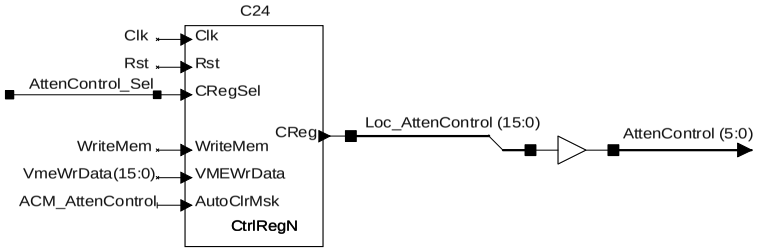
REGISTERS #1



control1

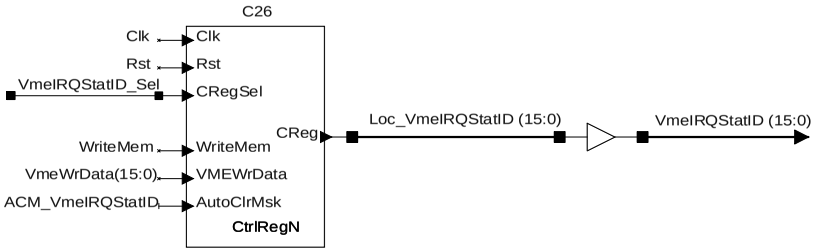


attenControl

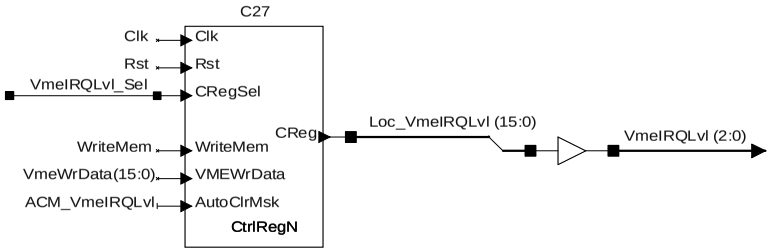


XCVIII

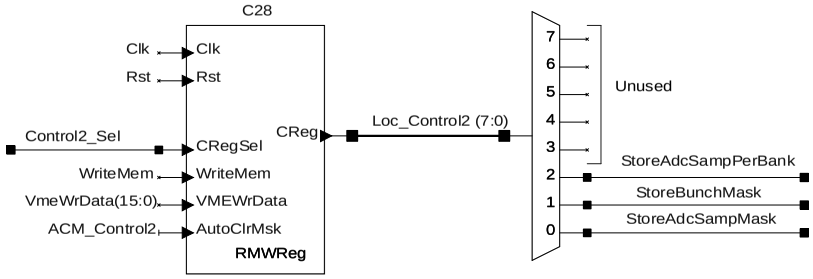
vmeIRQStatID



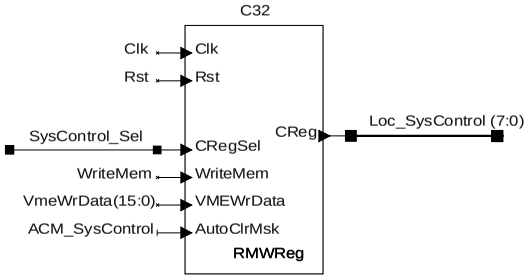
vmeIRQLvl

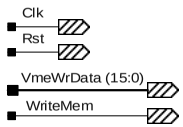


control2



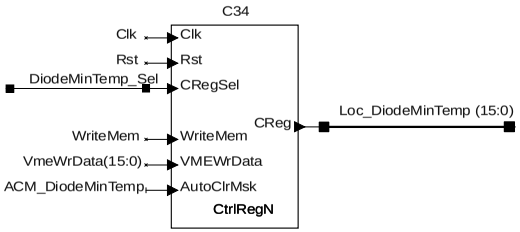
sysControl



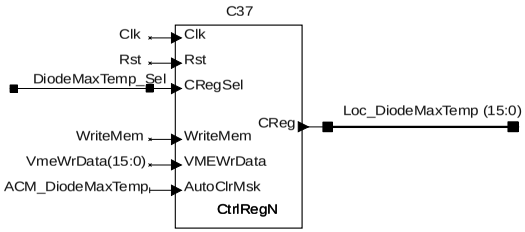


REGISTERS #2

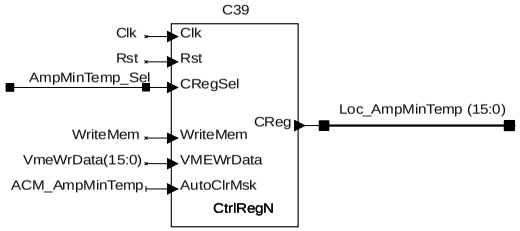
diodeMinTemp



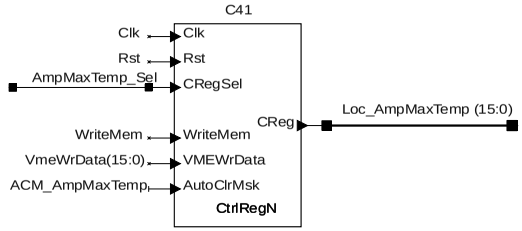
diodeMaxTemp



ampMinTemp

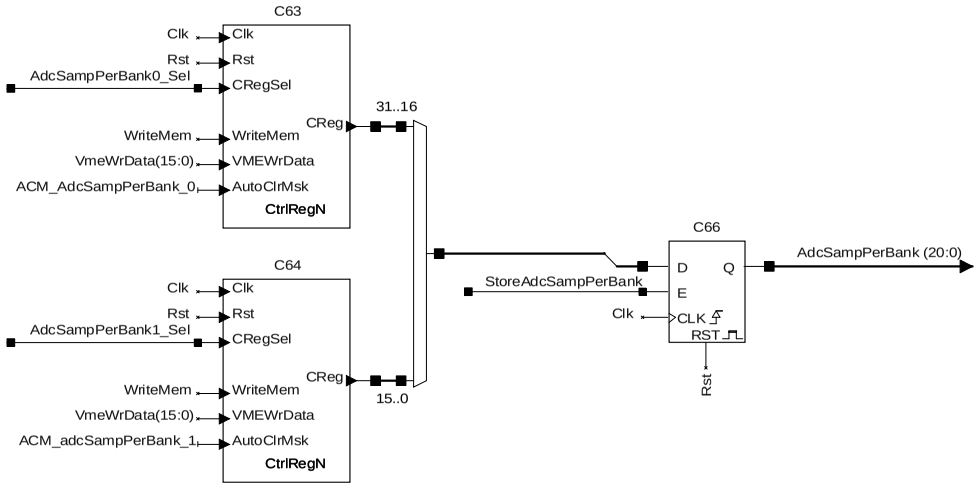


ampMaxTemp

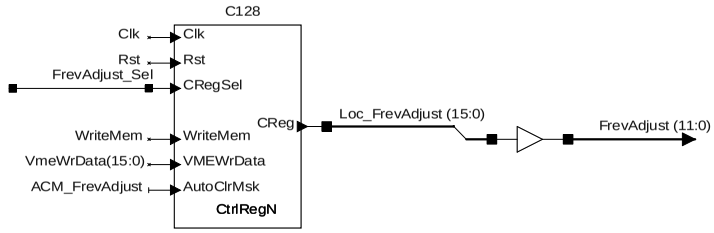


XCIX

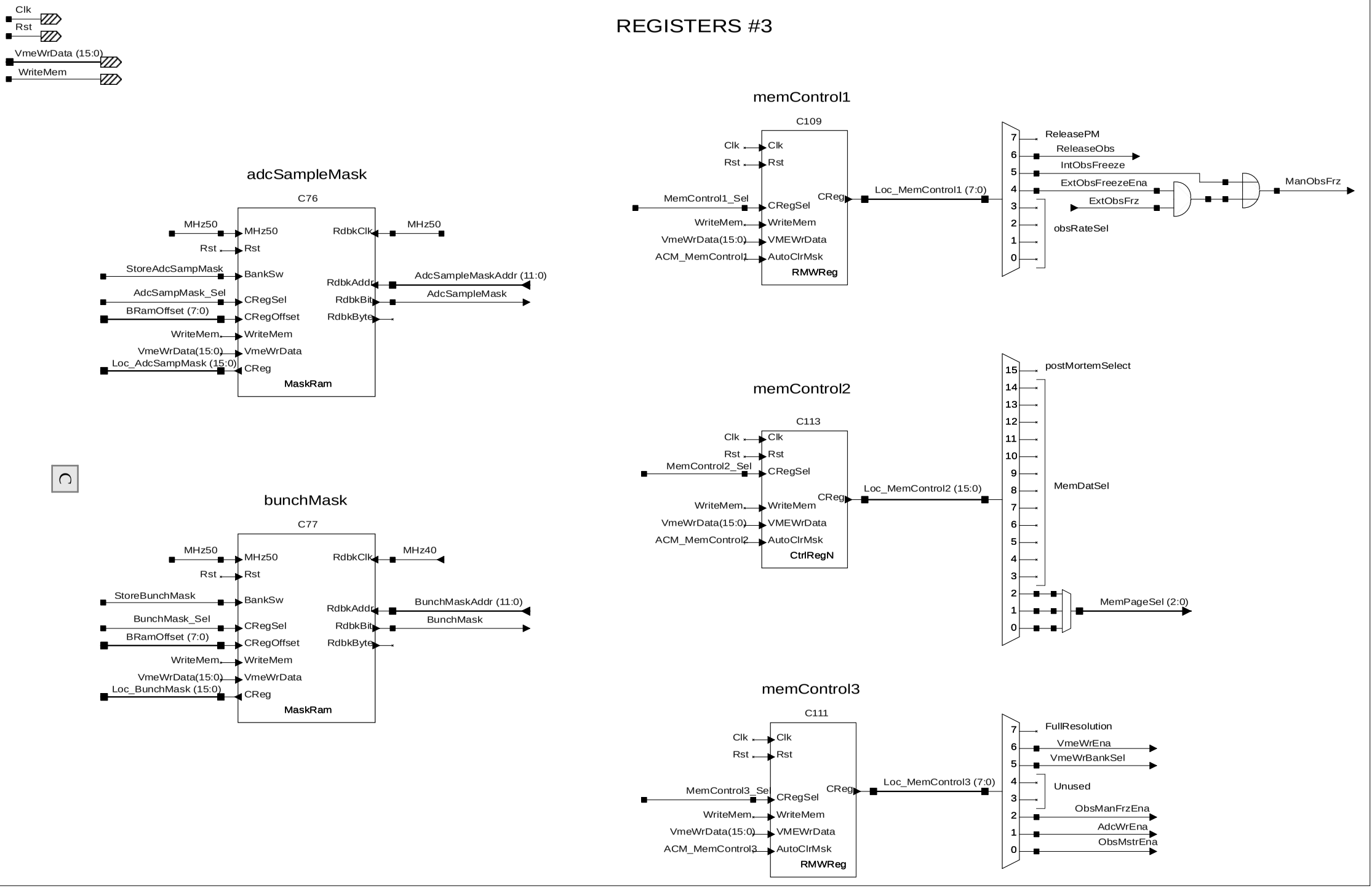
adcSampPerBank



frevAdjust

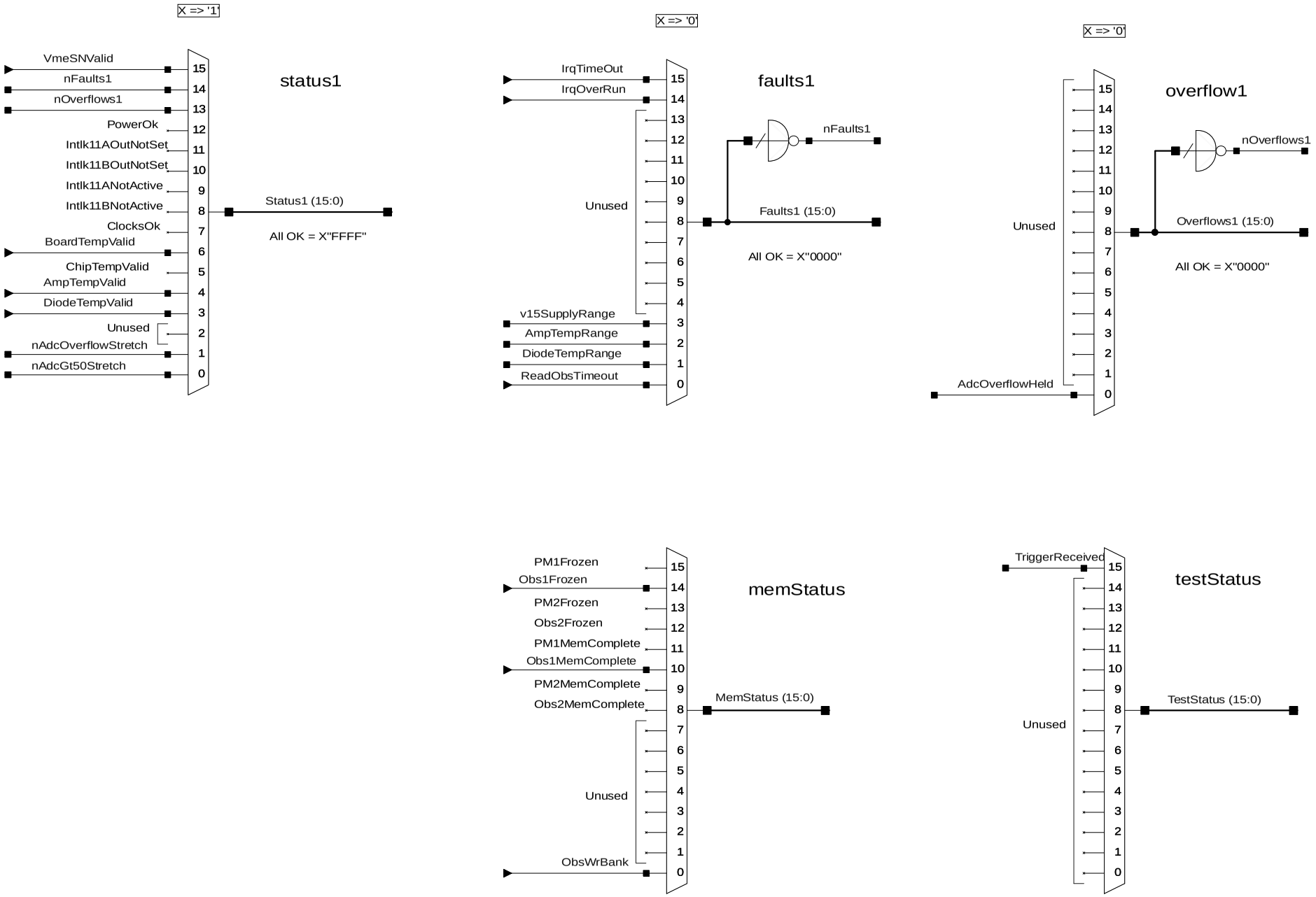


REGISTERS #3



Design: PeakDetector:RegCntrl (Sheet 4 of 7)	Date: Fri Dec 03 17:34:42 2010	Page: 4

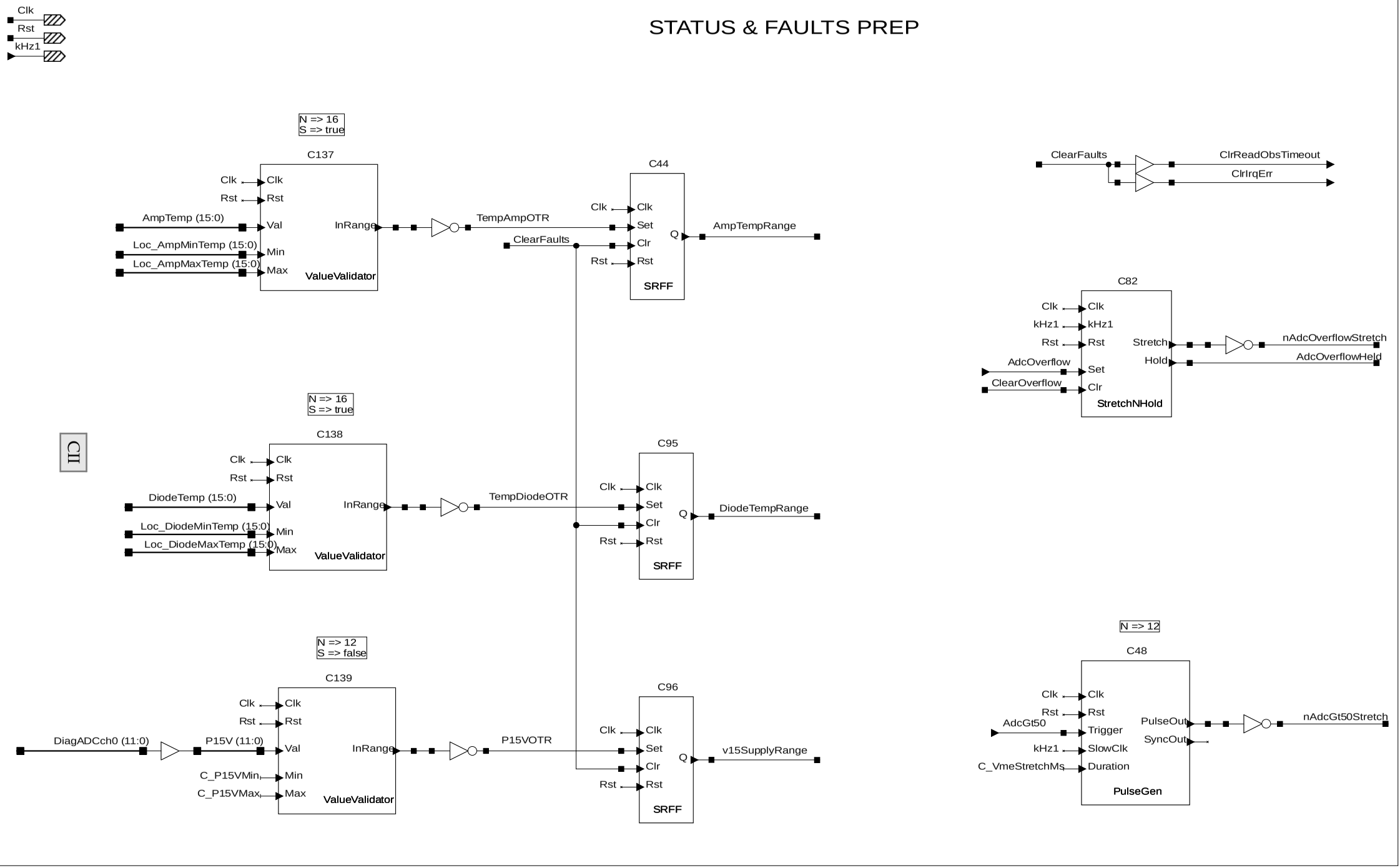
STATUS & FAULTS



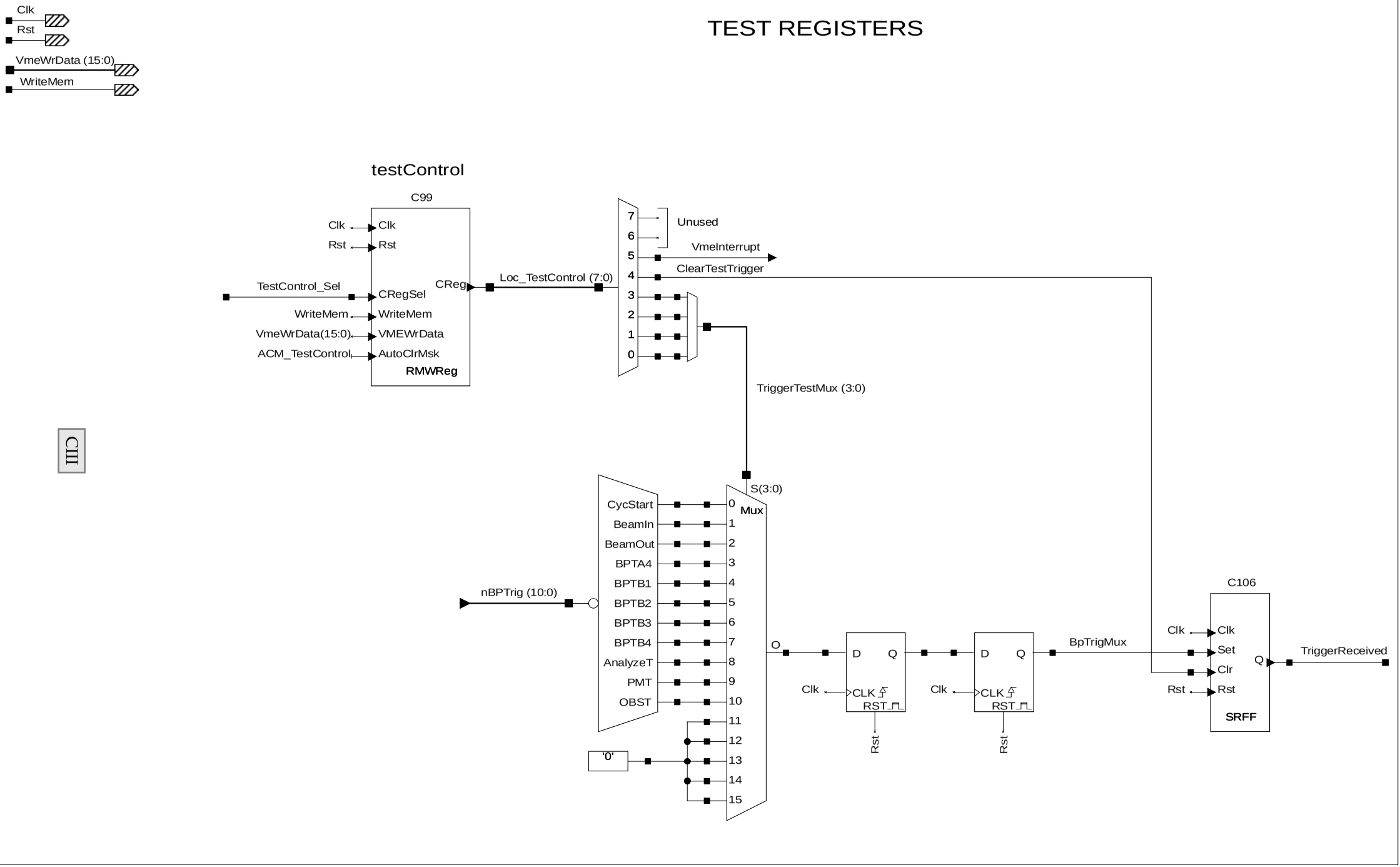
C1

Design: PeakDetector:RegCntrl (Sheet 5 of 7)	Date: Fri Dec 03 17:34:42 2010	Page: 5

STATUS & FAULTS PREP



TEST REGISTERS



CIII

Design: PeakDetector:RegCntrl (Sheet 7 of 7)	Date: Fri Dec 03 17:34:42 2010	Page: 7

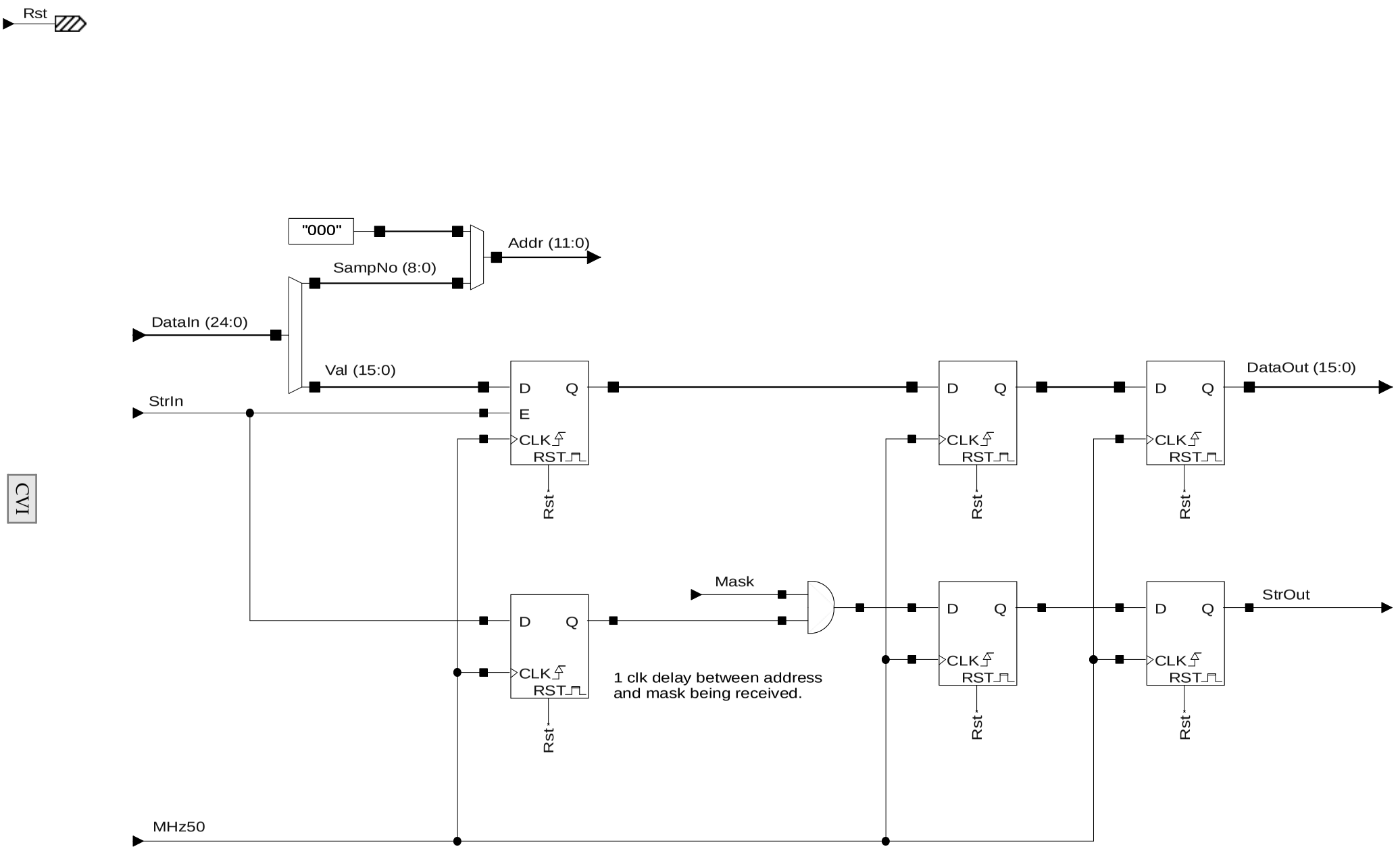
Truth Table: RegRdMux (1/2)

VmeAddr(19 downto 1) &	CRegRdOk	RdOk	VmeRdbk
Reg_Status1	'0'	'1'	Status1
Reg_Ident	'0'	'1'	JtagRemDisable & C_ExtendedID & C_CardID
Reg_Faults1	'0'	'1'	Faults1
Reg_Overflow1	'0'	'1'	Overflow1
Reg_PmAddr_0	'0'	'1'	X"0000"
Reg_PmAddr_1	'0'	'1'	X"0000"
Reg_ObsAddr_0	'0'	'1'	X"00" & "00" & ObsAddr(20 downto 15)
Reg_ObsAddr_1	'0'	'1'	ObsAddr(14 downto 0) & '0'
Reg_MemStatus	'0'	'1'	MemStatus
Reg_PeakValue	'0'	'1'	PeakValue
Reg_DiagADCch0	'0'	'1'	X"0" & DiagADCch0
Reg_DiagADCch1	'0'	'1'	X"0" & DiagADCch1
Reg_DiagADCch2	'0'	'1'	X"0" & DiagADCch2
Reg_DiagADCch3	'0'	'1'	X"0" & DiagADCch3
Reg_DiagADCch4	'0'	'1'	X"0" & DiagADCch4
Reg_DiagADCch5	'0'	'1'	X"0" & DiagADCch5
Reg_DiagADCch6	'0'	'1'	X"0" & DiagADCch6
Reg_DiagADCch7	'0'	'1'	X"0" & DiagADCch7
Reg_DiodeTemp	'0'	'1'	DiodeTemp
Reg_AmpTemp	'0'	'1'	AmpTemp
Reg_HardwareVersion	'0'	'1'	X"000" & HardwareVersion
Reg_DesignerId	'0'	'1'	C_DesignerID
Reg_BoardTemp	'0'	'1'	BoardTemp
Reg_ChipTemp	'0'	'1'	X"0000"
Reg_TestStatus	'0'	'1'	TestStatus
Reg_FirmwareVersion_0	'0'	'1'	C_FirmwareVersion(31 downto 16)
Reg_FirmwareVersion_1	'0'	'1'	C_FirmwareVersion(15 downto 0)
Reg_SerialNumber_0	'0'	'1'	SerialNumber(63 downto 48)
Reg_SerialNumber_1	'0'	'1'	SerialNumber(47 downto 32)
Reg_SerialNumber_2	'0'	'1'	SerialNumber(31 downto 16)
Reg_SerialNumber_3	'0'	'1'	SerialNumber(15 downto 0)
Reg_MemMapVersion_0	'0'	'1'	C_MemMapVersion(31 downto 16)
Reg_MemMapVersion_1	'0'	'1'	C_MemMapVersion(15 downto 0)
	'1'	'1'	CRegRdbk
		'0'	C_VmeNullRdbk

