



## Zebra Technical Manual

### Revision

Revision	Date	Name	Description
0.1	22/02/2013	Tom Cobb	Initial structure
0.2	17/04/2013	Isa Uzun	Various sections brought together.
0.3	02/05/2013	Isa Uzun	First draft.
0.4	28/08/2013	Isa Uzun	Physical specifications added.
0.5	12/11/2013	Tom Cobb	Added schematic diagrams, reorganize
0.6	17/01/2014	Tom Cobb	Added position capture examples
0.7	04/02/2014	Tom Cobb	Updated based on DASC group suggestions
0.8	18/03/2014	Tom Cobb	Added Firmware 0x21 features
0.9	17/04/2014	Tom Cobb	Added Qt GUI launch instructions, updated pictures, added timing information

Prepared by: Tom Cobb, Isa Uzun, Yuri Chernousko

## Contents

<b>1. INTRODUCTION .....</b>	<b>4</b>
<b>2. PHYSICAL SPECIFICATIONS .....</b>	<b>4</b>
2.1 FRONT PANEL.....	4
2.2 FRONT PANEL STATUS LEDS .....	4
2.3 REAR PANEL .....	5
2.4 REAR PANEL STATUS LEDS .....	5
2.5 HARDWARE CONNECTIONS FOR STANDALONE SETUP .....	5
<b>3. SYSTEM DESIGN .....</b>	<b>6</b>
3.1 LOGIC BLOCK TIMING .....	6
3.2 DEFAULT SETUP .....	6
<b>4. EPICS INTERFACE.....</b>	<b>7</b>
4.1 STANDALONE SETUP WITH PREBUILT IOC.....	7
4.2 DIAMOND SETUP WITH CUSTOM IOC .....	7
4.3 THE SYSTEM BUS .....	8
4.4 AND/OR BLOCKS .....	10
4.5 GATE BLOCKS .....	11
4.6 DIV BLOCKS.....	12
4.7 PULSE BLOCKS .....	13
4.8 QUAD BLOCK AND POSITION COUNTERS.....	14
4.9 PC POSITION CAPTURE BLOCK .....	15
4.9.1 EXAFS example .....	18
4.9.2 Tomography example .....	19
4.9.3 Logic analyser example .....	20
4.9.4 External trigger example .....	21
<b>5. ELECTRICAL CONNECTIONS .....</b>	<b>22</b>
5.1 FRONT PANEL INPUTS .....	22
5.1.1 TTL Inputs .....	22
5.1.2 NIM Inputs .....	22
5.1.3 LVDS Inputs .....	23
5.1.4 Open Collector (OC) Input .....	23
5.1.5 Comparator (CMP) Input .....	24
5.1.6 PECL Input .....	24
5.2 FRONT PANEL OUTPUTS .....	24
5.2.1 TTL Output .....	25
5.2.2 NIM Output .....	25
5.2.3 LVDS Output .....	25
5.2.4 Open Collector (OC) Output .....	26
5.2.5 PECL Output .....	26
5.3 REAR PANEL .....	26
5.3.1 Encoder I/O ports .....	27
5.3.2 RS232 port .....	28
5.3.3 JTAG .....	29

5.3.4	Power.....	29
<b>6.</b>	<b>FPGA DETAILS .....</b>	<b>30</b>
6.1	CLOCKS.....	31
6.2	POSITION CAPTURE .....	31
<b>7.</b>	<b>SOFTWARE INTEGRATION .....</b>	<b>33</b>
7.1	REGISTER ADDRESS MAPPING.....	33
7.1.1	<i>Zebra Logic .....</i>	<i>33</i>
7.1.2	<i>Position Capture .....</i>	<i>36</i>
7.1.3	<i>System Status and Control .....</i>	<i>37</i>
7.2	RS232 PROTOCOL .....	37
7.2.1	<i>Register Write .....</i>	<i>38</i>
7.2.2	<i>Register Read .....</i>	<i>38</i>
7.2.3	<i>Configuration Store .....</i>	<i>38</i>
7.2.4	<i>Configuration Restore .....</i>	<i>39</i>
7.2.5	<i>Position Capture Data Offload .....</i>	<i>39</i>

## 1. INTRODUCTION

Zebra is a digital signal level converter and position capture box. It takes the form of a 1U metal box with front panel BNCs and LEMOs and rear panel encoder DB15 connectors. Its function is to take front panel single channel inputs (TTL, LVDS, PECL, NIM, Open Collector) and rear panel encoder signals (RS422 quadrature with Z channel), pass them through the FPGA to implement logic gates and position capture circuitry, and output them to front and rear panel outputs of the same format as inputs. This document will describe the physical interface of Zebra, the RS232 interface to the FPGA, and the EPICS software that can be used to control it.

## 2. PHYSICAL SPECIFICATIONS

Zebra is manufactured in the form of a 1U metal box with the following dimensions:

Description	Height	Height H	Depth T
Perforated top cover and base plate	1 U	43.7 mm	220 mm

### 2.1 Front Panel

Zebra's front panel (see Figure 1) has 12 inputs and 12 outputs labeled with their input type (E.g.



TTL). The voltage level conversion as described in more detail in sections 5.1 and 5.2.

Figure 1: Zebra 1U front panel

### 2.2 Front Panel Status LEDs

Under each group of 3 inputs or outputs is a status LED. This LED will light for a minimum of 20ms on a rising edge of any of the 3 signals. This means that any group containing an output with a frequency greater than 50Hz will show a solidly lit LED, as will a group containing a constantly high output.

On the right hand side there is also an array of LEDs indicating the status of each power line in Zebra.

### 2.3 Rear Panel

Zebra's rear panel (see Figure 2) has 4 DB15 connector pairs each with a female encoder input and male encoder output, a power connector with fuse and earth stud, along with RS232 and JTAG connectors. The electrical connections are described in more detail in section 5.3.



Figure 2: Zebra 1U rear panel

### 2.4 Rear Panel Status LEDs

To the left of each connector is a pair of status LEDs: a red LED indicates that the encoder is disconnected, and a green LED indicates that the encoder is connected. Zebra uses an internal pull-up resistor to check the status of both halves of the A and B signals of each encoder, if either is disconnected then both halves of the differential signal will drift high and the red LED will indicate the encoder is disconnected. It should be noted that some encoders disconnect their outputs to indicate a problem, so a red LED could mean a disconnected cable or an encoder in error state.

### 2.5 Hardware Connections for Standalone Setup

If you are planning on using Zebra with the standalone software setup as shown in section 4.1 then please refer to the following diagram:

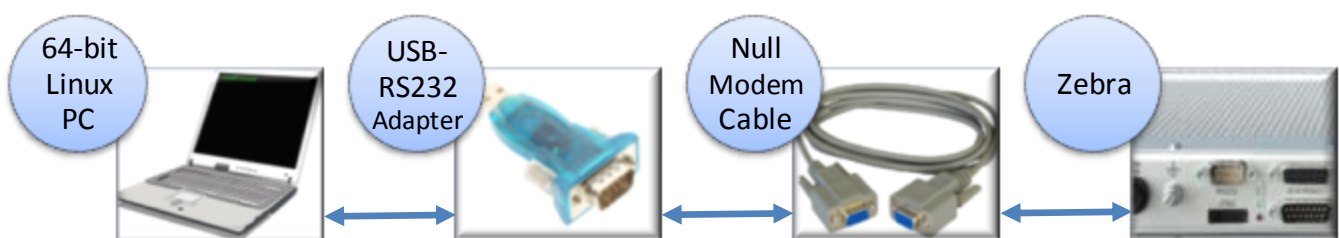


Figure 3: Zebra Standalone Setup

### 3. SYSTEM DESIGN

Inside the FPGA, there are a number of logic blocks, connected by the 64-bit wide system bus. This allows the input of any logic block to be taken from a physical input or the output from any other logic block. Each physical output is also taken from the system bus in the same way (see Figure 4).

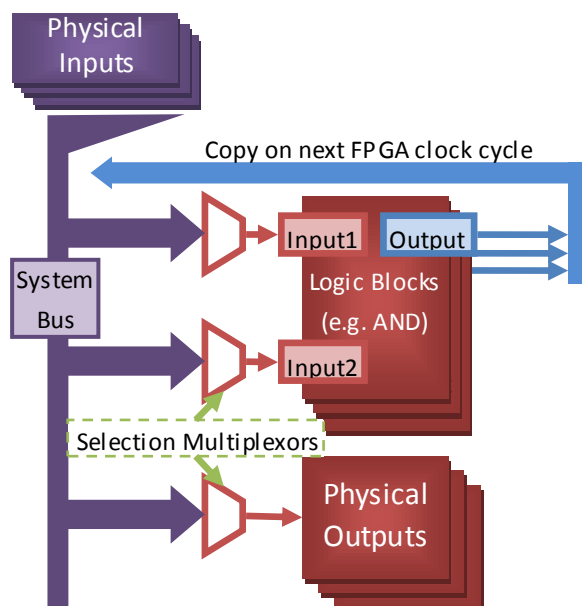


Figure 4: FPGA System Bus

#### 3.1 Logic Block Timing

On each FPGA clock tick (at 50MHz) the system bus values are created from the physical inputs and the last clock cycle's logic block outputs. Each logic block then uses a multiplexor to select a particular input from the bus, and prepares its outputs based on these inputs. This means that it will take 20ns for an input signal to get through each logic block, so an input signal that goes through 2 logic blocks and into an output will take 40ns to go through Zebra. For more details about the FPGA timing see section 6.

#### 3.2 Default setup

The default setup has all encoders passed straight through, and each front panel output to be the logic OR of the corresponding group of inputs. E.g. if IN\_TTL2, IN\_NIM2 or IN\_LVDS2 are high, OUT\_TTL2, OUT\_NIM2 and OUT\_LVDS2 will be high.

Zebra has some limited uses as a level converter with its default setup, but its real power comes when the logic blocks are rewired to suit the application. This is discussed in section 4 in the context of the EPICS interface.

CTRL		Doc No: TDI-CTRL-TNO-042 Issue: 0.7 Date: 4 <sup>th</sup> February 2014 Page: 7
------	---	--

## 4. EPICS INTERFACE

The EPICS support module Zebra (<http://controls.diamond.ac.uk/downloads/support/Zebra>) provides an EPICS driver that communicates with Zebra over a serial line.

### 4.1 Standalone setup with prebuilt IOC

If you download and unpack the Zebra support module, you will see a “startStandalone.sh” script in the root of the directory. This assumes you have connected Zebra to the PC as shown in Figure 3. It also assumes the PC runs a recent 64-bit Linux OS (tested on Red Hat Enterprise Linux 6.3 and Ubuntu 14.04) and that the USB-RS232 appears as /dev/ttyUSB0.

If you run the scrip it will start a precompiled IOC with the PV Prefix set to “TESTZEBRA” and then start the GUI in a new window. If you pass an argument to this script it will use this as the PV Prefix instead (you must do this if your PC is on the same network as another PC running the same Zebra application).

The standalone GUI is based on Qt rather than EDM, so there are some visual differences between this and the EDM GUI shown in the examples below. The main difference is that there is a table display of the position compare data rather than a graph widget. All the PVs serving the data still exist in the standalone IOC, but the GUI has not been written to display them at the moment.

If you are running position compare then please note that you will need to fill in the resolution and offset fields and tell Zebra the current position of the motor on the ENC tab of the GUI whenever you disconnect or power cycle Zebra.

### 4.2 Diamond setup with custom IOC

The standalone IOC illustrates how to make an instance of the zebra.template database and zebraConfig() start-up script function. If you make your own IOC you have the option of pointing the database template at a motor record instance for each encoder, which will provide it with figures for resolution and offset.

The sections below assume you have setup an IOC according to the instructions included with this module, and run up appropriate EDM screens for your IOC.

Figure 5 shows the basic elements of the GUI:

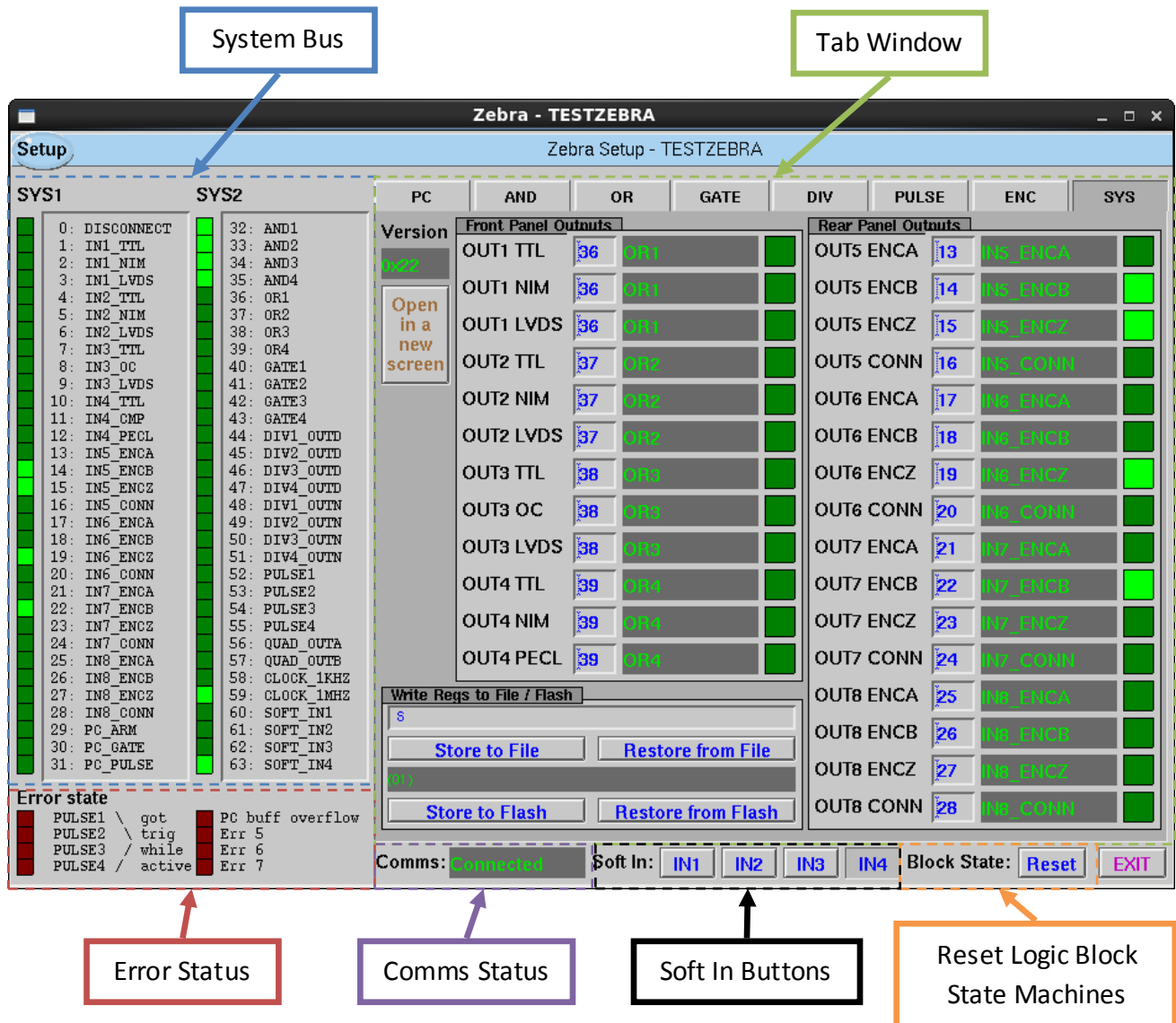


Figure 5: Screenshot of Zebra system setup

### 4.3 The System Bus

As described in section 3, there is a 64 signal system bus that connects Zebra's inputs, logic blocks, and outputs. In the above screenshot the current status of the System Bus is shown along with numbered and named elements. The SYS tab of the screen allows you to wire any element of the system bus to any outputs. For instance, if we wanted to have the output of the OR1 logic block be output from OUT1\_TTL, we would type the number 36 in the blue text entry box, and Zebra would display the name and current value of the signal in the green readback box.

