Chapter 2

V1495 Module by CAEN

2.1 Overview of V1495 Module

From the V1495 User Manual[4]:

> The Mod. V1495 is a VME 6U board, 1U wide, suitable for various digital Gate/Trigger/Translate/Buffer/Test applications, which can be directly customised by the User, and whose management is handled by two FPGA's: FPGA "Bridge", which is used for the VME interface and for the connection between the VME interface and the 2nd FPGA (FPGA "User") through a proprietary local bus. FPGA "Bridge" manages also the programming via VME of the FPGA "User". FPGA "User", which manages the front panel I/O channels. FPGA "User" is provided with a basic firmware which allows to perform coincidence matrix, I/O register and asynchronous timers functions.
>
> FPGA "User" can be also free reprogrammed by the user with own custom logic function (see § 5.1). It is connected as slave to the FPGA "Bridge" via CAEN Local Bus, whose protocol shall be used in order to communicate with the FPGA "Bridge" and thus with the VME bus.
>
> The I/O channel digital interface is composed by four sections (A, B, C, G) placed on the motherboard (see § 1.2). The channel interface can be expanded in the D, E, F sections by using up to 3 mezzanine boards (see § 2.6 and § 2.7), which can be added, choosing between the four types developed in order to cover the I/O functions and the ECL, PECL, LVDS, NIM, TTL electrical standard

(see § 1.2). The maximum number of channels can be expanded up to 194.

The FPGA “User” can be programmed “on the fly” directly via VME, without external hardware tools, without disconnecting the board from the set up, without resetting it or turning the crate off, allowing quick debug operations by the developer with his own firmware. A flash memory on the board can store the different programming file, which can be loaded to the FPGA “User” at any moment.

Four independent digital programmable asynchronous timers are available for Gate/Trigger applications. It is possible to chain them for generating complex Gate/Trigger pulse.

2.2 MCLK/LCLK/PLLCLK/PLLCLK_90

There are two digital clocks that synchronize all of the digital state machines in the V1495 as well as the VFATs for the GEMReadout project. Only the local clock is used in the v1495usr_firmare project.

The local clock, LCLK, for the V1495 has a fixed frequency of 40Mhz and is the clock to which all other internal digital states other than those governed by the MCLK are synchronized. This includes the important read/write/reset operations involving the VME bus. This clock is necessary for all V1495 projects.

The MCLK is a variable clock for the GEMReadout project that is injected by an external pulse generator (usually G0) that controls the read/write cycles of the VFAT as well as the data that is received from the VFATs by the V1495. Usually when referring to the signal as seen by the VFATs, it is referred to as MCLK per the VFAT2 literature. Internal to the V1495 all data sent and received is synchronized to the rising edge of MCLK, but here it is referred to as PLLCLK. Depending on the frequency of MCLK this may be controlled by a PLL. See the section PLLBlock.vhd for more information. PLLCLK_90 is the improperly named, inverted version of PLLCLK. This is the signal sent to the VFATs since their read/write cycle is 180 degrees out of phase with the V1495.

Having two different clocks internal to the V1495 has posed some particularly difficult problems. Care has been taken in this project when one signal is changed in sync with one of the clocks and then analyzed in sync with the other clock. An example of this is the CalPulse and Reset functionality in the V1495. When dealing with HDL one has to be careful to include conditional statements on two separate lines when the changes that occur are due to the two different clocks; otherwise the HDL translator takes the liberty of changing one of the signals on the rising edge of the incorrect clock. This is not particularly easy to diagnose as it does not behave as one might expect from an intelligently designed compiler.

2.3 Inputs and Outputs

As one can see from the block diagram in Figure 2-1 below, and as also found in the V1495 User Manual, the v1495 has two LVDS, 32-channel input ports (e.g. A and B), one LVDS, 32-channel output port (C), and two NIM/TTL I/Os (G0, G1) that come standard. There are three other mezzanine expansion ports that can be purchased with addition 32-channel LVDS/ ECL/PECL, 8-channel NIM/TTL, or even 8-channel 16-bit resolution analogue ports. For more information on this refer to the CAEN website. A diagram of the physical layout of these ports from the perspective of the front panel in also included in Figure 2-2 on page 14.

