Master Oscillator Distribution Board

(V2)

9/14/18

Contributors

F.J. Barbosa – Design / PCB layout

C. Stanislav – PCB layout / Assembly

E. Jastrzembski – FPGA firmware / Testing / Documentation

B. Moffit – Software driver

A.Toro – Design

Summary

The Master Oscillator Distribution board (MO) accepts a sinusoidal control signal from the accelerator (nominal 499 MHz) and produces multiple logic level signals having frequencies that are integer divisors of the input signal frequency. The objective is to record these correlated signals in each detector subsystem, and thus identify the timing relationships among the subsystems. The device is implemented as a 6U VME module. All output signal characteristics are programmed through standard VME protocols.

Input/Output Signals

Figure 1 identifies the MO front panel connections and the signal levels they carry.

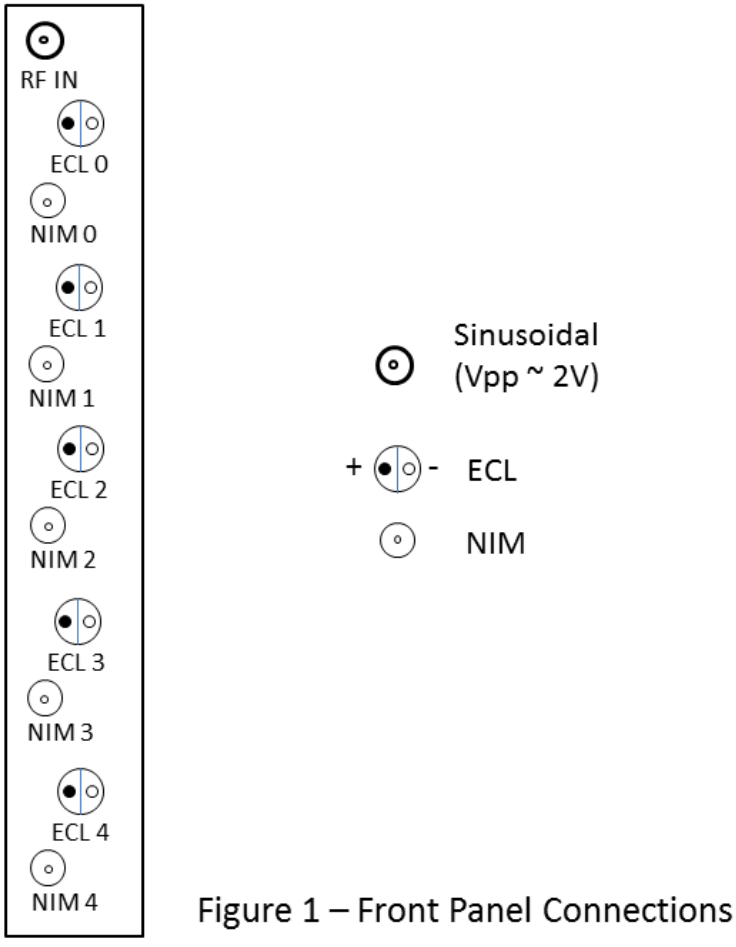

Figure 1 – Front Panel Connections

Overview

A functional schematic of the Master Oscillator Distribution Board is shown in Figure 2. A single sinusoidal input signal is accepted from the accelerator control system (nominal 499 MHz) and is converted to a logic level signal. This logic level signal undergoes two stages of division. In the first stage (PS_0), an exact copy of this logic level signal is produced (÷1). A logic level signal with a frequency equal to the input frequency divided by 2, 4, 8, or 16 is also generated. The dividing factor of this signal is selected under software control (PRESCALE CONTROL register).