Also on the SYS tab, we can see the following:

- **Open in New Screen:** This just opens the SYS tab window in a new screen so you can change the source of the outputs without swapping between tabs



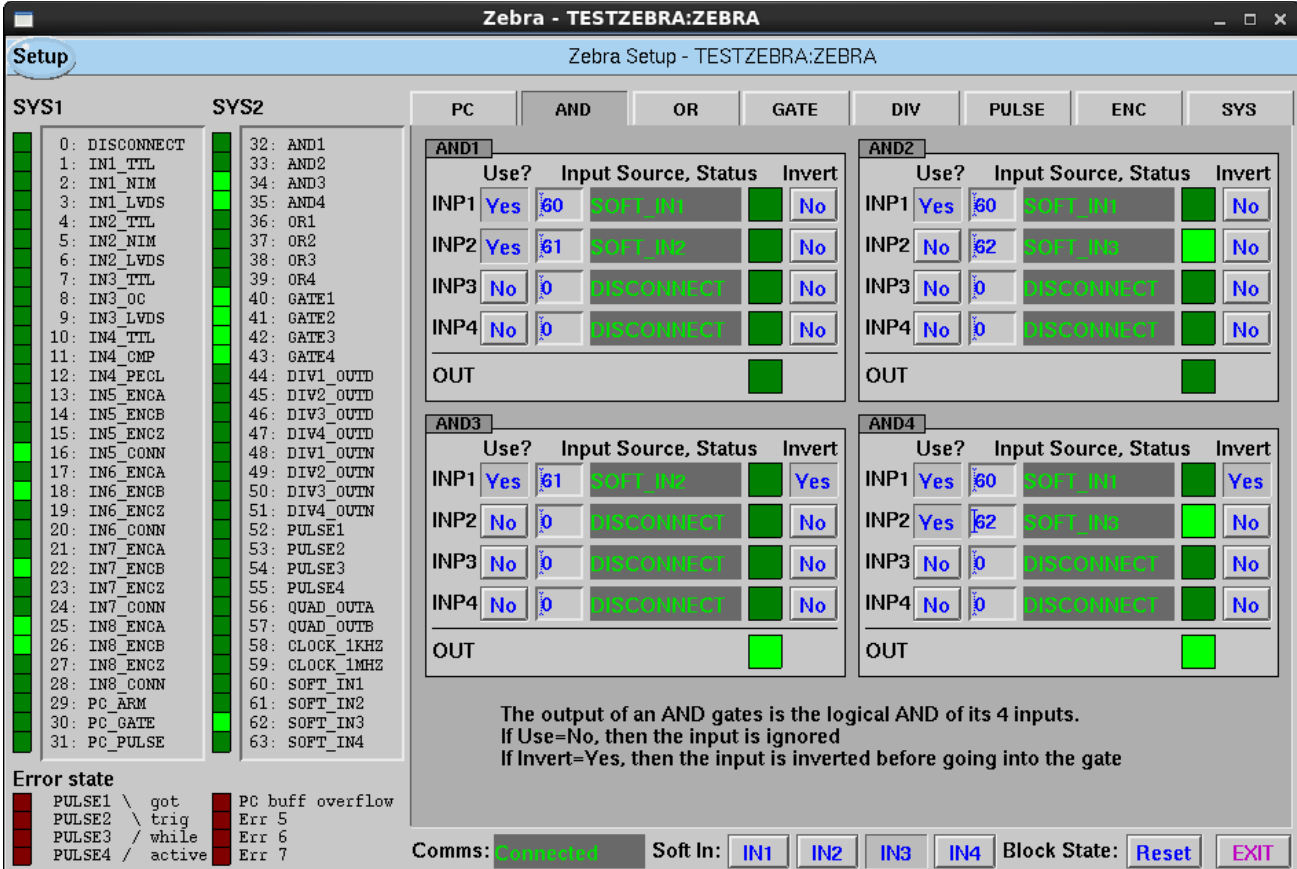
<b>CTRL</b>	 <b>diamond</b>	<b>Doc No: TDI-CTRL-TNO-042</b> <b>Issue: 0.7</b> <b>Date: 4<sup>th</sup> February 2014</b> <b>Page: 9</b>
-------------	--	---

- Version: The FPGA Firmware version number in Hex
- Store to File Button: Zebra's registers are being constantly polled, so the EPICS driver keeps a local cache of these registers. When this button is pressed, this cache is written to the filename specified.
- Restore from File Button: This will restore Zebra's register settings from a local .ini file specified in the box above. The box below gives the operation status. This is used to duplicate settings between multiple Zebras.
- Store to Flash: This will store Zebra's register settings to flash so that they will be restored after a power cycle.
- Restore from Flash: This will restore Zebra's register settings from flash. It does the same as power cycling Zebra.

On the rest of the screen we can also see:

- Comms status: Zebra's System Bus and Error Status are polled continuously at 1Hz. If Zebra fails to respond this field will show "Not Connected"
- Soft In Buttons: These toggle the state of the SOFT\_IN1..4 elements on the System Bus
- Reset: This button will reset all state machines to their default state:
  - Position capture will disarm and stop downloading points
  - Gate blocks outputs will be reset to low
  - Div blocks will go back to waiting for their initial pulse with their counters reset to 0
  - Pulse blocks will be reset to waiting for trigger
  - Quad blocks will be reset to their initial state with both outputs low.
 Parameters are retained even if they haven't been written to flash:

#### 4.4 AND/OR Blocks



**Zebra - TESTZEBRA:ZEBRA**

Zebra Setup - TESTZEBRA:ZEBRA

**SYS1**

- 0: DISCONNECT
- 1: IN1\_TTL
- 2: IN1\_NIM
- 3: IN1\_LVDS
- 4: IN2\_TTL
- 5: IN2\_NIM
- 6: IN2\_LVDS
- 7: IN3\_TTL
- 8: IN3\_OC
- 9: IN3\_LVDS
- 10: IN4\_TTL
- 11: IN4\_CMP
- 12: IN4\_PEC
- 13: IN5\_ENCA
- 14: IN5\_ENCB
- 15: IN5\_ENCC
- 16: IN5\_CONN
- 17: IN6\_ENCA
- 18: IN6\_ENCB
- 19: IN6\_ENCC
- 20: IN6\_CONN
- 21: IN7\_ENCA
- 22: IN7\_ENCB
- 23: IN7\_ENCC
- 24: IN7\_CONN
- 25: IN8\_ENCA
- 26: IN8\_ENCB
- 27: IN8\_ENCC
- 28: IN8\_CONN
- 29: PC\_ARM
- 30: PC\_GATE
- 31: PC\_PULSE

**SYS2**

- 32: AND1
- 33: AND2
- 34: AND3
- 35: AND4
- 36: OR1
- 37: OR2
- 38: OR3
- 39: OR4
- 40: GATE1
- 41: GATE2
- 42: GATE3
- 43: GATE4
- 44: DIV1\_OUTD
- 45: DIV2\_OUTD
- 46: DIV3\_OUTD
- 47: DIV4\_OUTD
- 48: DIV1\_OUTN
- 49: DIV2\_OUTN
- 50: DIV3\_OUTN
- 51: DIV4\_OUTN
- 52: PULSE1
- 53: PULSE2
- 54: PULSE3
- 55: PULSE4
- 56: QUAD\_OUTA
- 57: QUAD\_OUTB
- 58: CLOCK\_1KHZ
- 59: CLOCK\_1MHZ
- 60: SOFT\_IN1
- 61: SOFT\_IN2
- 62: SOFT\_IN3
- 63: SOFT\_IN4

**AND1**

Use?	Input Source	Status	Invert
Yes	60	SOFT_IN1	No
Yes	61	SOFT_IN2	No
No	0	DISCONNECT	No
No	0	DISCONNECT	No

**AND2**

Use?	Input Source	Status	Invert
Yes	60	SOFT_IN1	No
No	62	SOFT_IN3	No
No	0	DISCONNECT	No
No	0	DISCONNECT	No

**AND3**

Use?	Input Source	Status	Invert
Yes	61	SOFT_IN2	Yes
No	0	DISCONNECT	No
No	0	DISCONNECT	No
No	0	DISCONNECT	No

**AND4**

Use?	Input Source	Status	Invert
Yes	60	SOFT_IN1	Yes
Yes	62	SOFT_IN3	No
No	0	DISCONNECT	No
No	0	DISCONNECT	No

The output of an AND gates is the logical AND of its 4 inputs.  
If Use=No, then the input is ignored  
If Invert=Yes, then the input is inverted before going into the gate

**Error state**

- PULSE1 \ got
- PULSE2 \ trig
- PULSE3 / while
- PULSE4 / active
- PC buff overflow
- Err 5
- Err 6
- Err 7

Comms: **Connected** Soft In: **IN1 IN2 IN3 IN4** Block State: **Reset EXIT**

Figure 6: Screenshot of Zebra AND blocks

The AND blocks and OR blocks work in a similar way. Both mask out unused inputs according to the "Use" button, invert the inputs according to the "Invert" button, then take the logical AND/OR of those inputs. The example below shows a logical AND, the logical OR is similar.

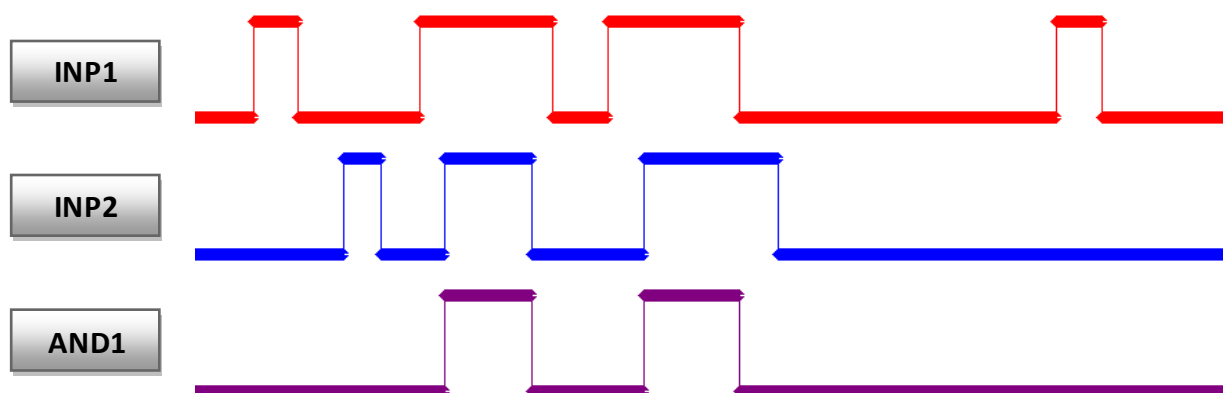


Figure 7: AND block example (AND1 setup as shown in Figure 6)

## 4.5 GATE Blocks

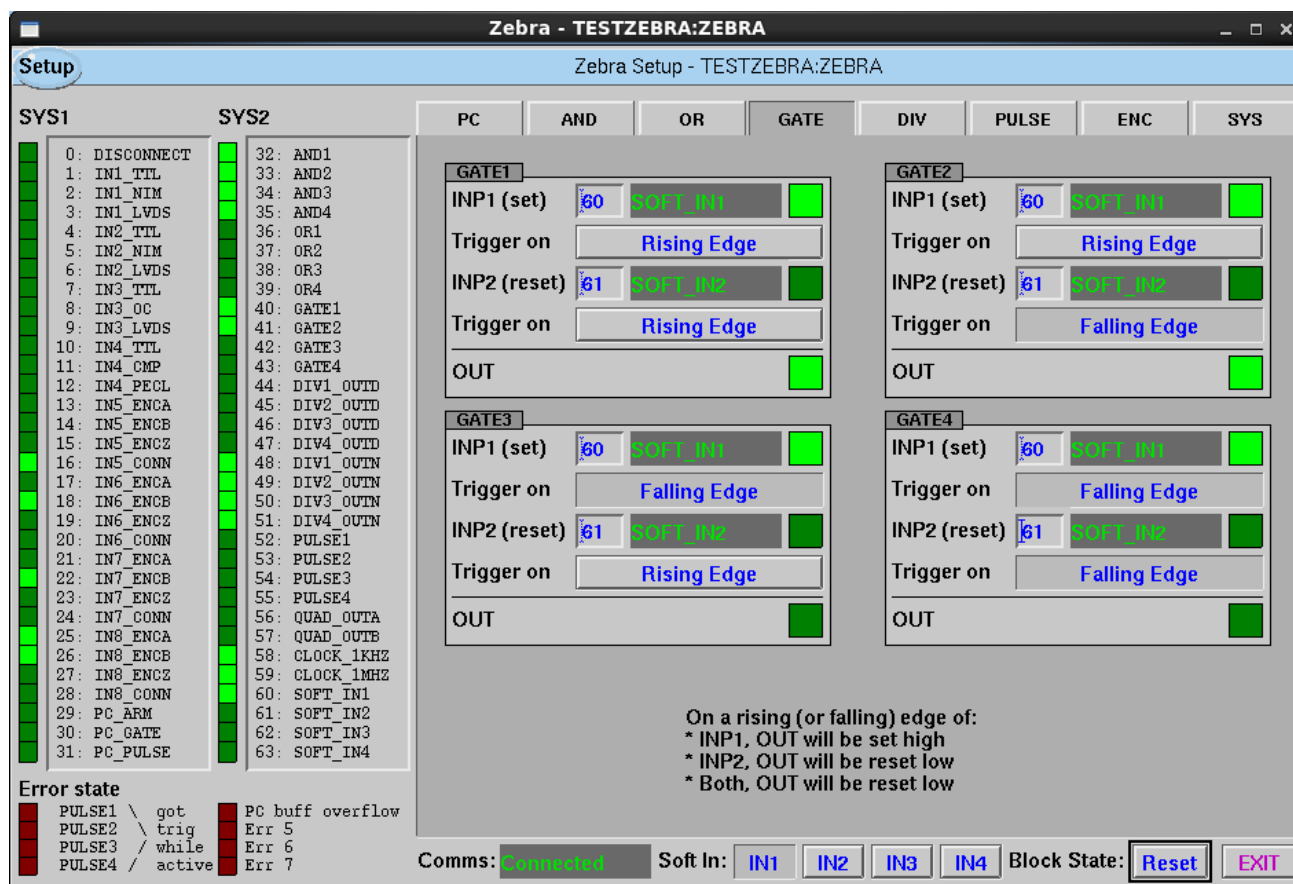


Figure 8: Screenshot of Zebra GATE blocks

GATE blocks have an internal state that can be modified by the rising (or optionally falling) edge of either its inputs. On the selected edge of INP1, the internal state is set high. On the selected edge of INP2 (or if both edges happen in the same clock tick) the internal state is set low. The behaviour is similar to an SR flip-flop or bistable, although the GATE block is edge triggered rather than level triggered.

