INTRODUCTION TO PROJECT

PROJECT NAME: ULTRA FAST MULTI-CHANNEL SCALER

AIM:

I. To implement a FPGA based ultra-fast multi-channel to count electrical impulses based on the number of photons received by a photon detector in the experiment.

II. Design the board for the same having custom designed clock generators.

III. In-circuit checking of the core on an FPGA and the software interface.

SCOPE OF PROJECT:

1. To Program the MCS system in Verilog HDL
2. To test the MCS synthesized onto a typical FPGAs or an ASIC Library.
3. To interface the FPGA to IBM ™ PC using ISA standard.

SPECIFICATION FOR MULTICHANNEL SCALAR

1. Input Rate: 100 MHz
2. Minimum gate width for counter: 20 ns
3. Pulse width Resolution: 10ns.
4. Dwell Time from 20ns to 16us.
5. Input:
    - TTL with 100 MHz (Max).
    - NIM with – 0.8V 100Mhz (Max).
    - Plug-in-card (ISA Interface).
6. Power Supply: ±12V, ±5V, ±3.3V.

PRINCIPLE

A Multi-Channel Scaler (MCS) records the counting rate of events as a function of time. When a scan is started, the MCS begins counting input events in the first channel of its digital memory. At the end of the pre-selected dwell time, the MCS advances to the next channel of memory to count the events. This dwell and advance process is repeated until the MCS has scanned through all the channels in its memory. A display of the contents of the memory shows the counting rate of the input events versus time. In repetitive measurements, where the start of the scan can be synchronized with the start of the events, multiple scans can be summed to diminish the statistical scatter in the recorded pattern. In repetitive measurements, where the start of the scan can be synchronized with the start of the events; multiple scans can be summed to diminish the statistical scatter in the recorded pattern.

A Multi-channel Scalars counts the number of events that occur during the time interval t to dt as function of time. The interval dt is called the dwell time. Then time t is quantized into channels or bins by the relation t = i*dt were I is the bin number (an integer).Dwell times can be selected from nanosecond to hours. and the total number of bins ranges from 4 to 16,000. When the scan is started, the MCS begins counting input events in the first channel of its digital memory. At the end of the reselected dwell time the MCS advances to the next channel memory to count the events, this dwell and advanced process is repeated until the MCS has scanned through all the channels of its memory. A display of the contents of memory shows the counting rate of the input events versus times.

MCS vs. a Counter and Timer: Although a computer-controlled counter and timer can also be used to record counting rate as function of time, the MCS technology offers several important advantages. The time taken to read, clear, and start a conventional counter at the end of each counting interval can range from microseconds to milliseconds. This dead time causes significant gaps in the recorded data. High performance Multi-channel sealers have negligible dead time between counting intervals (channels), and this avoids blank regions in the recorded time profiles. Conventional counters and timers rarely handle time intervals shorter than 10ms, whereas Multi-channel scalar are 5ns.

The MCS can also function as a multiple-stop time spectrometer. In a typical application, the MCS scan is started when a LASER emits a brief flash of light. The light photons are reflected back to the detector located near the LASER as they encounter objects at various distances in the line of sight. The resulting "stop" pulses generated in the detector are counted as input events by the MCS. Thus a spectrum of the number of photons versus their round-trip flight times is recorded in the MCS memory. By design, the MCS can accept multiple stop pulses in each scan. The channel numbers in memory can be calibrated to read in terms of round-trip flight time or in distance to the reflecting object. Summing the spectra from multiple LASER flashes improves the signal-to-noise ratio.

LITERATURE REVIEW

FPGA

It is the basic chip where the codes are downloaded. The codes are first synthesized and then downloaded onto this chip using the xchecker cable. The FPGA then behaves as MCS itself. FPGA (Field programmable gate arrays) these are also programmable logic devices that have some advantages when compared with CPLDs. The main fabrication technology behind FPGA is CMOS process technology.

FPGA is a piece of programmable hardware. The functioning of circuit is determined by individual users. The configuration of architecture of hardware could be specified by the user with a "configuration bit stream", that is a piece of software which instructs the hardware how to configure or wire itself up.

FPGAs can be used virtually in any digital logic system to perform function of a custom LSI circuit, like a gate array. However these devices are user programmable and even reprogrammable in the system. Standard off the shelf devices in many different densities, speeds, operating temp. ranges and packages are available from which the user selects the appropriate device and then converts the design idea or schematic into configuration data file using a development system software and loads this file into FPGA.

Advantages:

- *High integration*: The different login functions such as flip flops, counters, latches, adders, multiplexers, decoders logic elements such as and, nor, nand, etc. can be implemented in a single device.

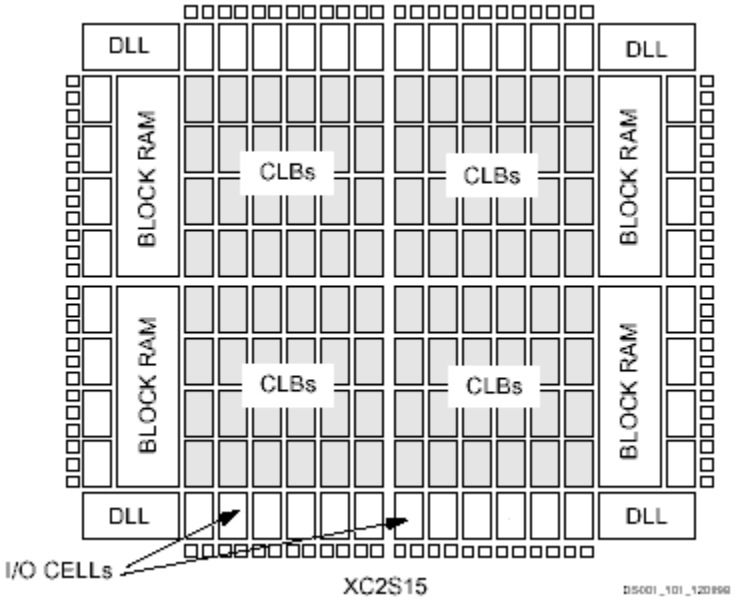
  The digital system is traditionally implemented in a single device using standard devices like LSIs, MSIs, PALs, etc. can now be integrated into a single device providing high integration. This gives the following advantages:

- Less board space
- High reliability.
- Low power consumption
- Low cost.
- Easy design changes: FPGAs can be reprogrammed in the system to incorporate changes in the design.
- Shorter time to market: Systems based on FPGAs can be designed and validated in shorter cycle times which is important for faster introduction of your product in the market.
- No non-recurring engg. Cost: FPGAs do not have the NRE cost and associated risk typically required for mask programmed gate array solutions (ASICs).
- Latest evelopment tools support

FPGAs are designed with EDA (electronic design automation) tools which improve design productivity and reduce development time.

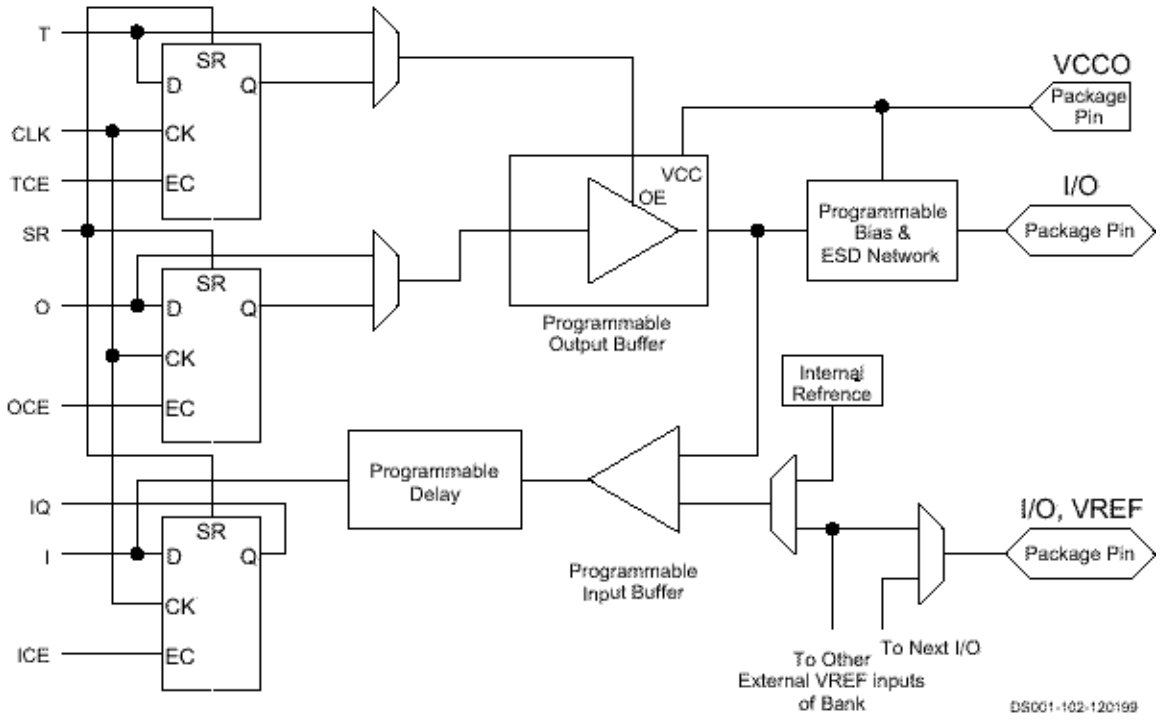
The basic architecture of FPGA:

Field Programmable Gate Arrays (FPGAs) is implemented with a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs), interconnected by a powerful hierarchy of versatile routing resources. The architecture also provides advanced functions such as Block RAM and clock control blocks.



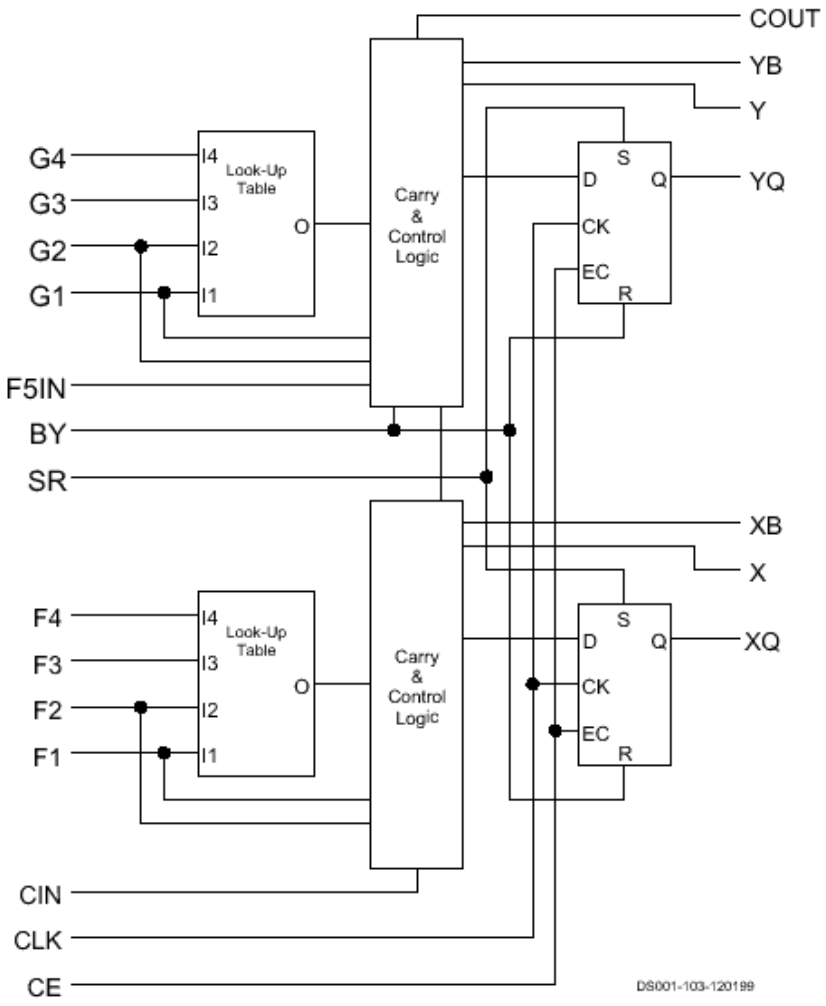
A. Input/Output Block

IOB features inputs and outputs that support 16 I/O signaling standards, including LVCMOS, HSTL, SSTL, and GTL. These high-speed inputs and outputs are capable of supporting various state of the art memory and bus interfaces. The three IOB registers function either as edge-triggered D-type flip-flops or as level sensitive latches. Each IOB has a clock signal (CLK) shared by the three registers and independent clock enable (CE) signals for each register. In addition to the CLK and CE control signals, the three registers share a Set/Reset (SR). For each register, this signal can be independently configured as a synchronous Set, a synchronous Reset, an asynchronous Preset, or an asynchronous Clear.



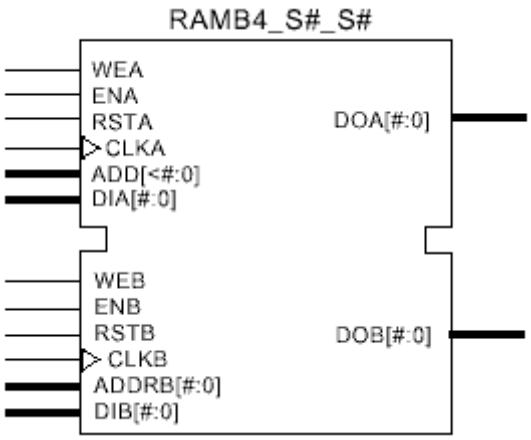
B. Logic Cells

The basic building block of the CLB is the logic cell (LC). An LC includes a four-input function generator, carry logic, and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each CLB contains four LCs, organized in two similar slices. In addition to the four basic LCs, CLB contains logic that combines function generators to provide functions of five or six inputs. Consequently, when estimating the number of system gates provided by a given device, each CLB counts as 4.5 LCs. Function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 x 1-bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16 x 2-bit or 32 x 1-bit synchronous RAM, or a 16x1-bit dual-port synchronous RAM. The LUT can also provide a 16-bit shift register that is ideal for capturing high-speed or burst-mode data. This mode can also be used to store data in applications such as digital signal processing. The storage elements in the slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches.



C. Block RAM

FPGAs incorporate several large Block SelectRAM+ memories. These complement the distributed SelectRAM+ resources that provide shallow RAM structures implemented in CLBs. Block SelectRAM+ memory blocks are organized in columns. Devices contain two such columns, one along each vertical edge. These columns extend the full height of the chip. Each memory block is four CLBs high, and consequently, a Spartan-II device 8 CLBs high will contain 2 memory blocks per column, and a total of 4 blocks.



D. Delay-Locked Loop

Associated with each global clock input buffer is a fully digital Delay-Locked Loop (DLL) that can eliminate skew between the clock input pad and internal clock input pins throughout the device. Each DLL can drive two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges reach internal flip-flops exactly one clock period after they arrive at the input. This closed-loop system effectively eliminates clock-distribution delay by ensuring that clock edges arrive at internal flip-flops in synchronism with clock edges arriving at the input.

Although the above details are of the xilinx FPGA in particular the architecture of the various FPGA families is almost similar to that of Xilinx.

Configuration and Power Down

Configuration is the process by which the FPGA is programmed with the configuration file generated by the Xilinx development system. Spartan-II devices support both serial configuration, using the master/slave serial and JTAG modes, as well as byte-wide configuration employing the slave parallel mode (similar to the Express mode). The Spartan-II Family also includes a Power Down pin that can be asserted to bring quiescent current down to 100 uA typical. In Power Down mode, all I/Os are disabled and all registers and memory retain their data. The FPGA used in our PCB is **XC2s200** which is from Spartan II family of FPGA.

Introduction to Verilog HDL

Designing with Boolean Equation: It wold be hard to design any digital system without understanding such basic building blocks as gates and flip-flops. Most logic circuits based on gates and flip-flops have been traditionally designed with Boolean equations. Many techniques were developed to optimize this method, including minimization of equation for a more effective use of gates and flip-flops.

The Boolean design technique requires an equation for each flip-flop and blocks of gates. This makes Boolean equation impractical for large designs containing hundreds of flip-flops as this would result in an even larger number of logic equations.

Theoretically, any system can be represented by Boolean equation. However, it would e impractical to deal with the thousands of logic equations that today's designs may require.

Schematic-Based Design: Schematic-Based design methods expanded he capabilities of Boolean equations by accepting not only gates and flip-flops but some additional circuits, a hierarchical structure can e created. This allows entering hierarchical designs that can accommodate larger numbers of components with less effort. For most people designing is more user-friendly than designing with Boolean equations.

Most people prefer a graphical representation of a design because if shows more clearly the relationships between the various design blocks.

Since schematics provide a graphical representation, their use is quite popular. For may years, schematics were considered the optimal choice for design representation until new device densities made them too limited and time consuming.

Drawbacks of Traditional Methods: Despite their ease of use, the traditional Boolean equation and schematic design methods have some drawbacks:

1. The most important of these is that a system is always specified as a network of interconnected elements. But this is not how the specification of a system is

created. A system specification is always given in the form of expected system behavior i.e. what should a system do in a particular set of circumstances.

2. Another shortcoming of traditional design method is he handling of large complex designs. Dealing with hundreds of logical equations is hard but feasible. However, handling thousands of logical equation is hard to imagine. Its is widely accepted that schematics with over six thousand gates are becoming incomprehensible. Since he newest integrated circuits contain millions of gates and their densities are still growing, this calls for a new design method.

Hardware Description Languages

One of the major drawbacks of traditional design methods is the manual translation of design description into a set of logical equations or a schematic. This step can be eliminated with hardware description languages (HDL). For example, most HDL tools allow using finite state machines foe sequential systems and truth tables for combinatorial modules. Such design descriptions can be automatically converted into HDL code that can be implemented by synthesis tools.

Hardware description languages fount their principal application in Programmable Logic Devices (PLD) of various complexities from simple PLD's up to Complex PLD's (CPLD's) and Field programmable Gate Arrays (FPGA's). There are several HDL languages in use today. The most popular ones are Abel, Palasm and Cupl (used for less complex devices) and Verilog and VHDL (for larger CPLD and FPGA devices).

Verilog HDL

Verilog, or Verilog HDL is a hardware description language developed in 1984-1985 by Philip Moorby who needed a simple, intuitive and effective way of describing digital circuits for modeling, simulation and analysis purposed. The language became property of Gateway Design Automation, which was late acquired by Cadence Design Systems. From 1990 Cadence opened the language to the public, which led to the standardization of the language by IEEE in 1995.

Features of Verilog:

1. Strong Background : The Verilog development and promotion has been supported since 1991 by the Open Verilog International (OVI), which was established for promotion of this language. Until 1990 Verilog was owned by a commercial company, which strengthened its practical approach, but also restricted its popularity. Verilog has been standardized in 1995 b IEEE as Standard 1364.

2. Industrial Support : Verilog Has always been very popular with ASIC designers because it is easy to learn, and allows for fast simulation and effective synthesis. According to EE Times of all design submitted in 1993 o ASCI foundries, 85% were designed in Verilog. It is estimated that over one billion US dollars have been invested in Verilog tools within the last ten years.

3. Verilog is Universal: Verilog allows the entire design process to be performed within on design environment, including analysis and verification. However, Verilog is not well suited for complex system level design and this is its main drawback.

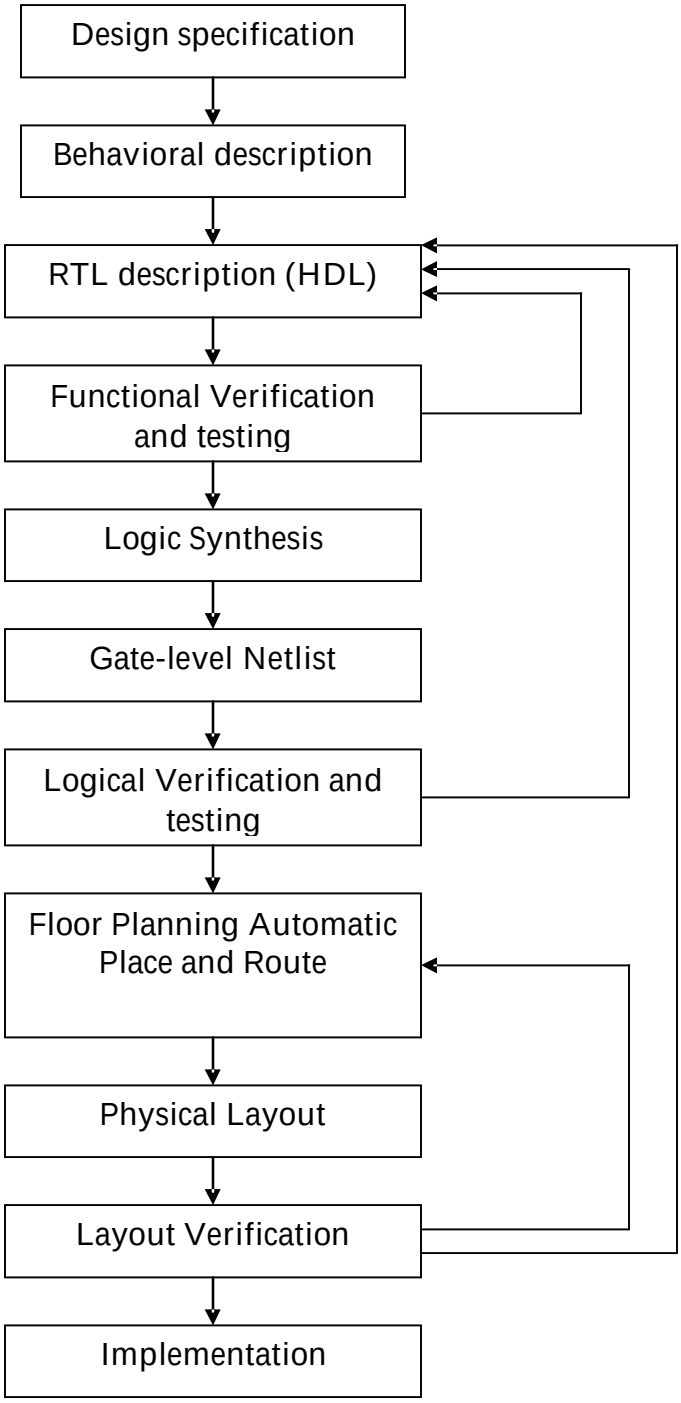
4. Extensibility: The IEEE Standard 1364 contains definition of Verilog PLI – Programming Language Interface that allows for extension of Verilog's capabilities.