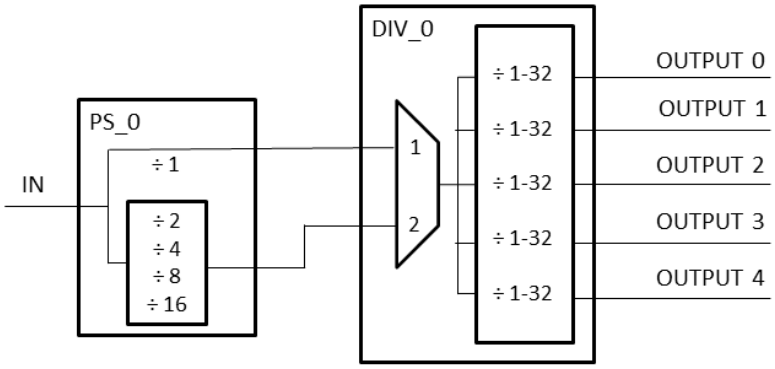

Figure 2 – Functional schematic of Master Oscillator Distribution Board

The second stage of division (DIV_0) is implemented with a high-performance clock distribution chip (AD9512). This chip accepts two possible sources of input signal. Five independent outputs can be individually programmed with integer dividing factors of 1 to 32. A wide range of duty cycles and phase shifts can be programmed for the individual outputs. The outputs can be synchronized to start up with a programmed relationship upon receipt of a synchronization command (SYNC).

The AD9512 has 90 on-chip registers to control its behavior. Appendix 1 lists the most important of these. For more sophisticated configurations, see the AD9512 data sheet (http://www.analog.com/media/en/technical-documentation/data-sheets/AD9512.pdf). All registers are accessed through a serial control port. The AD9512 serial control port is a flexible, synchronous, serial communications port that is compatible with the **S**erial **P**eripheral **I**nterface Bus protocol (SPI). The serial control port allows read/write access to all registers that configure the AD9512. Only single byte data transfers are supported in our implementation; thus, a 24-bit quantity is serialized. The AD9512 SPI bus is connected to the module's FPGA. Access to the SPI bus is through a module register (DIVIDER CONTROL). Access is transparent – the lower 24-bits of this register are *exactly* the 24-bit quantity to be serialized. For register reads of DIV_0, the data (single byte) appears in the register DIVIDER 0 READ after the transaction is started through register DIVIDER CONTROL.

Data is written into a serial control port buffer area, not directly into the AD9512's actual control registers. An additional operation is needed to transfer the serial control port buffer contents to the actual control registers of the AD9512, causing them to take effect. This update command consists writing to Register 5Ah<0> = 1b. This update bit is self-clearing (it is not required to write 0 to it to clear it). Since any number of bytes of data can be changed before issuing an update command, the update simultaneously enables all register changes since any previous update. Phase offsets or divider synchronization will not become effective until a SYNC is issued. A readback request reads the data that is in the serial control port buffer area, not the active data in the AD9512's actual control registers.

Setting the Divide Ratio and Duty Factor

The divide ratio is determined by the values written via the SPI to the registers that control each individual *chip output*, OUT0 to OUT4. These are the even numbered registers beginning at 4Ah and going through 52h. Each of these registers is divided into bits that control the number of clock cycles the divider output stays high (high_cycles <3:0>) and the number of clock cycles the divider output stays low (low_cycles <7:4>). Each value is 4 bits and has the range of 0 to 15.

The divide ratio is set by

$$\textit{Divide Ratio} = (\textit{high\_cycles} + 1) + (\textit{low\_cycles} + 1)$$

Example 1:

Set the Divide Ratio = 2

high_cycles = 0

low_cycles = 0

Divide Ratio = (0 + 1) + (0 + 1) = 2

Example 2:

Set Divide Ratio = 8

high_cycles = 3

low_cycles = 3

Divide Ratio = (3 + 1) + (3 + 1) = 8

Note that a Divide Ratio of 8 may also be obtained by setting:

high_cycles = 2

low_cycles = 4

Divide Ratio = (2 + 1) + (4 + 1) = 8

Although the second set of settings produces the same divide ratio, the resulting duty cycle is not the same.

The duty cycle and the divide ratio are related. Different divide ratios have different duty cycle options. For example, if Divide Ratio = 2, the only duty cycle possible is 50%. If the Divide Ratio = 4, the duty cycle can be 25%, 50%, or 75%.

The duty cycle is set by

$$Duty\ Cycle = (high\_cycles + 1)/[(high\_cycles + 1) + (low\_cycles + 1)]$$

From Figure 2 we can see that a maximum total dividing factor of 512 is achievable for any output of the Module.

Module Programming Steps

(1) Program first stage dividing factor.
(2) Program DIV_0 clock selection. To use ÷ 1 from first stage, CLK 1 must be selected . To use ÷ 2, ÷ 4, ÷ 8, or ÷ 16 from first stage, CLK 2 must be selected.
(3) Program second stage divider, phase shifts for all outputs.
(4) Update register values for the divider chip.
(5) Synchronize outputs by issuing SYNC command to the divider chip.