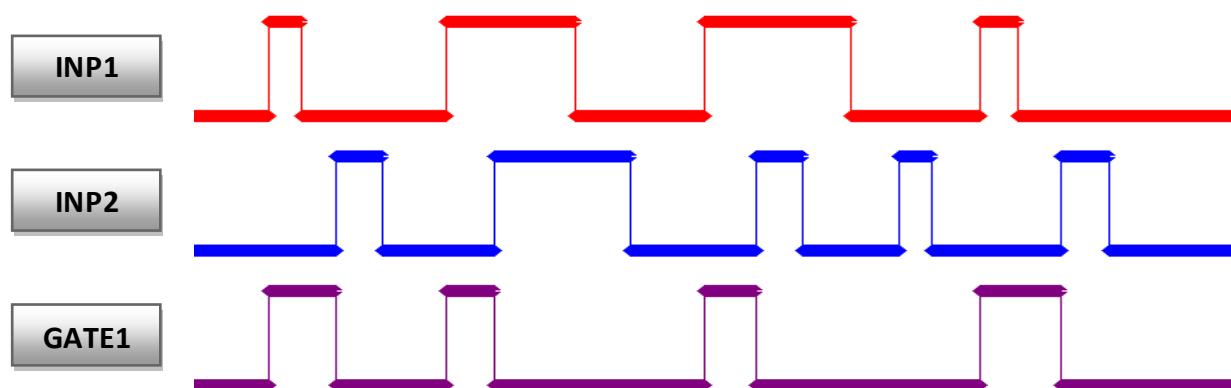


Figure 9: GATE block example (GATE1 setup as shown in Figure 8)

## 4.6 DIV Blocks

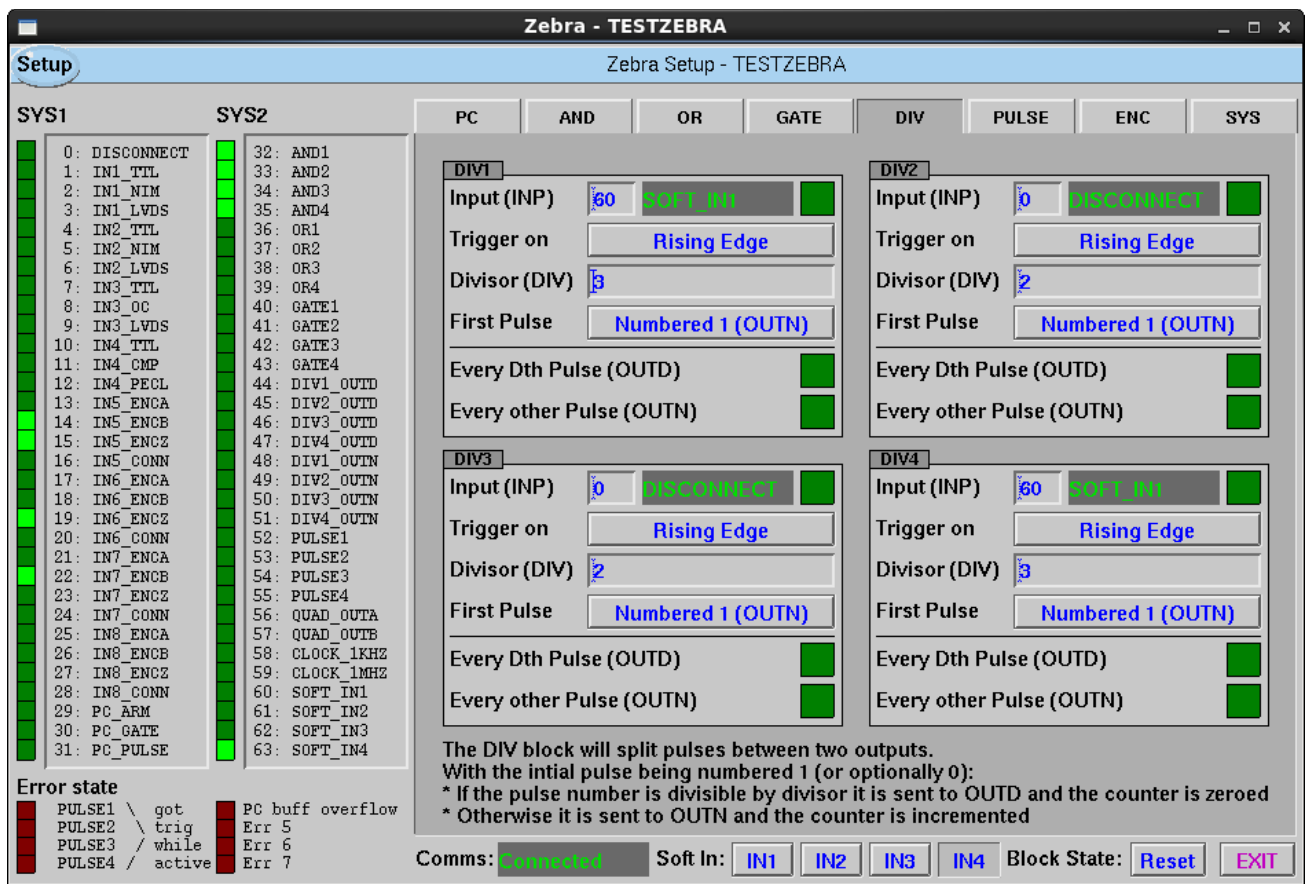


Figure 10: Screenshot of Zebra DIV blocks

DIV blocks have an internal counter that counts from 0 to DIV-1. On each rising (or optionally falling) edge, if the counter == DIV-1, then it is set to 0 and the pulse is sent to OUTD, otherwise the counter increments and the pulse is sent to OUTN. DIV can be in the range 0 to  $2^{32} - 1$

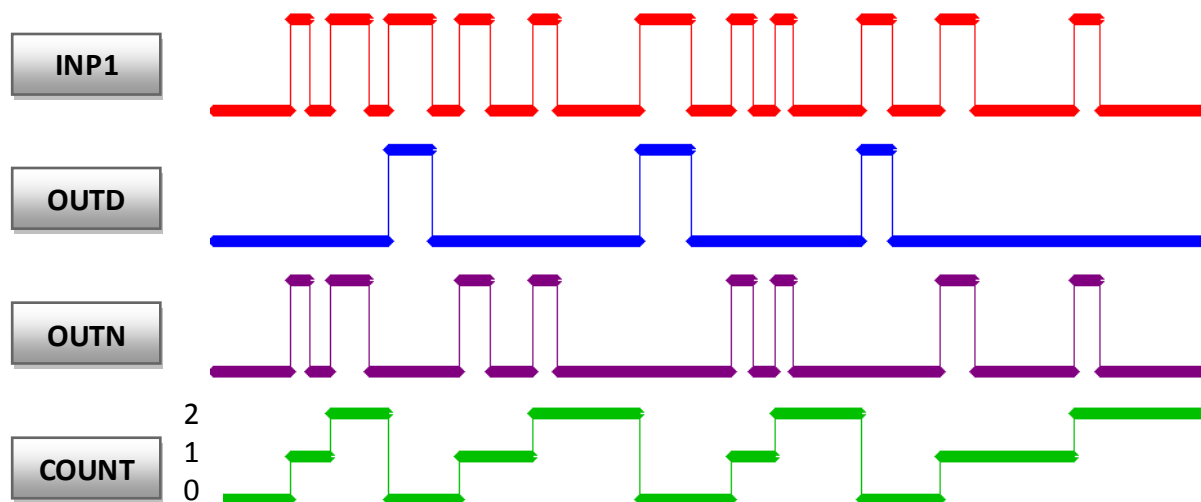


Figure 11: DIV block example (DIV1 setup as shown in Figure 10)

## 4.7 PULSE Blocks

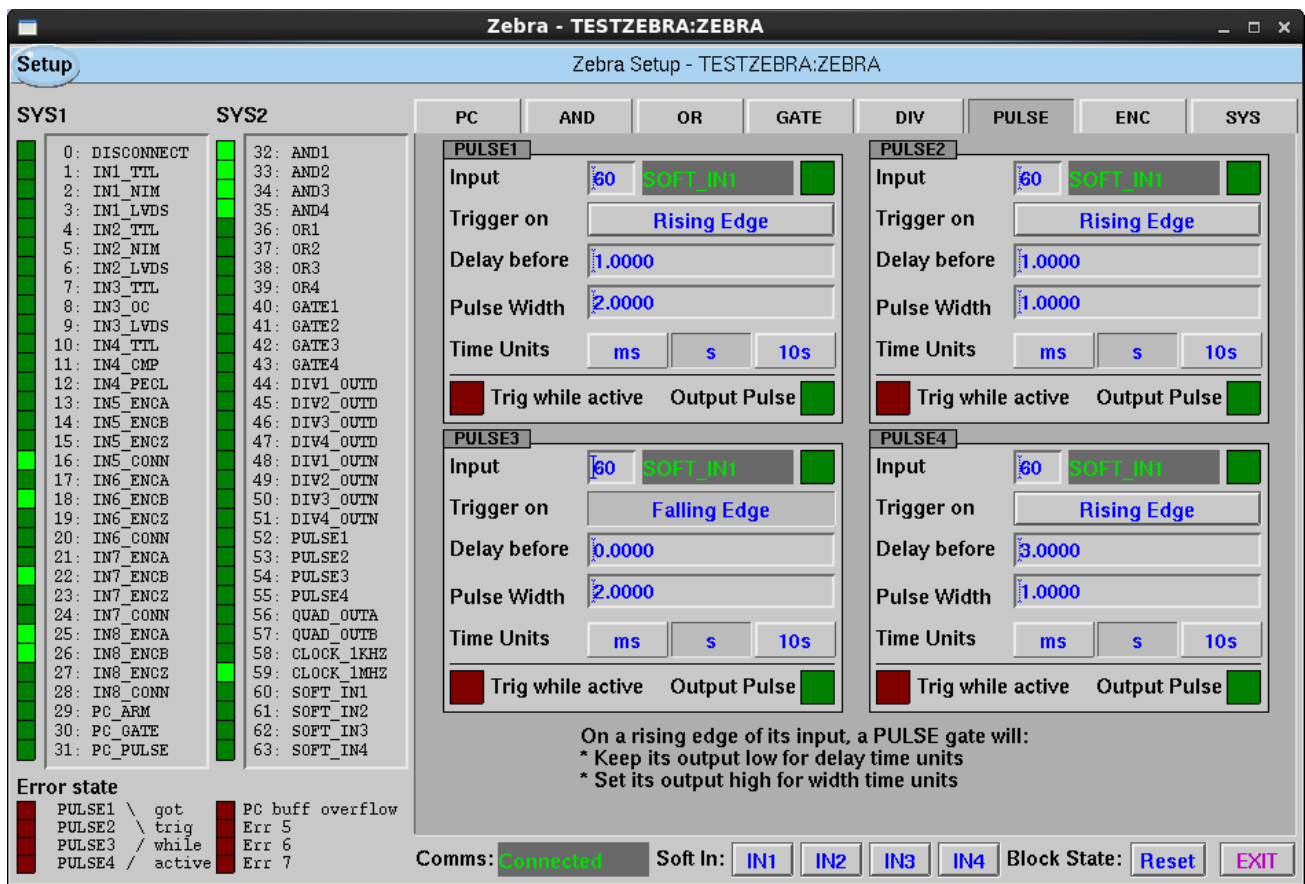


Figure 12: Screenshot of Zebra PULSE blocks

PULSE blocks send out fixed length pulses based on their input. On each rising (or optionally falling) edge, the pulse block keeps its output low for delay time units, then produces a high pulse of width time units. If it gets another edge while it is in a delay or pulse producing state it will flag an error in the bits at the bottom left of the screen. To reset this error, use the Reset button. Delay and pulse width can be in the range 0 to 6.5535 time units.

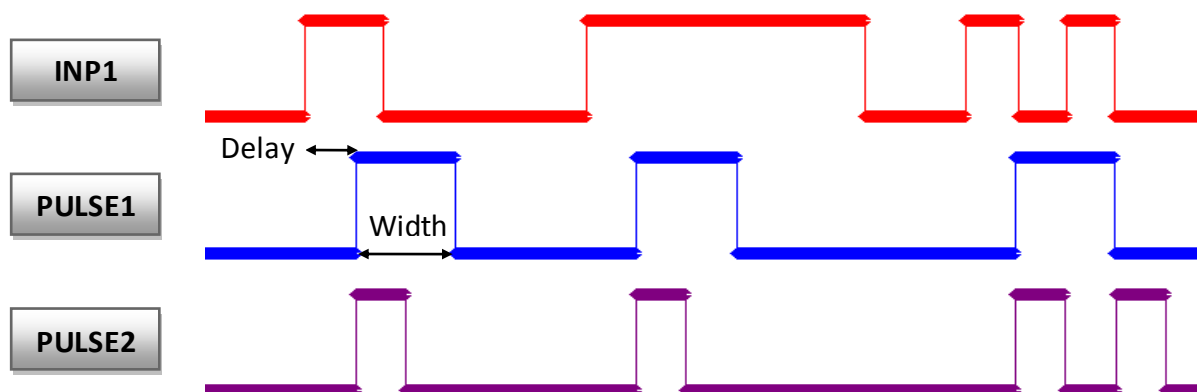


Figure 13: PULSE block example (PULSE1 and PULSE2 setup as shown in Figure 12)

#### 4.8 QUAD Block and Position Counters

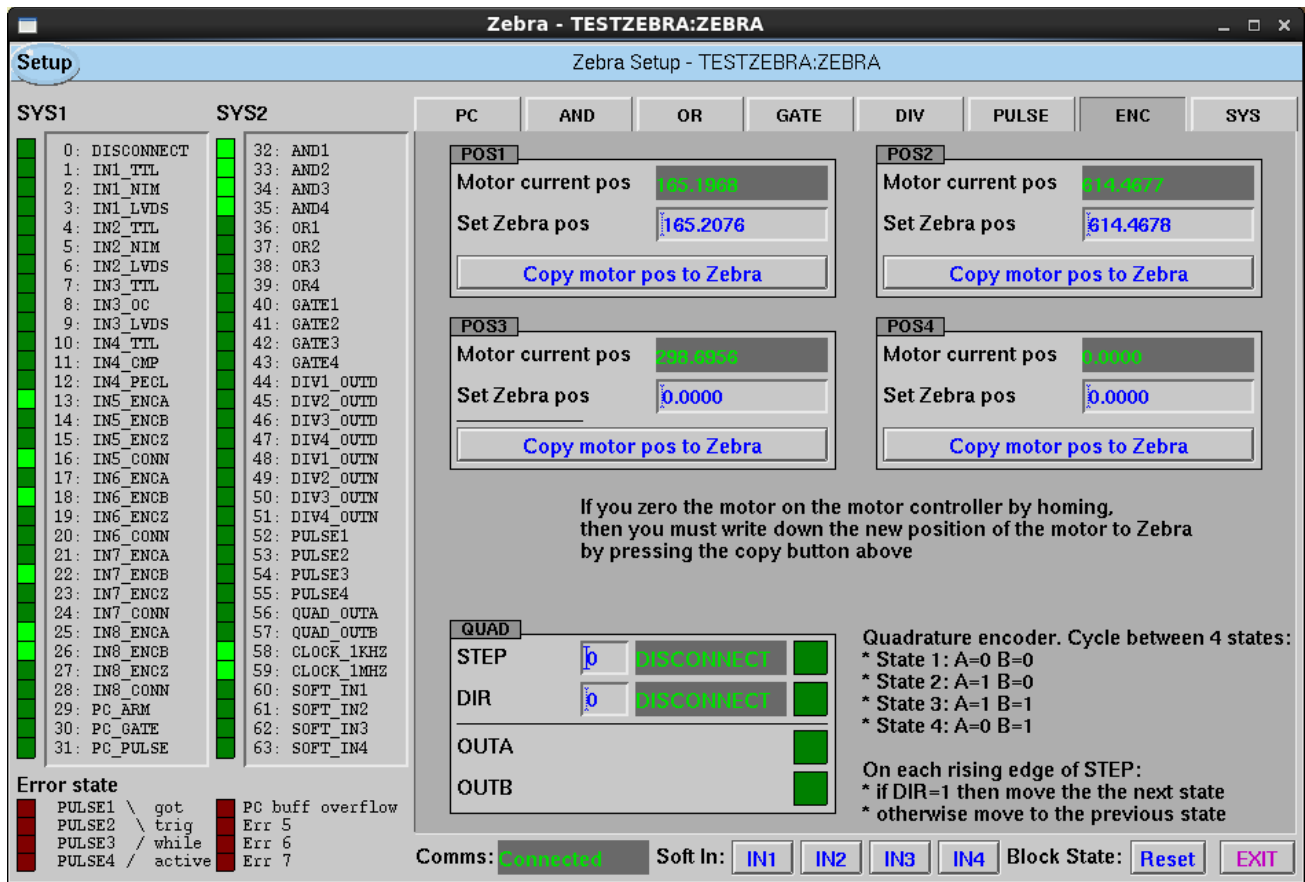


Figure 14: Screenshot of Zebra QUAD block

The QUAD block take a direction bit, and a pulse stream, and quadrature encodes them on its OUTA and OUTB outputs. In the example below, DIR is set to 0 so it counts backwards through the states.

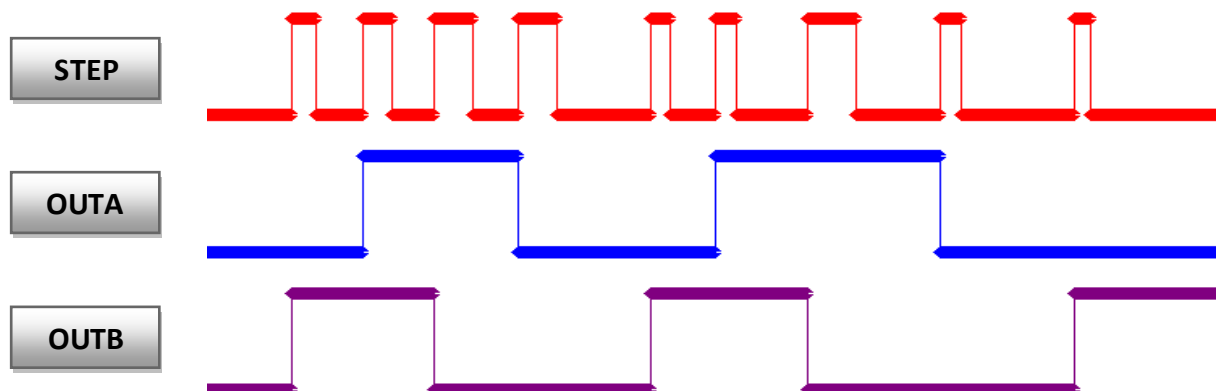


Figure 15: PULSE block example (QUAD setup as shown in Figure 14)

Also on this screen you can set the position counters that are used for position capture. After Zebra is power cycled or the motors are homed, the relevant "Copy" buttons should be pressed to copy the motor positions to Zebra.