Verilog vs. VHDL

Verilog	**VHDL**
Very good acceptance in ASIC, particularly lower level designs (register level transfer ad below)	Relatively weaker in lower level designs, but superior in higher and system-level designs
Verilog is known for fast simulations	Results in slower simulation but constantly improving
Relatively simple, easy I first contacts	Flexible but also difficult and has a complex character
Used mostly in North America, Asia and Japan, especially among industrial supporters. Not popular in Europe	Popular in Europe but significant number in US and Canada. Disliked in Japan but gaining popularity worldwide

Design Methodology:

The typical design flow for designing the VLSI circuits is shown in the diagram below. This design flow is typically used by the designers who use HDLs.



In any design, specifications are written first. Specification describes abstractly the functionality, interface and overall architecture of the digital circuit to be designed.

A behavioral description is then created to analyze the design in terms of functionality performance, compliance to standards, and also high level issues. Behavioral description can be written in HDLs.

The behavioral description is manually converted to an RTL description in an HDL. The designer has to describe the data flow and that will implement the desired digital circuit.

Logic synthesis tools convert the RTL description to a gate level netlist. A get-level netlist is a description of the circuit in terms of gates and the connections between them.

The gate-level netlist is the input to an Automatic Place and Route tool, which creates a layout.

The layout is verified and then fabricated on chip.

Here at each the verification of the design is carried out at each level. The function verification and testing at the RTL description level and logical verification and testing after the logic synthesis and also the final layout verification before actual fabrication is carried out by simulation of the software core using various different simulators.

Digital design provides the following advantages over the hardware design:

Top-Down Approach for Large Projects - HDLs are used to create complex designs. The top-down approach to system design supported by HDLs is advantageous for large projects that require many designers working together. After the overall design plan is determined, designers can work independently on separate sections of the code.

Functional Simulation Early in the Design Flow - You can verify the functionality of your design early in the design flow by simulating the HDL description. Testing your design decisions before the design is implemented at the RTL or gate level allows you to make any necessary changes early in the design process. But if we use the hardware it can be tested only after the total fabrication is done and circuit is made.

Automated synthesis - you can apply the automation techniques used by the synthesis tool (such as machine encoding styles or automatic I/O insertion) during the optimization of your design to the original HDL code, resulting in greater efficiency.

Reuse of RTL Code - You can retarget RTL code to new FPGA architectures with a minimum of recoding. The core once designed and verified can be used anytime in future for designing a complex circuit with the core as a part of it. For example the 8051 core designed can be interfaced with USB or GPIB core.

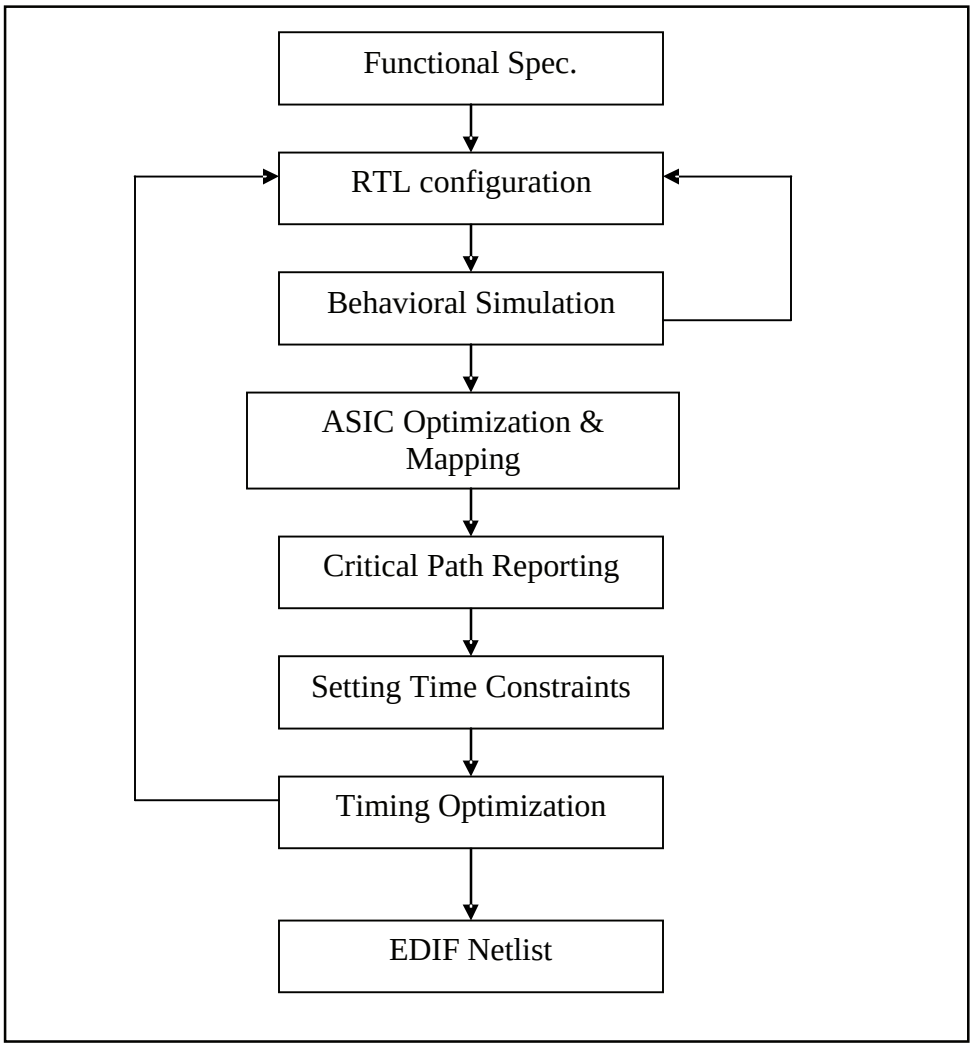
High integration- The soft core can be downloaded into FPGAs and ASICs and hence provide the advantages of high complexity, density, reliability, low cost, power consumption and small physical size.

Due to all these advantages provided by the soft core the use of these cores is increasing and it is replacing the hardware existing in the market.

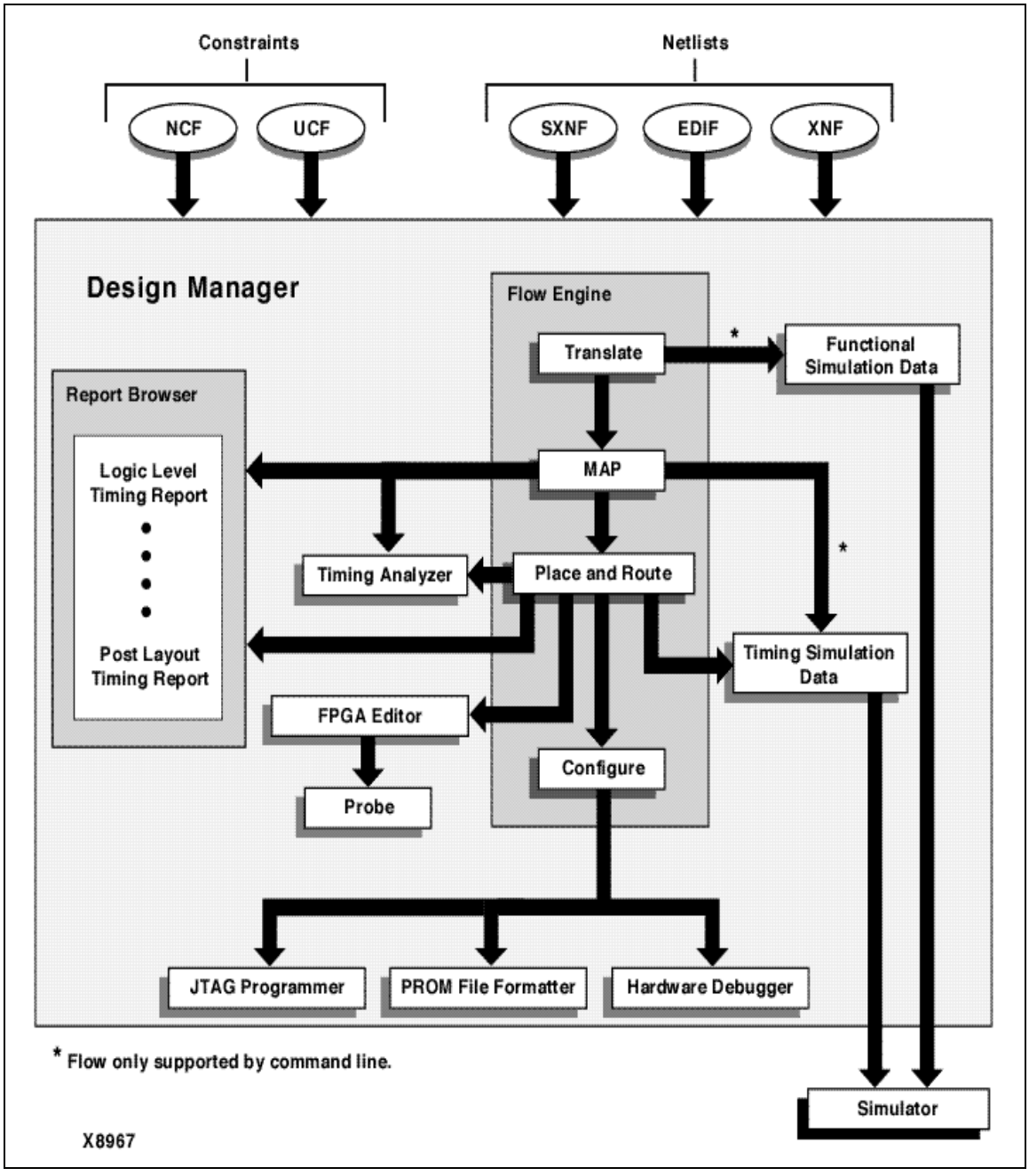
Synthesis

Synthesis is a process of converting a high level description of the design into an optimized gate-level representation, given a standard cell library and certain design constraints. A standards cell library can have simple cells, such as basic logic gates like and, or, and nor, or macro cells, such as adders, mixes and special flip-flops. A standard cell library is also known as technology library. We are carrying out synthesis using Leonardo exlemplar software.

The basic flow of the process is explained in the **block diagram** below:



The **.edif** file which is the output of the synthesis process is now converted into **.bit** file which can be downloaded onto the FPGA, in the following manner:



The various process in the block diagram are explained in brief below:

- **Translation:** During translation, the NGD Build program is launched, and performs the following functions:
    - Converts input design netlists and writes results to a single merged NGD netlist. The merged netlist describes the logic in the design as well as any location and timing constraints.
    - Performs timing specification and logical design rule checks.
    - Adds the UCF to the merged netlist.
- **Using the Constraints Editor:** The Constraints Editor allows you to edit constraints previously defined (through a UCF), and to add new constraints to your design. Input files to the Constraints Editor include the following:
- **NGD (Native Generic Database) file:** This file is input to the Map program, which then outputs the physical design database, an NCD (Native Circuit Description) file.
- **Corresponding UCF:** By default, when the NGD file is opened, an existing UCF file with the same base name as the NGD file is used. Alternatively, you can specify the name of the UCF file.

  Upon successful completion, the Constraints Editor writes out a valid UCF file. NGDBuild uses the UCF file, along with design source netlists to produce a newer NGD file that incorporates the changes made. The NGD is then read by the MAP program (the next step in the design flow).
- **Mapping the Design:** Now that all implementation strategies have been defined (options and constraints), the next step is implementing the design.

  The Map program performs the following functions.
    - Allocates CLB and IOB resources for all basic logic elements in the design.

- Processes all location and timing constraints, performs target device optimizations, and runs a design rule check on the resulting mapped netlist.

- **Using Timing Analysis to Evaluate Block Delays After Mapping:** After the design is mapped, you can use the Logic Level Timing Report to evaluate the logical paths in the design. Because the design is not placed and routed yet, actual routing delay information is not yet available. The timing report describes the logical block delays and estimated routing delays. The net delays that are provided are based on an optimal distance between blocks

- **Placing and Routing the Design:** After the mapped design is evaluated to verify that block delays are reasonable given the design specifications, the design can be placed and routed. The Flow Engine can perform the following place and route algorithms.

    - Timing Driven - run PAR with timing constraints specified from within the input netlist or from a constraints file

    - Non-Timing Driven - run PAR and ignore all timing constraints

- **Timing Simulation of Your Design:** Timing simulation is important in verifying the operation of your circuit after the worst-case placed and routed delays are calculated for your design. In many cases, you can use the same test bench that you used for functional simulation to perform a more accurate simulation with less effort. You can compare the results from the two simulations to verify that your desipgn is performing as initially specified.

- **Downloading to the Device and In-system Debugging:** After you have verified the functionality and timing of your placed and routed design, you can create a design data file to download for in-system verification. The design data or bitstream (.bit) file is created from the placed and routed .ncd file. A bitstream file is a stream of data that contains location information for logic on a device, that is, the placement of Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs), TBUFs, pins, and routing elements. The bitstream also includes empty placeholders that are filled with the logical states sent by the device during a readback. Only the memory elements,

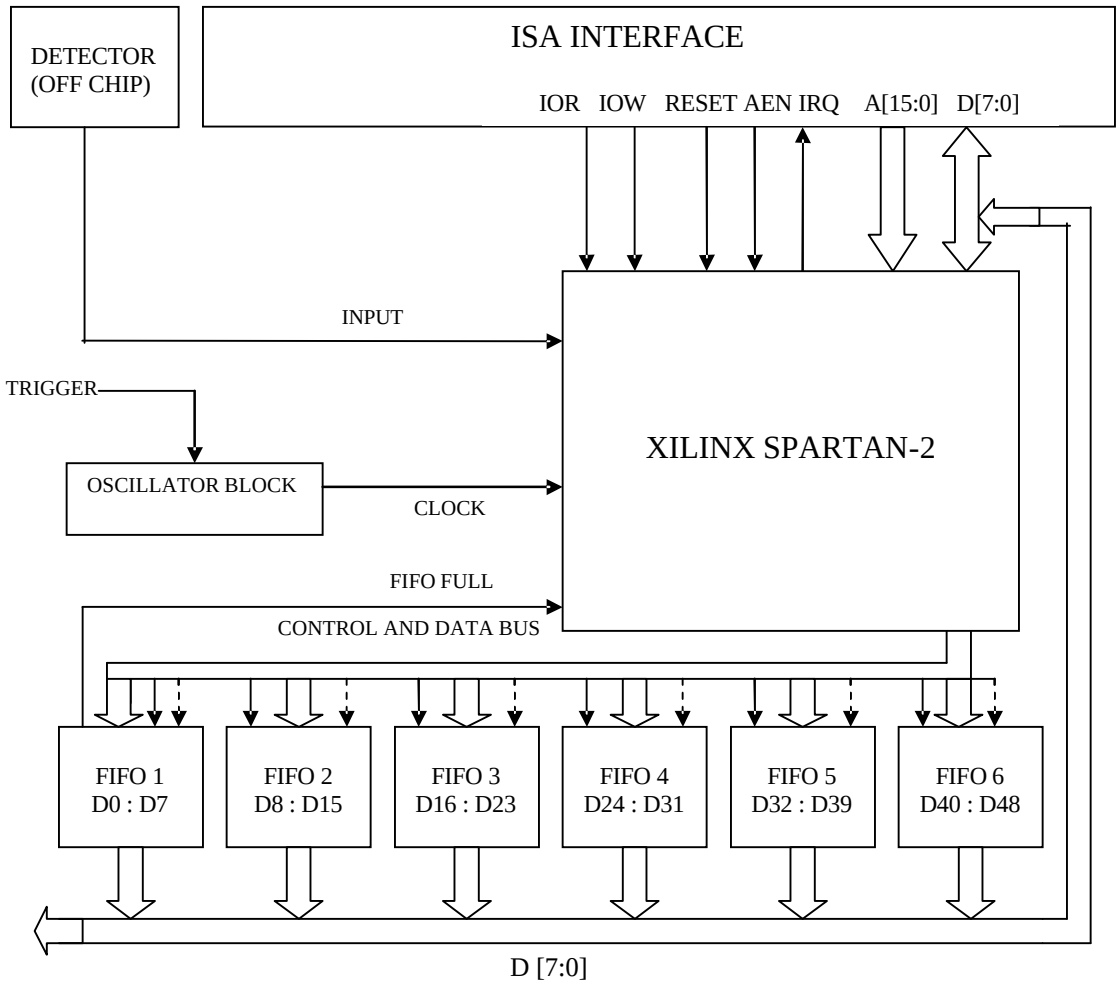
such as flip-flops, RAMs, and CLB outputs, are mapped to these placeholders, because their contents are likely to change from one state to another. When downloaded to a device, a bitstream configures the logic of a device and programs the device so that the states of that device can be read back.

The Hardware Debugger allows downloading the data to the FPGA using your computer's serial port. The Hardware Debugger can also synchronously or asynchronously probe external or internal nodes in the FPGA. Waveforms can be created from this data and correlated to the simulation data for true in-system verification of your design.

- **Creating a PROM File for Stand-Alone Operation:** After verifying that the FPGA works in the circuit, you can create a PROM file from the .bit file to program a PROM or other data storage device. You can then use this file to program the FPGA in-circuit during normal operation. The Prom File Formatter is used to create the PROM file.

DETAILED DESCRIPTION

SYSTEM BLOCK DIAGRAM



The experiment is started with the help of a trigger pulse form the detector system which also provides the pulses that need to be counted. The trigger is used to kick start the oscillators. We need special oscillators that can start at an instant of pulse edge. Since there are no such oscillators available in the market we need to have custom designed oscillators. As these oscillators are not that much reliable and there operation can be fully realized only when fabricated hence we have kept two spare oscillators. The three oscillators being used are a crystal oscillator, Colpitts oscillator and a digital oscillator. These pulses enter the FPGA which contains all the counters and other processing circuit.

We can pre-select the bin-time from a PC using the ISA interface. After each bin-time the counting moves on to the next counter and the value in the previous counter is first written onto the corresponding FIFO and then cleared for the next count. The experiment can stop in two ways:

1) The input from the detector stops and so does the clock input to the FPGA. Then the counts stored in the FIFOs are processed and the required information gathered.

2) In the other case if the experiment goes on for a long time then the FIFO will get full and thus asserting FIFO-full signal which is given to the FPGA and thus the logic inside stops he counting., also the PC is interrupted using IRQ5 line an the message is displayed to the user.

In the software model there is a provision for selecting different architectures as will be explained later so that experiments with small and large bin-times can be carried out successfully.

The data form the FIFO can be read directly through the ISA interfaceas shown in the diagram.