Module Registers

The MO is programmed by the user through VMEbus protocols (ANSI/IEEE STD1014-1987). The device meets all VMEbus standards. The MO is categorized as an A24 – D32 VMEbus slave. All storage locations can be accessed as both Supervisory and Non-privileged data (AM = 3D, 39).

The base address (A23 – A8) is selected by four rotary switches on the board. The module occupies 256 bytes of VME address space (64 32-bit registers). Only seven registers are currently defined. The remaining space is reserved for testing and future use.

VERSION (local address = 0x0)

[31…16] – (R) – MO identifier = 0x1000

[15…8] – (R) – board revision = **0x02**

[7…0] – (R) – firmware revision

CONTROL/STATUS (CSR) (local address = 0x4)

31 – (W) – hard reset

30 – (W) – soft reset

29 – (W) – reset initial prescale chip

[28…0] – reserved

PRESCALE CONTROL (local address = 0x8)

[1…0] – (R/W) – initial prescale value: 0 => ÷ 2,
1 => ÷ 4,
2 => ÷ 8,
3 => ÷ 16

[31…2] – reserved

DIVIDER CONTROL (local address = 0xC)

31 – reserved (read as '0')

30 – reserved (read as '1')

[29…24] – (R/W) – all bits must be '0'

23 – (R/W) – '1' = read serial data, '0' = write serial data

[22…15] – (R/W) – all bits must be '0'

[14…8] – (R/W) – divider internal register address (see Appendix 1)

[7…0] – (R/W) – data for write to internal register (ignored for serial read)

DIVIDER 0 READ (local address = 0x10)

[31…8] – (R) – reserved (read as '0')

[7…0] – (R) – data from serial read of divider 0

SPARE 1 (local address = 0x14)

[31…0] – (R/W) – (not currently used)

TEST (local address = 0x18)

[31…0] – (R/W) – data for test of data integrity

Appendix 1 – Important Registers of the AD9512

Addr = 0x00 Serial Port Configuration

[0…3] – not used

4 – (R/W) – must be ‘1’

5 – (R/W) – soft reset (must toggle bit)

6 – (R/W) – must be ‘0’

7 – (R/W) – must be ‘0’

Addr = 0x45 Clocks Select

0 – (R/W) – ‘1’ = CLK 1, ‘0’ = CLK 2

1 – (R/W) – power down CLK 1

2 – (R/W) – power down CLK 2

[3…7] – not used

Addr = 0x4A Divider – channel 0 (**output 4**)

[3…0] – (R/W) – high cycles

[7…4] – (R/W) – low cycles

Addr = 0x4B Divider – channel 0 (**output 4**)

[6…0] – (R/W) – ‘0’

7 – (R/W) – bypass divider (for ÷ 1)

Addr = 0x4C Divider – channel 1 (**output 1**)

[3…0] – (R/W) – high cycles

[7…4] – (R/W) – low cycles

Addr = 0x4D Divider – channel 1 (**output 1**)

[6…0] – (R/W) – ‘0’

7 – (R/W) – bypass divider (for ÷ 1)

Addr = 0x4E Divider – channel 2 (**output 0**)

[3…0] – (R/W) – high cycles

[7…4] – (R/W) – low cycles

Addr = 0x4F Divider – channel 2 (**output 0**)

[6…0] – (R/W) – ‘0’

7 – (R/W) – bypass divider (for ÷ 1)

Addr = 0x50 Divider – channel 3 (**output 3**)

[3…0] – (R/W) – high cycles

[7…4] – (R/W) – low cycles

Addr = 0x51 Divider – channel 3 (**output 3**)

[6…0] – (R/W) – ‘0’

7 – (R/W) – bypass divider (for ÷ 1)

Addr = 0x52 Divider – channel 4 (**output 2**)

[3…0] – (R/W) – high cycles

[7…4] – (R/W) – low cycles

Addr = 0x53 Divider – channel 4 (**output 2**)

[6…0] – (R/W) – ‘0’

7 – (R/W) – bypass divider (for ÷ 1)

Addr = 0x58 Synchronize outputs

[0…1] – (R/W) – must be ‘0’

2 – (R/W) – synchronize (must toggle bit)

[3…7] – (R/W) – must be ‘0’

Addr = 0x5A Update registers

0 – (W) – update registers

[1…7] – not used