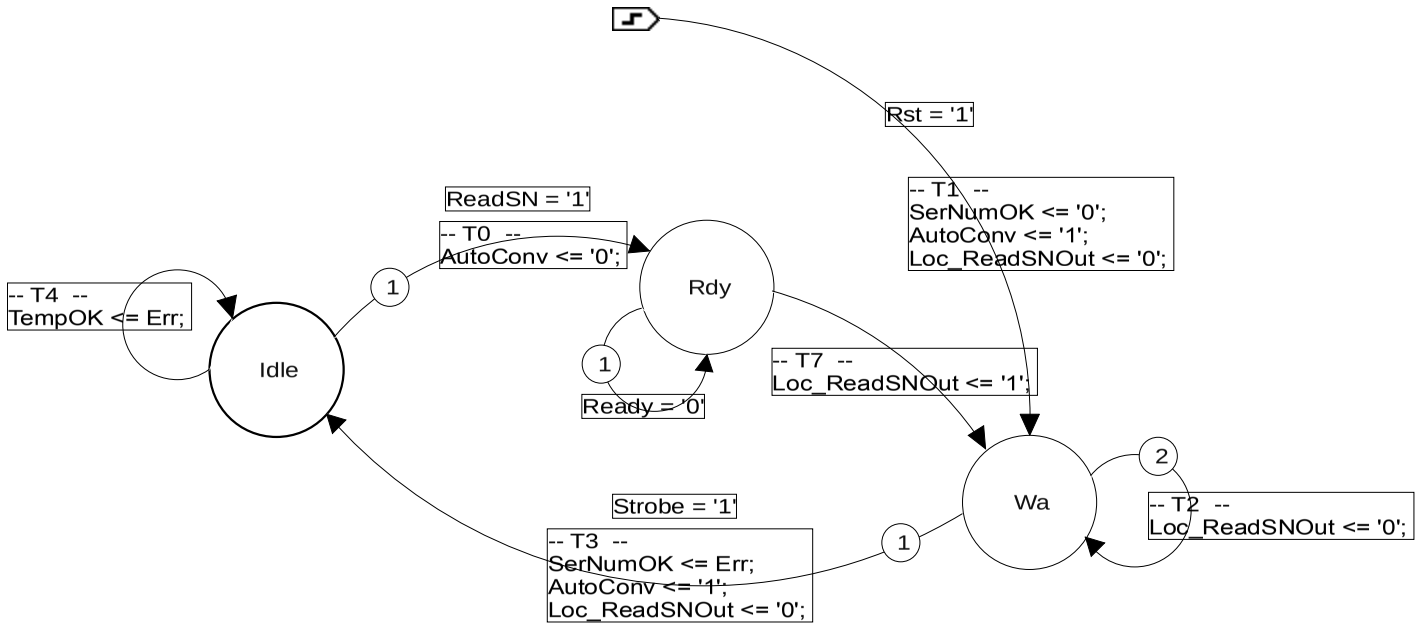
Truth Table: RegRdMux (2/2)

Packages Used
ieee.STD_LOGIC_1164
work.VmeMemMap

Interface
VmeAddr : in std_logic_vector (19 downto 0);
VmeRdbk : out std_logic_vector (15 downto 0);
RdOK : out std_logic ;
Status1 : in std_logic_vector (15 downto 0);
JtagRemDisable : in std_logic ;
Faults1 : in std_logic_vector (15 downto 0);
Overflow1 : in std_logic_vector (15 downto 0);
ObsAddr : in std_logic_vector (20 downto 0);
MemStatus : in std_logic_vector (15 downto 0);
PeakValue : in std_logic_vector (15 downto 0);
DiagADCch0 : in std_logic_vector (11 downto 0);
DiagADCch1 : in std_logic_vector (11 downto 0);
DiagADCch2 : in std_logic_vector (11 downto 0);
DiagADCch3 : in std_logic_vector (11 downto 0);
DiagADCch4 : in std_logic_vector (11 downto 0);
DiagADCch5 : in std_logic_vector (11 downto 0);
DiagADCch6 : in std_logic_vector (11 downto 0);
DiagADCch7 : in std_logic_vector (11 downto 0);
DiodeTemp : in std_logic_vector (15 downto 0);
AmpTemp : in std_logic_vector (15 downto 0);
HardwareVersion : in std_logic_vector (3 downto 0);
BoardTemp : in std_logic_vector (15 downto 0);
TestStatus : in std_logic_vector (15 downto 0);
SerialNumber : in std_logic_vector (63 downto 0);
CRegRdbk : in std_logic_vector (15 downto 0);
CRegRdOk : in std_logic ;

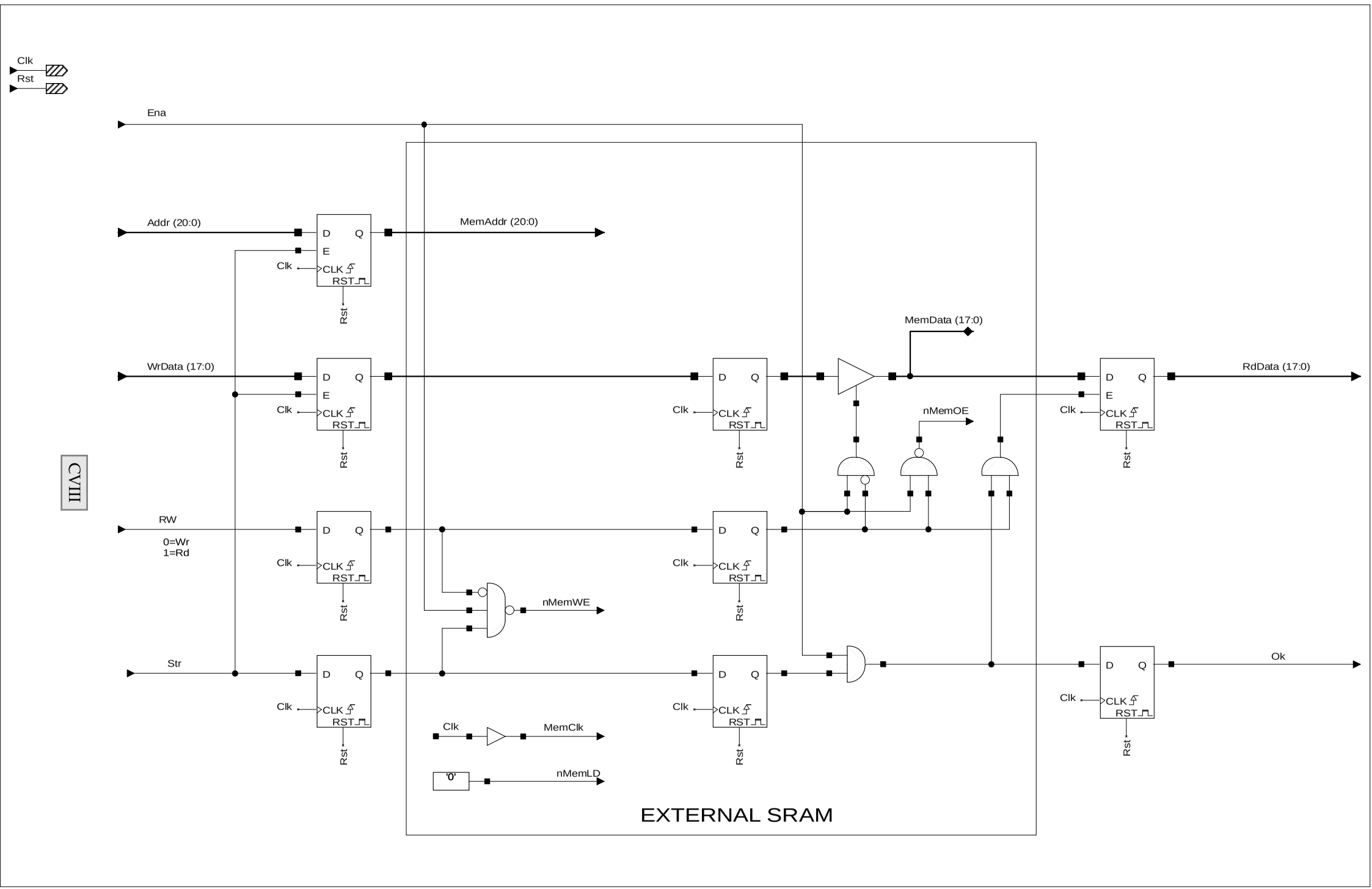


Design: PeakDetector:SampleGating	Date: Fri Dec 03 17:35:11 2010	Page: 1 (1,1)



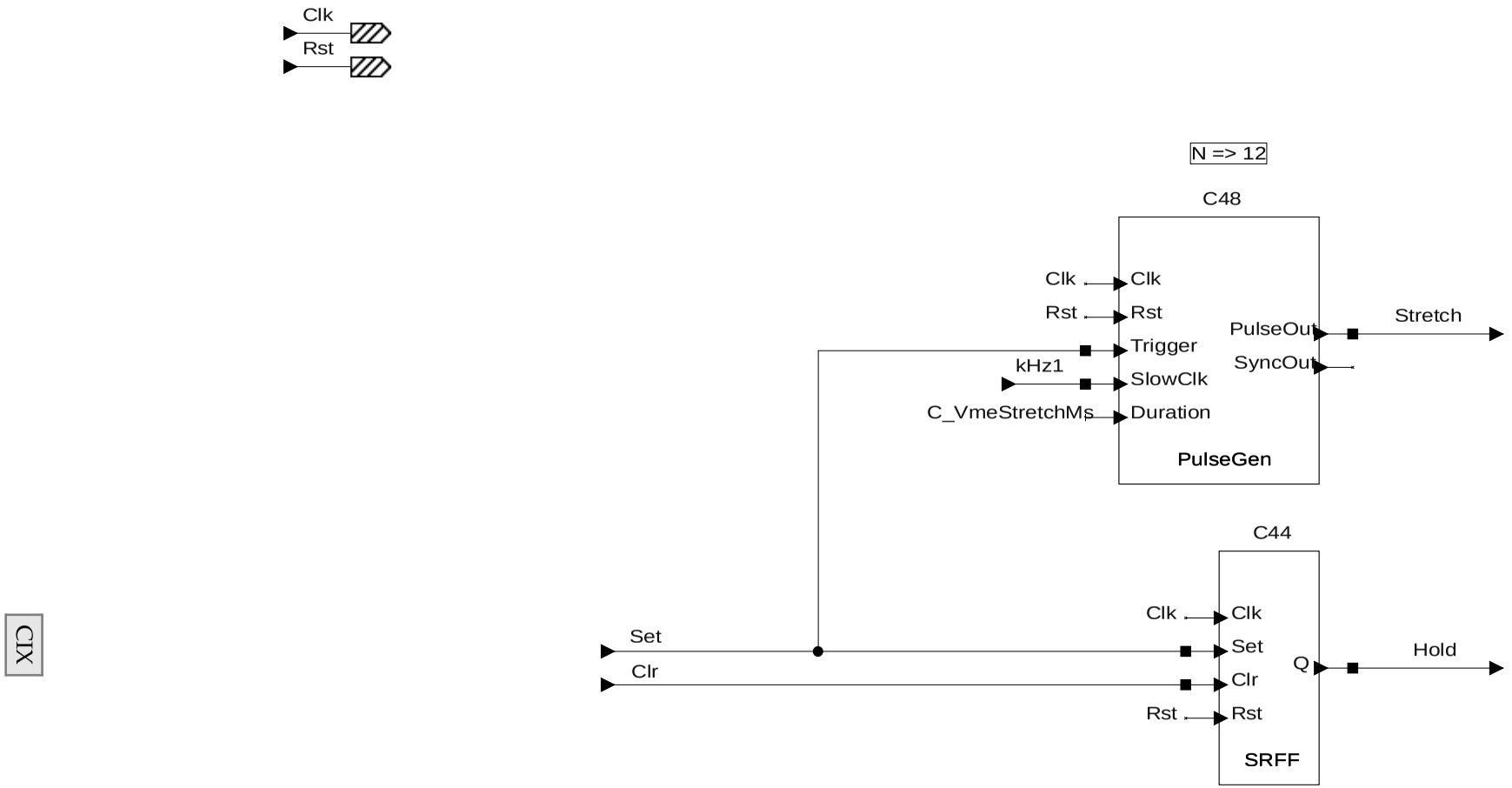
CVII

Design: PeakDetector:SerNumValidSeq	Date: Fri Dec 03 17:38:29 2010	Page: 1 (1,1)



CVIII

Design: PeakDetector:SramCtrl	Date: Fri Dec 03 17:37:36 2010	Page: 1



Design: PeakDetector:StretchNHold	Date: Fri Dec 03 17:38:05 2010	Page: 1 (1,1)

G.21 VmeMemMap

```
------------------------------------------
------------------------------------------
-- Date          : Fri Oct 22 10:58:04 2010
--
-- Author        : Tom Levens <tom.levens@cern.ch>
--
-- Company       : CERN, BE-RF-FB
--
-- Description   : VME memory map for Peak Detector.
--
------------------------------------------
------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library CommonVisual;
use CommonVisual.DesignerID.all;

package  VmeMemMap  is

  -- Firmware & Memory Map Versions.
  constant FirmwareVersion        : integer   := 20101124;
  constant MemMapVersion          : integer   := 20101122;

  -- Subtypes (for ease of typing).
  subtype vme_addr is std_logic_vector(19 downto 0); -- VME Address.
  subtype rmw_mask is std_logic_vector(7  downto 0); -- RMW register auto-clear mask (8 bits).
  subtype rwo_mask is std_logic_vector(15 downto 0); -- RW or WO register auto-clear mask (16
    bits).

  -- Constants.
  constant C_FirmwareVersion      : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(FirmwareVersion, 32));
  constant C_MemMapVersion        : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(MemMapVersion, 32));
  constant C_CardID               : std_logic_vector(7 downto 0)  := X"1D";
  constant C_ExtendedID           : std_logic_vector(6 downto 0)  := "0000000";
  constant C_DesignerID           : std_logic_vector(15 downto 0) := Tom;
  constant C_VmeNullRdbk          : std_logic_vector(15 downto 0) := X"CAFE";

  -- Auto-clear masks for RMW and RW registers.
  constant ACM_Control1           : rmw_mask := "11101100";
  constant ACM_VmeIRQStatID       : rwo_mask := "0000000000000000";
  constant ACM_VmeIRQLvl          : rwo_mask := "0000000000000000";
  constant ACM_MemControl1        : rmw_mask := "11101111";
  constant ACM_MemControl2        : rwo_mask := "1111111111111100";
  constant ACM_MemControl3        : rmw_mask := "10011000";
  constant ACM_Control2           : rmw_mask := "11111111";
  constant ACM_AttenControl       : rwo_mask := "1111111111000000";
  constant ACM_FrevAdjust         : rwo_mask := "1111000000000000";
  constant ACM_DiodeMinTemp       : rwo_mask := "0000000000000000";
  constant ACM_DiodeMaxTemp       : rwo_mask := "0000000000000000";
  constant ACM_AmpMinTemp         : rwo_mask := "0000000000000000";
  constant ACM_AmpMaxTemp         : rwo_mask := "0000000000000000";
  constant ACM_AdcSampPerBank_0   : rwo_mask := "1111111111100000";
  constant ACM_AdcSampPerBank_1   : rwo_mask := "0000000000000000";
  constant ACM_SysControl         : rmw_mask := "00000000";
  constant ACM_TestControl        : rmw_mask := "11110000";

  -- Lower card independant registers.
  constant Reg_Control1           : vme_addr := X"00000"; -- RMW  2
  constant Reg_VmeIRQStatID       : vme_addr := X"00002"; -- RW   2
  constant Reg_VmeIRQLvl          : vme_addr := X"00004"; -- RW   2
  constant Reg_Status1            : vme_addr := X"00006"; -- RO   2
  constant Reg_Ident              : vme_addr := X"00008"; -- RO   2
  constant Reg_Faults1            : vme_addr := X"0000A"; -- RO   2
  constant Reg_Overflow1          : vme_addr := X"0000C"; -- RO   2
```

```
  -- Memory specific registers.
  constant Reg_PmAddr_0           : vme_addr := X"0000E"; -- RO   4[0]
  constant Reg_PmAddr_1           : vme_addr := X"00010"; -- RO   4[1]
  constant Reg_ObsAddr_0          : vme_addr := X"00012"; -- RO   4[0]
  constant Reg_ObsAddr_1          : vme_addr := X"00014"; -- RO   4[1]
  constant Reg_MemStatus          : vme_addr := X"00016"; -- RO   2
  constant Reg_MemControl1        : vme_addr := X"00018"; -- RMW  2
  constant Reg_MemControl2        : vme_addr := X"0001A"; -- RW   2
  constant Reg_MemControl3        : vme_addr := X"0001C"; -- RMW  2

  -- Card registers
  constant Reg_Control2           : vme_addr := X"0001E"; -- RW   2
  constant Reg_AttenControl       : vme_addr := X"00020"; -- RMW  2
  constant Reg_PeakValue          : vme_addr := X"00022"; -- RO   2
  constant Reg_FrevAdjust         : vme_addr := X"00024"; -- RW   2

  constant Reg_DiagADCch0         : vme_addr := X"00030"; -- RO   2
  constant Reg_DiagADCch1         : vme_addr := X"00032"; -- RO   2
  constant Reg_DiagADCch2         : vme_addr := X"00034"; -- RO   2
  constant Reg_DiagADCch3         : vme_addr := X"00036"; -- RO   2
  constant Reg_DiagADCch4         : vme_addr := X"00038"; -- RO   2
  constant Reg_DiagADCch5         : vme_addr := X"0003A"; -- RO   2
  constant Reg_DiagADCch6         : vme_addr := X"0003C"; -- RO   2
  constant Reg_DiagADCch7         : vme_addr := X"0003E"; -- RO   2

  constant Reg_DiodeTemp          : vme_addr := X"00040"; -- RO   2
  constant Reg_AmpTemp            : vme_addr := X"00042"; -- RO   2
  constant Reg_DiodeMinTemp       : vme_addr := X"00044"; -- RW   2
  constant Reg_DiodeMaxTemp       : vme_addr := X"00046"; -- RW   2
  constant Reg_AmpMinTemp         : vme_addr := X"00048"; -- RW   2
  constant Reg_AmpMaxTemp         : vme_addr := X"0004A"; -- RW   2

  constant Reg_AdcSampPerBank_0   : vme_addr := X"0004C"; -- RW   4[0]
  constant Reg_AdcSampPerBank_1   : vme_addr := X"0004E"; -- RW   4[1]

  -- Upper card independant registers.
  constant Reg_SysControl         : vme_addr := X"000E0"; -- RMW  2
  constant Reg_HardwareVersion    : vme_addr := X"000E2"; -- RO   2
  constant Reg_DesignerId         : vme_addr := X"000E4"; -- RO   2
  constant Reg_BoardTemp          : vme_addr := X"000E6"; -- RO   2
  constant Reg_ChipTemp           : vme_addr := X"000E8"; -- RO   2
  constant Reg_TestControl        : vme_addr := X"000EC"; -- RMW  2
  constant Reg_TestStatus         : vme_addr := X"000EE"; -- RO   2
  constant Reg_FirmwareVersion_0  : vme_addr := X"000F0"; -- RO   4[0]
  constant Reg_FirmwareVersion_1  : vme_addr := X"000F2"; -- RO   4[1]
  constant Reg_SerialNumber_0     : vme_addr := X"000F4"; -- RO   8[0]
  constant Reg_SerialNumber_1     : vme_addr := X"000F6"; -- RO   8[1]
  constant Reg_SerialNumber_2     : vme_addr := X"000F8"; -- RO   8[2]
  constant Reg_SerialNumber_3     : vme_addr := X"000FA"; -- RO   8[3]
  constant Reg_MemMapVersion_0    : vme_addr := X"000FC"; -- RO   4[0]
  constant Reg_MemMapVersion_1    : vme_addr := X"000FE"; -- RO   4[1]

  constant Reg_AdcSampleMask      : vme_addr := X"00200"; -- RW   38[0]
  constant Reg_AdcSampleMask_End  : vme_addr := X"00225"; -- RW   38[38]
  constant Reg_BunchMask          : vme_addr := X"00400"; -- RW   446[0]
  constant Reg_BunchMask_End      : vme_addr := X"005BD"; -- RW   446[N]

  -- Memory viewport.
  constant Reg_MemViewport        : vme_addr := X"80000"; -- RW   512k[0]
  constant Reg_MemViewport_End    : vme_addr := X"FFFFF"; -- RW   512k[N]
end;

--EOF
```

Truth Table: VmeMux

VmeAddr	VmeAddr	VmeRdData	RegWrStr	RegRdStr	MemWrStr	MemRdStr	VmeRdDone
>= Reg_MemViewport	<= Reg_MemViewport_End	MemRdData	'0'	'0'	VmeWrStr	VmeRdStr	MemRdOk
		RegRdData	VmeWrStr	VmeRdStr	'0'	'0'	RegRdOk

Packages Used
ieee.STD_LOGIC_1164
work.VmeMemMap

Interface
VmeAddr : in std_logic_vector (19 downto 0);
VmeRdData : out std_logic_vector (15 downto 0);
MemRdData : in std_logic_vector (15 downto 0);
RegRdData : in std_logic_vector (15 downto 0);
MemAddr : out std_logic_vector (20 downto 0);
MemPageSel : in std_logic_vector (2 downto 0);
VmeWrData : in std_logic_vector (15 downto 0);
RegAddr : out std_logic_vector (19 downto 0);
MemWrData : out std_logic_vector (15 downto 0);
RegWrData : out std_logic_vector (15 downto 0);
MemRdStr : out std_logic ;
MemWrStr : out std_logic ;
RegRdStr : out std_logic ;
RegWrStr : out std_logic ;
RegRdOk : in std_logic ;
RegWrOk : in std_logic ;
MemRdOk : in std_logic ;
MemWrOk : in std_logic ;
VmeRdStr : in std_logic ;
VmeWrStr : in std_logic ;
VmeRdDone : out std_logic ;
VmeWrDone : out std_logic ;

Assignments
MemAddr(20 downto 18) <= MemPageSel(2 downto 0);
MemAddr(17 downto 0) <= VmeAddr(18 downto 1);
MemWrData <= VmeWrData;
RegWrData <= VmeWrData;
RegAddr <= VmeAddr;

VME Peak Detector V2 Memory Map

Memory Map Version	20101122	
Documentation	20101216	

Offset	Size	Function	Mode	Remarks
0xN00000	2	control1	RMW	
0xN00002	2	vmeIRQStatID	RW	
0xN00004	2	vmeIRQLevel	RW	
0xN00006	2	status1	RO	
0xN00008	2	ident	RO	*Card ID = 0x1D*
0xN0000A	2	faults1	RO	
0xN0000C	2	overflow1	RO	
0xN0000E	4	pmAddr	RO	
0xN00012	4	obsAddr	RO	
0xN00016	2	memStatus	RO	
0xN00018	2	memControl1	RMW	
0xN0001A	2	memControl2	RW	
0xN0001C	2	memControl3	RMW	
0xN0001E	2	control2	RMW	
0xN00020	2	attenControl	RW	*Bits 5..0, others ignored*
0xN00022	2	peakValue	RO	*Unsigned 14b, with flags in b15 and b16*
0xN00024	2	frevAdjust	RW	*Unsigned 12b, range 0..3563*
0xN00026	10			
0xN00030	2	v15Supply	RO	*Unsigned 12b, LSB=7.3mV*
0xN00032	2	diagADCch1	RO	
0xN00034	2	diagADCch2	RO	
0xN00036	2	diagADCch3	RO	
0xN00038	2	diagADCch4	RO	
0xN0003A	2	diagADCch5	RO	
0xN0003C	2	diagADCch6	RO	
0xN0003E	2	diagADCch7	RO	
0xN00040	2	diodeTemp	RO	*Signed 16b, LSB=0.0625℃*
0xN00042	2	ampTemp	RO	*Signed 16b, LSB=0.0625℃*
0xN00044	2	diodeMinTemp	RW	*Signed 16b, LSB=0.0625℃* ***[5]***
0xN00046	2	diodeMaxTemp	RW	*Signed 16b, LSB=0.0625℃* ***[5]***
0xN00048	2	ampMinTemp	RW	*Signed 16b, LSB=0.0625℃* ***[5]***
0xN0004A	2	ampMaxTemp	RW	*Signed 16b, LSB=0.0625℃* ***[5]***
0xN0004C	4	adcSampPerBank	RW	*Unsigned 21b number.*
0xN0004E	146			
0xN000E0	2	sysControl	RMW	
0xN000E2	2	hardwareVersion	RO	*Hardcoded to 0x0 in hardware V1*
0xN000E4	2	designerID	RO	*Firmware designer ID*
0xN000E6	2	boardTemp	RO	*Does not readback in hardware V1*
0xN000E8	2	chipTemp	RO	*Not available, readback 0x0000*
0xN000EA	2	reserved		
0xN000EC	2	testControl	RMW	
0xN000EE	2	testStatus	RO	
0xN000F0	4	firmwareVersion	RO	*Firmware version (MutadF)*
0xN000F4	8	serialNumber	RO	*64 bit silicon serial number*
0xN000FC	4	memMapVersion	RO	*Memory map version (MutadF)*
0xN00100	256			
0xN00200	38	adcSampleMask	RW	*Bits 296..0, others ignored* ***[3]***
0xN00226	74			*Reserved for increase in ADC samp. rate.*
0xN00270	624			
0xN00400	446	bunchMask	RW	*Bits 3563..0, others ignored* ***[3]***
0xN005BE	~318k			
0xN50000	64k	reserved	—	*CTRV VME Slot 1*
0xN60000	64k	reserved	—	*CTRV VME Slot 2*
0xN70000	64k	reserved	—	*CTRV VME Slot 3*
0xN80000	512k	memViewport	RO	

	Card independant register function/offset area.
	RMW register mask bits.
	Unused memory.
	Memory reserved for future expansion.
	Card independent register/bit implemented but unused.
	Card independent memory related area.

control1 **RMW** 0xN00000

Bit	Function	Remarks
15..8	rmwMask	*1=Modify, 0=Retain*
7	clearFaults	*Auto-cleared*
6	clearOverflow	*Auto-cleared*
5	getSerialNumber	*Auto-cleared*
4	enableIRQ	*1=Enable*
3		*Write=ignored, Read=0*
2	v15SupplyOn	*1=On, 0=Off*
1	gateEna	*1=Enable, 0=Disable*
0	frevEdgeSel	*0=Pos, 1=Neg*

vmeIRQStatID **RW** 0xN00002

This register containing the IRQ Vector is initialized by the Driver

vmeIRQLevel **RW** 0xN00004

This register containing the IRQ Level is initialized by the Driver

status1 **RO** 0xN00006

Bit	Function	Remarks
15	vmeSNValid	*1=Valid*
14	noFaults	*1=No faults occurred*
13	noOverflow	*1=No oveflow occurred*
12	powerOk	*Read=1*
11	intlk11AOutNotSet	*Read=1*
10	intlk11BOutNotSet	*Read=1*
9	intlk11ANotActive	*Read=1*
8	intlk11BNotActive	*Read=1*
7	clocksOk	*Read=1*
6	boardTempValid	*1=Valid*
5	chipTempValid	*Read=1*
4	ampTempValid	*1=Valid*
3	diodeTempValid	*1=Valid*
2		
1	adcGt50	*0=>50%, stretched to 1.34s*
0	adcOverflow	*0=Overflow, stretched to 1.34s*

ident **RO** 0xN00008

Bit	Function	Remarks
15	jtagRemDisable	*HW Strap setting*
14..8	extendedID	*Piggy-back ID*
7..0	cardID	*Main Card ID*

faults1 **RO** 0xN0000A

Bit	Function	Remarks
15	irqTimeOut	*1=IRQ not serviced within 655us*
14	irqOverRun	*1=IRQ triggered before previous serviced*
13		*Read=0*
12		*Read=0*
11		*Read=0*
10		*Read=0*
9		*Read=0*
8		*Read=0*
7		*Read=0*
6		*Read=0*
5		*Read=0*
4		*Read=0*
3	v15SupplyRange	*1=Out of range*
2	ampTempRange	*1=Out of range*
1	diodeTempRange	*1=Out of range*
0	readObsTimeout	*1=Obs buffer not read in time* ***[1]***

overflow1 **RO** 0xN0000C

Bit	Function	Remarks
15		*Read=0*
14		*Read=0*
13		*Read=0*
12		*Read=0*
11		*Read=0*
10		*Read=0*
9		*Read=0*
8		*Read=0*
7		*Read=0*
6		*Read=0*
5		*Read=0*
4		*Read=0*
3		*Read=0*
2		*Read=0*
1		*Read=0*
0	adcOverflow	*1=Out of range*

pmAddr **RO** 0xN0000E

Bit	Function	Remarks
31..0	frzPMAddr	*Read=0x00000000*

obsAddr **RO** 0xN00012

Bit	Function	Remarks
31..0	frzObsAddr	*Next mem address on OBS freeze*

memStatus **RO** 0xN00016

Bit	Function	Remarks
15	pm1Frozen	*Read=0*
14	obs1Frozen	*Activated by bank full flag **[1]***
13	pm2Frozen	*Read=0*
12	obs2Frozen	*Read=0*
11	pm1MemComplete	*Read=0*
10	obs1MemComplete	*1=All mem written, 0=Partially written*
9	pm2MemComplete	*Read=0*
8	obs2MemComplete	*Read=0*
7..1		*Read=0*
0	obsWrBank	*1=Bank 1, 0=Bank 0*

memControl1 **RMW** 0xN00018

Bit	Function	Remarks
15..8	rmwMask	*1=Modify, 0=Retain*
7	releasePM	*Write=ignored, Read=0*
6	releaseObs	*Auto-cleared **[1]***
5	intObsFreeze	*Auto-cleared*
4	extObsFreezeEna	*1=Enabled, 0=Disabled*
3..0	obsRateSel	*Write=ignored, Read=0*

memControl2 **RW** 0xN0001A

Bit	Function	Remarks
15	postMortemSel	*Write=ignored, Read=0*
14..1	memDatSel	*Write=ignored, Read=0*
2..0	memViewPortSel	

memControl3 **RMW** 0xN0001C

Bit	Function	Remarks
15..8	rmwMask	*1=Modify, 0=Retain*
7	fullResolution	*Write=ignored, Read=0*
6	vmeWrEna	*1=VME mem write, 0=ADC mem write **[2]***
5	vmeWrBankSel	*1=Bank 1, 0=Bank 0*
4		*Write=ignored, Read=0*
3		*Write=ignored, Read=0*
2	obsManFrzEna	*1=Enable, 0=Disable **[6]***
1	adcWrEna	*1=Enable, 0=Disable*
0	obsMstrEna	*1=Enable, 0=Disable*

control2 **RMW** 0xN0001E

Bit	Function	Remarks
15..8	rmwMask	*1=Modify, 0=Retain*
7		*Write=ignored, Read=0*
6		*Write=ignored, Read=0*
5		*Write=ignored, Read=0*
4		*Write=ignored, Read=0*
3		*Write=ignored, Read=0*
2	storeAdcSampPerBank	*Auto-cleared **[4]***
1	storeBunchMask	*Auto-cleared **[4]***
0	storeAdcSampleMask	*Auto-cleared **[4]***

attenControl **RW** 0xN00020

Bit	Function	Remarks
15..6		*Write=ignored, Read=0*
5	db32Ena	*1=Enable, 0=Disable*
4	db16Ena	*1=Enable, 0=Disable*
3	db8Ena	*1=Enable, 0=Disable*
2	db4Ena	*1=Enable, 0=Disable*
1	db2Ena	*1=Enable, 0=Disable*
0	db1Ena	*1=Enable, 0=Disable*

peakValue **RO** 0xN00022

Bit	Function	Remarks
15	frevMarker	*1=Sample taken in bucket 0*
14	adcOverflow	*1=ADC data out of range*
13..0	adcValue	*Unsigned 14b*

frevAdjust **RW** 0xN00024

Bit	Function	Remarks
15..12		*Write=ignored, Read=0*
11..0	frevAdjust	*Unsigned 12b, range 0..3563*

testControl **RMW** 0xN000EC

Bit	Function	Remarks
15..8	rmwMask	*1=Modify, 0=Retain*
7..6		*Write=ignored, Read=0*
5	vmeInterrupt	*Auto-cleared*
4	clearTestTrigger	*Auto-cleared*
3..0	triggerTestMux	

testStatus **RO** 0xN000EE

Bit	Function	Remarks
15	triggerReceived	
14..0		*Write=ignored, Read=0*

Notes

[1] Memory is double banked and is automatically switched by hardware once a certain number of samples (given by register **adcSampPerBank**) have been recorded. Once this happens, the OBS memory will be automatically frozen and an interrupt will be generated. Once this happens the host PC must read out the observation memory before the second bank has been filled with the same number of samples. Once the memory has been completely read, the OBS buffer should be released by setting the **releaseObs** flag in register **memControl1**. If a the OBS buffer has not been released by the time the switch is due to take place, an error will be flagged and the current bank will be overwritten.