Hardware Description

Input Conditioning Circuit:-

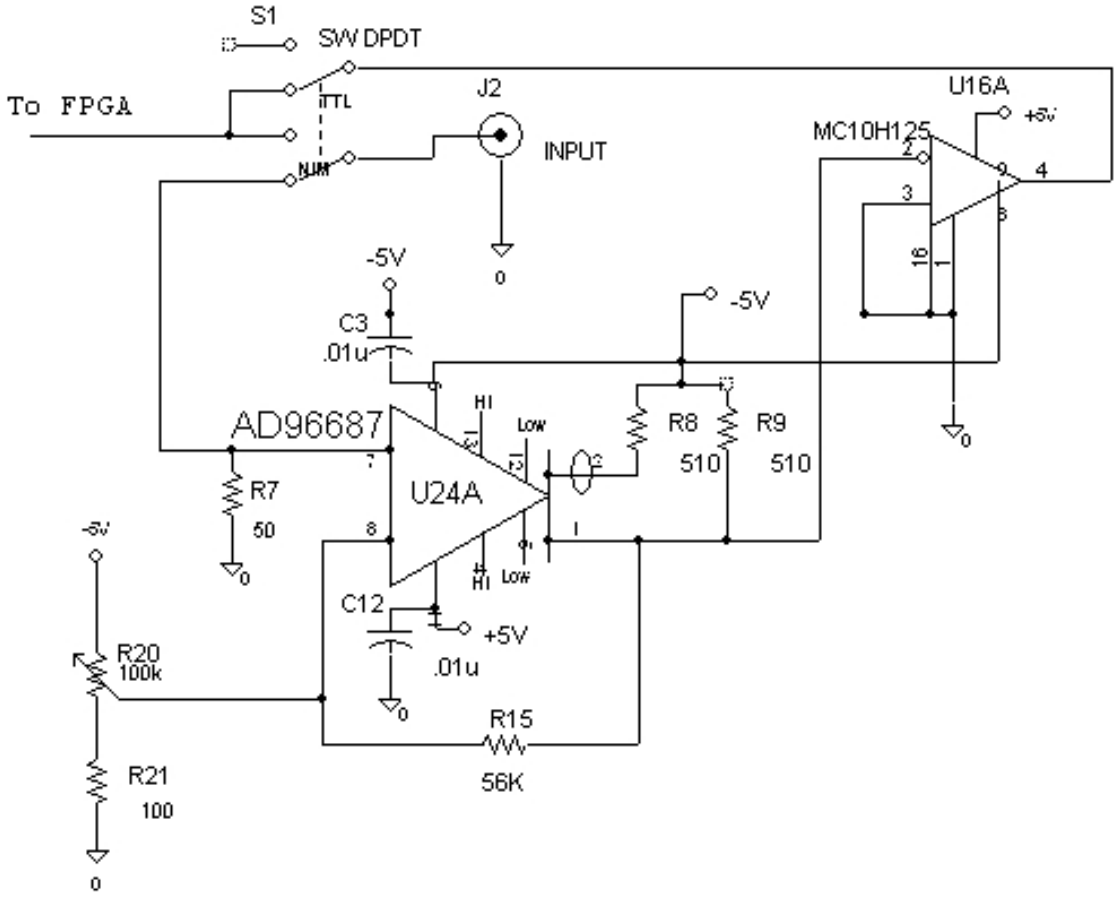


Fig No.1: Input Conditioning Circuit

The input is given to the card through a BNC connector and a Double Pole Double Throw (DPDT) which is given to a global clock pin on the Spartan. The input pin of Spartan in TTL compatible, whereas the input to the system can be NIM or TTL. If the input is TTL, then the jumper directly shorts it to the FPGA pin. However, if it is NIM compatible then it is preprocessed. The input is passed through a comparator so that output is ECL logic. The ECL to TTL converter MC10125 is then used to convert ECL input to TTL logic which is passed to the Spartan.

AD96687 is an ultra-fast comparator which is operated in “compare” mode. Both power supply connections should be separately decoupled to ground through 0.01 µF ceramic and 0.001 µF mica capacitors. The basic design of comparator circuits makes the negative supply somewhat more sensitive to variations. As a result, more attention should be placed on ensuring a “clean” negative supply. The LATCH ENABLE input of the AD96687 should be tied to –2.0 V or left “floating,” to disable the latching function. The best performance will be achieved with the use of proper ECL terminations. The open-emitter outputs of the AD96685 / AD96687 are designed to be terminated through 50 • resistors to –2.0 V, or any other equivalent ECL termination.

ECL to TTL Translator MC10125

The MC10125 is a quad translator for interfacing data and control signals between the MECL section and saturated logic sections of digital systems. The MC10125 incorporates differential inputs and Schottky TTL “totem pole” outputs. Differential inputs allow for use as an inverting/ non–inverting translator or as a differential line receiver. The VBB reference voltage is available on pin 1 for use in single–ended input biasing. The outputs of the MC10125 go to a low logic level whenever the inputs are left floating.

Gated Digital Oscillator (up to 10 MHz):-

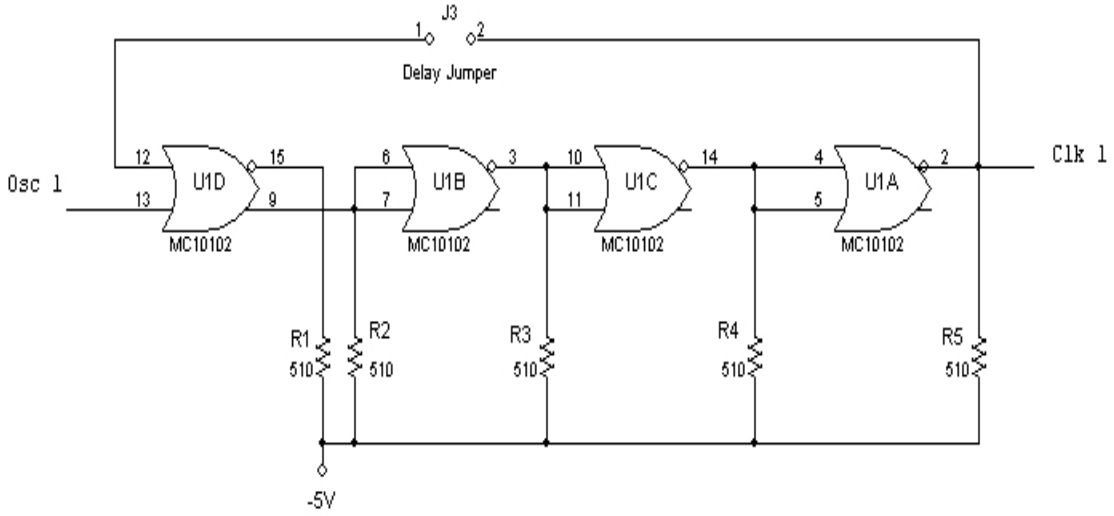


Figure No. 2 – Digital Oscillator

The digital oscillator works on the principle of propagation delays provided by NOR gates implemented as Inverters. The input to first NOR gate acts as an Inhibit or Gating signal for the oscillator. When input to first gate is at logic 0, then output is fixed at one level. However, when input is switched to logic 1 the output starts oscillating. In other words the first gate is inhibited/enabled using the OSC1 signal from the trigger conditioning circuit.

The maximum theoretical frequency can be obtained by simply adding the gate delays of all the NOR gates with the delay jumper shorted. This frequency comes out to,

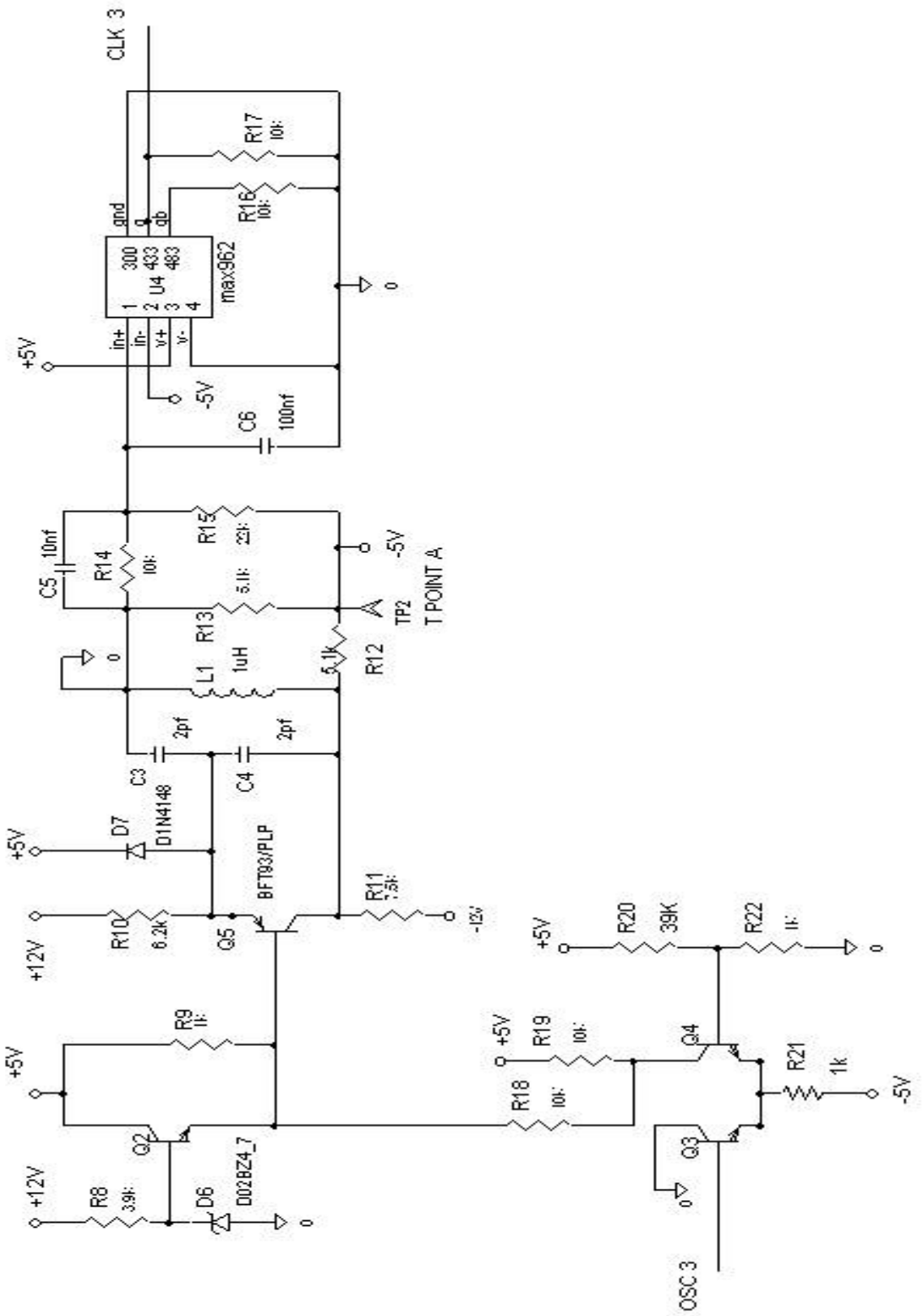
F_{max} = 1 / {N * (Delay of each gate) + Jumper Delay}

For MC10102, Delay = 6.7 ns

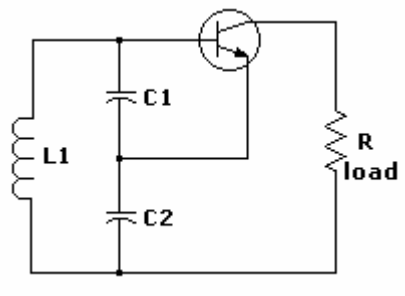
So, F_{max} = 1 / {4 * 6.7 ns + 0ns}
= 1 / {26.8 ns}
Maximum Theoretical Frequency = 37 MHz (approx)

But, practically the oscillator works well up to 10 MHz

Gated Colpitt's Oscillator (up to 100 MHz):-



The basic Colpitts oscillator circuit look like this:.



schematic of a collpitts oscillator

If you consider positive feedback is applied to compensate for the losses in the tuned circuit, the amplifier and feedback circuit create a negative resistor. When Z1 and Z2 are capacitive, the impedance across the capacitors can be estimated from a formula I won't lay on you here because it includesbeta, hie, as well as X_{C1} and X_{C2}. Suffice to sayit can be shown that the input impedance is a negative resistor in series with C1 and C2. And the frequency is in accordance with:

$$fo = \frac{1}{2\pi [LC_1C_2/(C_1 + C_2)]^{1/2}}$$

Reducing Phase Noise:

- Maximize the Q of the resonator.
- Maximize reactive energy by means of a high RF voltage across the resonator.
- Use a low LC ratio.
- Avoid device saturation and try to use anti parallel (back to back) tuning diodes.
- Choose your active device with the lowest NF.
- Choose a device with low flicker noise, this can be reduced by RF feedback. A bipolar transistor with an unby-passed emitter resistor of 10 to 30 ohms can improve flicker noise by as much as 40 dB.
- The output circuits should be isolated from the oscillator circuit and take as little power as possible.

Gated Monostable Multivibrator Oscillator (up to 10 MHz):-

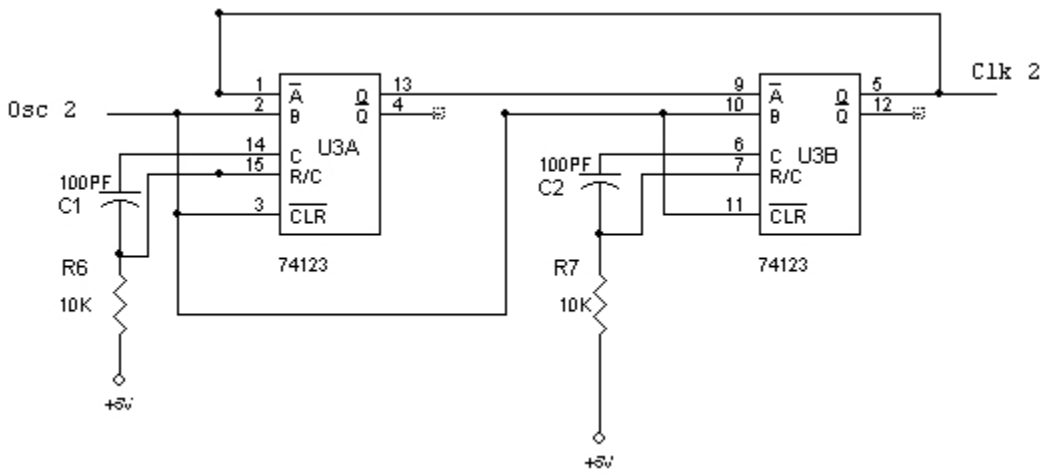


Fig no. 3 – Monostable Multivibrator Oscillator using IC 74123

This Oscillator consists of two monostable multivibrator timers connected in a loop such that the each one triggers the other. The 74123 is a dual retriggerable monostable multivibrator capable of generating output pulses from a few nano-seconds to extremely long duration up to 100% duty cycle. Each device has three inputs permitting the choice of either leading edge or trailing edge triggering. Pin (A) is an active low transition trigger input and pin (B) is an active-high transition trigger input. A low at the clear (CLR) input terminates the output pulse: which also inhibits triggering. An internal connection from CLR to the input gate makes it possible to trigger the circuit by a positive-going signal on CLR as shown in the truth table.

Triggering Truth Table

Inputs			Response
A	B	CLR	
X	X	L	No Trigger
↘	L	X	No Trigger
↘	H	H	Trigger
H	↗	X	No Trigger
L	↗	H	Trigger
L	H	↗	Trigger

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

Fig no. 4 – IC 74123 truth table

Both the monostable multivibrator timers of IC 74123 are used in the third mode in the truth-table (A = ⌝, B = High and CLR = High). The gate input when high makes B and CLR pins high. So, only a high to low transition pulse is required to trigger the monostable multivibrator. An external resistor (R) and external capacitor (C) are required for proper operation. The value of C may vary from 0 to any necessary value.

The output pulse width (TW) for C l 1000 pF is defined as follows:

$T_W = 0.28 * R * C * (1 + 0.7/R)$

So, to reduce the width of the pulse we require low values of external resistor and capacitor (R and C) respectively. So, practically we depend only on the parasitic capacitor for the C external giving oscillations up to 10 MHz.

Crystal Oscillator (100 MHz):-

Fig no. 5 – Crystal Oscillator

The crystal oscillator is the most stable high frequency oscillator. It gives highly stable oscillations in a narrow band of frequencies. But, gating a crystal oscillator is difficult due to the time it takes to start oscillations on receiving a gate pulse. So the oscillator output is taken as an input to Spartan and the gating is done by a simple AND operation.

The presence of four oscillators onboard is to introduce redundancy in the oscillator block. So, even if one oscillator fails the board testing can go on using the other working oscillators. The presence of lower frequency oscillators simply help in testing of the system.

Trigger Conditioning Circuit (fig 6):-

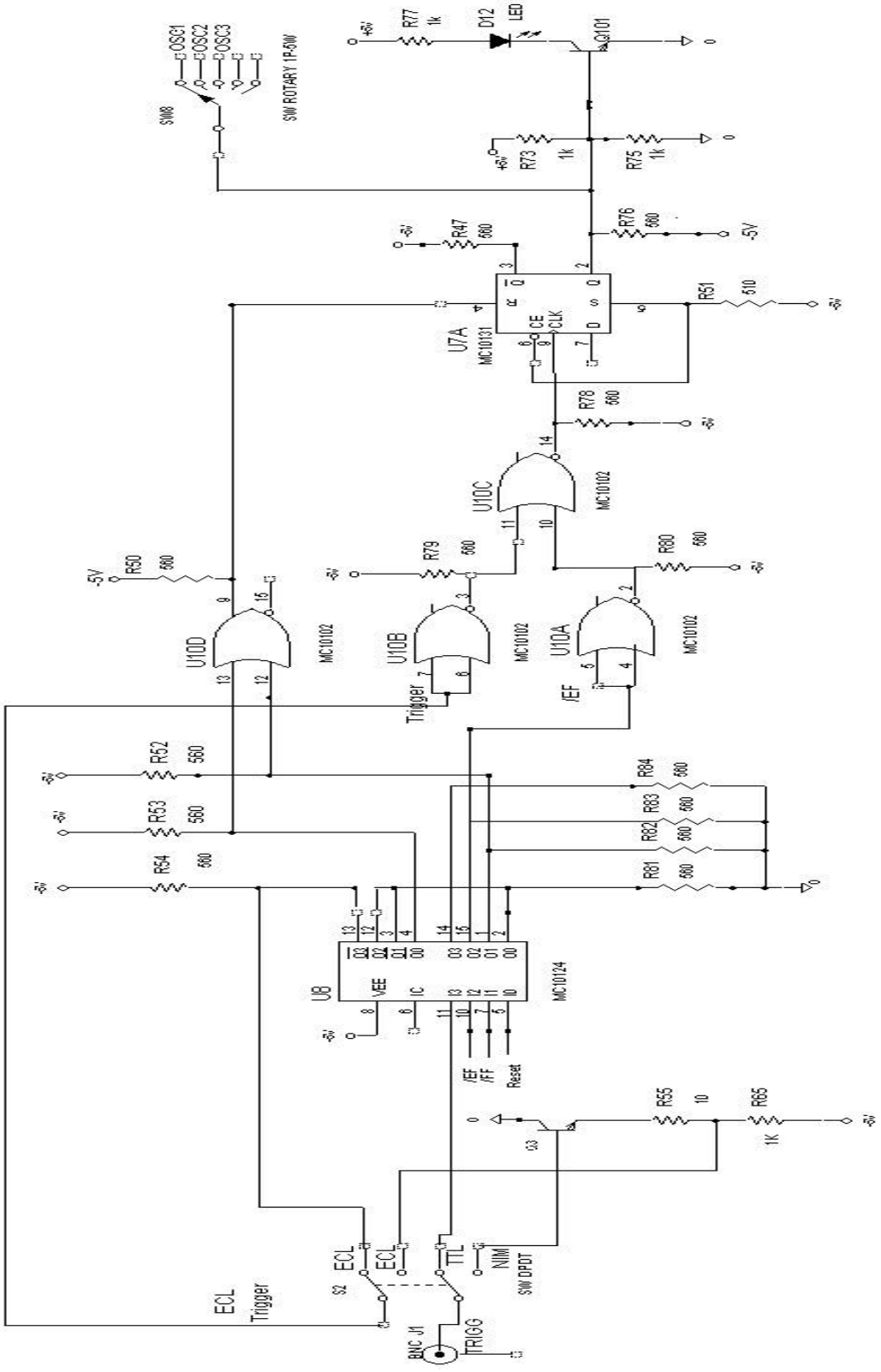


Figure 6 shows the trigger conditioning circuitry. The trigger logic is responsible for latching the input trigger provided the input conditions are satisfied. The latched output is responsible for driving the gated oscillator. The oscillators start running as long as the output of the latch is high. The latch is reset when the FIFO is full or the system is reset.
The Logic Equations for the Logic circuit are:-

1. $CLK = Trigger.\ \overline{EF}$

2. $R = Reset + \overline{FF}$

The Latch and Gates used for the logic circuit are required to be fast. So, we use ECL compatible gates to improve on speed. However, the FF, EF, Reset and Trigger inputs to the logic is TTL compatible. So, a TTL to ECL converter MC 10124 is used. The MC10124 has TTL compatible inputs, and ECL complementary open–emitter outputs that allow use as an inverting/ non–inverting translator or as a differential line driver. Outputs are terminated through a 500–ohm resistor to –5 volts.
The Latched output is used to drive a LED so as to indicate when the gated oscillator is given a high gate pulse. The MC10131 is a dual master–slave type D flip–flop. Asynchronous Set (S) and Reset (R) override Clock (CC) and Clock Enable (CE) inputs. Each flip–flop may be clocked separately by holding the common clock in the low state and using the enable inputs for the clocking function. If the common clock is to be used to clock the flip–flop, the Clock Enable inputs must be in the low state. In this case, the enable inputs perform the function of controlling the common clock. The output states of the flip–flop change on the positive transition of the clock.
The trigger input can be TTL or NIM compatible while the logic circuit is ECL compatible. So, additional NIM to ECL converter logic is also included. This ECL trigger can then be used for further processing.

Software model:

The MCS is programmed into a FPGA using Verilog HDL. There are three major units in the MCS

1. Counters for counting the pulses.
2. Gate signal generation for counters.
3. Write-read signal generation for FIFOs that are external to the FPGA.