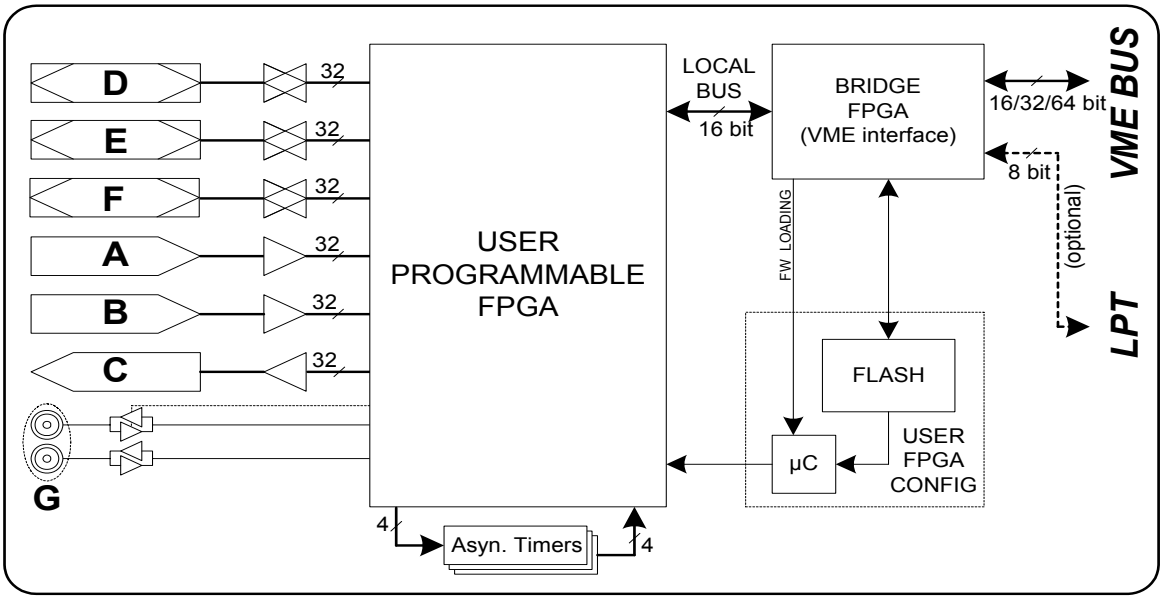


Figure 2-1: Mod. V1495 Block Diagram

For the purposes of this project, ports A, C, G0, and G1 have been utilized with port G configured as inputs with TTL logic and 50-Ohm termination. The termination switches are physically located on the board near the G port, and it is not specifically mentioned in the V1495 User Manual how to configure this. Furthermore, the inner workings of the V1495 (i.e. board layout) are not readily available even if the user wishes to have these for configuration purposes. The easiest way to determine how to program and utilize the V1495 is to familiarize oneself with the concepts of an FPGA and how the Cyclone FPGA for this board is connected to the external world via the V1495_USR_FIRMWARE that is provided for free. A basic overview of this firmware is provided in its own respective section.

All signal received by the V1495 are determined simply on the rising edge of their respective clocks. There are no other dexterous schemes implemented for noise rejection or signal degradation.

There are also two LEDs on the front panel of the V1495 for user interface. The DTACK LED blinks green whenever a VME read/write operation is performed on the board. This can

be used to determine whether or not a firmware update is working. It is not utilized during normal operation since most read/write operations to the V1495 happen faster than the LED can blink let alone how fast the human eye can perceive a pulse of this light. The USER LED has three colors: green/orange/red (really this is just green and red with the orange being the combination of green and red). The green part of this LED blinks in sync with the system clock, LCLK, at a rate of 0.60Hz and a 50% duty cycle. This comes from bit 25 of the signal "HEART_BEAT_CNT" which is incremented at the rising edge of LCLK. The red LED is on whenever the system reset is active. This happens when either the asynchronous system reset, nLBRES, is active low or the PLL is not locked. For the version of the GEMReadout firmware where there is no PLL (e.g. at MCLK frequencies lower than 15Mhz), this LED can only be active when the asynchronous reset is low.

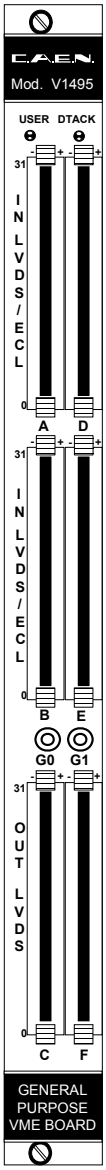


Figure 2-2: Model V1495 front panel (with A395A/B/C piggy back boards)

2.4 Addressing the V1495

It is important to note that the VME bus address is different than the memory-mapped logical address for the board as referred to by the PowerPC controller. For most of the functions written for the Read Out Controller (ROC), these will reference the PowerPC logical address rather than the VME bus address. The one major exception for this is the v1495Init() function which uses the VME bus address to check for the presence of the V1495 and then returns the logical memory address. More on this function is described in the sections on the ROC software.

For this project we have set the address lines of the V1495 to the hex values "0811" respectively. As can be seen in the silk screen print on the board in Figure 2-3, this is the base address for the 32 most significant bits in the VME bus address (BASE ADDRESS [31:16]). This, of course, signifies that the V1495 uses a 32-bit addressing scheme. According to the V1495 User Manual 24-bit addressing should also be possible, but we have not used this.

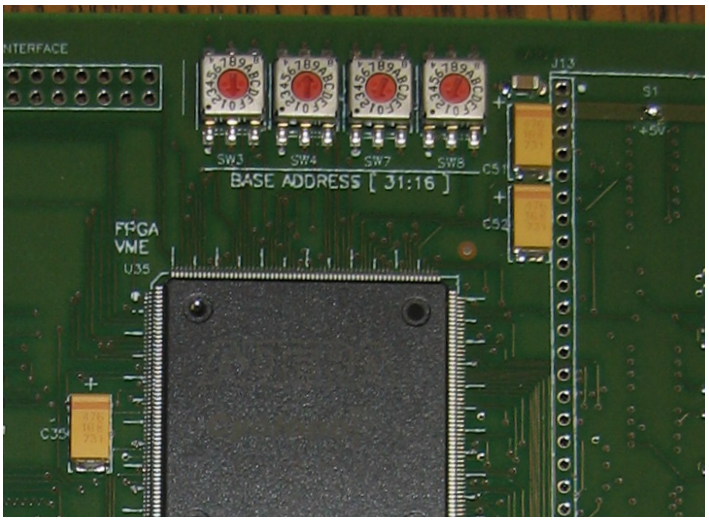


Figure 2-3: HEX Address Switches on V1495

Determining the exact relationship between the VME bus address and the ROC memory address has proven difficult and at best one can only surmise that the two are related by a simple offset of 0x7800_0000. With the VME bus address of 0x0811_0000 this gives a memory address of 0x8011_0000.

All register addresses in the V1495 are located using this offset plus their respective addresses. These registers can essentially be divided into to different classes: Register addresses 0x0000 – 0x7FFC are the USER FPGA Access registers and register addresses 0x8000 – 0x801FE contain the V1495 CAEN interface registers.

The USER FPGA registers are determined in the v1495usr_pkg.vhd module. Table 2-1 shows these registers with their "struct" equivalents in the ROC software. These registers

contain most of the pertinent information being communicated between the V1495 and the ROC: for example, A_GEMA0_EVENTDATA, address 0x4000, contains the information from the top of the DataFIFO for the first VFAT detector when requested.