[2] When this bit is enabled, ADC data will not be recorded to memory and instead the bank is considered writable by VME. Bank switch is not performed automatically, but can be switched with **vmeWrBankSel** bit in **memControl3**.

[3] For masks the least-significant-address is bits 0-15, next 16-31 etc.

[4] The masks are double banked and the bank switch is performed when this bit is set. Samp/Bank has a similar functionality, and is only commited when it's bit is set, but is not double banked.

[5] These min/max values set the temp range outwith which a fault flag should be set.

[6] When enabled, the memory bank will not be automatically switched when N samples have been recorded. Instead, they will be switched when an OBS freeze is received. If the memory bank is filled before this, data acquisition will stop.

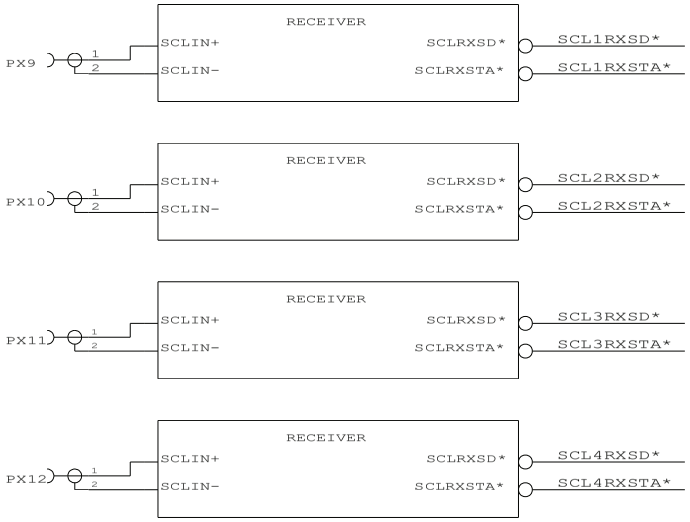
I Serial Link Router and Tester — Design Files

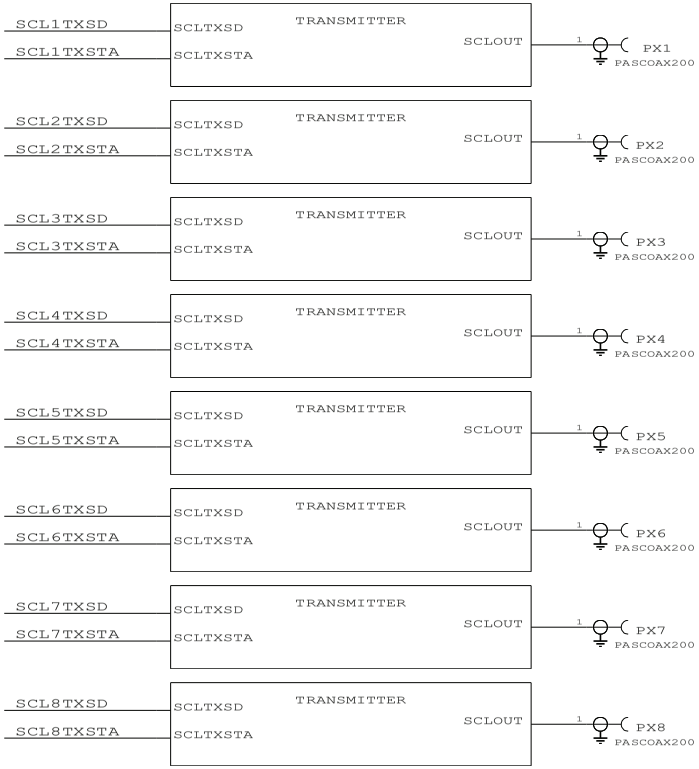
This appendix contains the schematic, PCB and mechanical parts for the Serial Link Router and Serial Link Tester modules. The following documents are included:

All documents were prepared by Mr. Jean-Marc Combe in the CERN TE-DEM department. Duplicate pages of the schematic are omitted for clarity. For the PCB: green lines indicate tracks on the top layer and red indicates tracks on the bottom.

All files related the the DTU are published on CERN's EDMS service with the identifier EDA-02163-V2-0 (SLR) [28] and EDA-02163-V2-1 (SLT) [29].

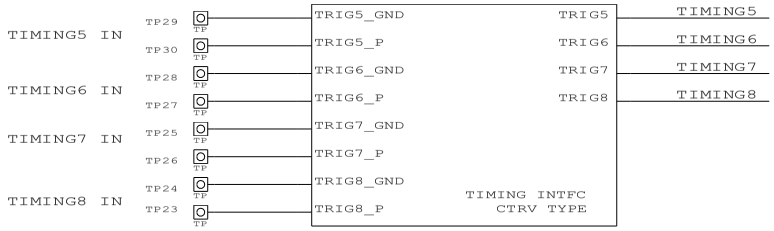
CONNECTORS FOR SCL2: DO NOT ASSEMBLE
PREPARE COAX CONN. PINS TO SOLDER A CABLE

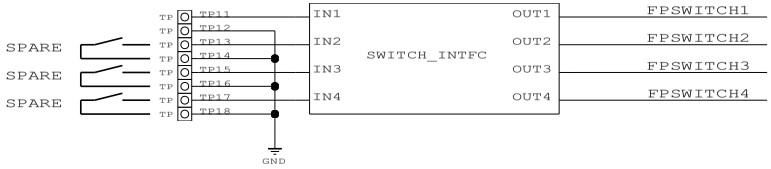


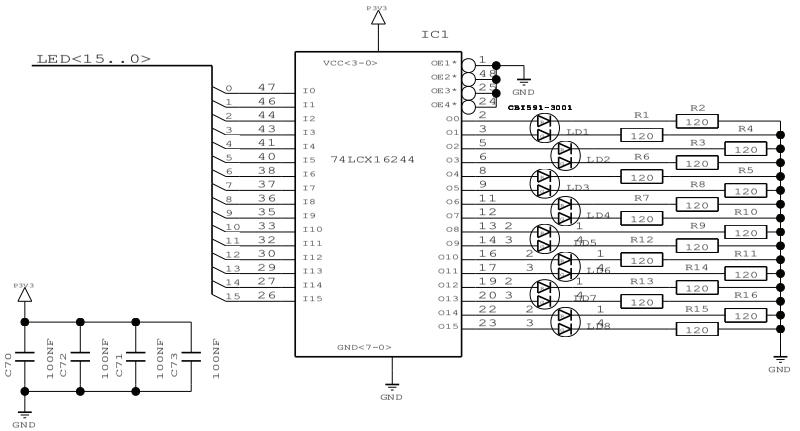


POWER_REGULATORS

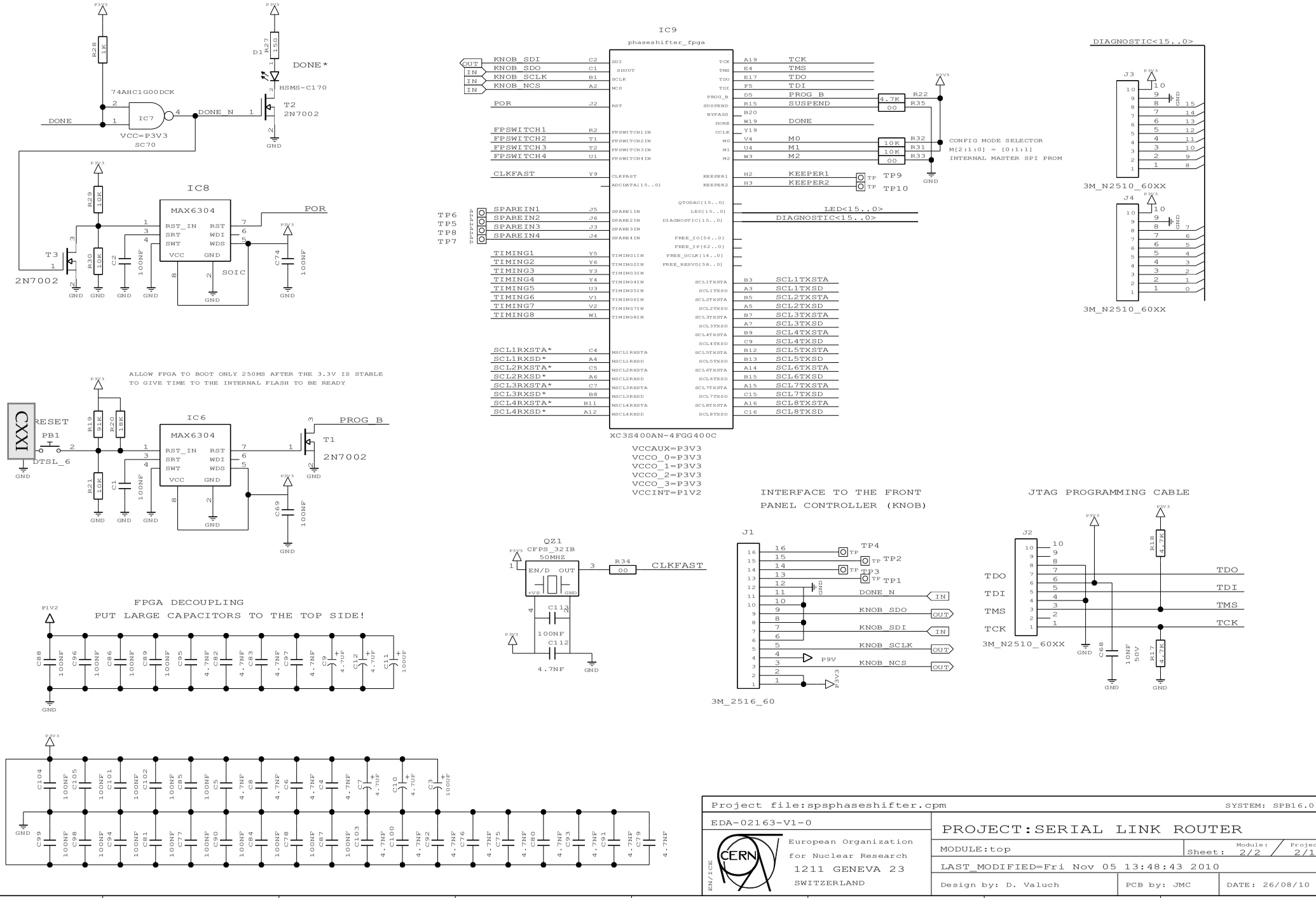
CXX



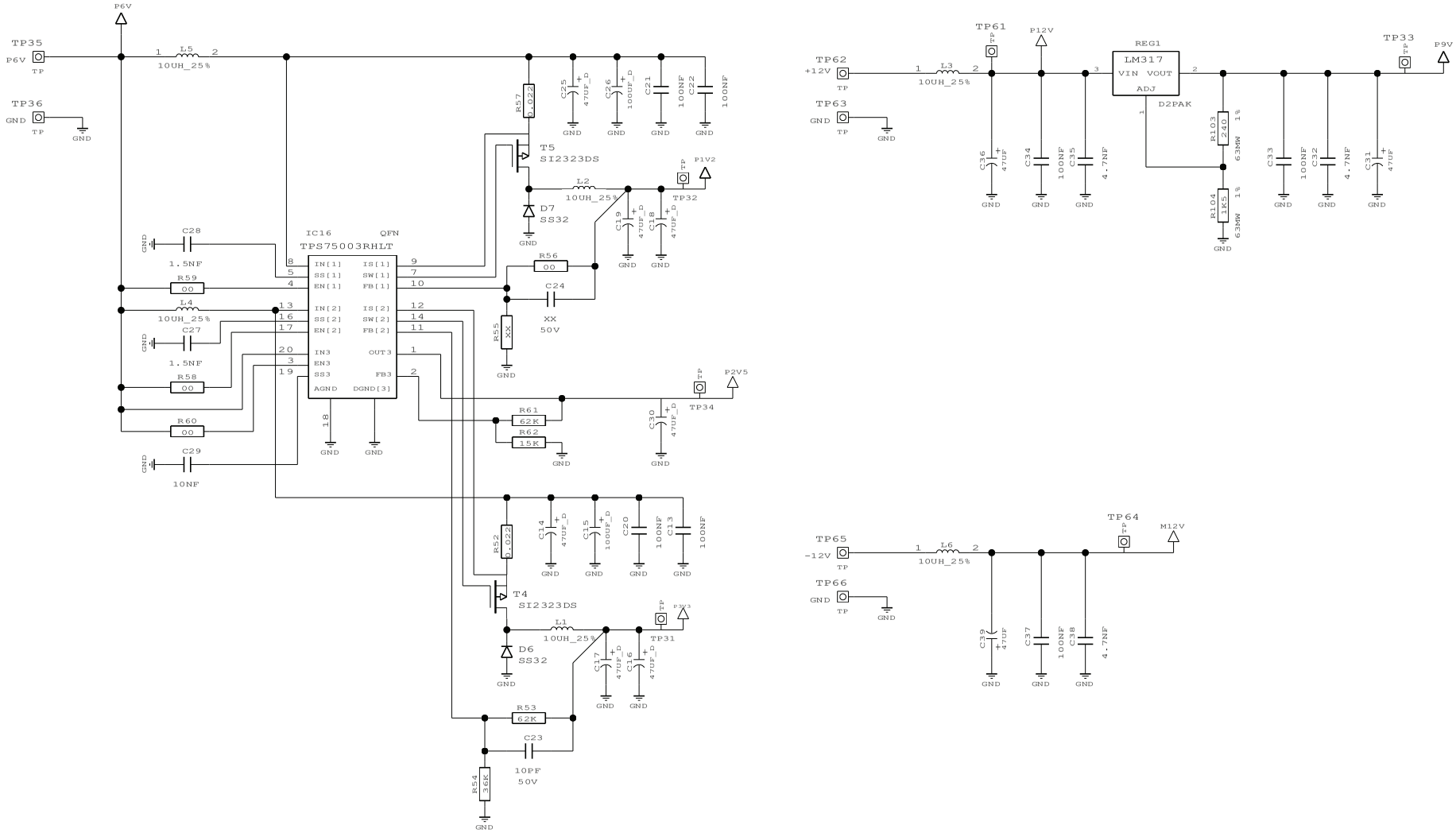




Project file:spsphaseshifter.cpm		SYSTEM: SPB16.01
EDA-02163-V1-0	PROJECT:SERIAL LINK ROUTER	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:top	Module: Sheet: 1/2 / Project: 1/18
	LAST_MODIFIED=Fri Nov 05 13:48:42 2010	
	Design by: D. Valuch / PCB by: JMC	DATE: 26/08/10

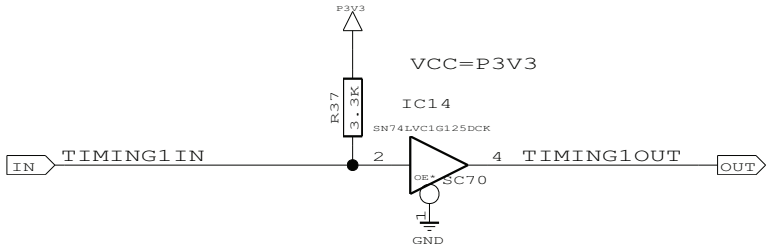


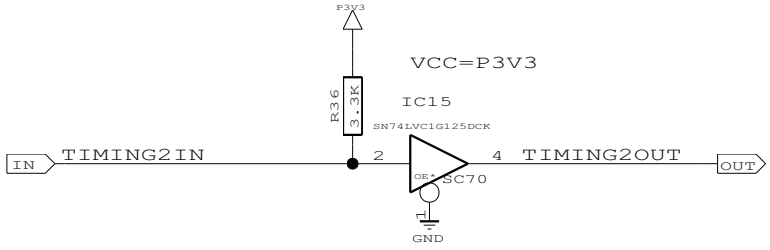
Project file: spsphaseshifter.cpm	SYSTEM: SPB16.01
EDA-02163-V1-0	PROJECT: SERIAL LINK ROUTER
European Organization for Nuclear Research, 1211 GENEVA 23, SWITZERLAND	MODULE: top; Sheet: 2/2; Project: 2/18
	LAST_MODIFIED=Fri Nov 05 13:48:43 2010
	Design by: D. Valuch; PCB by: JMC; DATE: 26/08/10

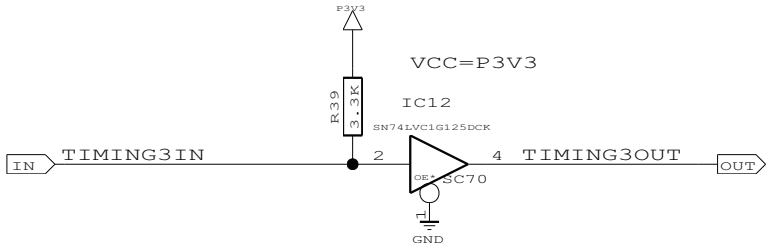


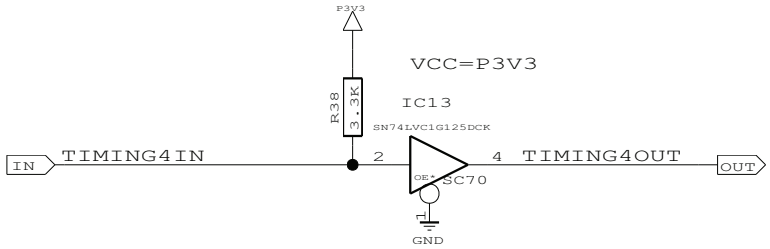
CXXIII

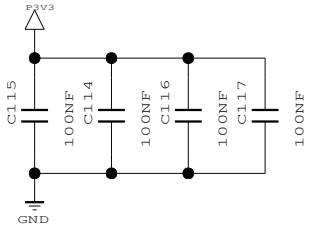
Project file:spsphaseshifter.cpm		SYSTEM: SPB16.01
EDA-02163-V1-0	PROJECT:SERIAL LINK ROUTER	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:power_regulators	Sheet: 1/1 / Module: / Project: 3/18
EN/ICE	LAST_MODIFIED=Fri Nov 05 13:48:48 2010	
	Design by: D. Valuch / PCB by: JMC	DATE: 26/08/10











Project file:spsphaseshifter.cpm SYSTEM: SPB16.01

EDA-02163-V1-0

CERN
European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND
TS/DEM

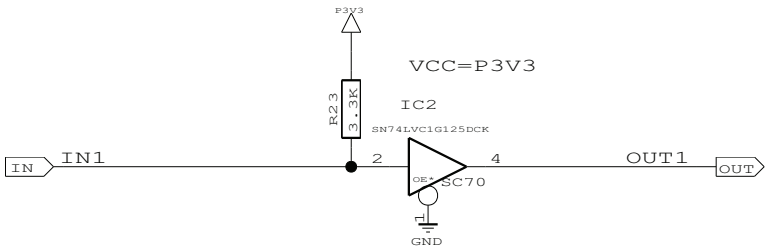
PROJECT:SERIAL LINK ROUTER

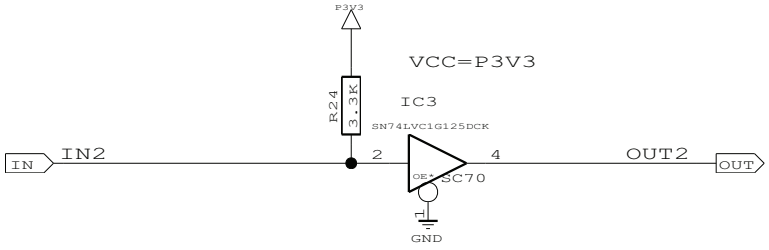
MODULE:timing_rcv | Sheet: Module: 1/1 / Project: 4/18

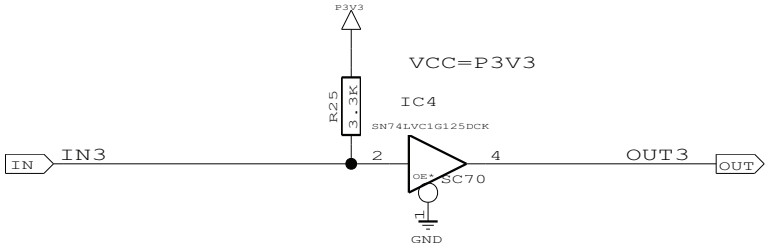
LAST_MODIFIED=Fri Nov 05 13:48:49 2010

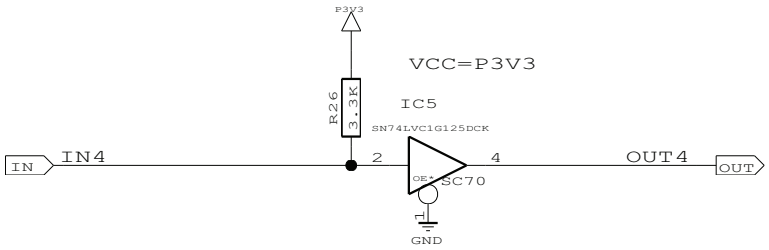
Design by: [illegible]ch | PCB by: JMC | DATE: 26/08/10

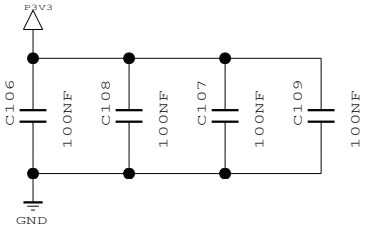
CXXIII











Project file:spsphaseshifter.cpm SYSTEM: SPB16.01

EDA-02163-V1-0

European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND
TS/DEM

PROJECT:SERIAL LINK ROUTER

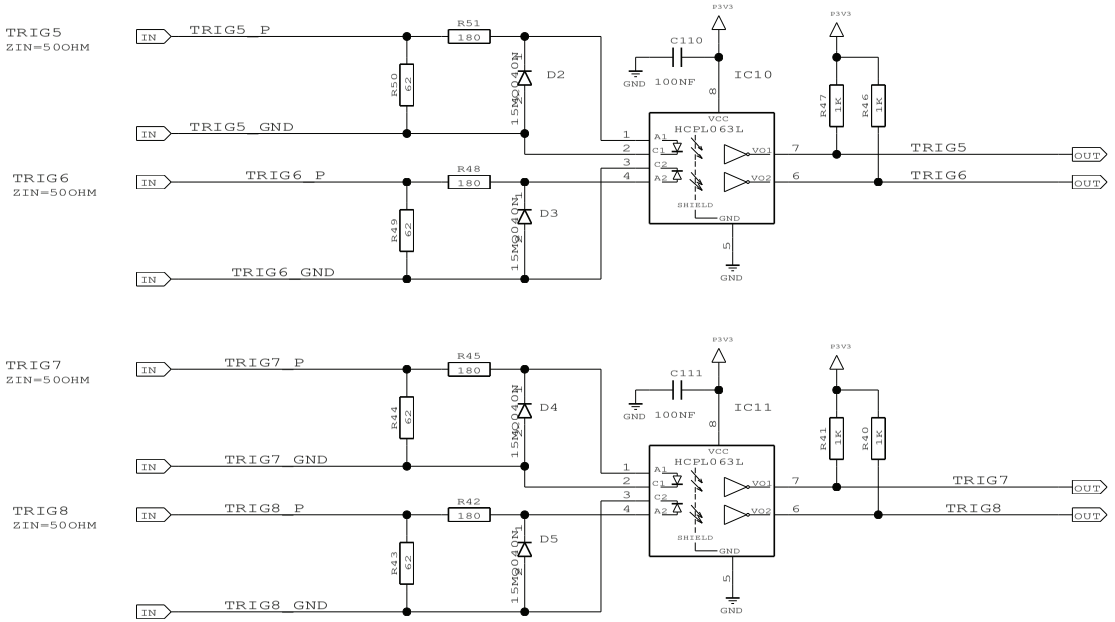
MODULE:switch_intfc Sheet: Module: 1/1 Project: 5/18

LAST_MODIFIED=Fri Nov 05 13:48:50 2010

Design by: [illegible]ch PCB by: JMC DATE: 26/08/10

CXXIV

OPTO-ISOLATION FOR CTRV TRIGGERS

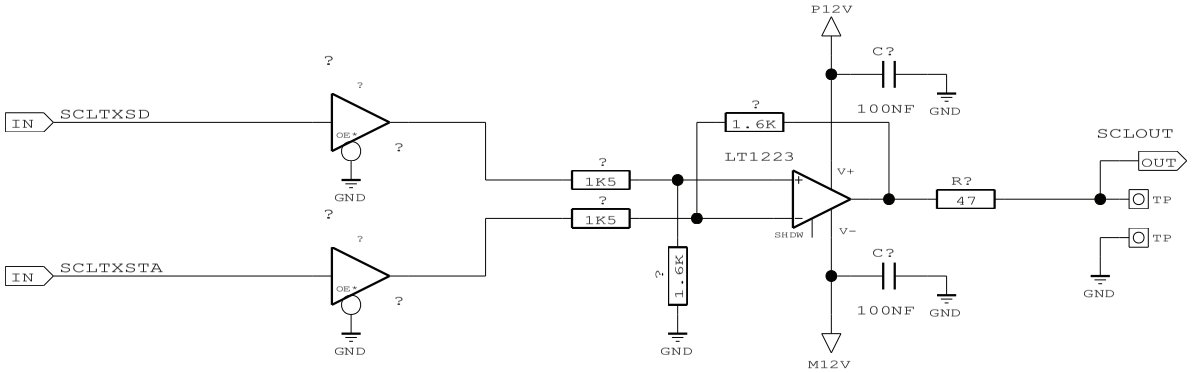


CXXV

Project file:spsphaseshifter.cpm		SYSTEM: SPB16.01
EDA-02163-V1-0	PROJECT:SERIAL LINK ROUTER	
European Organization for Nuclear Research 1211 GENEVA 23 SWITZERLAND	MODULE:fpintfce	Sheet: 1/1 (Module: / Project: 6/18)
EN/ICE	LAST_MODIFIED=Fri Nov 05 13:48:51 2010	
	Design by: D. Valuch / PCB by: JMC	DATE: 26/08/10

SERIAL CONTROL LINK TRANSMITTER

3.3V VERSION



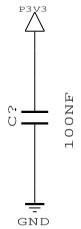
SERIAL LINK OUT:

BACK PANEL SMC

PCB CABLE TERMINAL

CXV1

1G125 DECOUPLING



Project file:spsphaseshifter.cpm SYSTEM: SPB16.01

EDA-02163-V1-0

European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND
TS/DEM

PROJECT:SERIAL LINK ROUTER

MODULE:transmitter Sheet: Module: 1/1 Project: 7/18

LAST_MODIFIED=Wed Jan 05 13:04:35 2011

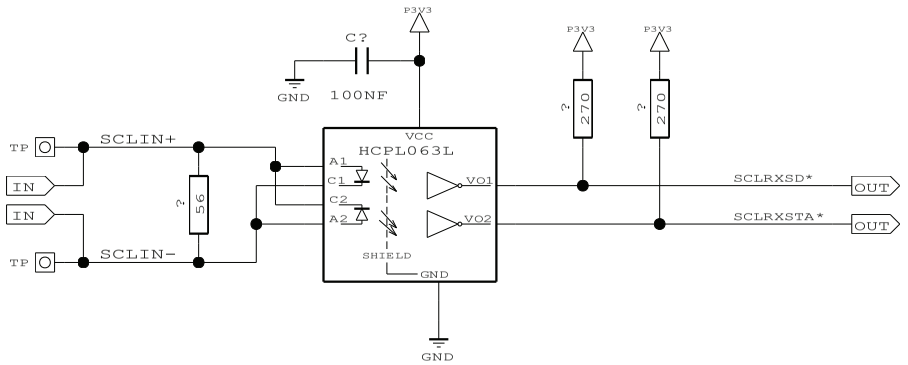
Design by: D. Valuch | PCB by: JMC | DATE: 26/08/10

SERIAL CONTROL LINK RECEIVER

SERIAL LINK IN:

BACK PANEL SMC

PCB CABLE TERMINAL



Project file:spsphaseshifter.cpm SYSTEM: SPB16.01

EDA-02163-V1-0

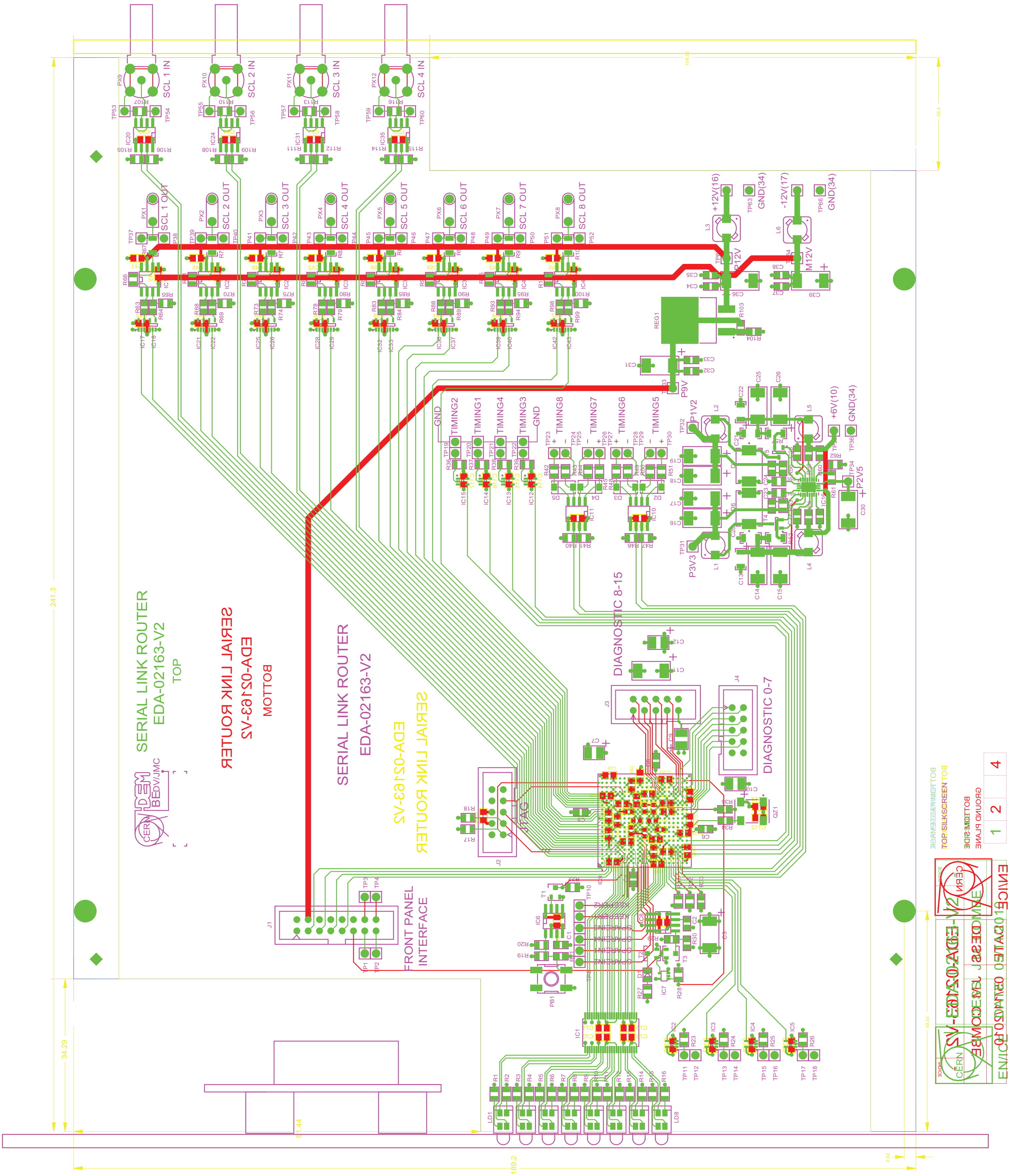
TS/DEM
CERN
European Organization
for Nuclear Research
1211 GENEVA 23
SWITZERLAND

PROJECT:SERIAL LINK ROUTER

MODULE:receiver Sheet: Module: 1/1 Project: 15/18

LAST_MODIFIED=Wed Jan 05 13:05:09 2011

Design by: D. Valuch | PCB by: JMC | DATE: 26/08/10

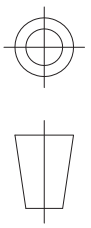

SERIAL LINK ROUTER
EDA-02163-V2
TOP
SERIAL LINK ROUTER
EDA-02163-V2
BOTTOM
SERIAL LINK ROUTER
EDA-02163-V2
FRONT PANEL INTERFACE
DIAGNOSTIC 8-15
DIAGNOSTIC 0-7
JTAG

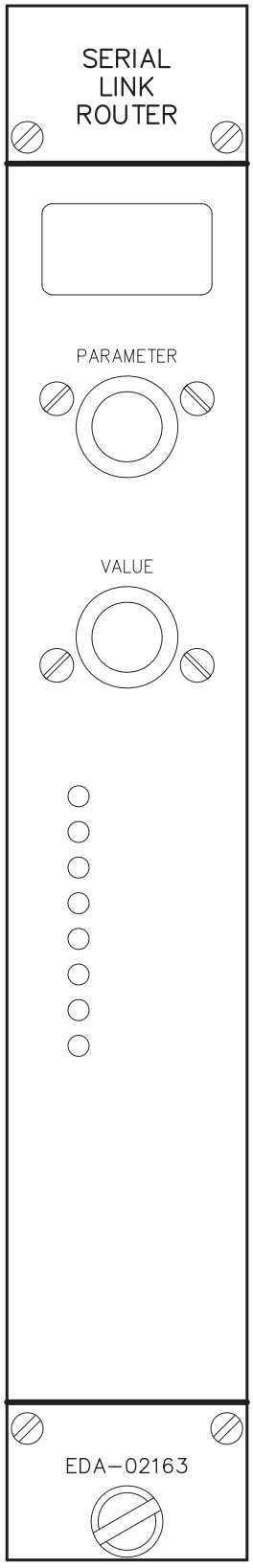
GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON NORMES ISO

DRAWING, RUGOSITY, TOLERANCES ACCORDING TO ISO STANDARDS

PROJECTION





CXXIX

EDA-02163 SERIAL LINK ROUTER

FRONT PANEL ARRANGEMENT

SCALE 1:1

DES JMC 01/09/10

MOD.-

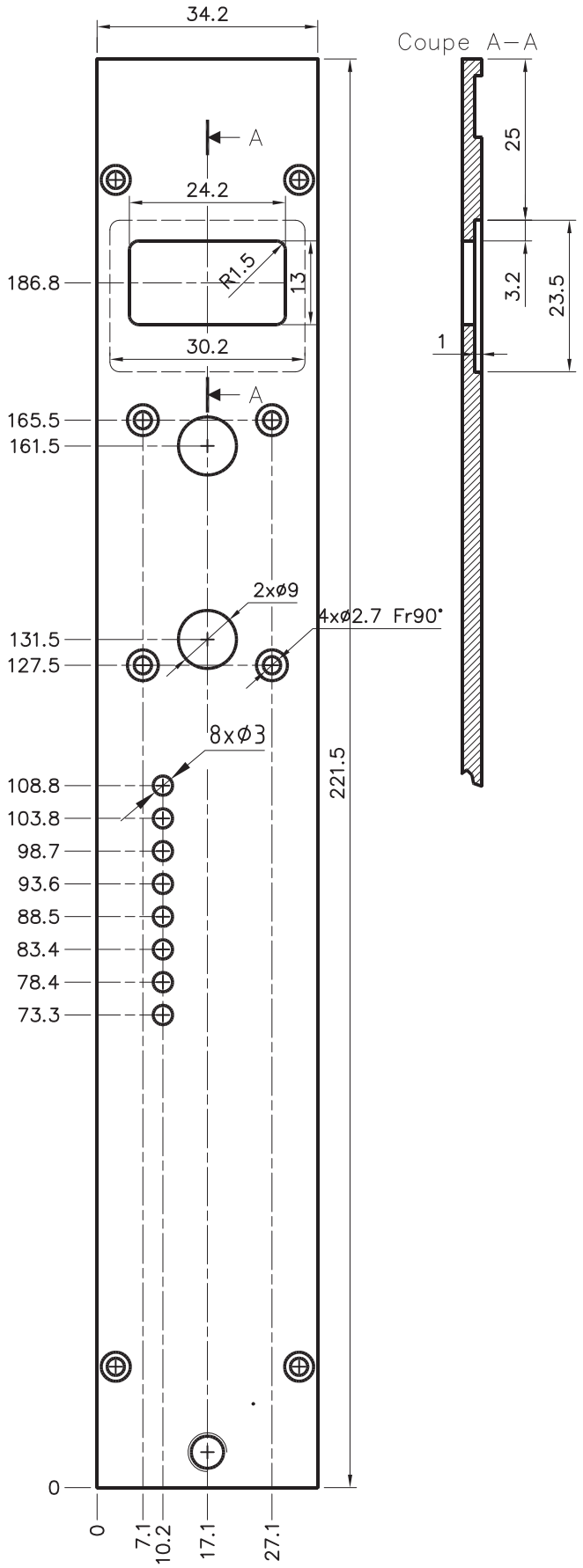
N° EDA-02163-V1-0_fp.dwg

EDA-02163-V1-0_fp-pres 4

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON **NORMES ISO**
DRAWING, RUGOSITY, TOLERANCES ACCORDING TO **ISO STANDARDS**

PROJECTION



PANNEAU FRONTAL 5Hx1L
(CHASSIS CIM 8905)

SCEM: 06.61.53.551.1

Traitement de Surface : Oxydation anodique (Anodisc CXXX

EDA-02163 SERIAL LINK ROUTER			
FRONT PANEL MECHANICAL	SCALE 1:1	DES JMC 03/09/10	
		MOD.-	
		N° EDA-02163-V1-0_fp.dwg	
EDA-02163-V1-0_fp-mech			4

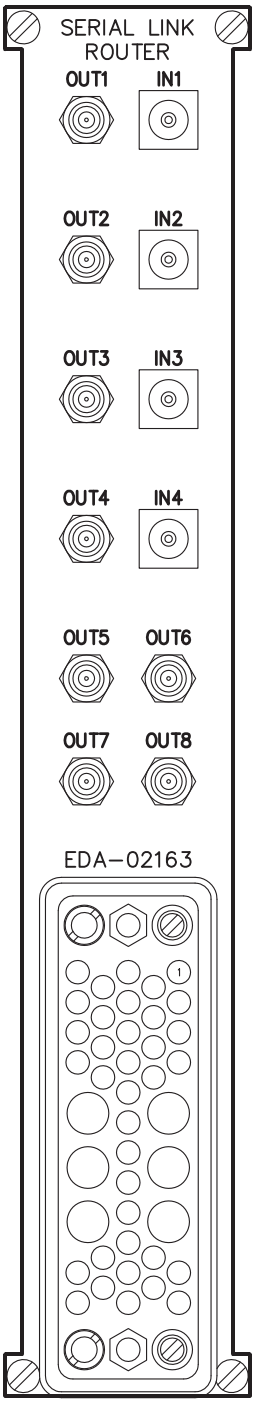
ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
GENEVE

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON **NORMES ISO**

DRAWING, RUGOSITY, TOLERANCES ACCORDING TO **ISO STANDARDS**

PROJECTION



ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
GENEVE

CXXXI

EDA–02163 SERIAL LINK ROUTER

REAR PANEL ARRANGEMENT

SCALE 1:1

DES JMC 01/09/10

MOD. JMC 07/10/10

N° EDA–02163–V1–0_rp.dwg

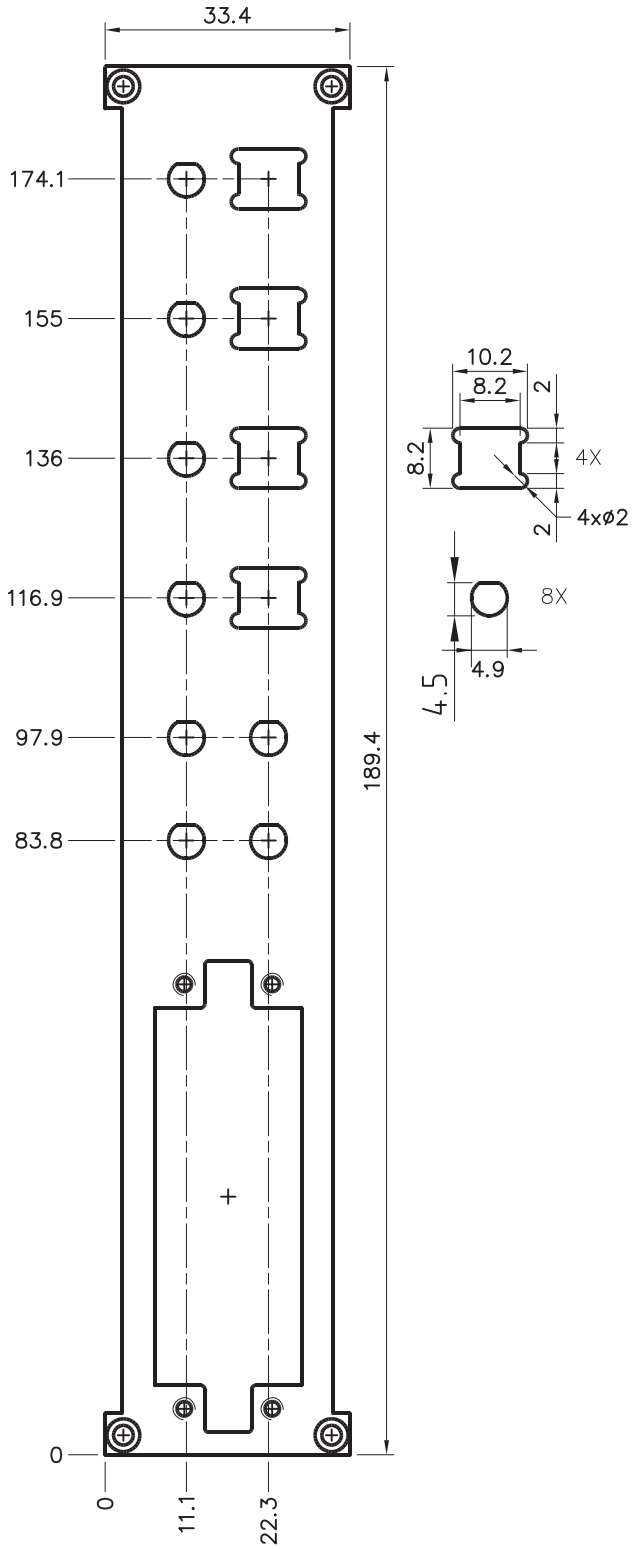
EDA–02163–V1–0_rp–pres 4

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON **NORMES ISO**

DRAWING, RUGOSITY, TOLERANCES ACCORDING TO **ISO STANDARDS**

PROJECTION



CXXXII

EDA-02163 SERIAL LINK ROUTER

REAR PANEL MECHANICAL

SCALE 1:1

DES	JMC	03/09/10
MOD.	JMC	05/10/10

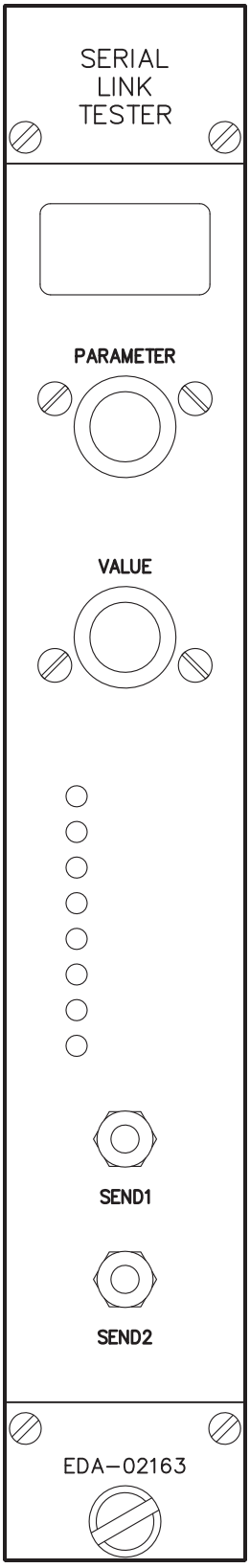
N° EDA-02163-V1-0_rp.dwg

EDA-02163-V1-0_rp-pres 4

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON **NORMES ISO**
DRAWING, RUGOSITY, TOLERANCES ACCORDING TO **ISO STANDARDS**

PROJECTION



ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
GENEVE

CXXXIII

EDA-02163 SERIAL LINK TESTER

FRONT PANEL ARRANGEMENT

SCALE 1:1

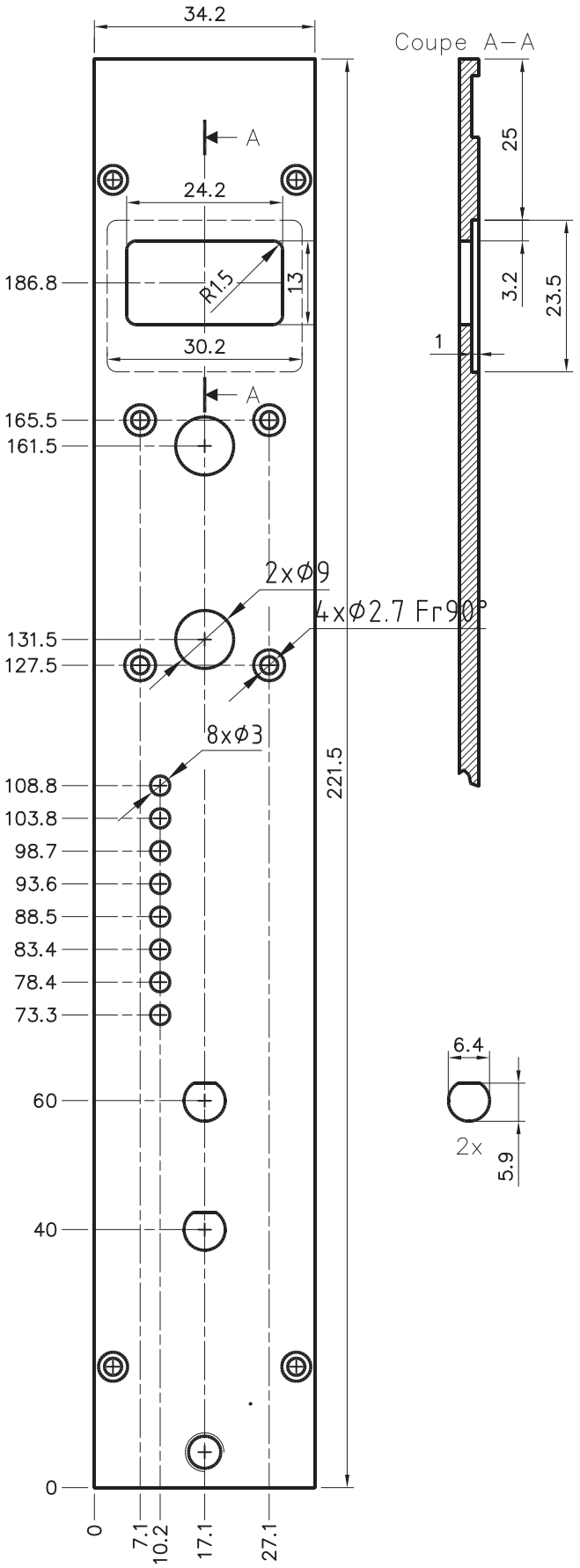
DES JMC 30/09/10
MOD.- JMC 07/10/10
N° EDA-02163-V1-1_fp.dwg

EDA-02163-V1-1_fp-pres 4

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON **NORMES ISO**
DRAWING, RUGOSITY, TOLERANCES ACCORDING TO **ISO STANDARDS**

PROJECTION



PANNEAU FRONTAL 5Hx1L
(CHASSIS CIM 8905)

SCEM: 06.61.53.551.1

Traitement de Surface : Oxydation anodique (Anodi CXXXIV

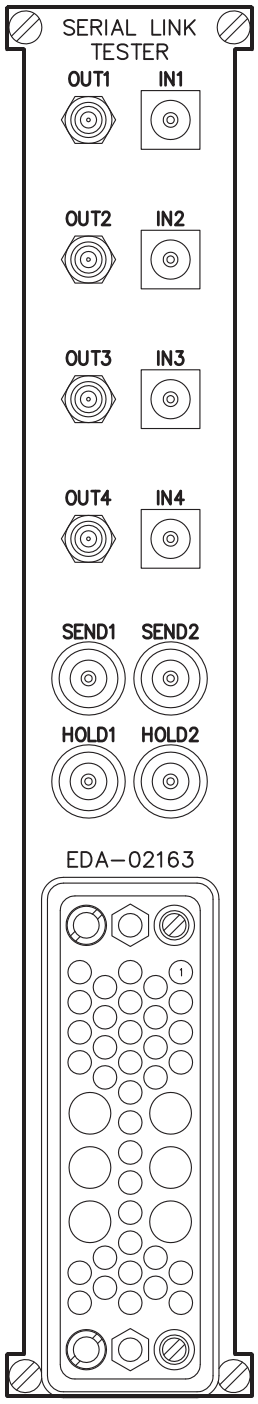
EDA-02163 SERIAL LINK TESTER			
FRONT PANEL MECHANICAL	SCALE 1:1	DES JMC 30/09/10	
		MOD.-	
		N° EDA-02163-V1-1_fp.dwg	
EDA-02163-V1-1_fp-mech			4

ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
GENEVE

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON **NORMES ISO**
DRAWING, RUGOSITY, TOLERANCES ACCORDING TO **ISO STANDARDS**

PROJECTION



CXXXV

EDA-02163 SERIAL LINK ROUTER

REAR PANEL ARRANGEMENT

SCALE 1:1

DES JMC 30/09/10

MOD.

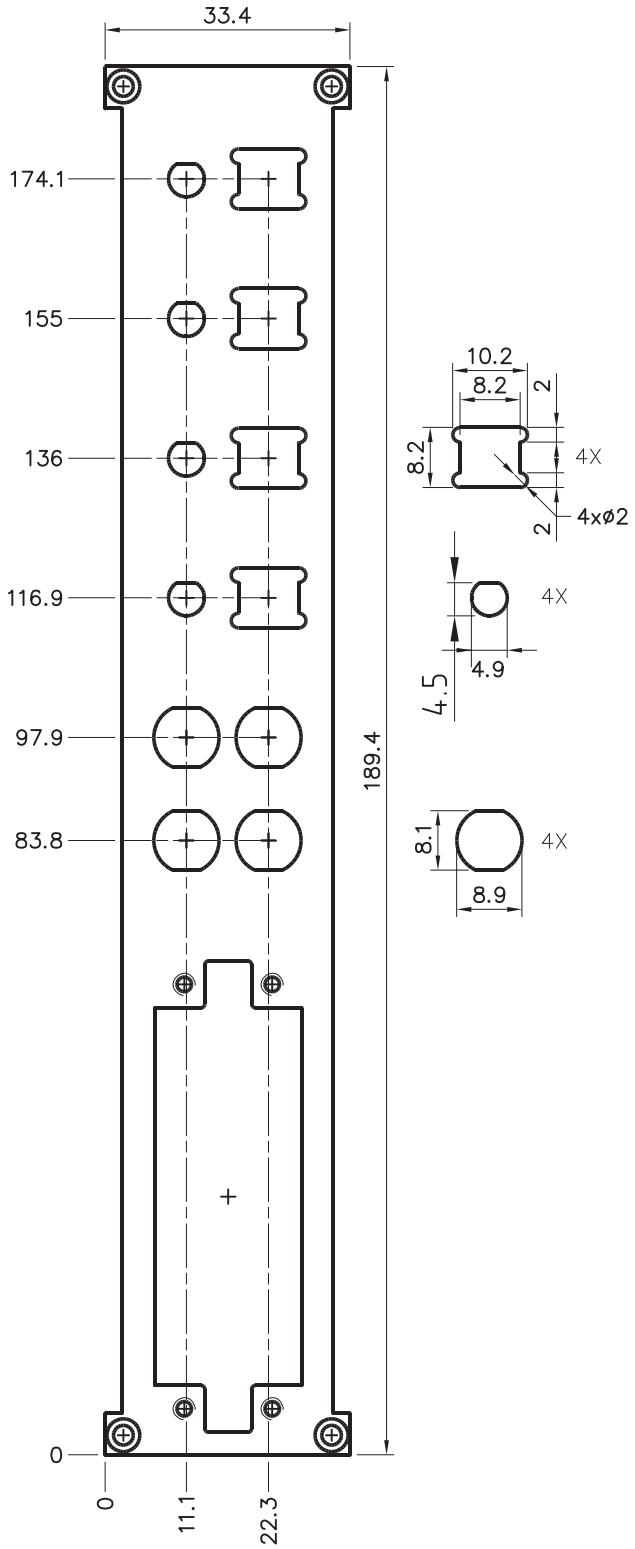
N° EDA-02163-V1-1_rp.dwg

EDA-02163-V1-1_rp-pres 4

GENERAL TOLERANCES GENERALES	DIMENSION	<=6	> 6	> 30	> 120	> 315	>1000	>2000
	USINAGE MOYEN/MEDIUM MACHINING	± 0.1	± 0.2	± 0.3	± 0.5	± 0.8	± 1.2	± 2
	MECANO. SOUDURE/WELDED STRUCTURE	± 0.5	± 1	± 2	± 3	± 5	± 7	± 10

DESSIN, RUGOSITE, TOLERANCES SELON **NORMES ISO**
DRAWING, RUGOSITY, TOLERANCES ACCORDING TO **ISO STANDARDS**

PROJECTION



EDA-02163 SERIAL LINK ROUTER

REAR PANEL MECHANICAL

SCALE 1:1

DES JMC 30/09/10

MOD.

N° EDA-02163-V1-1_rp.dwg

EDA-02163-V1-1_rp-pres 4

J Serial Link Router — Firmware

This appendix contains the firmware for the Serial Link Router module. Only blocks that are unique to this project are included, and are as follows:

SerialLinkRouter_Top CXXXVIII

Top level of the firmware.

KnobRegisters CXLII

Registers for VHI communication.

LED58Decoder CXLIII

Mapping from internal signals to LED pattern.

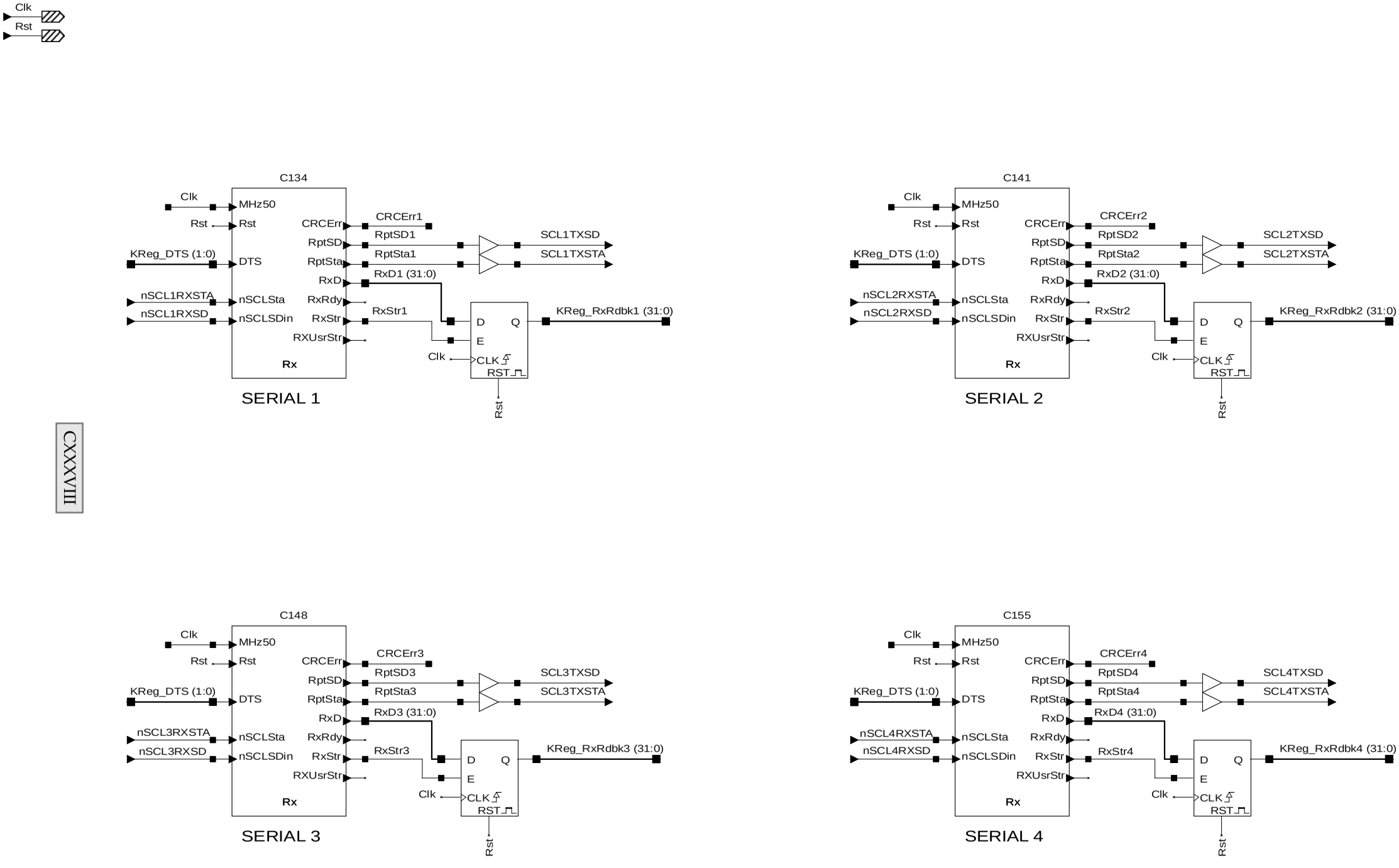
Rx CXLIV

Serial receiver that is switchable between SerialLink[16] (for the SPS) and Serial Control Link (for the LHC).