## 4.9 PC Position Capture Block

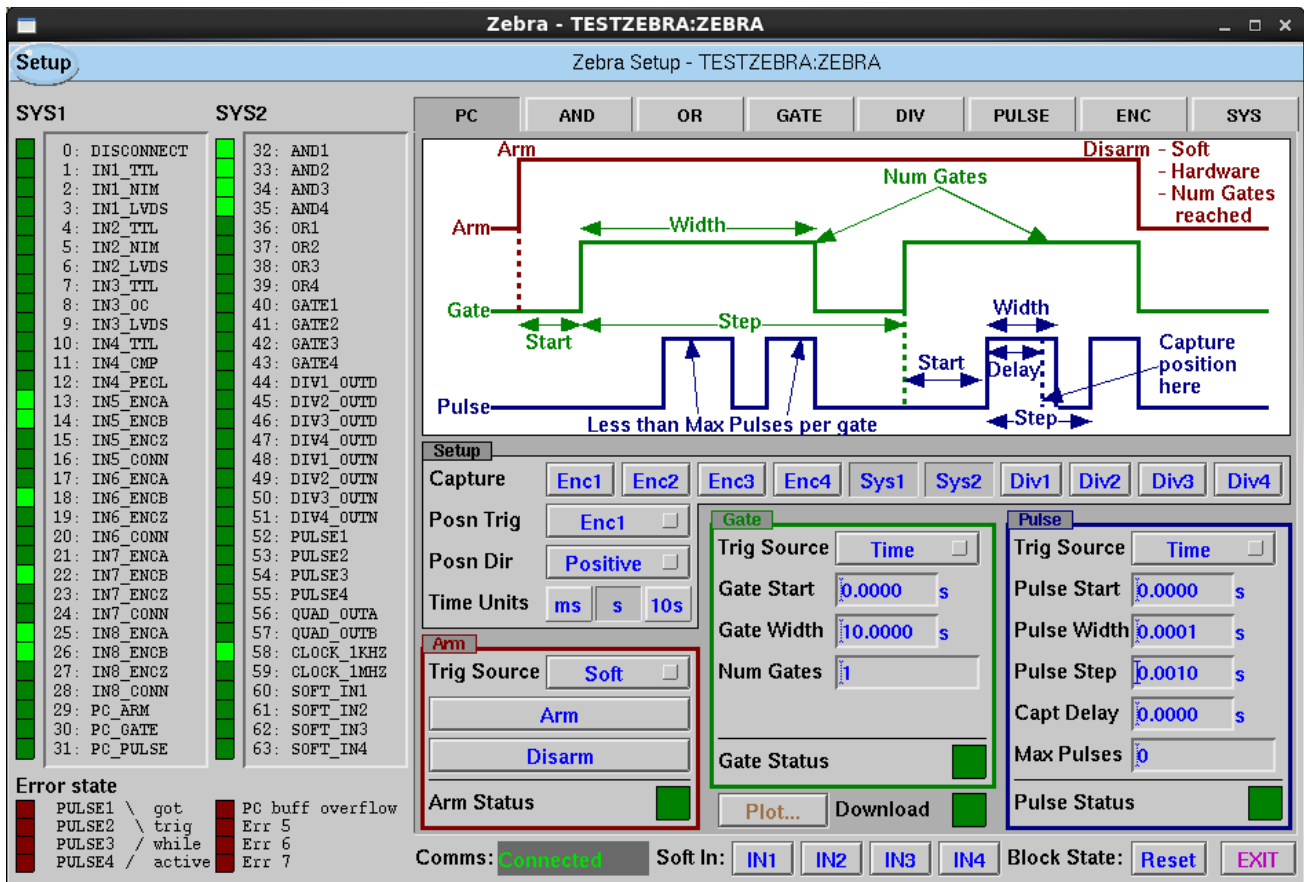


Figure 16: Screenshot of Zebra Position Capture block

The PC Block produces pulses based on time, position, or external triggers, and can capture time, position, the state of the system bus and the pulse divider counters. This is a very flexible block that can be used in a number of ways, described from sections 4.9.4 to 4.9.3.

The main window contains a diagram that describes the function of the Arm, Gate and Pulse signals, boxes to setup each of these signals and other settings that must be set before arm:

- Setup Box (Black):
  - **Capture**: Selects which elements will be captured each time a pulse is output. Enc 1-4 are the encoder inputs on the back panel, Sys 1 and 2 are the upper and lower halves of the system bus, Div 1-4 are the internal counters of the DIV blocks.
  - **Posn Trigger**: If Gate or Pulse **Trig Source** are set to Position, which encoder should be used to compare against.
  - **Posn Dir**: If Gate or Pulse **Trig Source** are set to Position, which direction should the compare act in. E.g. should the Gate width be added or subtracted from the gate start to produce the end position.
  - **Time Units**: If Gate or Pulse **Trig Source** are set to Time, this setting is the unit of time.

- Arm Box (Red):
  - Trig Source:
    - Soft - arm on the click of a button.
    - External - arm on the rising edge of a signal on the System Bus.
  - Arm Button: If Arm Trig Source is Soft, then this button will show, and can be clicked to arm the PC block.
  - External Source: If Arm Trig Source is External, then this is the signal on the system bus that will arm the block.
  - Disarm Button: In either mode, this allows a manual disarm. The PC block will disarm itself at the end of the last gate signal (if Num Gates != 0).
  
- Gate Box (Green):
  - Trig Source:
    - Position - define a position window in motor EGUs.
    - Time - define a time window in Time Units.
    - External - use an external signal to define each Gate.
  - Gate Start:
    - If Gate Trig Source = Position, then the first gate signal will go high when the Posn Trigger encoder first crosses this position.
    - If Gate Trig Source = Time, then this is a delay in Time Units from arm.
  - Gate Width:
    - If Gate Trig Source = Position, the gate will go low at Gate Start + Gate Width if Posn Dir is positive, or Gate Start – Gate Width if Posn Dir is negative.
    - If Gate Trig Source = Time, the gate signal will be high for this amount of time.
  - External Source: If Gate Trig Source = External, then this is the signal on the system bus that will act as a gate signal.
  - Num Gates: How many gate signals to output. If 0, then generate gate signals forever, otherwise drop the arm signal and stop outputting gates after Num Gates are produced.
  - Gate Step: If Num Gates != 1 and Gate Trig Source != External, then this is the distance between the start of consecutive gate signals, in Time Units or Posn Trigger position units. This will act in the direction of Posn Dir.
  
- Pulse Box (Blue)
  - Trig Source:
    - Position - produce pulses based on encoder position.
    - Time - define pulses in Time Units
    - External - use an external signal to define each pulse.

Each time a pulse is output, the elements specified by Capture are latched and output onto the graph
  - Pulse Delay:
    - If Pulse Trig Source = Position, and Posn Dir is positive, the first pulse in each gate will be at Gate Start + Pulse Delay. If Posn Dir is negative, the first pulse will be at Gate Start – Pulse Delay.
    - If Pulse Trig Source = Time, the first pulse will be Pulse Delay Time Units after the gate goes high.

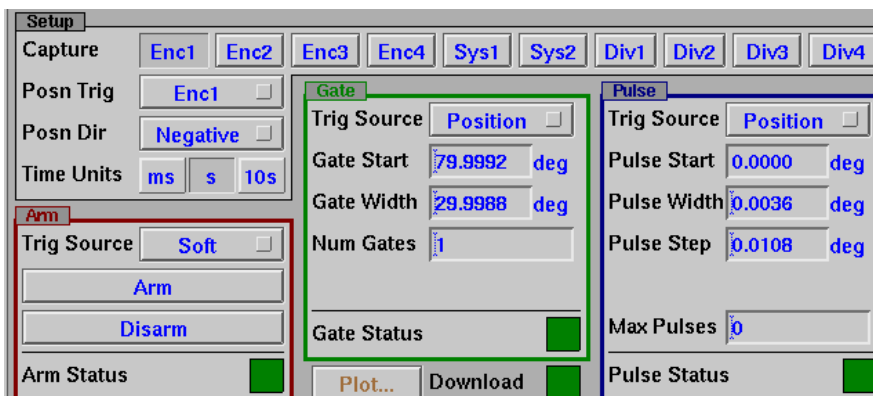


- **Pulse Width:** The width of each pulse in [Posn Trigger](#) position units or [Time Units](#).
- **Pulse Step:** The distance between the rising edge of each pulse, in [Posn Trigger](#) position units or [Time Units](#). Again, this will act in the direction of [Posn Dir](#).
- **External Source:** If [Pulse Trig Source](#) = External, then this is the signal on the system bus that will act as a pulse signal, also latching the elements specified by [Capture](#).
- **Max Pulses:** If non-zero, this is the maximum number of pulses that will be output each gate.

All values are 32-bit signed integers when sent to Zebra, but are translated into either motor engineering units or time units by the EPICS IOC.

At the bottom of the screen, there is a button to bring up the plot window and an indicator to show that data download is in progress. The plot window will be discussed in the examples below. The download indicator lights when the block has armed and is ready to start sending data, and turns off when the block disarms and the last data point has been sent. Note that data download can be interrupted by pressing the Reset button.

## 4.9.1 EXAFS example



**Setup**

Capture: **Enc1** **Enc2** **Enc3** **Enc4** **Sys1** **Sys2** **Div1** **Div2** **Div3** **Div4**

Posn Trig: **Enc1** ☐

Posn Dir: **Negative** ☐

Time Units: **ms** **s** **10s**

**Arm**

Trig Source: **Soft** ☐

**Arm**

**Disarm**

Arm Status: ☒

**Gate**

Trig Source: **Position** ☐

Gate Start: **79.9992** deg

Gate Width: **29.9988** deg

Num Gates: **1**

Gate Status: ☒

**Pulse**

Trig Source: **Position** ☐

Pulse Start: **0.0000** deg

Pulse Width: **0.0036** deg

Pulse Step: **0.0108** deg

Max Pulses: **0**

Pulse Status: ☒

**Plot...** **Download** ☒

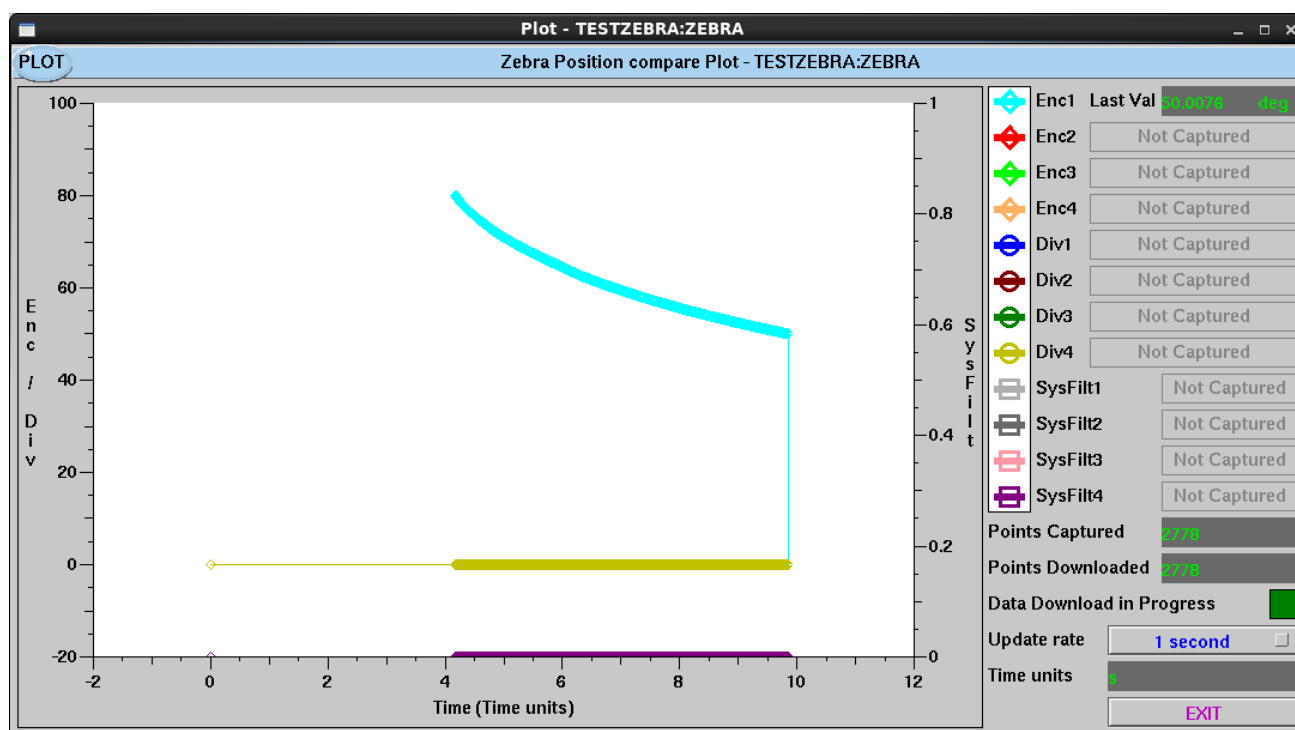


Figure 17: Screenshot of Zebra Position Capture EXAFS example

This example illustrates an EXAFS experiment where the Bragg angle of a monochromator is scanned linearly in energy and a scalar card measuring the transmission and initial photon counts is triggered on position change. The gate was setup to count from 79.9992 degrees down to 50.0003 degrees (roughly 2keV up to 3keV) to illustrate the non-linearity of the motion.

Zebra produces the first pulse at 79.9992 degrees, then at 79.9883 degrees, and so on until it reaches 50.0003 degrees. It captures the time that each pulse was produced and plots it on the graph. Note that if the motor jitters back over a threshold that it has already produced a pulse for, it will not produce another pulse for that threshold. The timing information and motor positions can then be used with the data from the scalar cards to produce the absorption edge of the sample.