The V1495 CAEN interface registers contain information on the V1495 board itself as well as controlling how to reflash and reset the VME and USER FPGAs. A discussion of these functions would be superfluous and as such the reader is referred to the V1495 User Manual pg. 17-21.[4]

USER FPGA Registers	V1495ReadoutCtrl struct	Register Numbers	Description
A_BOARDIDS	MezzanineIDs	0x0000 - 0x0001	Mezzanine IDs for D, E, and F
A_REVISION	Revision	0x0002 - 0x0003	Firmware Revision (hardcoded in firmware)
A_RESET	Reset	0x0004 - 0x0005	Used to Reset V1495
	Reserved1[5]	0x0006 - 0x000F	
A_GEM_TX_START	TxStart	0x0010 - 0x0011	Sends GEM_TX_START to TxChannel
A_GEM_SOFT_TRIG	SoftTrigger	0x0012 - 0x0013	Sends SOFT_TRIGGER signal
A_GEM_TRIG_WORD	TriggerWord	0x0014 - 0x0015	Contains SOFT and HARD trigger words
A_GEM_TX_WORD_(0 - B)	TxWord[12]	0x0016 - 0x002D	By default, this is V1495 VME Input register
	Reserved2[1]	0x002E - 0x002F	
A_GEMA(0-5)_FIFOSIZE	FIFOLength[12]	0x0030 - 0x0047	Contains total number of words currently in dataFIFO as well as the sizeFIFO
A_GEMA(0-5)_EVENTSIZE A_GEMB(0-5)_EVENTSIZE	EventSize[12]	0x0048 - 0x005F	Contains the value for the number of words stored in dataFIFO from most resent event
	Reserved3[16]	0x0060 - 0x007F	
A_GEMA(0-5)_EVENTS_SENT_H A_GEMB(0-5)_EVENTS_SENT_H	EventsSentH[12]	0x0080 - 0x0097	Contains bits 32 - 16 of the value of the number of events sent
A_GEMA(0-5)_EVENTS_SENT_L A_GEMB(0-5)_EVENTS_SENT_L	EventsSentL[12]	0x00A0 - 0x00B6	Contains bits 15 - 0 of the value of the number of events sent
A_GEMA(0-5)_EVENTDATA A_GEMB(0-5)_EVENTDATA	EventData[12][128]	0x4000 - 0x4BFF	Contains the actual DataOut data as captured in 16-bit segments from the VFATs

V1495ReadoutStatus struct	Register Numbers
data[16384]	0x0000 - 0x7FFF
control	0x8000
status	0x8002
intLevel	0x8004
intVector	0x8006
geoAddr	0x8008
moduleReset	0x800A
firmwareRev	0x800C
selflashVME	0x800E
flashVME	0x8010
selflashUSER	0x8012
flashUSER	0x8014
configUSER	0x8016
scratch16	0x8018
res1[3]	0x801A
scratch32	0x8020
res2[110]	0x8024
configROM[127]	0x8100

Table 2-1: USER FPGA Registers and “struct” Equivalents

2.5 Brief Overview of FPGAs

FPGAs or “Field Programmable Gate Arrays” are electronic ICs -- typically digital -- that have the ability to be reprogrammed for different purposes. Quite often these circuits are used in the initial design phases of an IC when a hardwired circuit would be much more costly to produce for each revision of the product. Once the IC has been proven to work correctly, the design can then become hardwired onto an IC that can be mass produced.

In our experiment, we also use FPGAs for their ability to be reprogrammed, but we do not intend on mass-producing a hardwired version of our IC. We simply use its reprogrammable abilities to perform our highly specialized functionality. In this instance, we control the LVDS T1 signal going to the VFATs, extract the LVDS packets from the VFAT, store the data, and then pass it on to the ROC.

When referring to electronics, HDL stands for “Hardware Description Language”. It is the term used for the code that describes hardware layout. For this reason it is also commonly referred to as “firmware” as it is neither completely software nor hardware. There are many different HDL languages, but the only two utilized for this project are VHDL (Very High Speed HDL) and Verilog. Most of the modules that the user will even want to change are those in VHDL; it is therefore recommended that if one wants to alter the user firmware for the V1495 that one spends most of one’s time learning VHDL. There is only one module in Verilog, V1495usr_hal.vqm, the purpose of which is described in its respective section.

HDL’s utility lies in its ability to describe hardware using a behavioral, a dataflow, or a structural model to describe the circuit one wants to build.

A behavioral model is one in which the user uses more traditional software flow-control statements to control the logical properties of the circuit; at this level in VHDL, these include “Process” statements, “If” statements, “Case” statements, Loop statements (e.g. “Do”, ”While”), “Next”/”Exit” statements, as well as “NULL” statements. It is at this level that the classic Mealy and Moore state machines are implemented.

Dataflow modeling is similar to behavioral modeling except instead of using flow-control similar to other traditional programs, one uses concurrent assignment operators to control the logical flow of data with operators such as “OR”, “AND”, “XOR”, etc. It has some conditional control like behavioral modeling, but it remains more low-level. It is conceivable that we could have used this form of modeling for the register-access modules, but these modules are in behavioral form of modeling.

Structural modeling is virtually a literal translation of traditional digital logic symbols (e.g. AND-gates, OR-gates, etc) into a software format. In structural modeling one uses previously-defined components to create the necessary logical operations by connecting their I/O’s properly. This form of modeling is very similar to Object Oriented Programming; the components can be thought of as the classes and the logical flow can be thought of as using the

specific instantiations of the classes/components. For this thesis, we do not utilize this form of modeling either.

2.6 Brief Overview of Quartus II

Instructions for Downloading and running Quartus II

For this project we use the free Quartus II Web Edition Software. At the time of publication for this thesis, it is available at the following web address: http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html

Fortunately, for the purposes of this project, we only need to use it for its ability to handle and compile modules for the Cyclone family of FPGAs. It is, however, also useful for viewing the "Help" files on a number of modules in this project as well as viewing the pinouts for the FPGA (Assignments->Pins menu in Quartus). Although this level of detail is not necessary for this project, it does help one understand which signals are internal to the FPGA and therefore capable of being altered by the user firmware on the v1495, and which signals are external to the FPGA and not capable of being altered by the user firmware.

2.7 V1495_USR_firmware

2.7.1 Overview

The V1495_USR_firmware is provided by CAEN for the purpose of demonstrating how to use the v1495 module. This firmware is provided as well on the CD included with this thesis but can also be found on CAEN's website. A number of functions are provided to introduce the V1495 user to the utilities of the V1495.

2.8 GEMReadout Firmware

2.8.1 VHDL Modules

v1495usr.vhd

This is the lowest hierarchical level for V1495 user firmware. It includes all of the I/Os as can be seen on the Pinout for the project. The Pinout for any project in Quartus II can be found in the Assignments->Pins menu.

tristate_if_rtl.vhd

This module simply contains the logic necessary to control the direction of the FPGA data bits found in v1495usr.vhd and the logic to control the direction of the bidirectional I/Os associated with the VME data bus. Although there is probably little reason to change this section of code a brief description is provided below.