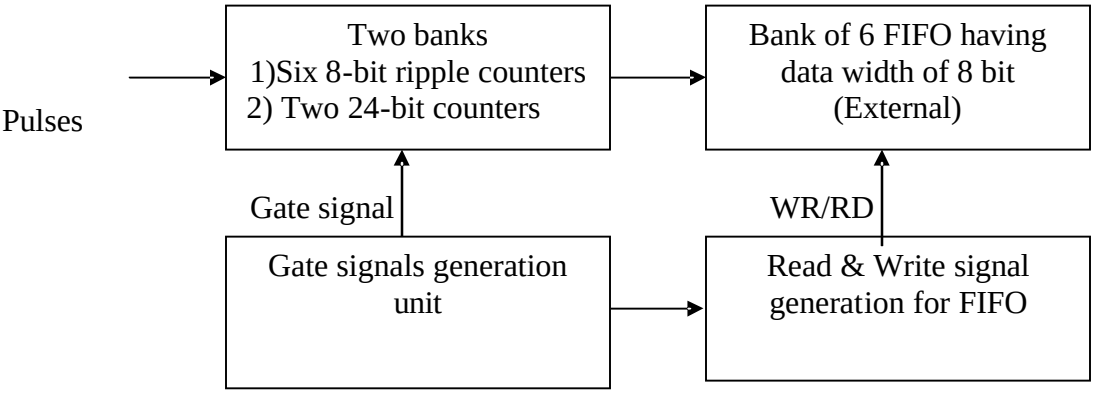
When a scan is started, the MCS begins counting input events or pulses in the first channel of its digital memory. At the end of the pre-selected dwell time the MCS advances to the next channel of memory to count the events.

There are two Architectures:

1. First with six 8-bit binary counters.
2. Second with two 24-bit binary counters.

Both are ripple counters. The switching between them is done depending upon gate signal length keeping in mind that 8-bit doesn't get full while counting the pulses. Signal Archsel and 8 X 2:1 Mux are used for swapping the architecture.

MCS Software Block diagram (fig a)



There are six 8-bit counters for counting the input pulses, which are gated by gate signal from Gate generating unit one by one. All the counters are provided power on reset. After gate signal is de-asserted counting stops and the counts are stored in FIFO. When gate signal of first counter is de-asserted then the gate signal of next means second counter is asserted. Like this all six counters will count the pulses one by one as and when there gate signals are asserted. Then each counter is reseted on gate signal of its previous counter (till this time it holds its count) and it is ready to count for next cycle.

Reset Logic:

Counter number	Reset signal
0	g5
1	g0
2	g1
3	g2
4	g3
5	g4

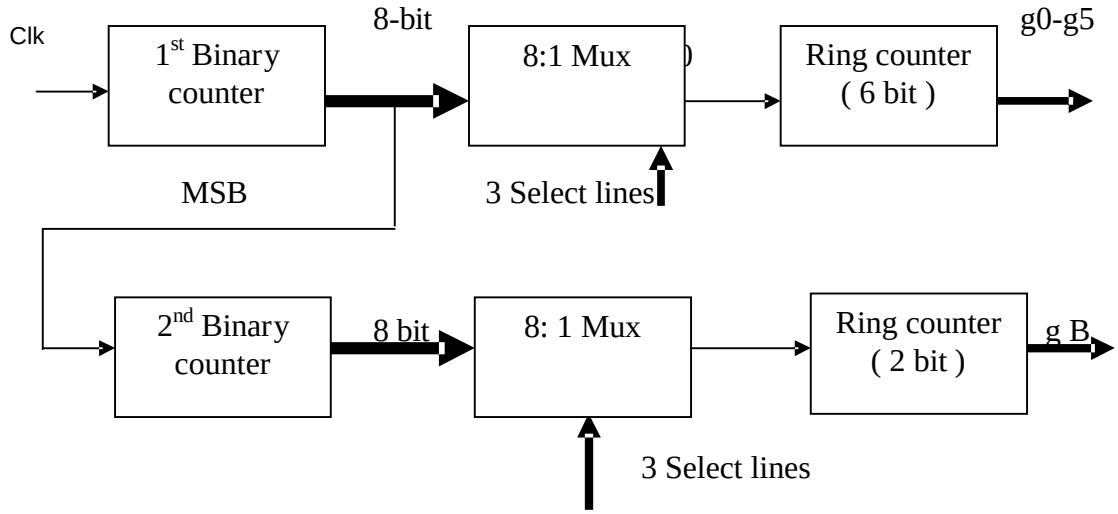
There are two 24-bit counters to count input pulses. This also works same as first architecture.

Reset logic:

Counter number	Reset signal
A	Gate B and bit 6 of 1st binary counter of Gate generating signal
B	Gate A and bit 6 of 1st binary counter of Gate generating signal

Gate signals are of 20 * (2^N) multiple are generated. The circuit consists of 8-bit binary counter, 8 to 1 MUX , Ring counter. The outputs of all f/f of counter are given to

Mux. Then Mux chooses the input depending upon required gate signal length. Output of Mux is given as clock to Ring counter. Output of ring counter is gate signal for counters, which are going to count the pulses.



Write signals:

8 bit Counter	Gate signal on which write signal generated
0	g1
1	g2
2	g3
3	g4
4	g5
5	g0

24 bit counter	Write signal
A	Gate B and bit 3 of 1st binary counter of Gate generating signal
B	Gate A and bit 3 of 1st binary counter of Gate generating signal

Implementation

Hardware Circuit:-

After the board is designed it is given for pcb layout & manufacturing. After the pcb is manufactured the components are soldered onto the board. The board contains the signal conditioning circuits, oscillators, FIFOs, voltage regulators, ports, test points and the Xilinx SPARTAN-2. The working of the individual blocks has already been explained before, now we shall look at the overall working of the system.

The voltage regulators provided on board give +2.5V & +3.3V outputs with 12V input from he ISA slot. These voltage levels are required for the Xilinx SPARTAN-2. The unused I/O pins of the FPGA are brought out in the form of jumpers for future use as well as for debugging purpose. A number of on board switches determine the mode and choice of oscillator, etc.

Testing stage: -

First the board is slotted onto a free ISA slot on the motherboard. During the testing stage as the programs are not finalized we use slave serial programming using JTAG cable. JTAG, an acronym for Joint Test Action Group, is the usual name used for the IEEE 1149.1 standard for Test Access Port and Boundary Scan, primarily used for testing integrated circuits, but also useful as a mechanism for debugging embedded systems.
A JTAG interface is a special four-pin (data in, data out, TCK, TMS) interface added to a chip, designed so that multiple chips on a board can have their JTAG lines daisy-chained together, and a test probe need only connect to a single "JTAG port" to have access to all chips on a circuit board. Since only the one data line is available, the protocol is necessarily serial. The clock input is at the TCK pin. Configuration is performed by manipulating a state machine one bit at a time through a TMS pin. One bit of data is transferred in and out per TCK clock pulse at the TDI and TDO pins, respectively. Different instruction modes can be loaded to read the chip ID, sample input pins, drive

(or float) output pins, manipulate chip functions, or bypass (pipe TDI to TDO to logically shorten chains of multiple chips). The operating frequency of TCK varies depending on the chip, but it is typically 10-100MHz (10-100ns per bit). Once the FPGA is programmed we can start the experiment.

We have also provided for a EEPROM in a PLCC (*Plastic leaded chip carrier. Normally a four-sided quad package in which an IC is installed with J-type leads extending out from the sides of the package then downward rolled under the body of the device*) package so that onsite testing & debugging where the availability of a PC is not guaranteed is possible.

We use the Xilinx Webpack(version 4.2) for programming the FPGA. First a new design is created in the webpack where the right choice of the FPGA chip and other design parameters are made. The verilog files are then loaded & compiled for errors. When error free, we then write the ucf file (user constraint file) for mapping all the pins of the FPGA. Proper assignment has to be made for different categories of pins and special pins. If the pin assignment is proper then we go for Place & route of the design. This step performs the virtual routing of the hardware designed and determines if enough I/O ports, CLBs, etc are actually available for the design on the chip. It also determines the critical path and the maximum frequency of operation for the given design. If satisfied with the results of place & route we proceed to generate the Device Programming file after selecting he mode from boundary scan, slave-serial & parallel. A .bit file is generated which is to be downloaded to the FPGA for configuring the device.

Experiment Run: -

Once our programs are finalized it is burned on to the one time programmable ROM (OTP-ROM) which sits on the main board. When the FPGA is powered on the FPGA can be programmed through this OTP ROM. Once the FPGA is programmed we can proceed with configuring the experiment. By using the ISA interfacing we can select one of the two architectures as well as the desired frequency of operation as explained in the software model. Now we are ready to start the experiment.

The experiment starts when a trigger is received by the trigger conditioning circuit. This trigger then is used to start any of the three oscillators by the particular combination of on board switches. The pulses generated by the oscillators are first converted into TTL format if already not so and then input to the SPARTAN at one of its global clock pins. At the same time the input pulses that need to be counted arrive at the input pin after being conditioned to TTL logic by the input conditioning circuit at the input pin which is again another global clock pin of the SPARTAN. The input & clock pulses have to be fed to global clock pins as they are used at numerous CLBs inside the SPARTAN and hence need to be routed properly without much delay and skew.

The counting is performed by the software model as explained elsewhere. The counted values, momentarily stored in the counters, are stored into the FIFOs as per the frequency of operation decided before the start of the experiment through the ISA interface by the user. When the FIFOs are full then the fifo full signal of the last FIFO is used as a interrupt to the experiment to stop the counting process further. This interrupt is further passed onto the user through the IRQ5/ IRQ3 interrupt line of the ISA interface. This interrupt prompts the user to start reading the counts stored in the FIFOs. The user then can read all the values from the FIFOS. Once the last FIFO is empty, the fifo empty line is used as an interrupt to the system to tell that the experiment can again start counting i.e. the system is reset. .

Software Implementation:-

Verilog System Implementation Modules (Top-Down):-

MCS:

It is the top level of the design and provides the interfacing for the lower modules. It takes care of ISA interface, FIFO interface, control signals for FIFO etc. This module calls the lower modules of ipblock and trigger for providing the functions requested by the MCS module.

```
module mcs(rst,addr,ior,iow,clk,fifo_f,fifo_e,wr,rd,data1,inv_rst,irq3,irq5);
        input rst;
        input [15:0] addr;
        input ior,iow;
        input clk;
        input fifo_f,fifo_e;

        output [5:0] wr;
        output [5:0] rd;
        output [7:0] data1;
        output inv_rst;
        output irq3,irq5;

        wire [7:0] rd1,data,d111,d2,d3,d4,d5,d6;
        wire ce1,lat,archsel,rrst;
        wire [5:0] q;
        wire [47:0] data2;
        wire [2:0] fsel;
        wire [7:0] data3;
        wire [1:0] arch2;
        wire [5:0] arch1;
        wire ip;
```

```
        nor o4 (rrst, iow, addr[15], addr[14], addr[13], addr[12], addr[11], addr[10],
        ~addr[9], ~addr[8], addr[7], ~addr[6], ~addr[5], ~addr[4], ~addr[3], ~addr[2],
        ~addr[1], ~addr[0]);

        nor o1 (ce1, addr[15], addr[14], addr[13], addr[12], addr[11], addr[10], ~addr[9],
        ~addr[8], addr[7], addr[6], addr[5], addr[4], addr[3]);

        or a6 (rst1,rst,rrst);
        not n1(inv_rst,rst1);
        decoder3_8 d1(ce1,addr[2:0],ior,rd1);
        count8 c1(clk,rst1,ce1,data[7:0]);

        //address logic

        nor o2 (lat, addr[15], addr[14], addr[13], addr[12], addr[11], addr[10], ~addr[9],
        ~addr[8], ~addr[7]);

        assign archsel=data3[3];
        assign fsel[2:0]=data3[2:0];

        wire six,three,out;

        trigger tr (ce1,clk,rst1,fsel,arch1,arch2,six,three,out);

        ipblock ip1 (clk,ce1,arch1,arch2,archsel,rst1,three,six,data1,wr,out);

        assign irq3 = fifo_e;
        assign irq5 = fifo_f;
        assign rd[5:0] = ~rd1[5:0];
endmodule
```

Ipblock:

The module provides two architecture of

i. Six 8-bit counters
ii. Two 24-bit counters

The selection between the two architectures depends upon the architecture select input signals and gives a byte-wide output bus with the appropriate write signals.

```
module ipblock ( ip,ce,arch1,arch2,archsel,clr,three,six,op,wr,out);
        //Definitions of inputs
        input ip,ce,clr,archsel,three,six,out;
        input [5:0] arch1;
        input [1:0] arch2;

        //Definitions of outputs
        output [7:0] op;
        output [5:0] wr;

        //Definitions of wires
        wire [47:0] q1,q2,q3;
        wire [7:0] w;
        wire [5:0] wr1,wr2;
        wire archsel11;
        wire [5:0] www1,www2;

        // Logic for implementation of IP block
        and     a1(w[0],ip,arch1[0]);
        and     a2(w[1],ip,arch1[1]);
        and     a3(w[2],ip,arch1[2]);
        and     a4(w[3],ip,arch1[3]);
        and     a5(w[4],ip,arch1[4]);
        and     a6(w[5],ip,arch1[5]);
        and     a7(w[6],ip,arch2[0]);
```

```
and      a8(w[7],ip,arch2[1]);

//architecture 1
count8 c1(w[0],clr|arch1[5],ce,q1[7:0]);
count8 c2(w[1],clr|arch1[0],ce,q1[15:8]);
count8 c3(w[2],clr|arch1[1],ce,q1[23:16]);
count8 c4(w[3],clr|arch1[2],ce,q1[31:24]);
count8 c5(w[4],clr|arch1[3],ce,q1[39:32]);
count8 c6(w[5],clr|arch1[4],ce,q1[47:40]);

//architecture 2
count24 c7(w[6],clr|(six & arch2[1]),ce,q2[23:0]);
count24 c8(w[7],clr|(six & arch2[0]),ce,q2[47:24]);


//multiplexers
mux8_2_1 m1(q1[7:0],q2[7:0],archsel,q3[7:0]);
mux8_2_1 m2(q1[15:8],q2[15:8],archsel,q3[15:8]);
mux8_2_1 m3(q1[23:16],q2[23:16],archsel,q3[23:16]);
mux8_2_1 m4(q1[31:24],q2[31:24],archsel,q3[31:24]);
mux8_2_1 m5(q1[39:32],q2[39:32],archsel,q3[39:32]);
mux8_2_1 m6(q1[47:40],q2[47:40],archsel,q3[47:40]);

//write pulses
not n1(archsel11,archsel);

and a9(wr1[0],arch1[2],archsel11);
and a10(wr1[1],arch1[3],archsel11);
and a11(wr1[2],arch1[4],archsel11);
and a12(wr1[3],arch1[5],archsel11);
and a13(wr1[4],arch1[0],archsel11);
and a14(wr1[5],arch1[1],archsel11);
```

```
    or r1(wr[0],wr1[0],wr2[0]);
    or r2(wr[1],wr1[1],wr2[1]);
    or r3(wr[2],wr1[2],wr2[2]);
    or r4(wr[3],wr1[3],wr2[3]);
    or r5(wr[4],wr1[4],wr2[4]);
    or r6(wr[5],wr1[5],wr2[5]);

    ring6_w rr1 (out,clr,three,www1);
    ring6_w rr2 (out,clr,six,www2);

    mux8_8_1 m (wr,q3[7:0],q3[15:8],q3[23:16],q3[31:24],q3[39:32],q3[47:40],op);

    assign wr2[5]=www2[5];
    assign wr2[4]=www2[4];
    assign wr2[3]=www2[3];
    assign wr2[2]=www1[5];
    assign wr2[1]=www1[4];
    assign wr2[0]=www1[3];

endmodule
```

Trigger:

The trigger module models the gate generation block diagram shown in fig b of the software model of MCS described in the previous section. It consists of counters used as frequency dividers and the appropriate frequency is selected depending on the fsel input.

```
module trigger (ce,clk,clr,fsel,arch1,arch2,six,three,out);
    //Definitions of inputs
    input clk,clr,ce;
    input  [2:0] fsel ;
```

```
//Definitions of outputs
output [5:0] arch1;
output [1:0] arch2;
output six,three,out;

//Definitions of wire
wire [7:0] op1,op3;
wire op2,op4;//op5;
wire [5:0] out1,int1,int2;
wire [1:0] out2;

//Logic for implementation of Trigger block
count8 c1(clk,clr,ce,op1);
mux8_1 m1(op1,fsel,op2);
ring6 r1(op2,clr,ce,out1);

count8 c2(op1[7],clr,ce,op3);
mux8_1 m2(op3,fsel,op4);
ring2 r2(op4,clr,ce,out2);

seq s1(out2[1],clk,clr,ce,int2[1],int2[5]);
seq s2(out2[0],clk,clr,ce,int1[1],int1[5]);

or o1 (six,int1[5],int2[5]);
or o2 (three,int1[1],int2[1]);

assign arch1 = out1;
assign arch2 = out2;
assign out=op1[4];

endmodule
```

Count8:

The module provides a 8-bit asynchronous counter with clear and chip enable input. It is used in Ipblock as a counter and in Trigger block as a frequency divider.

```
module count8 (clk,clr,ce,Q);
        input clk,clr,ce;
        output [7:0] Q;

        wire [7:0] w;

        //   Instantiations for dffs to generate 8 bit counter
        dffs d1(clr,ce,~w[0],w[0],clk);
        dffs d2(clr,ce,~w[1],w[1],w[0]);
        dffs d3(clr,ce,~w[2],w[2],w[1]);
        dffs d4(clr,ce,~w[3],w[3],w[2]);
        dffs d5(clr,ce,~w[4],w[4],w[3]);
        dffs d6(clr,ce,~w[5],w[5],w[4]);
        dffs d7(clr,ce,~w[6],w[6],w[5]);
        dffs d8(clr,ce,~w[7],w[7],w[6]);

        assign Q = ~w;

endmodule
```

Decoder3_8:

This module models the functioning of a 3:8 Decoder using the actual And-Or gate combination. The 3-bit wide addr signal is decoded into rd signal with ior as an enabling input.

```
module decoder3_8(ce,addr,ior,rd);

        input  [2:0] addr;
```

```
    input ce,ior;

    output [7:0] rd;

    wire [2:0] addr1;
    wire ior1;

    not n1(addr1[2],addr[2]);
    not n2(addr1[1],addr[1]);
    not n3(addr1[0],addr[0]);
    not n4(ior1,ior);

    and a0(rd[0],addr1[2],addr1[1],addr1[0],ce,ior1);
    and a1(rd[1],addr1[2],addr1[1],addr[0],ce,ior1);
    and a2(rd[2],addr1[2],addr[1],addr1[0],ce,ior1);
    and a3(rd[3],addr1[2],addr[1],addr[0],ce,ior1);
    and a4(rd[4],addr[2],addr1[1],addr1[0],ce,ior1);
    and a5(rd[5],addr[2],addr1[1],addr[0],ce,ior1);
    and a6(rd[6],addr[2],addr[1],addr1[0],ce,ior1);
    and a7(rd[7],addr[2],addr[1],addr[0],ce,ior1);

endmodule
```

Count24:

The module provides a 24-bit asynchronous counter with clear and chip enable input. It is used in Ipblock as a counter.

```
module count24 (clk,clr,ce,Q);
    //defining inputs
    input clk,clr,ce;
    //defining outputs
    output [23:0] Q;
```

```
//defining wires
wire [23:0] w;

//Instantiations of  24 Dffs,each for 1 bit to create 24 bit counter
dffs d1 (clr,ce,~w[0],w[0],clk);
dffs d2 (clr,ce,~w[1],w[1],w[0]);
dffs d3 (clr,ce,~w[2],w[2],w[1]);
dffs d4 (clr,ce,~w[3],w[3],w[2]);
dffs d5 (clr,ce,~w[4],w[4],w[3]);
dffs d6 (clr,ce,~w[5],w[5],w[4]);
dffs d7 (clr,ce,~w[6],w[6],w[5]);
dffs d8 (clr,ce,~w[7],w[7],w[6]);
dffs d9 (clr,ce,~w[8],w[8],w[7]);
dffs d10(clr,ce,~w[9],w[9],w[8]);
dffs d11(clr,ce,~w[10],w[10],w[9]);
dffs d12(clr,ce,~w[11],w[11],w[10]);
dffs d13(clr,ce,~w[12],w[12],w[11]);
dffs d14(clr,ce,~w[13],w[13],w[12]);
dffs d15(clr,ce,~w[14],w[14],w[13]);
dffs d16(clr,ce,~w[15],w[15],w[14]);
dffs d17(clr,ce,~w[16],w[16],w[15]);
dffs d18(clr,ce,~w[17],w[17],w[16]);
dffs d19(clr,ce,~w[18],w[18],w[17]);
dffs d20(clr,ce,~w[19],w[19],w[18]);
dffs d21(clr,ce,~w[20],w[20],w[19]);
dffs d22(clr,ce,~w[21],w[21],w[20]);
dffs d23(clr,ce,~w[22],w[22],w[21]);
dffs d24(clr,ce,~w[23],w[23],w[22]);

assign Q = ~w;

endmodule
```

Mux8_2_1:

This module is a combination of eight 2:1 multiplexers in parallel. It uses an underlying mux2_1 module to obtain the desired functionality.

```
module mux8_2_1 (i1,i2,sel,op);
        //Definitions of  inputs
        input [7:0]i1,i2;
        input sel;
        //Definitions of output
        output [7:0] op;
        //Definitions of wire
        wire [7:0] o;

        //Logic for 8 Multiplexer 2:1,each input being 8 bits
        // 8 Instantiations of Multiplexer 2:1 have been used
        mux2_1 m0 (i1[0],i2[0],sel,o[0]);
        mux2_1 m1 (i1[1],i2[1],sel,o[1]);
        mux2_1 m2 (i1[2],i2[2],sel,o[2]);
        mux2_1 m3 (i1[3],i2[3],sel,o[3]);
        mux2_1 m4 (i1[4],i2[4],sel,o[4]);
        mux2_1 m5 (i1[5],i2[5],sel,o[5]);
        mux2_1 m6 (i1[6],i2[6],sel,o[6]);
        mux2_1 m7 (i1[7],i2[7],sel,o[7]);

        assign op = o;

endmodule
```