## 4.9.2 Tomography example

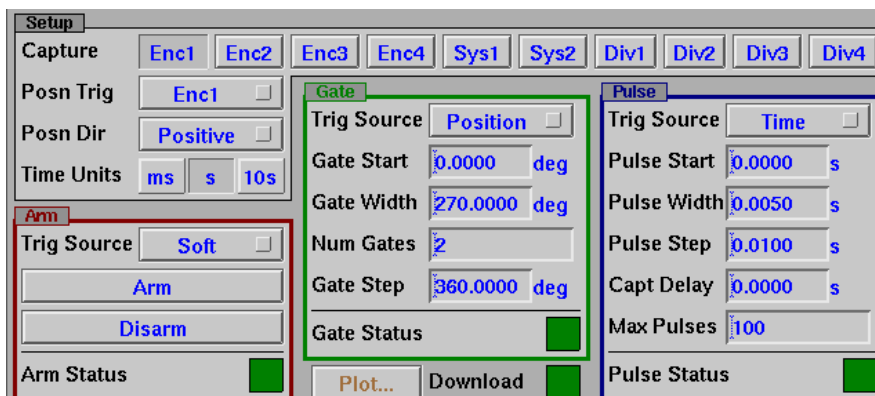
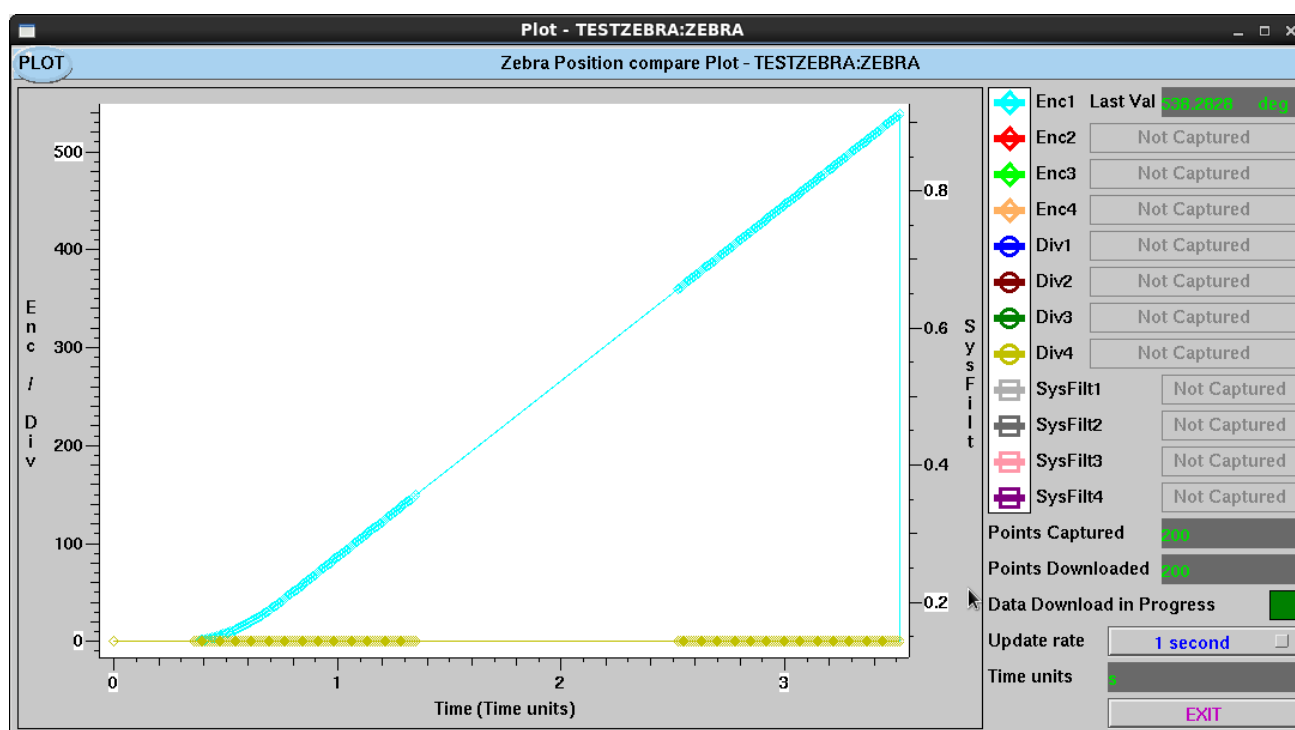



Figure 18: Screenshot of Zebra Position Capture tomography example

This example illustrates a tomography experiment in which the sample is rotated in the beam, and a series of exposures is taken on a detector which can be used to reconstruct the structure of the sample. Multiple rotations (2 in this example) are used to increase accuracy, but only the first 180 degrees of each rotation is captured to allow the detector readout time. The motor is a rotation stage capable of 180 degrees a second. The detector is set to expose so it is capable of 100Hz acquisition. As the detector has a fixed exposure time, Zebra needs to trigger it at exactly 100Hz. On a software arm, Zebra waits until the motor has reached position 0, raises the gate signal, then produces up to 100 pulses at 100Hz with 5ms width in the window up to 270 degrees when it drops the gate signal. It then waits until 360 degrees when it repeats the above. The positions of the motor when these pulses are output are recorded in the plot window.

## 4.9.3 Logic analyser example

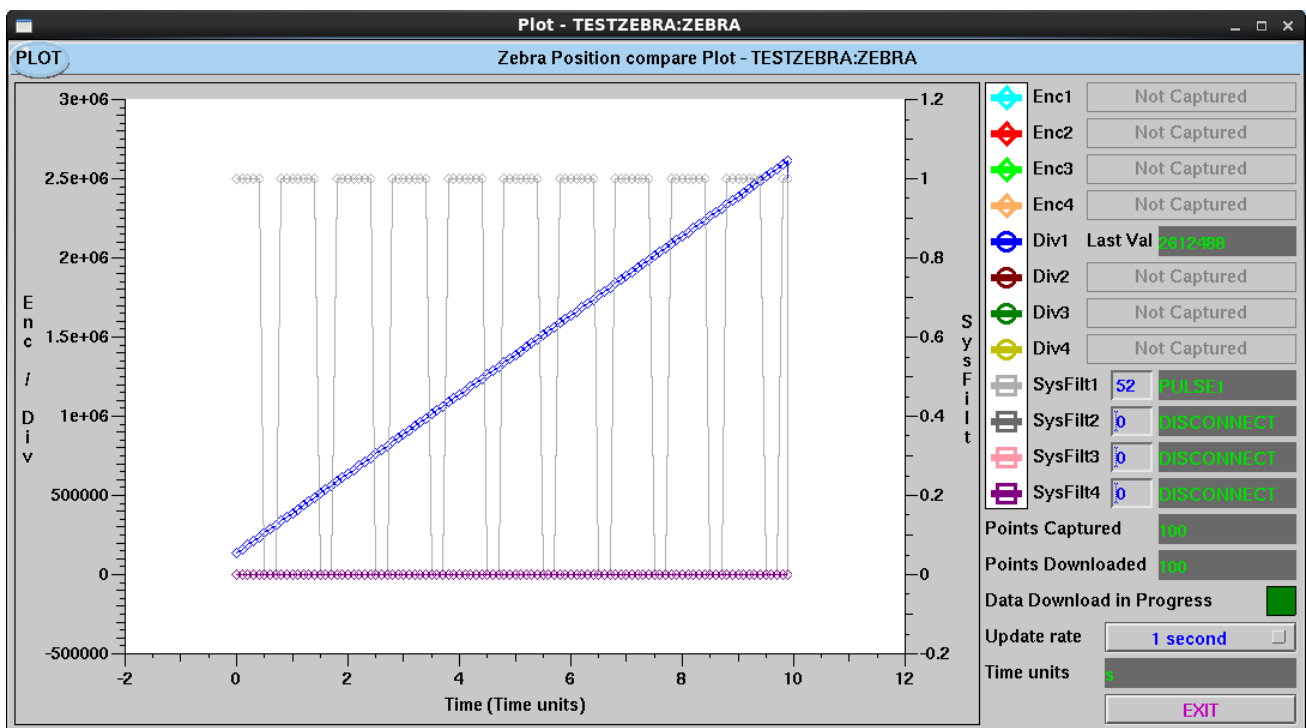
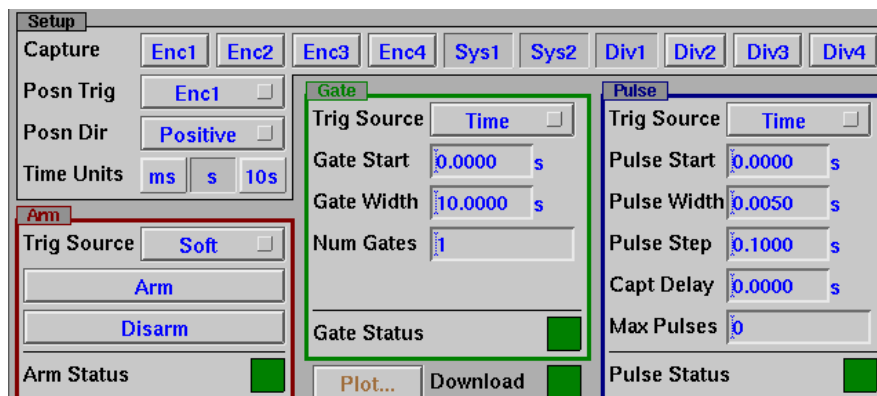


Figure 19: Screenshot of Zebra Position Capture logic analyser example

This example shows how to use the position capture functionality to analyse the signals that come into Zebra. In this example we have a 250kHz signal coming into the DIV1 pulse divider and the PULSE1 module is being repeatedly triggered at 1Hz with a 70% duty cycle. We setup Zebra to capture the entire system bus (SYS1 and SYS2) as well as DIV1 counter value. We then set a fixed 10 second acquisition with pulses every 100ms. The plot shows that the counter value has been captured in blue here, and there are 2475000 pulses difference between the first and last value of DIV1. We can also look at any element on the system bus. If we set SysFilt1 to 52 (PULSE1) we can see plotted in grey that there are 7 point at logic 1, and 3 points at logic 0. With a little effort this means that Zebra can be used as an impromptu scaler card, or can capture the signal from a voltage to frequency converter as part of an acquisition. It is also useful for debugging the wiring of blocks.

## 4.9.4 External trigger example

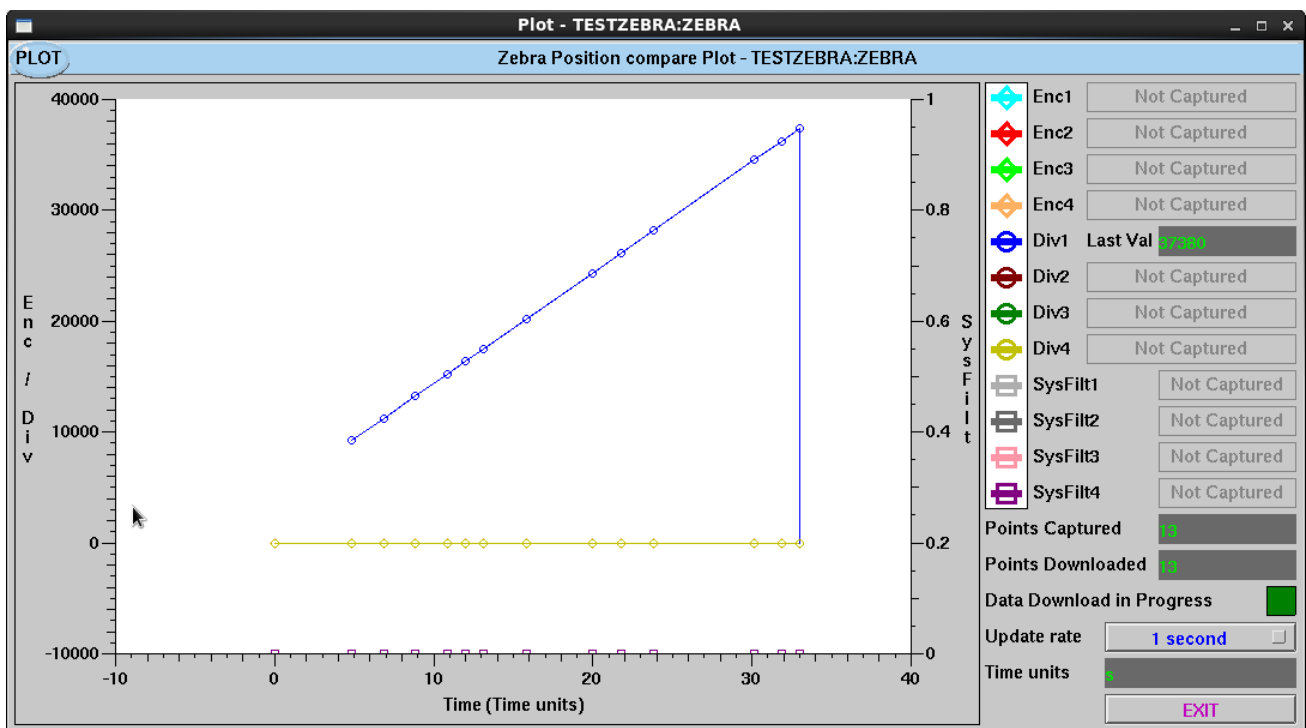
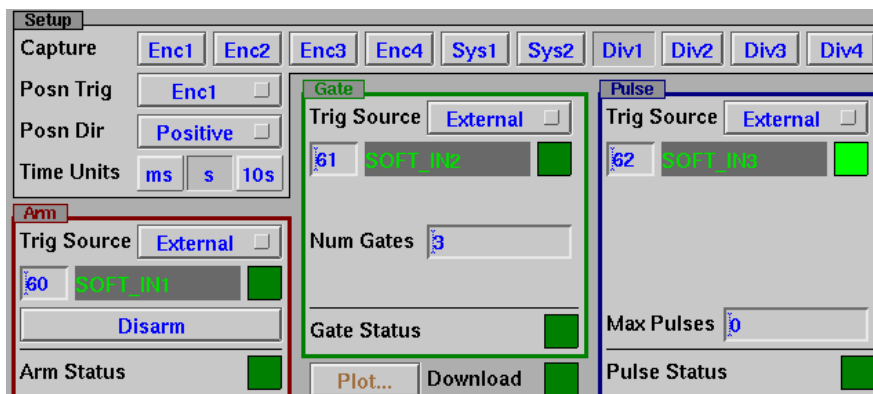


Figure 20: Screenshot of Zebra Position Capture tomography example

This example shows how to trigger Zebra from an external source. The Arm, Gate and Pulse signals can all be taken from any element on the system bus. This allows Zebra to be triggered by a detector for instance, or for multiple Zebras to be chained together to capture the positions of more than 4 encoders. In this example the Soft In buttons have been used to simulate some signals, but this is not likely to be a commonly used configuration. It is important to note that the arm signal is edge triggered, so a manual disarm is still available, and dropping the arm signal will not end acquisition. It is also important to note that when using external signals, each pulse can capture positions even outside of a gate signal. The gate signal is only used to count the number of gates input so that Zebra can disarm itself at the appropriate time.

## 5. ELECTRICAL CONNECTIONS

This section details the electrical specifications of the front and rear panel inputs and outputs.

### 5.1 Front Panel inputs

The front panel inputs are a collection of single ended and differential inputs on BNC and LEMO connections. The diagrams below describe the input circuitry that converts each signal level to the FPGA's LVTTTL level.

#### 5.1.1 TTL Inputs

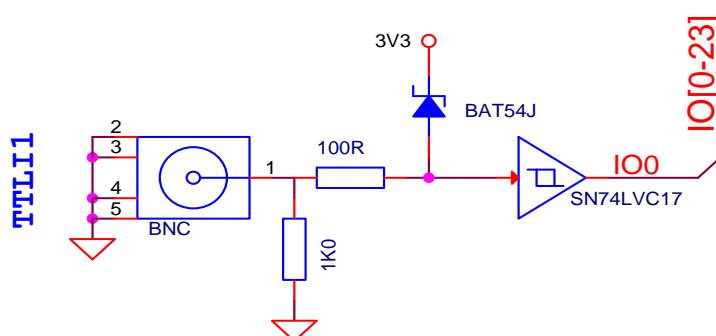


Figure 21: TTL input schematic

TTL Inputs are able to take TTL, LVTTTL, LVCMOS and CMOS signals with amplitude of 2.3V - 5.5V. The channels are terminated with a 1kΩ resistor. If a 50Ω TTL output is used, some ringing may occur and an external 50Ω coax terminator may be used (like <http://accessories.picotech.com/attenuators.html#TA051>), however tests have shown this is generally not needed.

#### 5.1.2 NIM Inputs

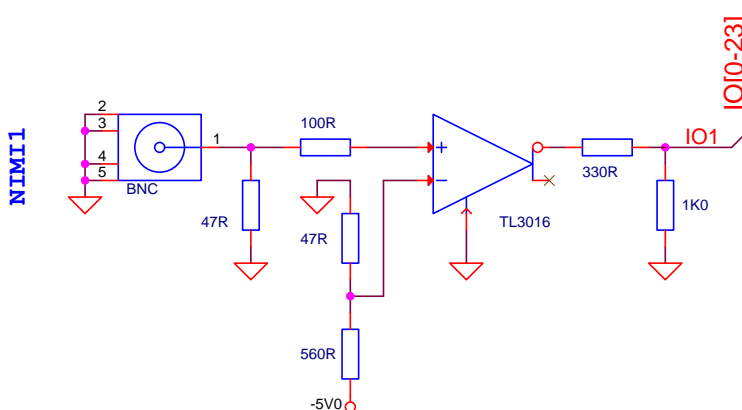


Figure 22: NIM input schematic

NIM input logic levels are defined by levels of current into a 50Ω terminating resistor. Logic "0" is 0mA, and logic "1" is a -16mA current, giving a -0.8V signal.