By default the FPGA data bits are set at High-Z (as can be seen by following the signal into v1495usr_hal.vqm). Since these bits are not used in this project this is of little consequence.

More importantly, the direction of the "LAD" bits (Local Data Bus) is controlled by "LAD_OE". If "LAD_OE" is high then the "LAD" bits will be configured as outputs. As reason would predict, if "LAD_OE" is low then the "LAD" bits will be inputs. Ultimately the value of "LAD_OE" is controlled by convoluted code found in v1495usr_hal.vqm. It should be noted that in GEMReadout.vhd the signals that determines whether a read or write operation is to be performed are "REG_RDEN" and "REG_WREN" respectively. Again, these signals are generated in the convoluted logic found in v1495usr_hal.vqm.

spare_if_rtl.vhd

This module simply contains the logic necessary to control the direction of the spare bidirectional I/Os included with the Cyclone FPGA. These are not utilized with the current version of the GEMReadout program but could potentially be adapted for future utility by CAEN for the V1495. It would be difficult to adapt their usage at this level of design because we are not privy to the inner schematics of the V1495 board to see where these signals go.

GEMReadout.vhd

This module handles all of the receiving/transmitting functionality to/from the VFAT ICs, the FIFO storage, the PLL timing (if used), control of the LEDs on the V1495 and the ROC interface capabilities via the VME bus.

As previously stated, it is not the intent of this thesis to clearly describe how VHDL works which is needed if one wants to be able to read the code and precisely determine how this module works. A UML has been provided to help elucidate the operation of this module. See

section “Inputs and Outputs” for a description of the I/Os for the V1495 including the LEDs.

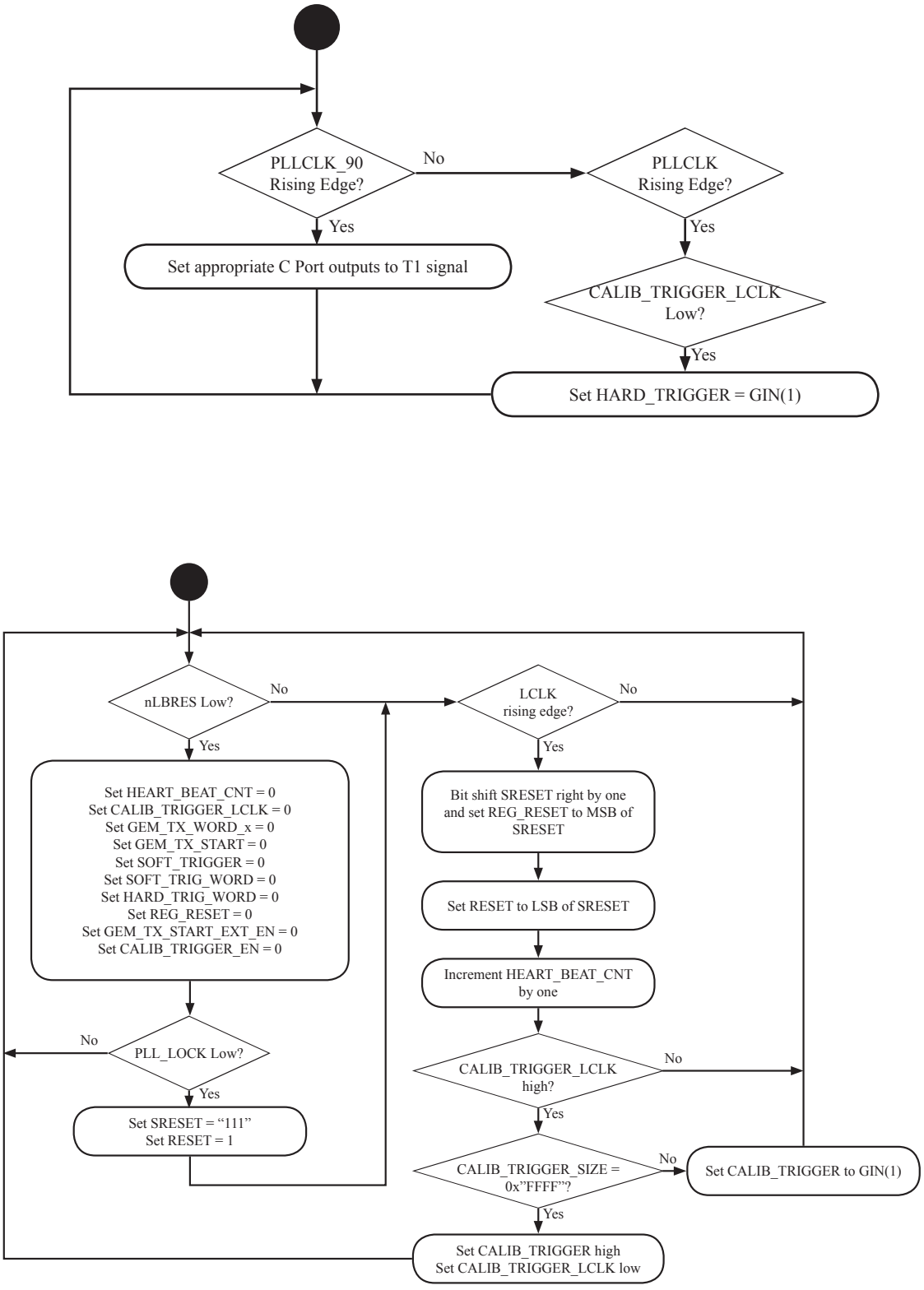
GEMTrigger.vhd

The GEMTrigger module handles the conversion of a single-pulse trigger to a 3-bit (or 8-bit in the case of the Calibration Pulse signal) trigger word which is sent to the T1 line of the VFATs which is in sync with the MCLK. The single-pulse trigger can be generated as a hard trigger from the G0 port, as a soft trigger internally, as a reset trigger internally, or even as a calibration trigger which can be generated either internally or externally depending on the value written to the A_GEM_CALIB_START register. Each of these triggers is more thoroughly described below.