SerialLinkRouter_Pack CXLV

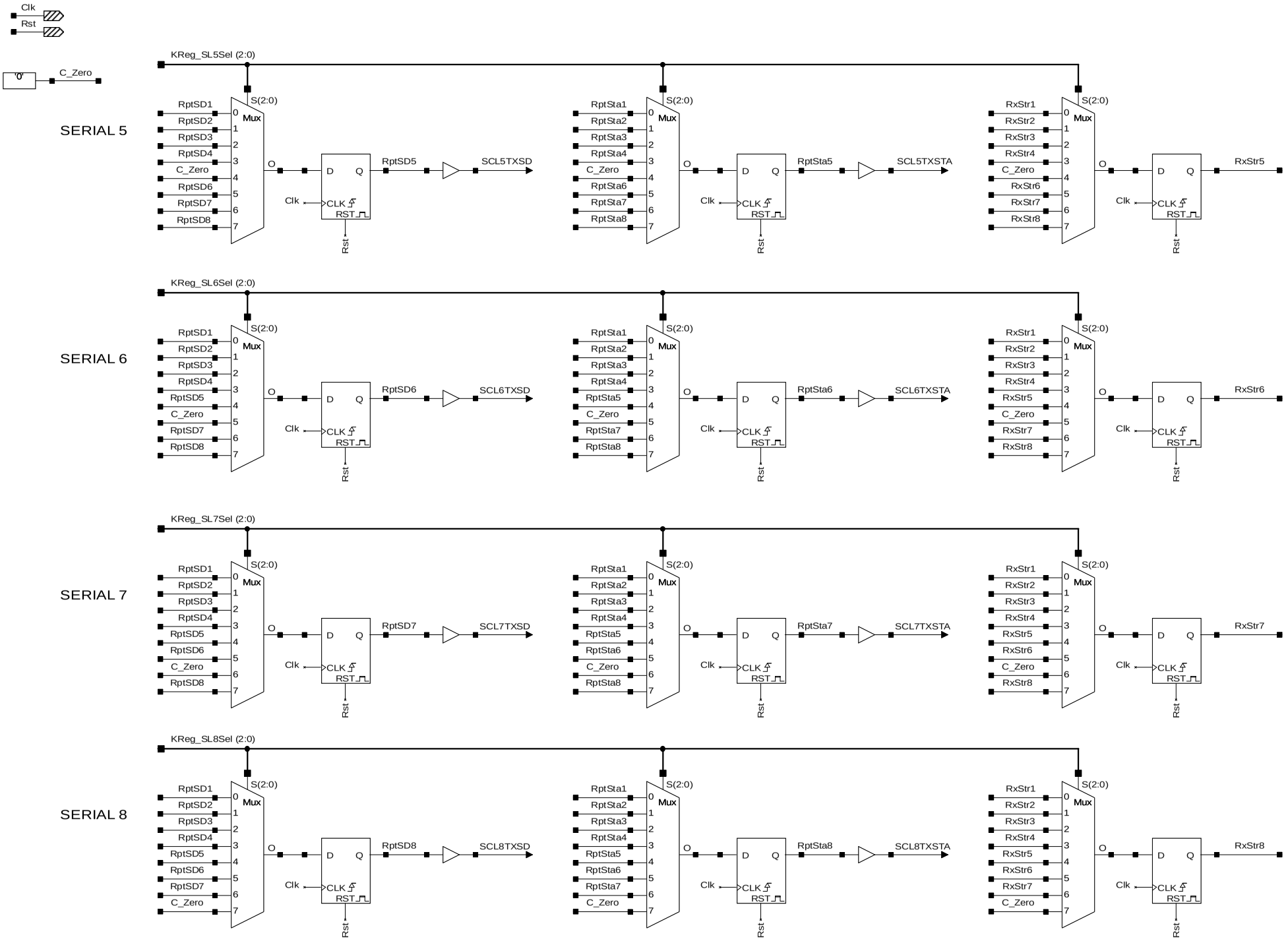
Register address definitions for the VHI.

This firmware for this module also relies upon the VHI SPI Controller (Appendix Q), the SerialLink[16] Controller (Appendix O) and blocks common to all modules (Appendix N). Any blocks not included from any of these sources are recycled from other projects and are not the work of the author.



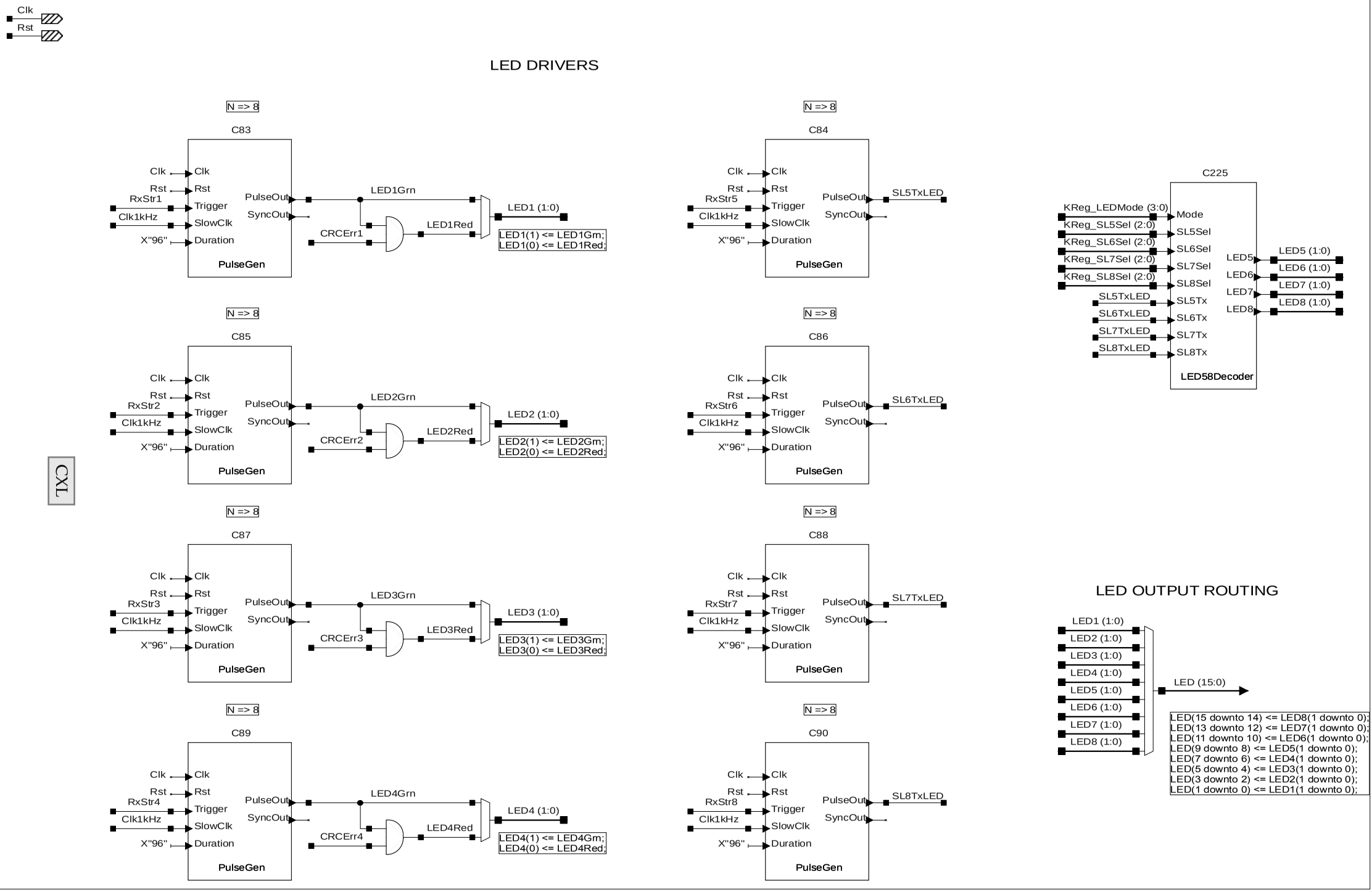
CXXXVIII

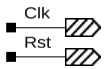
Design: SerialLinkRouter:SerialLinkRouter_Top (Sheet 1 of 4)	Date: Sun Dec 12 19:49:03 2010	Page: 1



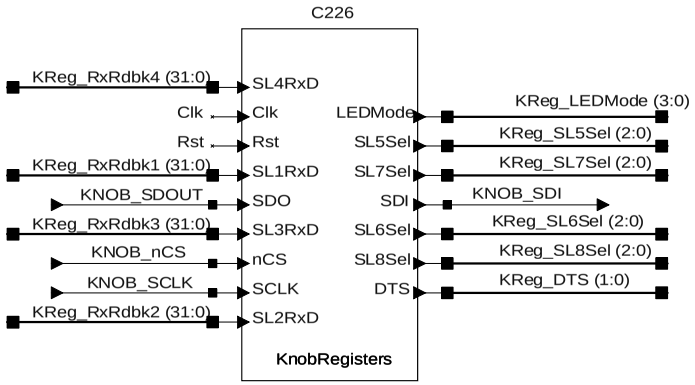
CXXXIX

Design: SerialLinkRouter:SerialLinkRouter_Top (Sheet 2 of 4)	Date: Sun Dec 12 19:49:03 2010	Page: 2

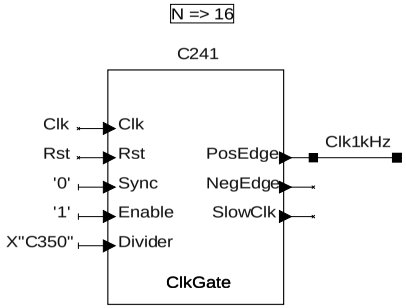

Clk
Rst
LED DRIVERS
N => 8
C83
Clk
Rst
RxStr1
Clk1kHz
X"96"
Trigger
SlowClk
Duration
PulseGen
PulseOut
SyncOut
LED1Grn
CRCErr1
LED1Red
LED1 (1:0)
LED1(1) <= LED1Gm;
LED1(0) <= LED1Red;
C84
RxStr5
SL5TxLED
C85
RxStr2
LED2Grn
CRCErr2
LED2Red
LED2 (1:0)
LED2(1) <= LED2Gm;
LED2(0) <= LED2Red;
C86
RxStr6
SL6TxLED
C87
RxStr3
LED3Grn
CRCErr3
LED3Red
LED3 (1:0)
LED3(1) <= LED3Gm;
LED3(0) <= LED3Red;
C88
RxStr7
SL7TxLED
C89
RxStr4
LED4Grn
CRCErr4
LED4Red
LED4 (1:0)
LED4(1) <= LED4Gm;
LED4(0) <= LED4Red;
C90
RxStr8
SL8TxLED
C225
KReg_LEDMode (3:0)
KReg_SL5Sel (2:0)
KReg_SL6Sel (2:0)
KReg_SL7Sel (2:0)
KReg_SL8Sel (2:0)
SL5TxLED
SL6TxLED
SL7TxLED
SL8TxLED
Mode
SL5Sel
SL6Sel
SL7Sel
SL8Sel
SL5Tx
SL6Tx
SL7Tx
SL8Tx
LED58Decoder
LED5
LED6
LED7
LED8
LED5 (1:0)
LED6 (1:0)
LED7 (1:0)
LED8 (1:0)
LED OUTPUT ROUTING
LED1 (1:0)
LED2 (1:0)
LED3 (1:0)
LED4 (1:0)
LED5 (1:0)
LED6 (1:0)
LED7 (1:0)
LED8 (1:0)
LED (15:0)
LED(15 downto 14) <= LED8(1 downto 0);
LED(13 downto 12) <= LED7(1 downto 0);
LED(11 downto 10) <= LED6(1 downto 0);
LED(9 downto 8) <= LED5(1 downto 0);
LED(7 downto 6) <= LED4(1 downto 0);
LED(5 downto 4) <= LED3(1 downto 0);
LED(3 downto 2) <= LED2(1 downto 0);
LED(1 downto 0) <= LED1(1 downto 0);
CXL



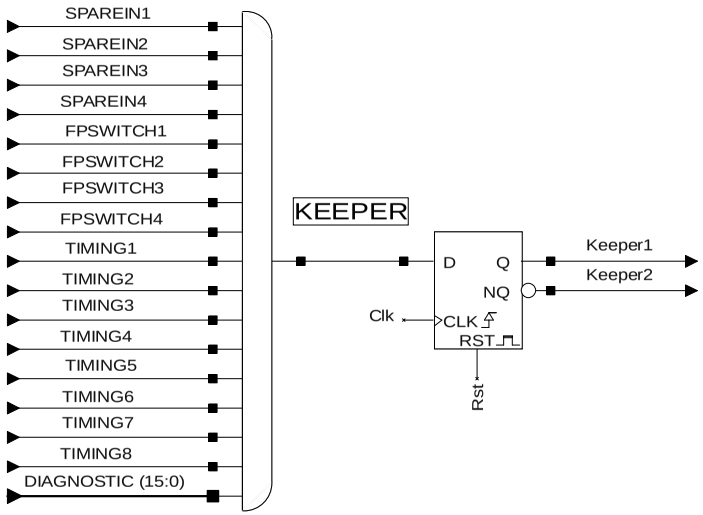
KNOB SPI COMMS



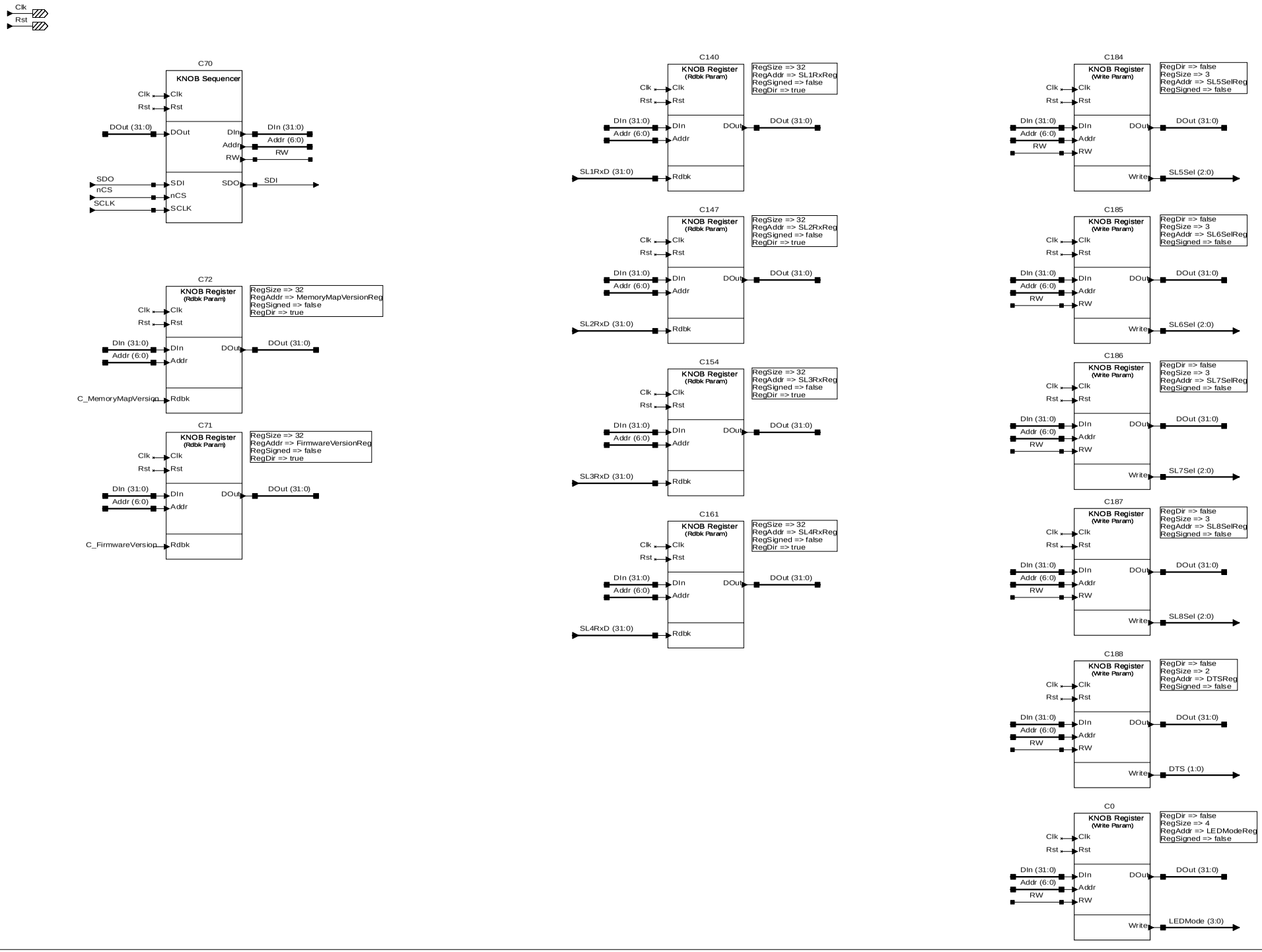
1kHz CLK GENERATOR



CXLI



Design: SerialLinkRouter:SerialLinkRouter_Top (Sheet 4 of 4)	Date: Sun Dec 12 19:49:03 2010	Page: 4



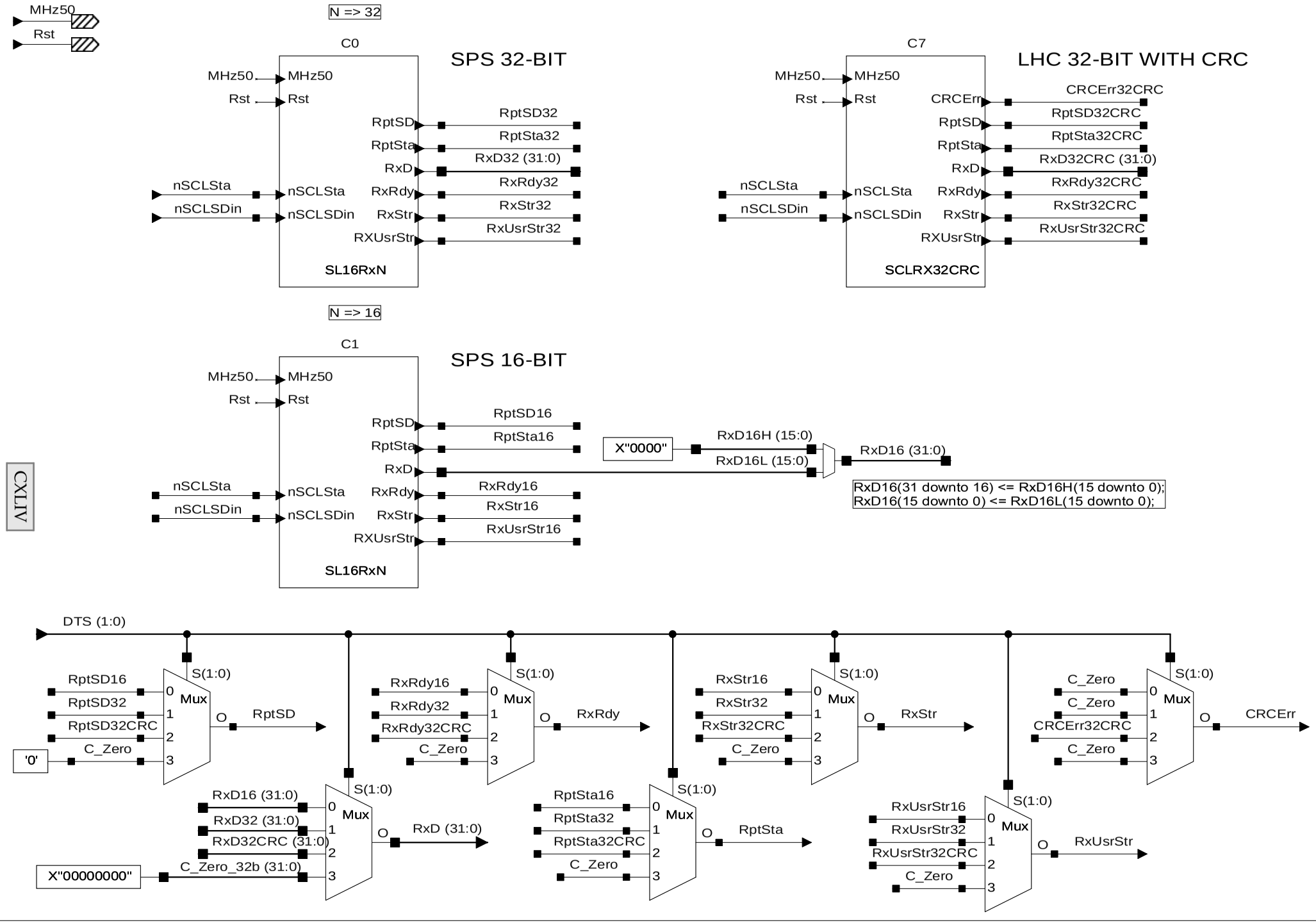
Design: SerialLinkRouter:KnobRegisters	Date: Fri Dec 03 15:19:34 2010	Page: 1

Truth Table: LED58Decoder

Mode	SL5Sel	SL6Sel	SL7Sel	SL8Sel	LED5	LED6	LED7	LED8
X"0"					SL5Tx & '0'	SL6Tx & '0'	SL7Tx & '0'	SL8Tx & '0'
X"1"	"000"				"10"	"00"	"00"	"00"
X"1"	"001"				"00"	"10"	"00"	"00"
X"1"	"010"				"00"	"00"	"10"	"00"
X"1"	"011"				"00"	"00"	"00"	"10"
X"1"	"100"				"01"	"00"	"00"	"00"
X"1"	"101"				"00"	"11"	"00"	"00"
X"1"	"110"				"00"	"00"	"11"	"00"
X"1"	"111"				"00"	"00"	"00"	"11"
X"2"		"000"			"10"	"00"	"00"	"00"
X"2"		"001"			"00"	"10"	"00"	"00"
X"2"		"010"			"00"	"00"	"10"	"00"
X"2"		"011"			"00"	"00"	"00"	"10"
X"2"		"100"			"11"	"00"	"00"	"00"
X"2"		"101"			"00"	"01"	"00"	"00"
X"2"		"110"			"00"	"00"	"11"	"00"
X"2"		"111"			"00"	"00"	"00"	"11"
X"3"			"000"		"10"	"00"	"00"	"00"
X"3"			"001"		"00"	"10"	"00"	"00"
X"3"			"010"		"00"	"00"	"10"	"00"
X"3"			"011"		"00"	"00"	"00"	"10"
X"3"			"100"		"11"	"00"	"00"	"00"
X"3"			"101"		"00"	"11"	"00"	"00"
X"3"			"110"		"00"	"00"	"01"	"00"
X"3"			"111"		"00"	"00"	"00"	"11"
X"4"				"000"	"10"	"00"	"00"	"00"
X"4"				"001"	"00"	"10"	"00"	"00"
X"4"				"010"	"00"	"00"	"10"	"00"
X"4"				"011"	"00"	"00"	"00"	"10"
X"4"				"100"	"11"	"00"	"00"	"00"
X"4"				"101"	"00"	"11"	"00"	"00"
X"4"				"110"	"00"	"00"	"11"	"00"
X"4"				"111"	"00"	"00"	"00"	"01"
					"00"	"00"	"00"	"00"

Packages Used
ieee.STD_LOGIC_1164

Interface
Mode : in std_logic_vector (3 downto 0);
SL5Sel : in std_logic_vector (2 downto 0);
SL6Sel : in std_logic_vector (2 downto 0);
SL7Sel : in std_logic_vector (2 downto 0);
SL8Sel : in std_logic_vector (2 downto 0);
LED5 : out std_logic_vector (1 downto 0);
LED6 : out std_logic_vector (1 downto 0);
LED7 : out std_logic_vector (1 downto 0);
LED8 : out std_logic_vector (1 downto 0);
SL5Tx : in std_logic ;
SL6Tx : in std_logic ;
SL7Tx : in std_logic ;
SL8Tx : in std_logic ;



Design: SerialLinkRouter:Rx	Date: Fri Dec 03 15:20:57 2010	Page: 1

J.5 SerialLinkRouter_Pack

```
------------------------------------------
------------------------------------------
-- Date        : Fri Sep 03 15:51:08 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Registers for Serial Link Router.
--
------------------------------------------
------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

package  SerialLinkRouter_Pack  is
  -- Firmware version code.
  constant    C_FirmwareVersion   : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(20101212, 32));

  -- KNOB memory-map version code.
  constant    C_MemoryMapVersion  : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(20101115, 32));

  -- KNOB read-back registers.
  constant    FirmwareVersionReg  : std_logic_vector(6 downto 0)  := "0000000"; -- 00h
  constant    MemoryMapVersionReg : std_logic_vector(6 downto 0)  := "0000001"; -- 01h
  constant    SL1RxReg            : std_logic_vector(6 downto 0)  := "0010000"; -- 10h
  constant    SL2RxReg            : std_logic_vector(6 downto 0)  := "0010001"; -- 11h
  constant    SL3RxReg            : std_logic_vector(6 downto 0)  := "0010010"; -- 12h
  constant    SL4RxReg            : std_logic_vector(6 downto 0)  := "0010011"; -- 13h
  constant    SL5SelReg           : std_logic_vector(6 downto 0)  := "0010100"; -- 14h
  constant    SL6SelReg           : std_logic_vector(6 downto 0)  := "0010101"; -- 15h
  constant    SL7SelReg           : std_logic_vector(6 downto 0)  := "0010110"; -- 16h
  constant    SL8SelReg           : std_logic_vector(6 downto 0)  := "0010111"; -- 17h
  constant    DTSReg              : std_logic_vector(6 downto 0)  := "0011000"; -- 18h
  constant    LEDModeReg          : std_logic_vector(6 downto 0)  := "0011001"; -- 19h
end;

-- EOF
```

K Serial Link Router — VHI Parameter Definition

```
#include "parameters.h"

volatile parameter r0 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  UNSIGNED,                   // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Firmware Version",         // unsigned char name[16]
  "",                         // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x00,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r1 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-1 Rx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x10,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r2 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-2 Rx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x11,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r3 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-3 Rx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x12,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};
```

```
volatile parameter r4 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-4 Rx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x13,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

const plist_item vlist1[] = {
  LIST_STARTING_STRING,
  "SL-1",
  "SL-2",
  "SL-3",
  "SL-4",
  "SL-5",
  "SL-6",
  "SL-7",
  "SL-8",
  LIST_TERMINATION_STRING
};

volatile parameter w1 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-5 Source",              // unsigned char name[16]
  "",                         // unsigned char comment[16]
  4,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  7.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x14,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist1[0])      // address of the referenced list array
};

volatile parameter w2 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-6 Source",              // unsigned char name[16]
  "",                         // unsigned char comment[16]
  4,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  7.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x15,                       // address (INT8)
  4.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist1[0])      // address of the referenced list array
};

volatile parameter w3 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-7 Source",              // unsigned char name[16]
  "",                         // unsigned char comment[16]
  4,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  7.0,                        // range_max (FLOAT)
```

```
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x16,                       // address (INT8)
  4.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist1[0])      // address of the referenced list array
};

volatile parameter w4 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-8 Source",              // unsigned char name[16]
  "",                         // unsigned char comment[16]
  4,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  7.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x17,                       // address (INT8)
  4.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist1[0])      // address of the referenced list array
};

const plist_item vlist2[] = {
  LIST_STARTING_STRING,
  "SPS-16b",
  "SPS-32b",
  "LHC-32b",
  LIST_TERMINATION_STRING
};

volatile parameter w5 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL Type/Length",           // unsigned char name[16]
  "",                         // unsigned char comment[16]
  7,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  2.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x18,                       // address (INT8)
  2.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist2[0])      // address of the referenced list array
};

const plist_item vlist3[] = {
  LIST_STARTING_STRING,
  "SL-5..8 Tx",
  "SL-5 Conns",
  "SL-6 Conns",
  "SL-7 Conns",
  "SL-8 Conns",
  LIST_TERMINATION_STRING
};

volatile parameter w6 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Ouput LED Func",           // unsigned char name[16]
  "",                         // unsigned char comment[16]
  10,                         // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  4.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x19,                       // address (INT8)
  0.0,                        // default/safe value
```

```
  1,                          // instant write
  (unsigned)(&vlist3[0])      // address of the referenced list array
};

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++// Keep this
  parameter for proper MENU FPGA WRITE definition.
// Mandatory setting is only BASIS = PARAM_MENU, other vars are ignored for now
// (KNOB version 1.0)
volatile parameter write_menu = {
  PARAM_MENU,                 // basis (PARAM_EDIT, PARAM_READBACK)
  FPGA_WRITE,                 // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "FPGA STORE",               // unsigned char name[16]
  "Push to store parameters", // unsigned char comment[16]
  0,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  0.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x00,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#define MAX_PARAMETERS  (sizeof(plist)/4)

// parameters list
volatile parameter *plist[] = { &r1, &r2, &r3, &r4,
                                &w1, &w2, &w3, &w4, &w5, &w6,
                                &write_menu,
                                &r0 };

//EOF
```

L Serial Link Tester — Firmware

This appendix contains the firmware for the Serial Link Tester module. Only blocks unique to this project are included in this appendix. The blocks included are as follows:

SerialLinkTester_Top CLII

Top level of the firmware.

KnobRegisters CLV

Registers for VHI communication.

Rx CLVI

Serial receiver that is switchable between SerialLink^{16} (for the SPS) and Serial Control Link (for the LHC).

SerialLinkTester_Pack CLVII

Register address definitions for the VHI.

Tx CLVIII

Serial transmitter that is switchable between SerialLink^{16} (for the SPS) and Serial Control Link (for the LHC).

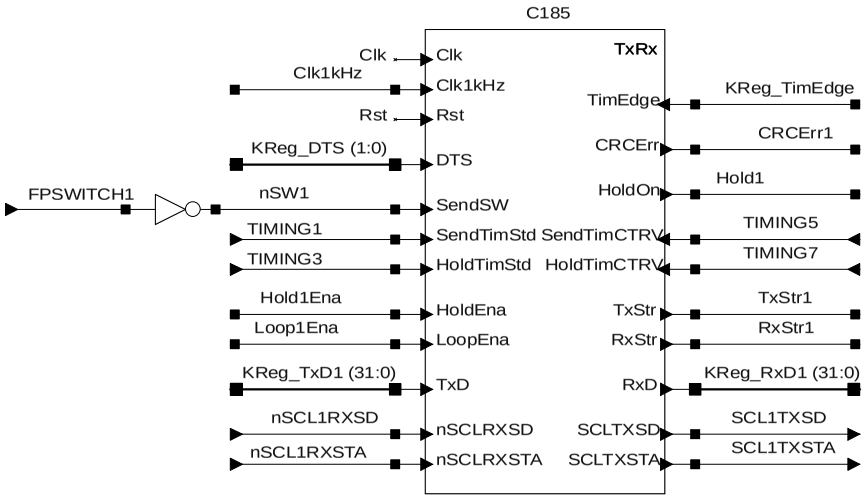
TxRx CLIX

Wrapper for transmitter and receiver which implements the hold, loop and send functionality.