### 5.1.3 LVDS Inputs

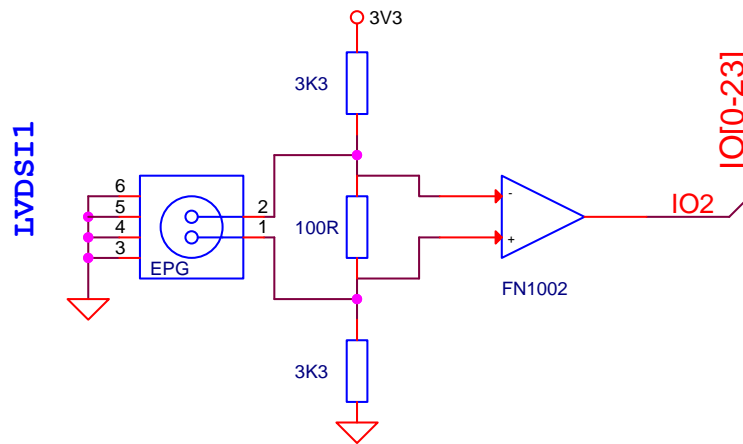


Figure 23: LVDS input schematic

LVDS inputs are able to take and LVDS signal with a differential voltage of magnitude 100mV to 3.3V. The channels are terminated with a 100Ω resistor.

### 5.1.4 Open Collector (OC) Input

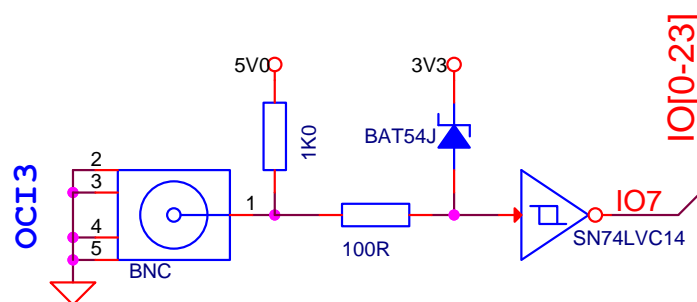


Figure 24: Open Collector input schematic

The Open Collector input does not take a specific voltage or current. Instead, it converts open circuit to logic “0”, and a connection to ground as logic “1”. It does this by pulling the input to 5V with a 1kΩ terminating resistor, and taking the output through an inverting level converter so that the 5V signal produced when the input is open circuit corresponds the a logic “0” in the FPGA.

### 5.1.5 Comparator (CMP) Input

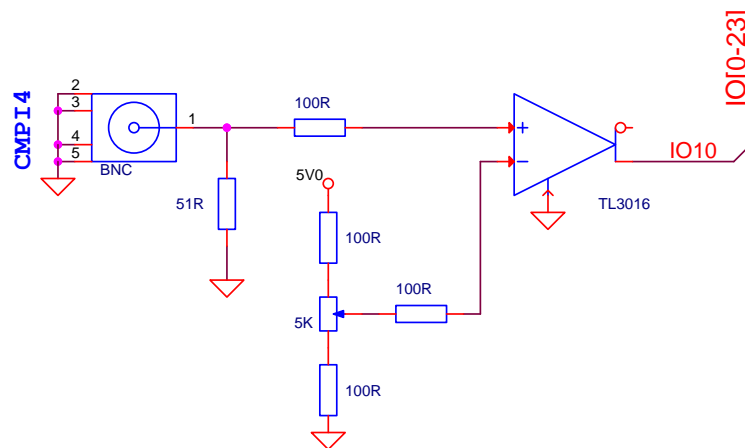


Figure 25: Comparator input schematic

The comparator input takes 0.5V - 4.5V signals, comparing them to a reference voltage level produced by a potentiometer (R12, 5k $\Omega$ ). The channel is terminated with a 100 $\Omega$  resistor, and by default the reference level is 1.5V.

### 5.1.6 PECL Input

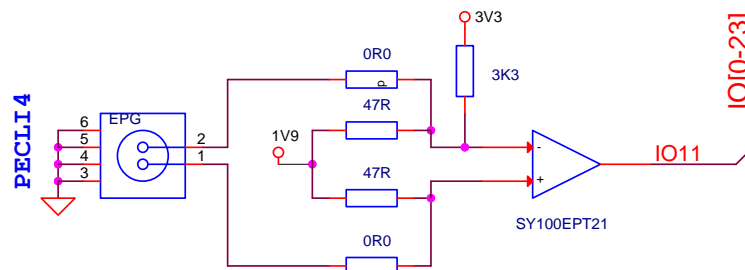


Figure 26: PECL input schematic

The PECL input is able to take LVPECL signals with a differential voltage of magnitude 0.6V - 0.8V. Zebra comes supplied with 0 $\Omega$  resistors R35 and R43 fitted, which puts the input in DC Coupling mode. To use AC coupling mode and to make it suitable for both PECL and LVPECL inputs, replace these resistors with 100nF capacitors.

## 5.2 Front Panel Outputs

The front panel outputs are a collection of single ended and differential outputs on BNC and LEMO connections. The diagrams below describe the output circuitry that converts the FPGA's LVTTTL level to the relevant output signal level.



### 5.2.1 TTL Output

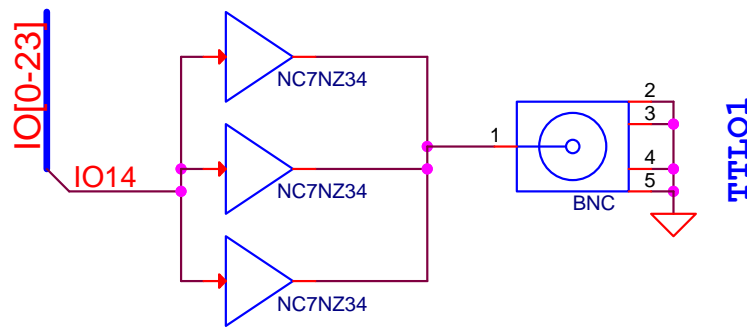


Figure 27: TTL output schematic

TTL outputs convert low power FPGA signals into LVTTTL signals, which are also compatible with TTL and LVCMOS levels, and are able to drive a 50Ω external load.

### 5.2.2 NIM Output

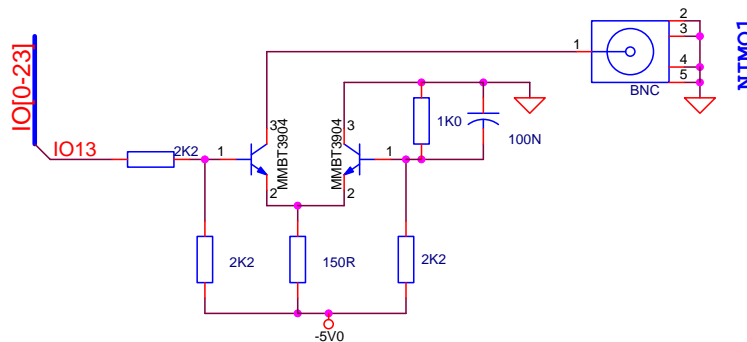


Figure 28: NIM output schematic

NIM outputs convert the FPGA signals into 0mA (logic “0”) and -16mA (logic “1”) current signal which are designed to drive a 50Ω external load.

### 5.2.3 LVDS Output

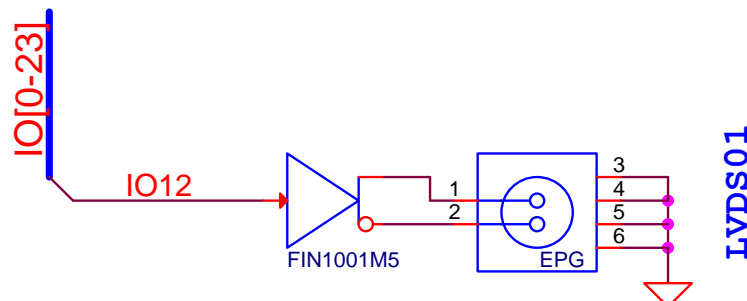


Figure 29: LVDS output schematic

LVDS outputs produce a signal with a differential voltage of 350mV. It is designed to drive a 100Ω load.

### 5.2.4 Open Collector (OC) Output

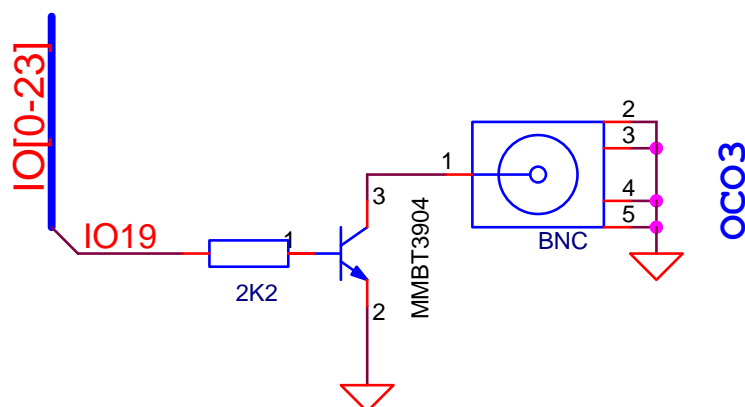


Figure 30: Open Collector output schematic

The open collector output produces an open circuit on logic “0”, and ties the output to ground on logic “1”. It is designed to be connected to a sensing circuit with a suitable pull-up resistor.

### 5.2.5 PECL Output

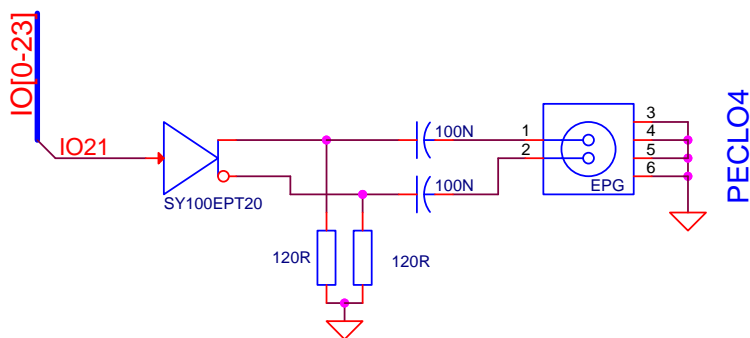


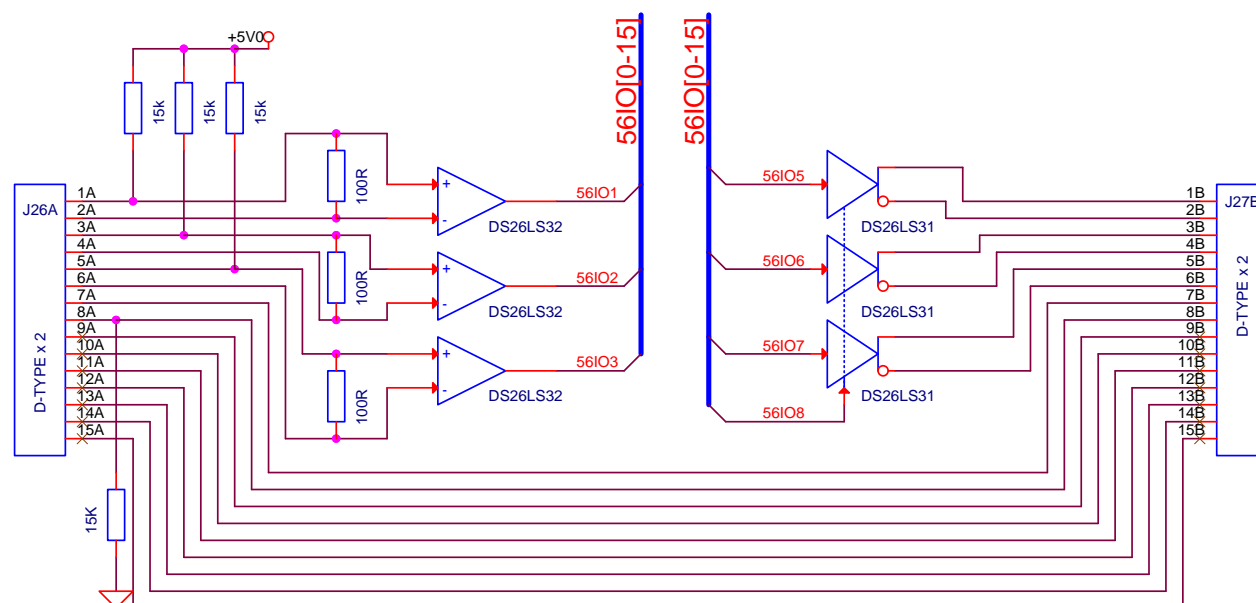
Figure 31: PECL output schematic

The PECL output is differential (with a swing of approximately  $\pm 800\text{mV}$ ) and AC coupled. Any receiver connected to this output should internally bias the line to its required level (typically  $V_{CC} - 1.2\text{V}$  for PECL and LVPECL) and match the impedance of the connecting cable (typically  $100\Omega$  differential). Note that, due to the AC coupling capacitors, this output will not sustain a DC output.

## 5.3 Rear Panel

Zebra’s rear panel has 4 DB15 connector pairs for encoder inputs and outputs, a power connector with fuse and earth stud, along with RS232 and JTAG connectors. The electrical interface to each of these connections is detailed below.

## 5.3.1 Encoder I/O ports



Input (Female socket, Top)		Connection	Output (Male socket, Bottom)	
Function	Pin		Pin	Function
RXA+	1A	To/from FPGA	1B	TXA+
RXA-	2A	To/from FPGA	2B	TXA-
RXB+	3A	To/from FPGA	3B	TXB+
RXB-	4A	To/from FPGA	4B	TXB-
RXZ+	5A	To/from FPGA	5B	TXZ+
RXZ-	6A	To/from FPGA	6B	TXZ-
EXT PWR	7A	Direct connection	7B	EXT PWR
EXT GND	8A	Direct connection	8B	EXT GND
Not defined	9A-15A	Direct connection	9B-15B	Not defined

Figure 32: Encoder input/output schematic

The encoder inputs/outputs support RS422 quadrature encoders. These encoders have 3 differential channels: channels A (pins 1 and 2) and B (pins 3 and 4) provide a quadrature encoded pulse stream which can be decoded into the position of the motor, and channel Z (pins 5 and 6) is typically an index pulse to give homing information. Each channel is 100Ω terminated, and both halves are pulled up to 5V via a 15kΩ resistor. These signals are converted to LVTTTL and passed to the FPGA, and the FPGA outputs converted back to RS422 for output. Pin 7 is designated for encoder power, but is taken straight through Zebra so any low voltage power can be used. Pin 8 is pulled to ground via a 15kΩ resistor. Pins 9-15 are passed straight through and can be used for any purpose.

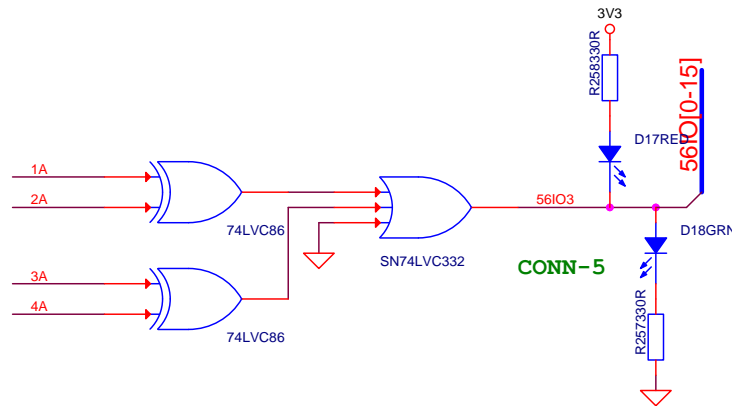


Figure 33: Encoder loss detection circuit

The function of the encoder loss detection circuit is to test for an open circuit condition on channel A or B. If the cable is disconnected, both halves of the differential input will float to 5V because of the pull-up resistors shown in Figure 32. Zebra uses an XOR gate for each channel to detect this, and the logical OR of these signals is passed to the FPGA. This is also used to drive a red LED (to indicate encoder loss) or a green LED (to indicate encoder present).

It should be noted that some encoders disconnect their outputs to indicate a problem, so a red LED could mean a disconnected cable or an encoder in error state.

### 5.3.2 RS232 port

This port is used for host communication with the FPGA. A dual channel RS-232 line transceiver (MAX3232) is used for this with the following pin-out on the DB-9 connector:

Function	PIN
DCD	1
RXD	2
TXD	3
DTR	4
GND	5
DSR	6
RTS	7
CTS	8
RI	9

This communication line uses 115200 baud rate, 8 data bits, no parity, and 1 stop bit. A null modem lead should be used to connect Zebra to a host PC via a USB-RS232 adapter. For more information on the protocol, see section 7.2.

### 5.3.3 JTAG

This should only be used to reprogram the FPGA.