Mux2_1:

This module is a simple 2:1 multiplexer designed using And-Or gate combination.

```
module mux2_1 (i1,i2,sel,op);
        //Definitions for input
        input i1,i2,sel;
        //Definitions for output
        output op;

        //Definitions for wire
        wire o1,o2,sel1,op1;

        //Logic of Multiplexer 2:1
        not n1(sel1,sel);
        and a1(o1,i1,sel1);
        and a2(o2,i2,sel);
        or  r1(op1,o1,o2);

        assign op = op1;

endmodule
```

Mux8_8_1:

This module closely resemble a multiplexer operation but has slightly different functionality. The en signal is 6-bits wide and used to enable one of the eight input signals each 8-bits wide.

```
module mux8_8_1(en,ip1,ip2,ip3,ip4,ip5,ip6,op);

        input [7:0] ip1,ip2,ip3,ip4,ip5,ip6;
        input [5:0] en;

        output [7:0] op;

        wire [7:0] op1,op2,op3,op4,op5,op6;
```

```
and aa10(op1[0],ip1[0],en[0]);
and aa20(op1[1],ip1[1],en[0]);
and aa30(op1[2],ip1[2],en[0]);
and aa40(op1[3],ip1[3],en[0]);
and aa50(op1[4],ip1[4],en[0]);
and aa60(op1[5],ip1[5],en[0]);
and aa70(op1[6],ip6[6],en[0]);
and aa80(op1[7],ip6[7],en[0]);

and aa11(op2[0],ip2[0],en[1]);
and aa21(op2[1],ip2[1],en[1]);
and aa31(op2[2],ip2[2],en[1]);
and aa41(op2[3],ip2[3],en[1]);
and aa51(op2[4],ip2[4],en[1]);
and aa61(op2[5],ip2[5],en[1]);
and aa71(op2[6],ip6[6],en[1]);
and aa81(op2[7],ip6[7],en[1]);

and aa12(op3[0],ip3[0],en[2]);
and aa22(op3[1],ip3[1],en[2]);
and aa32(op3[2],ip3[2],en[2]);
and aa42(op3[3],ip3[3],en[2]);
and aa52(op3[4],ip3[4],en[2]);
and aa62(op3[5],ip3[5],en[2]);
and aa72(op3[6],ip6[6],en[2]);
and aa82(op3[7],ip6[7],en[2]);

and aa13(op4[0],ip4[0],en[3]);
and aa23(op4[1],ip4[1],en[3]);
and aa33(op4[2],ip4[2],en[3]);
and aa43(op4[3],ip4[3],en[3]);
```

```
and aa53(op4[4],ip4[4],en[3]);
and aa63(op4[5],ip4[5],en[3]);
and aa73(op4[6],ip6[6],en[3]);
and aa83(op4[7],ip6[7],en[3]);

and aa14(op5[0],ip5[0],en[4]);
and aa24(op5[1],ip5[1],en[4]);
and aa34(op5[2],ip5[2],en[4]);
and aa44(op5[3],ip5[3],en[4]);
and aa54(op5[4],ip5[4],en[4]);
and aa64(op5[5],ip5[5],en[4]);
and aa74(op5[6],ip6[6],en[4]);
and aa84(op5[7],ip6[7],en[4]);

and aa15(op6[0],ip6[0],en[5]);
and aa25(op6[1],ip6[1],en[5]);
and aa35(op6[2],ip6[2],en[5]);
and aa45(op6[3],ip6[3],en[5]);
and aa55(op6[4],ip6[4],en[5]);
and aa65(op6[5],ip6[5],en[5]);
and aa75(op6[6],ip6[6],en[5]);
and aa85(op6[7],ip6[7],en[5]);

or o1 (op[0],op1[7],op2[7],op3[7],op4[7],op5[7],op6[7]);
or o2 (op[1],op1[6],op2[6],op3[7],op4[7],op5[7],op6[7]);
or o3 (op[2],op1[5],op2[5],op3[7],op4[7],op5[7],op6[7]);
or o4 (op[3],op1[4],op2[4],op3[7],op4[7],op5[7],op6[7]);
or o5 (op[4],op1[3],op2[3],op3[7],op4[7],op5[7],op6[7]);
or o6 (op[5],op1[2],op2[2],op3[7],op4[7],op5[7],op6[7]);
or o7 (op[6],op1[1],op2[1],op3[7],op4[7],op5[7],op6[7]);
or o8 (op[7],op1[0],op2[0],op3[7],op4[7],op5[7],op6[7]);
```

```
endmodule
```

Ring6_w:

The module is a ring counter with the output of one Flip-Flop connected at input of the next. This ring is of 6 such Flip-Flops with the input to the first Flip-Flop being 0.

```
module ring6_w(clk,clr,ce,Q);

    input clk,clr,ce;
    output [5:0] Q;

    wire [5:0] w;

    // Instantiations of Dffs and Dffr are used to generate the model
    dffs d1 (clr,ce,1'b0,w[0],clk);
    dffr d2 (clr,ce,w[0],w[1],clk);
    dffr d3 (clr,ce,w[1],w[2],clk);
    dffr d4 (clr,ce,w[2],w[3],clk);
    dffr d5 (clr,ce,w[3],w[4],clk);
    dffr d6 (clr,ce,w[4],w[5],clk);

    assign Q=w;

endmodule
```

Mux8_1:

This module is a simple 8:1 multiplexer designed using And-Or gate combination.

```
module mux8_1 (ip,sel,op);
    //Defining the inputs
    input [7:0] ip;
    input [2:0] sel;
```

```
    //Defining the outpurs
    output op;

    //Defining the wires
    wire [7:0]o;
    wire [2:0] sel1;

    //Multiplexer Logic
    not a1(sel1[0],sel[0]);
    not a2(sel1[1],sel[1]);
    not a3(sel1[2],sel[2]);

    and n1(o[0],sel1[2],sel1[1],sel1[0],ip[0]);
    and n2(o[1],sel1[2],sel1[1],sel[0],ip[1]);
    and n3(o[2],sel1[2],sel[1],sel1[0],ip[2]);
    and n4(o[3],sel1[2],sel[1],sel[0],ip[3]);
    and n5(o[4],sel[2],sel1[1],sel1[0],ip[4]);
    and n6(o[5],sel[2],sel1[1],sel[0],ip[5]);
    and n7(o[6],sel[2],sel[1],sel1[0],ip[6]);
    and n8(o[7],sel[2],sel[1],sel[0],ip[7]);

    or o1 (op,o[0],o[1],o[2],o[3],o[4],o[5],o[6],o[7]);

endmodule
```

Ring2:

The module is a ring counter with the output of one Flip-Flop connected at input of the next. This ring is of 2 such Flip-Flops with the input to the first Flip-Flop being the output of the second.

```
module ring2(clk,clr,ce,Q);
    input clk,clr,ce;
```

```
        output [1:0] Q;

        wire [1:0] w;

        //    Using Instantiations of Dffs and Dffr
        dffs d1 (clr,ce,w[1],w[0],clk);
        dffr d2 (clr,ce,w[0],w[1],clk);

        //count8 c1 (clk,clr,ce,q);
        assign Q[0]=w[0];
        assign Q[1]=w[1];

endmodule
```

Ring6:

The module is a ring counter with the output of one Flip-Flop connected at input of the next. This ring is of 6 such Flip-Flops with the input to the first Flip-Flop being the output of the last Flip-Flop.

```
module ring6(clk,clr,ce,Q);
        input clk,clr,ce;
        output [5:0] Q;

        wire [5:0] w;
        //    Instantiations of Dffs and Dffr are used to generate the model
        dffs d1 (clr,ce,w[5],w[0],clk);
        dffr d2 (clr,ce,w[0],w[1],clk);
        dffr d3 (clr,ce,w[1],w[2],clk);
        dffr d4 (clr,ce,w[2],w[3],clk);
        dffr d5 (clr,ce,w[3],w[4],clk);
        dffr d6 (clr,ce,w[4],w[5],clk);
```

```
    assign Q=w;

endmodule
```

Seq:

The sequence generator is a sequential circuit designed to allow the 24-bit counters to be ready for the next input during the period the other counter is counting. It allows the counter to settle, saves the value of count in FIFO, resets the counter and gets it ready for the next operation.

```
module seq(x,clk,clr,ce,Z1,Z2);
    input x,clk,clr,ce;
    output Z1,Z2;

    wire d2,d1,d0,q2,q1,q0,q21,q11,q01;
    wire i1,i2,i3,i4,i5,i6,i7,i8,i9,i10;

    dffs g2(clr,ce,d2,q2,clk);
    dffs g1(clr,ce,d1,q1,clk);
    dffs g0(clr,ce,d0,q0,clk);

    not n2(q21,q2);
    not n1(q11,q1);
    not n0(q01,q0);

    and a1(i1,q21,q0);
    and a2(i2,q21,q1);
    and a3(i3,x,q2);
    and a4(i4,x,q1,q0);
    and a5(i5,x,q11,q0);
    and a6(i6,x,q2,q1);
    and a7(i7,x,q1,q01);
```

```
        and a8(i8,x,q01);
        and a9(i9,q2,q11,q0);
        and a10(i10,q2,q1,q01);

        or o1(Z1,i1,i2);
        or o2(d2,i3,i4);
        or o3(d1,i5,i6,i7);
        or o4(d0,i6,i8);
        or o5(Z2,i9,i10);

endmodule
```

Dffr:

The D-flip flop is a behaviorally coded module which models a D flip-flop with reset.

```
module dffr (clr , ce , D , Q , clk );
        input clr , ce , clk ;
        input D ;
        output Q ;

        reg Q_TEMP;
        initial
                begin
                        if(ce==1'b1)
                        Q_TEMP = 1'b0;
                end

        always @(posedge clr or posedge clk )
        begin
                if (clr == 1'b1)
                        Q_TEMP = 1'b0;
                else if (ce == 1'b1)
```

```
                Q_TEMP = D ;
        end

    assign Q = Q_TEMP;
endmodule
```

Dffs:

The D-flip flop is a behaviorally coded module which models a D flip-flop with set.

```
module dffs (clr , ce , D , Q , clk );
    input clr , ce , clk ;
    input D ;
    output Q ;

    reg Q_TEMP;
    initial
        if(ce==1'b1)
            Q_TEMP = 1'b1;

    always @(posedge clr or posedge clk )
    begin
        if (clr == 1'b1)
            Q_TEMP = 1'b1;
        else if (ce == 1'b1)
            Q_TEMP = D ;
    end

    assign Q = Q_TEMP;
endmodule
```

Application Program/User Interface (C Program):-

Mcs1.cpp

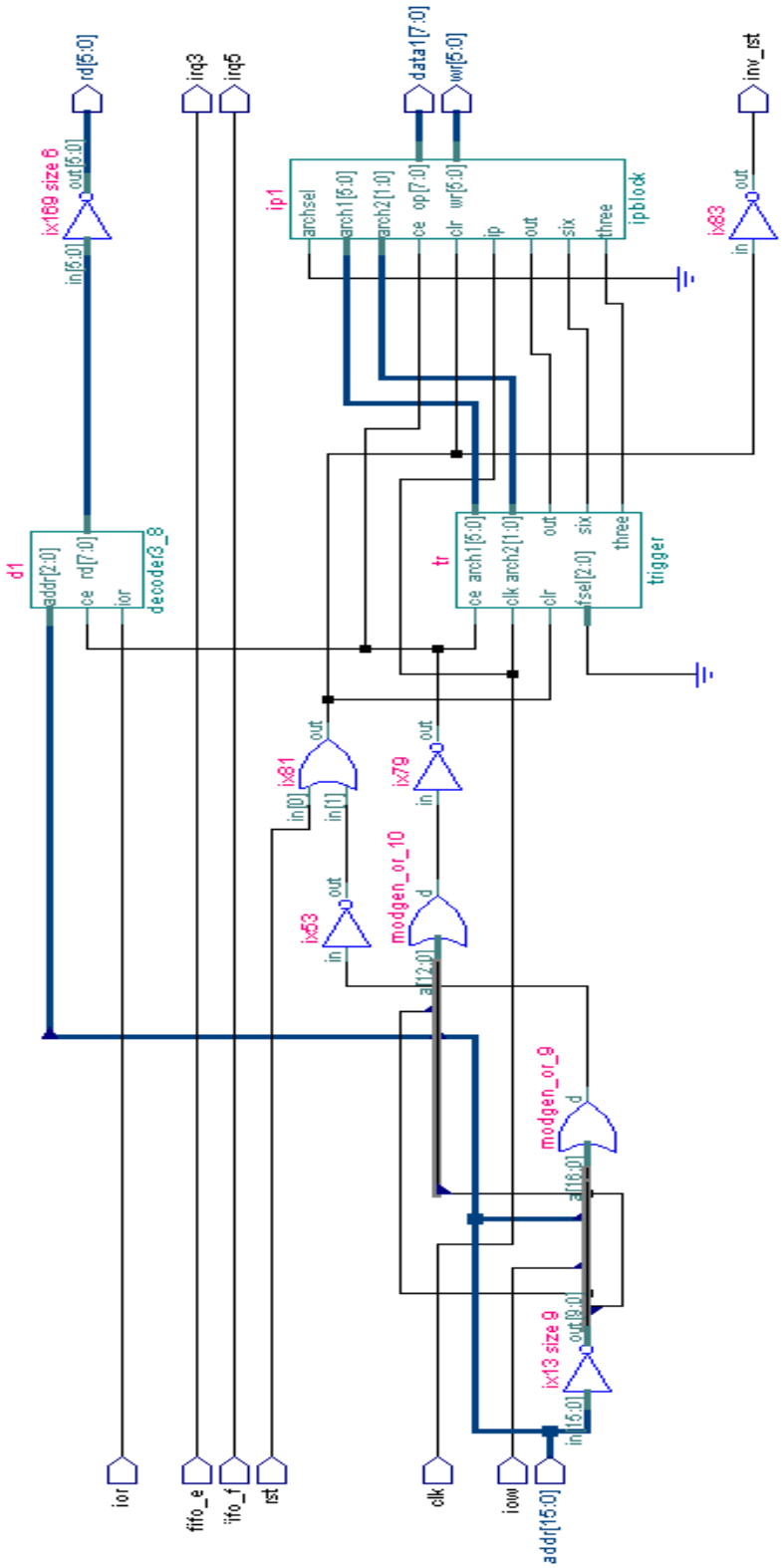
```
/* Start reading FIFO for INT3(ff_full) and end on INT5(ff_empty) */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INT3 0x13
#define INT5 0x15
#define FIFO_NO 8096

void interrupt ( *oldhandler)();

int terminate=0;

void interrupt handler5()
{
      Printf(“FIFO is Empty”);
}

void interrupt handler3()
{
  for(int j=0;j<FIFO_NO;j++)
  {
    ch=inportb(0x0300);
    printf("\nData read from FIFO 0 :  %002X \\",ch);
    ch=inportb(0x0301);
    printf("\nData read from FIFO 1 :  %002X | 1st 24 bit counter",ch);
    ch=inportb(0x0302);
    printf("\nData read from FIFO 2 :  %002X /",ch);
    ch=inportb(0x0303);
    printf("\nData read from FIFO 3 :  %002X \\",ch);
    ch=inportb(0x0304);
    printf("\nData read from FIFO 4 :  %002X | 2nd 24 bit counter",ch);
    ch=inportb(0x0305);
    printf("\nData read from FIFO 5 :  %002X /\n",ch);
```

```
    getch();
  }
  terminate = 1;
}

int main(void)
{
   outportb(0x037f,0xff); //Reset Address
   for(int i=0;i<2;i++) // Architecture Select and Frequency Select
   {
     outportb(0x0380,0xff);
   }
   getch();

/* save the old interrupt vector */
   oldhandler3 = getvect(INT3);
   oldhandler5 = getvect(INT5);

/* install the new interrupt handler */
   setvect(INT3, handler3);
   setvect(INT5, handler5);

/* wait till ff_empty */
   while (terminate == 0 )
   {
      printf("Reading value %d = %d",count,no);
   }

/* reset the old interrupt handler */
   setvect(INT3, oldhandler3);
   setvect(INT5, oldhandler5);

   return 0;
}
```