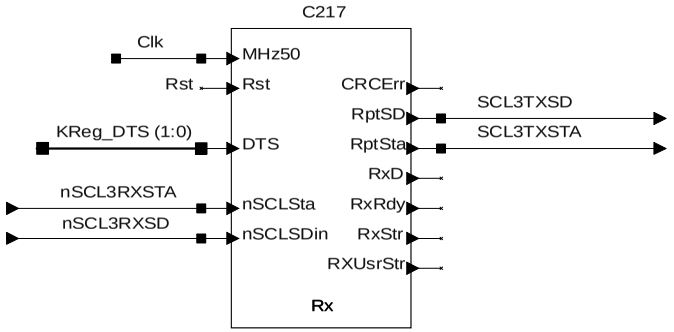
This firmware for this module also relies upon the VHI SPI Controller (Appendix Q), the SerialLink^{16} Controller (Appendix O) and blocks common to all modules (Appendix N). Any blocks not included from any of these sources are recycled from other projects and are not the work of the author.

Clk
Rst

SERIAL 1

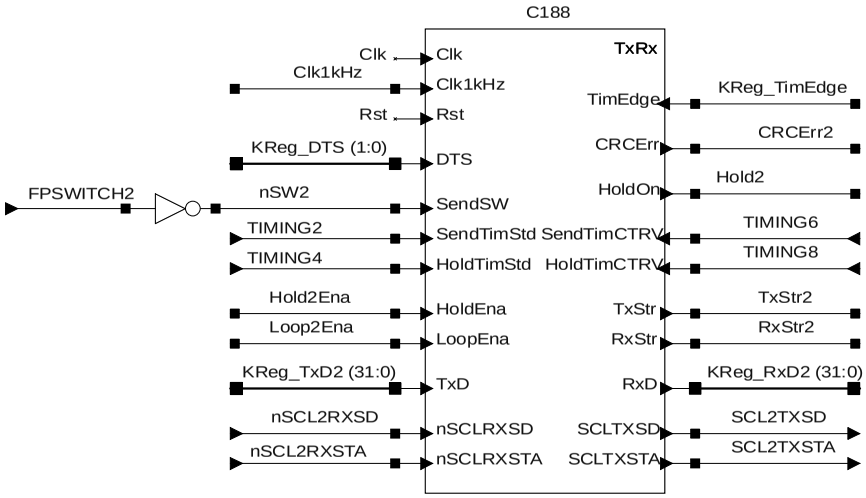


SERIAL 3

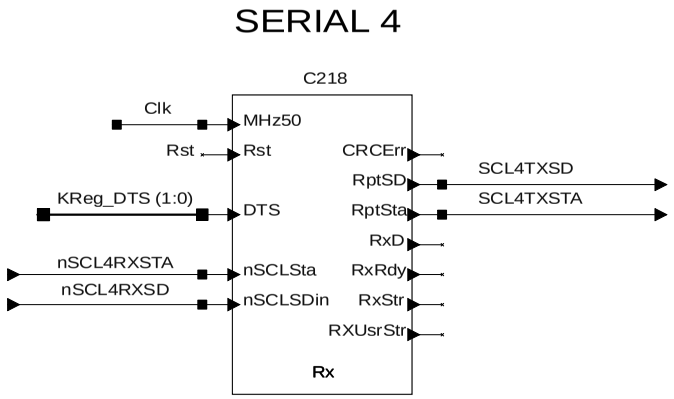


CLII

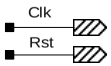
SERIAL 2



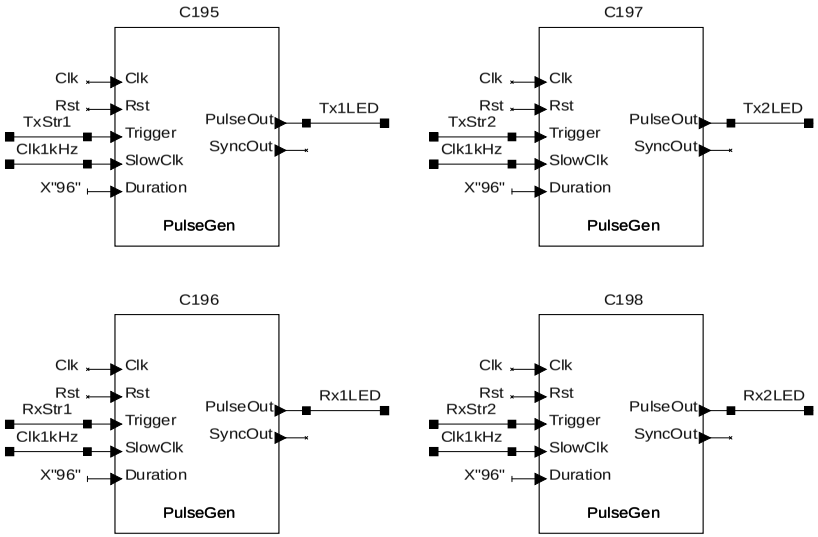
SERIAL 4



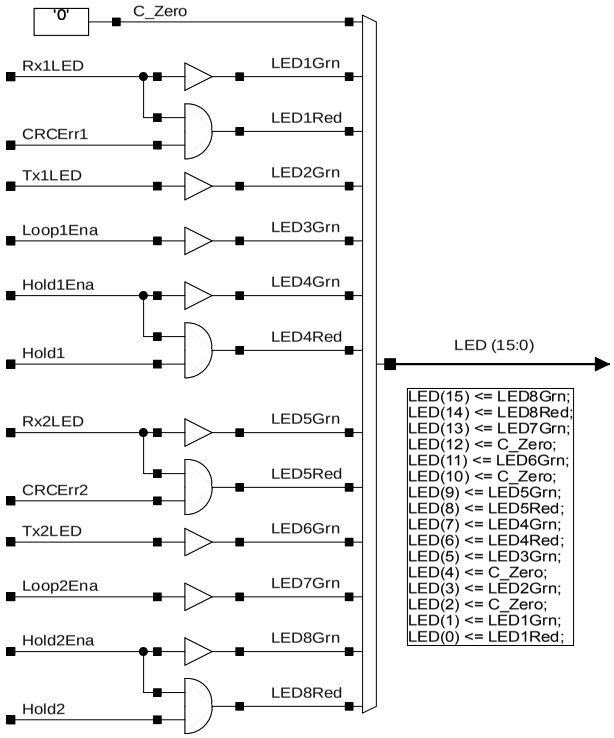
Design: SerialLinkTester:SerialLinkTester_Top (Sheet 1 of 3)	Date: Mon Dec 13 09:18:10 2010	Page: 1



150ms LED DRIVERS

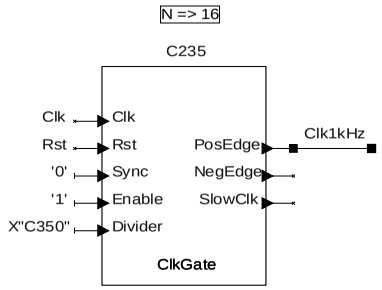


LED OUTPUT ROUTING

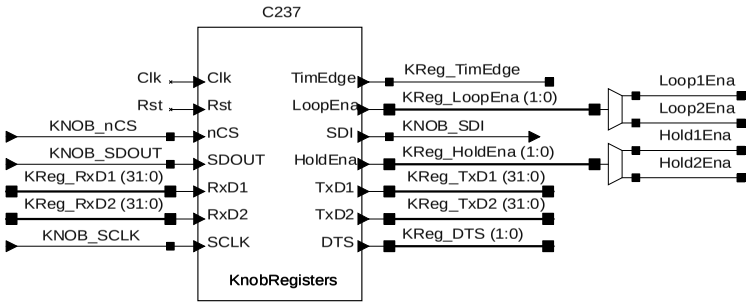


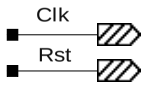
CLIII

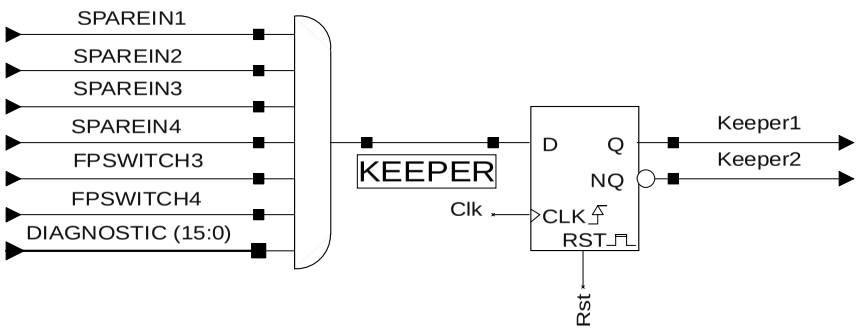
1kHz CLK GENERATOR



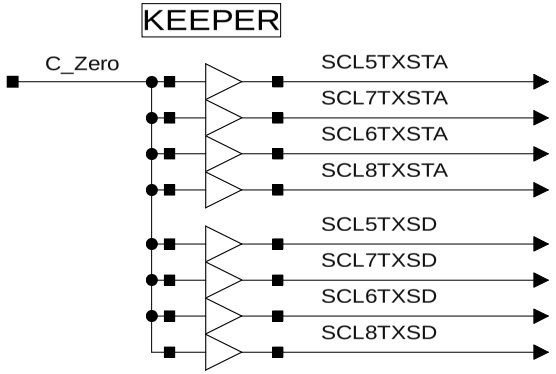
KNOB REGISTERS



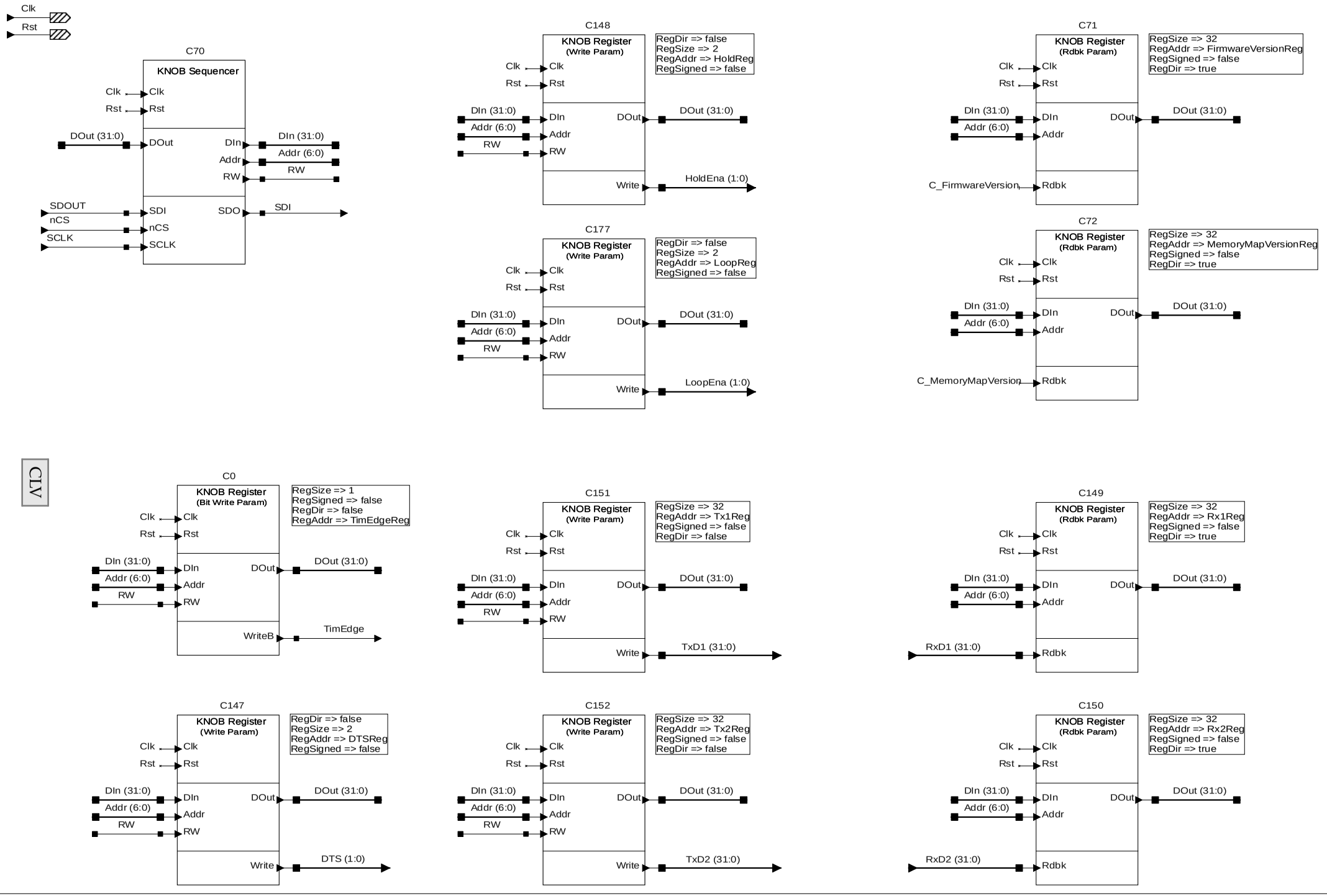




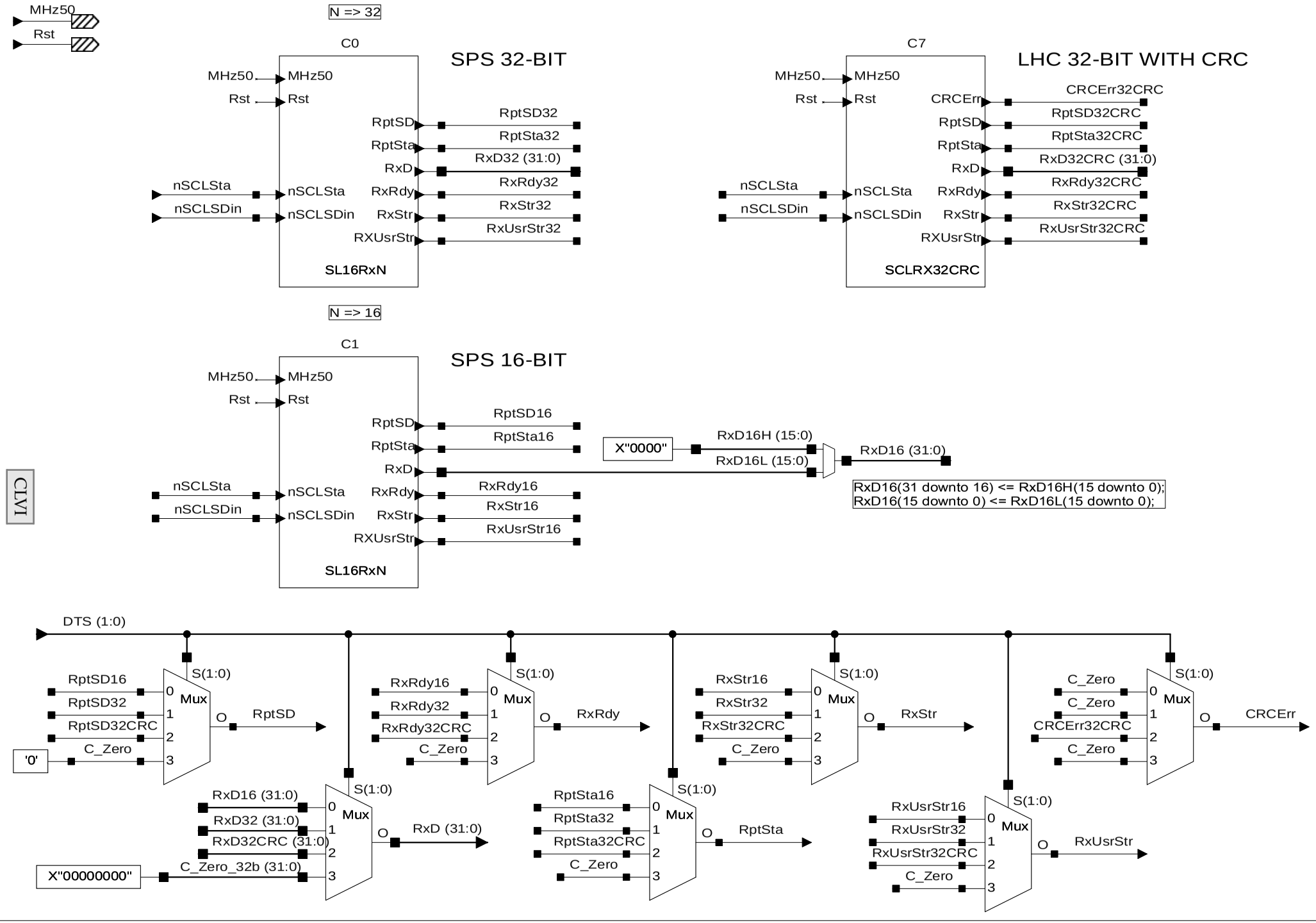
CLIV



Design: SerialLinkTester:SerialLinkTester_Top (Sheet 3 of 3)	Date: Mon Dec 13 09:18:10 2010	Page: 3



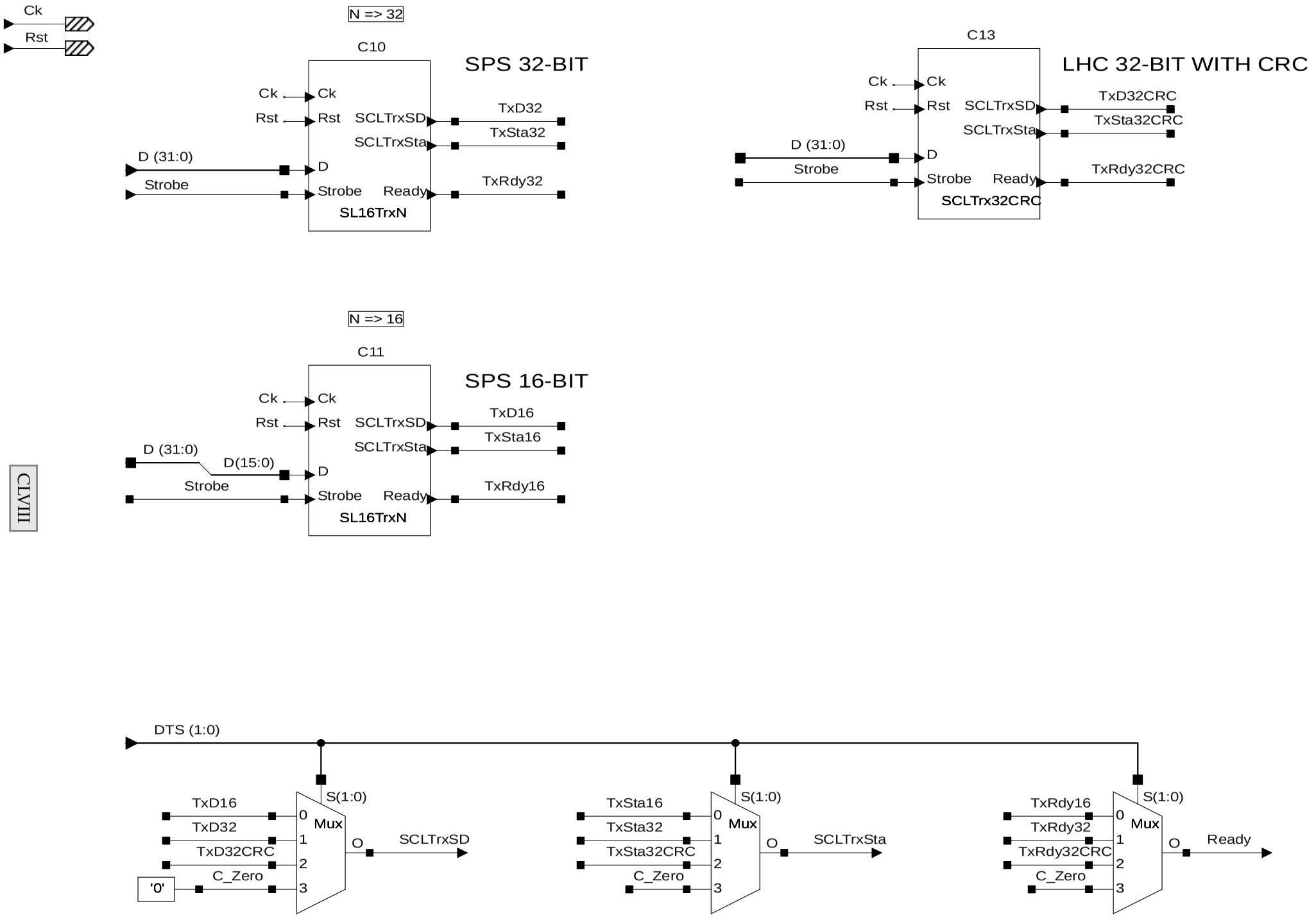
Design: SerialLinkTester:KnobRegisters	Date: Sun Dec 12 19:19:56 2010	Page: 1



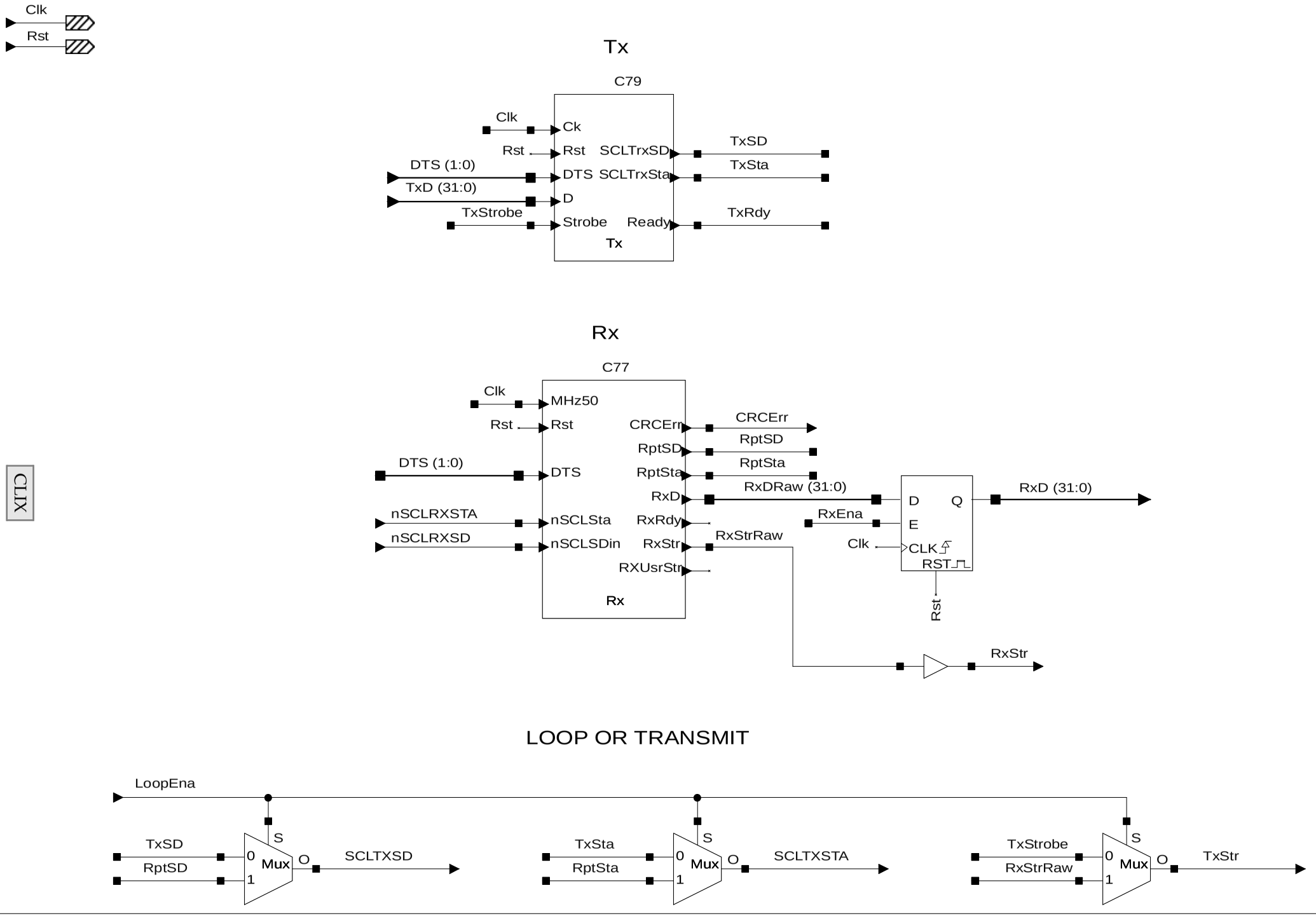
Design: SerialLinkTester:Rx	Date: Fri Dec 03 16:13:04 2010	Page: 1

L.4 SerialLinkTester_Pack

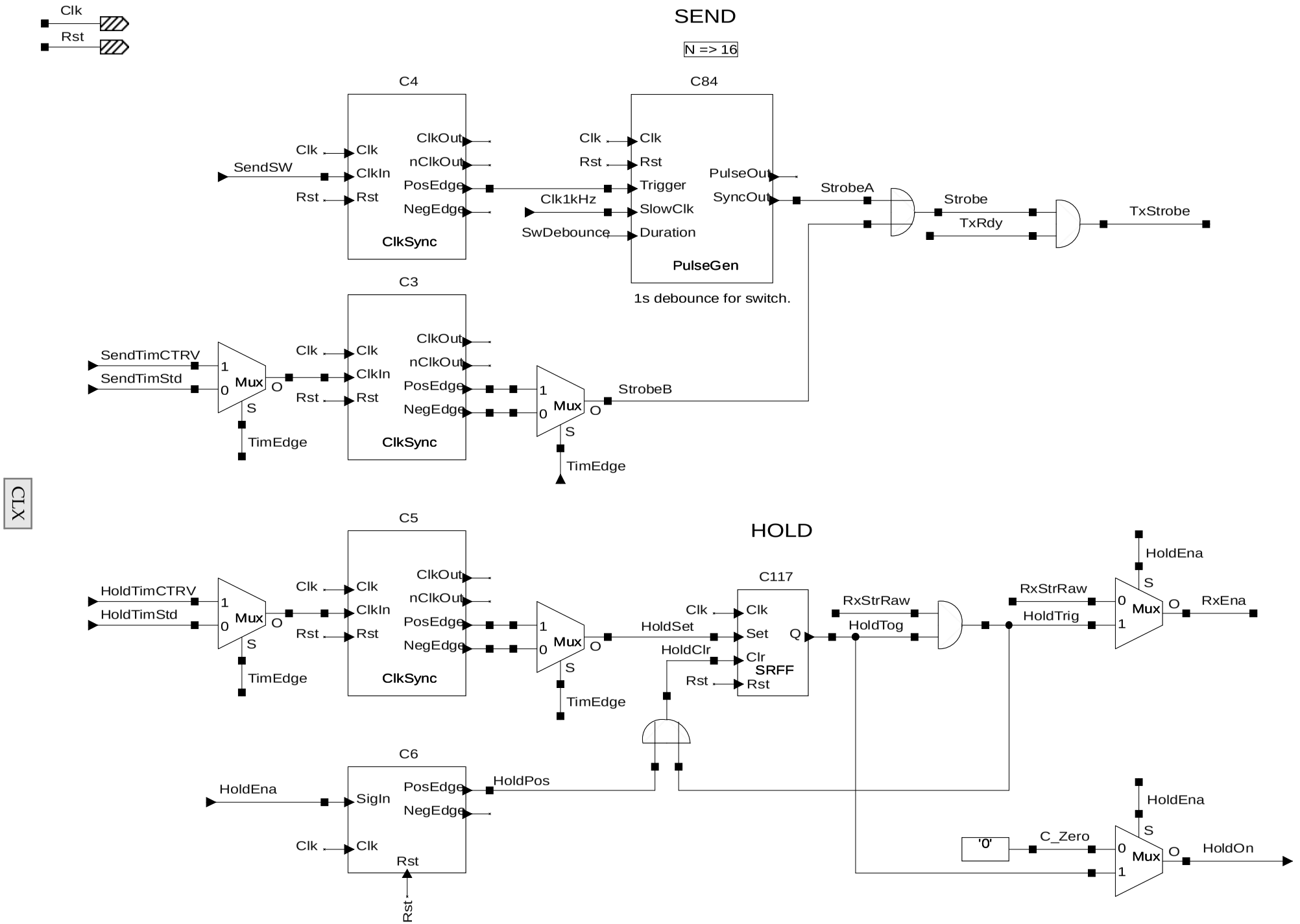
```
------------------------------------------
------------------------------------------
-- Date        : Fri Sep 03 15:51:08 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Register definitions for Serial Link Tester.
--
------------------------------------------
------------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

package  SerialLinkTester_Pack  is
  -- Firmware version code.
  constant    C_FirmwareVersion   : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(20101212, 32));

  -- KNOB memory-map version code.
  constant    C_MemoryMapVersion  : std_logic_vector(31 downto 0) := std_logic_vector(
    to_unsigned(20101212, 32));

  -- KNOB read-back registers.
  constant    FirmwareVersionReg  : std_logic_vector(6 downto 0)  := "0000000"; -- 00h
  constant    MemoryMapVersionReg : std_logic_vector(6 downto 0)  := "0000001"; -- 01h
  constant    Rx1Reg              : std_logic_vector(6 downto 0)  := "0010001"; -- 11h
  constant    Rx2Reg              : std_logic_vector(6 downto 0)  := "0010011"; -- 13h

  -- KNOB write registers.
  constant    Tx1Reg              : std_logic_vector(6 downto 0)  := "0010000"; -- 10h
  constant    Tx2Reg              : std_logic_vector(6 downto 0)  := "0010010"; -- 12h
  constant    DTSReg              : std_logic_vector(6 downto 0)  := "0010100"; -- 14h
  constant    HoldReg             : std_logic_vector(6 downto 0)  := "0010101"; -- 15h
  constant    LoopReg             : std_logic_vector(6 downto 0)  := "0010110"; -- 16h
  constant    TimEdgeReg          : std_logic_vector(6 downto 0)  := "0010111"; -- 17h

  -- Other constants.
  constant    SwDebounce          : std_logic_vector(15 downto 0) := X"03E8"; -- 1s

end;
```