### 5.3.4 Power

Mains power (100-240V AC) is supplied via an IEC 60320 C14 socket on the back (using a standard “kettle lead”). The state of the internal +5V, +3.3V, +1.2V and -5V lines is indicated with status LEDs on the front, and the power switch LED will light when mains is applied.

Zebra is protected with a 0.5A fuse accessed with a screw in connector on the back panel.

An earthing stud is provided to tie the chassis to ground and reduce noise. This is internally connected to the earth of the mains connector. In operation Zebra should be earthed either through the mains lead or through the earthing stud.

General power consumption of Zebra is 6-7W. There are ventilation holes in the top panel to allow for heat dissipation, but forced ventilation is not required.

## 6. FPGA DETAILS

Zebra hardware includes a Xilinx Spartan-3 FPGA (part number XC6SLX9-2TQG144) and surrounding digital components all interfacing directly to FPGA.

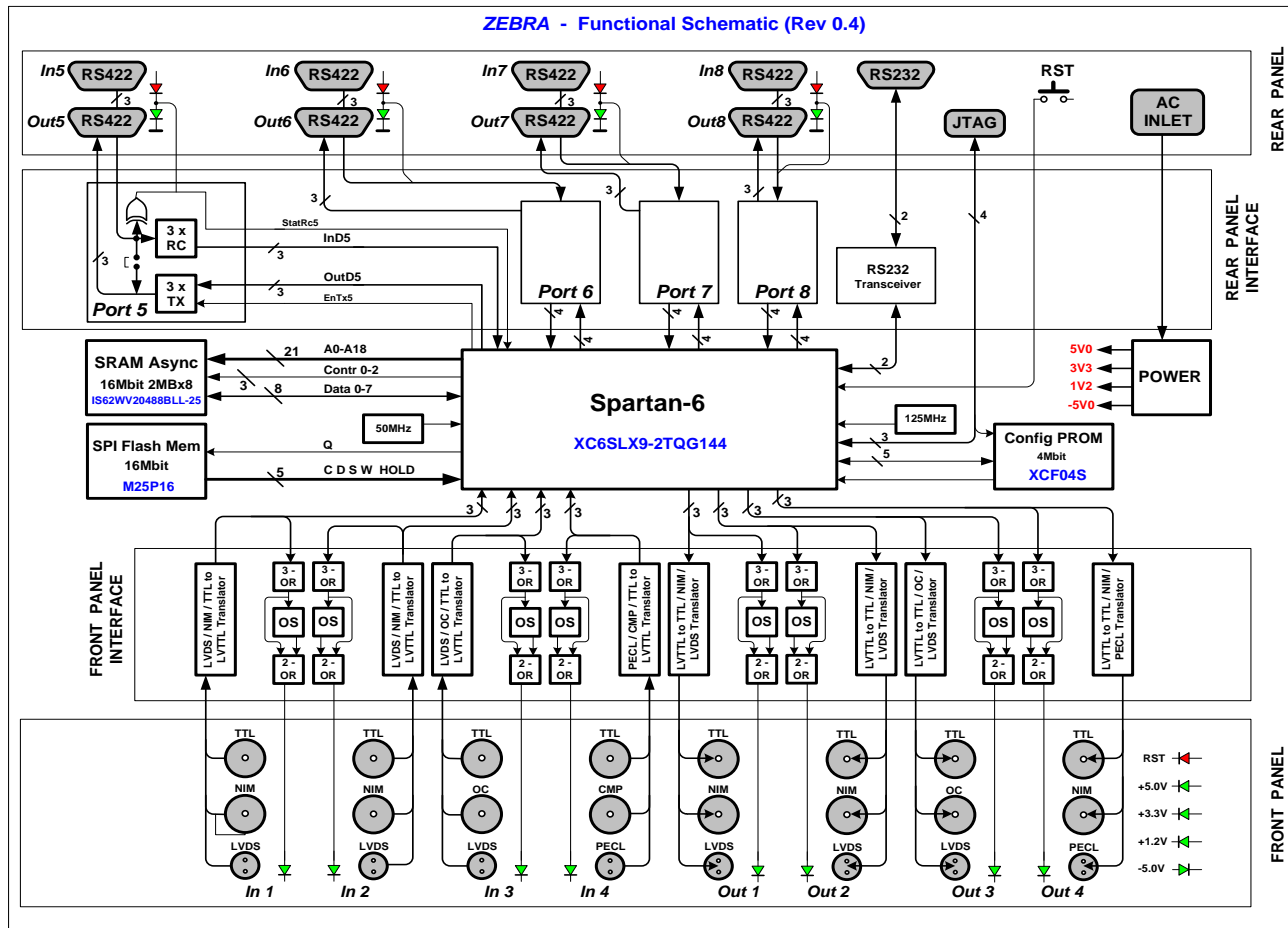


Figure 34: Zebra functional schematic

The design is divided into 4 main functional blocks, and all of these modules have external interfaces to specific IO channels. The details of FPGA firmware design modules and their implementation are described in *TDI-CTRL-TNO-033 Zebra FPGA Firmware Documentation*.

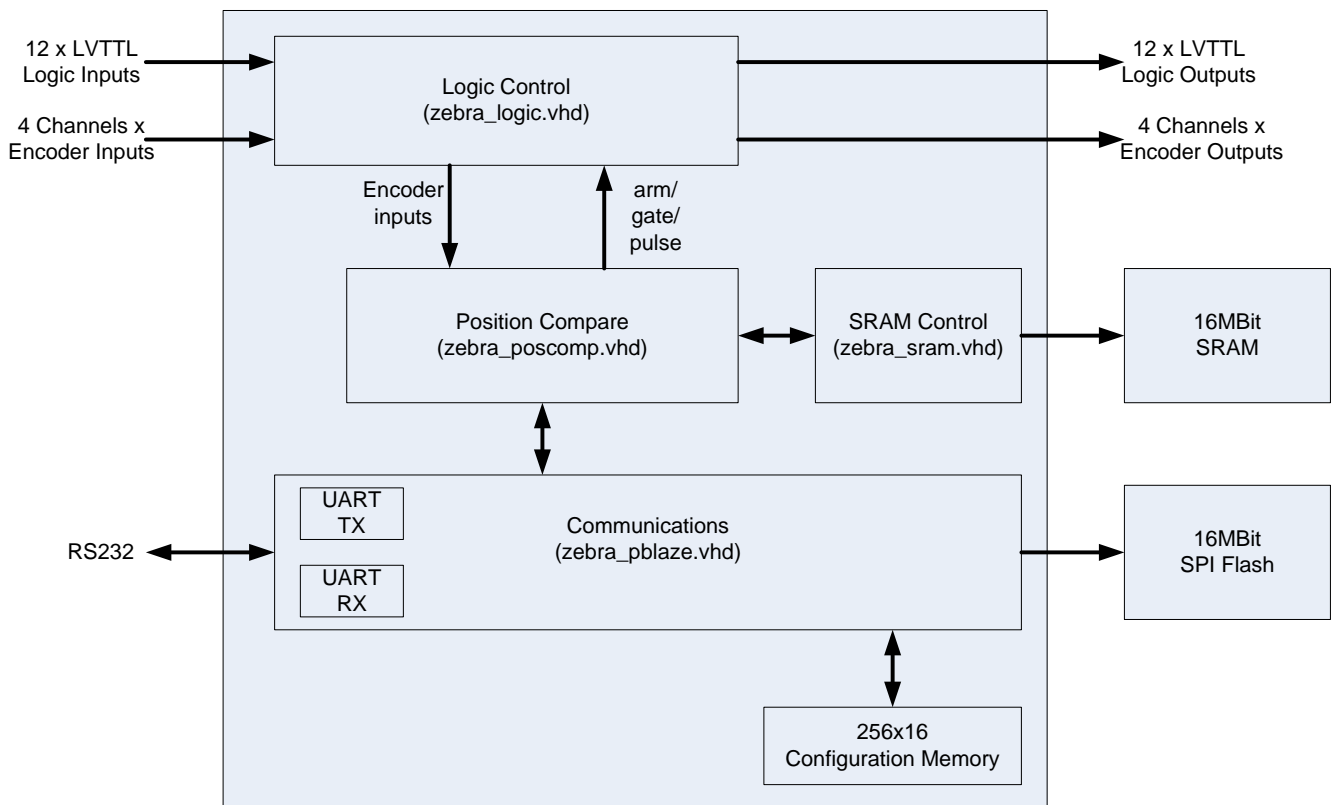


Figure 35: Zebra FPGA firmware top-level block diagram

## 6.1 Clocks

The FPGA design operates on two asynchronous clock domains which are 50 MHz system clock, and 40MHz SRAM clock. System clock is provided using an external 50MHz oscillator through and 40MHz SRAM clock is generated by an internal PLL. J35 is used to select this 50 MHz clock and should be in place.

## 6.2 Position Capture

The aim of this Position Capture is to output pulses and store encoder values either at set positions, on an internal clock, or an external trigger. The following diagram shows a typical Position Capture output.

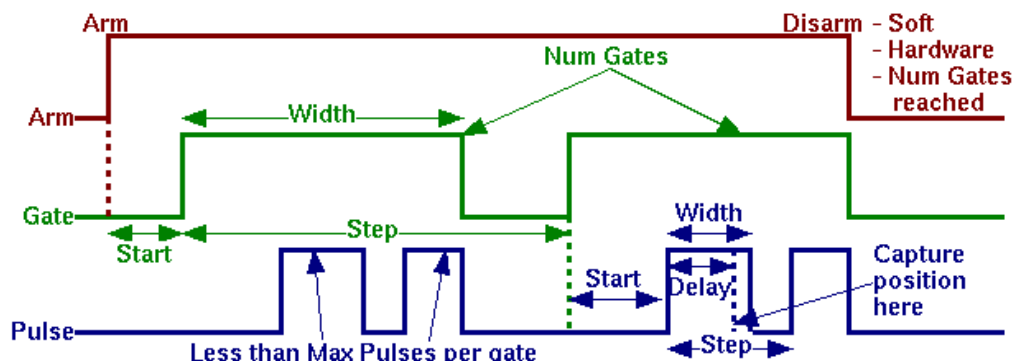


Figure 36: Position Capture timing diagram.

Position Capture module is tightly coupled with the external SRAM module (IS62WV20488BLL) which is a high-speed, low power, 2M-word by 8-bit CMOS static RAM. This memory system is connected to the FPGA and used in position-capture mode in order to store timestamp, encoder, divider, and system bus outputs. The maximum number of pulse output samples that can be stored depends on the user selected fields and is calculated as:

$$\text{Number of samples} = 2e6 / (4 * \text{<\# of fields selected>})$$

Where fields correspond to:

Timestamp	Encoder #1	Encoder #2	Encoder #3	Encoder #4	Sys Bus LSB	Sys Bus MSB	Divider #1	Divider #2	Divider #3	Divider #4
-----------	---------------	---------------	---------------	---------------	----------------	----------------	---------------	---------------	---------------	---------------

Each field is a 32-bit value, and can be selected/de-selected for capture by the user using the PC\_BIT\_CAP.

The SRAM takes approx 58 clock cycles to store a set of fields to it, so the minimum pulse step is 0.0012ms. If the prescaler on the block is set to work with seconds or 10 seconds as the time unit, this minimum increases to 0.0003s or 0.003s respectively. If position compare timings below these are used then the number of points downloaded will not match with the number of pulses produced, and Zebra will require a soft reset to recover.

The time taken to decode the RS422 signals is between 9 and 10 FPGA clock cycles, depending on how the input pulse is aligned which takes 180-200ns. Adding the RS422 input conversion delay of 23ns and a TTL output delay of 4ns, it takes ~227ns between an encoder signal pulse arriving at Zebra and it producing a position compare pulse from a TTL output.



## 7. SOFTWARE INTEGRATION

This section provides third-parties with the information for integrating Zebra into their control system.

### 7.1 Register Address Mapping

From the RS232 serial link, addresses 0x00 to 0xFF can be accessed for read and write access. Some of these addresses are meaningful for either read or write only as indicated in the following table.

#### 7.1.1 Zebra Logic

Addr	Name	Register Description	Used Bits	R/W
0x00	AND1_INV	And4 gate-1 invert mask	[ 3 : 0]	R/W
0x01	AND2_INV	And4 gate-2 invert mask	[ 3 : 0]	R/W
0x02	AND3_INV	And4 gate-3 invert mask	[ 3 : 0]	R/W
0x03	AND4_INV	And4 gate-4 invert mask	[ 3 : 0]	R/W
0x04	AND1_ENA	And4 gate-1 input mask	[ 3 : 0]	R/W
0x05	AND2_ENA	And4 gate-2 input mask	[ 3 : 0]	R/W
0x06	AND3_ENA	And4 gate-3 input mask	[ 3 : 0]	R/W
0x07	AND4_ENA	And4 gate-4 input mask	[ 3 : 0]	R/W
0x08	AND1_INP1	And4 gate-1, input 1 select	[ 5 : 0]	R/W
0x09	AND1_INP2	And4 gate-1, input 2 select	[ 5 : 0]	R/W
0x0A	AND1_INP3	And4 gate-1, input 3 select	[ 5 : 0]	R/W
0x0B	AND1_INP4	And4 gate-1, input 4 select	[ 5 : 0]	R/W
0x0C	AND2_INP1	And4 gate-2, input 1 select	[ 5 : 0]	R/W
0x0D	AND2_INP2	And4 gate-2, input 2 select	[ 5 : 0]	R/W
0x0E	AND2_INP3	And4 gate-2, input 3 select	[ 5 : 0]	R/W
0x0F	AND2_INP4	And4 gate-2, input 4 select	[ 5 : 0]	R/W
0x10	AND3_INP1	And4 gate-3, input 1 select	[ 5 : 0]	R/W
0x11	AND3_INP2	And4 gate-3, input 2 select	[ 5 : 0]	R/W
0x12	AND3_INP3	And4 gate-3, input 3 select	[ 5 : 0]	R/W
0x13	AND3_INP4	And4 gate-3, input 4 select	[ 5 : 0]	R/W
0x14	AND4_INP1	And4 gate 4, input 1 select	[ 5 : 0]	R/W
0x15	AND4_INP2	And4 gate 4, input 2 select	[ 5 : 0]	R/W
0x16	AND4_INP3	And4 gate 4, input 3 select	[ 5 : 0]	R/W
0x17	AND4_INP4	And4 gate 4, input 4 select	[ 5 : 0]	R/W
0x18	OR1_INV	Or4 gate-1 invert mask	[ 3 : 0]	R/W
0x19	OR2_INV	Or4 gate-2 invert mask	[ 3 : 0]	R/W
0x1A	OR3_INV	Or4 gate-3 invert mask	[ 3 : 0]	R/W
0x1B	OR4_INV	Or4 gate-4 invert mask	[ 3 : 0]	R/W
0x1C	OR1_ENA	Or4 gate-1 input mask	[ 3 : 0]	R/W
0x1D	OR2_ENA	Or4 gate-2 input mask	[ 3 : 0]	R/W
0x1E	OR3_ENA	Or4 gate-3 input mask	[ 3 : 0]	R/W
0x1F	OR4_ENA	Or4 gate-4 input mask	[ 3 : 0]	R/W