A hard trigger from the G0 input port when active generates whatever signal is located in HARD_TRIG_WORD. By default this is the LV1A signal, “100”, as this is probably used the most often. From the VFAT2 Manual we know that this is a request to the VFAT for a data packet of all of the hits on the lines. For a more thorough explanation of this process see the section in the VFAT2 Manual on page covering an LV1A request. HARD_TRIG_WORD is

GEMReadout	#Bits	I/O	Description
nLBRES	1	In	Async Reset (active low)
LCLK	1	In	Local Bus Clock
REG_WREN	1	In	Write pulse (active high)
REG_RDEN	1	In	Read pulse (active high)
REG_ADDR	16	In	Register address
REG_DIN	16	In	Data from CAEN Local Bus
REG_DOUT	16	Out	Data to CAEN Local Bus
USR_ACCESS	1	In	Current register access is
A	32	In	In A (32 x LVDS/ECL)
B	32	In	In B (32 x LVDS/ECL)
C	32	Out	Out C (32 x LVDS)
SELG	1	Out	Output Level Select (NIM/TTL)
nOEG	1	Out	Output Enable
GOUT	2	Out	Out G - LEMO (2 x NIM/TTL)
GIN	2	In	In G - LEMO (2 x NIM/TTL)
IDD	3	In	D slot mezzanine Identifier
SELD	1	Out	D slot Port Signal Standard Select
nOED	1	Out	D slot Port Direction
D	32	In	D slot Data In Bus
IDE	3	In	E slot mezzanine Identifier
SELE	1	Out	E slot Port Signal Standard Select
nOEE	1	Out	E slot Port Direction
E	32	Out	E slot Data In Bus
IDF	3	In	F slot mezzanine Identifier
SELF	1	Out	F slot Port Signal Standard Select
nOEF	1	Out	F slot Port Direction
F	32	In	F slot Data Out Bus
SPARE_OUT	12	Out	Spare Data Out
SPARE_IN	12	In	Spare Data In
SPARE_DIR	12	Out	Spare Direction (0 => Out, 1 => In)
RED_PULSE	1	Out	RED Led Pulse (active high)
GREEN_PULSE	1	Out	GREEN Led Pulse (active high)

UML Diagram 2-1: GEMReadout.vhd



UML Diagram 2-2: GEMReadout.vhd

programmable by writing to bits 5-3 in register A_GEM_TRIG_WORD (offset + 0x0014). See Table 2-1 on page 16 for a list of the available registers on the v1495.

A soft trigger is similar to a hard trigger except that is requested by a write operation to the A_GEM_SOFT_TRIG register. The SOFT_TRIGGER_WORD is located in bits 2-0 in register A_GEM_TRIG_WORD (offset + 0x0014). The default value for this word is “000”.

The calibration trigger is the most complicated of all the triggers. Its main purpose is for quickly producing the S-curves necessary to adjust the threshold values for the channels on the VFATs high enough to reject systemic noise but still low enough to detect sensitive, real signals. A single sequence of the calibration trigger is comprised of a CalPulse signal , “110”, followed by a single blank clock cycle which is then followed by an LV1A signal, “100” to the T1 line.

There are two different ways of using the calibration trigger pulse. A write to the A_GEM_CALIB_START (offset + 0x000E) register with words 0x0000 – 0xFFFE (anything other than 0xFFFF) will cause a single burst of the calibration pulse sequence immediately following the write request. Writing 0xFFFF to this register causes the calibration pulse sequence to only pulse when a single external trigger pulse is applied to the G0 input. This can be done indefinitely and is changed by writing a value other than 0xFFFF to the aforementioned register. Originally it was intended to have the number being written to this register control how many times the calibration pulse sequence was executed, but this functionality awaits future development.

v1495usr_pkg.vhd

This module simply contains the offset address for all of the user registers of the v1495. Table 2-1 listed in “Addressing the V1495” gives a detailed description of the names and addresses of these registers as listed in this module as well as their equivalent names in v1495Lib.o data structures.

PLLBlock.vhd

A PLL, Phase Lock Loop, is a timing stabilization mechanism. It is available to the users of the Cyclone for clock frequencies between 15Mhz and 1Ghz. At these extreme frequencies clock phase synchronization can become problematic for the internal hardware of the FPGA. By using at least one of the two available PLLs on the Cyclone FPGA, one can mitigate the clock phase synchronization error and thus achieve a much higher running frequency. For frequencies below 15Mhz the PLLBlock module is completely bypassed in the firmware. Usually these versions of the firmware include the suffix “NoPLL” in their names to indicate the difference.

For the GEMReadout user firmware this module takes care to replicate the original MCLK

signal coming in on G0 and outputting it to the PLLCLK signal (when referring to the VFAT is the MCLK). It also creates a clock signal that is an exact inverse of this signal. This signal has the misnomer PLLCLK_90. Since the VFATs change their output bits on a rising edge of MCLK and evaluate incoming bits on the falling edge of MCLK, the inputs and outputs to the v1495 are evaluated and changed on the rising edge of a clock 180 degrees out of phase with this (i.e. PLLCLK_90).

The main portion of this module uses the "altpll" component from the Quartus II library. For a complete description of all of the attributes of this module see the help files in Quartus II on "altpll". There are, however, a number of attributes that the user may need to change in order to run MCLK for the v1495 at a specific frequency between 15Mhz and 1Ghz. A listing and a brief description of each of these is provided below.

inclk0_input_frequency: This is the value given for the frequency of the primary clock. The value is the period given in pico seconds and is not enclosed in quotations. This is not listed in the help-file literature. Example: A value of 25000 would give a primary running frequency of 40Mhz since 1/25000*10^-12 = 40Mhz

clk1_phase_shift: This is the value given for the amount of phase shift on the secondary clock output relative to the primary clock output. This, too, is measured in picoseconds, but this attribute's value does require quotations. Example: For the previous examples of 40Mhz, if the user requires a phase shift of 180 degrees the value of "12500" must be entered (i.e. half of the frequency period).