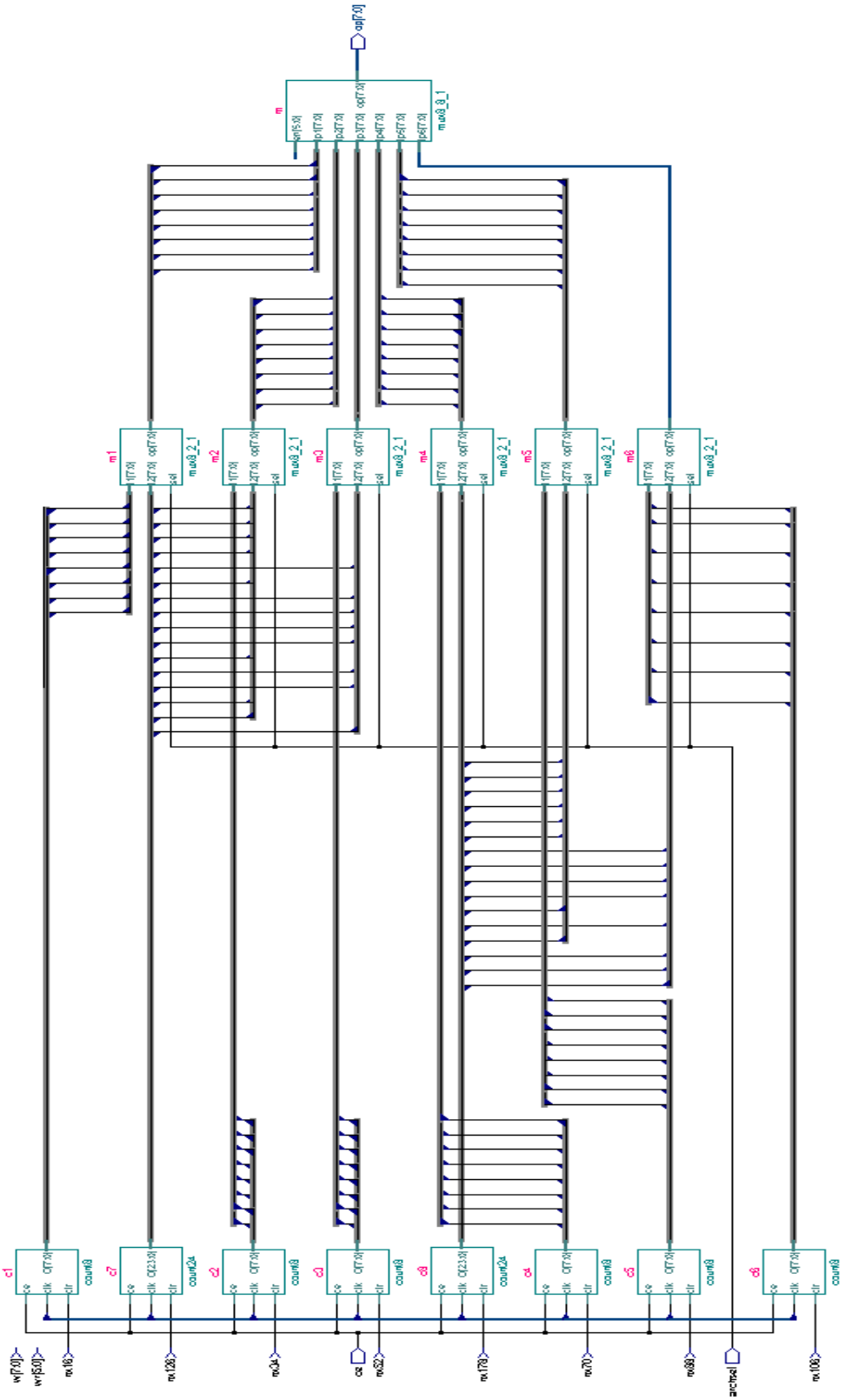
Simulation and Testing

Active-HDL timing diagrams: -

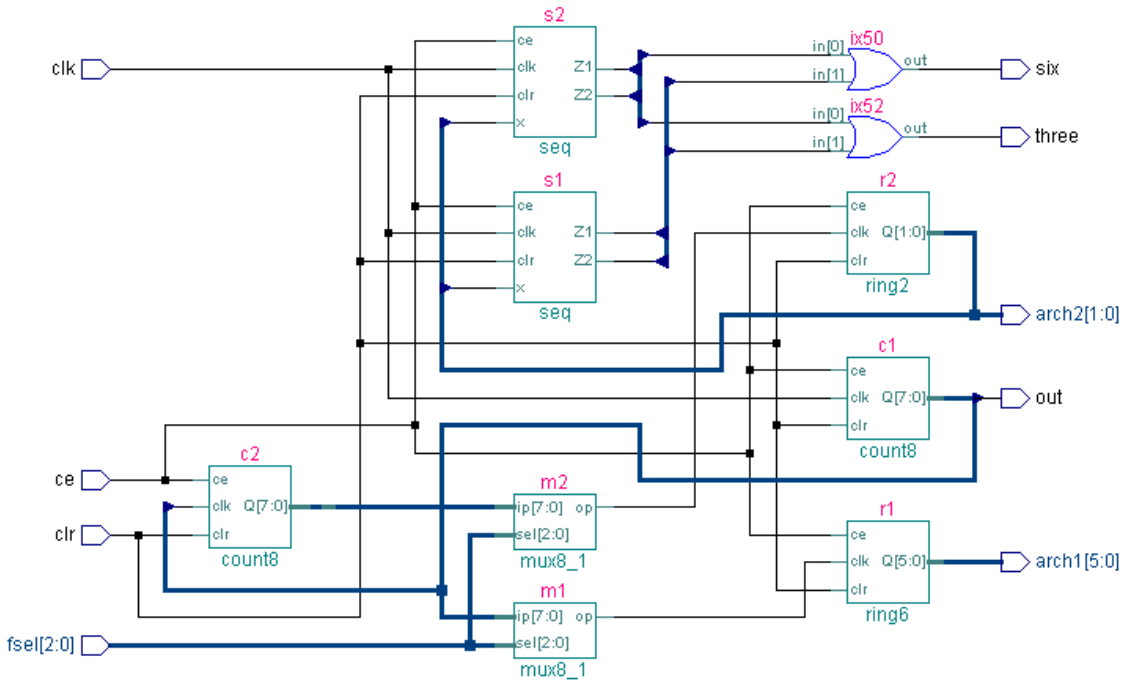
Synthesized circuit: -



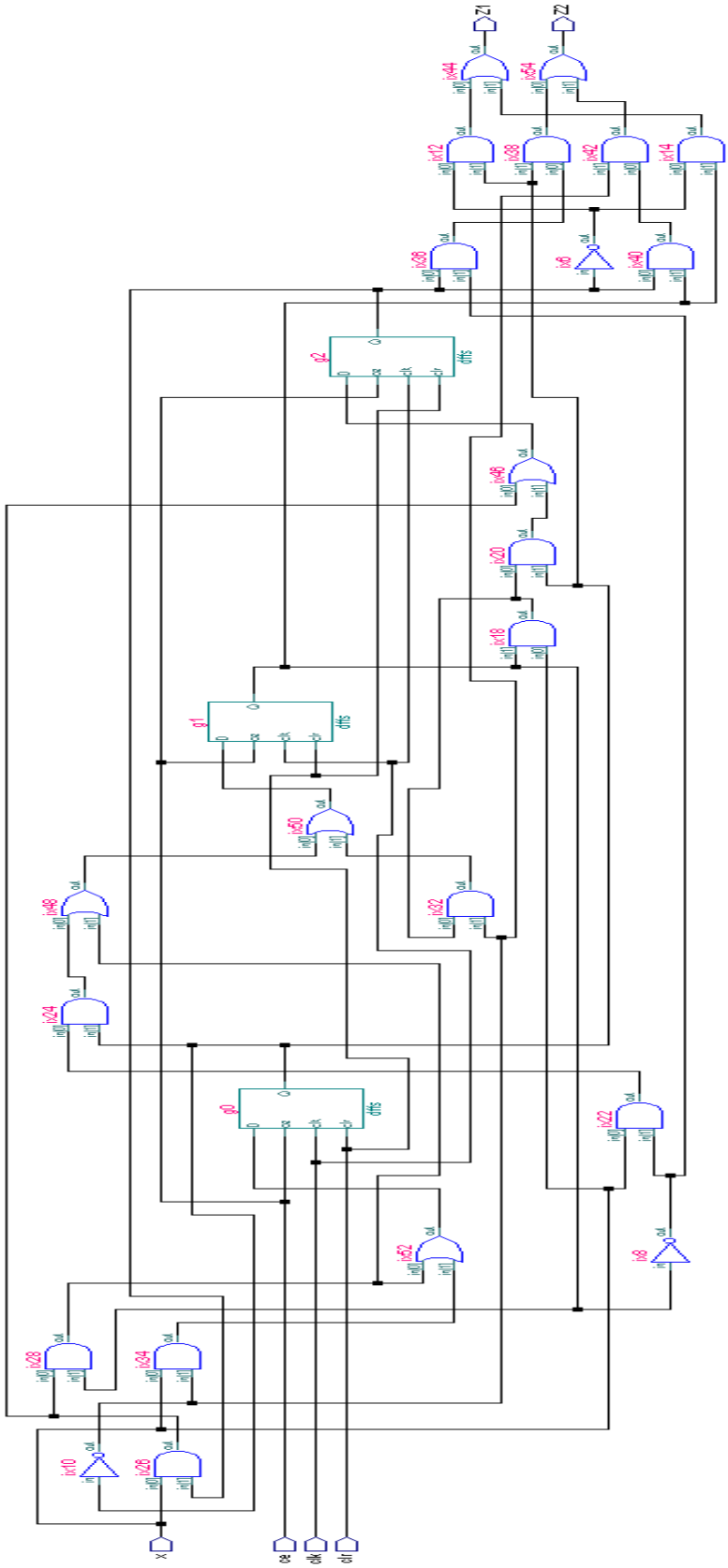
MCS System



Input-Block



Trigger Block



Sequence Generator

CONCLUSION

The multi-channel scaler that has been implemented provides flexibility with accuracy as there is no dead time between two counts as the counting moves on to the next counter after the bin-time. As the application program averages the counts accrued over a period of time the effect of noise signals is nullified.

By providing two selectable architectures for different time-bin improves the time range within which it can operate. Spartan II FPGA from Xilinx provides the all-important flexibility of programming the device and allows for on-field troubleshooting. The ISA interface to IBM PC enables the data from the experiments to be stored, analyzed with all the modern tools available for the PC platform. The project can be further modified to a stand-alone application with a micro-controller also designed within the FPGA thus further increasing the efficiency. If the system is found to function without any discrepancies then the whole system can also be designed as an ASIC chip and given for fabrication.

BIBLIOGAPHY

We used these published articles, books and data-sheets during the course of our project:

1. Samir Palnitkar, Verilog HDL- A Guide to VLSI Circuits and systems, John Wiley & sons,2001.
2. Sung-Mo Kang and Yusuf Leblebici, CMOS – Digital Integrated Circuits, Tata McGraw Hill
3. John F. Wakerley, Digital Design, 3rd edition, Pearson education.
4. Product Manual, Stanford Research Systems.
5. Nuclear Instrumentation Systems, BARC.
6. Multi-channel Scaler for general Statistical analysis of dynamic light gathering, Rudolf Sqrik and Edwin Baaij .
7. Spartan II datasheets – www.Xilinx.com.

APPENDIX

Pin Definitions of Spartan:

Pin Name	Dedicated Pin	Direction	Description
GCK0, GCK1, GCK2, GCK3	No	Input	Clock input pins that connect to Global Clock Buffers. These pins become user inputs when not needed for clocks.
M0, M1, M2	Yes	Input	Mode pins are used to specify the configuration mode.
CCLK	Yes	Input or Output	The configuration Clock I/O pin. It is an input for slave-parallel and slave-serial modes, and output in master-serial mode.
PROGRAM	Yes	Input	Initiates a configuration sequence when asserted Low.
DONE	Yes	Bidirectional	Indicates that configuration loading is complete, and that the start-up sequence is in progress. The output may be open drain.
INIT	No	Bidirectional (Open-drain)	When Low, indicates that the configuration memory is being cleared. This pin becomes a user I/O after configuration.
BUSY/DOUT	No	Output	In Slave Parallel mode, BUSY controls the rate at which configuration data is loaded. This pin becomes a user I/O after configuration unless the Slave Parallel port is retained. In serial modes, DOUT provides configuration data to downstream devices in a daisy-chain. This pin becomes a user I/O after configuration.
D0/DIN, D1, D2, D3, D4, D5, D6, D7	No	Input or Output	In Slave Parallel mode, D0-D7 are configuration data input pins. During readback, D0-D7 are output pins. These pins become user I/Os after configuration unless the Slave Parallel port is retained. In serial modes, DIN is the single data input. This pin becomes a user I/O after configuration.
WRITE	No	Input	In Slave Parallel mode, the active-low Write Enable signal. This pin becomes a user I/O after configuration unless the Slave Parallel port is retained.
CS	No	Input	In Slave Parallel mode, the active-low Chip Select signal. This pin becomes a user I/O after configuration unless the Slave Parallel port is retained.
TDI, TDO, TMS,	Yes	Mixed	Boundary Scan Test Access Port pins (IEEE 1149.1).

TCK			
V_{CCINT}	Yes	Input	Power supply pins for the internal core logic.
V_{CCO}	Yes	Input	Power supply pins for output drivers (subject to banking rules)
V_{REF}	No	Input	Input threshold voltage pins. Become user I/Os when an external threshold voltage is not needed (subject to banking rules).
GND	Yes	Input	Ground.
IRDY, TRDY	No	See PCI core documentation	These signals can only be accessed when using Xilinx PCI cores. If the cores are not used, these pins are available as user I/Os.

National Semiconductor

June 1989

5474/DM5474/DM7474
Dual Positive-Edge-Triggered D Flip-Flops with Preset, Clear and Complementary Outputs

General Description

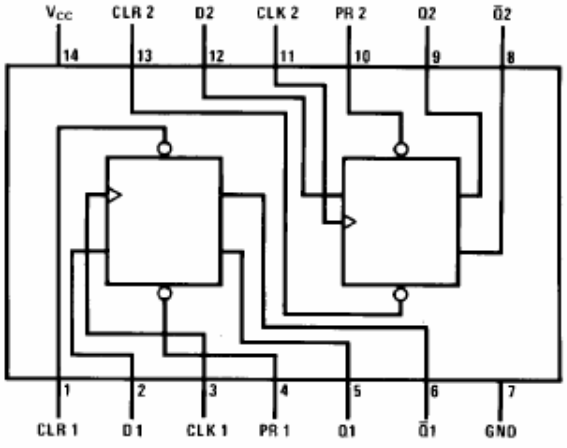
This device contains two independent positive-edge-triggered D flip-flops with complementary outputs. The information on the D input is accepted by the flip-flops on the positive going edge of the clock pulse. The triggering occurs at a voltage level and is not directly related to the transition time of the rising edge of the clock. The data on the D input may be changed while the clock is low or high without affecting the outputs as long as the data setup and hold times are not violated. A low logic level on the preset or clear inputs will set or reset the outputs regardless of the logic levels of the other inputs.

Features

- Alternate Military/Aerospace device (5474) is available. Contact a National Semiconductor Sales Office/Distributor for specifications.

Connection Diagram

Dual-In-Line Package



TL/F/6526–1

Order Number 5474DMQB, 5474FMQB, DM5474J, DM5474W, DM7474M or DM7474N
See NS Package Number J14A, M14A, N14A or W14B

Function Table

Inputs				Outputs	
PR	CLR	CLK	D	Q	$\overline{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H*	H*
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\overline{Q}_0$

H = High Logic Level
X = Either Low or High Logic Level
L = Low Logic Level
↑ = Positive-going transition of the clock.
* = This configuration is nonstable; that is, it will not persist when either the preset and/or clear inputs return to their inactive (high) level.
Q_0 = The output logic level of Q before the indicated input conditions were established.

National Semiconductor

March 1993

DM54123/DM74123 Dual Retriggerable One-Shot with Clear and Complementary Outputs

General Description

The '123 is a dual retriggerable monostable multivibrator capable of generating output pulses from a few nano-seconds to extremely long duration up to 100% duty cycle. Each device has three inputs permitting the choice of either leading-edge or trailing edge triggering. Pin (A) is an active-low transition trigger input and pin (B) is an active-high transition trigger input. A low at the clear (CLR) input terminates the output pulse: which also inhibits triggering. An internal connection from CLR to the input gate makes it possible to trigger the circuit by a positive-going signal on CLR as shown in the truth table.

To obtain the best and trouble free operation from this device please read the operating rules as well as the NSC one–shot application notes carefully and observe recommendations.

Features

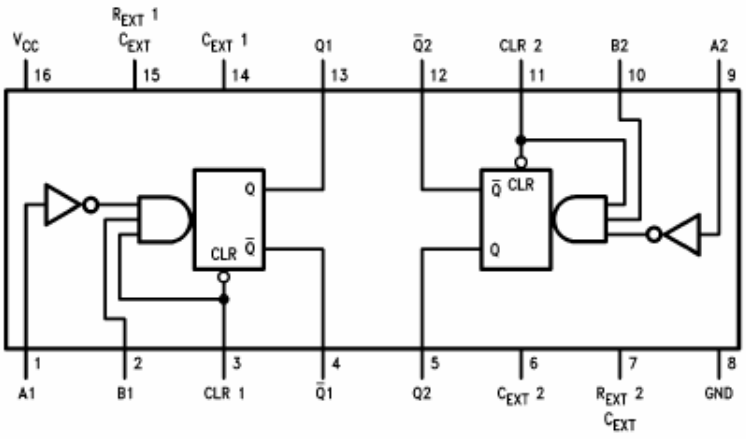
- DC triggered from active-high transition or active-low transition inputs
- Retriggerable to 100% duty cycle
- Direct reset terminates output pulse
- Compensated for V_{CC} and temperature variations
- DTL, TTL compatible
- Input clamp diodes

Functional Description

The basic output pulse width is determined by selection of an external resistor (R_X) and capacitor (C_X). Once triggered, the basic pulse width may be extended by retriggering the gated active-low transition or active-high transition inputs or be reduced by use of the active-low transition clear input. Retriggering to 100% duty cycle is possible by application of an input pulse train whose cycle time is shorter than the output cycle time such that a continuous "HIGH" logic state is maintained at the "Q" output.

Connection Diagram

Dual-In-Line Package



TL/F/6539–1

Order Number DM54123J-MIL, DM54123W-MIL or DM74123N
See NS Package Number J16A, N16A or W16A

Triggering Truth Table

Inputs			Response
A	B	CLR	
X	X	L	No Trigger
↓	L	X	No Trigger
↓	H	H	Trigger
H	↑	X	No Trigger
L	↑	H	Trigger
L	H	↑	Trigger

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

Operating Rules

1. An external resistor (R_X) and external capacitor (C_X) are required for proper operation. The value of C_X may vary from 0 to any necessary value. For small time constants high-grade mica, glass, polypropylene, polycarbonate, or polystyrene material capacitors may be used. For large time constants use tantalum or special aluminum capacitors. If the timing capacitors have leakages approaching 100 nA or if stray capacitance from either terminal to ground is greater than 50 pF the timing equations may not represent the pulse width the device generates.
2. When an electrolytic capacitor is used for C_X a switching diode is often required for standard TTL one-shots to prevent high inverse leakage current *(Figure 1)*. However, its use in general is not recommended with retriggerable operation.
3. The output pulse width (T_W) for $C_X > 1000$ pF is defined as follows:

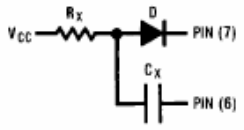
   $T_W = K R_X C_X (1 + 0.7/R_X)$

   where [R_X is in Kilo-ohm]

   [C_X is in pico Farad]

   [T_W is in nano second]

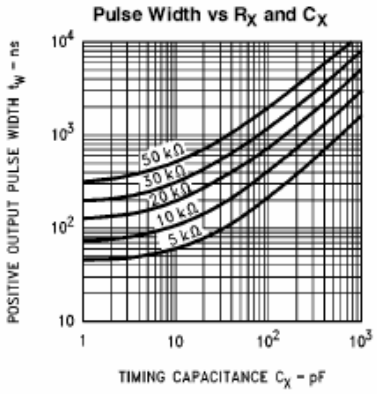
   [$K \approx 0.28$]



TL/F/6539–3

FIGURE 1

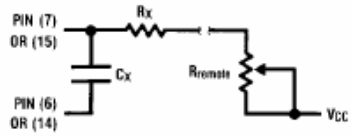
4. For $C_X < 1000$ pF see *Figure 2* for T_W vs C_X family curves with R_X as a parameter:



TL/F/6539–4

FIGURE 2

5. To obtain variable pulse width by remote trimming, the following circuit is recommended:



TL/F/6539–5

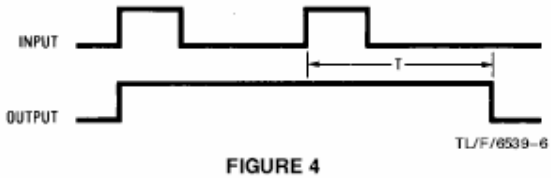
Note: "R_{remote}" should be as close to the one-shot as possible.

FIGURE 3

6. The retriggerable pulse width is calculated as shown below:

   $$T = T_W + t_{PLH} = K \times R_X \times C_X + t_{PLH}$$

   The retriggered pulse width is equal to the pulse width plus a delay time period *(Figure 4)*.



TL/F/6539–6

FIGURE 4

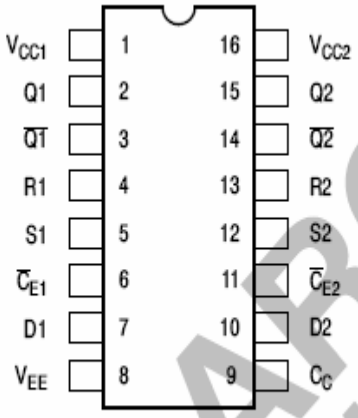
7. Under any operating condition C_X and R_X must be kept as close to the one-shot device pins as possible to minimize stray capacitance, to reduce noise pick-up, and to reduce $I \times R$ and Ldi/dt voltage developed along their connecting paths. If the lead length from C_X to pins (6) and (7) or pins (14) and (15) is greater than 3 cm, for example, the output pulse width might be quite different from values predicted from the appropriate equations. A non-inductive and low capacitive path is necessary to ensure complete discharge of C_X in each cycle of its operation so that the output pulse width will be accurate.
8. V_{CC} and ground wiring should conform to good high-frequency standards and practices so that switching transients on the V_{CC} and ground return leads do not cause interaction between one-shots. A 0.01 μF to 0.10 μF bypass capacitor (disk ceramic or monolithic type) from V_{CC} to ground is necessary on each device. Furthermore, the bypass capacitor should be located as close to the V_{CC} pin as space permits.

MC10131
Dual Type D Master-Slave Flip-Flop

The MC10131 is a dual master–slave type D flip–flop. Asynchronous Set (S) and Reset (R) override Clock (C_C) and Clock Enable (C_E) inputs. Each flip–flop may be clocked separately by holding the common clock in the low state and using the enable inputs for the clocking function. If the common clock is to be used to clock the flip–flop, the Clock Enable inputs must be in the low state. In this case, the enable inputs perform the function of controlling the common clock. The output states of the flip–flop change on the positive transition of the clock. A change in the information present at the data (D) input will not affect the output information at any other time due to master slave construction.

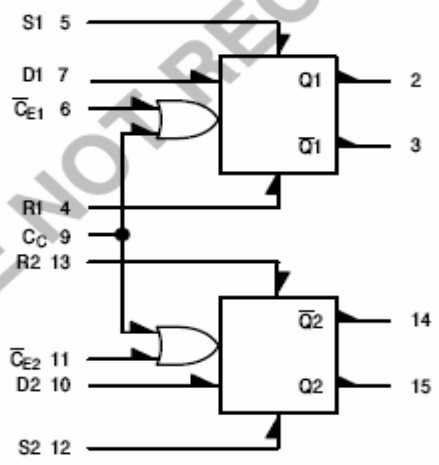
- P_D = 235 mW typ/pkg (No Load)
- F_{Tog} = 160 MHz typ
- t_{pd} = 3.0 ns typ
- t_r, t_f = 2.5 ns typ (20%–80%)

DIP PIN ASSIGNMENT



Pin assignment is for Dual–in–Line Package.
For PLCC pin assignment, see the Pin Conversion Tables on page 18 of the ON Semiconductor MECL Data Book (DL122/D).

LOGIC DIAGRAM



V_{CC1} = PIN 1
V_{CC2} = PIN 16
V_{EE} = PIN 8

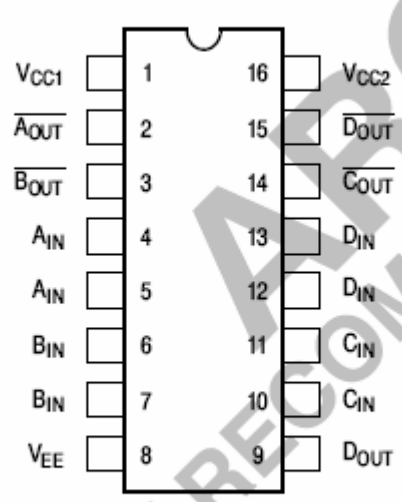
MC10102

Quad 2-Input NOR Gate

The MC10102 is a quad 2–input NOR gate. The MC10102 provides one gate with OR/NOR outputs.

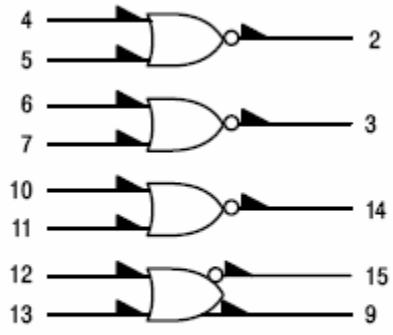
- P_D = 25 mW typ/gate (No Load)
- t_{pd} = 2.0 ns typ
- t_r, t_f = 2.0 ns typ (20%–80%)

DIP
PIN ASSIGNMENT



Pin assignment is for Dual–in–Line Package.
For PLCC pin assignment, see the Pin Conversion Tables on page 18 of the ON Semiconductor MECL Data Book (DL122/D).