0x20	OR1_INP1	Or4 gate-1, input 1 select	[ 5:0]	R/W
0x21	OR1_INP2	Or4 gate-1, input 2 select	[ 5:0]	R/W
0x22	OR1_INP3	Or4 gate-1, input 3 select	[ 5:0]	R/W
0x23	OR1_INP4	Or4 gate-1, input 4 select	[ 5:0]	R/W
0x24	OR2_INP1	Or4 gate-2, input 1 select	[ 5:0]	R/W
0x25	OR2_INP2	Or4 gate-2, input 2 select	[ 5:0]	R/W
0x26	OR2_INP3	Or4 gate-2, input 3 select	[ 5:0]	R/W
0x27	OR2_INP4	Or4 gate-2, input 4 select	[ 5:0]	R/W
0x28	OR3_INP1	Or4 gate-3, input 1 select	[ 5:0]	R/W
0x29	OR3_INP2	Or4 gate-3, input 2 select	[ 5:0]	R/W
0x2A	OR3_INP3	Or4 gate-3, input 3 select	[ 5:0]	R/W
0x2B	OR3_INP4	Or4 gate-3, input 4 select	[ 5:0]	R/W
0x2C	OR4_INP1	Or4 gate 4, input 1 select	[ 5:0]	R/W
0x2D	OR4_INP2	Or4 gate 4, input 2 select	[ 5:0]	R/W
0x2E	OR4_INP3	Or4 gate 4, input 3 select	[ 5:0]	R/W
0x2F	OR4_INP4	Or4 gate 4, input 4 select	[ 5:0]	R/W
0x30	GATE1_INP1	Gate generator 1, Set Input #1 Select	[5:0]	R/W
0x31	GATE2_INP1	Gate generator 2, Set Input #1 Select	[5:0]	R/W
0x32	GATE3_INP1	Gate generator 3, Set Input #1 Select	[5:0]	R/W
0x33	GATE4_INP1	Gate generator 4, Set Input #1 Select	[5:0]	R/W
0x34	GATE1_INP2	Gate generator 1, Rst Input #2 Select	[5:0]	R/W
0x35	GATE2_INP2	Gate generator 2, Rst Input #2 Select	[5:0]	R/W
0x36	GATE3_INP2	Gate generator 3, Rst Input #2 Select	[5:0]	R/W
0x37	GATE4_INP2	Gate generator 4, Rst Input #2 Select	[5:0]	R/W
0x38	DIV1_DIVLO	Pulse Divider 1, Divisor Low Word	[15:0]	R/W
0x39	DIV1_DIVHI	Pulse Divider 2, Divisor High Word	[15:0]	R/W
0x3A	DIV2_DIVLO	Pulse Divider 2, Divisor Low Word	[15:0]	R/W
0x3B	DIV2_DIVHI	Pulse Divider 2, Divisor High Word	[15:0]	R/W
0x3C	DIV3_DIVLO	Pulse Divider 1, Divisor Low Word	[15:0]	R/W
0x3D	DIV3_DIVHI	Pulse Divider 2, Divisor High Word	[15:0]	R/W
0x3E	DIV4_DIVLO	Pulse Divider 2, Divisor Low Word	[15:0]	R/W
0x3F	DIV4_DIVHI	Pulse Divider 2, Divisor High Word	[15:0]	R/W
0x40	DIV1_INP	Pulse Divider 1, Input Select	[5:0]	R/W
0x41	DIV2_INP	Pulse Divider 2, Input Select	[5:0]	R/W
0x42	DIV3_INP	Pulse Divider 3, Input Select	[5:0]	R/W
0x43	DIV4_INP	Pulse Divider 4, Input Select	[5:0]	R/W
0x44	PULSE1_DLY	Pulse generator 1, Delay	[15:0]	R/W
0x45	PULSE2_DLY	Pulse generator 2, Delay	[15:0]	R/W
0x46	PULSE3_DLY	Pulse generator 3, Delay	[15:0]	R/W
0x47	PULSE4_DLY	Pulse generator 4, Delay	[15:0]	R/W
0x48	PULSE1_WID	Pulse generator 1, Pulse Width	[15:0]	R/W
0x49	PULSE2_WID	Pulse generator 2, Pulse Width	[15:0]	R/W

0x4A	PULSE3_WID	Pulse generator 3, Pulse Width	[15:0]	R/W
0x4B	PULSE4_WID	Pulse generator 4, Pulse Width	[15:0]	R/W
0x4C	PULSE1_PRE	Pulse generator 1, Prescaler (time unit select)	[15:0]	R/W
0x4D	PULSE2_PRE	Pulse generator 2, Prescaler (time unit select)	[15:0]	R/W
0x4E	PULSE3_PRE	Pulse generator 3, Prescaler (time unit select)	[15:0]	R/W
0x4F	PULSE4_PRE	Pulse generator 4, Prescaler (time unit select)	[15:0]	R/W
0x50	PULSE1_INP	Pulse generator 1, Input Select	[5:0]	R/W
0x51	PULSE2_INP	Pulse generator 2, Input Select	[5:0]	R/W
0x52	PULSE3_INP	Pulse generator 3, Input Select	[5:0]	R/W
0x53	PULSE4_INP	Pulse generator 4, Input Select	[5:0]	R/W
0x54	POLARITY	Functional blocks polarity select (0: Rising edge, 1: Falling edge bits[3 : 0] : Gate generator N input-1 polarity bits[7 : 4] : Gate generator N input-2 polarity bits[11: 8] : Pulse divider N input polarity bits[15:12] : Pulse generator N input polarity	[15:0]	R/W
0x55	QUAD_DIR	Quadrature Encoder Direction input select	[5:0]	R/W
0x56	QUAD_STEP	Quadrature Encoder Step input select	[5:0]	R/W
0x57	PC_ARM_INP	External arm input select	[5:0]	R/W
0x58	PC_GATE_INP	External gate input select	[5:0]	R/W
0x59	PC_PULSE_INP	External pulse input select	[5:0]	R/W
0x60	OUT1_TTL	Output multiplexer select for Channel #1 TTL Output	[5:0]	R/W
0x61	OUT1_NIM	Output multiplexer select for Channel #1 NIM Output	[5:0]	R/W
0x62	OUT1_LVDS	Output multiplexer select for Channel #1 LVDS Output	[5:0]	R/W
0x63	OUT2_TTL	Output multiplexer select for Channel #2 TTL Output	[5:0]	R/W
0x64	OUT2_NIM	Output multiplexer select for Channel #2 NIM Output	[5:0]	R/W
0x65	OUT2_LVDS	Output multiplexer select for Channel #2 LVDS Output	[5:0]	R/W
0x66	OUT3_TTL	Output multiplexer select for Channel #3 TTL Output	[5:0]	R/W
0x67	OUT3_OC	Output multiplexer select for Channel #3 Open Collector Output	[5:0]	R/W
0x68	OUT3_LVDS	Output multiplexer select for Channel #3 LVDS Output	[5:0]	R/W
0x69	OUT4_TTL	Output multiplexer select for Channel #4 TTL Output	[5:0]	R/W
0x6A	OUT4_NIM	Output multiplexer select for Channel #4 NIM Output	[5:0]	R/W
0x6B	OUT4_PECL	Output multiplexer select for Channel #4 PECL Output	[5:0]	R/W
0x6C	OUT5_ENCA	Output multiplexer select for Encoder Channel #1 A Output	[5:0]	R/W
0x6D	OUT5_ENCB	Output multiplexer select for Encoder Channel #1 B Output	[5:0]	R/W
0x6E	OUT5_ENCZ	Output multiplexer select for Encoder Channel #1 Z Output	[5:0]	R/W
0x6F	OUT5_CONN	Output multiplexer select for Encoder Channel #1 Disconnect Output	[5:0]	R/W
0x70	OUT6_ENCA	Output multiplexer select for Encoder Channel #2 A Output	[5:0]	R/W
0x71	OUT6_ENCB	Output multiplexer select for Encoder Channel #2 B Output	[5:0]	R/W
0x72	OUT6_ENCZ	Output multiplexer select for Encoder Channel #2 Z Output	[5:0]	R/W
0x73	OUT6_CONN	Output multiplexer select for Encoder Channel #2 Disconnect Output	[5:0]	R/W
0x74	OUT7_ENCA	Output multiplexer select for Encoder Channel #3 A Output	[5:0]	R/W
0x75	OUT7_ENCB	Output multiplexer select for Encoder Channel #3 B Output	[5:0]	R/W

0x76	OUT7_ENCZ	Output multiplexer select for Encoder Channel #3 Z Output	[5:0]	R/W
0x77	OUT7_CONN	Output multiplexer select for Encoder Channel #3 Disconnect Output	[5:0]	R/W
0x78	OUT8_ENCA	Output multiplexer select for Encoder Channel #4 A Output	[5:0]	R/W
0x79	OUT8_ENCB	Output multiplexer select for Encoder Channel #4 B Output	[5:0]	R/W
0x7A	OUT8_ENCZ	Output multiplexer select for Encoder Channel #4 Z Output	[5:0]	R/W
0x7B	OUT8_CONN	Output multiplexer select for Encoder Channel #4 Disconnect Output	[5:0]	R/W
0x7C	DIV_FIRST	Pulse divider <i>N</i> first pulse select. Set to 0 for first pulse out of OUTN, set to 1 for first pulse out of OUTD	[ 3:0]	R/W
0x7E	SYS_RESET	Soft Reset <ul style="list-style-type: none"> <li>Resets all functional blocks</li> </ul>	[0]	W
0x7F	SOFT_IN	Soft In register <ul style="list-style-type: none"> <li>4-bits software set value</li> </ul>	[3:0]	R/W

### 7.1.2 Position Capture

Addr	Name	Register Description	Used Bits	R/W
0x80	POS1_SETLO	Quadrature Decoder 1 Load Low Register	[15:0]	R/W
0x81	POS1_SETHI	Quadrature Decoder 1 Load High Register	[15:0]	R/W
0x82	POS2_SETLO	Quadrature Decoder 2 Load Low Register	[15:0]	R/W
0x83	POS2_SETHI	Quadrature Decoder 2 Load High Register	[15:0]	R/W
0x84	POS3_SETLO	Quadrature Decoder 3 Load Low Register	[15:0]	R/W
0x85	POS3_SETHI	Quadrature Decoder 3 Load High Register	[15:0]	R/W
0x86	POS4_SETLO	Quadrature Decoder 4 Load Low Register	[15:0]	R/W
0x87	POS4_SETHI	Quadrature Decoder 4 Load High Register	[15:0]	R/W
0x88	PC_ENC	Quadrature Channel Select for Position Capture <ul style="list-style-type: none"> <li>0 – Quad #1</li> <li>1 – Quad #2</li> <li>2 – Quad #3</li> <li>3 – Quad #4</li> <li>4 – Sum of all (active?) quadrature counter outputs</li> </ul>	[ 2:0]	R/W
0x89	PC_TSPRE	Timestamp Clock Prescaler (time unit select)	[15:0]	R/W
0x8A	PC_ARM_SEL	Arm Input Control <ul style="list-style-type: none"> <li>0 – Soft arm</li> <li>1 – External arm input</li> </ul>	[0]	R/W
0x8B	PC_ARM	Soft Arm (Self cleared)	[0]	R/W
0x8C	PC_DISARM	Soft Disarm (Self cleared)	[0]	R/W
0x8D	PC_GATE_SEL	Gate output counter control <ul style="list-style-type: none"> <li>0 – Quadrature decoder position output</li> <li>1 – Timestamp counter</li> <li>2 – External gate input</li> </ul>	[1:0]	R/W
0x8E	PC_GATE_STARTLO	Gate Start/Delay Low Value	[15:0]	R/W
0x8F	PC_GATE_STARTHI	Gate Start/Delay High Value	[15:0]	R/W
0x90	PC_GATE_WIDLO	Gate Width Low Value	[15:0]	R/W
0x91	PC_GATE_WIDHI	Gate Width High Value	[15:0]	R/W

0x92	PC_GATE_NGATELO	Gate NGates Low Value	[15:0]	R/W
0x93	PC_GATE_NGATEHI	Gate NGates High Value	[15:0]	R/W
0x94	PC_GATE_STEPL0	Gate Step Low Value	[15:0]	R/W
0x95	PC_GATE_STEPHI	Gate Step High Value	[15:0]	R/W
0x96	PC_PULSE_SEL	Pulse output counter control <ul style="list-style-type: none"> <li>0 – Quadrature decoder position output</li> <li>1 – Timestamp counter</li> <li>2 – External gate input</li> </ul>	[1:0]	R/W
0x97	PC_PULSE_STARTLO	Pulse Output Start/Delay Low Value	[15:0]	R/W
0x98	PC_PULSE_STARTHI	Pulse Output Start/Delay High Value	[15:0]	R/W
0x99	PC_PULSE_WIDLO	Pulse Output Width Low Value	[15:0]	R/W
0x9A	PC_PULSE_WIDHI	Pulse Output Width High Value	[15:0]	R/W
0x9B	PC_PULSE_STEPL0	Pulse Output Step Low Value	[15:0]	R/W
0x9C	PC_PULSE_STEPHI	Pulse Output Step High Value	[15:0]	R/W
0x9D	PC_PULSE_MAXLO	Pulse Output NPulse Low Value	[15:0]	R/W
0x9E	PC_PULSE_MAXHI	Pulse Output NPulse High Value	[15:0]	R/W
0x9F	PC_BIT_CAP	Position Capture – data capture mask bits[3 : 0] : Encoder N capture bits[5 : 4] : System bus 1, 2 capture bits[10: 6] : Pulse divider N counter value capture	[10:0]	R/W
0xA0	PC_DIR	Position Capture – scan direction	[0]	R/W
0xA1	PC_PULSE_DLYLO	Position Capture – pulse delay Low Value	[15:0]	R/W
0xA2	PC_PULSE_DLYHI	Position Capture – pulse delay High Value	[15:0]	R/W

### 7.1.3 System Status and Control

Addr	Name	Register Description	Used Bits	R/W
0xF0	SYS_VER	<i>FPGA Firmware Version</i>	[15:0]	R
0xF1	SYS_STATERR	System Status bits[3:0] : Pulsegen N module error bit[4] : SRAM interface buffer overflow error	[15:0]	R
0xF2	SYS_STAT1LO	System Bus – bits[15:0]	[15:0]	R
0xF3	SYS_STAT1HI	System Bus – bits[31:16]	[15:0]	R
0xF4	SYS_STAT2LO	System Bus – bits[47:32]	[15:0]	R
0xF5	SYS_STAT2HI	System Bus – bits[63:48]	[15:0]	R
0xF6	PC_NUM_CAPLO	Pulse counter lower word	[15:0]	R
0xF7	PC_NUM_CAPHI	Pulse counter upper word	[15:0]	R

## 7.2 RS232 Protocol

Communication to/from Zebra is performed over RS232 using an ASCII protocol. See section 5.3.2 for baud rate and serial port information. Each command terminates with “\n” delimiter character. Zebra ignores any carriage returns in the message.

### 7.2.1 Register Write

Write a 16-bit value into one of the R/W registers shown in section 7.1

Client sends:           W<AA><DDDD>\n

Zebra responds:       W<AA>OK\n

Where:

- <AA> : 8-bit hex register address as shown in section 7.1
- <DDDD> : 16-bit register write value, big endian

E.g. to write 1 to PC\_ARM (register 0x8B):

Client sends:           W8B0001\n

Zebra responds:       W8B<AA>OK\n

### 7.2.2 Register Read

Read back the value of a register shown in section 7.1

Client sends:           R<AA>\n

Zebra responds:       R<AA><DDDD>\n

Where:

- <AA> : 8-bit hex register address as shown in section 7.1
- <DDDD> : 16-bit register write value

E.g. to read OUT1\_TTL (register 0x60) which is set to AND1 (element 32 on the system bus):

Client sends:           R60\n

Zebra responds:       R6000200K\n

### 7.2.3 Configuration Store

Store the values of all registers shown in section 7.1 into external flash and respond when complete. This action takes about a second to complete.

Client sends:           S\n

Zebra responds:       SOK\n

#### 7.2.4 Configuration Restore

Load the values of all registers shown in section 7.1 from external flash and respond when complete. This action takes about a second to complete.

Client sends: L\n

Zebra responds: LOK\n

#### 7.2.5 Position Capture Data Offload

Position Capture data is offloaded automatically without user issuing a command. Note that all messages begin with the character P so that they can be filtered out of the command response messages above. Data is encapsulated between “PR” and “PX” character sets, and represented in hex ASCII format as shown below.

```
PR\n
P<TS><ENC1><ENC2><ENC3><ENC4><SYS1><SYS2><DIV1><DIV2><DIV3><DIV4>\n
P<TS><ENC1><ENC2><ENC3><ENC4><SYS1><SYS2><DIV1><DIV2><DIV3><DIV4>\n
....
P<TS><ENC1><ENC2><ENC3><ENC4><SYS1><SYS2><DIV1><DIV2><DIV3><DIV4>\n
PX\n
```

Where:

- <TS> : 32-bit timestamp, in 50MHz clock ticks since PC\_ARM
- <ENC*N*> : 32-bit encoder channel #*N* position
- <SYS*N*> : 32-bit upper and lower words of System Bus.
- <DIV*N*> : 32-bit divider channel #*N* output

Timestamp field is stored and sent for each pulse output, while all other fields are user selected depending on the value of register PC\_BIT\_CAP.

E.g. if **ENC1** and **DIV1** are selected, the interrupts will take the following form:

```
PR\n
P012345670123456701234567\n
....
PX\n
```

Data offload starts automatically by sending “PR” on Position Capture arm. As soon as data is captured, it is sent while scanning continues. When all captured data is offloaded from history buffer, “PX” is sent concluding the data transfer. In case of buffer overrun, an error flag in SYS\_STATERR is set.