GEMRxEventDataFIFO.vhd/ GEMRxEventSizeFIFO.vhd

These modules both contain the specific layout of the DataFIFO and the SizeFIFO and both are extremely similar in layout.

The dcfifo, of which the DataFIFO and SizeFIFO are comprised, is a true dual-port FIFO (one clock for reading and one for writing as long as it is not the same node) using M4K technology from the Cyclone family of devices. For reference to this device structure see the Quartus II help files on "dcfifo Megafunction" where a more detailed description of the parameters for specific parameters can be found.

The dcfifo megafunction works precisely as one would expect a FIFO to work. When a write request is received by the FIFO, the word currently on "data[]" will be written into the FIFO (as long as the FIFO is not full). For the DataFIFO this is the current word taken from the DataOut stream. If the FIFO is full, the FIFO will ignore the write request. Therefore, care must be taken to ensure that the FIFO does not reach capacity or else the V1495 will start dropping packets received from the VFATs. As long as the FIFO is not empty, when a read request is received the FIFO will place the oldest word in the FIFO on the "q[]" signal. In the case of the DataFIFO this is "EVENT_DATA.". In the case of the SizeFIFO this is "EVENT_SIZE." A detailed description of how the DataFIFO and SizeFIFO are used for our project can

be found in the section GEMRxChannel.

For our project some of the more important parameters for the DataFIFO are as follows:

- Number of words the DataFIFO can store = 1024
- Bit width of the data being read and written to the FIFO = 16
  - N.B. This matches the width of the data bus on the VME.
- Bit width of the number of words that are currently in the FIFO = 10
  - N.B. This matches the number of bits needed to report the maximum number of words that can be stored in the FIFO.
- The parameters for the SizeFIFO are as follows:
- Number of words the DataFIFO can store = 64
- Bit width of the data being read and written to the FIFO = 4
  - N.B. This matches the width of the data bus on the VME.
- Bit width of the number of words that are currently in the FIFO = 6
  - N.B. This matches the number of bits needed to report the maximum number of words that can be stored in the FIFO.

GEMRxChannel.vhd

The GEMRxChannel.vhd module governs the behavior of the data received from the VFATs on their respective DataOut and DataValid channels. This is shown below in UML Diagram 2-3 on page 25 for this module. For a more detailed description of the parameters settings for these FIFOs see section “GEMRxEventDataFIFO.vhd/ GEMRxEventSizeFIFO.vhd.”

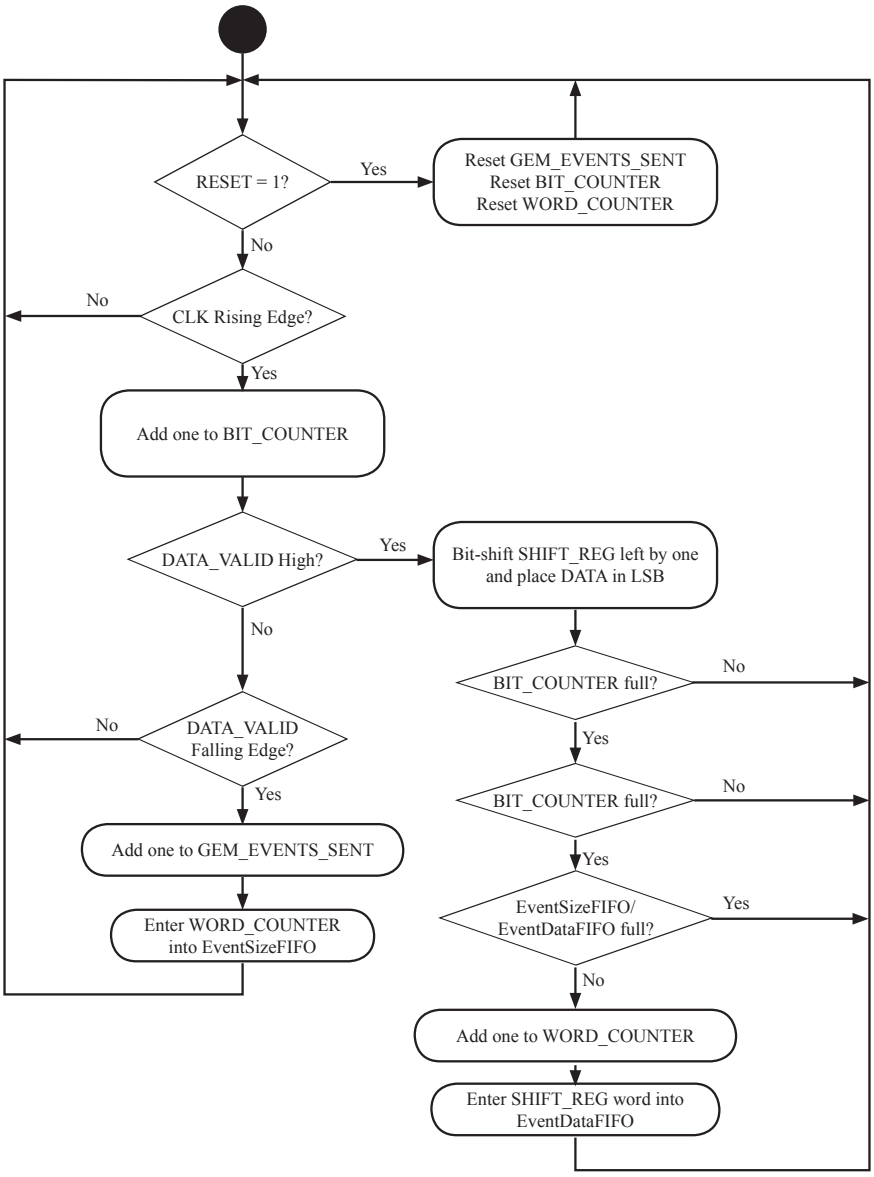
GEMTxChannel.vhd

This section was not utilized during the final stages of this project. It was used (along with the GEMReadout_tb) during the initial stages to provide debugging methods for the basic functionality of the I/Os on the V1495. It structure is simply as shown in UML Diagram 2-4 on page 26.