LOGIC DIAGRAM



V_{CC1} = PIN 1
V_{CC2} = PIN 16
V_{EE} = PIN 8

National Semiconductor

June 1989

5408/DM5408/DM7408
Quad 2-Input AND Gates

General Description

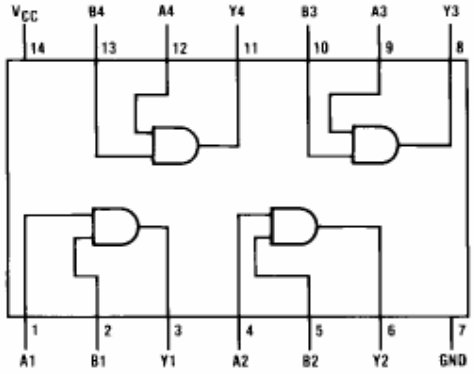
This device contains four independent gates each of which performs the logic AND function.

Features

- Alternate Military/Aerospace device (5408) is available. Contact a National Semiconductor Sales Office/Distributor for specifications.

Connection Diagram

Dual-In-Line Package



TL/F/6498–1

Order Number 5408DMQB, 5408FMQB, DM5408J, DM5408W or DM7408N
See NS Package Number J14A, N14A or W14B

Function Table

$Y = AB$

Inputs		Output
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

H = High Logic Level

L = Low Logic Level

MC10125

Quad MECL to TTL Translator

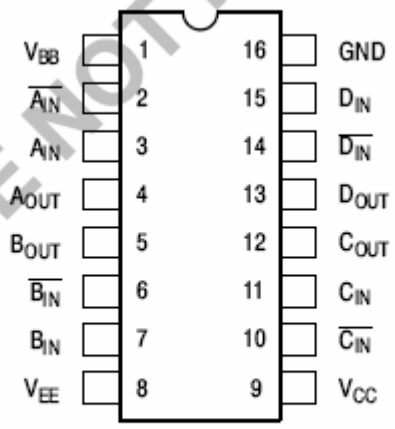
The MC10125 is a quad translator for interfacing data and control signals between the MECL section and saturated logic sections of digital systems. The MC10125 incorporates differential inputs and Schottky TTL "totem pole" outputs. Differential inputs allow for use as an inverting/ non–inverting translator or as a differential line receiver. The V_{BB} reference voltage is available on pin 1 for use in single–ended input biasing. The outputs of the MC10125 go to a low logic level whenever the inputs are left floating.

Power supply requirements are ground, +5.0 Volts and –5.2 Volts. Propagation delay of the MC10125 is typically 4.5 ns. The MC10125 has fanout of 10 TTL loads. The dc levels are MECL 10,000 in and Schottky TTL, or TTL out. This device has an input common mode noise rejection of ± 1.0 Volt.

An advantage of this device is that MECL level information can be received, via balanced twisted pair lines, in the TTL equipment. This isolates the MECL logic from the noisy TTL environment. This device is useful in computers, instrumentation, peripheral controllers, test equipment and digital communications systems.

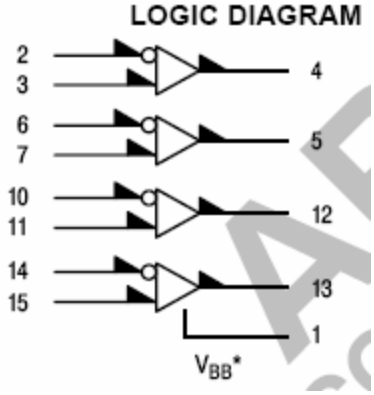
- P_D = 380 mW typ/pkg (No Load)
- t_{pd} = 4.5 ns typ (50% to + 1.5 Vdc out)
- t_r, t_f = 2.5 ns typ (1.0 V to 2.0 V)

DIP PIN ASSIGNMENT



Pin assignment is for Dual–in–Line Package.
For PLCC pin assignment, see the Pin Conversion Tables on page 18 of the ON Semiconductor MECL Data Book (DL122/D).

LOGIC DIAGRAM



MC10124

Quad TTL to MECL Translator

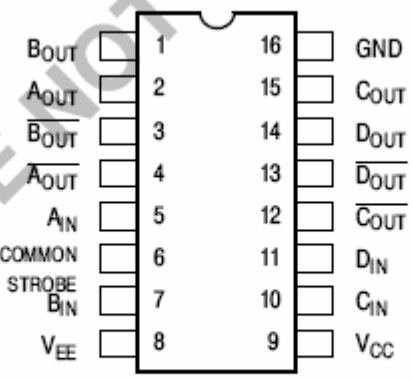
The MC10124 is a quad translator for interfacing data and controlsignals between a saturated logic section and the MECL section of digital systems. The MC10124 has TTL compatible inputs, and MECL complementary open–emitter outputs that allow use as an inverting/ non–inverting translator or as a differential line driver.

When the common strobe input is at the low logic level, it forces all true outputs to a MECL low logic state and all inverting outputs to a MECL high logic state. Power supply requirements are ground, +5.0 Volts, and –5.2 Volts.

Propagation delay of the MC10124 is typically 3.5 ns. The dc levels are standard or Schottky TTL in, MECL 10,000 out. An advantage of this device is that TTL level information can be transmitted differentially, via balanced twisted pair lines, to the MECL equipment, where the signal can be received by the MC10115 or MC10116 differential line receivers. The MC10124 is useful in computers, instrumentation, peripheral controllers, test equipment, and digital communications systems.

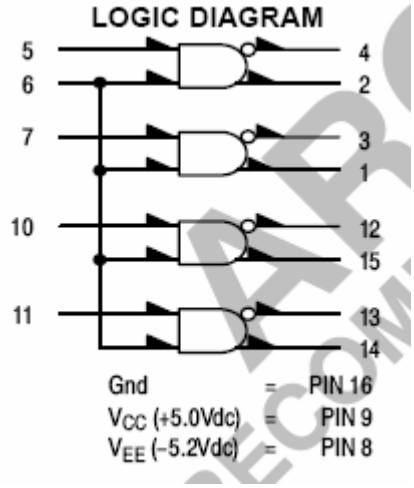
- P_D = 380 mW typ/pkg (No Load)
- t_{pd} = 3.5 ns typ (+ 1.5 Vdc in to 50% out)
- t_r, t_f = 2.5 ns typ (20%–80%)

DIP PIN ASSIGNMENT



Pin assignment is for Dual–in–Line Package.
For PLCC pin assignment, see the Pin Conversion Tables on page 18 of the ON Semiconductor MECL Data Book (DL122/D).

LOGIC DIAGRAM



National Semiconductor

December 1994

LM101A/LM201A/LM301A Operational Amplifiers

General Description

The LM101A series are general purpose operational amplifiers which feature improved performance over industry standards like the LM709. Advanced processing techniques make possible an order of magnitude reduction in input currents, and a redesign of the biasing circuitry reduces the temperature drift of input current. Improved specifications include:

- Offset voltage 3 mV maximum over temperature (LM101A/LM201A)
- Input current 100 nA maximum over temperature (LM101A/LM201A)
- Offset current 20 nA maximum over temperature (LM101A/LM201A)
- Guaranteed drift characteristics
- Offsets guaranteed over entire common mode and supply voltage ranges
- Slew rate of 10V/μs as a summing amplifier

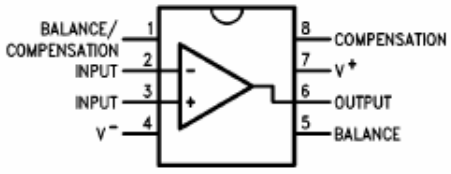
This amplifier offers many features which make its application nearly foolproof: overload protection on the input and output, no latch-up when the common mode range is exceeded, and freedom from oscillations and compensation with a single 30 pF capacitor. It has advantages over internally compensated amplifiers in that the frequency compensation can be tailored to the particular application. For example, in low frequency circuits it can be overcompensated for increased stability margin. Or the compensation can be optimized to give more than a factor of ten improvement in high frequency performance for most applications.

In addition, the device provides better accuracy and lower noise in high impedance circuitry. The low input currents also make it particularly well suited for long interval integrators or timers, sample and hold circuits and low frequency waveform generators. Further, replacing circuits where matched transistor pairs buffer the inputs of conventional IC op amps, it can give lower offset voltage and a drift at a lower cost.

The LM101A is guaranteed over a temperature range of −55°C to +125°C, the LM201A from −25°C to +85°C, and the LM301A from 0°C to +70°C.

Connection Diagrams (Top View)

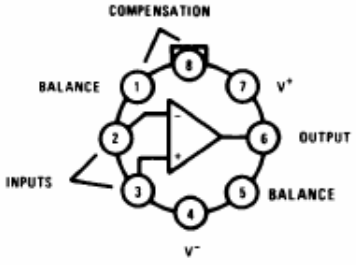
Dual-In-Line Package



TL/H/7752–4

Order Number LM101AJ, LM101J/883*, LM201AN or LM301AN
See NS Package Number J08A or N08A

Metal Can Package

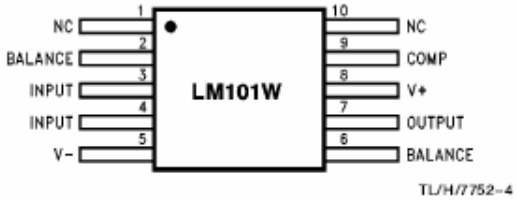


TL/H/7752–2

Note: Pin 4 connected to case.

Order Number LM101AH, LM101AH/883*, LM201AH or LM301AH
See NS Package Number H08C

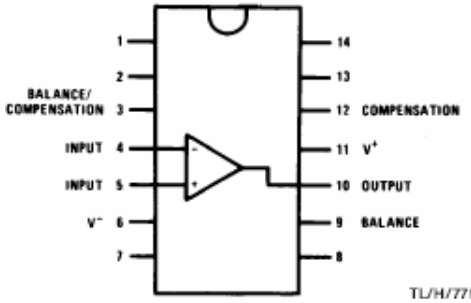
Ceramic Flatpack Package



TL/H/7752–4

Order Number LM101AW/883 or LM101W/883
See NS Package Number W10A

Dual-In-Line Package



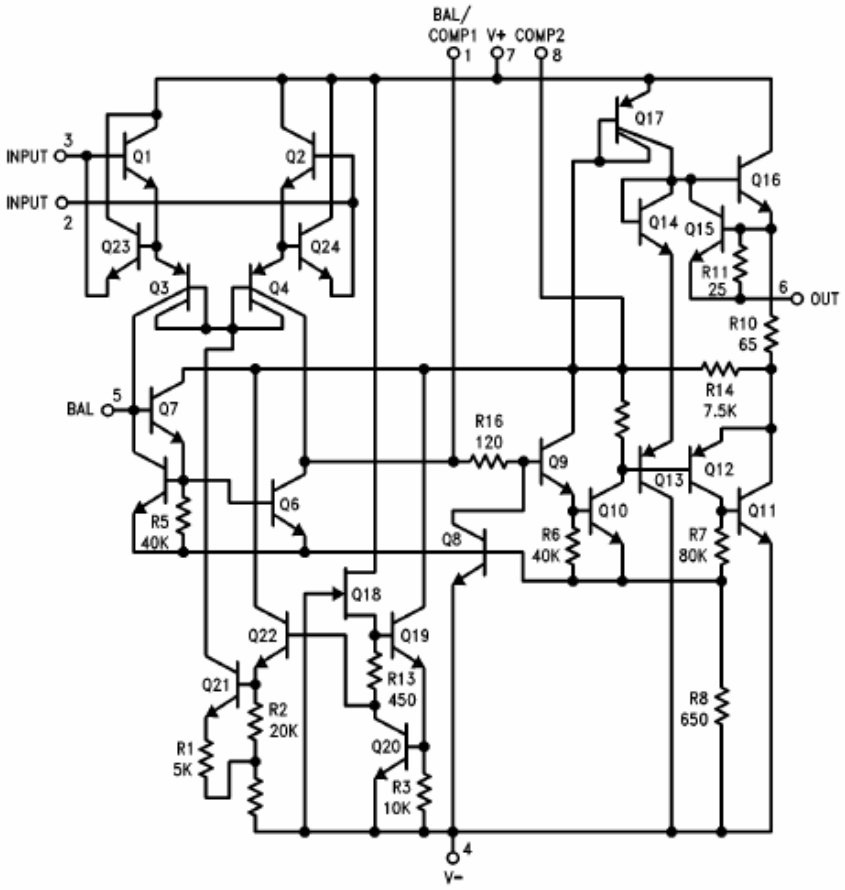
TL/H/7752–3

Order Number LM101AJ-14/883*
See NS Package Number J14A

*Available per JM38510/10103.

Schematic**

TL/H/7752–38



TL/H/7752–1

ANALOG DEVICES

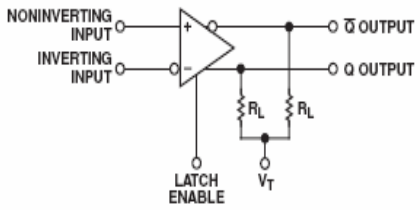
Ultrafast Comparators

AD96685/AD96687

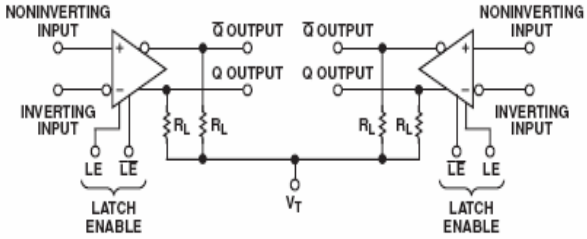
FEATURES
Fast: 2.5 ns Propagation Delay
Low Power: 118 mW per Comparator
Packages: DIP, SOIC, PLCC
Power Supplies: +5 V, –5.2 V
Logic Compatibility: ECL
50 ps Delay Dispersion

APPLICATIONS
High Speed Triggers
High Speed Line Receivers
Threshold Detectors
Window Comparators
Peak Detectors

AD96685 FUNCTIONAL BLOCK DIAGRAM



AD96687 FUNCTIONAL BLOCK DIAGRAM



THE OUTPUTS ARE OPEN EMITTERS, REQUIRING EXTERNAL PULL-DOWN RESISTORS. THESE RESISTORS MAY BE IN THE RANGE OF 50Ω-200Ω CONNECTED TO –2.0V, OR 200Ω-2000Ω

GENERAL DESCRIPTION

The AD96685 and AD96687 are ultrafast voltage comparators. The AD96685 is a single comparator with 2.5 ns propagation delay; the AD96687 is an equally fast dual comparator. Both devices feature 50 ps propagation delay dispersion which is a particularly important characteristic of high-speed comparators. It is a measure of the difference in propagation delay under differing overdrive conditions.

A fast, high precision differential input stage permits consistent propagation delay with a wide variety of signals in the common-mode range from –2.5 V to +5 V. Outputs are complementary digital signals fully compatible with ECL 10 K and 10 KH logic families. The outputs provide sufficient drive current to directly drive transmission lines terminated in 50 Ω to –2 V. A level sensitive latch input which permits tracking, track-hold, or sample-hold modes of operation is included.

The AD96685 is available in industrial –25°C to +85°C range in 16-pin SOIC.

The AD96687 is available in industrial range –25°C to +85°C, in 16-pin DIP, SOIC, and 20-lead PLCC.

National Semiconductor

June 1989

54LS245/DM54LS245/DM74LS245 TRI-STATE® Octal Bus Transceiver

General Description

These octal bus transceivers are designed for asynchronous two-way communication between data buses. The control function implementation minimizes external timing requirements.

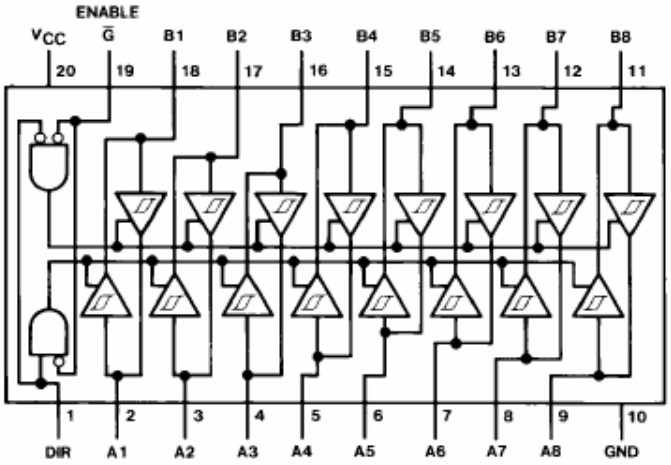
The device allows data transmission from the A bus to the B bus or from the B bus to the A bus depending upon the logic level at the direction control (DIR) input. The enable input ($\overline{G}$) can be used to disable the device so that the buses are effectively isolated.

Features

- Bi-Directional bus transceiver in a high-density 20-pin package
- TRI-STATE outputs drive bus lines directly
- PNP inputs reduce DC loading on bus lines
- Hysteresis at bus inputs improve noise margins
- Typical propagation delay times, port-to-port 8 ns
- Typical enable/disable times 17 ns
- I_{OL} (sink current)
  - 54LS 12 mA
  - 74LS 24 mA
- I_{OH} (source current)
  - 54LS −12 mA
  - 74LS −15 mA
- Alternate Military/Aerospace device (54LS245) is available. Contact a National Semiconductor Sales Office/ Distributor for specifications.

Connection Diagram

Dual-In-Line Package



TL/F/6413–1

Order Number 54LS245DMQB, 54LS245FMQB, 54LS245LMQB, DM54LS245J, DM54LS245W, DM74LS245WM or DM74LS245N
See NS Package Number E20A, J20A, M20B, N20A or W20A

Function Table

Enable $\overline{G}$	Direction Control DIR	Operation
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = High Level, L = Low Level, X = Irrelevant

National Semiconductor

August 1989

54LS244/DM74LS244 Octal TRI-STATE® Buffers/Line Drivers/Line Receivers

General Description

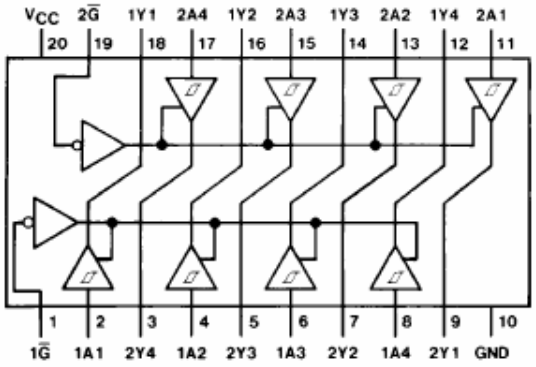
These buffers/line drivers are designed to improve both the performance and PC board density of TRI-STATE buffers/drivers employed as memory-address drivers, clock drivers, and bus-oriented transmitters/receivers. Featuring 400 mV of hysteresis at each low current PNP data line input, they provide improved noise rejection and high fanout outputs and can be used to drive terminated lines down to 133Ω.