Design: SerialLinkTester:Tx	Date: Fri Dec 03 16:14:18 2010	Page: 1



Design: SerialLinkTester:TxRx (Sheet 1 of 2)	Date: Mon Dec 13 09:17:32 2010	Page: 1



Design: SerialLinkTester:TxRx (Sheet 2 of 2)	Date: Mon Dec 13 09:17:32 2010	Page: 2

M Serial Link Tester — VHI Parameter Definition

```
#include "parameters.h"

volatile parameter r0 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  UNSIGNED,                   // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Firmware Version",         // unsigned char name[16]
  "",                         // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x00,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r1 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-1 Rx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x11,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter r2 = {
  PARAM_READBACK,             // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-2 Rx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x13,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};

volatile parameter w1 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-1 Tx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  (float)0xFFFFFFFF,          // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x10,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};
```

```
volatile parameter w2 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  HEX,                        // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL-2 Tx Data",             // unsigned char name[16]
  "[hex]",                    // unsigned char comment[16]
  8,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  (float)0xFFFFFFFF,          // range_max (FLOAT)
  1.0,                        // multiplier (double)
  32,                         // number_of_bits
  0x12,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};

const plist_item vlist1[] = {
  LIST_STARTING_STRING,
  "SPS-16b",
  "SPS-32b",
  "LHC-32b",
  LIST_TERMINATION_STRING
};

volatile parameter w3 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL Type/Length",           // unsigned char name[16]
  "",                         // unsigned char comment[16]
  7,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  1.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x14,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist1[0])      // address of the referenced list array
};

const plist_item vlist2[] = {
  LIST_STARTING_STRING,
  "Off ",
  "SL-1",
  "SL-2",
  "SL-1/2",
  LIST_TERMINATION_STRING
};

volatile parameter w4 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL Rx Hold",               // unsigned char name[16]
  "",                         // unsigned char comment[16]
  6,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  3.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x15,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist2[0])      // address of the referenced list array
};

volatile parameter w5 = {
  PARAM_EDIT,                 // basis (PARAM_EDIT, PARAM_READBACK)
  LIST,                       // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "SL Loop Thru",             // unsigned char name[16]
  "",                         // unsigned char comment[16]
```

```
  6,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  3.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x16,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  (unsigned)(&vlist2[0])      // address of the referenced list array
};

volatile parameter w5 = {
  PARAM_INVISIBLE,            // basis (PARAM_EDIT, PARAM_READBACK)
  BOOLEAN,                    // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "Timing Type",              // unsigned char name[16]
  "[0=Std, 1=CTRV]",          // unsigned char comment[16]
  1,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  1.0,                        // range_max (FLOAT)
  1.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x17,                       // address (INT8)
  0.0,                        // default/safe value
  1,                          // instant write
  NILPTR                      // address of the referenced list array
};

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Keep this parameter for proper MENU FPGA WRITE definition.
// Mandatory setting is only BASIS = PARAM_MENU, other vars are ignored for now
// (KNOB version 1.0)
volatile parameter write_menu = {
  PARAM_MENU,                 // basis (PARAM_EDIT, PARAM_READBACK)
  FPGA_WRITE,                 // type (SIGNED, UNSIGNED, BOOLEAN, HEX, LIST)
  "FPGA STORE",               // unsigned char name[16]
  "Push to store parameters", // unsigned char comment[16]
  0,                          // number_of_digits [1..9]
  0,                          // decimal_position [0..number_of_digits]
  0.0,                        // range_min (FLOAT)
  0.0,                        // range_max (FLOAT)
  0.0,                        // multiplier (double)
  8,                          // number_of_bits
  0x00,                       // address (INT8)
  0.0,                        // default/safe value
  0,                          // instant write
  NILPTR                      // address of the referenced list array
};
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#define MAX_PARAMETERS  (sizeof(plist)/4)

// parameters list
volatile parameter *plist[] = { &w1, &r1, &w2, &r2,
                                &w3, &w4, &w5, &w6,
                                &write_menu,
                                &r0 };

//EOF
```

N Common Firmware Blocks

This appendix contains firmware blocks that are common to all module. The blocks included are as follows:

N.1 ClkChk

```
----------------------------------------
----------------------------------------
-- Date        : Sat Dec 11 10:06:46 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Checks to ensure a clock is present.
--
-- Changelog   : 20101212 (1.1)
--                 Initial revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity  ClkChk  is
  generic (
    N       : natural := 16
  );

  port (
    Clk,
    Rst,
    ClkIn   : in  std_logic;

    ClkOk   : out std_logic;

    ClkMin,
    ClkMax  : in  std_logic_vector(N - 1 downto 0)
  );
end;

architecture  V1  of  ClkChk  is

signal    Count   : unsigned(N - 1 downto 0);

constant  C_Zero  : unsigned(N - 1 downto 0) := (others => '0');

begin
  process (Clk, Rst) begin
    if Rst = '1' then
      Count <= C_Zero;
      ClkOk <= '0';
    elsif rising_edge(Clk) then
      if ClkIn = '1' then
        if Count >= unsigned(ClkMin) and Count <= unsigned(ClkMax) then
          ClkOk <= '1';
        else
          ClkOk <= '0';
        end if;

        Count <= C_Zero;
      else
        if Count > unsigned(ClkMax) then
          ClkOk <= '0';
        end if;

        Count <= Count + 1;
      end if;
    end if;
  end process;
end;

--EOF
```

N.2 ClkGate

```
----------------------------------------
----------------------------------------
-- Date        : Mon Jun 07 16:03:06 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Simple clock divider. Output is a 50% square wave which can be
--               used as an ADC clock, etc. Also outputs single clock width
--               pulses on the positive and negative edges of this clock which
--               can be used as a gate for FPGA logic.
--
--               Evaluates both positive and negative edges of input clock so
--               is capable of even and odd clock divisors while maintaining
--               50% duty cycle.
--
--               Bit length of counter can be set with generic "N".
--
--               Sync pin causes output to be asserted high immediately and
--               PosEdge pulse to be output.
--
--               Enable pin can be left floating (defaults to '1').
--               Sync pin can be left floating (defaults to '0').
--
-- Changelog   : 20101212 (1.5)
--                 Missed some reset conditions, oops!
--
--               20101119 (1.4)
--                 Rewrite to maintain 50% duty cycle even when using odd
--                 values for the clock divider. Also to fix async. glitch
--                 noticed while testing with PeakDetector. Pin naming updated
--                 to make function more clear.
--
--               20101015 (1.3)
--                 Added 50% clock output for ADC clocking. Also added single
--                 clock width strobe output on negative edge of this clock.
--
--               20100906 (1.2)
--                 Version to improve logic usage.
--
--               20100903 (1.1)
--                 Inital revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity  ClkGate  is
  generic (
    N         : natural := 16
  );

  port (
    Clk,
    Rst       : in  std_logic;
    Sync      : in  std_logic := '0';
    Enable    : in  std_logic := '1';

    Divider   : in  std_logic_vector(N - 1 downto 0);

    PosEdge,
    NegEdge,
    SlowClk   : out std_logic
  );
end;
```

```
architecture  V1  of  ClkGate  is

signal    Count     : unsigned(N - 1 downto 0);
signal    Half      : unsigned(N - 1 downto 0);

signal    Odd       : std_logic;
signal    SC        : std_logic;
signal    SCOddCorr : std_logic;
signal    SCPrev    : std_logic;

constant  C_Zero    : unsigned(N - 1 downto 0) := (others => '0');

begin
  process (Clk, Rst) begin
    if Rst = '1' then
      SC      <= '0';
      Count   <= C_Zero;
      SCPrev  <= '0';
      Odd     <= '0';
      Half    <= C_Zero;
    elsif rising_edge(Clk) then
      if Enable = '1' then
        SCPrev <= SC;
        Count  <= Count - 1;

        if Count = C_Zero or Sync = '1' then
          Half  <= unsigned('0' & Divider(N - 1 downto 1));
          Count <= unsigned(Divider) - 1;
          Odd   <= Divider(0);
          SC    <= '1';
        elsif Count = Half then
          SC <= '0';
        end if;
       end if;
    end if;
  end process;

  process (Clk, Rst) begin
    if Rst = '1' then
      SCOddCorr <= '0';
    elsif falling_edge(Clk) then
      if Odd = '1' and Count = Half then
        SCOddCorr <= '0';
      else
        SCOddCorr <= '1';
      end if;
    end if;
  end process;

  SlowClk <= SC and SCOddCorr;
  PosEdge <= '1' when SC = '1' and SCPrev = '0' else '0';
  NegEdge <= '1' when SC = '0' and SCPrev = '1' else '0';
end;

--EOF
```

N.3 ClkSync

```
-----------------------------------------
-----------------------------------------
-- Date        : Thu Jun 17 09:15:15 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Synchronises ClkIn to Clk and produces pulses on the positive
--               and negative edges. Uses two flip-flops for synchronisation
--               and a final for edge detection.
--
-- Changelog   : 20100903 (1.1)
--                 Inital revision.
--
-----------------------------------------
-----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;

entity  ClkSync  is
  port (
    Clk,
    ClkIn,
    Rst       : in  std_logic;

    ClkOut,
    nClkOut,
    PosEdge,
    NegEdge   : out std_logic
  );
end;

architecture  V1  of  ClkSync  is

signal SR : std_logic_vector(2 downto 0);

begin
  process (Clk, Rst) begin
    if Rst = '1' then
      SR <= (others => '0');
    elsif rising_edge(Clk) then
      SR(2) <= SR(1);
      SR(1) <= SR(0);
      SR(0) <= ClkIn;
    end if;
  end process;

  PosEdge <= '1' when SR(2) = '0' and SR(1) = '1' else '0';
  NegEdge <= '1' when SR(2) = '1' and SR(1) = '0' else '0';

  ClkOut <= SR(1);
  nClkOut <= not SR(1);
end;

--EOF
```

N.4 Counter

```
----------------------------------------
----------------------------------------
-- Date        : Wed Oct 27 13:50:25 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Simple counter.
--
-- Changelog   : 20101027 (1.2)
--                 Added Max value at which the counter will wrap around.
--
--               20101027 (1.1)
--                 Initial revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity  Counter  is
  generic (
    N       : natural := 16
  );

  port (
    Clk,
    Rst,
    Ena,
    Clr     : in  std_logic;

    Pre     : in  std_logic_vector(N-1 downto 0) := (others => '0');
    Max     : in  std_logic_vector(N-1 downto 0) := (others => '1');

    Cnt     : out std_logic_vector(N-1 downto 0)
  );
end;

architecture  V1  of  Counter  is

signal Loc_Cnt  : unsigned(N-1 downto 0);

begin
  process (Clk, Rst, Pre) begin
    if Rst = '1' then
      Loc_Cnt <= unsigned(Pre);
    elsif rising_edge(Clk) then
      if Clr = '1' then
        Loc_Cnt <= unsigned(Pre);
      elsif Ena = '1' then
        if Loc_Cnt = unsigned(Max) then
          Loc_Cnt <= (others => '0');
        else
          Loc_Cnt <= Loc_Cnt + 1;
        end if;
      end if;
    end if;
  end process;

  Cnt <= std_logic_vector(Loc_Cnt);
end;

--EOF
```

N.5 EdgeDetect

```
----------------------------------------
----------------------------------------
-- Date        : Sun Dec 12 18:29:31 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : An edge detector.
--
-- Changelog   : 20101212 (1.1)
--                 Initial revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;

entity  EdgeDetect  is
  port (
    Clk,
    Rst,
    SigIn     : in    std_logic;

    PosEdge,
    NegEdge   : out   std_logic
  );
end;

architecture  V1  of  EdgeDetect  is

signal  SR  : std_logic_vector(1 downto 0);

begin
  process (Clk, Rst) begin
    if Rst = '1' then
      SR <= (others => '0');
    elsif rising_edge(Clk) then
      SR(1) <= SR(0);
      SR(0) <= SigIn;
    end if;
  end process;

  PosEdge <= '1' when SR(0) = '1' and SR(1) = '0' else '0';
  NegEdge <= '1' when SR(0) = '0' and SR(1) = '1' else '0';
end;

--EOF
```

N.6 PulseGen

```
----------------------------------------
----------------------------------------
-- Date        : Fri Jun 11 11:29:16 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Pulse generator (i.e. a monostable).
--
--               Variable bit length can be set with generic "N".
--
--               SlowClk pin can be left floating (default to '1') which runs
--               the pulse generator rate of Clk.
--
-- Changelog   : 20101015 (1.3)
--                 Fix to enable constant output by holding Trigger input high.
--                 Previously there would have been a 1 clock delay between
--                 each output pulse in this case. Now there is a single pulse
--                 which will last until the Duration after the Trigger goes
--                 low.
--
--               20100906 (1.2)
--                 Fix to supress SyncOut if there are multiple triggers within
--                 the output pulse. Now only a single one is output at the
--                 start.
--
--               20100903 (1.1)
--                 Initial revision.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity  PulseGen  is
  generic (
    N         : natural := 16
  );

  port (
    Clk,
    Rst,
    Trigger   : in  std_logic;
    SlowClk   : in  std_logic   := '1';

    Duration  : in  std_logic_vector(N - 1 downto 0);

    PulseOut,
    SyncOut   : out std_logic
  );
end;

architecture  V1  of  PulseGen  is

signal    Loc_Trigger : std_logic;
signal    Loc_Pulse   : std_logic;
signal    Loc_Count   : unsigned(N - 1 downto 0);

constant  C_Zero    : unsigned(N - 1 downto 0) := (others => '0');

begin
  process (Clk, Rst) begin
    if Rst = '1' then
      Loc_Count   <= C_Zero;
      Loc_Trigger <= '0';
      Loc_Pulse   <= '0';
      SyncOut     <= '0';
```

```
    elsif rising_edge(Clk) then
      SyncOut <= '0';

      if SlowClk = '0' then
        if Trigger = '1' and Loc_Pulse = '0' then
          Loc_Trigger <= '1';
        end if;
      else -- SlowClk = '1'
        if Loc_Pulse = '0' or Loc_Count = C_Zero then
          if Loc_Trigger = '1' or Trigger = '1' then
            Loc_Pulse   <= '1';
            Loc_Count   <= unsigned(Duration) - 1;
            Loc_Trigger <= '0';
            SyncOut     <= '1';
          else
            Loc_Pulse   <= '0';
          end if;
        else -- Loc_Pulse = '1' and Loc_Count /= 0
          Loc_Count   <= Loc_Count - 1;
          Loc_Pulse   <= '1';
        end if;
      end if;
    end if;
  end process;

  PulseOut <= Loc_Pulse;
end;

--EOF
```

O SerialLink[16] Controller — Firmware

This appendix contains firmware blocks that for the SerialLink[16] transmitter and receiver. The following blocks are included:

SCLRxSeq CLXXVI

State machine which implements the serial receiver and repeater functionality for both the Serial Control Link and SerialLink[16] protocols.

SCLTrxSeq CLXXVII

State machine which implements the serial transmitter functionality for both the Serial Control Link and SerialLink[16] protocols.

SL16RxN CLXXVIII

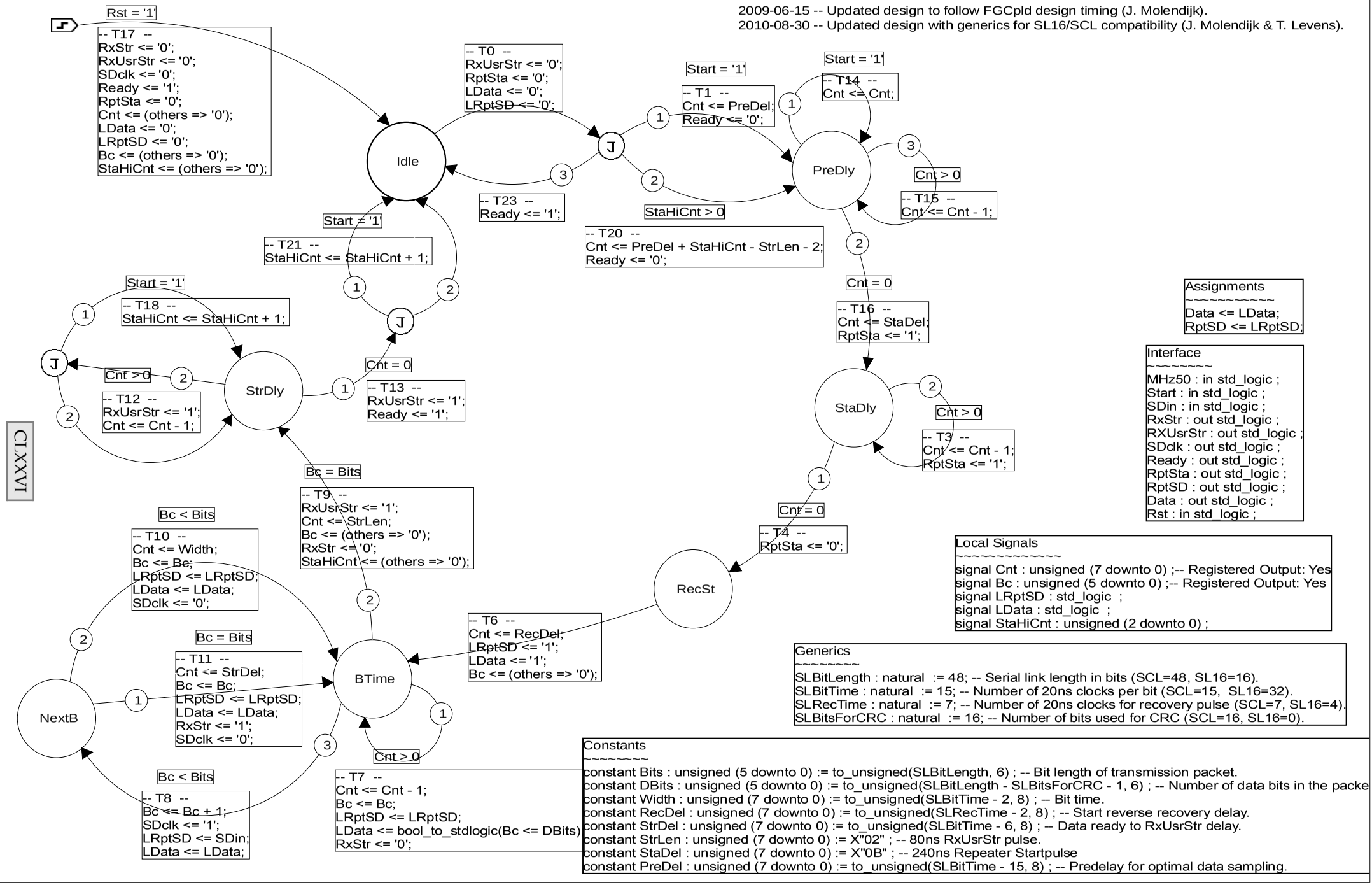
Wrapper block for the SerialLink[16] receiver and repeater. Contains the state-machine controller and an external shift register.

SL16TrxN CLXXIX

Wrapper block for the SerialLink[16] transmitter. Contains the state-machine controller and an external shift register.

SCL Rx Sequencer

2009-06-15 -- Updated design to follow FGCpld design timing (J. Molendijk).
2010-08-30 -- Updated design with generics for SL16/SCL compatibility (J. Molendijk & T. Levens).



CLXXVI

Assignments
~~~~~~~~~~~
Data <= LData;
RptSD <= LRptSD;

Interface
~~~~~~~~~
MHz50 : in std_logic ;
Start : in std_logic ;
SDin : in std_logic ;
RxStr : out std_logic ;
RXUsrStr : out std_logic ;
SDclk : out std_logic ;
Ready : out std_logic ;
RptSta : out std_logic ;
RptSD : out std_logic ;
Data : out std_logic ;
Rst : in std_logic ;

Local Signals
~~~~~~~~~~~~~
signal Cnt : unsigned (7 downto 0) ;-- Registered Output: Yes
signal Bc : unsigned (5 downto 0) ;-- Registered Output: Yes
signal LRptSD : std_logic ;
signal LData : std_logic ;
signal StaHiCnt : unsigned (2 downto 0) ;

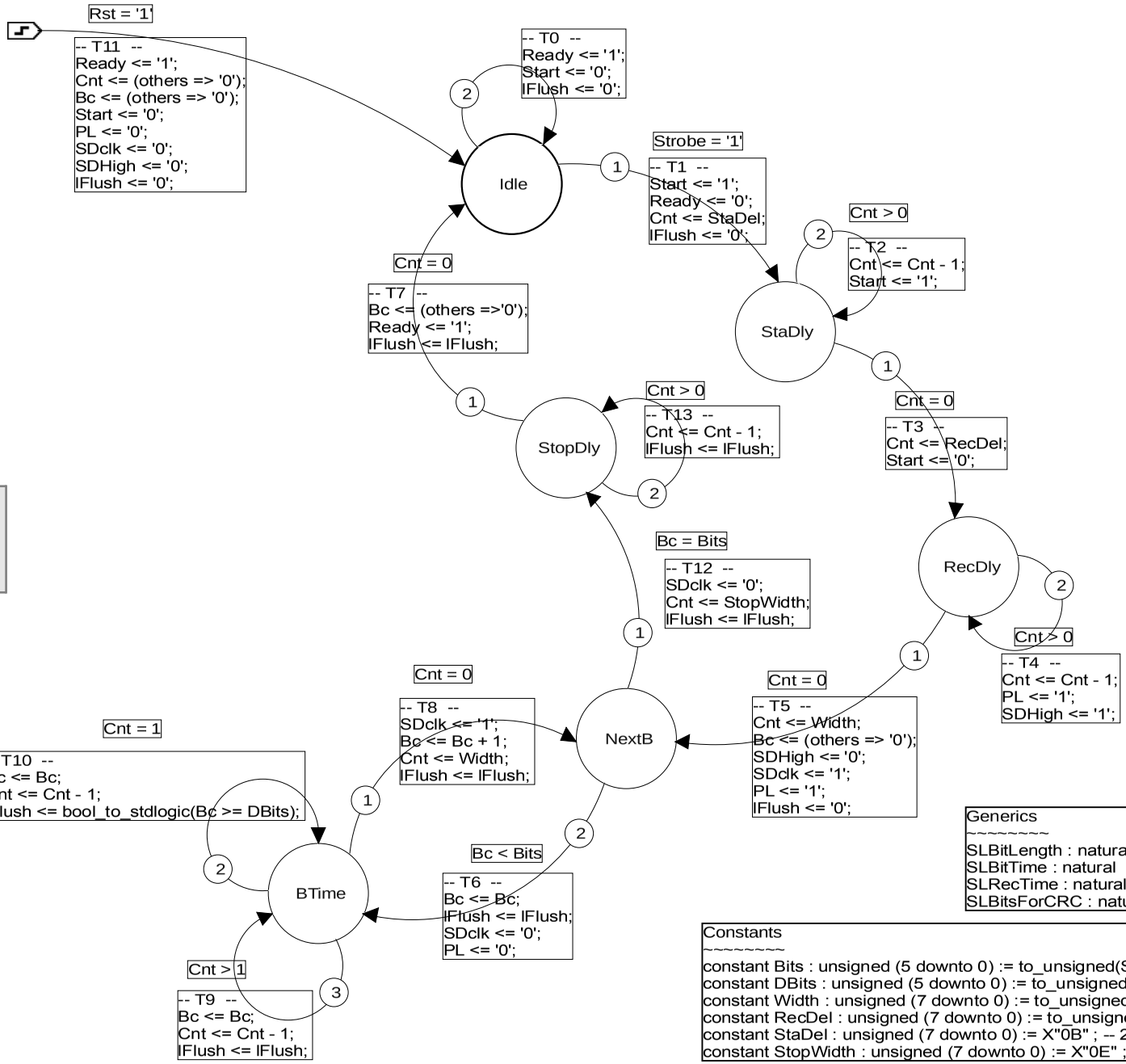
Generics
~~~~~~~~
SLBitLength : natural := 48; -- Serial link length in bits (SCL=48, SL16=16).
SLBitTime : natural := 15; -- Number of 20ns clocks per bit (SCL=15, SL16=32).
SLRecTime : natural := 7; -- Number of 20ns clocks for recovery pulse (SCL=7, SL16=4).
SLBitsForCRC : natural := 16; -- Number of bits used for CRC (SCL=16, SL16=0).

Constants
~~~~~~~~
constant Bits : unsigned (5 downto 0) := to_unsigned(SLBitLength, 6) ; -- Bit length of transmission packet.
constant DBits : unsigned (5 downto 0) := to_unsigned(SLBitLength - SLBitsForCRC - 1, 6) ; -- Number of data bits in the packe
constant Width : unsigned (7 downto 0) := to_unsigned(SLBitTime - 2, 8) ; -- Bit time.
constant RecDel : unsigned (7 downto 0) := to_unsigned(SLRecTime - 2, 8) ; -- Start reverse recovery delay.
constant StrDel : unsigned (7 downto 0) := to_unsigned(SLBitTime - 6, 8) ; -- Data ready to RxUsrStr delay.
constant StrLen : unsigned (7 downto 0) := X"02" ; -- 80ns RxUsrStr pulse.
constant StaDel : unsigned (7 downto 0) := X"0B" ; -- 240ns Repeater Startpulse
constant PreDel : unsigned (7 downto 0) := to_unsigned(SLBitTime - 15, 8) ; -- Predelay for optimal data sampling.

Design: CommonVisual:SCLRxSeq	Date: Fri Dec 03 14:48:58 2010	Page: 1
~~~~~~~~

SCL Tx Sequencer

2010-08-31 -- Updated design with generics for SL16/SCL compatibility (J. Molendijk & T. Levens).



CLXXVIII

Assignments
~~~~~~~~~~~~
Flush <= IFlush;

Local Signals
~~~~~~~~~~~~~~
signal Cnt : unsigned (7 downto 0) ;
signal Bc : unsigned (5 downto 0) ;
signal IFlush : std_logic ;

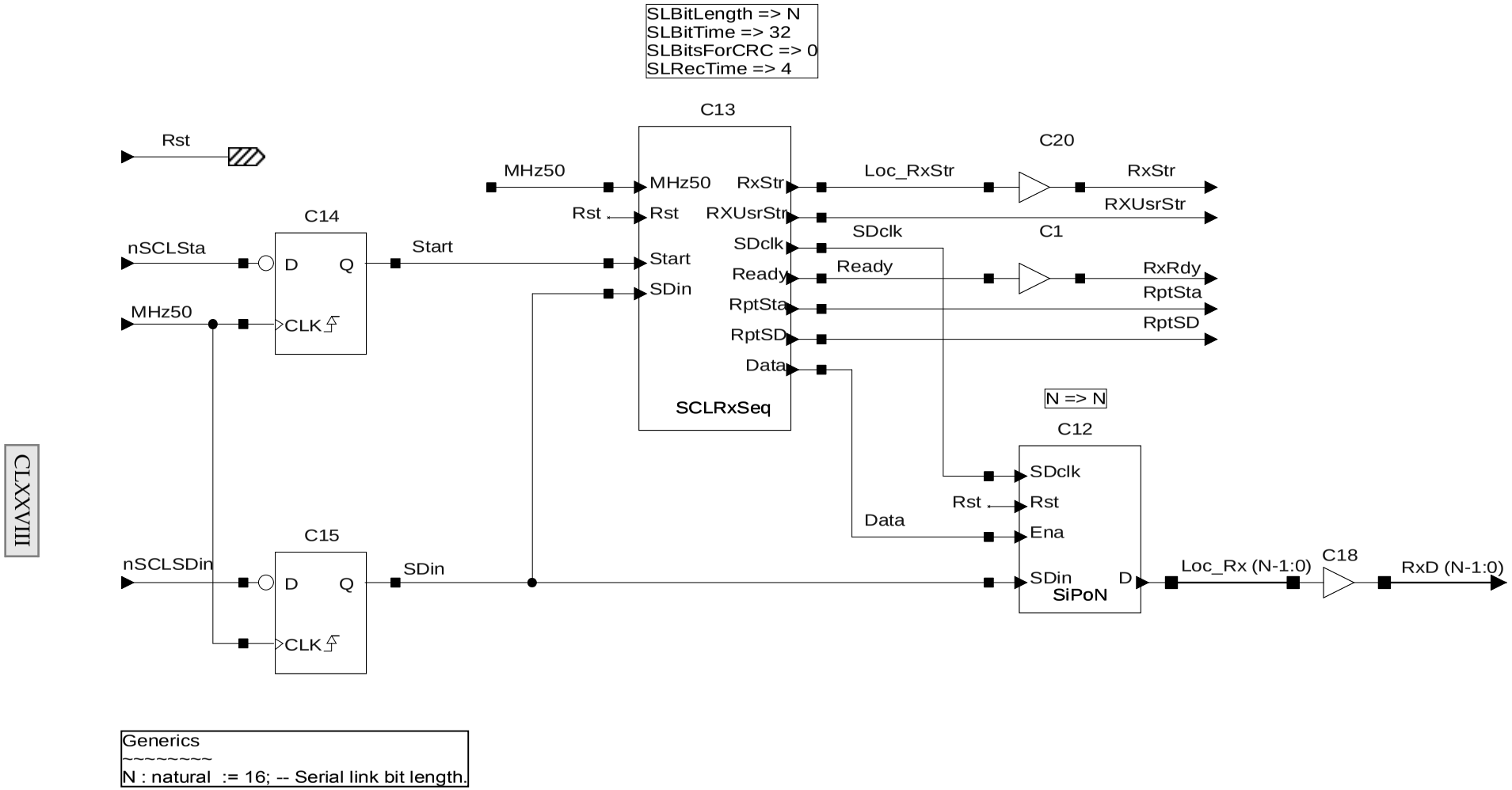
Interface
~~~~~~~~
MHz50 : in std_logic ; -- 50MHz clock.
Strobe : in std_logic ; -- User's data strobe.
Rst : in std_logic ;
Start : out std_logic ; -- Start pulse.
Ready : out std_logic ; -- Tx ready.
PL : out std_logic ; -- Parallel load.
SDclk : out std_logic ; -- Serial data clock.
SDHigh : out std_logic ; -- Serial data forced high.
Flush : out std_logic ; -- Flush CRC reg. control.

Generics
~~~~~~~~
SLBitLength : natural := 48; -- Serial link length in bits (SCL=48, SL16=16).
SLBitTime : natural := 15; -- Number of 20ns clocks per bit (SCL=15, SL16=32).
SLRecTime : natural := 7; -- Number of 20ns clocks for recovery pulse (SCL=7, SL16=4).
SLBitsForCRC : natural := 16; -- Number of bits used for CRC (SCL=16, SL16=0).