GEMReadout_tb.vhd

GEMRxChannel		GEMReadout	#Bits	I/O
LCLK	=>	LCLK	1	In
RD_EVENT_SIZE	=>	GEMx_RD_EVENT_SIZE	1	In
EVENT_SIZE	=>	GEMx_EVENT_SIZE	4	Out
EVENT_COUNT	=>	GEMx_EVENT_COUNT	6	Out
RD_EVENT_DATA	=>	GEMx_RD_EVENT_DATA	1	In
EVENT_DATA	=>	GEMx_EVENT_DATA	16	Out
EVENT_DATA_SIZE	=>	GEMx_EVENT_DATA_SIZE	10	Out
GEM_EVENTS_SENT	=>	GEMx_EVENTS_SENT	32	Out
CLK	=>	PLLCLK	1	In
RESET	=>	RESET	1	In
DATA	=>	GEMx_DATA	1	In
DATA_VALID	=>	GEMx_DATA_VALID	1	In

where 'x' is for the respective VFAT

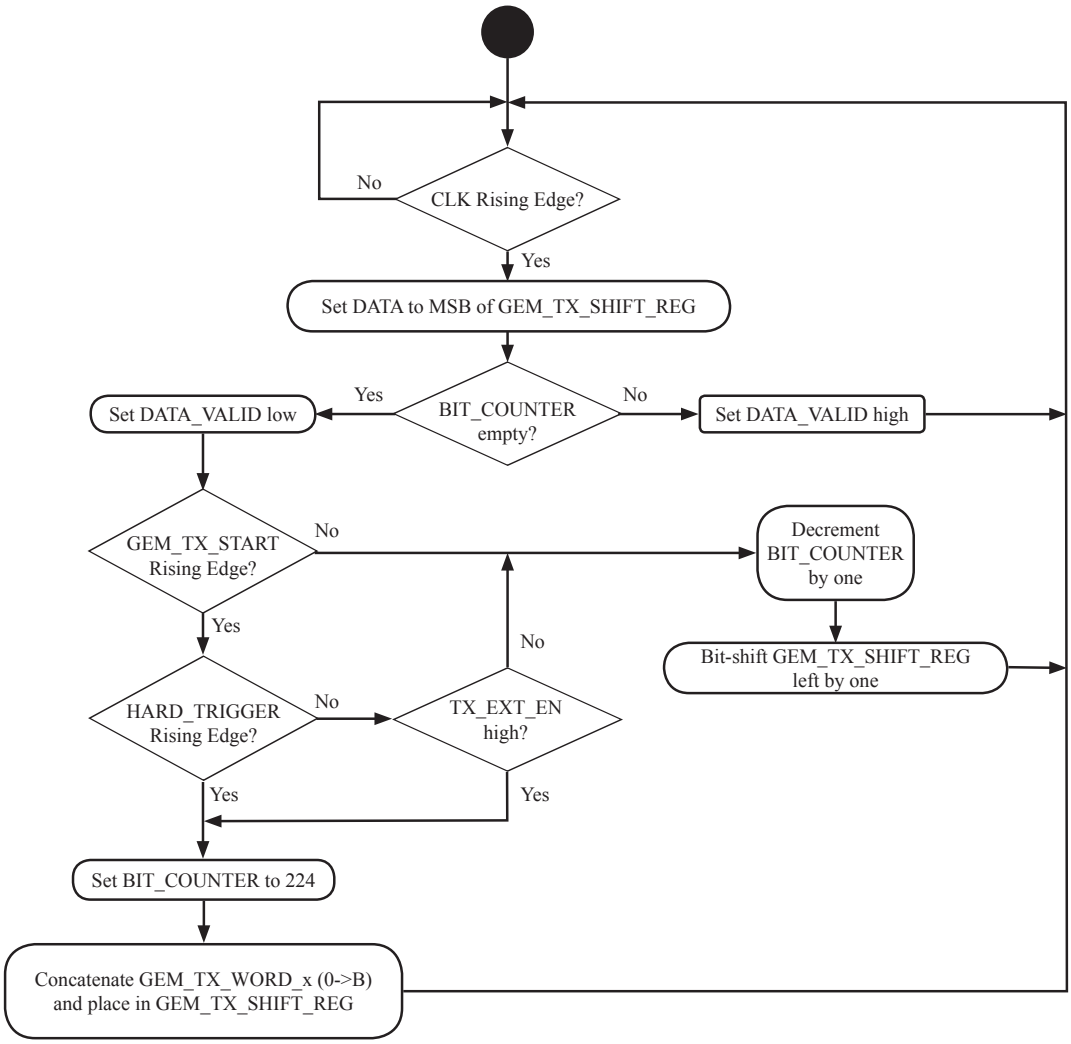


UML Diagram 2-3:GEMRxChannel.vhd

Although it was not used in the final stages of this project (that which was conducted at ISU), this module provides a useful test bench with which to test the basic I/O functionality of the V1495. This is particularly useful when one does not have access to VFAT chips for generating and capturing data packets. See UML Diagram 2-5 on page 27 and UML Diagram 2-6 on page 28.

GEMTxChannel		GEMReadout	#Bits	I/O
LCLK	=>	LCLK	1	In
GEM_TX_WORD_x	=>	GEM_TX_WORD_x	16	In
GEM_TX_START	=>	GEM_TX_START	1	In
CLK	=>	PLLCLK	1	In
TX_EXT_EN	=>	GEM_TX_START_EXT_EN	1	In
HARD_TRIGGER	=>	HARD_TRIGGER	1	In
DATA	=>	GEM_TX_DATA	1	Out
DATA_VALID	=>	GEM_TX_DATA_VALID	1	Out

where 'x' is for the respective TX_WORD (0-B)