Features

- TRI-STATE outputs drive bus lines directly
- PNP inputs reduce DC loading on bus lines
- Hysteresis at data inputs improves noise margins
- Typical I_{OL} (sink current)
  - 54LS 12 mA
  - 74LS 24 mA
- Typical I_{OH} (source current)
  - 54LS −12 mA
  - 74LS −15 mA
- Typical propagation delay times
  - Inverting 10.5 ns
  - Noninverting 12 ns
- Typical enable/disable time 18 ns
- Typical power dissipation (enabled)
  - Inverting 130 mW
  - Noninverting 135 mW

Connection Diagram

Dual-In-Line Package



TL/F/8442–1

Order Number 54LS244DMQB, 54LS244FMQB, 54LS244LMQB, DM74LS244WM or DM74LS244N
See NS Package Number E20A, J20A, M20B, N20A or W20A

Function Table

Inputs		Output
$\overline{G}$	A	Y
L	L	L
L	H	H
H	X	Z

L = Low Logic Level
H = High Logic Level
X = Either Low or High Logic Level
Z = High Impedance

XC2S200 Device Pinouts

XC2S200 Pad Name					
Function	Bank	PQ208	FG256	FG456	Bndry Scan
GND	-	P1	GND*	GND*	-
TMS	-	P2	D3	D3	-
I/O	7	P3	C2	B1	257
I/O	7	-	-	E4	263
I/O	7	-	-	C1	266
I/O	7	-	A2	F5	269
GND	-	-	GND*	GND*	-
I/O, V_{REF}	7	P4	B1	D2	272
I/O	7	-	-	E3	275
I/O	7	-	-	F4	281
GND	-	-	GND*	GND*	-
I/O	7	-	E3	G5	284
I/O	7	P5	D2	F3	287
GND	-	-	GND*	GND*	-
V_{CCO}	7	-	V_{CCO} Bank 7*	V_{CCO} Bank 7*	-
I/O, V_{REF}	7	P6	C1	E2	290
I/O	7	P7	F3	E1	293
I/O	7	-	-	G4	296
I/O	7	-	-	G3	299
I/O	7	-	E2	H5	302
GND	-	-	GND*	GND*	-
I/O	7	P8	E4	F2	305
I/O	7	-	-	F1	308
I/O, V_{REF}	7	P9	D1	H4	314
I/O	7	P10	E1	G1	317
GND	-	P11	GND*	GND*	-
V_{CCO}	7	P12	V_{CCO} Bank 7*	V_{CCO} Bank 7*	-
V_{CCINT}	-	P13	V_{CCINT}*	V_{CCINT}*	-
I/O	7	P14	F2	H3	320
I/O	7	P15	G3	H2	323
I/O	7	-	-	J4	326
I/O	7	-	-	H1	329
I/O	7	-	F1	J5	332
GND	-	-	GND*	GND*	-
I/O	7	P16	F4	J2	335
I/O	7	-	-	J3	338
I/O	7	-	-	J1	341
I/O	7	P17	F5	K5	344
I/O	7	P18	G2	K1	347
GND	-	P19	GND*	GND*	-

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name					
Function	Bank	PQ208	FG256	FG456	Bndry Scan
V_{CCO}	7	-	V_{CCO} Bank 7*	V_{CCO} Bank 7*	-
I/O, V_{REF}	7	P20	H3	K3	350
I/O	7	P21	G4	K4	353
I/O	7	-	-	K2	359
I/O	7	-	H2	L6	362
I/O	7	P22	G5	L1	365
I/O	7	-	-	L5	368
I/O	7	P23	H4	L4	374
I/O, IRDY(1)	7	P24	G1	L3	377
GND	-	P25	GND*	GND*	-
V_{CCO}	7	P26	V_{CCO} Bank 7*	V_{CCO} Bank 7*	-
V_{CCO}	6	P26	V_{CCO} Bank 6*	V_{CCO} Bank 6*	-
I/O, TRDY(1)	6	P27	J2	M1	380
V_{CCINT}	-	P28	V_{CCINT}*	V_{CCINT}*	-
I/O	6	-	-	M6	389
I/O	6	P29	H1	M3	392
I/O	6	-	J4	M4	395
I/O	6	-	-	N1	398
I/O	6	P30	J1	M5	404
I/O, V_{REF}	6	P31	J3	N2	407
V_{CCO}	6	-	V_{CCO} Bank 6*	V_{CCO} Bank 6*	-
GND	-	P32	GND*	GND*	-
I/O	6	P33	K5	N3	410
I/O	6	P34	K2	N4	413
I/O	6	-	-	P1	416
I/O	6	-	-	N5	419
I/O	6	P35	K1	P2	422
GND	-	-	GND*	GND*	-
I/O	6	-	K3	P4	425
I/O	6	-	-	R1	428
I/O	6	-	-	P5	431
I/O	6	P36	L1	P3	434
I/O	6	P37	L2	R2	437
V_{CCINT}	-	P38	V_{CCINT}*	V_{CCINT}*	-
V_{CCO}	6	P39	V_{CCO} Bank 6*	V_{CCO} Bank 6*	-
GND	-	P40	GND*	GND*	-
I/O	6	P41	K4	T1	440
I/O, V_{REF}	6	P42	M1	R4	443

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name					Bndry Scan
Function	Bank	PQ208	FG256	FG456	
I/O	6	-	-	T2	449
I/O	6	P43	L4	U1	452
GND	-	-	GND*	GND*	-
I/O	6	-	M2	R5	455
I/O	6	-	-	V1	458
I/O	6	-	-	T5	461
I/O	6	P44	L3	U2	464
I/O, V_{REF}	6	P45	N1	T3	467
V_{CCO}	6	-	V_{CCO} Bank 6*	V_{CCO} Bank 6*	-
GND	-	-	GND*	GND*	-
I/O	6	P46	P1	T4	470
I/O	6	-	L5	W1	473
GND	-	-	GND*	GND*	-
I/O	6	-	-	V2	476
I/O	6	-	-	U4	482
I/O, V_{REF}	6	P47	N2	Y1	485
GND	-	-	GND*	GND*	-
I/O	6	-	M4	W2	488
I/O	6	-	-	V3	491
I/O	6	-	-	V4	494
I/O	6	P48	R1	Y2	500
I/O	6	P49	M3	W3	503
M1	-	P50	P2	U5	506
GND	-	P51	GND*	GND*	-
M0	-	P52	N3	AB2	507
V_{CCO}	6	P53	V_{CCO} Bank 6*	V_{CCO} Bank 6*	-
V_{CCO}	5	P53	V_{CCO} Bank 5*	V_{CCO} Bank 5*	-
M2	-	P54	R3	Y4	508
I/O	5	-	-	W5	518
I/O	5	-	-	AB3	521
I/O	5	-	N5	V7	524
GND	-	-	GND*	GND*	-
I/O, V_{REF}	5	P57	T2	Y6	527
I/O	5	-	-	AA4	530
I/O	5	-	-	AB4	536
I/O	5	-	P5	W6	539
I/O	5	P58	T3	Y7	542
GND	-	-	GND*	GND*	-

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name					Bndry Scan
Function	Bank	PQ208	FG256	FG456	
V_{CCO}	5	-	V_{CCO} Bank 5*	V_{CCO} Bank 5*	-
I/O, V_{REF}	5	P59	T4	AA5	545
I/O	5	P60	M6	AB5	548
I/O	5	-	-	V8	551
I/O	5	-	-	AA6	554
I/O	5	-	T5	AB6	557
GND	-	-	GND*	GND*	-
I/O	5	P61	N6	AA7	560
I/O	5	-	-	W7	563
I/O, V_{REF}	5	P62	R5	W8	569
I/O	5	P63	P6	Y8	572
GND	-	P64	GND*	GND*	-
V_{CCO}	5	P65	V_{CCO} Bank 5*	V_{CCO} Bank 5*	-
V_{CCINT}	-	P66	V_{CCINT}*	V_{CCINT}*	-
I/O	5	P67	R6	AA8	575
I/O	5	P68	M7	V9	578
I/O	5	-	-	AB8	581
I/O	5	-	-	W9	584
I/O	5	-	-	AB9	587
GND	-	-	GND*	GND*	-
I/O	5	P69	N7	Y9	590
I/O	5	-	-	V10	593
I/O	5	-	-	AA9	596
I/O	5	P70	T6	W10	599
I/O	5	P71	P7	AB10	602
GND	-	P72	GND*	GND*	-
V_{CCO}	5	-	V_{CCO} Bank 5*	V_{CCO} Bank 5*	-
I/O, V_{REF}	5	P73	P8	Y10	605
I/O	5	P74	R7	V11	608
I/O	5	-	-	AA10	614
I/O	5	-	T7	W11	617
I/O	5	P75	T8	AB11	620
I/O	5	-	-	U11	623
V_{CCINT}	-	P76	V_{CCINT}*	V_{CCINT}*	-
I, GCK1	5	P77	R8	Y11	635
V_{CCO}	5	P78	V_{CCO} Bank 5*	V_{CCO} Bank 5*	-
V_{CCO}	4	P78	V_{CCO} Bank 4*	V_{CCO} Bank 4*	-
GND	-	P79	GND*	GND*	-

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name					
Function	Bank	PQ208	FG256	FG456	Bndry Scan
I, GCK0	4	P80	N8	W12	636
I/O	4	P81	N9	U12	640
I/O	4	-	-	V12	646
I/O	4	P82	R9	Y12	649
I/O	4	-	N10	AA12	652
I/O	4	-	-	W13	655
I/O	4	P83	T9	AB13	661
I/O, V_{REF}	4	P84	P9	AA13	664
V_{CCO}	4	-	V_{CCO} Bank 4*	V_{CCO} Bank 4*	-
GND	-	P85	GND*	GND*	-
I/O	4	P86	M10	Y13	667
I/O	4	P87	R10	V13	670
I/O	4	-	-	AB14	673
I/O	4	-	-	W14	676
I/O	4	P88	P10	AA14	679
GND	-	-	GND*	GND*	-
I/O	4	-	-	V14	682
I/O	4	-	-	Y14	685
I/O	4	-	-	W15	688
I/O	4	P89	T10	AB15	691
I/O	4	P90	R11	AA15	694
V_{CCINT}	-	P91	V_{CCINT}*	V_{CCINT}*	-
V_{CCO}	4	P92	V_{CCO} Bank 4*	V_{CCO} Bank 4*	-
GND	-	P93	GND*	GND*	-
I/O	4	P94	M11	Y15	697
I/O, V_{REF}	4	P95	T11	AB16	700
I/O	4	-	-	AB17	706
I/O	4	P96	N11	V15	709
GND	-	-	GND*	GND*	-
I/O	4	-	R12	Y16	712
I/O	4	-	-	AA17	715
I/O	4	-	-	W16	718
I/O	4	P97	P11	AB18	721
I/O, V_{REF}	4	P98	T12	AB19	724
V_{CCO}	4	-	V_{CCO} Bank 4*	V_{CCO} Bank 4*	-
GND	-	-	GND*	GND*	-
I/O	4	P99	T13	Y17	727
I/O	4	-	N12	V16	730
I/O	4	-	-	AA18	733

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name					
Function	Bank	PQ208	FG256	FG456	Bndry Scan
I/O	4	-	-	W17	739
I/O, V_{REF}	4	P100	R13	AB20	742
GND	-	-	GND*	GND*	-
I/O	4	-	P12	AA19	745
I/O	4	-	-	V17	748
I/O	4	-	-	Y18	751
I/O	4	P101	P13	AA20	757
I/O	4	P102	T14	W18	760
GND	-	P103	GND*	GND*	-
DONE	3	P104	R14	Y19	763
V_{CCO}	4	P105	V_{CCO} Bank 4*	V_{CCO} Bank 4*	-
V_{CCO}	3	P105	V_{CCO} Bank 3*	V_{CCO} Bank 3*	-
$\overline{PROGRAM}$	-	P106	P15	W20	766
I/O ($\overline{INIT}$)	3	P107	N15	V19	767
I/O (D7)	3	P108	N14	Y21	770
I/O	3	-	-	V20	776
I/O	3	-	-	AA22	779
I/O	3	-	T15	W21	782
GND	-	-	GND*	GND*	-
I/O, V_{REF}	3	P109	M13	U20	785
I/O	3	-	-	U19	788
I/O	3	-	-	V21	794
GND	-	-	GND*	GND*	-
I/O	3	-	R16	T18	797
I/O	3	P110	M14	W22	800
GND	-	-	GND*	GND*	-
V_{CCO}	3	-	V_{CCO} Bank 3*	V_{CCO} Bank 3*	-
I/O, V_{REF}	3	P111	L14	U21	803
I/O	3	P112	M15	T20	806
I/O	3	-	-	T19	809
I/O	3	-	-	V22	812
I/O	3	-	L12	T21	815
GND	-	-	GND*	GND*	-
I/O	3	P113	P16	R18	818
I/O	3	-	-	U22	821
I/O, V_{REF}	3	P114	L13	R19	827
I/O (D6)	3	P115	N16	T22	830
GND	-	P116	GND*	GND*	-

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name					
Function	Bank	PQ208	FG256	FG456	Bndry Scan
V_{CCO}	3	P117	V_{CCO} Bank 3*	V_{CCO} Bank 3*	-
V_{CCINT}	-	P118	V_{CCINT}*	V_{CCINT}*	-
I/O (D5)	3	P119	M16	R21	833
I/O	3	P120	K14	P18	836
I/O	3	-	-	R22	839
I/O	3	-	-	P19	842
I/O	3	-	L16	P20	845
GND	-	-	GND*	GND*	-
I/O	3	P121	K13	P21	848
I/O	3	-	-	N19	851
I/O	3	-	-	P22	854
I/O	3	P122	L15	N18	857
I/O	3	P123	K12	N20	860
GND	-	P124	GND*	GND*	-
V_{CCO}	3	-	V_{CCO} Bank 3*	V_{CCO} Bank 3*	-
I/O, V_{REF}	3	P125	K16	N21	863
I/O (D4)	3	P126	J16	N22	866
I/O	3	-	-	M17	872
I/O	3	-	J14	M19	875
I/O	3	P127	K15	M20	878
I/O	3	-	-	M18	881
V_{CCINT}	-	P128	V_{CCINT}*	V_{CCINT}*	-
I/O, TRDY(1)	3	P129	J15	M22	890
V_{CCO}	3	P130	V_{CCO} Bank 3*	V_{CCO} Bank 3*	-
V_{CCO}	2	P130	V_{CCO} Bank 2*	V_{CCO} Bank 2*	-
GND	-	P131	GND*	GND*	-
I/O, IRDY(1)	2	P132	H16	L20	893
I/O	2	P133	H14	L17	896
I/O	2	-	-	L18	902
I/O	2	P134	H15	L21	905
I/O	2	-	J13	L22	908
I/O	2	-	-	K19	911
I/O (D3)	2	P135	G16	K20	917
I/O, V_{REF}	2	P136	H13	K21	920
V_{CCO}	2	-	V_{CCO} Bank 2*	V_{CCO} Bank 2*	-
GND	-	P137	GND*	GND*	-
I/O	2	P138	G14	K22	923
I/O	2	P139	G15	J21	926

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name					
Function	Bank	PQ208	FG256	FG456	Bndry Scan
I/O	2	-	-	K18	929
I/O	2	-	-	J20	932
I/O	2	P140	G12	J18	935
GND	-	-	GND*	GND*	-
I/O	2	-	F16	J22	938
I/O	2	-	-	J19	941
I/O	2	-	-	H21	944
I/O	2	P141	G13	H19	947
I/O (D2)	2	P142	F15	H20	950
V_{CCINT}	-	P143	V_{CCINT}*	V_{CCINT}*	-
V_{CCO}	2	P144	V_{CCO} Bank 2*	V_{CCO} Bank 2*	-
GND	-	P145	GND*	GND*	-
I/O (D1)	2	P146	E16	H22	953
I/O, V_{REF}	2	P147	F14	H18	956
I/O	2	-	-	G21	962
I/O	2	P148	D16	G18	965
GND	-	-	GND*	GND*	-
I/O	2	-	F12	G20	968
I/O	2	-	-	G19	971
I/O	2	-	-	F22	974
I/O	2	P149	E15	F19	977
I/O, V_{REF}	2	P150	F13	F21	980
V_{CCO}	2	-	V_{CCO} Bank 2*	V_{CCO} Bank 2*	-
GND	-	-	GND*	GND*	-
I/O	2	P151	E14	F20	983
I/O	2	-	C16	F18	986
GND	-	-	GND*	GND*	-
I/O	2	-	-	E22	989
I/O	2	-	-	E21	995
I/O, V_{REF}	2	P152	E13	D22	998
GND	-	-	GND*	GND*	-
I/O	2	-	B16	E20	1001
I/O	2	-	-	D21	1004
I/O	2	-	-	C22	1007
I/O (DIN, D0)	2	P153	D14	D20	1013
I/O (DOUT, BUSY)	2	P154	C15	C21	1016
CCLK	2	P155	D15	B22	1019
V_{CCO}	2	P156	V_{CCO} Bank 2*	V_{CCO} Bank 2*	-

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
V_{CCO}	1	P156	V_{CCO} Bank 1*	V_{CCO} Bank 1*	-
TDO	2	P157	B14	A21	-
GND	-	P158	GND*	GND*	-
TDI	-	P159	A15	B20	-
I/O ($\overline{CS}$)	1	P160	B13	C19	0
I/O ($\overline{WRITE}$)	1	P161	C13	A20	3
I/O	1	-	-	B19	9
I/O	1	-	-	C18	12
I/O	1	-	C12	D17	15
GND	-	-	GND*	GND*	-
I/O, V_{REF}	1	P162	A14	A19	18
I/O	1	-	-	B18	21
I/O	1	-	-	E16	27
I/O	1	-	D12	C17	30
I/O	1	P163	B12	D16	33
GND	-	-	GND*	GND*	-
V_{CCO}	1	-	V_{CCO} Bank 1*	V_{CCO} Bank 1*	-
I/O, V_{REF}	1	P164	C11	A18	36
I/O	1	P165	A13	B17	39
I/O	1	-	-	E15	42
I/O	1	-	-	A17	45
I/O	1	-	D11	D15	48
GND	-	-	GND*	GND*	-
I/O	1	P166	A12	C16	51
I/O	1	-	-	D14	54
I/O, V_{REF}	1	P167	E11	E14	60
I/O	1	P168	B11	A16	63
GND	-	P169	GND*	GND*	-
V_{CCO}	1	P170	V_{CCO} Bank 1*	V_{CCO} Bank 1*	-
V_{CCINT}	-	P171	V_{CCINT}*	V_{CCINT}*	-
I/O	1	P172	A11	C15	66
I/O	1	P173	C10	B15	69
I/O	1	-	-	E13	72
I/O	1	-	-	A15	75
I/O	1	-	-	F12	78
GND	-	-	GND*	GND*	-
I/O	1	P174	B10	C14	81
I/O	1	-	-	B14	84
I/O	1	-	-	A14	87

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
I/O	1	P175	D10	D13	90
I/O	1	P176	A10	C13	93
GND	-	P177	GND*	GND*	-
V_{CCO}	1	-	V_{CCO} Bank 1*	V_{CCO} Bank 1*	-
I/O, V_{REF}	1	P178	B9	B13	96
I/O	1	P179	E10	E12	99
I/O	1	-	-	A13	105
I/O	1	-	A9	B12	108
I/O	1	P180	D9	D12	111
I/O	1	-	-	C12	114
I/O	1	P181	A8	D11	120
I, GCK2	1	P182	C9	A11	126
GND	-	P183	GND*	GND*	-
V_{CCO}	1	P184	V_{CCO} Bank 1*	V_{CCO} Bank 1*	-
V_{CCO}	0	P184	V_{CCO} Bank 0*	V_{CCO} Bank 0*	-
I, GCK3	0	P185	B8	C11	127
V_{CCINT}	-	P186	V_{CCINT}*	V_{CCINT}*	-
I/O	0	-	-	E11	137
I/O	0	P187	A7	A10	140
I/O	0	-	D8	B10	143
I/O	0	-	-	F11	146
I/O	0	P188	A6	C10	152
I/O, V_{REF}	0	P189	B7	A9	155
V_{CCO}	0	-	V_{CCO} Bank 0*	V_{CCO} Bank 0*	-
GND	-	P190	GND*	GND*	-
I/O	0	P191	C8	B9	158
I/O	0	P192	D7	E10	161
I/O	0	-	-	C9	164
I/O	0	-	-	D10	167
I/O	0	P193	E7	A8	170
GND	-	-	GND*	GND*	-
I/O	0	-	-	D9	173
I/O	0	-	-	B8	176
I/O	0	-	-	C8	179
I/O	0	P194	C7	E9	182
I/O	0	P195	B6	A7	185
V_{CCINT}	-	P196	V_{CCINT}*	V_{CCINT}*	-
V_{CCO}	0	P197	V_{CCO} Bank 0*	V_{CCO} Bank 0*	-

XC2S200 Device Pinouts *(Continued)*

XC2S200 Pad Name		PQ208	FG256	FG456	Bndry Scan
Function	Bank				
GND	-	P198	GND*	GND*	-
I/O	0	P199	A5	B7	188
I/O, V_{REF}	0	P200	C6	E8	191
I/O	0	-	-	D8	197
I/O	0	P201	B5	C7	200
GND	-	-	GND*	GND*	-
I/O	0	-	D6	D7	203
I/O	0	-	-	B6	206
I/O	0	-	-	A5	209
I/O	0	P202	A4	D6	212
I/O, V_{REF}	0	P203	B4	C6	215
V_{CCO}	0	-	V_{CCO} Bank 0*	V_{CCO} Bank 0*	-
GND	-	-	GND*	GND*	-
I/O	0	P204	E6	B5	218
I/O	0	-	D5	E7	221
I/O	0	-	-	A4	224
I/O	0	-	-	E6	230
I/O, V_{REF}	0	P205	A3	B4	233
GND	-	-	GND*	GND*	-
I/O	0	-	C5	A3	236
I/O	0	-	-	B3	239
I/O	0	-	-	D5	242
I/O	0	P206	B3	C5	248
TCK	-	P207	C4	C4	-
V_{CCO}	0	P208	V_{CCO} Bank 0*	V_{CCO} Bank 0*	-
V_{CCO}	7	P208	V_{CCO} Bank 7*	V_{CCO} Bank 7*	-

04/18/01

Notes:

1. IRDY and TRDY can only be accessed when using Xilinx PCI cores.
2. Pads labelled GND*, V_{CCINT}*, V_{CCO} Bank 0*, V_{CCO} Bank 1*, V_{CCO} Bank 2*, V_{CCO} Bank 3*, V_{CCO} Bank 4*, V_{CCO} Bank 5*, V_{CCO} Bank 6*, V_{CCO} Bank 7* are internally bonded to independent ground or power planes within the package.

Additional XC2S200 Package Pins

PQ208

Not Connected Pins					
P55	P56	-	-	-	-

11/02/00

FG256

V_{CCINT} Pins					
C3	C14	D4	D13	E5	E12
M5	M12	N4	N13	P3	P14
V_{CCO} Bank 0 Pins					
E8	F8	-	-	-	-
V_{CCO} Bank 1 Pins					
E9	F9	-	-	-	-
V_{CCO} Bank 2 Pins					
H11	H12	-	-	-	-
V_{CCO} Bank 3 Pins					
J11	J12	-	-	-	-
V_{CCO} Bank 4 Pins					
L9	M9	-	-	-	-
V_{CCO} Bank 5 Pins					
L8	M8	-	-	-	-
V_{CCO} Bank 6 Pins					
J5	J6	-	-	-	-
V_{CCO} Bank 7 Pins					
H5	H6	-	-	-	-
GND Pins					
A1	A16	B2	B15	F6	F7
F10	F11	G6	G7	G8	G9
G10	G11	H7	H8	H9	H10
J7	J8	J9	J10	K6	K7
K8	K9	K10	K11	L6	L7
L10	L11	R2	R15	T1	T16
Not Connected Pins					
P4	R4	-	-	-	-

11/02/00