Constants
~~~~~~~~
constant Bits : unsigned (5 downto 0) := to_unsigned(SLBitLength, 6) ; -- Bit length of transmission packet.
constant DBits : unsigned (5 downto 0) := to_unsigned(SLBitLength - SLBitsForCRC - 1, 6) ; -- Number of data bits in the packet
constant Width : unsigned (7 downto 0) := to_unsigned(SLBitTime - 2, 8) ; -- Bit time.
constant RecDel : unsigned (7 downto 0) := to_unsigned(SLRecTime - 1, 8) ; -- Start reverse recovery delay.
constant StaDel : unsigned (7 downto 0) := X"0B" ; -- 240ns start pulse.
constant StopWidth : unsigned (7 downto 0) := X"0E" ; -- 320ns stop bit.

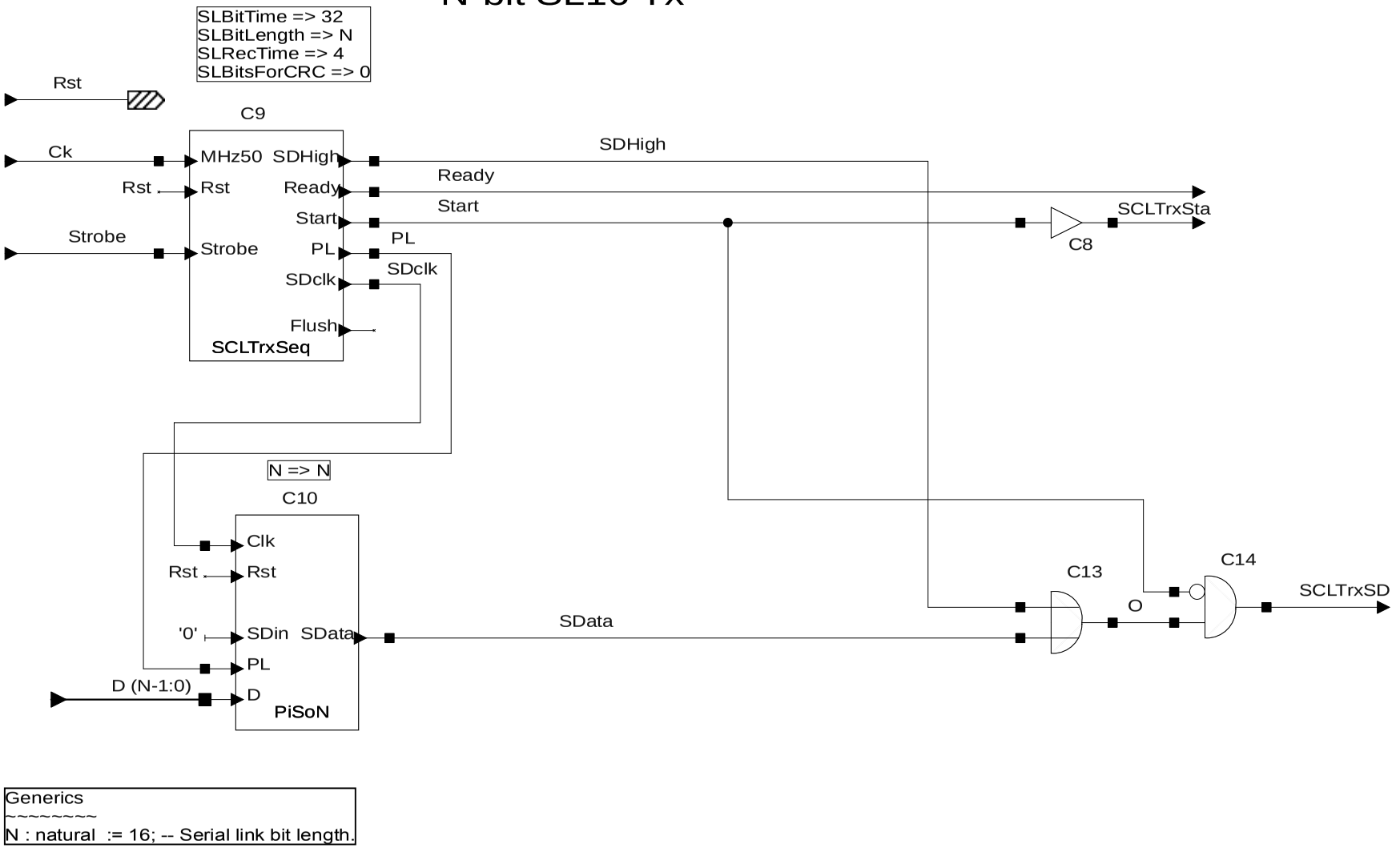
Design: CommonVisual:SCLTrxSeq	Date: Fri Dec 03 14:49:39 2010	Page: 1
~~~~~~~~

N-bit SL16 Rx



Design: CommonVisual:SL16RxN	Date: Fri Dec 03 14:50:57 2010	Page: 1 (1,1)

N-bit SL16 Tx



Generics
~~~~~~~~
N : natural := 16; -- Serial link bit length.

CLXXIX

Design: CommonVisual:SL16TrxN	Date: Fri Dec 03 14:50:38 2010	Page: 1 (1,1)
~~~~~~~~

P Maxim DS18B20 Controller — Firmware

This appendix contains firmware blocks that for the DS18B20 controller. The following blocks are included:

DS18B20_Intfce CLXXXII

Top level block for inclusion in projects.

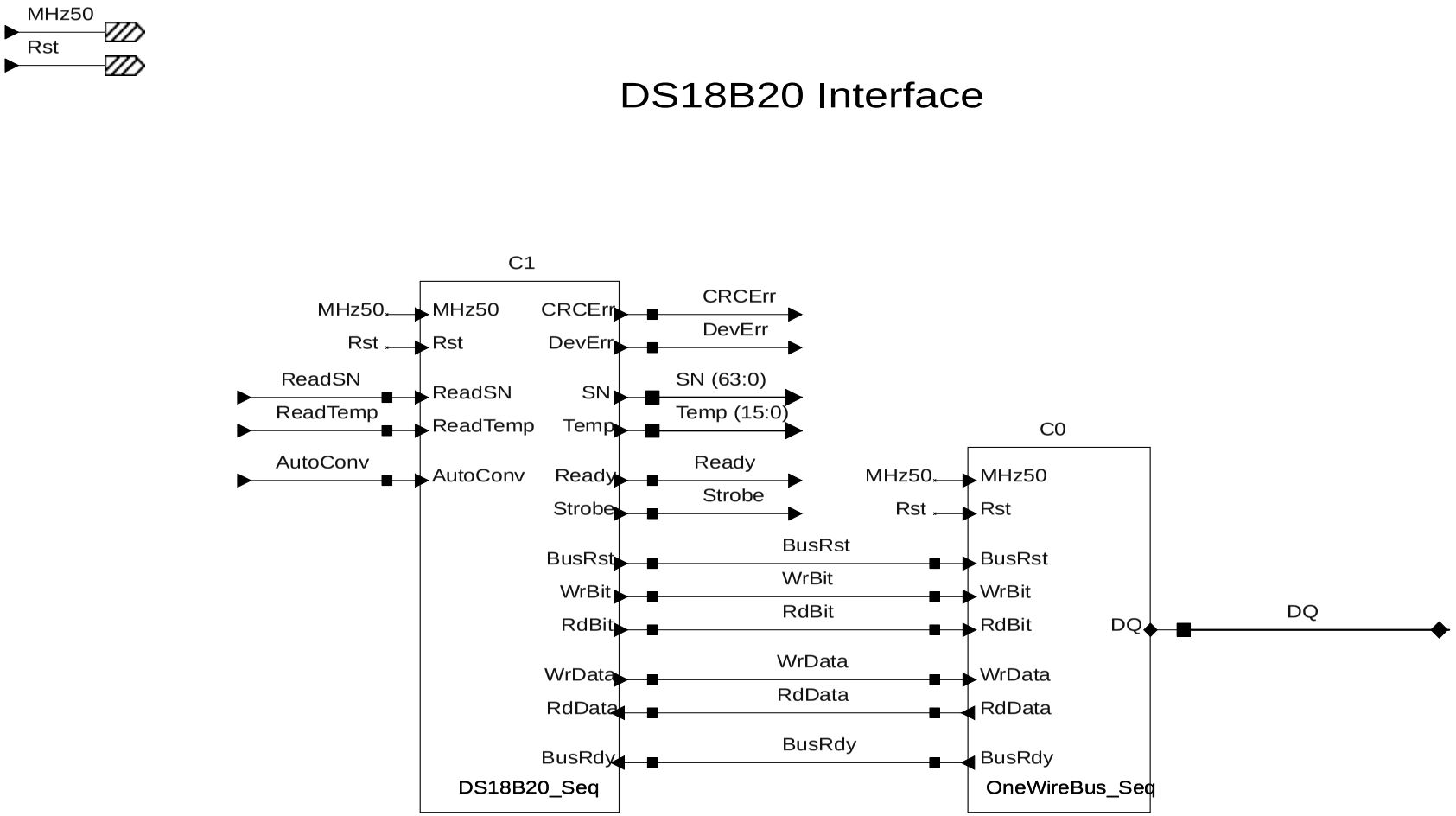
DS18B20_Seq CLXXXIII

DS18B20 command sequencer, responsible for triggering the correct bit pattern to communicate with the device.

OneWireBus_Seq CLXXXIV

1-Wire® Bus controller, responsible for controlling the timing of each bit on the bus.

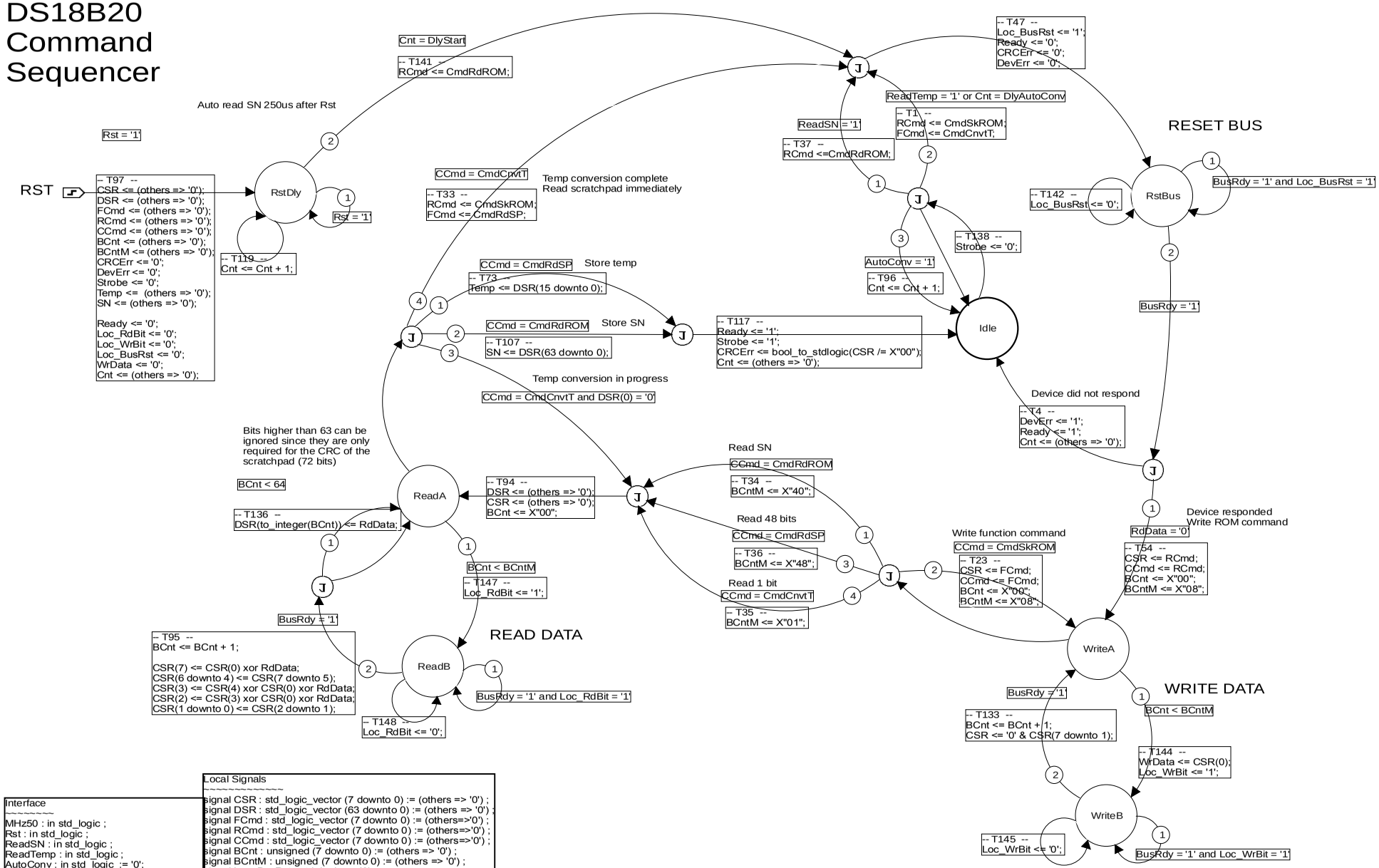
DS18B20 Interface



CLXXXII

Design: CommonVisual:DS18B20_Intfce	Date: Fri Dec 03 14:01:08 2010	Page: 1 (1,1)

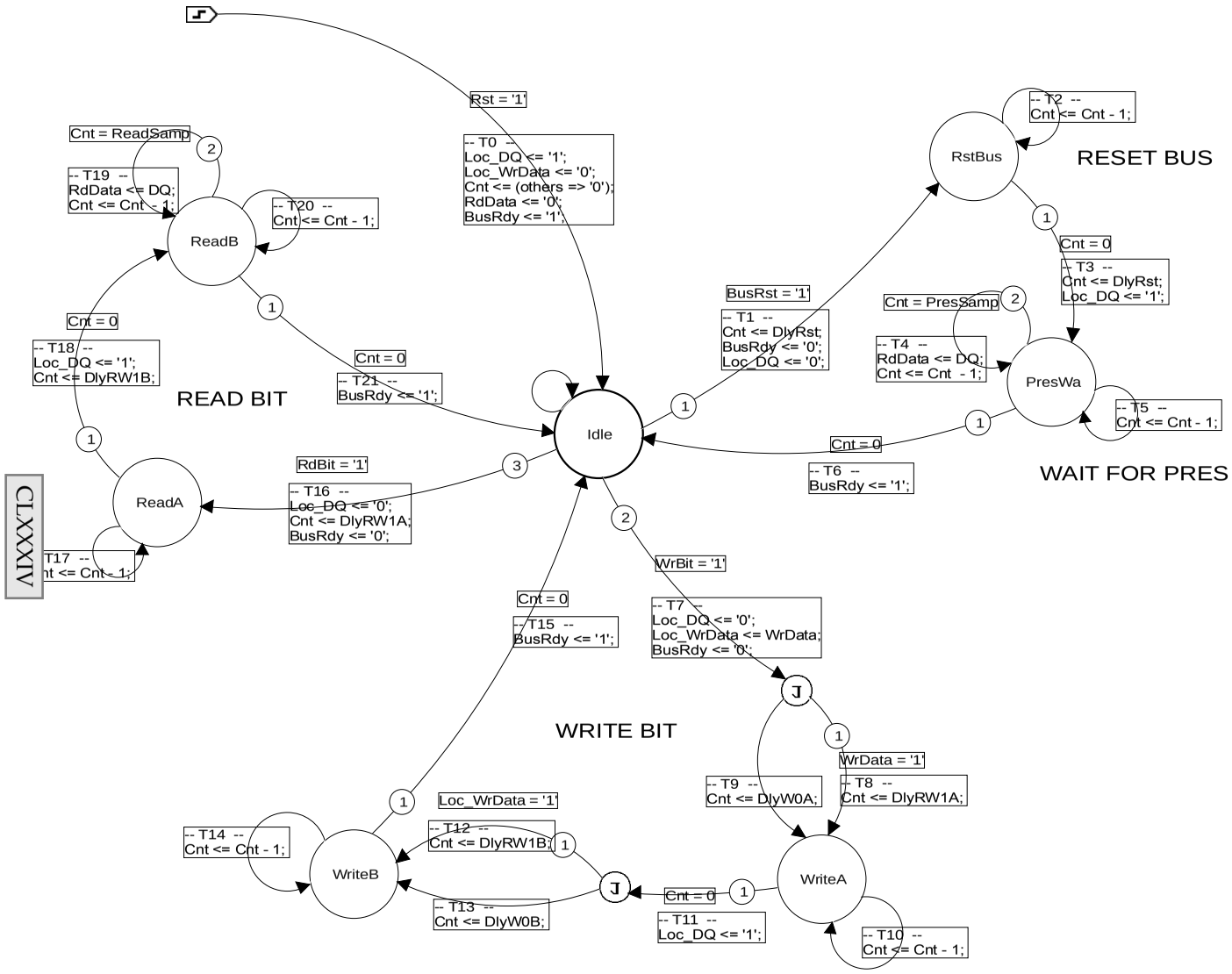
DS18B20 Command Sequencer



CLXXXIII

Design: CommonVisual:DS18B20_Seq	Date: Fri Dec 03 14:02:57 2010	Page: 1

MAXIM 1-WIRE BUS INTERFACE



Time delays are based on recommended values from AN126.

```
Constants
~~~~~~~~~
constant DlyRst : unsigned (15 downto 0) := X"5DC0" ; -- 480us
constant DlyRW1A : unsigned (15 downto 0) := X"012C" ; -- 6us
constant DlyRW1B : unsigned (15 downto 0) := X"0C80" ; -- 64us
constant DlyW0A : unsigned (15 downto 0) := X"0BB8" ; -- 60us
constant DlyW0B : unsigned (15 downto 0) := X"01F4" ; -- 10us
constant PresSamp : unsigned (15 downto 0) := X"5014" ; -- (480 -) 70us
constant ReadSamp : unsigned (15 downto 0) := X"0ABE" ; -- (64 -) 9us
```

```
Assignments
~~~~~~~~~~~
DQ <= '0' when Loc_DQ = '0' else 'Z';
```

```
Interface
~~~~~~~~~
MHz50 : in std_logic ;
Rst : in std_logic ;
DQ : inout std_logic ;
WrBit : in std_logic ;
RdBit : in std_logic ;
RdData : out std_logic ;
WrData : in std_logic ;
BusRdy : out std_logic ;
BusRst : in std_logic ;
```

```
Local Signals
~~~~~~~~~~~~~
signal Cnt : unsigned (15 downto 0) := (others => '0') ;
signal Loc_DQ : std_logic  := '1' ;
signal Loc_WrData : std_logic  := '0' ;
```

Design: CommonVisual:OneWireBus_Seq	Date: Fri Dec 03 14:02:11 2010	Page: 1

Q VHI SPI Controller — Firmware

This appendix contains firmware blocks for communication between a FPGA and the VHI interface over SPI. The following blocks are included:

KnobRegister CLXXXVI

Single register for data storage within the FPGA.

KnobSequencer CXC

State machine that implements the SPI slave protocol.

Q.1 KnobRegister

```
----------------------------------------
----------------------------------------
-- Date        : Wed Aug 18 09:08:22 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : I/O register for KNOB.
--
--               See file 'Readme' for full details.
--
----------------------------------------
----------------------------------------

library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity  KnobRegister  is
  generic (
    RegSize   : natural                 := 32;
    RegAddr   : std_logic_vector(6 downto 0)  := "0000000";
    RegSigned : boolean                 := false;
    RegDir    : boolean                 := false
  );

  port (
    Clk,
    Rst       : in  std_logic;
    RW        : in  std_logic   := '0';

    DIn       : in  std_logic_vector(31 downto 0);
    DOut      : out std_logic_vector(31 downto 0);

    Addr      : in  std_logic_vector(6 downto 0);

    Rdbk      : in  std_logic_vector(RegSize - 1 downto 0) := (others => '0');
    RdbkB     : in  std_logic                    := '0';
    Write     : out std_logic_vector(RegSize - 1 downto 0);
    WriteB    : out std_logic
  );
end;

architecture  V1  of  KnobRegister is

signal  Loc_Reg : std_logic_vector(RegSize - 1 downto 0);

begin
  process (Clk, Rst) begin
    if Rst = '1' then
      Loc_Reg <= (others => '0');
    elsif rising_edge(Clk) then
      if Addr = RegAddr then
        if RegDir = false then
          if RW = '1' then
            Loc_Reg <= DIn(RegSize - 1 downto 0);
          end if;
        else
          if RegSize /= 1 then
            Loc_Reg <= Rdbk;
          else
            Loc_Reg(0) <= RdbkB;
          end if;
        end if;
      end if;
    end if;
  end process;
```

```
  DOut <= std_logic_vector(resize(  signed(Loc_Reg), 8))  & X"000000" when RegSize <= 8  and
    RegSigned = true  and Addr = RegAddr else
          std_logic_vector(resize(unsigned(Loc_Reg), 8))  & X"000000" when RegSize <= 8  and
            RegSigned = false and Addr = RegAddr else
          std_logic_vector(resize(  signed(Loc_Reg), 16)) & X"0000"   when RegSize <= 16 and
            RegSigned = true  and Addr = RegAddr else
          std_logic_vector(resize(unsigned(Loc_Reg), 16)) & X"0000"   when RegSize <= 16 and
            RegSigned = false and Addr = RegAddr else
          std_logic_vector(resize(  signed(Loc_Reg), 32))             when RegSize <= 32 and
            RegSigned = true  and Addr = RegAddr else
          std_logic_vector(resize(unsigned(Loc_Reg), 32))             when RegSize <= 32 and
            RegSigned = false and Addr = RegAddr else
          (others => 'Z');

  Write <= Loc_Reg;
  WriteB <= Loc_Reg(0);
end;
```

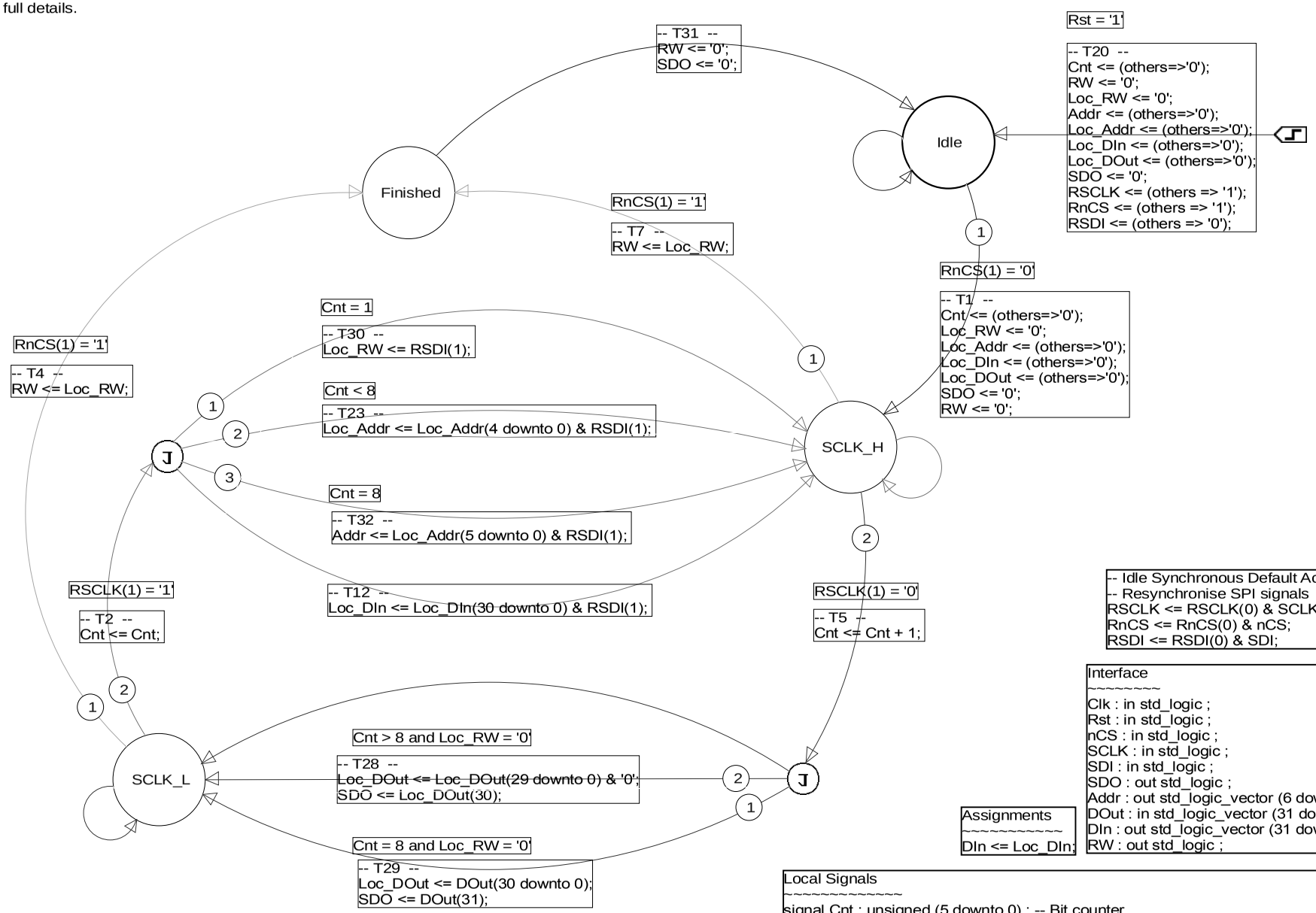
Q.1.1 KnobRegister 'read me' file

```
-----------------------------------------
-----------------------------------------
-- Date        : Thu Sep 02 14:06:32 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Read-me for block KnobRegister.
--
-----------------------------------------
-----------------------------------------

-- PURPOSE

KnobRegister is a single register that can be read from or written to
by the KNOB module.

-- GENERICS

RegDir [boolean]:
  Selects the register type:
  - true: Readback parameter, i.e. data from FPGA to KNOB.
  - false: Write paaramter, i.e. data from KNOB to FPGA.

RegSize [natural]:
  Size of the register in bits. Can be any number between 1 and 32.
  Data is padded to the minimum transmission frame size needed
  (either 8-, 16- or 32-bits) automatically.

RegSigned [boolean]:
  Selects whether the data is signed (true) or unsigned (false).
  Governs how the padding to frame size is done. In signed mode, the
  sign bit is repeated. In unsigned mode zeros are added.

RegAddr [std_logic_vector]:
  6-bit register address.

-- CONNECTIONS

All registers should be connected to a KnobSequencer block which
communicates with the KNOB module. This connection is via the DIn,
DOut, Addr & RW pins.

DOut is a tri-state bus, so all KnobRegister instances can be
connected to a common bus without any problems.

Note: These pins match the naming on the KnobSequencer so the
connections should be as follows (KnobSequencer => KnobRegister):

  DIn     => DIn
  DOut    => DOut
  Addr    => Addr
  RW      => RW

Connections to other blocks in the FPGA are via the Write, WriteB,
Rdbk or RdbkB pin. The generics RegDir and RegSize govern which
pin is used for a particular register. The pins ending in 'B' are
single bit (i.e. std_logic rather than std_logic_vector) and are used
when the RegSize generic is set to 1.

-- SYMBOLS

BitWrite_Symbol:
  Write parameter register with a single bit (std_logic) input.

Write_Symbol:
  Write parameter register with a bus (std_logic_vector) input.

BitRdbk_Symbol:
```

```
  Readback parameter register with a single bit (std_logic) output.

Rdbk_Symbol:
  Readback parameter register with a bus (std_logic_vector) output.

-- REFERENCE IMPLEMENTATION

For reference, a full working implementation can be found in the SPS
module 'DualTriggerUnit' in the 'KnobRegisters' block.

--EOF
```

KNOB SPI Sequencer

See file 'Readme' for full details.



CXC

Design: KnobLib:KnobSequencer	Date: Fri Dec 03 11:10:31 2010	Page: 1

Q.2.1 KnobSequencer 'read me' file

```
-----------------------------------------
-----------------------------------------
-- Date        : Thu Sep 02 13:50:47 2010
--
-- Author      : Tom Levens <tom.levens@cern.ch>
--
-- Company     : CERN, BE-RF-FB
--
-- Description : Read-me for state-machine KnobSequencer.
--
-----------------------------------------
-----------------------------------------

-- PURPOSE

KnobSequencer is a state-machine to control the FPGA side of the SPI
connection to a KNOB module. The KNOB is the master on the SPI bus and
controls the SCLK and nCS signals.

-- CONNECTIONS

KnobSequencer connects to the KNOB via the SCLK, nCS, SDI & SDO pins.

Note: SDI & SDO are input and output from the FPGA, so the connections
should be as follows (KnobSequencer => KNOB):

  SDO     => SDI
  SDI     => SDO
  nCS     => nCS
  SCLK    => SCLK

It can connect to any number of KnobRegister blocks which are used to
prepare, store and retrieve the data from the FPGA via the DIn, DOut,
Addr & RW pins.

Note: These match the naming on the KnobRegister so the connections
should be as follows (KnobSequencer => KnobRegister):

  DIn     => DIn
  DOut    => DOut
  Addr    => Addr
  RW      => RW

-- REFERENCE IMPLEMENTATION

For reference, a full working implementation can be found in the SPS
module 'DualTriggerUnit' in the 'KnobRegisters' block.

--EOF
```