UML Diagram 2-4:GEMTxChannel.vhd

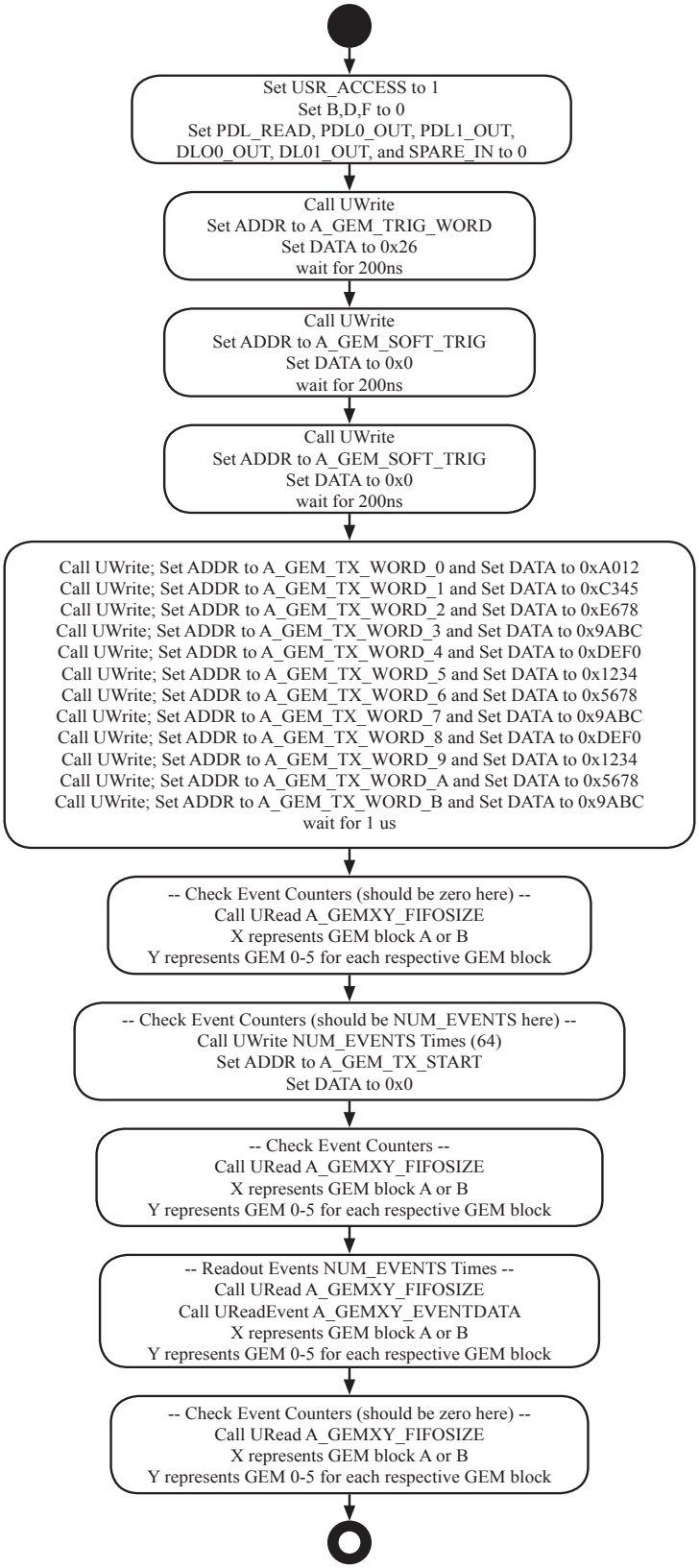
2.8.2 Verilog Modules

v1495usr_hal.vqm

This module contains many of the logical functions that serve as a building foundation for the v1495 module. These have been automatically generated using the Synopsys Synplify

GEMReadout_tb	#Bits	I/O	Description
nLBRES	1	In	Async Reset (active low)
LCLK	1	In	Local Bus Clock
REG_WREN	1	In	Write pulse (active high)
REG_RDEN	1	In	Read pulse (active high)
REG_ADDR	16	In	Register address
REG_DIN	16	In	Data from CAEN Local Bus
REG_DOUT	16	Out	Data to CAEN Local Bus
USR_ACCESS	1	In	Current register access is
A	32	In	In A (32 x LVDS/ECL)
B	32	In	In B (32 x LVDS/ECL)
C	32	Out	Out C (32 x LVDS)
SELG	1	Out	Output Level Select (NIM/TTL)
nOEG	1	Out	Output Enable
GOUT	2	Out	Out G - LEMO (2 x NIM/TTL)
GIN	2	In	In G - LEMO (2 x NIM/TTL)
IDD	3	In	D slot mezzanine Identifier
SELD	1	Out	D slot Port Signal Standard Select
nOED	1	Out	D slot Port Direction
D	32	In	D slot Data In Bus
IDE	3	In	E slot mezzanine Identifier
SELE	1	Out	E slot Port Signal Standard Select
nOEE	1	Out	E slot Port Direction
E	32	Out	E slot Data In Bus
IDF	3	In	F slot mezzanine Identifier
SELF	1	Out	F slot Port Signal Standard Select
nOEF	1	Out	F slot Port Direction
F	32	In	F slot Data Out Bus
PDL_WR	1	Out	Write Enable
PDL_SEL	1	Out	PDL Selection (0=>PDL0, 1=>PDL1)
PDL_READ	8	In	Read Data
PDL_WRITE	8	Out	Write Data
PDL_DIR	1	Out	Direction (0=>Write, 1=>Read)
PDL0_OUT	1	In	Signal from PDL0 Output
PDL1_OUT	1	In	Signal from PDL1 Output
DLO0_OUT	1	In	Signal from DLO0 Output
DLO1_OUT	1	In	Signal from DLO1 Output
PDL0_IN	1	Out	Signal TO PDL0 Input
PDL1_IN	1	Out	Signal TO PDL1 Input
DLO0_GATE	1	Out	DLO0 Gate (active high)
DLO1_GATE	1	Out	DLO1 Gate (active high)
SPARE_OUT	12	Out	SPARE Data Out
SPARE_IN	12	In	SPARE Data In
SPARE_DIR	12	Out	SPARE Direction (0 => OUT, 1 => IN)
RED_PULSE	1	Out	RED Led Pulse (active high)
GREEN_PULSE	1	Out	GREEN Led Pulse (active high)

UML Diagram 2-5: GEMReadout_tb.vhd



UML Diagram 2-6: GEMReadout_tb.vhd

design entry/synthesis tool. Synopsys Synplify is used to create, synthesize, and optimize a project and then generate a Verilog Quartus Mapping file (.vqm) for compilation in the Quartus II software. Reference here to the Altera website:

http://www.altera.com/support/software/nativelink/synthesis/synplicity/eda_view_using_synplty.html